

# Dirichlet Series

Manuel Eberl

February 6, 2026

## 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  $\varphi$ , Möbius  $\mu$ , Liouville's  $\lambda$ , the divisor function  $\sigma$ , von Mangoldt's  $\Lambda$ )
- 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

<b>1</b>	<b>Miscellaneous auxiliary facts</b>	<b>4</b>
<b>2</b>	<b>Multiplicative arithmetic functions</b>	<b>7</b>
2.1	Definition . . . . .	7
2.2	Indicator function . . . . .	13
<b>3</b>	<b>Dirichlet convolution</b>	<b>14</b>
<b>4</b>	<b>Formal Dirichlet series</b>	<b>26</b>
4.1	General properties . . . . .	29
4.2	Shifting the argument . . . . .	37
4.3	Scaling the argument . . . . .	40
4.4	Formal derivative . . . . .	43
4.5	Formal integral . . . . .	46
4.6	Formal logarithm . . . . .	47
4.7	Formal exponential . . . . .	48
4.8	Subseries . . . . .	50
4.9	Truncation . . . . .	52
4.10	Normed series . . . . .	56
4.11	Lifting a real series to a real algebra . . . . .	56
4.12	Convergence and connection to concrete functions . . . . .	57
<b>5</b>	<b>The Möbius <math>\mu</math> function</b>	<b>64</b>
<b>6</b>	<b>Euler's <math>\phi</math> function</b>	<b>72</b>
<b>7</b>	<b>The Liouville <math>\lambda</math> function</b>	<b>75</b>
<b>8</b>	<b>The divisor functions</b>	<b>78</b>
8.1	The general divisor function . . . . .	78
8.2	The divisor-counting function . . . . .	81
8.3	The divisor sum function . . . . .	82
<b>9</b>	<b>Summatory arithmetic functions</b>	<b>84</b>
9.1	Definition . . . . .	84
9.2	The Hyperbola method . . . . .	86
<b>10</b>	<b>Partial summation</b>	<b>88</b>
<b>11</b>	<b>Euler product expansions</b>	<b>91</b>

<b>12 Analytic properties of Dirichlet series</b>	<b>96</b>
12.1 Convergence and absolute convergence . . . . .	99
12.2 Derivative of a Dirichlet series . . . . .	113
12.3 Multiplication of two series . . . . .	121
12.4 Uniqueness . . . . .	130
12.5 Limit at infinity . . . . .	134
12.6 Normed series . . . . .	137
12.7 Logarithms of Dirichlet series . . . . .	139
12.8 Exponential and logarithm . . . . .	145
12.9 Euler products . . . . .	152
12.10 Non-negative Dirichlet series . . . . .	155
12.11 Convergence of the $\zeta$ and Möbius $\mu$ series . . . . .	163
12.12 Application to the Möbius $\mu$ function . . . . .	163
<b>13 Asymptotics of summatory arithmetic functions</b>	<b>164</b>
13.1 Auxiliary bounds . . . . .	164
13.2 Summatory totient function . . . . .	167
13.3 Asymptotic distribution of squarefree numbers . . . . .	169
13.4 The hyperbola method . . . . .	172
13.5 The asymptotic distribution of coprime pairs . . . . .	178
13.6 The asymptotics of the number of Farey fractions . . . . .	180
<b>14 Efficient code for number-theoretic functions</b>	<b>182</b>
14.1 Möbius $\mu$ function . . . . .	183
14.2 Euler's $\phi$ function . . . . .	184
14.3 Divisor Functions . . . . .	186
14.4 Liouville's $\lambda$ function . . . . .	186

# 1 Miscellaneous auxiliary facts

**theory** *Dirichlet-Misc*

**imports**

*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** –

**have**  $(\text{real } a - 1) * (\sum_{i < k}. \text{real } a^i) = \text{real } a^k - 1$

**using** *assms* **by** (*subst geometric-sum*) *auto*

**also have**  $(\text{real } a - 1) * (\sum_{i < k}. \text{real } a^i) = \text{real } ((a - 1) * (\sum_{i < k}. a^i))$

**using** *assms* **by** (*simp add: of-nat-diff*)

**also have**  $\text{real } a^k - 1 = \text{real } (a^k - 1)$  **using** *assms* **by** (*subst of-nat-diff*)

*auto*

**finally show**  $*$ :  $(a - 1) * (\sum_{i < k}. a^i) = a^k - 1$  **by** (*subst (asm) of-nat-eq-iff*)

**show**  $a - 1 \text{ dvd } a^k - 1$  **by** (*subst \* [symmetric]*) *simp*

**from** *assms* **show**  $(\sum_{i < k}. a^i) = (a^k - 1) \text{ div } (a - 1)$

**by** (*subst \* [symmetric]*) *simp*

**qed**

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

**by** *auto*

**lemma** *Set-filter-insert*:

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

**by** *auto*

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

**by** *auto*

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

**by** *auto*

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

**by** *auto*

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

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

**by** *auto*

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

by (auto simp: inj-on-def insert-eq-iff)

**lemma**

assumes *finite A A ≠ {}*

shows *card-even-subset-aux: card {B. B ⊆ A ∧ even (card B)} = 2 ^ (card A - 1)*

and *card-odd-subset-aux: card {B. B ⊆ A ∧ odd (card B)} = 2 ^ (card A - 1)*

and *card-even-odd-subset: card {B. B ⊆ A ∧ even (card B)} = card {B. B ⊆ A ∧ odd (card B)}*

**proof** -

from *assms* have \*:  $2 * \text{card } (\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A)) = 2 ^ \text{card } A$

**proof** (*induction A rule: finite-ne-induct*)

case (*singleton x*)

then have  $\text{Pow } \{x\} = \{\{\}, \{x\}\}$  by *auto*

moreover have  $\langle \{A. (A = \{\} \vee A = \{x\}) \wedge \text{even } (\text{card } A)\} = \{\{\}\} \rangle$

by *auto*

ultimately show ?*case*

by *simp*

**next**

case (*insert x A*)

**note** *fin = finite-subset[OF - ⟨finite A⟩]*

**have**  $\text{Pow } (\text{insert } x A) = \text{Pow } A \cup \text{insert } x \text{ ' Pow } A$  by (*rule Pow-insert*)

**have**  $\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } (\text{insert } x A)) =$   
 $\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A) \cup$   
 $\text{insert } x \text{ ' Set.filter } (\text{even} \circ \text{card} \circ \text{insert } x) (\text{Pow } A)$

**unfolding** *Pow-insert Set-filter-union Set-filter-image* by *blast*

**also have**  $\text{Set.filter } (\text{even} \circ \text{card} \circ \text{insert } x) (\text{Pow } A) = \text{Set.filter } (\text{odd} \circ \text{card})$   
 $(\text{Pow } A)$

**unfolding** *o-def*

by (*intro Set-filter-cong refl, subst card-insert-disjoint*)  
*(insert insert.hyps, auto dest: finite-subset)*

**also have**  $\text{card } (\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A) \cup \text{insert } x \text{ ' } \dots) =$   
 $\text{card } (\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A)) + \text{card } (\text{insert } x \text{ ' } \dots)$

(**is**  $\text{card } (?A \cup ?B) = -$ )

by (*intro card-Un-disjoint finite-filter finite-imageI*) (*auto simp: insert.hyps*)

**also have**  $\text{card } ?B = \text{card } (\text{Set.filter } (\text{odd} \circ \text{card}) (\text{Pow } A))$

using *insert.hyps* by (*intro card-image inj-on-insert'*) *auto*

**also have**  $\text{Set.filter } (\text{odd} \circ \text{card}) (\text{Pow } A) = \text{Pow } A - \text{Set.filter } (\text{even} \circ \text{card})$   
 $(\text{Pow } A)$

by *auto*

**also have**  $\text{card } \dots = \text{card } (\text{Pow } A) - \text{card } (\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A))$

using *insert.hyps* by (*subst card-Diff-subset*) (*auto simp: finite-filter*)

**also have**  $\text{card } (\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A)) + \dots = \text{card } (\text{Pow } A)$

by (*intro add-diff-inverse-nat, subst not-less, rule card-mono*) (*insert insert.hyps, auto*)

**also have**  $2 * \dots = 2 ^ \text{card } (\text{insert } x A)$

using *insert.hyps* by (*simp add: card-Pow*)

**finally show** ?*case* .

**qed**  
**from** \* **show**  $A$ :  $\text{card } \{B. B \subseteq A \wedge \text{even } (\text{card } B)\} = 2^{\wedge}(\text{card } A - 1)$   
**by** (*cases card A*) (*simp-all add:*)  
  
**have**  $\text{Set.filter } (\text{odd} \circ \text{card}) (\text{Pow } A) = \text{Pow } A - \text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A)$  **by** *auto*  
**also have**  $2 * \text{card } \dots = 2 * 2^{\wedge} \text{card } A - 2 * \text{card } (\text{Set.filter } (\text{even} \circ \text{card}) (\text{Pow } A))$   
**using** *assms* **by** (*subst card-Diff-subset*) (*auto intro!: finite-filter simp: card-Pow*)  
**also note** \*  
**also have**  $2 * 2^{\wedge} \text{card } A - 2^{\wedge} \text{card } A = (2^{\wedge} \text{card } A :: \text{nat})$  **by** *simp*  
**finally show**  $B$ :  $\text{card } \{B. B \subseteq A \wedge \text{odd } (\text{card } B)\} = 2^{\wedge}(\text{card } A - 1)$   
**by** (*cases card A*) (*simp-all add:*)

**from**  $A$  **and**  $B$  **show**  $\text{card } \{B. B \subseteq A \wedge \text{even } (\text{card } B)\} = \text{card } \{B. B \subseteq A \wedge \text{odd } (\text{card } B)\}$  **by** *simp*  
**qed**

**lemma** *bij-betw-prod-divisors-coprime*:  
**assumes** *coprime a (b :: nat)*  
**shows**  $\text{bij-betw } (\lambda x. \text{fst } x * \text{snd } x) (\{d. d \text{ dvd } a\} \times \{d. d \text{ dvd } b\}) \{k. k \text{ dvd } a * b\}$   
**unfolding** *bij-betw-def*  
**proof**  
**from** *assms* **show**  $\text{inj-on } (\lambda x. \text{fst } x * \text{snd } x) (\{d. d \text{ dvd } a\} \times \{d. d \text{ dvd } b\})$   
**by** (*auto simp: inj-on-def coprime-crossproduct-nat coprime-divisors*)  
**show**  $(\lambda x. \text{fst } x * \text{snd } x) ' (\{d. d \text{ dvd } a\} \times \{d. d \text{ dvd } b\}) = \{k. k \text{ dvd } a * b\}$   
**proof** *safe*  
**fix**  $x$  **assume**  $x \text{ dvd } a * b$   
**then obtain**  $b' c'$  **where**  $x = b' * c' b' \text{ dvd } a c' \text{ dvd } b$   
**using** *division-decomp* **by** *blast*  
**thus**  $x \in (\lambda x. \text{fst } x * \text{snd } x) ' (\{d. d \text{ dvd } a\} \times \{d. d \text{ dvd } b\})$  **by** *force*  
**qed** (*insert assms, auto intro: mult-dvd-mono*)  
**qed**

**lemma** *bij-betw-prime-power-divisors*:  
**assumes** *prime (p :: nat)*  
**shows**  $\text{bij-betw } ((\wedge) p) \{..k\} \{d. d \text{ dvd } p^{\wedge} k\}$   
**unfolding** *bij-betw-def*  
**proof**  
**from** *assms* **have**  $*$ :  $p > 1$  **by** (*simp add: prime-gt-Suc-0-nat*)  
**show**  $\text{inj-on } ((\wedge) p) \{..k\}$  **using** *assms*  
**by** (*auto simp: inj-on-def prime-gt-Suc-0-nat power-inject-exp[OF \*]*)  
**show**  $(\wedge) p ' \{..k\} = \{d. d \text{ dvd } p^{\wedge} k\}$   
**using** *assms* **by** (*auto simp: le-imp-power-dvd divides-primexpow-nat*)  
**qed**

**lemma** *sum-divisors-coprime-mult*:  
**assumes** *coprime a (b :: 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 -
  have  $(\sum r \mid r \text{ dvd } a. \sum s \mid s \text{ dvd } b. f (r * s)) =$ 
     $(\sum z \in \{r. r \text{ dvd } a\} \times \{s. s \text{ dvd } b\}. f (fst z * snd z))$ 
    by (subst sum.cartesian-product) (simp add: case-prod-unfold)
  also have  $\dots = (\sum d \mid d \text{ dvd } a * b. f d)$ 
    by (intro sum.reindex-bij-betw bij-betw-prod-divisors-coprime assms)
  finally show ?thesis ..
qed

end

```

## 2 Multiplicative arithmetic functions

```

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

```

### 2.1 Definition

```

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

```

```

lemma Suc-0 [simp]: f (Suc 0) = 1
  using one by (simp del: one)

```

```

lemma mult-coprime:
  assumes coprime a b
  shows f (a * b) = f a * f b
proof -
  {fix n :: nat consider n = 0 | n = 1 | n > 1 by force} note P = this
  show ?thesis by (cases a rule: P; cases b rule: P) (simp-all add: mult-coprime-aux
  assms)
qed

```

```

lemma prod-coprime:
  assumes  $\bigwedge x y. x \in A \implies y \in A \implies x \neq y \implies \text{coprime } (g x) (g y)$ 
  shows f (prod g A) =  $(\prod x \in A. f (g x))$ 
  using assms
proof (induction rule: infinite-finite-induct)
  case (insert x A)
  from insert have f (prod g (insert x A)) = f (g x * prod g A) by simp

```

```

also have ... = f (g x) * f (prod g A) using insert.premys insert.hyps
  by (auto intro: mult-coprime prod-coprime-right)
also have ... = (∏ x∈insert x A. f (g x)) using insert by simp
finally show ?case .
qed auto

lemma prod-prime-factors:
  assumes n > 0
  shows f n = (∏ p∈prime-factors n. f (p ^ multiplicity p n))
proof -
  have n = (∏ p∈prime-factors n. p ^ multiplicity p n)
    using Primes.prime-factorization-nat assms by blast
  also have f ... = (∏ p∈prime-factors n. f (p ^ multiplicity p n))
    by (rule prod-coprime) (auto simp add: in-prime-factors-imp-prime primes-coprime)

  finally show ?thesis .
qed

lemma multiplicative-sum-divisors: multiplicative-function (λn. ∑ d | d dvd n. f
d)
proof
  fix a b :: nat assume ab: a > 1 b > 1 coprime a b
  hence (∑ d | d dvd a * b. f d) = (∑ r | r dvd a. ∑ s | s dvd b. f (r * s))
    by (intro sum-divisors-coprime-mult)
  also have ... = (∑ r | r dvd a. ∑ s | s dvd b. f r * f s)
    using ab(3)
    by (auto intro!: sum.cong intro: mult-coprime coprime-imp-coprime dvd-trans)
  also have ... = (∑ r | r dvd a. f r) * (∑ s | s dvd b. f s)
    by (subst sum-distrib-right, subst sum-distrib-left) simp-all
  finally show (∑ d | d dvd a * b. f d) = (∑ r | r dvd a. f r) * (∑ s | s dvd b. f
s) .
qed auto

end

locale multiplicative-function' = multiplicative-function f for f :: nat ⇒ 'a ::
comm-semiring-1 +
  fixes f-prime-power :: nat ⇒ nat ⇒ 'a and f-prime :: nat ⇒ 'a
  assumes prime-power: prime p ⇒ k > 0 ⇒ f (p ^ k) = f-prime-power p k
  assumes prime-aux: prime p ⇒ f-prime-power p 1 = f-prime p
begin

lemma prime: prime p ⇒ f p = f-prime p
  using prime-power[of p 1] prime-aux[of p] by simp

lemma prod-prime-factors':
  assumes n > 0
  shows f n = (∏ p∈prime-factors n. f-prime-power p (multiplicity p n))
    by (subst prod-prime-factors[OF assms(1)])

```

(intro prod.cong refl prime-power, auto simp: prime-factors-multiplicity)

**lemma** *efficient-code-aux*:

**assumes**  $n > 0$  set  $ps = (\lambda p. (p, \text{multiplicity } p \ n - 1))$  ‘prime-factors  $n$  distinct  
 $ps$

**shows**  $f \ n = (\prod (p,d) \leftarrow ps. \text{f-prime-power } p \ (\text{Suc } d))$

**proof** –

**from** *assms* **have**

$(\prod (p,d) \leftarrow ps. \text{f-prime-power } p \ (\text{Suc } d)) =$

$(\prod (p,d) \in (\lambda p. (p, \text{multiplicity } p \ n - 1)) \text{ ‘prime-factors } n. \text{f-prime-power } p \ (\text{Suc } d))$

**by** (*subst prod.distinct-set-conv-list [symmetric]*) *simp-all*

**also have**  $\dots = (\prod x \in \text{prime-factors } n. \text{f-prime-power } x \ (\text{multiplicity } x \ n))$

**by** (*subst prod.reindex*) (*auto simp: inj-on-def prime-factors-multiplicity intro!: prod.cong*)

**also have**  $\dots = f \ n$  **by** (*rule prod-prime-factors' [symmetric]*) *fact+*

**finally show** *?thesis ..*

**qed**

**lemma** *efficient-code*:

**assumes** set  $(ps \ ()) = (\lambda p. (p, \text{multiplicity } p \ n - 1))$  ‘prime-factors  $n$  distinct  
 $(ps \ ())$

**shows**  $f \ n = (\text{if } n = 0 \text{ then } 0 \text{ else } (\prod (p,d) \leftarrow ps \ (). \text{f-prime-power } p \ (\text{Suc } d)))$

**using** *efficient-code-aux*[of  $n \ ps \ ()$ ] *assms* **by** *simp*

**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** –

{**fix**  $n :: \text{nat}$  **consider**  $n = 0 \mid n = 1 \mid n > 1$  **by force**} **note**  $P = \text{this}$

**show** *?thesis* **by** (*cases a rule: P; cases b rule: P*) (*simp-all add: zero-aux one-aux mult-aux*)

**qed**

**sublocale** *multiplicative-function*  $f$

**by** *standard* (*simp-all add: zero-aux one-aux mult*)

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

**by** (*induction A rule: infinite-finite-induct*) (*simp-all add: mult*)

**lemma** *power*:  $f \ (n \wedge m) = f \ n \wedge m$

by (induction m) (simp-all add: mult)

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

by (subst prime-factorization-nat) (simp-all add: prod power)

**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'':  $n > 0 \implies f\ n = (\prod_{p \in \text{prime-factors } n} f\text{-prime } p \wedge \text{multiplicity } p\ n)$

by (subst prod-prime-factors') (auto simp: f-prime prime-factors-multiplicity intro!: prod.cong)

**lemma** efficient-code-aux:  
**assumes**  $n > 0$  set ps =  $(\lambda p. (p, \text{multiplicity } p\ n - 1))$  'prime-factors n distinct ps  
**shows**  $f\ n = (\prod (p,d) \leftarrow ps. f\text{-prime } p \wedge \text{Suc } d)$   
**proof** –  
**from** assms **have**  
 $(\prod (p,d) \leftarrow ps. f\text{-prime } p \wedge \text{Suc } d) =$   
 $(\prod (p,d) \in (\lambda p. (p, \text{multiplicity } p\ n - 1)) \text{ 'prime-factors } n. f\text{-prime } p \wedge \text{Suc } d)$   
**by** (subst prod.distinct-set-conv-list [symmetric]) simp-all  
**also have**  $\dots = (\prod_{x \in \text{prime-factors } n} f\text{-prime } x \wedge \text{multiplicity } x\ n)$   
**by** (subst prod.reindex) (auto simp: inj-on-def prime-factors-multiplicity simp del: power-Suc intro!: prod.cong)  
**also have**  $\dots = f\ n$  **by** (rule prod-prime-factors'' [symmetric]) fact+  
**finally show** ?thesis ..  
**qed**

**lemma** efficient-code:  
**assumes** set (ps ()) =  $(\lambda p. (p, \text{multiplicity } p\ n - 1))$  'prime-factors n distinct (ps ())  
**shows**  $f\ n = (\text{if } n = 0 \text{ then } 0 \text{ else } (\prod (p,d) \leftarrow ps\ (). f\text{-prime } p \wedge \text{Suc } d))$   
**using** efficient-code-aux[of n ps ()] assms **by** simp

**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** –

```

interpret f: multiplicative-function f by fact
interpret g: multiplicative-function g by fact
show ?thesis
proof (cases n > 0)
  case True
  thus ?thesis
    using f.prod-prime-factors[OF True] g.prod-prime-factors[OF True]
    by (auto intro!: prod.cong assms simp: prime-factors-multiplicity)
qed simp-all
qed

```

```

lemma multiplicative-function-of-natI:
  multiplicative-function f  $\implies$  multiplicative-function ( $\lambda n. \text{of-nat } (f\ n)$ )
  unfolding multiplicative-function-def by auto

```

```

lemma multiplicative-function-of-natD:
  multiplicative-function ( $\lambda n. \text{of-nat } (f\ n)$ ) :: 'a :: {ring-char-0, comm-semiring-1}
 $\implies$ 
  multiplicative-function f
  unfolding multiplicative-function-def
  by (auto simp: of-nat-mult [symmetric] of-nat-eq-1-iff simp del: of-nat-mult)

```

```

lemma multiplicative-function-mult:
  assumes multiplicative-function f multiplicative-function g
  shows multiplicative-function ( $\lambda n. f\ n * g\ n$ )
proof
  interpret f: multiplicative-function f by fact
  interpret g: multiplicative-function g by fact
  show f 0 * g 0 = 0 f 1 * g 1 = 1 by simp-all
  fix a b :: nat assume a > 1 b > 1 coprime a b
  thus f (a * b) * g (a * b) = (f a * g a) * (f b * g b)
    by (simp-all add: f.mult-coprime g.mult-coprime mult-ac)
qed

```

```

lemma multiplicative-function-inverse:
  fixes f :: nat  $\Rightarrow$  'a :: field
  assumes multiplicative-function f
  shows multiplicative-function ( $\lambda n. \text{inverse } (f\ n)$ )
proof
  interpret f: multiplicative-function f by fact
  show inverse (f 0) = 0 inverse (f 1) = 1 by simp-all
  fix a b :: nat assume a > 1 b > 1 coprime a b
  thus inverse (f (a * b)) = inverse (f a) * inverse (f b)
    by (simp-all add: f.mult-coprime field-simps)
qed

```

```

lemma multiplicative-function-divide:
  fixes f :: nat  $\Rightarrow$  'a :: field
  assumes multiplicative-function f multiplicative-function g

```

**shows** *multiplicative-function*  $(\lambda n. f\ n / g\ n)$   
**proof** –  
**have** *multiplicative-function*  $(\lambda n. f\ n * \text{inverse}\ (g\ n))$   
**by** (*intro multiplicative-function-mult multiplicative-function-inverse assms*)  
**also have**  $(\lambda n. f\ n * \text{inverse}\ (g\ n)) = (\lambda n. f\ n / g\ n)$   
**by** (*simp add: field-simps*)  
**finally show** *?thesis* .  
**qed**

**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**  
**interpret**  $f$ : *completely-multiplicative-function*  $f$  **by fact**  
**interpret**  $g$ : *completely-multiplicative-function*  $g$  **by fact**  
**show**  $f\ 0 * g\ 0 = 0$   $f\ (\text{Suc}\ 0) * g\ (\text{Suc}\ 0) = 1$  **by** *simp-all*  
**fix**  $a\ b :: \text{nat}$  **assume**  $a > 1\ b > 1$   
**thus**  $f\ (a * b) * g\ (a * b) = (f\ a * g\ a) * (f\ b * g\ b)$   
**by** (*simp-all add: f.mult g.mult mult-ac*)  
**qed**

**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**  
**interpret**  $f$ : *completely-multiplicative-function*  $f$  **by fact**  
**show**  $\text{inverse}\ (f\ 0) = 0$   $\text{inverse}\ (f\ (\text{Suc}\ 0)) = 1$  **by** *simp-all*  
**fix**  $a\ b :: \text{nat}$  **assume**  $a > 1\ b > 1$   
**thus**  $\text{inverse}\ (f\ (a * b)) = \text{inverse}\ (f\ a) * \text{inverse}\ (f\ b)$   
**by** (*simp-all add: f.mult field-simps*)  
**qed**

**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*  $(\lambda n. f\ n / g\ n)$   
**proof** –  
**have** *completely-multiplicative-function*  $(\lambda n. f\ n * \text{inverse}\ (g\ n))$   
**by** (*intro completely-multiplicative-function-mult completely-multiplicative-function-inverse assms*)  
**also have**  $(\lambda n. f\ n * \text{inverse}\ (g\ n)) = (\lambda n. f\ n / g\ n)$   
**by** (*simp add: field-simps*)  
**finally show** *?thesis* .  
**qed**

**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*

**proof**

**fix**  $m\ n :: \text{nat}$  **assume**  $mn: m > 1\ n > 1$

**define**  $P$  **where**  $P = \text{prime-factors } (m * n)$

**have**  $f (m * n) = (\prod_{p \in P}. f (p \wedge \text{multiplicity } p (m * n)))$

**using**  $mn$  **by**  $(\text{subst prod-prime-factors}) (auto simp: P-def)$

**also have**  $\dots = (\prod_{p \in P}. f p \wedge \text{multiplicity } p (m * n))$

**by**  $(\text{intro prod.cong}) (auto simp: assms prime-factors-multiplicity P-def)$

**also have**  $\dots = (\prod_{p \in P}. f p \wedge \text{multiplicity } p m * f p \wedge \text{multiplicity } p n)$

**by**  $(\text{intro prod.cong refl, subst prime-elm-multiplicity-mult-distrib})$   
 $(\text{use } mn \text{ in } \langle auto simp: P-def prime-factors-multiplicity power-add \rangle)$

**also have**  $\dots = (\prod_{p \in P}. f p \wedge \text{multiplicity } p m) * (\prod_{p \in P}. f p \wedge \text{multiplicity } p n)$

**by**  $(\text{rule prod.distrib})$

**also have**  $(\prod_{p \in P}. f p \wedge \text{multiplicity } p m) = (\prod_{p \in \text{prime-factors } m}. f p \wedge \text{multiplicity } p m)$

**unfolding**  $P-def$  **by**  $(\text{intro prod.mono-neutral-right dvd-prime-factors finite-set-mset})$   
 $(\text{use } mn \text{ in } \langle auto simp: prime-factors-multiplicity \rangle)$

**also have**  $\dots = (\prod_{p \in \text{prime-factors } m}. f (p \wedge \text{multiplicity } p m))$

**by**  $(\text{intro prod.cong}) (auto simp: assms prime-factors-multiplicity)$

**also have**  $\dots = f m$

**using**  $mn$  **by**  $(\text{intro prod-prime-factors } [symmetric]) auto$

**also have**  $(\prod_{p \in P}. f p \wedge \text{multiplicity } p n) = (\prod_{p \in \text{prime-factors } n}. f p \wedge \text{multiplicity } p n)$

**unfolding**  $P-def$  **by**  $(\text{intro prod.mono-neutral-right dvd-prime-factors finite-set-mset})$   
 $(\text{use } mn \text{ in } \langle auto simp: prime-factors-multiplicity \rangle)$

**also have**  $\dots = (\prod_{p \in \text{prime-factors } n}. f (p \wedge \text{multiplicity } p n))$

**by**  $(\text{intro prod.cong}) (auto simp: assms prime-factors-multiplicity)$

**also have**  $\dots = f n$

**using**  $mn$  **by**  $(\text{intro prod-prime-factors } [symmetric]) auto$

**finally show**  $f (m * n) = f m * f n .$

**qed** *auto*

## 2.2 Indicator function

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

**lemma**  $ind-0$   $[simp]: ind\ P\ 0 = 0$  **by**  $(simp\ add: ind-def)$

**lemma**  $ind-nonzero: n > 0 \Longrightarrow ind\ P\ n = (\text{if } P\ n \text{ then } 1 \text{ else } 0)$   
**by**  $(simp\ add: ind-def)$

**lemma**  $ind-True$   $[simp]: P\ n \Longrightarrow n > 0 \Longrightarrow ind\ P\ n = 1$   
**by**  $(simp\ add: ind-nonzero)$

**lemma**  $ind-False$   $[simp]: \neg P\ n \Longrightarrow n > 0 \Longrightarrow ind\ P\ n = 0$   
**by**  $(simp\ add: ind-nonzero)$

**lemma** *ind-eq-1-iff*:  $\text{ind } P \ n = 1 \longleftrightarrow n > 0 \wedge P \ n$   
**by** (*simp add: ind-def*)

**lemma** *ind-eq-0-iff*:  $\text{ind } P \ n = 0 \longleftrightarrow n = 0 \vee \neg P \ n$   
**by** (*simp add: ind-def*)

**lemma** *multiplicative-function-ind* [*intro?*]:  
**assumes**  $P \ 1 \wedge a \ b. \ a > 1 \implies b > 1 \implies \text{coprime } a \ b \implies P \ (a * b) \longleftrightarrow P \ a$   
 $\wedge P \ b$   
**shows** *multiplicative-function* (*ind P*)  
**by** *standard* (*insert assms, auto simp: ind-nonzero*)

**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** (*intro sum.cong*)

**fix**  $r \ s$

**assume**  $r \in \{r. \ r \ \text{dvd} \ a\}$  **and**  $s \in \{s. \ s \ \text{dvd} \ b\}$

**then have**  $r \ \text{dvd} \ a$  **and**  $s \ \text{dvd} \ b$

**by** *simp-all*

**moreover from**  $\langle \text{coprime } a \ b \rangle$  **have**  $\text{coprime } r \ s$

**using**  $\langle r \ \text{dvd} \ a \rangle \langle s \ \text{dvd} \ b \rangle$

**by** (*auto intro: coprime-imp-coprime dvd-trans*)

**ultimately show**  $f \ r \ s = g \ r \ s$

**using** *that by simp*

**qed** *auto*

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

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

**proof** –

**have**  $(\lambda d \ \text{acc.} \ \text{if } d \ \text{dvd} \ n \ \text{then } f \ d + \text{acc} \ \text{else } \text{acc}) = (\lambda d \ \text{acc.} \ (\text{if } d \ \text{dvd} \ n \ \text{then } f \ d$   
 $\text{else } 0) + \text{acc})$

**by** (*simp add: fun-eq-iff*)

**hence**  $\text{fold-atLeastAtMost-nat } (\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } f d + \text{acc else acc}) 1 n 0$   
 $=$   
 $\text{fold-atLeastAtMost-nat } (\lambda d \text{ acc. (if } d \text{ dvd } n \text{ then } f d \text{ else } 0) + \text{acc}) 1 n 0$   
**by** (*simp only*: )  
**also have**  $\dots = (\sum d = 1..n. \text{if } d \text{ dvd } n \text{ then } f d \text{ else } 0)$   
**by** (*rule sum-atLeastAtMost-code [symmetric]*)  
**also from** *assms* **have**  $\dots = (\sum d \mid d \text{ dvd } n. f d)$   
**by** (*intro sum.mono-neutral-cong-right*) (*auto elim: dvdE dest: dvd-imp-le*)  
**finally show** *?thesis ..*  
**qed**

**lemma** *dirichlet-prod-code [code]*:  
 $\text{dirichlet-prod } f g n = (\text{if } n = 0 \text{ then } 0 \text{ else}$   
 $\text{fold-atLeastAtMost-nat } (\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } f d * g (n \text{ div } d) + \text{acc else}$   
 $\text{acc}) 1 n 0)$   
**unfolding** *dirichlet-prod-def* **by** (*simp add: sum-divisors-code*)

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

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

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

**proof**  
**fix**  $n :: \text{nat}$   
**show**  $\text{dirichlet-prod } f g n = \text{dirichlet-prod } f' g' n$   
**proof** (*cases n = 0*)  
**case** *False*  
**with** *assms* **show** *?thesis* **unfolding** *dirichlet-prod-def*  
**by** (*intro ext sum.cong refl*) (*auto elim!: dvdE*)  
**qed** *simp-all*  
**qed**

**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**  
**fix**  $n :: \text{nat}$   
**show**  $\text{dirichlet-prod } f g n = (\sum d \mid d \text{ dvd } n. f (n \text{ div } d) * g d)$   
**proof** (*cases n = 0*)  
**case** *False*  
**hence**  $\text{dirichlet-prod } f g n = (\sum d \mid d \text{ dvd } n. f (n \text{ div } (n \text{ div } d)) * g (n \text{ div } d))$   
**unfolding** *dirichlet-prod-def* **by** (*intro sum.cong refl*) (*auto elim!: dvdE*)  
**also from** *False* **have**  $\dots = (\sum d \mid d \text{ dvd } n. f (n \text{ div } d) * g d)$   
**by** (*intro sum.reindex-bij-witness[of - (div) n (div) n]*) (*auto elim!: dvdE*)  
**finally show** *?thesis .*

qed simp  
qed

lemma dirichlet-prod-altdef2:

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

proof

fix n

show dirichlet-prod f g n = ( $\sum (r,d) \mid r * d = n. f r * g d$ )

proof (cases n = 0)

case True

have ( $\lambda n::nat. (0, n)$ ) ' UNIV  $\subseteq \{(r,d). r * d = 0\}$  by auto

moreover have  $\neg$ finite (( $\lambda n::nat. (0, n)$ ) ' UNIV)

by (subst finite-image-iff) (auto simp: inj-on-def)

ultimately have infinite  $\{(r,d). r * d = (0::nat)\}$

by (blast dest: finite-subset)

with True show ?thesis by simp

next

case False

have ( $\sum d \mid d \text{ dvd } n. f d * g (n \text{ div } d)$ ) = ( $\sum r \mid r \text{ dvd } n. (\sum d \mid d = n \text{ div } r. f r * g d)$ )

by (intro sum.cong refl) auto

also from False have ... = ( $\sum (r,d) \in (\text{SIGMA } x:\{r. r \text{ dvd } n\}. \{d. d = n \text{ div } x\}). f r * g d$ )

by (intro sum.Sigma) auto

also from False have ( $\text{SIGMA } x:\{r. r \text{ dvd } n\}. \{d. d = n \text{ div } x\}$ ) =  $\{(r,d). r * d = n\}$

by auto

finally show ?thesis by (simp add: dirichlet-prod-def)

qed

qed

lemma dirichlet-prod-commutes:

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

proof

fix n :: nat

show dirichlet-prod f g n = dirichlet-prod g f n

proof (cases n = 0)

case False

have ( $\sum (r,d) \mid r * d = n. f r * g d$ ) = ( $\sum (d,r) \mid r * d = n. f r * g d$ )

by (rule sum.reindex-bij-witness [of -  $\lambda(x,y). (y,x)$   $\lambda(x,y). (y,x)$ ]) auto

thus ?thesis by (simp add: dirichlet-prod-altdef2 mult commute)

qed (simp add: dirichlet-prod-def)

qed

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

by (rule finite-subset[of -  $\{0 <..n\} \times \{0 <..n\}$ ]) auto

lemma dirichlet-prod-assoc-aux1:

assumes n > 0

**shows**  $\text{dirichlet-prod } f \text{ (dirichlet-prod } g \text{ h) } n =$   
 $(\sum (a, b, c) \in \{(a, b, c). a * b * c = n\}. f a * g b * h c)$   
**proof** –  
**have**  $\text{dirichlet-prod } f \text{ (dirichlet-prod } g \text{ h) } n =$   
 $(\sum x \in \{(a, b). a * b = n\}. (\sum (c, d) \mid c * d = \text{snd } x. f (fst x) * g c * h d))$   
**by** (*auto intro!*: *sum.cong simp: dirichlet-prod-altdef2 sum-distrib-left mult.assoc*)  
**also from** *assms* **have**  $\dots = (\sum x \in (\text{SIGMA } x: \{(a, b). a * b = n\}. \{(c, d). c * d = \text{snd } x\}).$   
 $\text{case } x \text{ of } (x, c, d) \Rightarrow f (fst x) * g c * h d)$   
**by** (*intro sum.Sigma finite-divisors-nat' ballI*) *auto*  
**also have**  $\dots = (\sum (a, b, c) \mid a * b * c = n. f a * g b * h c)$   
**by** (*rule sum.reindex-bij-witness*  
 $[of - \lambda(a, b, c). ((a, b * c), (b, c)) \lambda((a, b), (c, d)). (a, c, d)]$   
*(auto simp: mult-ac)*)  
**finally show** *?thesis* .  
**qed**

**lemma** *dirichlet-prod-assoc-aux2*:  
**assumes**  $n > 0$   
**shows**  $\text{dirichlet-prod (dirichlet-prod } f \text{ g) } h \text{ } n =$   
 $(\sum (a, b, c) \in \{(a, b, c). a * b * c = n\}. f a * g b * h c)$   
**proof** –  
**have**  $\text{dirichlet-prod (dirichlet-prod } f \text{ g) } h \text{ } n =$   
 $(\sum x \in \{(a, b). a * b = n\}. (\sum (c, d) \mid c * d = \text{fst } x. f c * g d * h (\text{snd } x)))$   
**by** (*auto intro!*: *sum.cong simp: dirichlet-prod-altdef2 sum-distrib-right mult.assoc*)  
**also from** *assms* **have**  $\dots = (\sum x \in (\text{SIGMA } x: \{(a, b). a * b = n\}. \{(c, d). c * d = \text{fst } x\}).$   
 $\text{case } x \text{ of } (x, c, d) \Rightarrow f c * g d * h (\text{snd } x))$   
**by** (*intro sum.Sigma finite-divisors-nat' ballI*) *auto*  
**also have**  $\dots = (\sum (a, b, c) \mid a * b * c = n. f a * g b * h c)$   
**by** (*rule sum.reindex-bij-witness*  
 $[of - \lambda(a, b, c). ((a * b, c), (a, b)) \lambda((a, b), (c, d)). (c, d, b)]$   
*(auto simp: mult-ac)*)  
**finally show** *?thesis* .  
**qed**

**lemma** *dirichlet-prod-assoc*:  
 $\text{dirichlet-prod (dirichlet-prod } f \text{ g) } h = \text{dirichlet-prod } f \text{ (dirichlet-prod } g \text{ h)}$   
**proof**  
**fix**  $n :: \text{nat}$   
**show**  $\text{dirichlet-prod (dirichlet-prod } f \text{ g) } h \text{ } n = \text{dirichlet-prod } f \text{ (dirichlet-prod } g \text{ h)}$   
 $n$   
**by** (*cases n = 0*) (*simp-all add: dirichlet-prod-assoc-aux1 dirichlet-prod-assoc-aux2*)  
**qed**

**lemma** *dirichlet-prod-const-right* [*simp*]:  
**assumes**  $n > 0$   
**shows**  $\text{dirichlet-prod } f \text{ } (\lambda n. \text{if } n = \text{Suc } 0 \text{ then } c \text{ else } 0) \text{ } n = f n * c$   
**proof** –

**have**  $\text{dirichlet-prod } f \ (\lambda n. \text{if } n = \text{Suc } 0 \text{ then } c \text{ else } 0) \ n =$   
 $(\sum d \mid d \text{ dvd } n. (\text{if } d = n \text{ then } f \ n * c \text{ else } 0))$   
**unfolding**  $\text{dirichlet-prod-def}$  **using**  $\text{assms}$   
**by**  $(\text{intro sum.cong refl}) \ (\text{auto elim!}: \text{dvdE split}: \text{if-splits})$   
**also have**  $\dots = f \ n * c$  **using**  $\text{assms}$  **by**  $(\text{subst sum.delta}) \ \text{auto}$   
**finally show**  $?thesis$  .  
**qed**

**lemma**  $\text{dirichlet-prod-const-left}$  [ $\text{simp}$ ]:  
**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** –  
**have**  $\text{dirichlet-prod } (\lambda n. \text{if } n = \text{Suc } 0 \text{ then } c \text{ else } 0) \ g \ n =$   
 $(\sum d \mid d \text{ dvd } n. (\text{if } d = 1 \text{ then } c * g \ n \text{ else } 0))$   
**unfolding**  $\text{dirichlet-prod-def}$  **using**  $\text{assms}$   
**by**  $(\text{intro sum.cong refl}) \ (\text{auto elim!}: \text{dvdE split}: \text{if-splits})$   
**also have**  $\dots = c * g \ n$  **using**  $\text{assms}$  **by**  $(\text{subst sum.delta}) \ \text{auto}$   
**finally show**  $?thesis$  .  
**qed**

**fun**  $\text{dirichlet-inverse} :: (\text{nat} \Rightarrow 'a :: \text{comm-ring-1}) \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a$  **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**  $\text{dirichlet-inverse-induct}$  [ $\text{case-names } 0 \ 1 \ \text{gt1}$ ]:  
 $P \ 0 \Longrightarrow P \ (\text{Suc } 0) \Longrightarrow (\bigwedge n. n > 1 \Longrightarrow (\bigwedge k. k < n \Longrightarrow P \ k) \Longrightarrow P \ n) \Longrightarrow P \ n$   
**by**  $\text{induction-schema}$  ( $\text{force}, \text{rule wf-measure [of id]}, \text{simp}$ )

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

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

**declare**  $\text{dirichlet-inverse.simps}$  [ $\text{simp del}$ ]

**lemma**  $\text{dirichlet-inverse-gt-1}$ :  
 $n > 1 \Longrightarrow \text{dirichlet-inverse } f \ i \ n =$   
 $-i * (\sum d \mid d \text{ dvd } n \wedge d < n. f \ (n \text{ div } d) * \text{dirichlet-inverse } f \ i \ d)$   
**by**  $(\text{simp add: dirichlet-inverse.simps})$

**lemma**  $\text{dirichlet-inverse-cong}$  [ $\text{cong}$ ]:  
**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** –  
**have**  $\text{dirichlet-inverse } f \ i \ n = \text{dirichlet-inverse } f' \ i' \ n$   
**using**  $\text{assms}(1)$

**proof** (*induction n rule: dirichlet-inverse-induct*)  
**case** (*gt1 n*)  
**have** \*: *dirichlet-inverse f i k = dirichlet-inverse f' i k if k dvd n ∧ k < n for k*  
**using** *that by (intro gt1) auto*  
**have** \*:  $(\sum d \mid d \text{ dvd } n \wedge d < n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d) =$   
 $(\sum d \mid d \text{ dvd } n \wedge d < n. f' (n \text{ div } d) * \text{dirichlet-inverse } f' i d)$   
**by** (*intro sum.cong refl*) (*subst gt1.prem, auto elim: dvdE simp: \**)  
**consider** *n = 0 | n = 1 | n > 1 by force*  
**thus** ?*case*  
**by** *cases (insert \*, simp-all add: dirichlet-inverse-gt-1 \* cong: sum.cong)*  
**qed** *auto*  
**with** *assms(2,3) show ?thesis by simp*  
**qed**

**lemma** *dirichlet-inverse-gt-1'*:  
**assumes** *n > 1*  
**shows** *dirichlet-inverse f i n =*  
*-i \* dirichlet-prod (λn. if n = 1 then 0 else f n) (dirichlet-inverse f i) n*  
**proof** –  
**have** *dirichlet-prod (λn. if n = 1 then 0 else f n) (dirichlet-inverse f i) n =*  
 $(\sum d \mid d \text{ dvd } n. (\text{if } n \text{ div } d = \text{Suc } 0 \text{ then } 0 \text{ else } f (n \text{ div } d)) * \text{dirichlet-inverse}$   
 $f i d)$   
**by** (*simp add: dirichlet-prod-altdef1*)  
**also from** *assms have ... =*  $(\sum d \mid d \text{ dvd } n \wedge d \neq n. f (n \text{ div } d) * \text{dirichlet-}$   
 $\text{inverse } f i d)$   
**by** (*intro sum.mono-neutral-cong-right*) (*auto elim: dvdE*)  
**also from** *assms have {d. d dvd n ∧ d ≠ n} = {d. d dvd n ∧ d < n}* **by** (*auto*  
*dest: dvd-imp-le*)  
**also from** *assms have -i \**  $(\sum d \in \dots f (n \text{ div } d) * \text{dirichlet-inverse } f i d) =$   
 $\text{dirichlet-inverse } f i n$   
**by** (*simp add: dirichlet-inverse-gt-1*)  
**finally show** ?*thesis ..*  
**qed**

**lemma** *of-int-dirichlet-prod*:  
*of-int (dirichlet-prod f g n) = dirichlet-prod (λn. of-int (f n)) (λn. of-int (g n)) n*  
**by** (*simp add: dirichlet-prod-def*)

**lemma** *of-int-dirichlet-inverse*:  
*of-int (dirichlet-inverse f i n) = dirichlet-inverse (λn. of-int (f n)) (of-int i) n*  
**proof** (*induction n rule: dirichlet-inverse-induct*)  
**case** (*gt1 n*)  
**from** *gt1 have (of-int (dirichlet-inverse f i n) :: 'a) =*  
 $-(\text{of-int } i * (\sum d \mid d \text{ dvd } n \wedge d < n. \text{of-int } (f (n \text{ div } d)) * \text{dirichlet-inverse } f i$   
 $d)))$   
**(is - = - (- \* ?A))**  
**by** (*simp add: dirichlet-inverse-gt-1 of-int-dirichlet-prod*)  
**also have** ?*A =*  $(\sum d \mid d \text{ dvd } n \wedge d < n. \text{of-int } (f (n \text{ div } d)) *$   
 $\text{dirichlet-inverse } (\lambda n. \text{of-int } (f n)) (\text{of-int } i) d)$

by (intro sum.cong refl) (auto simp: gt1)  
 also have  $-(\text{of-int } i * \dots) = \text{dirichlet-inverse } (\lambda n. \text{of-int } (f n)) (\text{of-int } i) n$   
 using gt1.hyps by (simp add: dirichlet-inverse-gt-1)  
 finally show ?case .  
 qed simp-all

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

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

**proof** –

consider  $n = 0 \mid n = 1 \mid n > 1$  by force  
 thus ?thesis  
**proof** cases  
 assume  $n: n > 1$   
 have \*:  $(\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } f (n \text{ div } d) * \text{dirichlet-inverse } f i d + \text{acc else acc}) =$   
 $(\lambda d \text{ acc. (if } d \text{ dvd } n \text{ then } f (n \text{ div } d) * \text{dirichlet-inverse } f i d \text{ else } 0) + \text{acc})$   
 by (simp add: fun-eq-iff)  
 have  $\text{fold-atLeastAtMost-nat } (\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } f (n \text{ div } d) * \text{dirichlet-inverse } f i d + \text{acc else acc}) 1 (n - 1) 0 =$   
 $(\sum d = 1..n - 1. \text{if } d \text{ dvd } n \text{ then } f (n \text{ div } d) * \text{dirichlet-inverse } f i d \text{ else } 0)$   
 by (subst \*, subst sum-atLeastAtMost-code [symmetric]) simp  
 also from  $n$  have  $\dots = (\sum d \mid d \text{ dvd } n \wedge d < n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d)$   
 by (intro sum.mono-neutral-cong-right; cases n)  
 (auto dest: dvd-imp-le elim: dvdE simp: Suc-le-eq intro!: Nat.gr0I)  
 also from  $n$  have  $-i * \dots = \text{dirichlet-inverse } f i n$   
 by (simp add: dirichlet-inverse-gt-1)  
 finally show ?thesis using  $n$  by simp  
 qed auto  
 qed

**lemma** *dirichlet-prod-inverse*:

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

**proof**

fix  $n :: \text{nat}$   
 consider  $n = 0 \mid n = 1 \mid n > 1$  by force  
 thus  $\text{dirichlet-prod } f (\text{dirichlet-inverse } f i) n = (\text{if } n = 1 \text{ then } 1 \text{ else } 0)$   
**proof** cases  
 assume  $n: n > 1$   
 have  $\text{fin: finite } \{d. d \text{ dvd } n \wedge d \neq n\}$   
 by (rule finite-subset[of - {d. d dvd n}]) (insert n, auto)  
 have  $\text{dirichlet-prod } f (\text{dirichlet-inverse } f i) n =$   
 $(\sum d \mid d \text{ dvd } n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d)$   
 by (simp add: dirichlet-prod-altdef1)

**also have**  $\{d. d \text{ dvd } n\} = \text{insert } n \{d. d \text{ dvd } n \wedge d \neq n\}$  **by** *auto*  
**also have**  $(\sum d \in \dots f (n \text{ div } d) * \text{dirichlet-inverse } f i d) =$   
 $f 1 * \text{dirichlet-inverse } f i n +$   
 $(\sum d \mid d \text{ dvd } n \wedge d \neq n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d)$   
**using** *fin n* **by** (*subst sum.insert*) *auto*  
**also from** *n* **have**  $\text{dirichlet-inverse } f i n =$   
 $- i * (\sum d \mid d \text{ dvd } n \wedge d < n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d)$   
**by** (*subst dirichlet-inverse-gt-1*) *auto*  
**also from** *n* **have**  $\{d. d \text{ dvd } n \wedge d < n\} = \{d. d \text{ dvd } n \wedge d \neq n\}$  **by** (*auto*  
*dest: dvd-imp-le*)  
**also have**  $f 1 * (- i * (\sum d \mid d \text{ dvd } n \wedge d \neq n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d)) =$   
 $-(f 1 * i) * (\sum d \mid d \text{ dvd } n \wedge d \neq n. f (n \text{ div } d) * \text{dirichlet-inverse } f i d)$   
**by** (*simp add: mult.assoc*)  
**also have**  $f 1 * i = 1$  **by** *fact*  
**finally show** *?thesis* **using** *n* **by** *simp*  
**qed** (*insert assms, simp-all add: dirichlet-prod-def*)  
**qed**

**lemma** *dirichlet-prod-inverse'*:  
**assumes**  $f 1 * i = 1$   
**shows**  $\text{dirichlet-prod } (\text{dirichlet-inverse } f i) f = (\lambda n. \text{if } n = 1 \text{ then } 1 \text{ else } 0)$   
**using** *dirichlet-prod-inverse[of f] assms* **by** (*simp add: dirichlet-prod-commutes*)

**lemma** *dirichlet-inverse-noninvertible*:  
**assumes**  $f (\text{Suc } 0) = (0 :: 'a :: \{\text{comm-ring-1}\})$   $i = 0$   
**shows**  $\text{dirichlet-inverse } f i n = 0$   
**using** *assms*  
**by** (*induction f i n rule: dirichlet-inverse.induct*) (*auto simp: dirichlet-inverse.simps*)

**lemma** *multiplicative-dirichlet-prod*:  
**assumes** *multiplicative-function f*  
**assumes** *multiplicative-function g*  
**shows** *multiplicative-function (dirichlet-prod f g)*  
**proof** –  
**interpret** *f: multiplicative-function f* **by** *fact*  
**interpret** *g: multiplicative-function g* **by** *fact*  
**show** *?thesis*  
**proof**  
**fix**  $a b :: \text{nat}$  **assume**  $a > 1$   $b > 1$  **and** *coprime: coprime a b*  
**hence**  $\text{dirichlet-prod } f g (a * b) =$   
 $(\sum r \mid r \text{ dvd } a. \sum s \mid s \text{ dvd } b. f (r * s) * g (a * b \text{ div } (r * s)))$   
**by** (*simp add: dirichlet-prod-def sum-divisors-coprime-mult*)  
**also have**  $\dots = (\sum r \mid r \text{ dvd } a. \sum s \mid s \text{ dvd } b. f r * f s * g (a \text{ div } r) * g (b \text{ div } s))$   
**using**  $\langle \text{coprime } a b \rangle$  **proof** (*rule sum-coprime-dvd-cong*)  
**fix**  $r s$   
**assume** *coprime r s* **and**  $r \text{ dvd } a$  **and**  $s \text{ dvd } b$

```

with ⟨a > 1⟩ ⟨b > 1⟩ have r > 0 s > 0
  by (auto intro: ccontr)
from ⟨coprime r s⟩ have f (r * s) = f r * f s
  by (rule f.mult-coprime)
moreover from ⟨coprime a b⟩ have ⟨coprime (a div r) (b div s)⟩
  using ⟨r > 0⟩ ⟨s > 0⟩ ⟨r dvd a⟩ ⟨s dvd b⟩ dvd-div-iff-mult [of r a]
dvd-div-iff-mult [of s b]
  by (auto dest: coprime-imp-coprime dvd-mult-left)
then have g (a div r * (b div s)) = g (a div r) * g (b div s)
  by (rule g.mult-coprime)
ultimately show f (r * s) * g (a * b div (r * s)) = f r * f s * g (a div r) *
g (b div s)
  using ⟨r dvd a⟩ ⟨s dvd b⟩ by (simp add: div-mult-div-if-dvd ac-simps)
qed
also have ... = dirichlet-prod f g a * dirichlet-prod f g b
  unfolding dirichlet-prod-def by (simp add: sum-product mult-ac)
finally show dirichlet-prod f g (a * b) = ... .
qed simp-all
qed

```

lemma *multiplicative-dirichlet-prodD1*:

```

fixes f g :: nat ⇒ 'a :: comm-semiring-1-cancel
assumes multiplicative-function (dirichlet-prod f g)
assumes multiplicative-function f
assumes [simp]: g 0 = 0
shows multiplicative-function g
proof -
interpret f: multiplicative-function f by fact
interpret fg: multiplicative-function dirichlet-prod f g by fact
show ?thesis
proof
have dirichlet-prod f g (Suc 0) = 1 by (rule fg.Suc-0)
also have dirichlet-prod f g (Suc 0) = g 1 by (subst dirichlet-prod-Suc-0) simp
finally show g 1 = 1 by simp
next
fix a b :: nat assume ab: a > 1 b > 1 coprime a b
hence a > 0 b > 0 coprime a b by simp-all
thus g (a * b) = g a * g b
proof (induction a * b arbitrary: a b rule: less-induct)
case (less a b)
have dirichlet-prod f g (a * b) + g a * g b =
  (∑ r | r dvd a * b. f r * g (a * b div r)) + g a * g b
  by (simp add: dirichlet-prod-def)
also have {r. r dvd a * b} = insert 1 {r. r dvd a * b ∧ r ≠ 1} by auto
also have (∑ r ∈ ... f r * g (a * b div r)) + g a * g b =
  g (a * b) + ((∑ r | r dvd a * b ∧ r ≠ 1. f r * g (a * b div r)) + g
a * g b)
using less.premis
  by (subst sum.insert) (auto intro!: finite-subset[OF - finite-divisors-nat])

```

*simp: add-ac*)  
**also have**  $(\sum r \mid r \text{ dvd } a * b \wedge r \neq 1. f r * g (a * b \text{ div } r)) =$   
 $(\sum r \mid r \text{ dvd } a * b. \text{if } r = 1 \text{ then } 0 \text{ else } f r * g (a * b \text{ div } r))$   
**using** *less.prem*s **by** (*intro sum.mono-neutral-cong-left*) (*auto intro: finite-divisors-nat'*)  
**also have**  $\dots = (\sum r \mid r \text{ dvd } a. \sum d \mid d \text{ dvd } b.$   
 $\text{if } r * d = 1 \text{ then } 0 \text{ else } f (r * d) * g (a * b \text{ div } (r * d)))$   
**using**  $\langle \text{coprime } a \ b \rangle$  **by** (*rule sum-divisors-coprime-mult*)  
**also have**  $\dots = (\sum r \mid r \text{ dvd } a. \sum d \mid d \text{ dvd } b.$   
 $\text{if } r * d = 1 \text{ then } 0 \text{ else } f (r * d) * g ((a \text{ div } r) * (b \text{ div } d)))$   
**by** (*intro sum.cong refl*) (*auto elim!: dvdE*)  
**also have**  $\dots = (\sum r \mid r \text{ dvd } a. \sum d \mid d \text{ dvd } b.$   
 $\text{if } r * d = 1 \text{ then } 0 \text{ else } f r * f d * g (a \text{ div } r) * g (b \text{ div } d))$   
**using**  $\langle \text{coprime } a \ b \rangle$  **proof** (*rule sum-coprime-dvd-cong*)  
**fix**  $r \ s$   
**assume** *coprime*  $r \ s$  **and**  $r \text{ dvd } a$  **and**  $s \text{ dvd } b$   
**with**  $\langle a > 0 \rangle \langle b > 0 \rangle$  **have**  $r > 0 \ s > 0$   
**by** (*auto intro: ccontr*)  
**from**  $\langle \text{coprime } r \ s \rangle$  **have**  $f (r * s) = f r * f s$   
**by** (*rule f.mult-coprime*)  
**show**  $(\text{if } r * s = 1 \text{ then } 0 \text{ else } f (r * s) * g (a \text{ div } r * (b \text{ div } s))) =$   
 $(\text{if } r * s = 1 \text{ then } 0 \text{ else } f r * f s * g (a \text{ div } r) * g (b \text{ div } s))$   
**proof** (*cases*  $r * s = 1$ )  
**case** *True*  
**then show** *?thesis*  
**by** *simp*  
**next**  
**case** *False*  
**with**  $\langle r \text{ dvd } a \rangle \langle s \text{ dvd } b \rangle$  *less.prem*s  
**have**  $(a \text{ div } r) * (b \text{ div } s) \neq a * b$   
**by** (*intro notI*) (*auto elim!: dvdE*)  
**moreover from**  $\langle r \text{ dvd } a \rangle \langle s \text{ dvd } b \rangle$  *less.prem*s  
**have**  $(a \text{ div } r) * (b \text{ div } s) \leq a * b$   
**by** (*intro dvd-imp-le mult-dvd-mono Nat.grOI*) (*auto elim!: dvdE*)  
**ultimately have**  $(a \text{ div } r) * (b \text{ div } s) < a * b$   
**by** *arith*  
**with**  $\langle r \text{ dvd } a \rangle \langle s \text{ dvd } b \rangle$  *less.prem*s  
**have**  $g ((a \text{ div } r) * (b \text{ div } s)) = g (a \text{ div } r) * g (b \text{ div } s)$   
**by** (*auto intro: less coprime-divisors [OF - - <coprime a b>]*) (*elim!: dvdE*)  
**from** *False* **show** *?thesis*  
**by** (*auto simp: less f g ac-simps*)  
**qed**  
**qed**  
**also have**  $\dots = (\sum (r,d) \in \{r. r \text{ dvd } a\} \times \{d. d \text{ dvd } b\}.$   
 $\text{if } r * d = 1 \text{ then } 0 \text{ else } f r * f d * g (a \text{ div } r) * g (b \text{ div } d))$   
**by** (*simp add: sum.cartesian-product*)  
**also have**  $\dots = (\sum (r1,r2) \in \{r1. r1 \text{ dvd } a\} \times \{r2. r2 \text{ dvd } b\} - \{(1,1)\}.$   
 $(f r1 * f r2) * g (a \text{ div } r1) * g (b \text{ div } r2))$  (*is - = sum ?f ?A*)  
**using** *less.prem*s **by** (*intro sum.mono-neutral-cong-right*) (*auto split: if-splits*)

```

also have ... + g a * g b = ?f (1, 1) + sum ?f ?A by (simp add: add-ac)
also have ... = sum ?f ({r1. r1 dvd a} × {r2. r2 dvd b}) using less.premis
  by (intro sum.remove [symmetric]) auto
also have ... = dirichlet-prod f g a * dirichlet-prod f g b
  by (simp add: sum.cartesian-product sum-product dirichlet-prod-def mult-ac)
also have g (a * b) + dirichlet-prod f g a * dirichlet-prod f g b =
  dirichlet-prod f g (a * b) + g (a * b)
  using less.premis by (simp add: fg.mult-coprime add-ac)
finally show ?case by simp
qed
qed simp-all
qed

```

```

lemma multiplicative-dirichlet-prodD2:
  fixes f g :: nat ⇒ 'a :: comm-semiring-1-cancel
  assumes multiplicative-function (dirichlet-prod f g)
  assumes multiplicative-function g
  assumes [simp]: f 0 = 0
  shows multiplicative-function f
proof -
  from assms(1) have multiplicative-function (dirichlet-prod g f)
  by (simp add: dirichlet-prod-commutes)
  from multiplicative-dirichlet-prodD1[OF this assms(2)] show ?thesis by simp
qed

```

```

lemma multiplicative-dirichlet-inverse:
  assumes multiplicative-function f
  shows multiplicative-function (dirichlet-inverse f 1)
proof (rule multiplicative-dirichlet-prodD1[OF - assms])
  interpret multiplicative-function f by fact
  have multiplicative-function (λn. if n = 1 then 1 else 0)
  by standard simp-all
  thus multiplicative-function (dirichlet-prod f (dirichlet-inverse f 1))
  by (subst dirichlet-prod-inverse) simp-all
qed simp-all

```

```

lemma dirichlet-prod-prime-power:
  assumes prime p
  shows dirichlet-prod f g (p ^ k) = (∑ i≤k. f (p ^ i) * g (p ^ (k - i)))
proof -
  have dirichlet-prod f g (p ^ k) = (∑ i≤k. f (p ^ i) * g (p ^ k div p ^ i))
  unfolding dirichlet-prod-def using assms
  by (intro sum.reindex-bij-betw [symmetric] bij-betw-prime-power-divisors)
  also from assms have ... = (∑ i≤k. f (p ^ i) * g (p ^ (k - i)))
  by (intro sum.cong refl) (auto simp: power-diff)
  finally show ?thesis .
qed

```

```

lemma dirichlet-prod-prime:

```

```

assumes prime p
shows dirichlet-prod f g p = f 1 * g p + f p * g 1
using dirichlet-prod-prime-power[of p f g 1] assms by simp

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

sublocale multiplicative-function dirichlet-prod f g
  by (intro multiplicative-dirichlet-prod
    f.multiplicative-function-axioms g.multiplicative-function-axioms)

end

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

sublocale multiplicative-dirichlet-prod f g ..

sublocale multiplicative-function' dirichlet-prod f g
   $\lambda p k.$  f-prime-power p k + g-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-prime p + g-prime p
proof (standard, goal-cases)
  case (1 p k)
  hence dirichlet-prod f g (p ^ k) = ( $\sum_{i \leq k} f \ (p \wedge i) * g \ (p \wedge (k - i))$ )
    by (intro dirichlet-prod-prime-power)
  also from 1 have {..k} = insert 0 (insert k {0 <..<k}) by auto
  also have ( $\sum_{i \in \dots} f \ (p \wedge i) * g \ (p \wedge (k - i))$ ) =
    f-prime-power p k + g-prime-power p k +
    ( $\sum_{i \in \{0 <..<k\}} f \ (p \wedge i) * g \ (p \wedge (k - i))$ ) using 1
    by (auto simp: f.prime-power g.prime-power add-ac)
  also have ( $\sum_{i \in \{0 <..<k\}} f \ (p \wedge i) * g \ (p \wedge (k - i))$ ) =
    ( $\sum_{i \in \{0 <..<k\}} f\text{-prime-power } p \ i * g\text{-prime-power } p \ (k - i)$ )
    using 1 by (intro sum.cong) (auto simp: f.prime-power g.prime-power)
  finally show ?case .
next
  case (2 p)
  have {0 <..<Suc 0} = {} by auto
  with 2 show ?case
    by (auto simp: f.prime-power [symmetric] g.prime-power [symmetric] f.prime
      g.prime add-ac)
qed

```

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**  $'a \text{ fds} = \text{UNIV} :: (\text{nat} \Rightarrow 'a) \text{ set}$   
**by** *simp*

**setup-lifting** *type-definition-fds*

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

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

**lemma** *fds-nth-fds*:  $\text{fds-nth} (\text{fds } f) \ n = (\text{if } n = 0 \ \text{then } 0 \ \text{else } f \ n)$   
**by** *transfer (simp split: nat.splits)*

**lemma** *fds-nth-fds'*:  $f \ 0 = 0 \implies \text{fds-nth} (\text{fds } f) = f$   
**by** *(simp add: fun-eq-iff fds-nth-fds)*

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

**lemma** *fds-nth-fds-pos* [*simp*]:  $n > 0 \implies \text{fds-nth} (\text{fds } f) \ n = f \ n$   
**by** *transfer (simp split: nat.splits)*

**lemma** *fds-fds-nth* [*simp*]:  $\text{fds} (\text{fds-nth } f) = f$   
**by** *transfer (simp add: fun-eq-iff split: nat.splits)*

**lemma** *fds-eq-fds-iff*:  
 $fds\ f = fds\ g \longleftrightarrow (\forall n > 0. f\ n = g\ n)$   
**proof** *transfer*  
**fix**  $f\ g :: nat \Rightarrow 'a$   
**have**  $(f \circ Suc = g \circ Suc) \longleftrightarrow (\forall n. f\ (Suc\ n) = g\ (Suc\ n))$  **by** (*auto simp: fun-eq-iff*)  
**also have**  $\dots \longleftrightarrow (\forall n > 0. f\ n = g\ n)$   
**proof** *safe*  
**fix**  $n :: nat$  **assume**  $\forall n. f\ (Suc\ n) = g\ (Suc\ n)$   $n > 0$   
**thus**  $f\ n = g\ n$  **by** (*cases n auto*)  
**qed** *auto*  
**finally show**  $(f \circ Suc = g \circ Suc) = (\forall n > 0. f\ n = g\ n)$  .  
**qed**

**lemma** *fds-eq-fds-iff'*:  $f\ 0 = g\ 0 \implies fds\ f = fds\ g \longleftrightarrow f = g$   
**proof** *safe*  
**assume**  $f\ 0 = g\ 0$   $fds\ f = fds\ g$   
**hence**  $f\ n = g\ n$  **for**  $n$  **by** (*cases n auto simp: fds-eq-fds-iff*)  
**thus**  $f = g$  **by** (*simp add: fun-eq-iff*)  
**qed**

**lemma** *fds-eqI* [*intro?*]:  
**assumes**  $(\bigwedge n. n > 0 \implies fds\text{-}nth\ f\ n = fds\text{-}nth\ g\ n)$   
**shows**  $f = g$   
**proof** –  
**from** *assms* **have**  $fds\text{-}nth\ f\ n = fds\text{-}nth\ g\ n$  **if**  $n > 0$  **for**  $n$   
**by** (*cases n (simp-all add: fun-eq-iff)*)  
**hence**  $fds\ (fds\text{-}nth\ f) = fds\ (fds\text{-}nth\ g)$  **by** (*subst fds-eq-fds-iff auto*)  
**thus** *?thesis* **by** *simp*  
**qed**

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

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

**lemma** *dirichlet-prod-fds-nth-fds-left* [*simp*]:  
 $dirichlet\text{-}prod\ (fds\text{-}nth\ (fds\ f))\ g = dirichlet\text{-}prod\ f\ g$   
**by** (*simp add: fds-nth-fds*)

**lemma** *dirichlet-prod-fds-nth-fds-right* [*simp*]:  
 $dirichlet\text{-}prod\ f\ (fds\text{-}nth\ (fds\ g)) = dirichlet\text{-}prod\ f\ g$   
**by** (*simp add: fds-nth-fds*)

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

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

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

**bundle** *fds-syntax*  
**begin**

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

**notation** *fds* (**binder**  $\langle \chi \rangle$  10)

**notation** *dirichlet-prod* (**infixl**  $\langle * \rangle$  70)

**end**

**instantiation** *fds* :: (*zero*) *zero*

**begin**

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

**instance** ..

**end**

**instantiation** *fds* :: ( $\{ \text{zero}, \text{one} \}$ ) *one*

**begin**

**definition** *one-fds* :: 'a *fds* **where** *one-fds* = *fds* ( $\lambda n. \text{ if } n = 1 \text{ then } 1 \text{ else } 0$ )

**instance** ..

**end**

**instantiation** *fds* :: ( $\{ \text{plus}, \text{zero} \}$ ) *plus*

**begin**

**definition** *plus-fds* :: 'a *fds*  $\Rightarrow$  'a *fds*  $\Rightarrow$  'a *fds*

**where** *plus-fds*  $f g = \text{fds } (\lambda n. \text{ fds-nth } f n + \text{ fds-nth } g n)$

**instance** ..

**end**

**instantiation** *fds* :: (*semiring-0*) *times*

**begin**

**definition** *times-fds* :: 'a *fds*  $\Rightarrow$  'a *fds*  $\Rightarrow$  'a *fds*

**where** *times-fds*  $f g = \text{fds } (\text{dirichlet-prod } (\text{fds-nth } f) (\text{fds-nth } g))$

**instance** ..

**end**

**instantiation** *fds* :: ( $\{ \text{uminus}, \text{zero} \}$ ) *uminus*

**begin**

**definition** *uminus-fds* :: 'a *fds*  $\Rightarrow$  'a *fds*

**where** *uminus-fds*  $f = \text{fds } (\lambda n. -\text{fds-nth } f n)$

**instance** ..

**end**

**instantiation** *fds* :: ( $\{ \text{minus}, \text{zero} \}$ ) *minus*

**begin**

**definition** *minus-fds* :: 'a fds  $\Rightarrow$  'a fds  $\Rightarrow$  'a fds  
 where *minus-fds* f g = fds ( $\lambda n. \text{fds-nth } f \ n - \text{fds-nth } g \ n$ )  
**instance** ..  
**end**

#### 4.1 General properties

**lemma** *fds-nth-zero* [*simp*]: *fds-nth* 0 = ( $\lambda \cdot. 0$ )  
 by (*simp add: zero-fds-def fds-nth-fds fun-eq-iff*)

**lemma** *fds-nth-one*: *fds-nth* 1 = ( $\lambda n. \text{if } n = 1 \text{ then } 1 \text{ else } 0$ )  
 by (*simp add: one-fds-def fds-nth-fds fun-eq-iff*)

**lemma** *fds-nth-one-Suc-0* [*simp*]: *fds-nth* 1 (*Suc* 0) = 1  
 by (*simp add: fds-nth-one*)

**lemma** *fds-nth-one-not-Suc-0* [*simp*]:  $n \neq \text{Suc } 0 \implies \text{fds-nth } 1 \ n = 0$   
 by (*simp add: fds-nth-one*)

**lemma** *fds-nth-plus* [*simp*]:  
*fds-nth* (f + g) = ( $\lambda n. \text{fds-nth } f \ n + \text{fds-nth } g \ n :: 'a :: \text{monoid-add}$ )  
 by (*simp add: plus-fds-def fds-nth-fds fun-eq-iff*)

**lemma** *fds-nth-minus* [*simp*]:  
*fds-nth* (f - g) = ( $\lambda n. \text{fds-nth } f \ n - \text{fds-nth } g \ n :: 'a :: \{\text{cancel-comm-monoid-add}\}$ )  
 by (*simp add: minus-fds-def fds-nth-fds fun-eq-iff*)

**lemma** *fds-nth-uminus* [*simp*]: *fds-nth* (-g) = ( $\lambda n. - \text{fds-nth } g \ n :: 'a :: \text{group-add}$ )  
 by (*simp add: uminus-fds-def fds-nth-fds fun-eq-iff*)

**lemma** *fds-nth-mult*: *fds-nth* (f \* g) = *dirichlet-prod* (*fds-nth* f) (*fds-nth* g)  
 by (*simp add: times-fds-def fds-nth-fds dirichlet-prod-def fun-eq-iff*)

**lemma** *fds-nth-mult-const-left* [*simp*]: *fds-nth* (*fds-const* c \* f) n = c \* *fds-nth* f n  
 by (*cases n = 0*) (*simp-all add: fds-nth-mult fds-const-def*)

**lemma** *fds-nth-mult-const-right* [*simp*]: *fds-nth* (f \* *fds-const* c) n = *fds-nth* f n \*  
 c  
 by (*cases n = 0*) (*simp-all add: fds-nth-mult fds-const-def*)

**instance** *fds* :: ( $\{\text{semigroup-add, zero}\}$ ) *semigroup-add*  
 by *standard* (*simp-all add: fds-eq-iff algebra-simps plus-fds-def*)

**instance** *fds* :: ( $\{\text{ab-semigroup-add, zero}\}$ ) *ab-semigroup-add*  
 by *standard* (*simp-all add: fds-eq-iff algebra-simps plus-fds-def*)

**instance** *fds* :: ( $\{\text{cancel-semigroup-add, zero}\}$ ) *cancel-semigroup-add*  
 by *standard* (*simp-all add: fds-eq-iff algebra-simps plus-fds-def*)

```

instance fds :: ({cancel-ab-semigroup-add, zero}) cancel-ab-semigroup-add
  by standard (simp-all add: fds-eq-iff algebra-simps plus-fds-def minus-fds-def)

instance fds :: (monoid-add) monoid-add
  by standard (simp-all add: fds-eq-iff algebra-simps)

instance fds :: (comm-monoid-add) comm-monoid-add
  by standard (simp-all add: fds-eq-iff algebra-simps)

instance fds :: (cancel-comm-monoid-add) cancel-comm-monoid-add
  by standard (simp-all add: fds-eq-iff algebra-simps)

instance fds :: (group-add) group-add
  by standard (simp-all add: fds-eq-iff algebra-simps minus-fds-def)

instance fds :: (ab-group-add) ab-group-add
  by standard (simp-all add: fds-eq-iff algebra-simps)

instance fds :: (semiring-0) semiring-0
proof
  fix f g h :: 'a fds
  show (f + g) * h = f * h + g * h
    by (simp add: fds-eq-iff fds-nth-mult dirichlet-prod-def algebra-simps sum.distrib)
next
  fix f g h :: 'a fds
  show f * g * h = f * (g * h)
    by (intro fds-eqI) (simp add: fds-nth-mult dirichlet-prod-assoc)
qed (simp-all add: fds-eq-iff fds-nth-mult dirichlet-prod-def algebra-simps sum.distrib)

instance fds :: (comm-semiring-0) comm-semiring-0
proof
  fix f g :: 'a fds
  show f * g = g * f
    by (simp add: fds-eq-iff fds-nth-mult dirichlet-prod-commutes)
qed (simp-all add: fds-eq-iff fds-nth-mult dirichlet-prod-def algebra-simps sum.distrib)

instance fds :: (semiring-0-cancel) semiring-0-cancel
  by standard (simp-all add: fds-eq-iff fds-nth-one fds-nth-mult)

instance fds :: (comm-semiring-0-cancel) comm-semiring-0-cancel ..

instance fds :: (semiring-1) semiring-1
  by standard (simp-all add: fds-eq-iff fds-nth-one fds-nth-mult)

instance fds :: (comm-semiring-1) comm-semiring-1
  by standard (simp-all add: fds-eq-iff fds-nth-one fds-nth-mult)

instance fds :: (semiring-1-cancel) semiring-1-cancel ..

```

```

instance fds :: (ring) ring ..
instance fds :: (ring-1) ring-1 ..
instance fds :: (comm-ring) comm-ring ..

instance fds :: (semiring-no-zero-divisors) semiring-no-zero-divisors
proof
  fix f g :: 'a fds
  assume f ≠ 0 g ≠ 0
  hence ex: ∃ m > 0. fds-nth f m ≠ 0 ∃ n > 0. fds-nth g n ≠ 0
    by (auto simp: fds-eq-iff)
  define m where m = (LEAST m. m > 0 ∧ fds-nth f m ≠ 0)
  define n where n = (LEAST n. n > 0 ∧ fds-nth g n ≠ 0)
  from ex[THEN LeastI-ex, folded m-def n-def]
    have mn: m > 0 fds-nth f m ≠ 0 n > 0 fds-nth g n ≠ 0 by auto

  have *: m ≤ m' if m' > 0 fds-nth f m' ≠ 0 for m'
    using conjI[OF that] unfolding m-def by (rule Least-le)
  have m': fds-nth f m' = 0 if m' ∈ {0 <..for m' using that *[of m'] by
  auto

  have *: n ≤ n' if n' > 0 fds-nth g n' ≠ 0 for n'
    using conjI[OF that] unfolding n-def by (rule Least-le)
  have n': fds-nth g n' = 0 if n' ∈ {0 <..for n' using that *[of n'] by auto

  have fds-nth (f * g) (m * n) =
    (∑ d | d dvd m * n. fds-nth f d * fds-nth g (m * n div d))
    by (simp add: fds-nth-mult dirichlet-prod-def)
  also have ... = (∑ d | d dvd m * n. if d = m then fds-nth f m * fds-nth g n
  else 0)
  proof (intro sum.cong refl, goal-cases)
    case (1 d)
    thus ?case
    proof (cases d ≤ m)
      case True
        with mn(1,3) 1 show ?thesis by (auto elim!: dvdE simp: m' n' split: if-splits)
      next
        case False
          from 1 obtain k where k: m * n = d * k by (auto elim!: dvdE)
          with mn(1,3) have [simp]: k > 0 by (auto intro!: Nat.gr0I)
          from False mn(3) have m * n < d * n by (intro mult-strict-right-mono)
    auto
    also note k
    finally have k < n by (subst (asm) mult-less-cancel1) auto
    with mn(1,3) and 1 and False show ?thesis
      by (auto simp: k m' n' split: if-splits)
  qed
  qed
  also have ... = fds-nth f m * fds-nth g n using mn(1,3) by (subst sum.delta)
  auto

```

also have  $\dots \neq 0$  using *mn* by *auto*  
 finally show  $f * g \neq 0$  by *auto*  
 qed

**instance** *fds* :: (ring-no-zero-divisors) ring-no-zero-divisors ..  
**instance** *fds* :: (idom) idom ..

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

**definition** *scaleR-fds* :: real  $\Rightarrow$  'a *fds*  $\Rightarrow$  'a *fds* **where**  
*scaleR-fds* c f = *fds* ( $\lambda n. c *_R \text{fds-nth } f \ n$ )

**lemma** *fds-nth-scaleR* [*simp*]: *fds-nth* (c \*\_R f) = ( $\lambda n. c *_R \text{fds-nth } f \ n$ )  
 by (*simp* add: *scaleR-fds-def* *fun-eq-iff* *fds-nth-fds*)

**instance** by *standard* (*simp-all* add: *fds-eq-iff* *algebra-simps*)

**end**

**instance** *fds* :: (real-algebra) real-algebra  
 by *standard* (*simp-all* add: *fds-eq-iff* *algebra-simps* *fds-nth-mult*  
*dirichlet-prod-def* *scaleR-sum-right*)

**instance** *fds* :: (real-algebra-1) real-algebra-1 ..

**lemma** *fds-nth-sum* [*simp*]: *fds-nth* (sum f A) n = sum ( $\lambda x. \text{fds-nth } (f \ x) \ n$ ) A  
 by (*induction* A rule: *infinite-finite-induct*) *auto*

**lemma** *sum-fds* [*simp*]: ( $\sum x \in A. \text{fds } (f \ x)$ ) = *fds* ( $\lambda n. \sum x \in A. f \ x \ n$ )  
 by (*rule* *fds-eqI*) *simp-all*

**lemma** *fds-nth-const*: *fds-nth* (*fds-const* c) = ( $\lambda n. \text{if } n = 1 \text{ then } c \text{ else } 0$ )  
 by (*simp* add: *fds-const-def* *fds-nth-fds* *fun-eq-iff*)

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

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

**lemma** *fds-const-zero* [*simp*]: *fds-const* 0 = 0  
 by (*simp* add: *fds-eq-iff* *fds-nth-const*)

**lemma** *fds-const-one* [*simp*]: *fds-const* 1 = 1  
 by (*simp* add: *fds-eq-iff* *fds-nth-const* *fds-nth-one*)

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

**by** (*simp add: fds-eq-iff fds-nth-const*)

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

$\text{fds-const } (a - b :: 'a :: \text{cancel-comm-monoid-add}) = \text{fds-const } a - \text{fds-const } b$

**by** (*simp add: fds-eq-iff fds-nth-const*)

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

$\text{fds-const } (- b :: 'a :: \text{ab-group-add}) = - \text{fds-const } b$

**by** (*simp add: fds-eq-iff fds-nth-const*)

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

$\text{fds-const } (a * b :: 'a :: \text{semiring-0}) = \text{fds-const } a * \text{fds-const } b$

**by** (*simp add: fds-eq-iff fds-nth-const fds-nth-mult*)

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

**by** (*induction c*) (*simp-all*)

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

**by** (*cases c*) *simp-all*

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

**by** (*simp add: of-real-def fds-eq-iff fds-const-def fds-nth-one*)

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

**begin**

**definition** *inverse-fds* ::  $'a \text{ fds} \Rightarrow 'a \text{ fds}$  **where**

$\text{inverse-fds } f = \text{fds } (\lambda n. \text{dirichlet-inverse } (\text{fds-nth } f) (\text{inverse } (\text{fds-nth } f 1)) n)$

**definition** *divide-fds* ::  $'a \text{ fds} \Rightarrow 'a \text{ fds} \Rightarrow 'a \text{ fds}$  **where**

$\text{divide-fds } f g = f * \text{inverse } g$

**instance** ..

**end**

**lemma** *numeral-fds*:  $\text{numeral } n = \text{fds-const } (\text{numeral } n)$

**proof** –

**have**  $\text{numeral } n = (\text{of-nat } (\text{numeral } n) :: 'a \text{ fds})$  **by** *simp*

**also have**  $\dots = \text{fds-const } (\text{of-nat } (\text{numeral } n))$  **by** (*rule fds-const-of-nat [symmetric]*)

**also have**  $\text{of-nat } (\text{numeral } n) = (\text{numeral } n :: 'a)$  **by** *simp*

**finally show** *?thesis* .

**qed**

**lemma** *fds-ind-False* [*simp*]:  $\text{fds-ind } (\lambda-. \text{False}) = 0$

**by** (*rule fds-eqI*) *simp*

**lemma** *fds-commutes*:

**assumes**  $\bigwedge m n. m > 0 \implies n > 0 \implies \text{fds-nth } f \ m * \text{fds-nth } g \ n = \text{fds-nth } g \ n * \text{fds-nth } f \ m$   
**shows**  $f * g = g * f$   
**by** (*intro fds-eqI, unfold fds-nth-mult, subst dirichlet-prod-def, subst dirichlet-prod-altdef1, intro sum.cong refl assms*) (*auto elim: dvdE*)

**lemma** *fds-nth-mult-Suc-0* [*simp*]:

$\text{fds-nth } (f * g) \ (\text{Suc } 0) = \text{fds-nth } f \ (\text{Suc } 0) * \text{fds-nth } g \ (\text{Suc } 0)$   
**by** (*simp add: fds-nth-mult*)

**lemma** *fds-nth-inverse*:

$\text{fds-nth } (\text{inverse } f) = \text{dirichlet-inverse } (\text{fds-nth } f) \ (\text{inverse } (\text{fds-nth } f \ 1))$   
**by** (*simp add: inverse-fds-def fds-nth-fds fun-eq-iff*)

**lemma** *inverse-fds-nonunit*:

$\text{fds-nth } f \ 1 = (0 :: 'a :: \text{field}) \implies \text{inverse } f = 0$   
**by** (*auto simp: fds-eq-iff fds-nth-inverse dirichlet-inverse-noninvertible*)

**lemma** *inverse-0-fds* [*simp*]:  $\text{inverse } (0 :: 'a :: \text{field } \text{fds}) = 0$

**by** (*simp add: inverse-fds-def fds-eq-iff dirichlet-inverse.simps*)

**lemma** *fds-left-inverse*:

$\text{fds-nth } f \ 1 \neq (0 :: 'a :: \text{field}) \implies \text{inverse } f * f = 1$   
**by** (*auto simp: fds-eq-iff fds-nth-mult fds-nth-inverse dirichlet-prod-inverse' fds-nth-one*)

**lemma** *fds-right-inverse*:

$\text{fds-nth } f \ 1 \neq (0 :: 'a :: \text{field}) \implies f * \text{inverse } f = 1$   
**by** (*auto simp: fds-eq-iff fds-nth-mult fds-nth-inverse dirichlet-prod-inverse fds-nth-one*)

**lemma** *fds-left-inverse-unique*:

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

**proof** –

**have**  $\text{fds-nth } (f * g) \ 1 = 1$  **by** (*subst assms*) *simp*

**hence**  $\text{fds-nth } g \ 1 \neq 0$  **by** *auto*

**hence**  $(f - \text{inverse } g) * g = 0$

**unfolding** *ring-distrib* **by** (*subst fds-left-inverse*) (*simp-all add: assms*)

**moreover from** *assms* **have**  $g \neq 0$  **by** *auto*

**ultimately show**  $f = \text{inverse } g$  **by** *simp*

**qed**

**lemma** *fds-right-inverse-unique*:

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

**shows**  $g = \text{inverse } f$

**using** *fds-left-inverse-unique*[*of g f*] *assms* **by** (*simp add: mult commute*)

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

by (rule *fds-left-inverse-unique* [*symmetric*]) *simp*

**lemma** *inverse-const-fds* [*simp*]:  
 $inverse (fds\text{-}const\ c :: 'a :: field\ fds) = fds\text{-}const (inverse\ c)$   
**proof** (*cases*  $c = 0$ )  
  **case** *False*  
  **thus** *?thesis*  
  by (intro *fds-right-inverse-unique*[*symmetric*])  
  (*auto simp del: fds-const-mult simp: fds-const-mult* [*symmetric*])  
**qed** *auto*

**lemma** *inverse-mult-fds*:  $inverse (f * g :: 'a :: field\ fds) = inverse\ f * inverse\ g$   
**proof** (*cases*  $fds\text{-}nth (f * g) (Suc\ 0) = 0$ )  
  **case** *False*  
  **hence**  $(f * inverse\ f) * (g * inverse\ g) = 1$  **by** (*subst* (1 2) *fds-right-inverse*)  
  *auto*  
  **thus** *?thesis* **by** (intro *fds-right-inverse-unique* [*symmetric*]) (*simp-all add: mult-ac*)  
**qed** (*auto simp: inverse-fds-nonunit*)

**definition** *fds-zeta* ::  $'a :: one\ fds$   
  **where**  $fds\text{-}zeta = fds (\lambda\cdot. 1)$

**lemma** *fds-zeta-altdef*:  $fds\text{-}zeta = fds (\lambda n. if\ n = 0\ then\ 0\ else\ 1)$   
  **by** (*rule* *fds-eqI*) (*simp add: fds-zeta-def*)

**lemma** *fds-nth-zeta*:  $fds\text{-}nth\ fds\text{-}zeta = (\lambda n. if\ n = 0\ then\ 0\ else\ 1)$   
  **by** (*simp add: fds-zeta-def fun-eq-iff*)

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

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

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

**lemma** *finite-extensional-prod-nat*:  
  **assumes**  $finite\ A\ b > 0$   
  **shows**  $finite\ \{d \in extensional\ A. prod\ d\ A = (b :: nat)\}$   
**proof** (*rule* *finite-subset*)  
  **from** *assms*(1) **show**  $finite (PiE\ A (\lambda\cdot. \{..b\}))$  **by** (*rule* *finite-PiE*) *auto*  
  {  
    **fix**  $d :: 'a \Rightarrow nat$  **and**  $x :: 'a$  **assume**  $*$ :  $x \in A\ prod\ d\ A = b$   
    **with** *prod-dvd-prod-subset*[*of*  $A\ \{x\}\ d$ ] *assms* **have**  $d\ x\ dvd\ b$  **by** *auto*  
    **with** *assms* **have**  $d\ x \leq b$  **by** (*auto dest: dvd-imp-le*)  
  }  
  **thus**  $\{d \in extensional\ A. prod\ d\ A = (b :: nat)\} \subseteq \dots$

by (auto simp: extensional-def)  
qed

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

**lemma** *fds-nth-prod*:

assumes *finite A A ≠ {} n > 0*

shows  $fds\_nth (\prod_{x \in A} f x) n =$

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

using *assms*

**proof** (*induction arbitrary: n rule: finite-ne-induct*)

case (*singleton x n*)

have  $\{d \in \text{extensional } \{x\}. d x = n\} = \{\lambda y. \text{if } y = x \text{ then } n \text{ else undefined}\}$

by (auto simp: extensional-def)

thus ?case by simp

next

case (*insert x A n*)

let ?f =  $\lambda d. ((d x, n \text{ div } d x), d(x := \text{undefined}))$

let ?g =  $\lambda(z, d). d(x := fst z)$

from *insert* have  $fds\_nth (\prod_{x \in \text{insert } x A} f x) n =$

$$\left( \sum z \mid fst z * snd z = n. \sum d \mid d \in \text{extensional } A \wedge \text{prod } d A = snd z. \right. \\ \left. fds\_nth (f x) (fst z) * \left( \prod_{x \in A} fds\_nth (f x) (d x) \right) \right)$$

by (*simp add: fds-nth-mult dirichlet-prod-altdef2 sum-distrib-left case-prod-unfold*)

also have  $\dots = \left( \sum (z, d) \in (\text{SIGMA } x: \{z. fst z * snd z = n\}. \{d \in \text{extensional } A. \text{prod } d A = snd x\}). \right.$

$$\left. fds\_nth (f x) (fst z) * \left( \prod_{x \in A} fds\_nth (f x) (d x) \right) \right)$$

using *finite-divisors-nat*<sup>[of n]</sup> and *insert.hyps* and  $\langle n > 0 \rangle$

by (*intro sum.Sigma finite-extensional-prod-nat ballI*) (auto simp: case-prod-unfold)

also have  $\dots = \left( \sum d \mid d \in \text{extensional } (\text{insert } x A) \wedge \text{prod } d (\text{insert } x A) = n. \right.$

$$\left. \left( \prod_{x \in \text{insert } x A} fds\_nth (f x) (d x) \right) \right)$$

**proof** (*rule sum.reindex-bij-witness [of - ?f ?g], goal-cases*)

case (*1 z*)

thus ?case using *insert.hyps insert.prem*s by (auto simp: extensional-def)

next

case (*2 z*)

thus ?case using *insert.hyps insert.prem*s

by (auto simp: extensional-def sum.delta intro!: prod.cong)

next

case (*4 z*)

thus ?case using *insert.hyps insert.prem*s by (auto intro!: prod.cong)

next

case (*5 z*)

with *insert.hyps insert.prem*s

have  $\left( \prod_{xa \in A} fds\_nth (f xa) (\text{if } xa = x \text{ then } fst (fst z) \text{ else } snd z xa) \right) =$

$$\left( \prod_{x \in A} fds\_nth (f x) (snd z x) \right) \text{ by (intro prod.cong) auto}$$

with *5 insert.hyps insert.prem*s show ?case by (*simp add: case-prod-unfold*)

qed auto

finally show ?case .

qed

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

**lemma** *fds-nth-prod-Suc-0* [simp]:  $\text{fds-nth } (\text{prod } f A) (\text{Suc } 0) = (\prod_{x \in A} \text{fds-nth } (f x) (\text{Suc } 0))$   
by (induction A rule: infinite-finite-induct) simp-all

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

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

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

using *assms*(1)

**proof** (induction k arbitrary: n)

case 0

thus ?case by (simp add: one-fds-def)

next

case (Suc k n)

have  $\text{fds-nth } (f \wedge \text{Suc } k) n = \text{dirichlet-prod } (\text{fds-nth } (f \wedge k)) (\text{fds-nth } f) n$

by (subst *power-Suc2*) (simp add: *fds-nth-mult dirichlet-prod-commutes*)

also have  $\dots = 0$  unfolding *dirichlet-prod-def*

**proof** (intro *sum.neutral ballI*)

fix d assume  $d: d \in \{d. d \text{ dvd } n\}$

show  $\text{fds-nth } (f \wedge k) d * \text{fds-nth } f (n \text{ div } d) = 0$

**proof** (cases  $d < 2 \wedge k$ )

case True

thus ?thesis using *Suc.IH*[of d] by simp

next

case False

hence  $(n \text{ div } d) * 2 \wedge k \leq (n \text{ div } d) * d$  by (intro *mult-left-mono*) auto

also from d have  $(n \text{ div } d) * d = n$  by simp

also from *Suc* have  $n < 2 * 2 \wedge k$  by simp

finally have  $n \text{ div } d \leq 1$  by simp

with *assms*(2) show ?thesis by (cases n div d) simp-all

qed

qed

finally show ?case .

qed

## 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 * nat-power n b
  assumes nat-power-mult-distrib:
    m > 0  $\implies$  n > 0  $\implies$  nat-power (m * n) a = nat-power m a * nat-power n a
  assumes nat-power-power:
    n > 0  $\implies$  nat-power n (a * of-nat m) = nat-power n a ^ m
begin

lemma nat-power-of-nat [simp]: m > 0  $\implies$  nat-power m (of-nat n) = of-nat (m ^ n)
  by (induction n) (simp-all add: nat-power-add)

lemma nat-power-power-left: m > 0  $\implies$  nat-power (m ^ k) n = nat-power m n ^ k
  by (induction k) (simp-all add: nat-power-mult-distrib)

end

class nat-power-field = nat-power + field +
  assumes nat-power-nonzero [simp]: n > 0  $\implies$  nat-power n z  $\neq$  0
begin

lemma nat-power-diff: n > 0  $\implies$  nat-power n (a - b) = nat-power n a / nat-power n b
  using nat-power-add[of n a - b] by (simp add: divide-simps)

end

instantiation nat :: nat-power
begin
definition [simp]: nat-power-nat a b = (a ^ b :: nat)
instance by standard (simp-all add: power-add power-mult-distrib power-mult)
end

instantiation real :: nat-power-field
begin
definition [simp]: nat-power-real a b = (real a powr b)
instance proof
  fix n m :: nat and a :: real assume n > 0
  thus nat-power n (a * real m) = nat-power n a ^ m
  by (simp add: powr-def exp-of-nat-mult [symmetric])
qed (simp-all add: powr-add powr-mult)
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  $\Rightarrow$  'a *fds*  $\Rightarrow$  'a *fds* **where**  
*fds-shift* c f = *fds* ( $\lambda n.$  *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  
**by** (*simp* *add*: *fds-shift-def* *fds-nth-fds*)

**lemma** *fds-shift-shift* [*simp*]: *fds-shift* c (*fds-shift* c' f) = *fds-shift* (c' + c) f  
**by** (*rule* *fds-eqI*) (*simp* *add*: *nat-power-add* *mult-ac*)

**lemma** *fds-shift-zero* [*simp*]: *fds-shift* c 0 = 0  
**by** (*rule* *fds-eqI*) *simp*

**lemma** *fds-shift-1* [*simp*]: *fds-shift* a 1 = 1  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-shift-def* *one-fds-def*)

**lemma** *fds-shift-const* [*simp*]: *fds-shift* a (*fds-const* c) = *fds-const* c  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-shift-def* *fds-const-def*)

**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  
**by** (*rule* *fds-eqI*) (*simp* *add*: *algebra-simps*)

**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  
**by** (*rule* *fds-eqI*) (*simp* *add*: *algebra-simps*)

**lemma** *fds-shift-uminus* [*simp*]:  
**fixes** f :: 'a :: {*ring*, *nat-power*} *fds*  
**shows** *fds-shift* c (-f) = -*fds-shift* c f  
**by** (*rule* *fds-eqI*) (*simp* *add*: *algebra-simps*)

**lemma** *fds-shift-mult* [*simp*]:  
**fixes** f g :: 'a :: {*comm-semiring*, *nat-power*} *fds*  
**shows** *fds-shift* c (f \* g) = *fds-shift* c f \* *fds-shift* c g  
**by** (*rule* *fds-eqI*)  
*(auto simp: algebra-simps fds-nth-mult dirichlet-prod-altdef2 sum-distrib-left sum-distrib-right nat-power-mult-distrib intro!: sum.cong)*

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

**lemma** *fds-shift-by-0* [*simp*]: *fds-shift 0 f = f*  
**by** (*simp add: fds-shift-def*)

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

**proof** (*cases fds-nth f 1 = 0*)

**case** *False*

**have** *fds-shift a f \* fds-shift a (inverse f) = fds-shift a (f \* inverse f)*  
**by** *simp*

**also from** *False* **have** *f \* inverse f = 1* **by** (*intro fds-right-inverse*)

**finally have** *fds-shift a f \* fds-shift a (inverse f) = 1* **by** *simp*

**thus** *?thesis* **by** (*rule fds-right-inverse-unique*)

**qed** (*auto simp: inverse-fds-nonunit*)

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

*fds-shift (a :: 'a :: {field, nat-power}) (f / g) = fds-shift a f / fds-shift a g*  
**by** (*simp add: divide-fds-def*)

**lemma** *fds-shift-sum* [*simp*]: *fds-shift a (∑ x∈A. f x) = (∑ x∈A. fds-shift a (f x))*  
**by** (*induction A rule: infinite-finite-induct*) *simp-all*

**lemma** *fds-shift-prod* [*simp*]: *fds-shift a (∏ x∈A. f x) = (∏ x∈A. fds-shift a (f x))*  
**by** (*induction A rule: infinite-finite-induct*) *simp-all*

### 4.3 Scaling the argument

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

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

into the series

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

**definition** *fds-scale* :: *nat* ⇒ (*'a* :: *zero*) *fds* ⇒ *'a fds* **where**

*fds-scale c f =*

*fds (λn. if n > 0 ∧ is-nth-power c n then fds-nth f (nth-root-nat c n) else 0)*

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

**by** (*auto simp: fds-scale-def fds-eq-iff*)

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

**by** (*auto simp: fds-scale-def fds-eq-iff*)

**lemma** *fds-nth-scale-power* [*simp*]:

*c > 0 ⇒ fds-nth (fds-scale c f) (n ^ c) = fds-nth f n*

by (simp add: fds-scale-def fds-nth-fds)

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

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

**lemma** *fds-scale-const* [simp]:  $c > 0 \implies \text{fds-scale } c \ (\text{fds-const } c') = \text{fds-const } c'$   
by (rule fds-eqI) (auto simp: fds-nth-scale fds-nth-const elim!: is-nth-powerE)

**lemma** *fds-scale-zero* [simp]:  $\text{fds-scale } c \ 0 = 0$   
by (rule fds-eqI) (simp add: fds-nth-scale)

**lemma** *fds-scale-one* [simp]:  $c > 0 \implies \text{fds-scale } c \ 1 = 1$   
by (simp only: fds-const-one [symmetric] fds-scale-const)

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

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

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

**lemma** *fds-scale-scale*:  $\text{fds-scale } c \ (\text{fds-scale } c' \ f) = \text{fds-scale } (c * c') \ f$   
**proof** (cases  $c = 0 \vee c' = 0$ )  
case *False*  
hence  $cc'$ :  $c > 0 \ c' > 0$  by auto  
show ?thesis  
**proof** (rule *fds-eqI*, goal-cases)  
case (1  $n$ )  
show ?case  
**proof** (cases *is-nth-power*  $(c * c') \ n$ )  
case *False*  
with  $cc' \ 1$  have  $\text{fds-nth } (\text{fds-scale } c \ (\text{fds-scale } c' \ f)) \ n = 0$   
by (auto simp: *fds-nth-scale is-nth-power-def power-mult [symmetric] mult.commute*)  
with *False*  $cc'$  show ?thesis by simp  
next  
case *True*  
from *True* obtain  $n'$  where [simp]:  $n = n' \wedge (c' * c)$   
by (auto elim: *is-nth-powerE simp: mult.commute*)  
with  $cc'$  have  $\text{fds-nth } (\text{fds-scale } (c * c') \ f) \ n = \text{fds-nth } f \ n'$   
by (simp add: *mult.commute*)  
also have  $\dots = \text{fds-nth } (\text{fds-scale } c \ (\text{fds-scale } c' \ f)) \ n$

```

    using cc' by (simp add: power-mult)
    finally show ?thesis ..
  qed
qed
qed auto

lemma fds-scale-add [simp]:
  fixes f g :: 'a :: monoid-add fds
  shows fds-scale c (f + g) = fds-scale c f + fds-scale c g
  by (rule fds-eqI) (auto simp: fds-nth-scale)

lemma fds-scale-minus [simp]:
  fixes f g :: 'a :: {cancel-comm-monoid-add} fds
  shows fds-scale c (f - g) = fds-scale c f - fds-scale c g
  by (rule fds-eqI) (auto simp: fds-nth-scale)

lemma fds-scale-uminus [simp]:
  fixes f :: 'a :: group-add fds
  shows fds-scale c (-f) = -fds-scale c f
  by (rule fds-eqI) (auto simp: fds-nth-scale)

lemma fds-scale-mult [simp]:
  fixes f g :: 'a :: semiring-0 fds
  shows fds-scale c (f * g) = fds-scale c f * fds-scale c g
proof (cases c > 0)
  case True
  show ?thesis
  proof (rule fds-eqI, goal-cases)
    case (1 n)
    show ?case
    proof (cases is-nth-power c n)
      case False
      have fds-nth (fds-scale c f * fds-scale c g) n =
        (∑ (r, d) | r * d = n. fds-nth (fds-scale c f) r * fds-nth (fds-scale c g)
d)
      by (simp add: fds-nth-mult dirichlet-prod-altdef2)
      also from False have ... = (∑ (r, d) | r * d = n. 0)
      by (intro sum.cong refl) (auto simp: fds-nth-scale dest: is-nth-power-mult)
      also from False have ... = fds-nth (fds-scale c (f * g)) n by simp
      finally show ?thesis ..
    next
    case True
    then obtain n' where [simp]: n = n' ^ c by (elim is-nth-powerE)
    define h where h = map-prod (nth-root-nat c) (nth-root-nat c)
    define i where i = map-prod (λn::nat. n ^ c) (λn::nat. n ^ c)
    define A where A = {(r, d). r * d = n}
    define S where S = {rs∈A. ¬is-nth-power c (fst rs) ∨ ¬is-nth-power c (snd
rs)}
  end
end

```

**have**  $\text{fds-nth } (\text{fds-scale } c \ f * \text{fds-scale } c \ g) \ n =$   
 $(\sum (r, d) \mid r * d = n. \text{fds-nth } (\text{fds-scale } c \ f) \ r * \text{fds-nth } (\text{fds-scale } c \ g)$   
*d)*  
**by** (*simp add: fds-nth-mult dirichlet-prod-altdef2*)  
**also have**  $\dots = (\sum (r, d) \mid r * d = n'. \text{fds-nth } f \ r * \text{fds-nth } g \ d)$   
**proof** (*rule sym, intro sum.reindex-bij-witness-not-neutral[of {} S - h i]*)  
**show** *finite S unfolding S-def A-def*  
**by** (*rule finite-subset[OF - finite-divisors-nat'[of n]] (insert <n > 0>, auto)*)  
**show**  $i \ (h \ rd) = rd$  **if**  $rd \in \{(r, d). r * d = n\} - S$  **for**  $rd$   
**using**  $\langle c > 0 \rangle$  **that** **by** (*auto elim!: is-nth-powerE simp: S-def i-def h-def*  
*A-def*)  
**show**  $h \ rd \in \{(r, d). r * d = n'\} - \{\}$  **if**  $rd \in \{(r, d). r * d = n\} - S$  **for**  
 $rd$   
**using**  $\langle c > 0 \rangle$  **that** **by** (*auto elim!: is-nth-powerE*  
*simp: S-def i-def h-def A-def power-mult-distrib [symmetric] power-eq-iff-eq-base*)  
**show**  $h \ (i \ rd) = rd$  **if**  $rd \in \{(r, d). r * d = n'\} - \{\}$  **for**  $rd$   
**using** *that*  $\langle c > 0 \rangle$  **by** (*auto simp: h-def i-def*)  
**show**  $i \ rd \in \{(r, d). r * d = n\} - S$  **if**  $rd \in \{(r, d). r * d = n'\} - \{\}$  **for**  $rd$   
**using** *that*  $\langle c > 0 \rangle$  **by** (*auto simp: i-def S-def power-mult-distrib [symmetric]*)  
**show** (*case rd of (r, d)  $\Rightarrow$  fds-nth (fds-scale c f) r \* fds-nth (fds-scale c g)*  
*d) = 0*)  
**if**  $rd \in S$  **for**  $rd$  **using** *that* **by** (*auto simp: S-def case-prod-unfold*)  
**qed** (*insert <c > 0>, auto simp: case-prod-unfold i-def*)  
**also have**  $\dots = \text{fds-nth } (f * g) \ n'$  **by** (*simp add: fds-nth-mult dirich-*  
*let-prod-altdef2*)  
**also from**  $\langle c > 0 \rangle$  **have**  $\dots = \text{fds-nth } (\text{fds-scale } c \ (f * g)) \ n$  **by** *simp*  
**finally show** *?thesis ..*  
**qed**  
**qed**  
**qed** *auto*

**lemma** *fds-scale-shift:*

$\text{fds-shift } d \ (\text{fds-scale } c \ f) = \text{fds-scale } c \ (\text{fds-shift } (c * d) \ f)$   
**proof** (*cases c > 0*)  
**case** *True*  
**thus** *?thesis*  
**by** (*intro fds-eqI (auto simp: fds-nth-scale power-mult elim!: is-nth-powerE)*)  
**qed** *auto*

**lemma** *fds-ind-nth-power:*  $k > 0 \implies \text{fds-ind } (\text{is-nth-power } k) = \text{fds-scale } k \ \text{fds-zeta}$   
**by** (*rule fds-eqI (auto simp: ind-def fds-nth-scale elim!: is-nth-powerE)*)

## 4.4 Formal derivative

The formal derivative of a series

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

can easily be seen to be

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

**definition** *fds-deriv* :: 'a :: real-algebra *fds*  $\Rightarrow$  'a *fds* **where**  
*fds-deriv* *f* = *fds* ( $\lambda n. - \ln (\text{real } n) *_{\mathbb{R}} \text{fds-nth } f \ n$ )

**lemma** *fds-nth-deriv*: *fds-nth* (*fds-deriv* *f*) *n* =  $-\ln (\text{real } n) *_{\mathbb{R}} \text{fds-nth } f \ n$   
**by** (*cases* *n* = 0) (*simp-all* *add*: *fds-deriv-def*)

**lemma** *fds-deriv-const* [*simp*]: *fds-deriv* (*fds-const* *c*) = 0  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-nth-deriv* *fds-nth-const*)

**lemma** *fds-deriv-0* [*simp*]: *fds-deriv* 0 = 0  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-nth-deriv*)

**lemma** *fds-deriv-1* [*simp*]: *fds-deriv* 1 = 0  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-nth-deriv* *fds-nth-one*)

**lemma** *fds-deriv-of-nat* [*simp*]: *fds-deriv* (*of-nat* *n*) = 0  
**by** (*simp* *only*: *fds-const-of-nat* [*symmetric*] *fds-deriv-const*)

**lemma** *fds-deriv-of-int* [*simp*]: *fds-deriv* (*of-int* *n*) = 0  
**by** (*simp* *only*: *fds-const-of-int* [*symmetric*] *fds-deriv-const*)

**lemma** *fds-deriv-of-real* [*simp*]: *fds-deriv* (*of-real* *n*) = 0  
**by** (*simp* *only*: *fds-const-of-real* [*symmetric*] *fds-deriv-const*)

**lemma** *fds-deriv-uminus* [*simp*]: *fds-deriv* (-*f*) =  $- \text{fds-deriv } f$   
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-nth-deriv*)

**lemma** *fds-deriv-add* [*simp*]: *fds-deriv* (*f* + *g*) = *fds-deriv* *f* + *fds-deriv* *g*  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-nth-deriv* *algebra-simps*)

**lemma** *fds-deriv-minus* [*simp*]: *fds-deriv* (*f* - *g*) = *fds-deriv* *f* - *fds-deriv* *g*  
**by** (*rule* *fds-eqI*) (*simp* *add*: *fds-nth-deriv* *algebra-simps*)

**lemma** *fds-deriv-times* [*simp*]:  
*fds-deriv* (*f* \* *g*) = *fds-deriv* *f* \* *g* + *f* \* *fds-deriv* *g*  
**by** (*rule* *fds-eqI*)  
(*auto* *simp* *add*: *fds-nth-deriv* *fds-nth-mult* *dirichlet-prod-altdef2* *scaleR-right.sum*  
*algebra-simps* *sum.distrib* [*symmetric*] *ln-mult* *intro!*: *sum.cong*)

**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*) =  $- \text{fds-deriv } f / f \wedge 2$

**proof** –  
**have**  $(0 :: 'a \text{ fds}) = \text{fds-deriv } 1$  **by** *simp*  
**also from** *assms* **have**  $(1 :: 'a \text{ fds}) = \text{inverse } f * f$  **by** (*simp add: fds-left-inverse*)  
**also have**  $\text{fds-deriv } \dots = \text{fds-deriv } (\text{inverse } f) * f + \text{inverse } f * \text{fds-deriv } f$  **by**  
*simp*  
**also have**  $\dots * \text{inverse } f = \text{fds-deriv } (\text{inverse } f) * (f * \text{inverse } f) +$   
 $\text{inverse } f ^ 2 * \text{fds-deriv } f$   
**by** (*simp add: algebra-simps power2-eq-square*)  
**also from** *assms* **have**  $f * \text{inverse } f = 1$  **by** (*simp add: fds-right-inverse*)  
**finally show** *?thesis*  
**by** (*simp add: algebra-simps power2-eq-square divide-fds-def inverse-mult-fds*  
*add-eq-0-iff*)  
**qed**

**lemma** *fds-deriv-shift* [*simp*]:  $\text{fds-deriv } (\text{fds-shift } c \ f) = \text{fds-shift } c \ (\text{fds-deriv } f)$   
**by** (*rule fds-eqI*) (*simp add: fds-nth-deriv algebra-simps*)

**lemma** *fds-deriv-scale*:  $\text{fds-deriv } (\text{fds-scale } c \ f) = \text{of-nat } c * \text{fds-scale } c \ (\text{fds-deriv } f)$

**proof** (*cases c > 0*)  
**case** *True*  
**have**  $*$ :  $\text{of-nat } a * (b :: 'a) = \text{real } a *_{\mathbb{R}} b$  **for**  $a \ b$   
**by** (*induction a*) (*simp-all add: algebra-simps*)  
**from** *True* **show** *?thesis*  
**by** (*intro fds-eqI*)  
*(auto simp: fds-nth-deriv fds-nth-scale is-nth-powerE fds-const-of-nat [symmetric]*  
 $\text{ln-realpow} * \text{simp del: fds-const-of-nat elim!: is-nth-powerE}$ )

**qed** *auto*

**lemma** *fds-deriv-eq-imp-eq*:

**assumes**  $\text{fds-deriv } f = \text{fds-deriv } g \ \text{fds-nth } f \ (\text{Suc } 0) = \text{fds-nth } g \ (\text{Suc } 0)$   
**shows**  $f = g$

**proof** (*rule fds-eqI*)

**fix**  $n :: \text{nat}$  **assume**  $n: n > 0$

**show**  $\text{fds-nth } f \ n = \text{fds-nth } g \ n$

**proof** (*cases n = 1*)

**case** *False*

**with**  $n$  **have**  $n > 1$  **by** *auto*

**hence**  $\text{fds-nth } f \ n = -\text{fds-nth } (\text{fds-deriv } f) \ n \ /_{\mathbb{R}} \ \text{ln } n$

**by** (*simp add: fds-deriv-def*)

**also note** *assms(1)*

**also from**  $\langle n > 1 \rangle$  **have**  $-\text{fds-nth } (\text{fds-deriv } g) \ n \ /_{\mathbb{R}} \ \text{ln } n = \text{fds-nth } g \ n$

**by** (*simp add: fds-deriv-def*)

**finally show** *?thesis* .

**qed** (*auto simp: assms*)

**qed**

**lemma** *completely-multiplicative-fds-deriv*:

**assumes** *completely-multiplicative-function f*

**shows**  $\text{fds-deriv } (\text{fds } f) = -\text{fds } (\lambda n. f n * \text{mangoldt } n) * \text{fds } f$   
**proof** (*rule fds-eqI, goal-cases*)  
**case** ( $1\ n$ )  
**interpret** *completely-multiplicative-function f by fact*  
**have**  $\text{fds-nth } (-\text{fds } (\lambda n. f n * \text{mangoldt } n) * \text{fds } f) n =$   
 $-(\sum (r, d) \mid r * d = n. f r * \text{mangoldt } r * f d)$   
**by** (*simp add: fds-nth-mult fds-nth-deriv dirichlet-prod-altdef2*)  
**also have**  $(\sum (r, d) \mid r * d = n. f r * \text{mangoldt } r * f d) =$   
 $(\sum (r, d) \mid r * d = n. \text{mangoldt } r * f n)$   
**using**  $1$  **by** (*intro sum.mono-neutral-cong-right refl*)  
*(auto simp: mangoldt-def mult mult-ac intro!: finite-divisors-nat' split: if-splits)*  
**also have**  $\dots = (\sum r \mid r \text{ dvd } n. \text{mangoldt } r * f n)$  **using**  $1$   
**by** (*intro sum.reindex-bij-witness[of -  $\lambda r. (r, n \text{ div } r) \text{ fst}$ ] auto*)  
**also have**  $\dots = (\sum r \mid r \text{ dvd } n. \text{mangoldt } r) * f n$  (**is**  $- = ?S * -$ )  
**by** (*subst sum-distrib-right [symmetric] simp*)  
**also have**  $(\sum r \mid r \text{ dvd } n. \text{mangoldt } r) = \text{of-real } (\ln (\text{real } n))$   
**using**  $1$  **by** (*intro mangoldt-sum simp*)  
**also have**  $-(\text{of-real } (\ln (\text{real } n)) * f n) = \text{fds-nth } (\text{fds-deriv } (\text{fds } f)) n$   
**using**  $1$  **by** (*simp add: fds-nth-deriv scaleR-conv-of-real*)  
**finally show** *?case ..*  
**qed**

**lemma** *completely-multiplicative-fds-deriv'*:  
*completely-multiplicative-function (fds-nth f)  $\implies$*   
 $\text{fds-deriv } f = -\text{fds } (\lambda n. \text{fds-nth } f n * \text{mangoldt } n) * f$   
**using** *completely-multiplicative-fds-deriv[of fds-nth f] by simp*

**lemma** *fds-deriv-zeta*:  
 $\text{fds-deriv } \text{fds-zeta} =$   
 $-\text{fds } \text{mangoldt} * (\text{fds-zeta} :: 'a :: \{\text{comm-semiring-1, real-algebra-1}\} \text{fds})$   
**proof**  $-$   
**have** *completely-multiplicative-function*  $(\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1)$   
**by** *standard simp-all*  
**from** *completely-multiplicative-fds-deriv [OF this, folded fds-zeta-altdef]*  
**show** *?thesis by simp*  
**qed**

**lemma** *fds-mangoldt-times-zeta*:  $\text{fds } \text{mangoldt} * \text{fds-zeta} = \text{fds } (\lambda x. \text{of-real } (\ln (\text{real } x)))$   
**by** (*rule fds-eqI*) (*simp add: fds-nth-mult dirichlet-prod-def mangoldt-sum*)

**lemma** *fds-deriv-zeta'*:  $\text{fds-deriv } \text{fds-zeta} =$   
 $-\text{fds } (\lambda x. \text{of-real } (\ln (\text{real } x))) :: 'a :: \{\text{comm-semiring-1, real-algebra-1}\}$   
**by** (*simp add: fds-deriv-zeta fds-mangoldt-times-zeta*)

## 4.5 Formal integral

**definition** *fds-integral*  $:: 'a \Rightarrow 'a :: \text{real-algebra } \text{fds} \Rightarrow 'a \text{ fds}$  **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]:  $fds\text{-integral } a 0 = fds\text{-const } a$   
**by** (simp add:  $fds\text{-integral-def } fds\text{-eq-iff}$ )

**lemma**  $fds\text{-integral-add}$ :  $fds\text{-integral } (a + b) (f + g) = fds\text{-integral } a f + fds\text{-integral } b g$   
**by** (rule  $fds\text{-eqI}$ ) (auto simp:  $fds\text{-integral-def } scaleR\text{-diff-right}$ )

**lemma**  $fds\text{-integral-diff}$ :  $fds\text{-integral } (a - b) (f - g) = fds\text{-integral } a f - fds\text{-integral } b g$   
**by** (rule  $fds\text{-eqI}$ ) (auto simp:  $fds\text{-integral-def } scaleR\text{-diff-right}$ )

**lemma**  $fds\text{-integral-minus}$ :  $fds\text{-integral } (-a) (-f) = -fds\text{-integral } a f$   
**by** (rule  $fds\text{-eqI}$ ) (auto simp:  $fds\text{-integral-def } scaleR\text{-diff-right}$ )

**lemma**  $fds\text{-shift-integral}$ :  $fds\text{-shift } b (fds\text{-integral } a f) = fds\text{-integral } a (fds\text{-shift } b f)$   
**by** (rule  $fds\text{-eqI}$ ) (simp add:  $fds\text{-integral-def } fds\text{-shift-def}$ )

**lemma**  $fds\text{-deriv-fds-integral}$  [simp]:  
 $fds\text{-nth } f (Suc 0) = 0 \implies fds\text{-deriv } (fds\text{-integral } c f) = f$   
**by** (simp add:  $fds\text{-deriv-def } fds\text{-integral-def } fds\text{-eq-iff}$ )

**lemma**  $fds\text{-integral-fds-deriv}$  [simp]:  $fds\text{-integral } (fds\text{-nth } f 1) (fds\text{-deriv } f) = f$   
**by** (simp add:  $fds\text{-deriv-def } fds\text{-integral-def } fds\text{-eq-iff}$ )

## 4.6 Formal logarithm

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

**lemma**  $fds\text{-nth-Suc-0-fds-deriv}$  [simp]:  $fds\text{-nth } (fds\text{-deriv } f) (Suc 0) = 0$   
**by** (simp add:  $fds\text{-deriv-def}$ )

**lemma**  $fds\text{-deriv-fds-ln}$  [simp]:  $fds\text{-deriv } (fds\text{-ln } l f) = fds\text{-deriv } f / f$   
**unfolding**  $fds\text{-ln-def}$  **by** (subst  $fds\text{-deriv-fds-integral}$ ) (simp-all add:  $divide\text{-fds-def}$ )

**lemma**  $fds\text{-nth-Suc-0-fds-ln}$  [simp]:  $fds\text{-nth } (fds\text{-ln } l f) (Suc 0) = l$   
**by** (simp add:  $fds\text{-ln-def } fds\text{-integral-def}$ )

**lemma**  $fds\text{-ln-const}$  [simp]:  $fds\text{-ln } l (fds\text{-const } c) = fds\text{-const } l$   
**by** (rule  $fds\text{-eqI}$ ) (simp add:  $fds\text{-ln-def } fds\text{-integral-def } divide\text{-fds-def}$ )

**lemma**  $fds\text{-ln-0}$  [simp]:  $fds\text{-ln } l 0 = fds\text{-const } l$   
**by** (rule  $fds\text{-eqI}$ ) (simp add:  $fds\text{-ln-def } fds\text{-integral-def } divide\text{-fds-def}$ )

**lemma**  $fds\text{-ln-1}$  [simp]:  $fds\text{-ln } l 1 = fds\text{-const } l$   
**by** (rule  $fds\text{-eqI}$ ) (simp add:  $fds\text{-ln-def } fds\text{-integral-def } divide\text{-fds-def}$ )

**lemma** *fds-shift-ln* [*simp*]:  $\text{fds-shift } a \text{ (fds-ln } l \text{ } f) = \text{fds-ln } l \text{ (fds-shift } a \text{ } f)$   
**by** (*simp add: fds-ln-def fds-shift-integral*)

**lemma** *fds-ln-mult*:

**assumes**  $\text{fds-nth } f \ 1 \neq 0 \ \text{fds-nth } g \ 1 \neq 0 \ l' + l'' = l$

**shows**  $\text{fds-ln } l \text{ (} f * g \text{)} = \text{fds-ln } l' \text{ } f + \text{fds-ln } l'' \text{ } g$

**proof** –

**have**  $\text{fds-ln } l \text{ (} f * g \text{)} = \text{fds-integral } (l' + l'') \text{ ((fds-deriv } f * g + f * \text{fds-deriv } g) / (f * g))$

**by** (*simp add: fds-ln-def assms*)

**also have**  $(\text{fds-deriv } f * g + f * \text{fds-deriv } g) / (f * g) =$

$\text{fds-deriv } f / f * (g * \text{inverse } g) + \text{fds-deriv } g / g * (f * \text{inverse } f)$

**by** (*simp add: divide-fds-def algebra-simps inverse-mult-fds*)

**also from** *assms* **have**  $f * \text{inverse } f = 1$  **by** (*intro fds-right-inverse*) *auto*

**also from** *assms* **have**  $g * \text{inverse } g = 1$  **by** (*intro fds-right-inverse*) *auto*

**finally show** *?thesis* **by** (*simp add: fds-integral-add fds-ln-def*)

**qed**

**lemma** *fds-ln-power*:

**assumes**  $\text{fds-nth } f \ 1 \neq 0 \ l = \text{of-nat } n * l'$

**shows**  $\text{fds-ln } l \text{ (} f \wedge^n \text{)} = \text{of-nat } n * \text{fds-ln } l' \text{ } f$

**proof** –

**have**  $\text{fds-ln } (\text{of-nat } n * l') \text{ (} f \wedge^n \text{)} = \text{of-nat } n * \text{fds-ln } l' \text{ } f$

**using** *assms*(1) **by** (*induction n*) (*simp-all add: fds-ln-mult algebra-simps*)

**with** *assms* **show** *?thesis* **by** *simp*

**qed**

**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**  $\text{fds-ln } l \text{ (} \prod x \in A. f \ x \text{)} = (\sum x \in A. \text{fds-ln } (l' \ x) \text{ (} f \ x \text{)})$

**proof** –

**have**  $\text{fds-ln } (\sum x \in A. l' \ x) \text{ (} \prod x \in A. f \ x \text{)} = (\sum x \in A. \text{fds-ln } (l' \ x) \text{ (} f \ x \text{)})$

**using** *assms*(1) **by** (*induction A rule: infinite-finite-induct*) (*simp-all add: fds-ln-mult*)

**with** *assms* **show** *?thesis* **by** *simp*

**qed**

## 4.7 Formal exponential

**definition** *fds-exp* :: 'a :: {*real-normed-algebra-1,banach*} *fds*  $\implies$  '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*]:  $\text{fds-nth } (\text{fds-exp } f) \text{ (Suc } 0 \text{)} = \text{exp } (\text{fds-nth } f \ 1)$

**proof** –

**have**  $\text{fds-nth } (\text{fds-exp } f) \text{ (Suc } 0 \text{)} = \text{exp } (\text{fds-nth } f \ 1) * (\sum k. 0 \wedge^k \ /_R \text{fact } k)$

**by** (*simp add: fds-exp-def*)

**also have**  $(\sum k. (0 :: 'a) \wedge^k \ /_R \text{fact } k) = (\sum k. \text{if } k = 0 \text{ then } 1 \text{ else } 0)$

by (*intro suminf-cong*) (*auto simp: power-0-left*)  
 also have  $\dots = 1$  using *sums-If-finite*[of  $\lambda k. k = 0 \ \lambda-. \ 1 \ :: 'a$ ]  
 by (*simp add: sums-iff*)  
 finally show *?thesis* by *simp*  
 qed

**lemma** *fds-exp-times-fds-nth-0*:  
 $\text{fds-const } (\text{exp } (\text{fds-nth } f \ (\text{Suc } 0))) * \text{fds-exp } (f - \text{fds-const } (\text{fds-nth } f \ (\text{Suc } 0)))$   
 $= \text{fds-exp } f$   
 by (*rule fds-eqI*) (*simp add: fds-exp-def fds-nth-fds' cong: if-cong*)

**lemma** *fds-exp-const* [*simp*]:  $\text{fds-exp } (\text{fds-const } c) = \text{fds-const } (\text{exp } c)$   
**proof** –  
 have  $\text{fds-exp } (\text{fds-const } c) = \text{fds } (\lambda n. \text{exp } c * (\sum k. \text{fds-nth } (\text{fds } (\lambda n. 0) \wedge k) \ n \ /_R \ \text{fact } k))$   
 by (*simp add: fds-exp-def fds-nth-fds' one-fds-def cong: if-cong*)  
 also have  $\text{fds } (\lambda-. \ 0 \ :: 'a) = 0$  by (*simp add: fds-eq-iff*)  
 also have  $(\lambda(k::\text{nat}) \ (n::\text{nat}). \ \text{fds-nth } (0 \wedge k) \ n) = (\lambda k \ n. \ \text{if } k = 0 \wedge n = 1 \ \text{then } 1 \ \text{else } 0)$   
 by (*intro ext*) (*auto simp: one-fds-def fds-nth-fds' power-0-left*)  
 also have  $(\lambda n::\text{nat}. \ \sum k. \ (\text{if } k = 0 \wedge n = 1 \ \text{then } 1 \ \text{else } (0::'a)) \ /_R \ \text{fact } k) =$   
 $(\lambda n. \ \text{if } n = 1 \ \text{then } (\sum k. \ (\text{if } k = 0 \ \text{then } 1 \ \text{else } 0)) \ /_R \ \text{fact } k \ \text{else } 0)$   
 by (*intro ext*) *auto*  
 also have  $\dots = (\lambda n::\text{nat}. \ \text{if } n = 1 \ \text{then } (\sum k \in \{0\}. \ (\text{if } k = (0::\text{nat}) \ \text{then } 1 \ \text{else } 0)) \ \text{else } 0 \ :: 'a)$   
 by (*subst suminf-finite*[of  $\{0\}$ ]) *auto*  
 also have  $\text{fds } (\lambda n. \ \text{exp } c * \dots \ n) = \text{fds-const } (\text{exp } c)$   
 by (*simp add: fds-const-def fds-eq-iff fds-nth-fds' cong: if-cong*)  
 finally show *?thesis* .  
 qed

**lemma** *fds-exp-numeral* [*simp*]:  $\text{fds-exp } (\text{numeral } n) = \text{fds-const } (\text{exp } (\text{numeral } n))$   
 using *fds-exp-const*[of *numeral*  $n \ :: 'a$ ] by (*simp del: fds-exp-const add: numeral-fds*)

**lemma** *fds-exp-0* [*simp*]:  $\text{fds-exp } 0 = 1$   
 using *fds-exp-const*[of  $0$ ] by (*simp del: fds-exp-const*)

**lemma** *fds-exp-1* [*simp*]:  $\text{fds-exp } 1 = \text{fds-const } (\text{exp } 1)$   
 using *fds-exp-const*[of  $1$ ] by (*simp del: fds-exp-const*)

**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** –  
 have  $(\sum k. \ 0 \wedge k \ /_R \ \text{fact } k) = (\sum k \in \{0\}. \ 0 \wedge k \ /_R \ \text{fact } k \ :: 'a)$   
 by (*intro suminf-finite*) (*auto simp: power-0-left*)  
 also have  $\dots = 1$  by *simp*  
 finally show *?thesis* by (*simp add: fds-exp-def*)  
 qed

## 4.8 Subseries

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

**lemma**  $\text{fds-nth-subseries}$ :

$\text{fds-nth } (\text{fds-subseries } P f) n = (\text{if } P n \text{ then } \text{fds-nth } f n \text{ else } 0)$   
**by** ( $\text{simp add: fds-subseries-def fds-nth-fds}$ )

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

**by** ( $\text{simp add: fds-subseries-def fds-eq-iff}$ )

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

**by** ( $\text{simp add: fds-subseries-def fds-eq-iff one-fds-def}$ )

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

**by** ( $\text{simp add: fds-subseries-def fds-eq-iff fds-const-def}$ )

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

**by** ( $\text{simp add: fds-subseries-def fds-eq-iff plus-fds-def}$ )

**lemma**  $\text{fds-subseries-diff}$  [ $\text{simp}$ ]:

$\text{fds-subseries } P (f - g :: 'a :: \text{ring-1 } \text{fds}) = \text{fds-subseries } P f - \text{fds-subseries } P g$   
**by** ( $\text{simp add: fds-subseries-def fds-eq-iff minus-fds-def}$ )

**lemma**  $\text{fds-subseries-minus}$  [ $\text{simp}$ ]:

$\text{fds-subseries } P (-f :: 'a :: \text{ring-1 } \text{fds}) = - \text{fds-subseries } P f$   
**by** ( $\text{simp add: fds-subseries-def fds-eq-iff minus-fds-def}$ )

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

**by** ( $\text{induction } A \text{ rule: infinite-finite-induct}$ )  $\text{simp-all}$

**lemma**  $\text{fds-subseries-shift}$  [ $\text{simp}$ ]:

$\text{fds-subseries } P (\text{fds-shift } c f) = \text{fds-shift } c (\text{fds-subseries } P f)$   
**by** ( $\text{simp add: fds-subseries-def fds-eq-iff}$ )

**lemma**  $\text{fds-subseries-deriv}$  [ $\text{simp}$ ]:

$\text{fds-subseries } P (\text{fds-deriv } f) = \text{fds-deriv } (\text{fds-subseries } P f)$   
**by** ( $\text{simp add: fds-subseries-def fds-deriv-def fds-eq-iff}$ )

**lemma**  $\text{fds-subseries-integral}$  [ $\text{simp}$ ]:

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

**by** ( $\text{auto simp: fds-subseries-def fds-integral-def fds-eq-iff}$ )

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

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

**lemma**  $fds\text{-primepow}\text{-subseries}\text{-mult}$  [simp]:

**fixes**  $p :: nat$

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

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

**proof** (rule  $fds\text{-eqI}$ )

**fix**  $n :: nat$

**consider**  $n = 0 \mid P n \ n > 0 \mid \neg P n \ n > 0$  **by** *blast*

**thus**  $fds\text{-nth } (fds\text{-subseries } P (f * g)) n = fds\text{-nth } (fds\text{-subseries } P f * fds\text{-subseries } P g) n$

**proof** *cases*

**case** 2

**have**  $P d$  **if**  $d \text{ dvd } n$  **for**  $d$

**proof** –

**have**  $\text{prime-factors } d \subseteq \text{prime-factors } n$  **using** *that* 2

**by** (*intro dvd-prime-factors*) *auto*

**also have**  $\dots \subseteq \{p\}$  **using** 2 **by** (*simp add: P-def*)

**finally show** *?thesis* **by** (*simp add: P-def*)

**qed**

**have**  $P' a P b$  **if**  $n = a * b$  **for**  $a b$

**using**  $P[\text{of } a] P[\text{of } b]$  **that** **by** *auto*

**have**  $fds\text{-nth } (fds\text{-subseries } P (f * g)) n = \text{dirichlet-prod } (fds\text{-nth } f) (fds\text{-nth } g)$

$n$

**using** 2 **by** (*simp add: fds-subseries-def fds-nth-fds' fds-nth-mult*)

**also have**  $\dots = \text{dirichlet-prod } (fds\text{-nth } (fds\text{-subseries } P f)) (fds\text{-nth } (fds\text{-subseries } P g)) n$

**unfolding** *dirichlet-prod-altdef2* **using** 2

**by** (*intro sum.cong refl*) (*auto simp: fds-subseries-def fds-nth-fds' dest: P'*)

**finally show** *?thesis* **by** (*simp add: fds-nth-mult*)

**next**

**case** 3

**have**  $\neg(P a \wedge P b)$  **if**  $n = a * b$  **for**  $a b$

**proof** –

**have**  $\text{prime-factors } n = \text{prime-factors } (a * b)$  **by** (*simp add: that*)

**also have**  $\dots = \text{prime-factors } a \cup \text{prime-factors } b$

**using** 3 **that** **by** (*intro prime-factors-product*) *auto*

**finally show** *?thesis* **using** 3 **by** (*auto simp: P-def*)

**qed**

**hence**  $\text{dirichlet-prod } (fds\text{-nth } (fds\text{-subseries } P f)) (fds\text{-nth } (fds\text{-subseries } P g))$

$n = 0$

**unfolding** *dirichlet-prod-altdef2*

**by** (*intro sum.neutral*) (*auto simp: fds-subseries-def fds-nth-fds'*)

**also have**  $\dots = fds\text{-nth } (fds\text{-subseries } P (f * g)) n$

**using** 3 **by** (*simp add: fds-subseries-def*)

**finally show** *?thesis* **by** (*simp add: fds-nth-mult*)

**qed** *auto*

**qed**

**lemma** *fds-primelow-subseries-power* [*simp*]:  
*fds-primelow-subseries*  $p$  ( $f \wedge n$ ) = *fds-primelow-subseries*  $p$   $f \wedge n$   
**by** (*induction*  $n$ ) *simp-all*

**lemma** *fds-primelow-subseries-prod* [*simp*]:  
*fds-primelow-subseries*  $p$  ( $\prod_{x \in A} f x$ ) = ( $\prod_{x \in A} \text{fds-primelow-subseries } p (f x)$ )  
**by** (*induction*  $A$  *rule: infinite-finite-induct*) *simp-all*

**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** –  
**interpret** *completely-multiplicative-function* *fds-nth*  $f$  **by** *fact*  
**show** *?thesis*  
**by** *standard* (*auto simp: fds-nth-subseries prime-factors-product mult*)  
**qed**

## 4.9 Truncation

**definition** *fds-truncate* ::  $\text{nat} \Rightarrow 'a :: \{\text{zero}\} \text{fds} \Rightarrow 'a \text{fds}$  **where**  
*fds-truncate*  $m$   $f$  = *fds* ( $\lambda n. \text{if } n \leq m \text{ then } \text{fds-nth } f n \text{ else } 0$ )

**lemma** *fds-nth-truncate*: *fds-nth* (*fds-truncate*  $m$   $f$ )  $n$  = (*if*  $n \leq m$  *then* *fds-nth*  $f$   $n$  *else*  $0$ )  
**by** (*simp add: fds-truncate-def fds-nth-fds'*)

**lemma** *fds-truncate-0* [*simp*]: *fds-truncate*  $0$   $f$  =  $0$   
**by** (*simp add: fds-eq-iff fds-nth-truncate*)

**lemma** *fds-truncate-zero* [*simp*]: *fds-truncate*  $m$   $0$  =  $0$   
**by** (*simp add: fds-truncate-def fds-eq-iff*)

**lemma** *fds-truncate-one* [*simp*]:  $m > 0 \implies \text{fds-truncate } m 1 = 1$   
**by** (*simp add: fds-truncate-def fds-eq-iff*)

**lemma** *fds-truncate-const* [*simp*]:  $m > 0 \implies \text{fds-truncate } m (\text{fds-const } c) = \text{fds-const } c$   
**by** (*simp add: fds-truncate-def fds-eq-iff*)

**lemma** *fds-truncate-truncate* [*simp*]: *fds-truncate*  $m$  (*fds-truncate*  $n$   $f$ ) = *fds-truncate* (*min*  $m$   $n$ )  $f$   
**by** (*rule fds-eqI*) (*simp add: fds-nth-truncate*)

**lemma** *fds-truncate-truncate'* [*simp*]: *fds-truncate*  $m$  (*fds-truncate*  $m$   $f$ ) = *fds-truncate*  $m$   $f$   
**by** (*rule fds-eqI*) (*simp add: fds-nth-truncate*)

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

**by** (*simp add: fds-eq-iff fds-nth-truncate*)

**lemma** *fds-truncate-add-strong*:

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

**by** (*auto simp: fds-eq-iff fds-nth-truncate*)

**lemma** *fds-truncate-add*:

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

**by** (*auto simp: fds-eq-iff fds-nth-truncate*)

**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** (*intro fds-eqI, goal-cases*)

**case** (1 *n*)

**show** *?case*

**proof** (*cases n ≤ m*)

**case** *True*

**hence**  $\text{fds-nth } ?B n = \text{dirichlet-prod } (\text{fds-nth } f) (\text{fds-nth } g) n$

**by** (*simp add: fds-nth-truncate fds-nth-mult*)

**also have**  $\dots = \text{dirichlet-prod } (\text{fds-nth } (\text{fds-truncate } m f)) (\text{fds-nth } (\text{fds-truncate } m g)) n$

**unfolding** *dirichlet-prod-def*

**proof** (*intro sum.cong refl, goal-cases*)

**case** (1 *d*)

**with**  $\langle n > 0 \rangle$  **have**  $d \leq m \ n \ \text{div } d \leq m$

**by** (*auto dest: dvd-imp-le intro: order.trans[OF - True]*)

**thus** *?case* **by** (*auto simp add: fds-nth-truncate*)

**qed**

**also have**  $\dots = \text{fds-nth } ?A n$  **using** *True* **by** (*simp add: fds-nth-truncate fds-nth-mult*)

**finally show** *?thesis ..*

**qed** (*auto simp: fds-nth-truncate*)

**qed**

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

**by** (*simp add: fds-eq-iff fds-nth-truncate fds-deriv-def*)

**lemma** *fds-truncate-integral*:

$m > 0 \vee c = 0 \implies \text{fds-truncate } m (\text{fds-integral } c f) = \text{fds-integral } c (\text{fds-truncate } m f)$

**by** (*auto simp: fds-eq-iff fds-nth-truncate fds-integral-def*)

**lemma** *fds-truncate-power*:  $\text{fds-truncate } m (\text{fds-truncate } m f ^ n) = \text{fds-truncate } m f ^ n$

$m (f \wedge n)$   
**proof** (cases  $m = 0$ )  
   **case** *False*  
   **show** *?thesis*  
   **proof** (induction  $n$ )  
     **case** (*Suc*  $n$ )  
     **have**  $\text{fds-truncate } m (\text{fds-truncate } m f \wedge \text{Suc } n) =$   
        $\text{fds-truncate } m (\text{fds-truncate } m f * \text{fds-truncate } m f \wedge n)$  **by** *simp*  
     **also have**  $\dots = \text{fds-truncate } m (\text{fds-truncate } m f * \text{fds-truncate } m (f \wedge n))$   
       **by** (*subst fds-truncate-mult [symmetric]*) (*simp add: Suc*)  
     **also have**  $\dots = \text{fds-truncate } m (f \wedge \text{Suc } n)$   
       **by** (*simp add: fds-truncate-mult*)  
     **finally show** *?case* .  
   **qed** (*simp-all add: fds-truncate-mult*)  
**qed** *simp-all*

**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** –  
**have**  $\text{dirichlet-inverse } f \ i \ n = \text{dirichlet-inverse } f' \ i \ n$   
**using** *assms(1)*  
**proof** (induction  $n$  *rule: dirichlet-inverse-induct*)  
**case** (*gt1*  $n$ )  
**have** \*:  $\text{dirichlet-inverse } f \ i \ k = \text{dirichlet-inverse } f' \ i \ k$  **if**  $k \text{ dvd } n \wedge k < n$  **for**  $k$   
   **using** *that* **by** (*intro gt1*) *auto*  
**have** \*:  $(\sum d \mid d \text{ dvd } n \wedge d < n. f (n \text{ div } d) * \text{dirichlet-inverse } f \ i \ d) =$   
    $(\sum d \mid d \text{ dvd } n \wedge d < n. f' (n \text{ div } d) * \text{dirichlet-inverse } f' \ i \ d)$   
   **by** (*intro sum.cong refl*) (*subst gt1.premis, auto elim: dvdE simp: \**)  
**consider**  $n = 0 \mid n = 1 \mid n > 1$  **by force**  
**thus** *?case*  
   **by cases** (*insert \*, simp-all add: dirichlet-inverse-gt-1 \* cong: sum.cong*)  
**qed** *auto*  
**with** *assms(2,3)* **show** *?thesis* **by** *simp*  
**qed**

**lemma** *fds-truncate-cong*:  
 $(\bigwedge n. m > 0 \implies n > 0 \implies n \leq m \implies \text{fds-nth } f \ n = \text{fds-nth } f' \ n) \implies$   
 $\text{fds-truncate } m \ f = \text{fds-truncate } m \ f'$   
**by** (*rule fds-eqI*) (*simp add: fds-nth-truncate*)

**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** (*rule fds-truncate-cong, goal-cases*)  
**case** ( $1 \ n$ )  
**have** \*:  $\text{dirichlet-inverse } (\lambda n. \text{if } n \leq m \text{ then } \text{fds-nth } f \ n \text{ else } 0) (\text{inverse } (\text{fds-nth}$   
 $f \ 1)) \ n =$   
 $\text{dirichlet-inverse } (\text{fds-nth } f) (\text{inverse } (\text{fds-nth } f \ 1)) \ n$  **using**  $1$

```

  by (intro dirichlet-inverse-cong-simp) auto
show ?case
proof (cases fds-nth f 1 = 0)
  case True
  thus ?thesis by (auto simp: inverse-fds-nonunit fds-nth-truncate)
qed (insert * 1, auto simp: inverse-fds-def fds-nth-fds' fds-nth-truncate Suc-le-eq)
qed

```

```

lemma fds-truncate-divide:
  fixes f g :: 'a :: field fds
  shows fds-truncate m (fds-truncate m f / fds-truncate m g) = fds-truncate m (f
/ g)
proof -
  have fds-truncate m (f / g) = fds-truncate m (fds-truncate m (fds-truncate m f)
*
  fds-truncate m (inverse (fds-truncate m g)))
  by (simp add: fds-truncate-inverse fds-truncate-mult divide-fds-def)
  also have ... = fds-truncate m (fds-truncate m f * inverse (fds-truncate m g))
  by (rule fds-truncate-mult)
  also have ... = fds-truncate m (fds-truncate m f / fds-truncate m g)
  by (simp add: divide-fds-def)
  finally show ?thesis ..
qed

```

```

lemma fds-truncate-ln:
  fixes f :: 'a :: real-normed-field fds
  shows fds-truncate m (fds-ln l (fds-truncate m f)) = fds-truncate m (fds-ln l f)
  by (cases m = 0)
  (simp-all add: fds-ln-def fds-truncate-integral fds-truncate-deriv [symmetric]
  fds-truncate-divide)

```

```

lemma fds-truncate-exp:
  shows fds-truncate m (fds-exp (fds-truncate m f)) = fds-truncate m (fds-exp f)
proof (rule fds-truncate-cong, goal-cases)
  case (1 n)
  define a where a = exp (fds-nth f (Suc 0))
  define f' where f' = fds (λn. if n = Suc 0 then 0 else fds-nth f n)
  have truncate-f': fds-truncate m f' = fds (λn. if n = Suc 0 then 0 else fds-nth
(fds-truncate m f) n)
  by (simp add: f'-def fds-eq-iff fds-nth-truncate)

  have fds-nth (fds-exp (fds-truncate m f)) n =
    a * (∑ k. fds-nth (fds-truncate m f' ^ k) n /R fact k) using 1
  by (simp add: fds-exp-def fds-nth-fds' a-def [symmetric] f'-def [symmetric]
  fds-nth-truncate truncate-f' [symmetric])
  also have (λk. fds-nth (fds-truncate m f' ^ k) n) = (λk. fds-nth (f' ^ k) n)
proof (rule ext, goal-cases)
  case (1 k)
  have fds-nth (fds-truncate m f' ^ k) n = fds-nth (fds-truncate m (fds-truncate

```

$m f' \wedge k) n$   
**using**  $\langle n \leq m \rangle$  **by** (*simp add: fds-nth-truncate*)  
**also have**  $\text{fds-truncate } m (\text{fds-truncate } m f' \wedge k) = \text{fds-truncate } m (f' \wedge k)$   
**by** (*simp add: fds-truncate-power*)  
**also have**  $\text{fds-nth } \dots n = \text{fds-nth } (f' \wedge k) n$  **using**  $\langle n \leq m \rangle$  **by** (*simp add: fds-nth-truncate*)  
**finally show** *?case* .  
**qed**  
**also have**  $a * (\sum k. \dots k /_{\mathbb{R}} \text{fact } k) = \text{fds-nth } (\text{fds-exp } f) n$   
**by** (*simp add: fds-exp-def fds-nth-fds' a-def f'-def*)  
**finally show** *?case* .  
**qed**

**lemma** *fds-eqI-truncate*:  
**assumes**  $\bigwedge m. m > 0 \implies \text{fds-truncate } m f = \text{fds-truncate } m g$   
**shows**  $f = g$   
**proof** (*rule fds-eqI*)  
**fix**  $n :: \text{nat}$  **assume**  $n > 0$   
**have**  $\text{fds-nth } f n = \text{fds-nth } (\text{fds-truncate } n f) n$   
**by** (*simp add: fds-nth-truncate*)  
**also note** *assms*[*OF*  $\langle n > 0 \rangle$ ]  
**also have**  $\text{fds-nth } (\text{fds-truncate } n g) n = \text{fds-nth } g n$   
**by** (*simp add: fds-nth-truncate*)  
**finally show**  $\text{fds-nth } f n = \text{fds-nth } g n$  .  
**qed**

## 4.10 Normed series

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

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

**lemma** *fds-norm-1* [*simp*]:  $\text{fds-norm } 1 = 1$   
**by** (*simp add: fds-eq-iff one-fds-def*)

**lemma** *fds-nth-norm-mult-le*:  
**shows**  $\text{norm } (\text{fds-nth } (f * g) n) \leq \text{fds-nth } (\text{fds-norm } f * \text{fds-norm } g) n$   
**by** (*auto simp add: fds-nth-mult dirichlet-prod-def norm-mult intro!: sum-norm-le*)

**lemma** *fds-nth-norm-mult-nonneg* [*simp*]:  $\text{fds-nth } (\text{fds-norm } f * \text{fds-norm } g) n \geq 0$   
**by** (*auto simp: fds-nth-mult dirichlet-prod-def intro!: sum-nonneg*)

## 4.11 Lifting a real series to a real algebra

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

**lemma** *fds-nth-of-real* [*simp*]:  $\text{fds-nth } (\text{fds-of-real } f) \ n = \text{of-real } (\text{fds-nth } f \ n)$   
**by** (*simp add: fds-of-real-def fds-nth-fds'*)

**lemma** *fds-of-real-0* [*simp*]:  $\text{fds-of-real } 0 = 0$   
**and** *fds-of-real-1* [*simp*]:  $\text{fds-of-real } 1 = 1$   
**and** *fds-of-real-const* [*simp*]:  $\text{fds-of-real } (\text{fds-const } c) = \text{fds-const } (\text{of-real } c)$   
**and** *fds-of-real-minus* [*simp*]:  $\text{fds-of-real } (-f) = -\text{fds-of-real } f$   
**and** *fds-of-real-add* [*simp*]:  $\text{fds-of-real } (f + g) = \text{fds-of-real } f + \text{fds-of-real } g$   
**and** *fds-of-real-mult* [*simp*]:  $\text{fds-of-real } (f * g) = \text{fds-of-real } f * \text{fds-of-real } g$   
**and** *fds-of-real-deriv* [*simp*]:  $\text{fds-of-real } (\text{fds-deriv } f) = \text{fds-deriv } (\text{fds-of-real } f)$   
**by** (*simp-all add: fds-eq-iff one-fds-def fds-const-def fds-nth-mult dirichlet-prod-def fds-deriv-def scaleR-conv-of-real*)

**lemma** *fds-of-real-higher-deriv* [*simp*]:  
 $(\text{fds-deriv } \widetilde{\sim} \ n) (\text{fds-of-real } f) = \text{fds-of-real } ((\text{fds-deriv } \widetilde{\sim} \ n) f)$   
**by** (*induction n simp-all*)

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

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

**lemma** *eval-fds-eqI*:  
**assumes**  $(\lambda n. \text{fds-nth } f \ (\text{Suc } n) / \text{nat-power } (\text{Suc } n) \ s) \ \text{sums } L$   
**shows**  $\text{eval-fds } f \ s = L$

**proof** –

**from** *assms* **have**  $(\lambda n. \text{fds-nth } f \ n / \text{nat-power } n \ s) \ \text{sums } L$   
**by** (*subst (asm) sums-Suc-iff*) *auto*  
**thus** *?thesis* **by** (*simp add: eval-fds-def sums-iff*)

**qed**

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

**lemma** *fds-converges-iff*:  
 $\text{fds-converges } f \ s \longleftrightarrow (\lambda n. \text{fds-nth } f \ n / \text{nat-power } n \ s) \ \text{sums } \text{eval-fds } f \ s$   
**by** (*simp add: fds-converges-def sums-iff eval-fds-def*)

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

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

$fds-abs-converges\ f\ s \implies fds-converges\ f\ s$   
**unfolding**  $fds-abs-converges-def\ fds-converges-def$  **by** (rule summable-norm-cancel)

**lemma**  $fds-converges-altdef$ :

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

**unfolding**  $fds-converges-def\ summable-sums-iff$   
**by** (subst sums-Suc-iff) (simp-all add: eval-fds-def)

**lemma**  $fds-const-abs-converges$  [simp]:  $fds-abs-converges\ (fds-const\ c)\ s$

**proof** –

**have**  $summable\ (\lambda n. norm\ (fds-nth\ (fds-const\ c)\ n / nat-power\ n\ s)) \iff$   
 $summable\ (\lambda n. if\ n = 1\ then\ norm\ c\ else\ (0 :: real))$

**by** (intro summable-cong) simp

**also have** ... **by** simp

**finally show** ?thesis **by** (simp add: fds-abs-converges-def)

qed

**lemma**  $fds-const-converges$  [simp]:  $fds-converges\ (fds-const\ c)\ s$

**by** (rule fds-abs-converges-imp-converges) simp

**lemma**  $eval-fds-const$  [simp]:  $eval-fds\ (fds-const\ c) = (\lambda-. c)$

**proof**

**fix**  $s$

**have**  $eval-fds\ (fds-const\ c)\ s = (\sum\ n. if\ n = 1\ then\ c\ else\ 0)$  **unfolding**  $eval-fds-def$   
**by** (intro suminf-cong) simp

**also have** ... =  $c$  **using**  $sums-single$ [of 1  $\lambda-. c$ ] **by** (simp add: sums-iff)

**finally show**  $eval-fds\ (fds-const\ c)\ s = c$ .

qed

**lemma**  $fds-zero-abs-converges$  [simp]:  $fds-abs-converges\ 0\ s$

**by** (simp add: fds-abs-converges-def)

**lemma**  $fds-zero-converges$  [simp]:  $fds-converges\ 0\ s$

**by** (simp add: fds-converges-def)

**lemma**  $eval-fds-zero$  [simp]:  $eval-fds\ 0 = (\lambda-. 0)$

**by** (simp only: fds-const-zero [symmetric] eval-fds-const)

**lemma**  $fds-one-abs-converges$  [simp]:  $fds-abs-converges\ 1\ s$

**by** (simp only: fds-const-one [symmetric] fds-const-abs-converges)

**lemma**  $fds-one-converges$  [simp]:  $fds-converges\ 1\ s$

**by** (simp only: fds-const-one [symmetric] fds-const-converges)

**lemma**  $fds-converges-truncate$  [simp]:  $fds-converges\ (fds-truncate\ n\ f)\ s$

**proof** –

**have**  $summable\ (\lambda k. fds-nth\ (fds-truncate\ n\ f)\ k / nat-power\ k\ s) \iff summable$   
 $(\lambda-. 0 :: 'a)$

**by** (*intro summable-cong*[*OF eventually-mono*[*OF eventually-gt-at-top*[*of n*]]])  
 (*auto simp: fds-nth-truncate*)  
**thus** *?thesis* **by** (*simp add: fds-converges-def*)  
**qed**

**lemma** *fds-abs-converges-truncate* [*simp*]: *fds-abs-converges* (*fds-truncate n f*) *s*  
**proof** –  
**have** *summable* ( $\lambda k. \text{norm } (\text{fds-nth } (\text{fds-truncate } n \ f) \ k / \text{nat-power } k \ s)$ )  $\longleftrightarrow$   
*summable* ( $\lambda-. \ 0 :: \text{real}$ )  
**by** (*intro summable-cong*[*OF eventually-mono*[*OF eventually-gt-at-top*[*of n*]]])  
 (*auto simp: fds-nth-truncate*)  
**thus** *?thesis* **by** (*simp add: fds-abs-converges-def*)  
**qed**

**lemma** *fds-abs-converges-subseries* [*simp, intro*]:  
**assumes** *fds-abs-converges f s*  
**shows** *fds-abs-converges* (*fds-subseries P f*) *s*  
**unfolding** *fds-abs-converges-def*  
**proof** (*rule summable-comparison-test-ev*)  
**show** *summable* ( $\lambda n. \text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ s)$ )  
**using** *assms* **unfolding** *fds-abs-converges-def* .  
**qed** (*auto simp: fds-nth-subseries*)

**lemma** *eval-fds-one* [*simp*]: *eval-fds 1* = ( $\lambda-. \ 1$ )  
**by** (*simp only: fds-const-one [symmetric] eval-fds-const*)

**lemma** *eval-fds-truncate*: *eval-fds* (*fds-truncate n f*) *s* = ( $\sum_{k=1..n. \text{fds-nth } f \ k / \text{nat-power } k \ s}$ )  
**proof** –  
**have** *eval-fds* (*fds-truncate n f*) *s* = ( $\sum_{k=1..n. \text{fds-nth } (\text{fds-truncate } n \ f) \ k / \text{nat-power } k \ s}$ )  
**unfolding** *eval-fds-def* **by** (*intro suminf-finite*) (*auto simp: fds-nth-truncate Suc-le-eq*)  
**also have**  $\dots = (\sum_{k=1..n. \text{fds-nth } f \ k / \text{nat-power } k \ s}$ )  
**by** (*intro sum.cong*) (*auto simp: fds-nth-truncate*)  
**finally show** *?thesis* .  
**qed**

**lemma** *fds-converges-add*:  
**assumes** *fds-converges f s* *fds-converges g s*  
**shows** *fds-converges* (*f + g*) *s*  
**using** *summable-add*[*OF assms*[*unfolded fds-converges-def*]]  
**by** (*simp add: fds-converges-def add-divide-distrib*)

**lemma** *fds-abs-converges-add*:  
**assumes** *fds-abs-converges f s* *fds-abs-converges g s*  
**shows** *fds-abs-converges* (*f + g*) *s*  
**unfolding** *fds-abs-converges-def*

**proof** (rule summable-comparison-test, intro exI allI impI)  
**let** ?A = ( $\lambda n$ . norm (fds-nth f n / nat-power n s) + norm (fds-nth g n / nat-power n s))  
**from** summable-add[OF assms[unfolded fds-abs-converges-def]] **show** summable ?A .  
**fix** n :: nat  
**show** norm (norm (fds-nth (f + g) n / nat-power n s))  $\leq$  ?A n  
**by** (simp add: norm-triangle-ineq add-divide-distrib)  
**qed**

**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** –  
**from** assms **have** ( $\lambda n$ . fds-nth f n / nat-power n s) sums eval-fds f s  
( $\lambda n$ . fds-nth g n / nat-power n s) sums eval-fds g s  
**by** (simp-all add: fds-converges-def sums-iff eval-fds-def)  
**from** sums-add[OF this] **show** ?thesis **by** (simp add: eval-fds-def sums-iff add-divide-distrib)  
**qed**

**lemma** fds-converges-uminus:  
**assumes** fds-converges f s  
**shows** fds-converges (–f) s  
**using** summable-minus[OF assms[unfolded fds-converges-def]]  
**by** (simp add: fds-converges-def add-divide-distrib)

**lemma** The-cong: The P = The Q if  $\bigwedge x. P x \longleftrightarrow Q x$   
**proof** –  
**from** that **have** P = Q **by** auto  
**thus** ?thesis **by** simp  
**qed**

**lemma** fds-abs-converges-uminus:  
**assumes** fds-abs-converges f s  
**shows** fds-abs-converges (–f) s  
**using** assms **by** (simp add: fds-abs-converges-def)

**lemma** eval-fds-uminus:  $\text{fds-converges } f \text{ s} \implies \text{eval-fds } (–f) \text{ s} = –\text{eval-fds } f \text{ s}$   
**by** (simp add: fds-converges-def eval-fds-def suminf-minus)

**lemma** fds-converges-diff:  
**assumes** fds-converges f s fds-converges g s  
**shows** fds-converges (f – g) s  
**using** summable-diff[OF assms[unfolded fds-converges-def]]  
**by** (simp add: fds-converges-def diff-divide-distrib)

**lemma** fds-abs-converges-diff:

**assumes**  $fds-abs-converges\ f\ s\ fds-abs-converges\ g\ s$   
**shows**  $fds-abs-converges\ (f - g)\ s$   
**unfolding**  $fds-abs-converges-def$   
**proof** (*rule summable-comparison-test, intro exI allI impI*)  
**let**  $?A = (\lambda n. norm\ (fds-nth\ f\ n\ / \ nat-power\ n\ s) + norm\ (fds-nth\ g\ n\ / \ nat-power\ n\ s))$   
**from**  $summable-add[OF\ assms[unfolded\ fds-abs-converges-def]]$  **show**  $summable\ ?A$  .  
**fix**  $n :: nat$   
**show**  $norm\ (norm\ (fds-nth\ (f - g)\ n\ / \ nat-power\ n\ s)) \leq ?A\ n$   
**by** (*simp add: norm-triangle-ineq4 diff-divide-distrib*)  
**qed**

**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** -  
**from**  $assms$  **have**  $(\lambda n. fds-nth\ f\ n\ / \ nat-power\ n\ s)\ sums\ eval-fds\ f\ s$   
 $(\lambda n. fds-nth\ g\ n\ / \ nat-power\ n\ s)\ sums\ eval-fds\ g\ s$   
**by** (*simp-all add: fds-converges-def sums-iff eval-fds-def*)  
**from**  $sums-diff[OF\ this]$  **show**  $?thesis$  **by** (*simp add: eval-fds-def sums-iff diff-divide-distrib*)  
**qed**

**lemma**  $eval-fds-at-nat$ :  $eval-fds\ f\ (of-nat\ k) = (\sum n. fds-nth\ f\ n\ / \ of-nat\ n\ ^\ k)$   
**unfolding**  $eval-fds-def$   
**proof** (*intro suminf-cong, goal-cases*)  
**case**  $(1\ n)$   
**thus**  $?case$  **by** (*cases\ n = 0*) *simp-all*  
**qed**

**lemma**  $eval-fds-at-numeral$ :  $eval-fds\ f\ (numeral\ k) = (\sum n. fds-nth\ f\ n\ / \ of-nat\ n\ ^\ numeral\ k)$   
**using**  $eval-fds-at-nat[of\ f\ numeral\ k]$  **by** *simp*

**lemma**  $eval-fds-at-1$ :  $eval-fds\ f\ 1 = (\sum n. fds-nth\ f\ n\ / \ of-nat\ n)$   
**using**  $eval-fds-at-nat[of\ f\ 1]$  **by** *simp*

**lemma**  $eval-fds-at-0$ :  $eval-fds\ f\ 0 = (\sum n. fds-nth\ f\ n)$   
**using**  $eval-fds-at-nat[of\ f\ 0]$  **by** *simp*

**lemma**  $suminf-fds-zeta-aux$ :  
 $f\ 0 = 0 \implies (\sum n. fds-nth\ fds-zeta\ n\ / \ f\ n) = (\sum n. 1\ / \ f\ n :: 'a :: real-normed-field)$   
**by** (*intro suminf-cong*) (*auto simp: fds-nth-zeta*)

**lemma**  $fds-converges-shift$  [*simp*]:  
**fixes**  $z :: 'a :: \{banach, nat-power-field, real-normed-field\}$   
**shows**  $fds-converges\ (fds-shift\ c\ f)\ z \longleftrightarrow fds-converges\ f\ (z - c)$

**unfolding** *fds-converges-def*  
**by** (*intro summable-cong*)  
(*auto intro: eventually-mono [OF eventually-gt-at-top[of 0::nat]] simp: nat-power-diff*)

**lemma** *fds-abs-converges-shift* [*simp*]:  
**fixes**  $z :: 'a :: \{\text{banach, nat-power-field, real-normed-field}\}$   
**shows** *fds-abs-converges* (*fds-shift*  $c$   $f$ )  $z \longleftrightarrow \text{fds-abs-converges } f (z - c)$   
**unfolding** *fds-abs-converges-def*  
**by** (*intro summable-cong*)  
(*auto intro: eventually-mono [OF eventually-gt-at-top[of 0::nat]] simp: nat-power-diff*)

**lemma** *fds-eval-shift* [*simp*]:  
**fixes**  $z :: 'a :: \{\text{banach, nat-power-field, real-normed-field}\}$   
**shows** *eval-fds* (*fds-shift*  $c$   $f$ )  $z = \text{eval-fds } f (z - c)$   
**unfolding** *eval-fds-def*  
**proof** (*rule suminf-cong, goal-cases*)  
**case** ( $1\ n$ )  
**show** *?case* **by** (*cases n = 0*) (*simp-all add: nat-power-diff*)  
**qed**

**lemma** *fds-converges-scale* [*simp*]:  
**fixes**  $z :: 'a :: \{\text{banach, nat-power-field, real-normed-field}\}$   
**assumes**  $c: c > 0$   
**shows** *fds-converges* (*fds-scale*  $c$   $f$ )  $z \longleftrightarrow \text{fds-converges } f (\text{of-nat } c * z)$   
**proof** –  
**have** *fds-converges* (*fds-scale*  $c$   $f$ )  $z \longleftrightarrow$   
 $\text{summable } (\lambda n. \text{fds-nth } (\text{fds-scale } c\ f) (n \wedge c) / \text{nat-power } (n \wedge c) z)$   
(*is - = summable ?g*) **unfolding** *fds-converges-def*  
**by** (*rule summable-mono-reindex [symmetric]*)  
(*insert c, auto simp: fds-nth-scale is-nth-power-def strict-mono-def power-strict-mono*)  
**also have**  $?g = (\lambda n. \text{fds-nth } f\ n / \text{nat-power } n (\text{of-nat } c * z))$   
**proof** (*intro ext, goal-cases*)  
**case** ( $1\ n$ )  
**thus** *?case* **using**  $c$   
**by** (*cases n = 0*) (*simp-all add: nat-power-power-left nat-power-power [symmetric]*)  
*mult-ac*)  
**qed**  
**finally show** *?thesis* **by** (*simp add: fds-converges-def*)  
**qed**

**lemma** *fds-abs-converges-scale* [*simp*]:  
**fixes**  $z :: 'a :: \{\text{banach, nat-power-field, real-normed-field}\}$   
**assumes**  $c: c > 0$   
**shows** *fds-abs-converges* (*fds-scale*  $c$   $f$ )  $z \longleftrightarrow \text{fds-abs-converges } f (\text{of-nat } c * z)$   
**proof** –  
**have** *fds-abs-converges* (*fds-scale*  $c$   $f$ )  $z \longleftrightarrow$   
 $\text{summable } (\lambda n. \text{norm } (\text{fds-nth } (\text{fds-scale } c\ f) (n \wedge c) / \text{nat-power } (n \wedge c) z))$

```

(is - = summable ?g) unfolding fds-abs-converges-def
by (rule summable-mono-reindex [symmetric])
  (insert c, auto simp: fds-nth-scale is-nth-power-def strict-mono-def power-strict-mono)
also have ?g = (λn. norm (fds-nth f n / nat-power n (of-nat c * z)))
proof (intro ext, goal-cases)
  case (1 n)
  thus ?case using c
  by (cases n = 0) (simp-all add: nat-power-power-left nat-power-power [symmetric]
mult-ac)
qed
finally show ?thesis by (simp add: fds-abs-converges-def)
qed

```

```

lemma eval-fds-scale [simp]:
  fixes z :: 'a :: {banach, nat-power-field, real-normed-field}
  assumes c: c > 0
  shows eval-fds (fds-scale c f) z = eval-fds f (of-nat c * z)
proof -
  have eval-fds (fds-scale c f) z =
    (∑ n. fds-nth (fds-scale c f) (n ^ c) / nat-power (n ^ c) z)
  unfolding eval-fds-def
  by (rule suminf-mono-reindex [symmetric])
    (insert c, auto simp: fds-nth-scale is-nth-power-def strict-mono-def power-strict-mono)
  also have ... = (∑ n. fds-nth f n / nat-power n (of-nat c * z))
  proof (intro suminf-cong, goal-cases)
    case (1 n)
    thus ?case using c
    by (cases n = 0) (simp-all add: nat-power-power-left nat-power-power [symmetric]
mult-ac)
  qed
  finally show ?thesis by (simp add: eval-fds-def)
qed

```

```

lemma fds-abs-converges-integral:
  assumes fds-abs-converges f s
  shows fds-abs-converges (fds-integral c f) s
  unfolding fds-abs-converges-def
proof (rule summable-comparison-test-ev)
  show summable (λn. norm (fds-nth f n / nat-power n s))
    using assms by (simp add: fds-abs-converges-def)
  show eventually (λn. norm (norm (fds-nth (fds-integral c f) n / nat-power n s))
    ≤ norm (fds-nth f n / nat-power n s)) at-top
    using eventually-gt-at-top[of 3]
proof eventually-elim
  case (elim n)
  hence ln n ≥ ln (exp 1)
    using exp-le by (subst ln-le-cancel-iff) auto
  hence norm (fds-nth f n) * 1 ≤ norm (fds-nth f n) * ln (real n)
    by (intro mult-left-mono) auto

```

**with** *elim show ?case*  
**by** (*simp-all add: fds-integral-def norm-divide divide-simps*)  
**qed**  
**qed**

**lemma** *fds-abs-converges-ln*:  
**assumes** *fds-abs-converges (fds-deriv f / f) s*  
**shows** *fds-abs-converges (fds-ln l f) s*  
**using** *assms unfolding fds-ln-def* **by** (*intro fds-abs-converges-integral*)

**end**

## 5 The Möbius $\mu$ function

**theory** *Moebius-Mu*

**imports**

*Main*

*HOL-Number-Theory.Number-Theory*

*HOL-Computational-Algebra.Squarefree*

*Dirichlet-Series*

*Dirichlet-Misc*

**begin**

**definition** *moebius-mu* :: *nat*  $\Rightarrow$  *'a* :: *comm-ring-1* **where**

*moebius-mu n* =

(*if squarefree n then*  $(-1)^{\text{card (prime-factors } n)}$  *else*  $0$ )

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

**by** (*auto simp add: moebius-mu-def*)

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

**by** (*simp add: moebius-mu-def*)

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

**by** (*cases even n*) *simp-all*

**lemma** *moebius-mu-0 [simp]*: *moebius-mu 0* =  $0$

**by** (*simp add: moebius-mu-def*)

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

**by** (*simp add: fun-eq-iff fds-nth-fds*)

**lemma** *prime-factors-Suc-0 [simp]*: *prime-factors (Suc 0)* =  $\{\}$

**by** *simp*

**lemma** *moebius-mu-Suc-0 [simp]*: *moebius-mu (Suc 0)* =  $1$

**by** (*simp add: moebius-mu-def*)

**lemma** *moebius-mu-1 [simp]*: *moebius-mu 1* =  $1$

by (simp add: moebius-mu-def)

**lemma** moebius-mu-eq-zero-iff: moebius-mu n = 0  $\longleftrightarrow$   $\neg$ squarefree n  
 by (simp add: moebius-mu-def)

**lemma** moebius-mu-not-squarefree [simp]:  $\neg$ squarefree n  $\implies$  moebius-mu n = 0  
 by (simp add: moebius-mu-def)

**lemma** moebius-mu-power:  
 assumes a > 1 n > 1  
 shows moebius-mu (a ^ n) = 0  
**proof** –  
 from assms have a ^ 2 dvd a ^ n by (simp add: le-imp-power-dvd)  
 with moebius-mu-eq-zero-iff[of a ^ n] and ⟨a > 1⟩ show ?thesis by (auto simp: squarefree-def)  
**qed**

**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)  
 by (simp add: squarefree-power-iff)

**lemma** moebius-mu-squarefree-eq:  
 squarefree n  $\implies$  moebius-mu n = (-1) ^ card (prime-factors n)  
 by (simp add: moebius-mu-def split: if-splits)

**lemma** moebius-mu-squarefree-eq':  
 assumes squarefree n  
 shows moebius-mu n = (-1) ^ size (prime-factorization n)  
**proof** –  
 let ?P = prime-factorization n  
 from assms have [simp]: n > 0 by (auto intro!: Nat.gr0I)  
 have size ?P = sum (count ?P) (set-mset ?P) by (rule size-multiset-overloaded-eq)  
 also from assms have ... = sum (λ-. 1) (set-mset ?P)  
 by (intro sum.cong refl, subst count-prime-factorization-prime)  
 (auto simp: moebius-mu-eq-zero-iff squarefree-factorial-semiring')  
 also have ... = card (set-mset ?P) by simp  
 finally show ?thesis by (simp add: moebius-mu-squarefree-eq[OF assms])  
**qed**

**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** –  
 have  $(\sum d \mid d \text{ dvd } n. \text{moebius-mu } d :: \text{int}) =$   
 $(\sum d \in \text{Prod } \{P. P \subseteq \text{prime-factors } n\}. \text{moebius-mu } d)$   
**proof** (rule sum.mono-neutral-right; safe?)  
 fix A assume A: A  $\subseteq$  prime-factors n  
 from A have [simp]: finite A by (rule finite-subset) auto

```

from  $A$  have  $A'$ :  $x > 0$  prime  $x$  if  $x \in A$  for  $x$  using that
  by (auto simp: prime-factors-multiplicity prime-gt-0-nat)
from  $A'$  have  $A\text{-nz}$ :  $\prod A \neq 0$  by (intro notI) auto
from  $A'$  have prime-factorization  $(\prod A) = \text{sum prime-factorization } A$ 
  by (subst prime-factorization-prod) (auto dest: finite-subset)
also from  $A'$  have  $\dots = \text{sum } (\lambda x. \{\#x\}) A$ 
  by (intro sum.cong refl) (auto simp: prime-factorization-prime)
also have  $\dots = \text{mset-set } A$  by simp
also from  $A$  have  $\dots \subseteq\# \text{mset-set } (\text{prime-factors } n)$ 
  by (rule subset-imp-msubset-mset-set) simp-all
also have  $\dots \subseteq\# \text{prime-factorization } n$  by (rule mset-set-set-mset-msubset)
finally show  $\prod A \text{ dvd } n$  using  $A\text{-nz}$ 
  by (intro prime-factorization-subset-imp-dvd) auto
next
fix  $x$  assume  $x \notin \text{Prod } \{P. P \subseteq \text{prime-factors } n\}$   $x \text{ dvd } n$ 
from  $x$  assms have [simp]:  $x > 0$  by (auto intro!: Nat.gr0I)
{
  assume  $\text{nz}$ : moebius-mu  $x \neq 0$ 
  have  $(\prod (\text{set-mset } (\text{prime-factorization } x))) = (\prod_{p \in \text{prime-factors } x} p)^{\wedge \text{multiplicity } p } x$ 
    using  $\text{nz}$  by (intro prod.cong refl)
      (auto simp: moebius-mu-eq-zero-iff squarefree-factorial-semiring')
  also have  $\dots = x$  by (intro Primes.prime-factorization-nat [symmetric]) auto
  finally have  $x = \prod (\text{prime-factors } x)$   $\text{prime-factors } x \subseteq \text{prime-factors } n$ 
    using dvd-prime-factors[of n x] assms  $\langle x \text{ dvd } n \rangle$  by auto
  hence  $x \in \text{Prod } \{P. P \subseteq \text{prime-factors } n\}$  by blast
  with  $x(1)$  have False by contradiction
}
thus moebius-mu  $x = 0$  by blast
qed (insert assms, auto)
also have  $\dots = (\sum P \mid P \subseteq \text{prime-factors } n. \text{moebius-mu } (\prod P))$ 
  by (subst sum.reindex) (auto intro!: inj-on-Prod-primes dest: finite-subset)
also have  $\dots = (\sum P \mid P \subseteq \text{prime-factors } n. (-1)^{\wedge \text{card } P})$ 
proof (intro sum.cong refl)
  fix  $P$  assume  $P: P \in \{P. P \subseteq \text{prime-factors } n\}$ 
  hence [simp]: finite  $P$  by (auto dest: finite-subset)
  from  $P$  have prime: prime  $p$  if  $p \in P$  for  $p$  using that by (auto simp: prime-factors-dvd)
  hence squarefree  $(\prod P)$ 
    by (intro squarefree-prod-coprime prime-imp-coprime squarefree-prime)
      (auto simp: primes-dvd-imp-eq)
  hence moebius-mu  $(\prod P) = (-1)^{\wedge \text{card } (\text{prime-factors } (\prod P))}$ 
    by (rule moebius-mu-squarefree-eq)
  also from  $P$  have prime-factors  $(\prod P) = P$ 
    by (subst prime-factors-prod) (auto simp: prime-factorization-prime prime)
  finally show moebius-mu  $(\prod P) = (-1)^{\wedge \text{card } P}$  .
qed
also have  $\{P. P \subseteq \text{prime-factors } n\} =$ 
   $\{P. P \subseteq \text{prime-factors } n \wedge \text{even } (\text{card } P)\} \cup \{P. P \subseteq \text{prime-factors}$ 

```

$n \wedge \text{odd} (\text{card } P)\}$   
**(is - = ?A  $\cup$  ?B) by blast**  
**also have**  $(\sum P \in \dots (-1) \wedge \text{card } P) = (\sum P \in ?A. (-1) \wedge \text{card } P) + (\sum P \in ?B. (-1) \wedge \text{card } P)$   
**by** *(intro sum.union-disjoint) auto*  
**also have**  $(\sum P \in ?A. (-1) \wedge \text{card } P :: \text{int}) = (\sum P \in ?A. 1)$  **by** *(intro sum.cong refl) auto*  
**also have**  $\dots = \text{int} (\text{card } ?A)$  **by** *simp*  
**also have**  $(\sum P \in ?B. (-1) \wedge \text{card } P :: \text{int}) = (\sum P \in ?B. -1)$  **by** *(intro sum.cong refl) auto*  
**also have**  $\dots = -\text{int} (\text{card } ?B)$  **by** *simp*  
**also have**  $\text{card } ?B = \text{card } ?A$   
**by** *(rule card-even-odd-subset [symmetric])*  
*(insert assms, auto simp: prime-factorization-empty-iff)*  
**also have**  $\text{int} (\text{card } ?A) + (- \text{int} (\text{card } ?A)) = 0$  **by** *simp*  
**finally have**  $(\sum d \mid d \text{ dvd } n. \text{of-int} (\text{moebius-mu } d) :: 'a) = 0$   
**unfolding** *of-int-sum [symmetric]* **by** *(simp only: of-int-0)*  
**thus ?thesis by simp**  
**qed**

**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** –  
**have**  $n = 0 \vee n = 1 \vee n > 1$  **by** *force*  
**thus ?thesis using** *sum-moebius-mu-divisors[of n]* **by** *auto*  
**qed**

**lemma** *fds-zeta-times-moebius-mu*:  $\text{fds-zeta} * \text{fds moebius-mu} = 1$   
**proof**  
**fix**  $n :: \text{nat}$  **assume**  $n > 0$   
**from**  $n$  **have**  $\text{fds-nth} (\text{fds-zeta} * \text{fds moebius-mu} :: 'a \text{ fds}) n = (\sum d \mid d \text{ dvd } n. \text{moebius-mu } d)$   
**unfolding** *fds-nth-mult dirichlet-prod-altdef1*  
**by** *(intro sum.cong refl) (auto simp: fds-nth-fds elim: dvdE)*  
**also have**  $\dots = \text{fds-nth } 1 n$  **by** *(simp add: sum-moebius-mu-divisors')*  
**finally show**  $\text{fds-nth} (\text{fds-zeta} * \text{fds moebius-mu} :: 'a \text{ fds}) n = \text{fds-nth } 1 n$ .  
**qed**

**lemma** *fds-moebius-inverse-zeta*:  
 $\text{fds moebius-mu} = \text{inverse} (\text{fds-zeta} :: 'a :: \text{field } \text{fds})$   
**using** *fds-right-inverse-unique fds-zeta-times-moebius-mu* **by** *blast*

**lemma** *moebius-mu-formula-real*:  $(\text{moebius-mu } n :: \text{real}) = \text{dirichlet-inverse} (\lambda n. 1) 1 n$   
**proof** –  
**have**  $\text{moebius-mu } n = \text{fds-nth} (\text{fds moebius-mu}) n :: \text{real}$  **by** *simp*  
**also have**  $\text{fds moebius-mu} = (\text{inverse } \text{fds-zeta} :: \text{real } \text{fds})$  **by** *(fact fds-moebius-inverse-zeta)*  
**also have**  $\text{fds-nth} \dots n = \text{dirichlet-inverse} (\text{fds-nth } \text{fds-zeta}) 1 n$   
**unfolding** *fds-nth-inverse* **by** *simp*

**also have** ... = *dirichlet-inverse* ( $\lambda$ -. 1) 1 n **by** (*rule dirichlet-inverse-cong*)  
*simp-all*

**finally show** ?thesis .  
**qed**

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

**proof** –

**have** *real-of-int* (*moebius-mu* n) = *moebius-mu* n **by** *simp*  
**also have** ... = *dirichlet-inverse* ( $\lambda$ -. 1) 1 n **by** (*fact moebius-mu-formula-real*)  
**also have** ... = *real-of-int* (*dirichlet-inverse* ( $\lambda$ -. 1) 1 n)  
**by** (*induction n rule: dirichlet-inverse-induct*) (*simp-all add: dirichlet-inverse-gt-1*)  
**finally show** ?thesis **by** (*subst (asm) of-int-eq-iff*)

**qed**

**lemma** *moebius-mu-formula*: *moebius-mu* n = *dirichlet-inverse* ( $\lambda$ -. 1) 1 n

**by** (*subst of-int-moebius-mu [symmetric]*, *subst moebius-mu-formula-int*)  
(*simp add: of-int-dirichlet-inverse*)

**interpretation** *moebius-mu*: *multiplicative-function moebius-mu*

**proof** –

**have** *multiplicative-function* (*dirichlet-inverse* ( $\lambda n$ . if n = 0 then 0 else 1 :: 'a) 1)  
**by** (*rule multiplicative-dirichlet-inverse, standard*) *simp-all*  
**also have** *dirichlet-inverse* ( $\lambda n$ . if n = 0 then 0 else 1 :: 'a) 1 = *moebius-mu*  
**by** (*auto simp: fun-eq-iff moebius-mu-formula*)  
**finally show** *multiplicative-function* (*moebius-mu* :: nat  $\Rightarrow$  'a) .

**qed**

**interpretation** *moebius-mu*:

*multiplicative-function'* *moebius-mu*  $\lambda p k$ . if k = 1 then -1 else 0  $\lambda$ -. -1

**proof**

**fix** p k :: nat **assume** prime p k > 0  
**moreover from this have** *moebius-mu* p = -1  
**by** (*simp add: moebius-mu-def prime-factorization-prime squarefree-prime*)  
**ultimately show** *moebius-mu* (p ^ k) = (if k = 1 then -1 else 0)  
**by** (*auto simp: moebius-mu-power'*)

**qed** *auto*

**lemma** *moebius-mu-2* [*simp*]: *moebius-mu* 2 = -1

**and** *moebius-mu-3* [*simp*]: *moebius-mu* 3 = -1

**by** (*rule moebius-mu.prime; simp*)+

**lemma** *moebius-mu-code* [*code*]:

*moebius-mu* n = *of-int* (*dirichlet-inverse* ( $\lambda$ -. 1 :: int) 1 n)

**by** (*subst moebius-mu-formula-int [symmetric]*) *simp*

**lemma** *fds-moebius-inversion*:  $f = \text{fds moebius-mu} * g \longleftrightarrow g = f * \text{fds-zeta}$   
**by** (*metis fds-zeta-times-moebius-mu mult.commute mult.left-commute mult.right-neutral*)

**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** –

**from** *assms* **have**  $\text{fds } g = \text{fds } f * \text{fds-zeta}$

**by** (*intro fds-eqI*) (*simp add: fds-nth-mult dirichlet-prod-def*)

**thus** *?thesis* **using** *assms*

**by** (*subst (asm) fds-moebius-inversion [symmetric]*) (*simp add: fds-eq-iff fds-nth-mult*)

**qed**

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

**by** (*subst fds-moebius-inversion*) (*rule fds-mangoldt-times-zeta [symmetric]*)

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

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

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

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

**proof** –

**define**  $g$  **where**  $g = (\lambda n. \sum d \mid d \text{ dvd } n. \text{moebius-mu } d * f\ d)$

**define**  $g'$  **where**  $g' = \text{dirichlet-prod } (\lambda n. \text{moebius-mu } n * f\ n)$  ( *$\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1$* )

**interpret**  $f$ : *multiplicative-function*  $f$  **by** *fact*

**have** *multiplicative-function*  $(\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1 :: 'a)$

**by** *standard auto*

**interpret** *multiplicative-function*  $g'$  **unfolding**  $g'$ -*def*

**by** (*intro multiplicative-dirichlet-prod multiplicative-function-mult moebius-mu.multiplicative-function-axioms assms*) *fact+*

**have**  $g'$ -*primepow*:  $g' (p \wedge k) = 1 - f\ p$  **if** *prime*  $p\ k > 0$  **for**  $p\ k$

**proof** –

**have**  $g' (p \wedge k) = (\sum i \leq k. \text{moebius-mu } (p \wedge i) * f (p \wedge i))$

**using** *that* **by** (*simp add: g'-def dirichlet-prod-prime-power*)

**also** **have**  $\dots = (\sum i \in \{0, 1\}. \text{moebius-mu } (p \wedge i) * f (p \wedge i))$

**using** *that* **by** (*intro sum.mono-neutral-right*) (*auto simp: moebius-mu-power'*)

**also** **have**  $\dots = 1 - f\ p$

**using** *that* **by** (*simp add: moebius-mu.prime*)

**finally** **show** *?thesis* .

**qed**

**have**  $g' n = g\ n$

**by** (*simp add: g-def g'-def dirichlet-prod-def*)

**also** **from** *assms* **have**  $g' n = (\prod p \in \text{prime-factors } n. g' (p \wedge \text{multiplicity } p\ n))$

**by** (*intro prod-prime-factors*) *auto*

**also** **have**  $\dots = (\prod p \in \text{prime-factors } n. 1 - f\ p)$

by (intro prod.cong) (auto simp: g'-primepow prime-factors-multiplicity)  
**finally show** ?thesis by (simp add: g-def)  
**qed**

**lemma** completely-multiplicative-iff-inverse-moebius-mu:  
**fixes**  $f :: \text{nat} \Rightarrow 'a :: \{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$   
**assumes** multiplicative-function  $f$   
**defines**  $g \equiv \text{dirichlet-inverse } f \ 1$   
**shows** completely-multiplicative-function  $f \longleftrightarrow$   
 $(\forall n. g \ n = \text{moebius-mu } n * f \ n)$

**proof** –

**interpret** multiplicative-function  $f$  **by fact**

**show** ?thesis

**proof** safe

**assume** completely-multiplicative-function  $f$

**then interpret** completely-multiplicative-function  $f$  .

**have** [simp]:  $\text{fds } f \neq 0$  **by** (auto simp: fds-eq-iff)

**have**  $\text{fds } (\lambda n. \text{moebius-mu } n * f \ n) * \text{fds } f = 1$

**proof**

**fix**  $n :: \text{nat}$

**have**  $\text{fds-nth } (\text{fds } (\lambda n. \text{moebius-mu } n * f \ n) * \text{fds } f) \ n =$

$(\sum (r, d) \mid r * d = n. \text{moebius-mu } r * f \ (r * d))$

**by** (simp add: fds-eq-iff fds-nth-mult fds-nth-fds dirichlet-prod-altdef2 mult.mult.assoc)

**also have**  $\dots = (\sum (r, d) \mid r * d = n. \text{moebius-mu } r * f \ n)$

**by** (intro sum.cong) auto

**also have**  $\dots = \text{dirichlet-prod } \text{moebius-mu } (\lambda-. 1) \ n * f \ n$

**by** (simp add: dirichlet-prod-altdef2 sum-distrib-right case-prod-unfold mult)

**also have**  $\text{dirichlet-prod } \text{moebius-mu } (\lambda-. 1) \ n = \text{fds-nth } (\text{fds } \text{moebius-mu} * \text{fds-zeta}) \ n$

**by** (simp add: fds-nth-mult)

**also have**  $\text{fds } \text{moebius-mu} * \text{fds-zeta} = 1$

**by** (simp add: mult-ac fds-zeta-times-moebius-mu)

**also have**  $\text{fds-nth } 1 \ n * f \ n = \text{fds-nth } 1 \ n$

**by** (auto simp: fds-eq-iff fds-nth-one)

**finally show**  $\text{fds-nth } (\text{fds } (\lambda n. \text{moebius-mu } n * f \ n) * \text{fds } f) \ n = \text{fds-nth } 1 \ n$  .

**qed**

**also have**  $1 = \text{fds } g * \text{fds } f$

**by** (auto simp: fds-eq-iff g-def fds-nth-mult dirichlet-prod-inverse')

**finally have**  $\text{fds } g = \text{fds } (\lambda n. \text{moebius-mu } n * f \ n)$

**by** (subst (asm) mult-cancel-right) auto

**thus**  $g \ n = \text{moebius-mu } n * f \ n$  **for**  $n$

**by** (cases  $n = 0$ ) (auto simp: fds-eq-iff g-def)

**next**

**assume**  $g: \forall n. g \ n = \text{moebius-mu } n * f \ n$

**show** completely-multiplicative-function  $f$

```

proof (rule completely-multiplicativeI)
  fix p k :: nat assume pk: prime p k > 0
  show f (p ^ k) = f p ^ k
  proof (induction k)
    case (Suc k)
    have eq: dirichlet-prod g f n = 0 if n ≠ 1 for n
      unfolding g-def using dirichlet-prod-inverse'[of f 1] that by auto
    have dirichlet-prod g f (p ^ Suc k) = 0
      using pk by (intro eq) auto
    also have dirichlet-prod g f (p ^ Suc k) = (∑ i ≤ Suc k. g (p ^ i) * f (p ^
(Suc k - i)))
      by (intro dirichlet-prod-prime-power) fact+
    also have ... = (∑ i ≤ Suc k. moebius-mu (p ^ i) * f (p ^ i) * f (p ^ (Suc
k - i)))
      by (intro sum.cong refl, subst g) auto
    also have ... = (∑ i ∈ {0, 1}. moebius-mu (p ^ i) * f (p ^ i) * f (p ^ (Suc
k - i)))
      using pk by (intro sum.mono-neutral-right) (auto simp: moebius-mu-power')
    also have ... = f (p ^ Suc k) - f p ^ Suc k
      using pk Suc.IH by (auto simp: moebius-mu.prime)
    finally show f (p ^ Suc k) = f p ^ Suc k by simp
  qed auto
qed
qed
qed

```

```

lemma completely-multiplicative-fds-inverse:
  fixes f :: nat ⇒ 'a :: field
  assumes completely-multiplicative-function f
  shows inverse (fds f) = fds (λn. moebius-mu n * f n)
proof -
  interpret completely-multiplicative-function f by fact
  from assms show ?thesis
  by (subst (asm) completely-multiplicative-iff-inverse-moebius-mu)
  (auto simp: inverse-fds-def multiplicative-function-axioms)
qed

```

```

lemma completely-multiplicative-fds-inverse':
  fixes f :: 'a :: field fds
  assumes completely-multiplicative-function (fds-nth f)
  shows inverse f = fds (λn. moebius-mu n * fds-nth f n)
  by (metis assms completely-multiplicative-fds-inverse fds-fds-nth)

```

```

context
  includes fds-syntax
begin

```

```

lemma selberg-aux:

```

```

( $\chi$  n. of-real (( $\ln n$ )2)) * fds moebius-mu =
  (fds mangoldt)2 - fds-deriv (fds mangoldt :: 'a :: {comm-ring-1, real-algebra-1}
fds)
proof -
  have ( $\chi$  n. of-real ( $\ln$  (real n) ^ 2)) = fds-deriv (fds-deriv fds-zeta :: 'a fds)
  by (rule fds-eqI) (simp add: fds-nth-fds fds-nth-deriv power2-eq-square scaleR-conv-of-real)
  also have ... = (fds mangoldt ^ 2 - fds-deriv (fds mangoldt)) * fds-zeta
  by (simp add: fds-deriv-zeta algebra-simps power2-eq-square)
  also have ... * fds moebius-mu = ((fds mangoldt)2 - fds-deriv (fds mangoldt))
  *
    (fds-zeta * fds moebius-mu) by (simp add: mult-ac)
  also have fds-zeta * fds moebius-mu = (1 :: 'a fds) by (fact fds-zeta-times-moebius-mu)
  finally show ?thesis by simp
qed

```

```

lemma selberg-aux':
  mangoldt n * of-real ( $\ln n$ ) + (mangoldt * mangoldt) n =
    ((moebius-mu * ( $\lambda b. \text{of-real } (\ln b) ^ 2$ )) n
     :: 'a :: {comm-ring-1, real-algebra-1}) if n > 0
  using selberg-aux [symmetric] that
  by (auto simp add: fds-eq-iff fds-nth-mult power2-eq-square fds-nth-deriv
    dirichlet-prod-commutes algebra-simps scaleR-conv-of-real)

```

end

end

## 6 Euler's $\phi$ function

```

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

```

```

lemma fds-totient-times-zeta:
  fds ( $\lambda n. \text{of-nat } (\text{totient } n)$  :: 'a :: comm-semiring-1) * fds-zeta = fds of-nat
proof
  fix n :: nat assume n: n > 0
  have fds-nth (fds ( $\lambda n. \text{of-nat } (\text{totient } n)$ ) * fds-zeta) n =
    dirichlet-prod ( $\lambda n. \text{of-nat } (\text{totient } n)$ ) ( $\lambda-. 1$ ) n
  by (simp add: fds-nth-mult)
  also from n have ... = fds-nth (fds of-nat) n
  by (simp add: fds-nth-fds dirichlet-prod-def totient-divisor-sum of-nat-sum [symmetric]
    del: of-nat-sum)
  finally show fds-nth (fds ( $\lambda n. \text{of-nat } (\text{totient } n)$ ) * fds-zeta) n = fds-nth (fds
of-nat) n .
qed

```

**lemma** *fds-totient-times-zeta'*:  $\text{fds totient} * \text{fds-zeta} = \text{fds id}$   
**using** *fds-totient-times-zeta*[**where** 'a = nat] **by** *simp*

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

**proof** –

**have**  $\text{fds} (\lambda n. \text{of-nat} (\text{totient } n)) * \text{fds-zeta} * \text{fds moebius-mu} = \text{fds of-nat} * \text{fds moebius-mu}$

**by** (*simp add: fds-totient-times-zeta*)

**also have**  $\text{fds} (\lambda n. \text{of-nat} (\text{totient } n)) * \text{fds-zeta} * \text{fds moebius-mu} = \text{fds} (\lambda n. \text{of-nat} (\text{totient } n))$

**by** (*simp only: mult.assoc fds-zeta-times-moebius-mu mult-1-right*)

**finally show** *?thesis* .

**qed**

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

$\text{int} (\text{totient } n) = \text{dirichlet-prod moebius-mu int } n$

**proof** (*cases n = 0*)

**case** *False*

**show** *?thesis*

**by** (*rule moebius-inversion*)

(*insert False, simp-all add: of-nat-sum [symmetric] totient-divisor-sum del: of-nat-sum*)

**qed** *simp-all*

**interpretation** *totient: multiplicative-function totient*

**proof** –

**have** *multiplicative-function int* **by** *standard simp-all*

**hence** *multiplicative-function (dirichlet-prod moebius-mu int)*

**by** (*intro multiplicative-dirichlet-prod moebius-mu.multiplicative-function-axioms*)

**also have**  $\text{dirichlet-prod moebius-mu int} = (\lambda n. \text{int} (\text{totient } n))$

**by** (*simp add: fun-eq-iff totient-conv-moebius-mu*)

**finally show** *multiplicative-function totient* **by** (*rule multiplicative-function-of-natD*)

**qed**

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

**using** *prime-odd-nat[of p] prime-gt-1-nat[of p]* **by** (*cases p = 2*) *auto*

**lemma** *twopow-dvd-totient*:

**fixes**  $n :: \text{nat}$

**assumes**  $n > 0$

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

**shows**  $2^k \text{dvd totient } n$

**proof** –

**define**  $P$  **where**  $P = \{p \in \text{prime-factors } n. \text{odd } p\}$

**define**  $P'$  **where**  $P' = \{p \in \text{prime-factors } n. \text{even } p\}$

**define**  $r$  **where**  $r = (\lambda p. \text{multiplicity } p n)$

**from**  $\langle n > 0 \rangle$  **have**  $\text{totient } n = (\prod_{p \in \text{prime-factors } n. \text{totient } (p^r p)}$

**unfolding** *r-def* **by** (*rule totient.prod-prime-factors*)

**also have**  $\text{prime-factors } n = P \cup P'$

by (auto simp: P-def P'-def)  
 also have  $(\prod p \in \dots \text{totient } (p \wedge r p)) =$   
 $(\prod p \in P. \text{totient } (p \wedge r p)) * (\prod p \in P'. \text{totient } (p \wedge r p))$   
 by (subst prod.union-disjoint) (auto simp: P-def P'-def)  
 finally have eq: totient n = ... .

have  $p \wedge r p > 2$  if  $p \in P$  for  $p$

proof -

have  $p \neq 2$  using that by (auto simp: P-def)  
 moreover have  $p > 1$  using prime-gt-1-nat[of p] that by (auto simp: P-def)  
 ultimately have  $2 < p$  by linarith  
 also have  $p = p \wedge 1$  by simp  
 also have  $p \wedge 1 \leq p \wedge r p$   
 using that prime-gt-1-nat[of p]  
 by (intro power-increasing) (auto simp: P-def prime-factors-multiplicity r-def)  
 finally show ?thesis .

qed

hence  $(\prod p \in P. 2) \text{ dvd } (\prod p \in P. \text{totient } (p \wedge r p))$

by (intro prod-dvd-prod totient-even)

hence  $2 \wedge \text{card } P \text{ dvd } (\prod p \in P. \text{totient } (p \wedge r p))$

by simp

also have ... dvd  $(\prod p \in P. \text{totient } (p \wedge r p)) * (\prod p \in P'. \text{totient } (p \wedge r p))$

by simp

also have ... = totient n

by (rule eq [symmetric])

finally show ?thesis unfolding k-def P-def .

qed

lemma totient-conv-moebius-mu':

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

shows  $\text{real } (\text{totient } n) = \text{real } n * (\sum d \mid d \text{ dvd } n. \text{moebius-mu } d / \text{real } d)$

proof -

have  $\text{real } (\text{totient } n) = \text{of-int } (\text{int } (\text{totient } n))$  by simp

also have  $\text{int } (\text{totient } n) = (\sum d \mid d \text{ dvd } n. \text{moebius-mu } d * \text{int } (n \text{ div } d))$

using totient-conv-moebius-mu by (simp add: dirichlet-prod-def assms)

also have  $\text{real-of-int } (\sum d \mid d \text{ dvd } n. \text{moebius-mu } d * \text{int } (n \text{ div } d)) =$

$(\sum d \mid d \text{ dvd } n. \text{moebius-mu } d * \text{real } (n \text{ div } d))$  by simp

also have ... =  $(\sum d \mid d \text{ dvd } n. \text{real } n * \text{moebius-mu } d / \text{real } d)$

by (rule sum.cong) (simp-all add: field-char-0-class.of-nat-div)

also have ... =  $\text{real } n * (\sum d \mid d \text{ dvd } n. \text{moebius-mu } d / \text{real } d)$

by (simp add: sum-distrib-left)

finally show ?thesis .

qed

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 -

have  $\text{totient } (p \wedge \text{Suc } n) = p \wedge \text{Suc } n - \text{card } ((*) p \text{ ' } \{0 <.. p \wedge n\})$

**unfolding** *totient-def totatives-prime-power-Suc*[*OF assms*]  
**by** (*subst card-Diff-subset*) (*insert assms, auto simp: prime-gt-0-nat*)  
**also from** *assms* **have**  $\text{card } (\{0 <.. p^n\}) = p^n$   
**by** (*subst card-image*) (*auto simp: inj-on-def*)  
**finally show** *?thesis* .  
**qed**

**interpretation** *totient: multiplicative-function'* *totient*  $\lambda p k. p^k - p^{(k-1)}$   
 $\lambda p. p - 1$

**proof**  
**fix**  $p k :: \text{nat}$  **assume** *prime*  $p k > 0$   
**thus**  $\text{totient } (p^k) = p^k - p^{(k-1)}$   
**by** (*cases k*) (*simp-all add: totient-prime-power-Suc del: power-Suc*)  
**qed** *simp-all*

**end**

## 7 The Liouville $\lambda$ 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**  
 $\text{liouville-lambda } n = (\text{if } n = 0 \text{ then } 0 \text{ else } (-1)^{\text{size } (\text{prime-factorization } n)})$

**interpretation** *liouville-lambda: completely-multiplicative-function'* *liouville-lambda*  
 $\lambda \cdot. -1$

**proof**

**fix**  $a b :: \text{nat}$  **assume**  $a > 1 b > 1$   
**thus**  $\text{liouville-lambda } (a * b) = \text{liouville-lambda } a * \text{liouville-lambda } b$   
**by** (*simp add: liouville-lambda-def prime-factorization-mult power-add*)  
**qed** (*simp-all add: liouville-lambda-def prime-factorization-prime One-nat-def [symmetric]*)

*del: One-nat-def*)

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

**lemma** *liouville-lambda-prime-power* [*simp*]:  $\text{prime } p \implies \text{liouville-lambda } (p^k) = (-1)^k$   
**by** (*simp add: liouville-lambda-def prime-factorization-prime-power*)

**lemma** *liouville-lambda-squarefree*:  $\text{squarefree } n \implies \text{liouville-lambda } n = \text{moe-}$

*bius-mu n*  
**by** (*auto simp: liouville-lambda-def moebius-mu-squarefree-eq' intro!: Nat.gr0I*)

**lemma** *power-neg-one-If*:  $(-1)^n = (\text{if even } n \text{ then } 1 \text{ else } -1 :: 'a :: \text{ring-1})$   
**by** (*induction n*) (*simp-all split: if-splits*)

**lemma** *liouville-lambda-power-even*:  
 $n > 0 \implies \text{even } m \implies \text{liouville-lambda } (n \wedge m) = 1$   
**by** (*subst liouville-lambda.power*) (*auto elim!: evenE simp: liouville-lambda-def power-neg-one-If*)

**lemma** *liouville-lambda-power-odd*:  
 $\text{odd } m \implies \text{liouville-lambda } (n \wedge m) = \text{liouville-lambda } n$   
**by** (*subst liouville-lambda.power*) (*auto elim!: oddE simp: liouville-lambda-def power-neg-one-If*)

**lemma** *liouville-lambda-power*:  
 $\text{liouville-lambda } (n \wedge m) =$   
 $(\text{if } n = 0 \wedge m > 0 \text{ then } 0 \text{ else if even } m \text{ then } 1 \text{ else liouville-lambda } n)$   
**by** (*auto simp: liouville-lambda-power-even liouville-lambda-power-odd power-0-left*)

**interpretation** *squarefree: multiplicative-function'*  
*ind squarefree*  $\lambda p k. \text{if } k > 1 \text{ then } 0 \text{ else } 1 \lambda-. 1$   
**proof**  
**fix**  $p k :: \text{nat}$  **assume**  $\text{prime } p k > 0$   
**thus** *ind squarefree*  $(p \wedge k) = (\text{if } 1 < k \text{ then } 0 \text{ else } 1 :: 'a)$   
**by** (*cases k = 1*) (*auto simp: squarefree-power-iff squarefree-prime ind-def*)  
**qed** (*auto simp: squarefree-mult-coprime squarefree-power-iff ind-def dest: square-free-multD*  
*simp del: One-nat-def*)

**interpretation** *is-nth-power: multiplicative-function ind (is-nth-power n)*  
**by** *standard* (*auto simp: is-nth-power-mult-coprime-nat-iff*)

**interpretation** *is-nth-power: multiplicative-function'*  
*ind (is-nth-power n)*  $\lambda p k. \text{if } n \text{ dvd } k \text{ then } 1 \text{ else } 0 \lambda-. \text{if } n = 1 \text{ then } 1 \text{ else } 0$   
**by** *standard* (*simp-all add: is-nth-power-prime-power-nat-iff ind-def*)

**interpretation** *is-square: multiplicative-function ind is-square*  
**by** *standard* (*auto simp: is-nth-power-mult-coprime-nat-iff*)

**interpretation** *is-square: multiplicative-function'*  
*ind is-square*  $\lambda p k. \text{if even } k \text{ then } 1 \text{ else } 0 \lambda-. 0$   
**by** *standard* (*simp-all add: is-nth-power-prime-power-nat-iff ind-def*)

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

**proof** (rule *multiplicative-function-eqI*)  
**show** *multiplicative-function* ( $\lambda n. (\sum d \mid d \text{ dvd } n. \text{liouville-lambda } d)$ )  
**by** (rule *liouville-lambda.multiplicative-sum-divisors*)  
**show** *multiplicative-function* (*ind is-square*)  
**by** (rule *is-nth-power.multiplicative-function-axioms*)  
**next**  
**fix**  $p \ k :: \text{nat}$  **assume**  $pk: \text{prime } p \ k > 0$   
**hence**  $p\text{-gt-1}: p > 1$  **by** (*simp add: prime-gt-Suc-0-nat*)  
**have**  $(\sum d \mid d \text{ dvd } p^k. \text{liouville-lambda } d) = (\sum d \in (\lambda i. p^i) \text{ ' } \{..k\}. \text{liouville-lambda } d)$   
**using**  $pk$  **by** (*intro sum.cong refl*) (*auto intro: le-imp-power-dvd simp: divides-primelow-nat*)  
**also from**  $pk$  **and**  $p\text{-gt-1}$  **have**  $\dots = (\sum i \leq k. \text{liouville-lambda } (p^i))$   
**by** (*subst sum.reindex*) (*auto simp: inj-on-def prime-gt-1-nat*)  
**also from**  $pk$  **have**  $\dots = (\sum i \leq k. (-1)^i)$  **by** (*intro sum.cong refl*) *simp*  
**also have**  $\dots = (\text{if even } k \text{ then } 1 \text{ else } 0)$  **by** (*induction k*) *auto*  
**also from**  $pk$  **have**  $\dots = \text{ind is-square } (p^k)$  **by** (*simp add: is-square.prime-power*)  
**finally show**  $(\sum d \mid d \text{ dvd } p^k. \text{liouville-lambda } d) = \text{ind is-square } (p^k)$  .  
**qed**

**lemma** *fds-liouville-lambda-times-zeta*:  $\text{fds liouville-lambda} * \text{fds-zeta} = \text{fds-ind is-square}$

**by** (rule *fds-eqI*) (*simp add: liouville-lambda-divisors-sum fds-nth-mult dirichlet-prod-def*)

**lemma** *fds-liouville-lambda*:  $\text{fds liouville-lambda} = \text{fds-ind is-square} * \text{fds moebius-mu}$

**proof** –

**have**  $\text{fds liouville-lambda} * \text{fds-zeta} * \text{fds moebius-mu} = \text{fds-ind is-square} * \text{fds moebius-mu}$

**by** (*simp add: fds-liouville-lambda-times-zeta*)

**also have**  $\text{fds liouville-lambda} * \text{fds-zeta} * \text{fds moebius-mu} = \text{fds liouville-lambda}$

**by** (*simp only: mult.assoc fds-zeta-times-moebius-mu mult-1-right*)

**finally show** *?thesis* .

**qed**

**lemma** *liouville-lambda-altdef*:

$\text{liouville-lambda } n = (\sum d \mid d^2 \text{ dvd } n. \text{moebius-mu } (n \text{ div } d^2))$

**proof** (*cases n = 0*)

**case** *False*

**have**  $\text{liouville-lambda } n = \text{fds-nth } (\text{fds liouville-lambda}) \ n$  **by** (*simp add: fds-nth-fds*)

**also have**  $\text{fds liouville-lambda} = \text{fds-ind is-square} * (\text{fds moebius-mu} :: \text{'a fds})$

**by** (rule *fds-liouville-lambda*)

**also have**  $\text{fds-nth } \dots \ n = (\sum d \mid d \text{ dvd } n. \text{ind is-square } d * \text{moebius-mu } (n \text{ div } d))$

**by** (*simp add: fds-nth-mult dirichlet-prod-def*)

**also have**  $\dots = (\sum d \in (\lambda d. d^2) \text{ ' } \{d. d^2 \text{ dvd } n\}. \text{moebius-mu } (n \text{ div } d))$

**using** *False*

**by** (*intro sum.mono-neutral-cong-right*) (*auto simp: ind-def is-nth-power-def*)

**also have**  $\dots = (\sum d \mid d \wedge 2 \text{ dvd } n. \text{moebius-mu } (n \text{ div } d \wedge 2))$   
**by** (*subst sum.reindex*) (*auto simp: inj-on-def dest: power2-eq-imp-eq*)  
**finally show** *?thesis* .  
**qed** *auto*

**lemma** *abs-moebius-mu*:  $\text{abs } (\text{moebius-mu } n :: 'a :: \text{linordered-idom}) = \text{ind square-free } n$   
**by** (*auto simp: ind-def moebius-mu-def*)

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

**lemma** *divisor-sigma-0* [*simp*]:  $\text{divisor-sigma } x \ 0 = 0$   
**by** (*simp add: divisor-sigma-def*)

**lemma** *divisor-sigma-Suc-0* [*simp*]:  $\text{divisor-sigma } x \ (\text{Suc } 0) = 1$   
**by** (*simp add: divisor-sigma-def*)

**lemma** *divisor-sigma-1* [*simp*]:  $\text{divisor-sigma } x \ 1 = 1$   
**by** *simp*

**lemma** *fds-divisor-sigma*:  $\text{fds } (\text{divisor-sigma } x) = \text{fds-zeta} * \text{fds-shift } x \ \text{fds-zeta}$   
**by** (*rule fds-eqI*) (*simp add: fds-nth-mult dirichlet-prod-altdef1 divisor-sigma-def*)

**interpretation** *divisor-sigma*: *multiplicative-function divisor-sigma x*

**proof** –

**have** *multiplicative-function* (*dirichlet-prod* ( $\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1$ )  
 $(\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } \text{nat-power } n \ x)$ ) (**is** *multiplicative-function* *?f*)

**by** (*rule multiplicative-dirichlet-prod; standard*)

(*simp-all add: nat-power-mult-distrib*)

**also have**  $?f \ n = \text{divisor-sigma } x \ n$  **for**  $n$

**using** *fds-divisor-sigma*[*of x*]

**by** (*cases n = 0*) (*simp-all add: fds-eq-iff fds-nth-mult*)

**hence**  $?f = \text{divisor-sigma } x \ ..$

**finally show** *multiplicative-function* (*divisor-sigma* *x*) .  
**qed**

**lemma** *divisor-sigma-naive* [code]:

*divisor-sigma* *x* *n* = (if *n* = 0 then 0 else *fold-atLeastAtMost-nat*  
 $(\lambda d \text{ acc. if } d \text{ dvd } n \text{ then nat-power } d \text{ } x + \text{acc else acc}) \text{ } 1 \text{ } n \text{ } 0$ )

**proof** (*cases* *n* = 0)

**case** *False*

**have** *divisor-sigma* *x* *n* =  $(\sum d \in \{1..n\}. \text{if } d \text{ dvd } n \text{ then nat-power } d \text{ } x \text{ else } 0)$

**unfolding** *divisor-sigma-def* **using** *False* **by** (*intro* *sum.mono-neutral-cong-left*)  
*(auto elim: dvdE)*

**also have** ... = *fold-atLeastAtMost-nat*

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

**by** (*rule* *sum-atLeastAtMost-code*)

**also have**  $(\lambda d \text{ acc. (if } d \text{ dvd } n \text{ then nat-power } d \text{ } x \text{ else } 0) + \text{acc}) =$   
 $(\lambda d \text{ acc. (if } d \text{ dvd } n \text{ then nat-power } d \text{ } x + \text{acc else acc}))$

**by** (*auto simp: fun-eq-iff*)

**finally show** *?thesis* **using** *False* **by** *simp*

**qed** *auto*

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

**proof** (*cases* *n* = 0)

**case** *False*

**show** *?thesis* **unfolding** *divisor-sigma-def of-nat-sum*

**by** (*intro* *sum.cong refl, subst nat-power-of-nat*) (*insert* *False, auto elim: dvdE*)

**qed** *auto*

**lemma** *divisor-sigma-prime-power-field*:

**fixes** *x* :: 'a :: {field, nat-power}

**assumes** *prime* *p*

**shows** *divisor-sigma* *x* (*p* ^ *k*) =

$(\text{if nat-power } p \text{ } x = 1 \text{ then of-nat } (k + 1) \text{ else}$

$(\text{nat-power } p \text{ } x \text{ } ^{\text{Suc } k - 1}) / (\text{nat-power } p \text{ } x - 1))$

**proof** –

**have** *divisor-sigma* *x* (*p* ^ *k*) =  $(\sum i \leq k. \text{nat-power } (p \text{ } ^i) \text{ } x)$

**unfolding** *divisor-sigma-def*

**by** (*rule* *sum.reindex-bij-betw* [*symmetric*])

*(insert* *assms, auto simp: bij-betw-def inj-on-def prime-gt-Suc-0-nat*

*divides-primepow-nat intro: le-imp-power-dvd)*

**also have** ... =  $(\sum i \leq k. \text{nat-power } p \text{ } x \text{ } ^i)$

**using** *assms* **by** (*intro* *sum.cong refl*) (*simp-all add: prime-gt-0-nat nat-power-power-left*)

**also have** ... =  $(\text{if nat-power } p \text{ } x = 1 \text{ then of-nat } (k + 1) \text{ else}$

$(\text{nat-power } p \text{ } x \text{ } ^{\text{Suc } k - 1}) / (\text{nat-power } p \text{ } x - 1))$

**using** *geometric-sum*[*of nat-power p x Suc k*] **unfolding** *lessThan-Suc-atMost*

**by** (*auto split: if-splits*)

**finally show** *?thesis* .

**qed**

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

**assumes** *prime p*

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

**proof** (*cases x = 0*)

**case** *True*

**with** *assms* **have** *nat-power p (real x) = 1* **by** *simp*

**hence**  $\text{divisor-sigma } (\text{real } x) (p \wedge k) = \text{real } (\text{Suc } k)$

**by** (*subst divisor-sigma-prime-power-field*) (*simp-all del: nat-power-real-def add: assms*)

**thus** *?thesis unfolding divisor-sigma-of-nat* **by** (*subst (asm) of-nat-eq-iff*) (*insert True, simp*)

**next**

**case** *False*

**with** *assms* **have** *gt-1: p  $\wedge$  x > 1*

**using** *power-gt1 [of p x - 1]* **by** (*simp add: prime-gt-Suc-0-nat*)

**hence** *not-one: real p  $\wedge$  x  $\neq$  1*

**unfolding** *of-nat-power [symmetric] of-nat-eq-1-iff* **by** (*intro notI*) *simp*

**from** *gt-1* **have** *dvd: p  $\wedge$  x - 1 dvd p  $\wedge$  (x \* Suc k) - 1*

**using** *geometric-sum-nat-dvd [of p  $\wedge$  x Suc k] assms*

**by** (*simp add: power-mult prime-gt-Suc-0-nat power-add*)

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

**by** (*subst divisor-sigma-prime-power-field [OF assms, where 'a = real]*) (*insert assms False dvd not-one, auto simp del: power-Suc nat-power-real-def simp: prime-gt-0-nat real-of-nat-div of-nat-diff prime-ge-Suc-0-nat power-mult [symmetric]*)

**thus** *?thesis unfolding divisor-sigma-of-nat* **by** (*subst (asm) of-nat-eq-iff*)

**qed**

**interpretation** *divisor-sigma-field*:

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

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

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

**by** *standard (auto simp: divisor-sigma-prime-power-field prime-gt-0-nat field-simps)*

**interpretation** *divisor-sigma-real*:

*multiplicative-function' divisor-sigma (x :: real)*

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

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

**proof** (*standard, goal-cases*)

**case** (*1 p k*)

**thus** *?case*

**by** (*auto simp: divisor-sigma-prime-power-field prime-gt-0-nat powr-def of-nat-eq-1-iff exp-of-nat-mult [symmetric] mult-ac simp del: of-nat-Suc power-Suc*)

**next**

**case** (*2 p*)

hence  $\text{real } p \text{ powr } x \neq 1$  if  $x \neq 0$  by (auto simp: powr-def that prime-gt-0-nat of-nat-eq-1-iff)  
 with 2 show ?case by (auto simp: field-simps)  
 qed

**interpretation** *divisor-sigma-nat*:

*multiplicative-function'* *divisor-sigma* ( $x :: \text{nat}$ )  
 $\lambda p k. \text{if } x = 0 \text{ then } \text{Suc } k \text{ else } (p \wedge (\text{Suc } k * x) - 1) \text{ div } (p \wedge x - 1)$   
 $\lambda p. p \wedge x + 1$

**proof** (standard, goal-cases)

case (2 p)

have  $(p \wedge (x + x) - 1) = (p \wedge x + 1) * (p \wedge x - 1)$

by (simp add: algebra-simps power-add)

moreover have  $p \wedge x > 1$  if  $x > 0$  using that 2 one-less-power prime-gt-1-nat by blast

ultimately show ?case using prime-ge-Suc-0-nat[of p] by auto

qed (auto simp: divisor-sigma-prime-power-nat mult-ac)

**lemma** *divisor-sigma-prime*:

assumes prime p

shows *divisor-sigma* x p = nat-power p x + 1

**proof** –

have *divisor-sigma* x p =  $(\sum d \mid d \text{ dvd } p. \text{nat-power } d x)$

by (simp add: divisor-sigma-def)

also from assms have  $\{d. d \text{ dvd } p\} = \{1, p\}$  by (auto simp: prime-nat-iff)

also have  $(\sum d \in \dots. \text{nat-power } d x) = \text{nat-power } p x + 1$

using assms by (subst sum.insert) (auto simp: add-ac)

finally show ?thesis .

qed

## 8.2 The divisor-counting function

**definition** *divisor-count* :: nat  $\Rightarrow$  nat where

*divisor-count* n = card {d. d dvd n}

**lemma** *divisor-count-0* [simp]: *divisor-count* 0 = 0

by (simp add: divisor-count-def)

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

by (simp add: divisor-count-def)

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

by (simp add: divisor-sigma-def divisor-count-def)

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

unfolding *divisor-sigma-0-left-nat* [symmetric] *divisor-sigma-of-nat* [symmetric]

by simp

**lemma** *divisor-count-altdef*: *divisor-count* n = *divisor-sigma* 0 n

by (simp add: divisor-sigma-0-left)

**lemma** *divisor-count-naive* [code]:  
divisor-count  $n = (\text{if } n = 0 \text{ then } 0 \text{ else}$   
fold-atLeastAtMost-nat  $(\lambda d \text{ acc. if } d \text{ dvd } n \text{ then } \text{Suc acc} \text{ else } \text{acc}) \ 1 \ n \ 0)$   
using *divisor-sigma-naive*[of 0 :: nat n]  
by (simp split: if-splits add: divisor-count-altdef cong: if-cong)

**interpretation** *divisor-count*: multiplicative-function' divisor-count  $\lambda p \ k. \text{Suc } k$   
 $\lambda -. \ 2$   
by standard (simp-all add: divisor-count-altdef divisor-sigma.mult-coprime  
divisor-sigma-nat.prime-power)

**lemma** *divisor-count-dvd-mono*:  
assumes  $a \text{ dvd } b \ b \neq 0$   
shows  $\text{divisor-count } a \leq \text{divisor-count } b$   
using *assms* by (auto simp: divisor-count-def intro!: card-mono intro: dvd-trans)

### 8.3 The divisor sum function

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

**lemma** *divisor-sum-0* [simp]: divisor-sum 0 = 0  
by (simp add: divisor-sum-def)

**lemma** *divisor-sum-Suc-0* [simp]: divisor-sum (Suc 0) = Suc 0  
by (simp add: divisor-sum-def)

**lemma** *divisor-sigma-1-left-nat*: divisor-sigma (Suc 0)  $n = \text{divisor-sum } n$   
by (simp add: divisor-sum-def divisor-sigma-def)

**lemma** *divisor-sigma-1-left*: divisor-sigma 1  $n = \text{of-nat } (\text{divisor-sum } n)$   
by (simp add: divisor-sum-def divisor-sigma-def)

**lemma** *divisor-sum-altdef*: divisor-sum  $n = \text{divisor-sigma } 1 \ n$   
by (simp add: divisor-sigma-1-left-nat)

**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** (standard, goal-cases)

case (5 p)

thus ?case using *divisor-sigma-nat.prime-aux*[of p 1]

by (simp-all add: divisor-sum-altdef)

qed (simp-all add: divisor-sum-altdef divisor-sigma-nat.prime-power divisor-sigma.mult-coprime)

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

**using** *assms*  
**by** (*cases a = 0*) (*auto simp: divisor-sum-def intro!: sum-le-included intro: dvd-trans*)

**lemma** *divisor-sum-naive* [*code*]:  
*divisor-sum n = (if n = 0 then 0 else*  
*fold-atLeastAtMost-nat (λd acc. if d dvd n then d + acc else acc) 1 n 0)*  
**using** *divisor-sigma-naive*[*of Suc 0 n*]  
**by** (*simp split: if-splits add: divisor-sum-altdef cong: if-cong*)

**lemma** *fds-divisor-count*: *fds divisor-count = fds-zeta ^ 2*  
**by** (*rule fds-eqI*)  
(*simp add: fds-nth-mult dirichlet-prod-altdef1 divisor-count-def power2-eq-square*)

**lemma** *fds-shift-zeta-1*: *fds-shift 1 fds-zeta = fds of-nat*  
**by** (*rule fds-eqI*) (*simp add: fds-nth-mult*)

**lemma** *fds-shift-zeta-Suc-0*: *fds-shift (Suc 0) fds-zeta = fds id*  
**by** (*rule fds-eqI*) (*simp add: fds-nth-mult*)

**lemma** *fds-divisor-sum*: *fds divisor-sum = fds-zeta \* fds id*  
**by** (*rule fds-eqI*) (*simp add: fds-nth-mult dirichlet-prod-altdef1 divisor-sum-def*)

**lemma** *fds-divisor-sum-eq-totient-times-d*: *fds divisor-sum = fds totient \* fds di-*  
*visor-count*

**proof** –  
**have** *fds divisor-sum = fds-zeta \* fds id* **by** (*fact fds-divisor-sum*)  
**also have** *fds id = fds totient \* fds-zeta* **by** (*rule fds-totient-times-zeta' [symmetric]*)  
**also have** *fds-zeta \* ... = fds totient \* fds divisor-count*  
**using** *fds-divisor-count* **by** (*simp add: power2-eq-square mult-ac*)  
**finally show** *?thesis* .

**qed**

**lemma** *fds-divisor-sum-times-moebius-mu*:

*fds (divisor-sigma (1 :: 'a :: {nat-power, comm-ring-1})) \* fds moebius-mu = fds*  
*of-nat*

**proof** –  
**have** *fds (divisor-sigma 1) \* fds moebius-mu =*  
*fds of-nat \* (fds-zeta \* fds moebius-mu :: 'a fds)*  
**by** (*subst mult.assoc [symmetric], subst fds-zeta-commutes [symmetric]*)  
(*simp add: fds-divisor-sigma fds-shift-zeta-1*)  
**also have** *fds-zeta \* fds moebius-mu = (1 :: 'a fds)* **by** (*fact fds-zeta-times-moebius-mu*)  
**finally show** *?thesis* **by** *simp*

**qed**

**lemma** *inverse-divisor-sigma*:

**fixes** *a :: 'a :: {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** –  
**have**  $\text{fds} (\text{divisor-sigma } a) = \text{fds-zeta} * \text{fds-shift } a \text{fds-zeta}$   
**by** (*simp add: fds-divisor-sigma*)  
**also have**  $\text{inverse} \dots = \text{fds moebius-mu} * \text{inverse} (\text{fds-shift } a \text{fds-zeta})$   
**by** (*simp add: fds-moebius-inverse-zeta inverse-mult-fds*)  
**also have**  $\text{inverse} (\text{fds-shift } a \text{fds-zeta}) =$   
 $\text{fds} (\lambda n. \text{moebius-mu } n * \text{fds-nth} (\text{fds-shift } a \text{fds-zeta}) n)$   
**by** (*intro completely-multiplicative-fds-inverse', unfold-locales*)  
*(auto simp: nat-power-mult-distrib)*  
**also have**  $\dots = \text{fds-shift } a (\text{fds moebius-mu})$   
**by** (*auto simp: fds-eq-iff*)  
**finally show** *?thesis* **by** (*simp add: mult.commute*)  
**qed**  
**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*:  $\text{sum-upto } f x = (\sum i \in \{0 < .. \text{nat } \lfloor x \rfloor\}. f i)$

**unfolding** *sum-upto-def*

**by** (*cases x ≥ 0; intro sum.cong refl*) (*auto simp: le-nat-iff le-floor-iff*)

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

**by** (*simp add: sum-upto-altdef*)

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

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

**by** (*simp add: sum-upto-def*)

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

**proof** (*rule finite-subset*)

**show**  $\text{finite } \{n. n \leq \text{nat } \lfloor x \rfloor\}$  **by** *auto*

**show**  $\{n. 0 < n \wedge \text{real } n \leq x\} \subseteq \{n. n \leq \text{nat } \lfloor x \rfloor\}$  **by** *safe linarith*

qed

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

**proof** –

have  $sum-upto (ind P :: nat \Rightarrow 'a) x = (\sum n \mid 0 < n \wedge real n \leq x \wedge P n. 1)$

unfolding *sum-upto-def* by (intro *sum.mono-neutral-cong-right*) (auto simp: *ind-def*)

also have  $\dots = of-nat (card \{n. n > 0 \wedge real n \leq x \wedge P n\})$  by *simp*

finally show *?thesis* .

qed

**lemma** *sum-upto-sum-divisors*:

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

**proof** –

let  $?B = (SIGMA k:\{k. 0 < k \wedge real k \leq x\}. \{d. 0 < d \wedge real d \leq x / real k\})$

let  $?A = (SIGMA k:\{k. 0 < k \wedge real k \leq x\}. \{d. d dvd k\})$

have  $*$ :  $real a \leq x$  if  $real (a * b) \leq x$   $b > 0$  for  $a b$

**proof** –

have  $real a * 1 \leq real (a * b)$  unfolding *of-nat-mult* using that

by (intro *mult-left-mono*) auto

also have  $\dots \leq x$  by *fact*

finally show *?thesis* by *simp*

qed

have *bij*: *bij-betw*  $(\lambda(k,d). (d * k, k)) ?B ?A$

by (rule *bij-betwI*[**where**  $g = \lambda(k,d). (d, k \text{ div } d)$ ])

(auto simp: *\* divide-simps mult.commute elim!*: *dvdE*)

have  $sum-upto (\lambda n. \sum d \mid d dvd n. f n d) x = (\sum (k,d) \in ?A. f k d)$

unfolding *sum-upto-def* by (rule *sum.Sigma*) auto

also have  $\dots = (\sum (k,d) \in ?B. f (d * k) k)$

by (*subst sum.reindex-bij-betw*[*OF* *bij*, *symmetric*]) (auto simp: *case-prod-unfold*)

also have  $\dots = sum-upto (\lambda k. sum-upto (\lambda d. f (d * k) k) (x / k)) x$

unfolding *sum-upto-def* by (rule *sum.Sigma* [*symmetric*]) auto

finally show *?thesis* .

qed

**lemma** *sum-upto-dirichlet-prod*:

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

unfolding *dirichlet-prod-def*

by (*subst sum-upto-sum-divisors*) (*simp add: sum-upto-def sum-distrib-left*)

**lemma** *sum-upto-real*:

assumes  $x \geq 0$

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

**proof** –

have  $A: 2 * \sum \{1..n\} = n * Suc n$  for  $n$  by (*induction n*) *simp-all*

have  $2 * sum-upto real x = real (2 * \sum \{0<..nat \lfloor x \rfloor\})$  by (*simp add: sum-upto-altdef*)

**also have**  $\{0 < .. \text{nat } [x]\} = \{1 .. \text{nat } [x]\}$  **by** *auto*  
**also note**  $A$   
**also have**  $\text{real } (\text{nat } [x] * \text{Suc } (\text{nat } [x])) = \text{of-int } (\text{floor } x) * (\text{of-int } (\text{floor } x) + 1)$  **using** *assms*  
**by** (*simp add: algebra-simps*)  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *summable-imp-convergent-sum-upto*:  
**assumes** *summable* ( $f :: \text{nat} \Rightarrow 'a :: \text{real-normed-vector}$ )  
**obtains**  $c$  **where** ( $\text{sum-upto } f \longrightarrow c$ ) *at-top*  
**proof** –  
**from** *assms* **have** *summable* ( $\lambda n. f (\text{Suc } n)$ )  
**by** (*subst summable-Suc-iff*)  
**then obtain**  $c$  **where** ( $\lambda n. f (\text{Suc } n)$ ) *sums*  $c$  **by** (*auto simp: summable-def*)  
**hence** ( $\lambda n. \sum k < n. f (\text{Suc } k)$ )  $\longrightarrow c$  **by** (*auto simp: sums-def*)  
**also have** ( $\lambda n. \sum k < n. f (\text{Suc } k)$ ) = ( $\lambda n. \sum k \in \{0 < .. n\}. f k$ )  
**by** (*subst sum.atLeast1-atMost-eq [symmetric]*) (*auto simp: atLeastSucAtMost-greaterThanAtMost*)  
**finally have** ( $(\lambda x. \text{sum } f \{0 < .. \text{nat } [x]\}) \longrightarrow c$ ) *at-top*  
**by** (*rule filterlim-compose*)  
*(auto intro!: filterlim-compose[OF filterlim-nat-sequentially] filterlim-floor-sequentially)*  
**also have** ( $\lambda x. \text{sum } f \{0 < .. \text{nat } [x]\}) = \text{sum-upto } f$   
**by** (*intro ext*) (*simp-all add: sum-upto-altdef*)  
**finally show** *?thesis* **using** *that[of c]* **by** *blast*  
**qed**

## 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** –  
**from** *assms* **have** [*simp*]:  $x \geq 0$  **by** *auto*  
**{**  
**fix**  $a b :: \text{real}$  **assume**  $ab: a > 0 b > 0 x \geq 0 a * b \leq x a > A b > B$   
**hence**  $a * b > A * B$  **using** *assms* **by** (*intro mult-strict-mono*) *auto*  
**also from** *assms* **have**  $A * B = x$  **by** *simp*  
**finally have** *False* **using**  $\langle a * b \leq x \rangle$  **by** *simp*  
**}** **note**  $*$  = *this*  
**have**  $*$ :  $a \leq A \vee b \leq B$  **if**  $a * b \leq x a > 0 b > 0 x \geq 0$  **for**  $a b$   
**by** (*rule ccontr*) (*insert \*[of a b] that, auto*)

**have** *nat-mult-leD1*:  $\text{real } a \leq x$  **if**  $\text{real } a * \text{real } b \leq x b > 0$  **for**  $a b$

**proof** –

**from** *that* **have**  $\text{real } a * 1 \leq \text{real } a * \text{real } b$  **by** (*intro mult-left-mono*) *simp-all*  
**also have**  $\dots \leq x$  **by** *fact*

**finally show** *?thesis by simp*  
**qed**  
**have** *nat-mult-leD2: real b ≤ x if real a \* real b ≤ x a > 0 for a b*  
**using** *nat-mult-leD1 [of b a] that by (simp add: mult-ac)*

**have** *le-sqrt-mult-imp-le: a \* b ≤ x*  
**if** *a ≥ 0 b ≥ 0 a ≤ A b ≤ B for a b :: real*  
**proof** *-*  
**from** *that and assms have a \* b ≤ A \* B by (intro mult-mono) auto*  
**with** *assms show a \* b ≤ x by simp*  
**qed**

**define** *F G where F = sum-upto f and G = sum-upto g*  
**let** *?Bound = {0 <.. nat [x]} × {0 <.. nat [x]}*  
**let** *?B = {(r,d). 0 < r ∧ real r ≤ A ∧ 0 < d ∧ real d ≤ x / real r}*  
**let** *?C = {(r,d). 0 < d ∧ real d ≤ B ∧ 0 < r ∧ real r ≤ x / real d}*  
**let** *?B' = SIGMA r: {r. 0 < r ∧ real r ≤ A}. {d. 0 < d ∧ real d ≤ x / real r}*  
**let** *?C' = SIGMA d: {d. 0 < d ∧ real d ≤ B}. {r. 0 < r ∧ real r ≤ x / real d}*  
**have** *sum-upto (dirichlet-prod f g) x + F A \* G B =*  

$$(\sum (i,(r,d)) \in (\text{SIGMA } i: \{i. 0 < i \wedge \text{real } i \leq x\}. \{(r,d). r * d = i\}). f r * g d) +$$

$$\text{sum-upto } f A * \text{sum-upto } g B \text{ (is - = ?S + -)}$$
**unfolding** *sum-upto-def dirichlet-prod-altdef2 F-def G-def*  
**by** *(subst sum.Sigma) (auto intro: finite-divisors-nat')*  
**also have** *?S = (∑ (r,d) | 0 < r ∧ 0 < d ∧ real (r \* d) ≤ x. f r \* g d)*  
*(is - = sum - ?A) by (intro sum.reindex-bij-witness [of - λ(r,d). (r\*d,(r,d))*  
*snd]) auto*  
**also have** *?A = ?B ∪ ?C by (auto simp: field-simps dest: \*)*  
**also have** *sum-upto f A \* sum-upto g B =*  

$$(\sum r | 0 < r \wedge \text{real } r \leq A. \sum d | 0 < d \wedge \text{real } d \leq B. f r * g d)$$
**by** *(simp add: sum-upto-def sum-product)*  
**also have** *... = (∑ (r,d) ∈ {r. 0 < r ∧ real r ≤ A} × {d. 0 < d ∧ real d ≤ B}. f r \* g d)*  
*(is - = sum - ?X) by (rule sum.cartesian-product)*  
**also have** *?X = ?B ∩ ?C by (auto simp: field-simps le-sqrt-mult-imp-le)*  
**also have** 
$$(\sum (r,d) \in ?B \cup ?C. f r * g d) + (\sum (r,d) \in ?B \cap ?C. f r * g d) =$$

$$(\sum (r,d) \in ?B. f r * g d) + (\sum (r,d) \in ?C. f r * g d)$$
**by** *(intro sum.union-inter finite-subset [of ?B ?Bound] finite-subset [of ?C ?Bound])*  
*(auto simp: field-simps le-nat-iff le-floor-iff dest: nat-mult-leD1 nat-mult-leD2)*  
**also have** *?B = ?B' by auto*  
**hence**  $(\lambda f. \text{sum } f ?B) = (\lambda f. \text{sum } f ?B')$  **by** *simp*  
**also have**  $(\sum (r,d) \in ?B'. f r * g d) = \text{sum-upto } (\lambda n. f n * G (x / \text{real } n)) A$   
**by** *(subst sum.Sigma [symmetric]) (simp-all add: sum-upto-def sum-distrib-left G-def)*  
**also have**  $(\sum (r,d) \in ?C. f r * g d) = (\sum (d,r) \in ?C'. f r * g d)$   
**by** *(intro sum.reindex-bij-witness [of - λ(x,y). (y,x) λ(x,y). (y,x)]) auto*  
**also have** *... = sum-upto (λn. F (x / real n) \* g n) B*  
**by** *(subst sum.Sigma [symmetric]) (simp-all add: sum-upto-def sum-distrib-right F-def)*

**finally show** *?thesis* **by** (*simp only: F-def G-def*)  
**qed**

**lemma** *hyperbola-method-semiring-sqrt*:  
**fixes**  $f g :: \text{nat} \Rightarrow 'a :: \text{comm-semiring-0}$   
**assumes**  $x \geq 0$   
**shows**  $\text{sum-upto} (\text{dirichlet-prod } f g) x + \text{sum-upto } f (\text{sqrt } x) * \text{sum-upto } g (\text{sqrt } x) =$   
 $\text{sum-upto } (\lambda n. f n * \text{sum-upto } g (x / \text{real } n)) (\text{sqrt } x) +$   
 $\text{sum-upto } (\lambda n. \text{sum-upto } f (x / \text{real } n) * g n) (\text{sqrt } x)$   
**using** *assms hyperbola-method-semiring[of sqrt x sqrt x]* **by** *simp*

**lemma** *hyperbola-method*:  
**fixes**  $f g :: \text{nat} \Rightarrow 'a :: \text{comm-ring}$   
**assumes**  $A \geq 0 B \geq 0 A * B = x$   
**shows**  $\text{sum-upto} (\text{dirichlet-prod } f g) x =$   
 $\text{sum-upto } (\lambda n. f n * \text{sum-upto } g (x / \text{real } n)) A +$   
 $\text{sum-upto } (\lambda n. \text{sum-upto } f (x / \text{real } n) * g n) B -$   
 $\text{sum-upto } f A * \text{sum-upto } g B$   
**using** *hyperbola-method-semiring[OF assms, of f g]* **by** (*simp add: algebra-simps*)

**lemma** *hyperbola-method-sqrt*:  
**fixes**  $f g :: \text{nat} \Rightarrow 'a :: \text{comm-ring}$   
**assumes**  $x \geq 0$   
**shows**  $\text{sum-upto} (\text{dirichlet-prod } f g) x =$   
 $\text{sum-upto } (\lambda n. f n * \text{sum-upto } g (x / \text{real } n)) (\text{sqrt } x) +$   
 $\text{sum-upto } (\lambda n. \text{sum-upto } f (x / \text{real } n) * g n) (\text{sqrt } x) -$   
 $\text{sum-upto } f (\text{sqrt } x) * \text{sum-upto } g (\text{sqrt } x)$   
**using** *assms hyperbola-method[of sqrt x sqrt x]* **by** *simp*

**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** (*rule finite-subset*)

**show** *real* -  $\{y <..x\} \subseteq \{\text{nat } \lfloor y \rfloor .. \text{nat } \lceil x \rceil\}$

**by** (*cases*  $x \geq 0$ ; *cases*  $y \geq 0$ )

(*auto simp: nat-le-iff le-nat-iff le-ceiling-iff floor-le-iff*)

**qed** *auto*

**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  $x y :: \text{real}$ 
defines  $A \equiv \text{sum-upto } a$ 
assumes  $\text{fin}: \text{finite } X$ 
assumes  $xy: 0 \leq y \ y < x$ 
assumes  $\text{deriv}: \bigwedge z. z \in \{y..x\} - X \implies (f \text{ has-vector-derivative } f' z) (at z)$ 
assumes  $\text{cont-f}: \text{continuous-on } \{y..x\} f$ 
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 -
  define  $\text{chi} :: \text{nat} \Rightarrow \text{real} \Rightarrow \text{real}$  where  $\text{chi} = (\lambda n t. \text{if } n \leq t \text{ then } 1 \text{ else } 0)$ 
  have  $((\lambda t. \text{sum-upto } (\lambda n. a n * (\text{chi } n t *_{\mathbb{R}} f' t)) x) \text{ has-integral}$ 
     $(\text{sum-upto } (\lambda n. a n * (f x - f (\text{max } n y))) x)) \{y..x\}$  (is  $(- \text{ has-integral}$ 
     $?I) -)$ 
  unfolding sum-upto-def
proof (intro has-integral-sum ballI finite-Nats-le-real, goal-cases)
  case  $(1 n)$ 
  have  $(f' \text{ has-integral } (f x - f (\text{max } n y))) \{\text{max } n y..x\}$ 
    using  $xy 1$ 
  by (intro fundamental-theorem-of-calculus-strong[OF fin])
    (auto intro!: continuous-on-subset[OF cont-f] deriv)
  also have  $?this \longleftrightarrow ((\lambda t. (\text{if } t \in \{\text{max } n y..x\} \text{ then } 1 \text{ else } 0) *_{\mathbb{R}} f' t)$ 
     $\text{ has-integral } (f x - f (\text{max } n y))) \{\text{max } n y..x\}$ 
  by (intro has-integral-cong) (simp-all add: chi-def)
  finally have  $((\lambda t. (\text{if } t \in \{\text{max } n y..x\} \text{ then } 1 \text{ else } 0) *_{\mathbb{R}} f' t)$ 
     $\text{ has-integral } (f x - f (\text{max } n y))) \{y..x\}$ 
  by (rule has-integral-on-superset) auto
  also have  $?this \longleftrightarrow ((\lambda t. \text{chi } n t *_{\mathbb{R}} f' t) \text{ has-integral } (f x - f (\text{max } n y)))$ 
 $\{y..x\}$ 
  by (intro has-integral-cong) (auto simp: chi-def)
  finally show  $?case$  by (intro has-integral-mult-right)
qed
  also have  $?this \longleftrightarrow ((\lambda t. A t * f' t) \text{ has-integral } ?I) \{y..x\}$ 
  unfolding sum-upto-def A-def chi-def sum-distrib-right using  $xy$ 
  by (intro has-integral-cong sum.mono-neutral-cong-right finite-Nats-le-real) auto
  also have  $\text{sum-upto } (\lambda n. a n * (f x - f (\text{max } (\text{real } n) y))) x =$ 
     $A x * f x - (\sum n \mid n > 0 \wedge \text{real } n \leq x. a n * f (\text{max } (\text{real } n) y))$ 
  by (simp add: sum-upto-def ring-distrib sum-subtractf sum-distrib-right A-def)
  also have  $\{n. n > 0 \wedge \text{real } n \leq x\} = \{n. n > 0 \wedge \text{real } n \leq y\} \cup \text{real} - ' \{y<..x\}$ 
    using  $xy$  by auto
  also have  $\text{sum } (\lambda n. a n * f (\text{max } (\text{real } n) y)) \dots =$ 
     $(\sum n \mid 0 < n \wedge \text{real } n \leq y. a n * f (\text{max } (\text{real } n) y)) +$ 
     $(\sum n \in \text{real} - ' \{y<..x\}. a n * f (\text{max } (\text{real } n) y))$  (is  $- = ?S1 + ?S2)$ 
  by (intro sum.union-disjoint finite-Nats-le-real finite-vimage-real-of-nat-greaterThanAtMost)

```

*auto*  
**also have**  $?S1 = \text{sum-upto } (\lambda n. a \ n * f \ y) \ y$  **unfolding** *sum-upto-def*  
**by** (*intro sum.cong refl*) (*auto simp: max-def*)  
**also have**  $\dots = A \ y * f \ y$  **by** (*simp add: A-def sum-upto-def sum-distrib-right*)  
**also have**  $?S2 = (\sum n \in \text{real} - \{y < ..x\}. a \ n * f \ n)$   
**by** (*intro sum.cong refl*) (*auto simp: max-def*)  
**finally show**  $?thesis$  **by** (*simp add: algebra-simps*)  
**qed**

**lemma** *partial-summation-integrable-strong*:  
 $(\lambda t. A \ t * f' \ t)$  *integrable-on*  $\{y..x\}$   
**and** *partial-summation-strong'*:  
 $(\sum n \in \text{real} - \{y < ..x\}. a \ n * f \ n) =$   
 $A \ x * f \ x - A \ y * f \ y - \text{integral } \{y..x\} (\lambda t. A \ t * f' \ t)$   
**using** *partial-summation-strong* **by** (*simp-all add: has-integral-iff algebra-simps*)

**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** *fin*: *finite*  $X$   
**assumes**  $x: x > 0$   
**assumes** *deriv*:  $\bigwedge z. z \in \{0..x\} - X \implies (f \text{ has-vector-derivative } f' \ z) \text{ (at } z)$   
**assumes** *cont-f*: *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** –  
**have**  $(\sum n \in \text{real} - \{0 < ..x\}. a \ n * f \ n) = \text{sum-upto } (\lambda n. a \ n * f \ n) \ x$   
**unfolding** *sum-upto-def* **by** (*intro sum.cong refl*) *auto*  
**thus**  $?thesis$   
**using** *partial-summation-strong*[*OF fin order.refl x deriv cont-f, of a*]  
**by** (*simp-all add: A-def*)  
**qed**

**lemma** *partial-summation-integrable-sum-upto-strong*:  
 $(\lambda t. A \ t * f' \ t)$  *integrable-on*  $\{0..x\}$   
**and** *partial-summation-sum-upto-strong'*:  
 $\text{sum-upto } (\lambda n. a \ n * f \ n) \ x =$   
 $A \ x * f \ x - \text{integral } \{0..x\} (\lambda t. A \ t * f' \ t)$   
**using** *partial-summation-sum-upto-strong* **by** (*simp-all add: has-integral-iff alge-*

*bra-simps*)

**end**

**end**

## 11 Euler product expansions

**theory** *Euler-Products*

**imports**

*HOL-Analysis.Analysis*

*Multiplicative-Function*

**begin**

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

**no-notation** *Infinite-Sum.abs-summable-on* (**infixr**  $\langle \text{abs}'\text{-summable}'\text{-on} \rangle$  46)

**lemma** *prime-factors-power-subset*:

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

**by** (*cases*  $n = 0$ ) (*auto simp: prime-factors-power*)

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

$(\lambda g. \prod_{p \in \{p. p \leq (n::\text{nat}) \wedge \text{prime } p\}}. p \wedge g p)$

$\in (\{p. p \leq n \wedge \text{prime } p\} \rightarrow_E \text{UNIV}) \rightarrow$   
 $\{m. 0 < m \wedge \text{prime-factors } m \subseteq \{..n\}\}$

**proof** (*safe, goal-cases*)

**case** (*2 f p*)

**have**  $\text{prime-factors } (\prod_{p \in \{p. p \leq n \wedge \text{prime } p\}}. p \wedge f p) =$

$(\bigcup_{p \in \{p. p \leq n \wedge \text{prime } p\}}. \text{prime-factors } (p \wedge f p))$

**by** (*subst prime-factors-prod*) *auto*

**also have**  $\dots \subseteq (\bigcup_{p \in \{p. p \leq n \wedge \text{prime } p\}}. \text{prime-factors } p)$

**using** *prime-factors-power-subset* **by** *blast*

**also have**  $\dots \subseteq (\bigcup_{p \in \{p. p \leq n \wedge \text{prime } p\}}. \{p\})$

**by** (*auto simp: prime-factors-dvd prime-gt-0-nat dest!: dvd-imp-le*)

**also have**  $\dots \subseteq \{..n\}$  **by** *auto*

**finally show** *?case* **using** *2* **by** *auto*

**qed** (*auto simp: prime-gt-0-nat*)

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

**proof** (*intro inj-onI, clarify, goal-cases*)

**case** (*1 p m q n*)

**with** *prime-power-eq-imp-eq*[*of p q m n*] **and** *1*

**have**  $p = q$  **by** *auto*

**moreover from** *this* **have**  $m = n$

**using** *prime-gt-1-nat 1* **by** *auto*

**ultimately show** *?case* **by** *simp*

**qed**

**lemma** *bij-betw-prime-powers*:

```

    bij-betw (λg. ∏ p∈{p. p ≤ n ∧ prime p}. p ^ g p) ({p. p ≤ n ∧ prime p} →E
UNIV)
    {m. 0 < m ∧ prime-factors m ⊆ {..(n::nat)}}
proof (rule bij-betwI[of - - - (λm p. if p ≤ n ∧ prime p then multiplicity p m else
undefined)],
    goal-cases)
  case 1
  show ?case by (rule prime-power-product-in-Pi)
next
  case 2
  show ?case
    by (auto split: if-splits)
next
  case (3 f)
  show ?case
  proof (rule ext, goal-cases)
    case (1 q)
    show ?case
    proof (cases q ≤ n ∧ prime q)
      case True
      hence multiplicity q (∏ p∈{p. p ≤ n ∧ prime p}. p ^ f p) =
        (∑ x∈{p. p ≤ n ∧ prime p}. multiplicity q (x ^ f x))
      by (subst prime-elem-multiplicity-prod-distrib) auto
      also have ... = (∑ x∈{p. p ≤ n ∧ prime p}. if x = q then f q else 0)
      using True by (intro sum.cong refl) (auto simp: multiplicity-distinct-prime-power)
      also have ... = f q using True by auto
      finally show ?thesis using True by simp
    qed (insert 3, force+)
  qed
next
  case (4 m)
  have (∏ p | p ≤ n ∧ prime p. p ^ (if p ≤ n ∧ prime p then multiplicity p m else
undefined)) =
    (∏ p∈prime-factors m. p ^ multiplicity p m)
  proof (rule prod.mono-neutral-cong)
    show finite (prime-factors m) by simp
  qed (insert 4, auto simp: prime-factors-multiplicity)
  also from 4 have ... = m
    by (intro prime-factorization-nat [symmetric]) auto
  finally show ?case .
qed

lemma
fixes f :: nat ⇒ 'a :: {real-normed-field,banach,second-countable-topology}
assumes summable: summable (λn. norm (f n))
assumes multiplicative-function f
shows abs-convergent-euler-product:
  abs-convergent-prod (λp. if prime p then ∑ n. f (p ^ n) 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** –

**interpret**  $f$ : multiplicative-function  $f$  **by fact**

**define**  $N$  **where**  $N = (\sum n. \text{norm } (f n))$

**have**  $\text{summable}'$ :  $f$  *abs-summable-on*  $A$  **for**  $A$

**by** (*rule abs-summable-on-subset*[*of - UNIV*])

(*insert summable, auto simp: abs-summable-on-nat-iff'*)

**have**  $\text{summable}''$ :  $(\lambda x. f (p \wedge x))$  *abs-summable-on*  $A$  **if prime**  $p$  **for**  $A$   $p$

**proof** (*subst abs-summable-on-reindex-iff*[*of - - f*])

**from**  $\langle \text{prime } p \rangle$  **have**  $p > 1$

**by** (*rule prime-gt-1-nat*)

**thus** *inj-on*  $(\lambda i. p \wedge i)$   $A$

**by** (*auto simp: inj-on-def*)

**qed** (*intro summable'*)

**have**  $(\lambda n. \text{norm } ((\sum m. f m) - (\prod p \in \{p. p \leq n \wedge \text{prime } p\}. \sum i. f (p \wedge i))))$

$\longrightarrow 0$

(*is filterlim ?h - -*)

**proof** (*rule tendsto-sandwich*)

**show** *eventually*  $(\lambda n. ?h n \leq N - (\sum m \leq n. \text{norm } (f m)))$  *at-top*

**proof** (*intro always-eventually allI*)

**fix**  $n :: \text{nat}$

**interpret** *product-sigma-finite*  $\lambda :: \text{nat. count-space } (UNIV :: \text{nat set})$

**by** (*intro product-sigma-finite.intro sigma-finite-measure-count-space*)

**have**  $(\prod p \mid p \leq n \wedge \text{prime } p. \sum i. f (p \wedge i)) =$

$(\prod p \mid p \leq n \wedge \text{prime } p. \sum_a i \in UNIV. f (p \wedge i))$

**by** (*intro prod.cong refl infsetsum-nat' [symmetric] summable''*) *auto*

**also have**  $\dots = (\sum_a g \in \{p. p \leq n \wedge \text{prime } p\} \rightarrow_E UNIV.$

$\prod x \in \{p. p \leq n \wedge \text{prime } p\}. f (x \wedge g x))$

**by** (*subst infsetsum-prod-PiE [symmetric]*)

(*auto simp: prime-gt-Suc-0-nat summable''*)

**also have**  $\dots = (\sum_a g \in \{p. p \leq n \wedge \text{prime } p\} \rightarrow_E UNIV.$

$f (\prod x \in \{p. p \leq n \wedge \text{prime } p\}. x \wedge g x))$

**by** (*subst f.prod-coprime*) (*auto simp add: primes-coprime*)

**also have**  $\dots = (\sum_a m \mid m > 0 \wedge \text{prime-factors } m \subseteq \{..n\}. f m)$

**by** (*intro infsetsum-reindex-bij-betw bij-betw-prime-powers*)

**also have**  $(\sum_a m \in UNIV. f m) - \dots = (\sum_a m \in UNIV - \{m. m > 0 \wedge$

*prime-factors } m \subseteq \{..n\}\}. f m)*

**by** (*intro infsetsum-Diff [symmetric] summable'*) *auto*

**also have**  $(\sum_a m \in UNIV. f m) = (\sum m. f m)$

**by** (*intro infsetsum-nat' summable'*)

**also have**  $UNIV - \{m. m > 0 \wedge \text{prime-factors } m \subseteq \{..n\}\} =$

$\text{insert } 0 \{m. \neg \text{prime-factors } m \subseteq \{..n\}\}$

**by** *auto*

**also have**  $(\sum_a m \in \dots. f m) = (\sum_a m \mid \neg \text{prime-factors } m \subseteq \{..n\}. f m)$

**by** (*intro infsetsum-cong-neutral*) *auto*

**also have**  $\text{norm } \dots \leq (\sum_a m \mid \neg \text{prime-factors } m \subseteq \{..n\}. \text{norm } (f m))$   
**by** (*rule norm-infsetsum-bound*)  
**also have**  $\dots \leq (\sum_a m \in \{n<..\}. \text{norm } (f m))$   
**proof** (*intro infsetsum-mono-neutral-left summable' abs-summable-on-normI*)  
**show**  $\{m. \neg \text{prime-factors } m \subseteq \{..n\}\} \subseteq \{n<..\}$   
**proof** *safe*  
**fix**  $m k$  **assume**  $\neg m > n$  **and**  $k \in \text{prime-factors } m$   
**thus**  $k \leq n$  **by** (*cases m = 0*) (*auto simp: prime-factors-dvd dest: dvd-imp-le*)  
**qed**  
**qed** *auto*  
**also have**  $\{n<..\} = \text{UNIV} - \{..n\}$   
**by** *auto*  
**also have**  $(\sum_a m \in \dots. \text{norm } (f m)) = (\sum_a m \in \text{UNIV}. \text{norm } (f m)) -$   
 $(\sum_a m \in \{..n\}. \text{norm } (f m))$   
**using** *summable* **by** (*intro infsetsum-Diff*) (*auto simp: abs-summable-on-nat-iff'*)  
**also have**  $(\sum_a m \in \text{UNIV}. \text{norm } (f m)) = N$   
**unfolding** *N-def* **using** *summable*  
**by** (*intro infsetsum-nat'*) (*auto simp: abs-summable-on-nat-iff'*)  
**also have**  $(\sum_a m \in \{..n\}. \text{norm } (f m)) = (\sum m \leq n. \text{norm } (f m))$   
**by** (*simp add: suminf-finite*)  
**finally show**  $?h n \leq N - (\sum m \leq n. \text{norm } (f m))$  .  
**qed**  
**next**  
**show** *eventually*  $(\lambda n. ?h n \geq 0)$  *at-top* **by** *simp*  
**next**  
**show**  $(\lambda n. N - (\sum m \leq n. \text{norm } (f m))) \longrightarrow 0$  **unfolding** *N-def*  
**by** (*rule tendsto-eq-intros refl summable-LIMSEQ' summable*)**+** *simp-all*  
**qed** *simp-all*  
**hence**  $(\lambda n. (\sum m. f m) - (\prod p \in \{p. p \leq n \wedge \text{prime } p\}. \sum i. f (p \wedge i))) \longrightarrow 0$   
**by** (*simp add: tendsto-norm-zero-iff*)  
**from** *tendsto-diff[OF tendsto-const[of  $\sum m. f m$ ] this]*  
**have**  $(\lambda n. \prod p \mid p \leq n \wedge \text{prime } p. \sum i. f (p \wedge i)) \longrightarrow (\sum m. f m)$  **by** *simp*  
**also have**  $(\lambda n. \prod p \mid p \leq n \wedge \text{prime } p. \sum i. f (p \wedge i)) =$   
 $(\lambda n. \prod p \leq n. \text{if prime } p \text{ then } (\sum i. f (p \wedge i)) \text{ else } 1)$   
**by** (*intro ext prod.mono-neutral-cong-left*) *auto*  
**finally show**  $\dots \longrightarrow (\sum m. f m)$  .  
  
**show** *abs-convergent-prod*  $(\lambda p. \text{if prime } p \text{ then } (\sum i. f (p \wedge i)) \text{ else } 1)$   
**proof** (*rule summable-imp-abs-convergent-prod*)  
**have**  $(\lambda(p,i). f (p \wedge i)) \text{abs-summable-on } \{p. \text{prime } p\} \times \{0<..\}$   
**unfolding** *case-prod-unfold*  
**by** (*subst abs-summable-on-reindex-iff[OF inj-prime-power]*) *fact*  
**hence**  $(\lambda p. \sum_a i \in \{0<..\}. f (p \wedge i)) \text{abs-summable-on } \{p. \text{prime } p\}$   
**by** (*rule abs-summable-on-Sigma-project1'*) *simp-all*  
**also have**  $?this \iff (\lambda p. (\sum i. f (p \wedge i)) - 1) \text{abs-summable-on } \{p. \text{prime } p\}$   
**proof** (*intro abs-summable-on-cong refl*)  
**fix**  $p :: \text{nat}$  **assume**  $p: p \in \{p. \text{prime } p\}$   
**have**  $\{0<..\} = \text{UNIV} - \{0::\text{nat}\}$  **by** *auto*  
**also have**  $(\sum_a i \in \dots. f (p \wedge i)) = (\sum i. f (p \wedge i)) - 1$

**using**  $p$  **by** (*subst infsetsum-Diff*) (*simp-all add: infsetsum-nat' summable''*)  
**finally show**  $(\sum_{a i \in \{0 < ..\}}. f (p \wedge i)) = (\sum i. f (p \wedge i)) - 1$  .  
**qed**  
**finally have** *summable*  $(\lambda p. \text{if prime } p \text{ then norm } ((\sum i. f (p \wedge i)) - 1) \text{ else } 0)$   
**(is summable ?T) by** (*simp add: abs-summable-on-nat-iff*)  
**also have**  $?T = (\lambda p. \text{norm } ((\text{if prime } p \text{ then } \sum i. f (p \wedge i) \text{ else } 1) - 1))$   
**by** (*rule ext*) (*simp add: if-splits*)  
**finally show** *summable* ... .  
**qed**  
**qed**

**lemma**

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

**proof** –

**interpret**  $f$ : *completely-multiplicative-function*  $f$  **by fact**

**{**  
**fix**  $p :: \text{nat}$  **assume** *prime*  $p$   
**hence** *inj*  $(\lambda i. p \wedge i)$   
**by** (*auto simp: inj-on-def dest: prime-gt-1-nat*)  
**from** *summable-reindex[OF summable this]*  
**have**  $*$ : *summable*  $(\lambda i. \text{norm } (f (p \wedge i)))$  **by** (*auto simp: o-def*)  
**also have**  $(\lambda i. \text{norm } (f (p \wedge i))) = (\lambda i. \text{norm } (f p) \wedge i)$   
**by** (*simp add: f.power norm-power*)  
**finally show**  $\text{norm } (f p) < 1$   
**by** (*subst (asm) summable-geometric-iff*) *simp-all*  
**note**  $*$  **and** *this*  
**} note** *summable' = this*

**have** *eq*:  $(\lambda p. \text{if prime } p \text{ then } (\sum i. f (p \wedge i)) \text{ else } 1) =$   
 $(\lambda p. \text{if prime } p \text{ then inverse } (1 - f p) \text{ else } 1)$

**proof** (*rule ext, goal-cases*)

**case**  $(1 p)$

**show**  $?case$

**proof** (*cases prime p*)

**case** *True*

**hence**  $\text{norm } (f p) < 1$  **by** (*rule summable'*)

**from** *suminf-geometric[OF this]* **and** *True* **show**  $?thesis$

**by** (*simp add: field-simps f.power*)

**qed** *simp-all*

```

qed
hence eq':  $(\lambda n. \prod_{p \leq n}. \text{if prime } p \text{ then } \sum n. f (p \wedge n) \text{ else } 1) =$ 
            $(\lambda n. \prod_{p \leq n}. \text{if prime } p \text{ then inverse } (1 - f p) \text{ else } 1)$ 
by (auto simp: fun-eq-iff)

have f: multiplicative-function f ..
from abs-convergent-euler-product[OF assms(1) f] and euler-product-LIMSEQ[OF
assms(1) f]
show abs-convergent-prod  $(\lambda p. \text{if prime } p \text{ then inverse } (1 - f p) \text{ else } 1)$ 
and  $(\lambda n. \prod_{p \leq n}. \text{if prime } p \text{ then inverse } (1 - f p) \text{ else } 1) \longrightarrow (\sum n. f n)$ 
by (simp-all only: eq eq')
qed

end

```

## 12 Analytic properties of Dirichlet series

**theory** *Dirichlet-Series-Analysis*

**imports**

*HOL-Complex-Analysis.Complex-Analysis*

*HOL-Library.Going-To-Filter*

*HOL-Real-Asymp.Real-Asymp*

*Dirichlet-Series*

*Moebius-Mu*

*Partial-Summation*

*Euler-Products*

**begin**

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

**no-notation** *Infinite-Sum.abs-summable-on* (**infixr**  $\langle \text{abs}'\text{-summable}'\text{-on} \rangle$  46)

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

**lemma** *frequently-going-toI*:

**assumes** *filterlim*  $(\lambda n. f (g n)) F G$

**assumes** *eventually*  $(\lambda n. P (g n)) G$

**assumes** *eventually*  $(\lambda n. g n \in A) G$

**assumes**  $G \neq \text{bot}$

**shows** *frequently*  $P$  (*f going-to F within A*)

**unfolding** *frequently-def*

**proof**

**assume** *eventually*  $(\lambda x. \neg P x)$  (*f going-to F within A*)

**hence** *eventually*  $(\lambda x. \neg P x)$  (*inf (filtercomap f F) (principal A)*)

**by** (*simp add: going-to-within-def*)

**moreover** **have** *filterlim*  $(\lambda n. g n)$  (*inf (filtercomap f F) (principal A)*)  $G$

**using** *assms* **unfolding** *filterlim-inf filterlim-principal*

**by** (*auto simp add: filterlim-iff-le-filtercomap filtercomap-filtercomap*)

**ultimately have** *eventually*  $(\lambda n. \neg P (g n)) G$   
**by** *(rule eventually-compose-filterlim)*  
**with** *assms(2)* **have** *eventually*  $(\lambda n. False) G$  **by** *eventually-elim auto*  
**with** *assms(4)* **show** *False* **by** *simp*  
**qed**

**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*  $P (f \text{filtercomap } f F)$   
**using** *frequently-going-toI[of f g F G P UNIV]* *assms* **by** *(simp add: going-to-def)*

**lemma** *frequently-going-to-at-topE*:  
**fixes**  $f :: 'a \Rightarrow \text{real}$   
**assumes** *frequently*  $P (f \text{going-to at-top})$   
**obtains**  $g$  **where**  $\bigwedge n. P (g n)$  **and** *filterlim*  $(\lambda n. f (g n))$  *at-top sequentially*  
**proof** –  
**from** *assms* **have**  $\forall k. \exists x. f x \geq \text{real } k \wedge P x$   
**by** *(auto simp: frequently-def eventually-going-to-at-top-linorder)*  
**hence**  $\exists g. \forall k. f (g k) \geq \text{real } k \wedge P (g k)$   
**by** *metis*  
**then obtain**  $g$  **where**  $g: \bigwedge k. f (g k) \geq \text{real } k \wedge P (g k)$   
**by** *blast*  
**have** *filterlim*  $(\lambda n. f (g n))$  *at-top sequentially*  
**by** *(rule filterlim-at-top-mono[OF filterlim-real-sequentially])* *(use g in auto)*  
**from**  $g(2)$  **and this** **show** *?thesis* **using** *that[of g]* **by** *blast*  
**qed**

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*  $P (Re \text{going-to at-top})$ . This lemma shows that our statement is equivalent to his.

**lemma** *frequently-going-to-at-top-iff*:  
*frequently*  $P (f \text{going-to } (at-top :: \text{real filter})) \iff$   
 $(\exists g. \forall n. P (g n) \wedge \text{filterlim } (\lambda n. f (g n)) \text{at-top sequentially})$   
**by** *(auto intro: frequently-going-toI elim!: frequently-going-to-at-topE)*

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

**proof** *(rule surjI)*  
**fix**  $x :: \text{real}$  **show**  $(x *_R 1) \cdot (1 :: 'a) = x$   
**by** *(simp add: dot-square-norm)*

**qed**

**lemma** *bullet-1-going-to-at-top-neq-bot [simp]*:  
 $((\lambda s::'a::\{\text{real-normed-algebra-1}, \text{real-inner}\}. s \cdot 1) \text{going-to at-top}) \neq \text{bot}$   
**unfolding** *going-to-def* **by** *(rule filtercomap-neq-bot-surj[OF - surj-bullet-1]) auto*

**lemma** *fds-abs-converges-altdef*:  
*fds-abs-converges*  $f\ s \longleftrightarrow (\lambda n. \text{fds-nth } f\ n / \text{nat-power } n\ s) \text{ abs-summable-on } \{1..\}$   
**by** (*auto simp add: fds-abs-converges-def abs-summable-on-nat-iff*  
*intro!: summable-cong eventually-mono[OF eventually-gt-at-top[of 0]]*)

**lemma** *fds-abs-converges-altdef'*:  
*fds-abs-converges*  $f\ s \longleftrightarrow (\lambda n. \text{fds-nth } f\ n / \text{nat-power } n\ s) \text{ abs-summable-on } UNIV$   
**by** (*subst fds-abs-converges-altdef, rule abs-summable-on-cong-neutral*) (*auto simp: Suc-le-eq*)

**lemma** *eval-fds-altdef*:  
**assumes** *fds-abs-converges*  $f\ s$   
**shows**  $\text{eval-fds } f\ s = (\sum a\ n. \text{fds-nth } f\ n / \text{nat-power } n\ s)$   
**proof** –  
**have** *fds-abs-converges*  $f\ s \longleftrightarrow (\lambda n. \text{fds-nth } f\ n / \text{nat-power } n\ s) \text{ abs-summable-on } UNIV$   
**unfolding** *fds-abs-converges-altdef*  
**by** (*intro abs-summable-on-cong-neutral*) (*auto simp: Suc-le-eq*)  
**with** *assms show ?thesis unfolding eval-fds-def fds-abs-converges-altdef*  
**by** (*intro infsetsum-nat' [symmetric]*) *simp-all*  
**qed**

**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**  
**interpret**  $f$ : *multiplicative-function*  $f$  **by fact**  
**show**  $f\ 0 / \text{nat-power } 0\ s = 0\ f\ 1 / \text{nat-power } 1\ s = 1$   
**by** *simp-all*  
**fix**  $a\ b :: \text{nat}$  **assume**  $a > 1\ b > 1$  *coprime*  $a\ b$   
**thus**  $f\ (a * b) / \text{nat-power } (a * b)\ s = f\ a / \text{nat-power } a\ s * (f\ b / \text{nat-power } b\ s)$   
**by** (*simp-all add: f.mult-coprime nat-power-mult-distrib*)  
**qed**

**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**  
**interpret**  $f$ : *completely-multiplicative-function*  $f$  **by fact**  
**show**  $f\ 0 / \text{nat-power } 0\ s = 0\ f\ (\text{Suc } 0) / \text{nat-power } (\text{Suc } 0)\ s = 1$   
**by** *simp-all*  
**fix**  $a\ b :: \text{nat}$  **assume**  $a > 1\ b > 1$   
**thus**  $f\ (a * b) / \text{nat-power } (a * b)\ s = f\ a / \text{nat-power } a\ s * (f\ b / \text{nat-power } b\ s)$

s)  
 by (simp-all add: f.mult nat-power-mult-distrib)  
 qed

## 12.1 Convergence and absolute convergence

**class** *nat-power-normed-field* = *nat-power-field* + *real-normed-field* + *real-inner*  
 + *real-algebra-1* +  
**fixes** *real-power* :: *real*  $\Rightarrow$  *'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*:  $d > 0 \implies \text{real-power } d \ (a + b) = \text{real-power } d \ a *_{\mathbb{R}} \text{real-power } d \ b$   
**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*:  
 $\bigwedge x::'a. n > 0 \implies \text{LIM } y \ \text{inf-class.inf}$   
 $(\text{Inf } (\text{principal } \{S. \text{open } S \wedge x \in S\})) (\text{principal } (\text{UNIV} - \{x\})).$   
 $(\text{nat-power } n \ y - \text{nat-power } n \ x - \ln (\text{real } n) *_{\mathbb{R}} \text{nat-power } n \ x * (y$   
 $- x)) /_{\mathbb{R}}$   
 $\text{norm } (y - x) :> \text{Inf } (\text{principal } \{S. \text{open } S \wedge 0 \in S\})$   
**assumes** *has-vector-derivative-real-power-aux*:  
 $x > 0 \implies \text{filterlim } (\lambda y. (\text{real-power } y \ c - \text{real-power } x \ (c :: 'a) -$   
 $(y - x) *_{\mathbb{R}} (c * \text{real-power } x \ (c - 1))) /_{\mathbb{R}}$   
 $\text{norm } (y - x)) (\text{INF } S \in \{S. \text{open } S \wedge 0 \in S\}. \text{principal } S) (\text{at } x)$   
**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*:  $d > 0 \implies \text{real-power } d \ (a - b) = \text{real-power } d \ a /_{\mathbb{R}} \text{real-power } d \ b$   
**using** *real-power-add*[of *d b a - b*] **by** (simp add: field-simps)

**end**

**lemma** *real-power-1-right* [simp]:  $d > 0 \implies \text{real-power } d \ 1 = \text{of-real } d$   
**using** *real-power-1-right-aux*[of *d*] **by** (simp add: scaleR-conv-of-real)

**lemma** *has-vector-derivative-real-power* [derivative-intros]:  
 $x > 0 \implies ((\lambda y. \text{real-power } y \ c) \ \text{has-vector-derivative } c * \text{real-power } x \ (c - 1))$   
 (at *x* within *A*)  
**by** (rule *has-vector-derivative-at-within*)  
 (insert *has-vector-derivative-real-power-aux*[of *x c*],  
 simp add: *has-vector-derivative-def* *has-derivative-def*  
*nhds-def* *bounded-linear-scaleR-left*)

**lemma** *has-field-derivative-nat-power* [derivative-intros]:

$n > 0 \implies ((\lambda y. \text{nat-power } n \ y) \text{ has-field-derivative } \ln (\text{real } n) *_R \text{ nat-power } n \ x)$   
 (at  $(x :: 'a :: \text{nat-power-normed-field})$  within  $A$ )  
**by** (rule *has-field-derivative-at-within*)  
 (insert *has-field-derivative-nat-power-aux*[of  $n \ x$ ],  
*simp only: has-field-derivative-def has-derivative-def netlimit-at*,  
*simp add: nhds-def at-within-def bounded-linear-mult-right*)

**lemma** *continuous-on-real-power* [*continuous-intros*]:  
 $A \subseteq \{0 < ..\} \implies \text{continuous-on } A \ (\lambda x. \text{real-power } x \ s)$   
**by** (rule *continuous-on-vector-derivative has-vector-derivative-real-power*) $+$  *auto*

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

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

**instance proof** (*standard, goal-cases*)  
**case** ( $7 \ n \ x$ )  
**hence**  $((\lambda x. \text{nat-power } n \ x) \text{ has-field-derivative } \ln (\text{real } n) *_R \text{ nat-power } n \ x)$  (at  $x$ )  
**by** (*auto intro!: derivative-eq-intros simp: powr-def*)  
**thus**  $?case$  **unfolding** *has-field-derivative-def netlimit-at has-derivative-def*  
**by** (*simp add: nhds-def at-within-def*)  
**next**  
**case** ( $8 \ x \ c$ )  
**hence**  $((\lambda y. \text{real-power } y \ c) \text{ has-vector-derivative } c *_R \text{ real-power } x \ (c - 1))$  (at  $x$ )  
**by** (*auto intro!: derivative-eq-intros*  
*simp: has-real-derivative-iff-has-vector-derivative [symmetric]*)  
**thus**  $?case$  **by** (*simp add: has-vector-derivative-def has-derivative-def nhds-def*)  
**qed** (*simp-all add: powr-add*)

**end**

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

**definition** *nat-power-complex* :: *nat*  $\Rightarrow$  *complex*  $\Rightarrow$  *complex* **where**  
 [*simp*]: *nat-power-complex*  $n \ z = \text{of-nat } n \ \text{powr } z$

**definition** *real-power-complex* :: *real*  $\Rightarrow$  *complex*  $\Rightarrow$  *complex* **where**  
 [*simp*]: *real-power-complex* =  $(\lambda x \ y. \text{of-real } x \ \text{powr } y)$

**instance proof**  
**fix**  $m \ n :: \text{nat}$  **and**  $z :: \text{complex}$   
**assume**  $m > 0 \ n > 0$   
**thus**  $\text{nat-power } (m * n) \ z = \text{nat-power } m \ z *_R \text{ nat-power } n \ z$

```

  unfolding nat-power-complex-def of-nat-mult by (subst powr-times-real) simp-all
next
  fix n :: nat and z :: complex
  assume n > 0
  show norm (nat-power n z) = real n powr (z · 1) unfolding nat-power-complex-def
    using norm-powr-real-powr[of of-nat n z] by simp
next
  fix n :: nat and x :: complex assume n: n > 0
  hence ((λx. nat-power n x) has-field-derivative ln (real n) *R nat-power n x) (at
x)
  by (auto intro!: derivative-eq-intros simp: powr-def scaleR-conv-of-real mult-ac)
  thus LIM y inf-class.inf (Inf (principal ‘ {S. open S ∧ x ∈ S})) (principal (UNIV
– {x})).
    (nat-power n y – nat-power n x – ln (real n) *R nat-power n x * (y –
x)) /R
    cmod (y – x) :> (Inf (principal ‘ {S. open S ∧ 0 ∈ S}))
  unfolding has-field-derivative-def netlimit-at has-derivative-def
  by (simp add: nhds-def at-within-def)
next
  fix x :: real and c :: complex assume x > 0
  hence ((λy. real-power y c) has-vector-derivative c * real-power x (c – 1)) (at
x)
  by (auto intro!: derivative-eq-intros has-vector-derivative-real-field)
  thus LIM y at x. (real-power y c – real-power x c – (y – x) *R (c * real-power
x (c – 1))) /R
    norm (y – x) :> INF S∈{S. open S ∧ 0 ∈ S}. principal S
  by (simp add: has-vector-derivative-def has-derivative-def nhds-def)
next
  fix n :: nat and x :: real
  show nat-power n (x *R 1 :: complex) = (real n powr x) *R 1
  by (simp add: powr-Reals-eq scaleR-conv-of-real)
qed (auto simp: powr-def exp-add exp-of-nat-mult [symmetric] algebra-simps scaleR-conv-of-real
simp del: Ln-of-nat)

```

end

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

```

  nat-power n (of-real x :: 'a :: nat-power-normed-field) = of-real (real n powr x)
  using nat-power-of-real-aux[of n x] by (simp add: scaleR-conv-of-real)

```

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

```

  fds-abs-converges (fds-of-real f)
  (of-real s :: 'a :: {nat-power-normed-field,banach}) ⟷ fds-abs-converges f s
  unfolding fds-abs-converges-def
  by (subst (1 2) summable-Suc-iff [symmetric]) (simp add: norm-divide norm-nat-power)

```

**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)*  
**using** *assms unfolding eval-fds-def* **by** *(auto simp: fds-converges-def suminf-of-real)*

**lemma** *fds-abs-summable-zeta-iff* [*simp*]:  
**fixes**  $s :: 'a :: \{\text{banach, nat-power-normed-field}\}$   
**shows**  $\text{fds-abs-converges fds-zeta } s \longleftrightarrow s \cdot 1 > (1 :: \text{real})$   
**proof** –  
**have**  $\text{fds-abs-converges fds-zeta } s \longleftrightarrow \text{summable } (\lambda n. \text{real } n \text{ powr } -(s \cdot 1))$   
**unfolding** *fds-abs-converges-def*  
**by** *(intro summable-cong always-eventually)*  
*(auto simp: norm-divide fds-nth-zeta powr-minus norm-nat-power divide-simps)*  
**also have**  $\dots \longleftrightarrow s \cdot 1 > 1$  **by** *(simp add: summable-real-powr-iff)*  
**finally show** *?thesis* .  
**qed**

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

**lemma** *fds-abs-converges-moebius-mu*:  
**fixes**  $s :: 'a :: \{\text{banach, nat-power-normed-field}\}$   
**assumes**  $s \cdot 1 > 1$   
**shows**  $\text{fds-abs-converges (fds moebius-mu) } s$   
**unfolding** *fds-abs-converges-def*  
**proof** *(rule summable-comparison-test, intro exI allI impI)*  
**fix**  $n :: \text{nat}$   
**show**  $\text{norm (norm (fds-nth (fds moebius-mu) } n / \text{nat-power } n s)) \leq \text{real } n \text{ powr } (-s \cdot 1)$   
**by** *(auto simp: powr-minus divide-simps abs-moebius-mu-le norm-nat-power norm-divide moebius-mu-def norm-power)*  
**next**  
**from** *assms* **show**  $\text{summable } (\lambda n. \text{real } n \text{ powr } (-s \cdot 1))$  **by** *(simp add: summable-real-powr-iff)*  
**qed**

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

**definition** *abs-conv-abscissa*  
 $:: 'a :: \{\text{nat-power, banach, real-normed-field, real-inner}\} \text{fds} \Rightarrow \text{ereal}$  **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 conv-abcissa  $f \leq$  conv-abcissa  $g$ 
unfolding conv-abcissa-def by (rule INF-mono) (use assms in auto)

lemma abs-conv-abcissa-mono:
assumes  $\bigwedge s. \text{fds-abs-converges } g \ s \implies \text{fds-abs-converges } f \ s$ 
shows abs-conv-abcissa  $f \leq$  abs-conv-abcissa  $g$ 
unfolding abs-conv-abcissa-def by (rule INF-mono) (use assms in auto)

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

instance real :: dirichlet-series by standard simp-all
instance complex :: dirichlet-series by standard (simp-all add: Basis-complex-def)

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  $\text{fds-abs-converges } f \ z'$ 
unfolding fds-abs-converges-def
proof (rule summable-comparison-test, intro exI allI impI)
fix  $n :: \text{nat}$  assume  $n: n \geq 1$ 
thus  $\text{norm} (\text{norm} (\text{fds-nth } f \ n / \text{nat-power } n \ z')) \leq \text{norm} (\text{fds-nth } f \ n / \text{nat-power } n \ z)$ 
using assms(2) by (simp add: norm-divide norm-nat-power divide-simps powr-mono mult-left-mono)
qed (insert assms(1), simp add: fds-abs-converges-def)

lemma fds-abs-converges:
assumes  $s \cdot 1 > \text{abs-conv-abcissa } (f :: 'a \text{ fds})$ 
shows  $\text{fds-abs-converges } f \ s$ 
proof –
from assms obtain  $s0$  where  $\text{fds-abs-converges } f \ s0 \ s0 \cdot 1 < s \cdot 1$ 
by (auto simp: INF-less-iff abs-conv-abcissa-def)
with  $\text{fds-abs-converges-Re-le}$ [OF this(1), of  $s$ ] this(2) show ?thesis by simp
qed

lemma fds-abs-diverges:
assumes  $s \cdot 1 < \text{abs-conv-abcissa } (f :: 'a \text{ fds})$ 
shows  $\neg \text{fds-abs-converges } f \ s$ 
proof
assume  $\text{fds-abs-converges } f \ s$ 
hence  $\text{abs-conv-abcissa } f \leq s \cdot 1$  unfolding abs-conv-abcissa-def
by (intro INF-lower) auto

```

**with** *assms* **show** *False* **by** *simp*  
**qed**

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

**fixes** *s0* :: 'a :: *dirichlet-series*

**assumes** *bounded*: *Bseq* ( $\lambda n. \sum_{k \leq n}. \text{fds-nth } f \ k / \text{nat-power } k \ s0$ )

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

**shows** *uniformly-Cauchy-on* *B* ( $\lambda N z. \sum_{n \leq N}. \text{fds-nth } f \ n / \text{nat-power } n \ z$ )

**proof** (*cases* *B* =  $\{\}$ )

**case** *False*

**show** *?thesis*

**proof** (*rule* *uniformly-Cauchy-onI'*, *goal-cases*)

**case** (*1*  $\varepsilon$ )

**define**  $\sigma$  **where**  $\sigma = \text{Inf } ((\lambda s. s \cdot 1) \text{ ` } B)$

**have**  $\sigma$ -*le*:  $s \cdot 1 \geq \sigma$  **if**  $s \in B$  **for**  $s$

**unfolding**  $\sigma$ -*def* **using** *that*

**by** (*intro* *cInf-lower* *bounded-inner-imp-bdd-below* *compact-imp-bounded* *B*)

*auto*

**have**  $\sigma \in ((\lambda s. s \cdot 1) \text{ ` } B)$

**unfolding**  $\sigma$ -*def* **using**  $B \neq \{\}$

**by** (*intro* *closed-contains-Inf* *bounded-inner-imp-bdd-below* *compact-imp-bounded*

*B*

*compact-imp-closed* *compact-continuous-image* *continuous-intros*) *auto*

**with** *B*( $\emptyset$ ) **have**  $\sigma$ -*gt*:  $\sigma > s0 \cdot 1$  **by** *auto*

**define**  $\delta$  **where**  $\delta = \sigma - s0 \cdot 1$

**have** *bounded* *B* **by** (*rule* *compact-imp-bounded*) *fact*

**then obtain** *norm-B-aux* **where** *norm-B-aux*:  $\bigwedge s. s \in B \implies \text{norm } s \leq$

*norm-B-aux*

**by** (*auto* *simp*: *bounded-iff*)

**define** *norm-B* **where** *norm-B* = *norm-B-aux* + *norm* *s0*

**from** *norm-B-aux* **have** *norm-B*:  $\text{norm } (s - s0) \leq \text{norm-B}$  **if**  $s \in B$  **for**  $s$

**using** *norm-triangle-ineq4*[*of*  $s \ s0$ ] *norm-B-aux*[*OF* *that*] **by** (*simp* *add*:

*norm-B-def*)

**then have**  $0 \leq \text{norm-B}$

**by** (*meson*  $\langle \sigma \in (\lambda s. s \cdot 1) \text{ ` } B \rangle$  *imageE* *norm-ge-zero* *order.trans*)

**define** *A* **where** *A* = *sum-upto* ( $\lambda k. \text{fds-nth } f \ k / \text{nat-power } k \ s0$ )

**from** *bounded* **obtain** *C-aux* **where** *C-aux*:  $\bigwedge n. \text{norm } (\sum_{k \leq n}. \text{fds-nth } f \ k /$

*nat-power } k \ s0) \leq C*-*aux*

**by** (*auto* *simp*: *Bseq-def*)

**define** *C* **where** *C* = *max* *C-aux* *1*

**have** *C-pos*:  $C > 0$  **by** (*simp* *add*: *C-def*)

**have** *C*:  $\text{norm } (A \ x) \leq C$  **for**  $x$

**proof** -

**have**  $A \ x = (\sum_{k \leq \text{nat } \lfloor x \rfloor}. \text{fds-nth } f \ k / \text{nat-power } k \ s0)$

**unfolding** *A-def* *sum-upto-altdef* **by** (*intro* *sum.mono-neutral-left*) *auto*

**also have**  $\text{norm } \dots \leq C$ -*aux* **by** (*rule* *C-aux*)

**also have**  $\dots \leq C$  **by** (*simp* *add*: *C-def*)

**finally show** *?thesis* .  
**qed**

**have**  $(\lambda m. 2 * C * (1 + \text{norm-}B / \delta) * \text{real } m \text{ powr } (-\delta)) \longrightarrow 0$  **unfolding**  
 $\delta$ -def **using**  $\sigma$ -gt  
**by** (intro tendsto-mult-right-zero tendsto-neg-powr filterlim-real-sequentially)  
simp-all  
**from** order-tendstoD(2)[OF this  $\langle \varepsilon > 0 \rangle$ ] **obtain**  $M$  **where**  
 $M: \bigwedge m. m \geq M \implies 2 * C * (1 + \text{norm-}B / \delta) * \text{real } m \text{ powr } -\delta < \varepsilon$   
**by** (auto simp: eventually-at-top-linorder)

**show** *?case*  
**proof** (intro exI[of - max M 1] ballI allI impI, goal-cases)  
**case** (1 s m n)  
**from** 1 **have**  $s \cdot 1 > s0 \cdot 1$  **using** B(2)[of s] **by** simp  
**have**  $mn: m \geq M \ m < n \ m > 0 \ n > 0$  **using** 1 **by** (simp-all add: )  
**have**  $\text{dist } (\sum_{n \leq m}. \text{fds-nth } f \ n / \text{nat-power } n \ s) (\sum_{n \leq n}. \text{fds-nth } f \ n / \text{nat-power } n \ s) =$   
 $\text{dist } (\sum_{n \leq n}. \text{fds-nth } f \ n / \text{nat-power } n \ s) (\sum_{n \leq m}. \text{fds-nth } f \ n / \text{nat-power } n \ s)$   
**by** (simp add: dist-commute)  
**also from** 1 **have**  $\dots = \text{norm } (\sum_{k \in \{..n\} - \{..m\}}. \text{fds-nth } f \ k / \text{nat-power } k \ s)$   
**by** (subst Groups-Big.sum-diff) (simp-all add: dist-norm)  
**also from** 1 **have**  $\{..n\} - \{..m\} = \text{real } - \{ \text{real } m < .. \text{real } n \}$  **by** auto  
**also have**  $(\sum_{k \in \dots} \text{fds-nth } f \ k / \text{nat-power } k \ s) =$   
 $(\sum_{k \in \dots} \text{fds-nth } f \ k / \text{nat-power } k \ s0 * \text{real-power } (\text{real } k) (s0 - s))$   
(is - = ?S) **by** (intro sum.cong refl) (simp-all add: nat-power-diff real-power-nat-power)  
**also have**  $*$ :  $((\lambda t. A \ t * ((s0 - s) * \text{real-power } t (s0 - s - 1))) \text{has-integral } (A (\text{real } n) * \text{real-power } n (s0 - s) - A (\text{real } m) * \text{real-power } m (s0 - s) - ?S))$   
 $\{ \text{real } m .. \text{real } n \}$  (is (?h has-integral -) -) **unfolding** A-def **using**  
mn  
**by** (intro partial-summation-strong[of {}])  
(auto intro!: derivative-eq-intros continuous-intros)  
**hence**  $?S = A (\text{real } n) * \text{nat-power } n (s0 - s) - A (\text{real } m) * \text{nat-power } m (s0 - s) -$   
 $\text{integral } \{ \text{real } m .. \text{real } n \} \ ?h$   
**using** mn **by** (simp add: has-integral-iff real-power-nat-power)  
**also have**  $\text{norm } \dots \leq \text{norm } (A (\text{real } n) * \text{nat-power } n (s0 - s)) +$   
 $\text{norm } (A (\text{real } m) * \text{nat-power } m (s0 - s)) + \text{norm } (\text{integral } \{ \text{real } m .. \text{real } n \} \ ?h)$   
**by** (intro order.trans[OF norm-triangle-ineq4] add-right-mono order.refl)  
**also have**  $\text{norm } (A (\text{real } n) * \text{nat-power } n (s0 - s)) \leq C * \text{nat-power } m ((s0 - s) \cdot 1)$   
**using** mn  $\langle s \in B \rangle$  C-pos s  
**by** (auto simp: norm-mult norm-nat-power algebra-simps intro!: mult-mono C powr-mono2')

```

also have norm (A (real m) * nat-power m (s0 - s)) ≤ C * nat-power m
((s0 - s) · 1)
using mn by (auto simp: norm-mult norm-nat-power intro!: mult-mono C)
also have norm (integral {real m..real n} ?h) ≤
integral {real m..real n} (λt. C * (norm (s0 - s) * t powr ((s0 -
s) · 1 - 1)))
proof (intro integral-norm-bound-integral ballI, goal-cases)
case 1
with * show ?case by (simp add: has-integral-iff)
next
case 2
from mn show ?case by (auto intro!: integrable-continuous-real continu-
ous-intros)
next
case (3 t)
thus ?case unfolding norm-mult using C-pos mn
by (intro mult-mono C) (auto simp: norm-real-power dot-square-norm
algebra-simps)
qed
also have ... = C * norm (s0 - s) * integral {real m..real n} (λt. t powr
((s0 - s) · 1 - 1))
by (simp add: algebra-simps dot-square-norm)
also {
have ((λt. t powr ((s0 - s) · 1 - 1)) has-integral
(real n powr ((s0 - s) · 1) / ((s0 - s) · 1) -
real m powr ((s0 - s) · 1) / ((s0 - s) · 1))) {m..n}
(is (?l has-integral ?I) -) using mn s
by (intro fundamental-theorem-of-calculus)
(auto intro!: derivative-eq-intros
simp: has-real-derivative-iff-has-vector-derivative [symmetric]
inner-diff-left)
hence integral {real m..real n} ?l = ?I by (simp add: has-integral-iff)
also have ... ≤ -(real m powr ((s0 - s) · 1) / ((s0 - s) · 1)) using s mn
by (simp add: divide-simps inner-diff-left)
also have ... = 1 * (real m powr ((s0 - s) · 1) / ((s - s0) · 1))
using s by (simp add: field-simps inner-diff-left)
also have ... ≤ 2 * (real m powr ((s0 - s) · 1) / ((s - s0) · 1)) using
mn s
by (intro mult-right-mono divide-nonneg-pos) (simp-all add: inner-diff-left)
finally have integral {m..n} ?l ≤ ... .
}
hence C * norm (s0 - s) * integral {real m..real n} (λt. t powr ((s0 - s) ·
1 - 1)) ≤
C * norm (s0 - s) * (2 * (real m powr ((s0 - s) · 1) / ((s - s0) ·
1)))
using C-pos mn
by (intro mult-mono mult-nonneg-nonneg integral-nonneg
integrable-continuous-real continuous-intros) auto
also have C * nat-power m ((s0 - s) · 1) + C * nat-power m ((s0 - s) ·

```

$1) + \dots =$   
 $2 * C * \text{nat-power } m ((s0 - s) \cdot 1) * (1 + \text{norm } (s - s0) / ((s - s0) \cdot 1))$   
**by** (*simp add: algebra-simps norm-minus-commute*)  
**also have**  $\dots \leq 2 * C * \text{nat-power } m (-\delta) * (1 + \text{norm-}B / \delta)$   
**using** *C-pos s mn  $\sigma$ -le[*of s*]  $\langle s \in B \rangle \sigma$ -gt  $\langle 0 \leq \text{norm-}B \rangle$*   
**unfolding** *nat-power-real-def  $\delta$ -def*  
**by** (*intro mult-mono powr-mono frac-le add-mono norm- $B$ ; simp add: inner-diff-left*)  
**also have**  $\dots = 2 * C * (1 + \text{norm-}B / \delta) * \text{real } m \text{ powr } (-\delta)$  **by** *simp*  
**also from**  $\langle m \geq M \rangle$  **have**  $\dots < \varepsilon$  **by** (*rule M*)  
**finally show** *?case by - simp-all*  
**qed**  
**qed**  
**qed** (*auto simp: uniformly-Cauchy-on-def*)

**lemma** *uniformly-convergent-eval-fds-aux*:  
**assumes** *Bseq*  $(\lambda n. \sum_{k \leq n}. \text{fds-nth } f \ k / \text{nat-power } k \ (s0 :: 'a))$   
**assumes** *B*: *compact B*  $\bigwedge z. z \in B \implies z \cdot 1 > s0 \cdot 1$   
**shows** *uniformly-convergent-on B*  $(\lambda N z. \sum_{n \leq N}. \text{fds-nth } f \ n / \text{nat-power } n \ z)$   
**by** (*rule Cauchy-uniformly-convergent uniformly-Cauchy-eval-fds-aux assms*)**+**

**lemma** *uniformly-convergent-eval-fds-aux'*:  
**assumes** *conv*: *fds-converges f*  $(s0 :: 'a)$   
**assumes** *B*: *compact B*  $\bigwedge z. z \in B \implies z \cdot 1 > s0 \cdot 1$   
**shows** *uniformly-convergent-on B*  $(\lambda N z. \sum_{n \leq N}. \text{fds-nth } f \ n / \text{nat-power } n \ z)$   
**proof** (*rule uniformly-convergent-eval-fds-aux*)  
**from** *conv* **have** *convergent*  $(\lambda n. \sum_{k \leq n}. \text{fds-nth } f \ k / \text{nat-power } k \ s0)$   
**by** (*simp add: fds-converges-def summable-iff-convergent'*)  
**thus** *Bseq*  $(\lambda n. \sum_{k \leq n}. \text{fds-nth } f \ k / \text{nat-power } k \ s0)$  **by** (*rule convergent-imp-Bseq*)  
**qed** (*insert assms, auto*)

**lemma** *bounded-partial-sums-imp-fps-converges*:  
**fixes** *s0*  $:: 'a :: \text{dirichlet-series}$   
**assumes** *Bseq*  $(\lambda n. \sum_{k \leq n}. \text{fds-nth } f \ k / \text{nat-power } k \ s0)$  **and**  $s \cdot 1 > s0 \cdot 1$   
**shows** *fds-converges f s*  
**proof** -  
**have** *uniformly-convergent-on*  $\{s\}$   $(\lambda N z. \sum_{n \leq N}. \text{fds-nth } f \ n / \text{nat-power } n \ z)$   
**using** *assms(2)*  
**by** (*intro uniformly-convergent-eval-fds-aux[OF assms(1)] auto*)  
**thus** *?thesis*  
**by** (*auto simp: fds-converges-def summable-iff-convergent'*  
*dest: uniformly-convergent-imp-convergent*)  
**qed**

**theorem** *fds-converges-Re-le*:  
**assumes** *fds-converges f*  $(s0 :: 'a)$   $s \cdot 1 > s0 \cdot 1$   
**shows** *fds-converges f s*  
**proof** -

**have** *uniformly-convergent-on*  $\{s\}$   $(\lambda N z. \sum_{n \leq N}. \text{fds-nth } f \ n / \text{nat-power } n \ z)$   
**by**  $(\text{rule } \text{uniformly-convergent-eval-fds-aux}' \text{ assms}) + (\text{insert } \text{assms}(2), \text{auto})$   
**then obtain**  $l$  **where** *uniform-limit*  $\{s\}$   $(\lambda N z. \sum_{n \leq N}. \text{fds-nth } f \ n / \text{nat-power } n \ z)$   $l$  *at-top*  
**by**  $(\text{auto } \text{simp: } \text{uniformly-convergent-on-def})$   
**from** *tendsto-uniform-limitI* $[OF \ \text{this}, \text{of } s]$   
**have**  $(\lambda n. \text{fds-nth } f \ n / \text{nat-power } n \ s)$  *sums*  $l \ s$  **unfolding** *sums-def'*  
**by**  $(\text{simp } \text{add: } \text{atLeast0AtMost})$   
**thus** *?thesis* **by**  $(\text{simp } \text{add: } \text{fds-converges-def } \text{sums-iff})$   
**qed**

**lemma** *fds-converges*:

**assumes**  $s \cdot 1 > \text{conv-abscissa } (f :: 'a \ \text{fds})$   
**shows** *fds-converges*  $f \ s$   
**proof** –  
**from** *assms* **obtain**  $s0$  **where** *fds-converges*  $f \ s0$   $s0 \cdot 1 < s \cdot 1$   
**by**  $(\text{auto } \text{simp: } \text{INF-less-iff } \text{conv-abscissa-def})$   
**with** *fds-converges-Re-le* $[OF \ \text{this}(1), \text{of } s]$  *this*(2) **show** *?thesis* **by** *simp*  
**qed**

**lemma** *fds-diverges*:

**assumes**  $s \cdot 1 < \text{conv-abscissa } (f :: 'a \ \text{fds})$   
**shows**  $\neg \text{fds-converges } f \ s$   
**proof**  
**assume** *fds-converges*  $f \ s$   
**hence**  $\text{conv-abscissa } f \leq s \cdot 1$  **unfolding** *conv-abscissa-def*  
**by**  $(\text{intro } \text{INF-lower}) \text{ auto}$   
**with** *assms* **show** *False* **by** *simp*  
**qed**

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

**assumes** *fds-converges*  $(f :: 'a \ \text{fds})$   $s \ s' \cdot 1 > s \cdot 1 + 1$   
**shows** *fds-abs-converges*  $f \ s'$   
**unfolding** *fds-abs-converges-def*  
**proof**  $(\text{rule } \text{summable-comparison-test-ev})$   
**from** *assms*(2) **show** *summable*  $(\lambda n. \text{real } n \ \text{powr } ((s - s') \cdot 1))$   
**by**  $(\text{subst } \text{summable-real-powr-iff}) (\text{simp-all } \text{add: } \text{inner-diff-left})$   
**next**  
**from** *assms*(1) **have**  $(\lambda n. \text{fds-nth } f \ n / \text{nat-power } n \ s) \longrightarrow 0$   
**unfolding** *fds-converges-def* **by**  $(\text{rule } \text{summable-LIMSEQ-zero})$   
**from** *tendsto-norm* $[OF \ \text{this}]$  **have**  $(\lambda n. \text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ s))$   
 $\longrightarrow 0$  **by** *simp*  
**hence** *eventually*  $(\lambda n. \text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ s) < 1)$  *at-top*  
**by**  $(\text{rule } \text{order-tendstoD}) \text{ simp-all}$   
**thus** *eventually*  $(\lambda n. \text{norm } (\text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ s')) \leq$   
 $\text{real } n \ \text{powr } ((s - s') \cdot 1))$  *at-top*  
**proof** *eventually-elim*  
**case**  $(\text{elim } n)$   
**thus** *?case*

```

proof (cases n = 0)
  case False
  have norm (fds-nth f n / nat-power n s') =
    norm (fds-nth f n) / real n powr (s' · 1) using False
  by (simp add: norm-divide norm-nat-power)
  also have ... = norm (fds-nth f n / nat-power n s) / real n powr ((s' - s) ·
1) using False
  by (simp add: norm-divide norm-nat-power inner-diff-left powr-diff)
  also have ... ≤ 1 / real n powr ((s' - s) · 1) using elim
  by (intro divide-right-mono elim) simp-all
  also have ... = real n powr ((s - s') · 1) using False
  by (simp add: field-simps inner-diff-left powr-diff)
  finally show ?thesis by simp
qed simp-all
qed
qed

```

```

lemma conv-le-abs-conv-abscissa: conv-abscissa f ≤ abs-conv-abscissa f
unfolding conv-abscissa-def abs-conv-abscissa-def
by (intro INF-superset-mono) auto

```

```

lemma conv-abscissa-PInf-iff: conv-abscissa f = ∞ ↔ (∀ s. ¬fds-converges f s)
unfolding conv-abscissa-def by (subst Inf-eq-PInfTy) auto

```

```

lemma conv-abscissa-PInfI [intro]: (∧ s. ¬fds-converges f s) ⇒ conv-abscissa f =
∞
by (subst conv-abscissa-PInf-iff) auto

```

```

lemma conv-abscissa-MInf-iff: conv-abscissa (f :: 'a fds) = -∞ ↔ (∀ s. fds-converges
f s)

```

```

proof safe
  assume *: ∀ s. fds-converges f s
  have conv-abscissa f ≤ B for B :: real
  using spec[OF *, of of-real B] fds-diverges[of of-real B f]
  by (cases conv-abscissa f ≤ B) simp-all
  thus conv-abscissa f = -∞ by (rule ereal-bot)
qed (auto intro: fds-converges)

```

```

lemma conv-abscissa-MInfI [intro]: (∧ s. fds-converges (f :: 'a fds) s) ⇒ conv-abscissa
f = -∞
by (subst conv-abscissa-MInf-iff) auto

```

```

lemma abs-conv-abscissa-PInf-iff: abs-conv-abscissa f = ∞ ↔ (∀ s. ¬fds-abs-converges
f s)
unfolding abs-conv-abscissa-def by (subst Inf-eq-PInfTy) auto

```

```

lemma abs-conv-abscissa-PInfI [intro]: (∧ s. ¬fds-converges f s) ⇒ abs-conv-abscissa
f = ∞
by (subst abs-conv-abscissa-PInf-iff) auto

```

**lemma** *abs-conv-abscissa-MInf-iff*:  
 $abs-conv-abscissa (f :: 'a fds) = -\infty \longleftrightarrow (\forall s. fds-abs-converges f s)$   
**proof** *safe*  
**assume** \*:  $\forall s. fds-abs-converges f s$   
**have**  $abs-conv-abscissa f \leq B$  **for**  $B :: real$   
**using** *spec[OF \*, of of-real B] fds-abs-diverges[of of-real B f]*  
**by** (*cases abs-conv-abscissa f ≤ B simp-all*)  
**thus**  $abs-conv-abscissa f = -\infty$  **by** (*rule ereal-bot*)  
**qed** (*auto intro: fds-abs-converges*)

**lemma** *abs-conv-abscissa-MInfI [intro]*:  
 $(\bigwedge s. fds-abs-converges (f :: 'a fds) s) \implies abs-conv-abscissa f = -\infty$   
**by** (*subst abs-conv-abscissa-MInf-iff auto*)

**lemma** *conv-abscissa-geI*:  
**assumes**  $\bigwedge c'. ereal c' < c \implies \exists s. s \cdot 1 = c' \wedge \neg fds-converges f s$   
**shows**  $conv-abscissa (f :: 'a fds) \geq c$   
**proof** (*rule ccontr*)  
**assume**  $\neg conv-abscissa f \geq c$   
**hence**  $c > conv-abscissa f$  **by** *simp*  
**from** *ereal-dense2[OF this] obtain c' where c > ereal c' c' > conv-abscissa f*  
**by** *auto*  
**moreover from** *assms[OF this(1)] obtain s where s \cdot 1 = c' \neg fds-converges f s*  
**by** *blast*  
**ultimately show** *False using fds-converges[of f s] by auto*  
**qed**

**lemma** *conv-abscissa-leI*:  
**assumes**  $\bigwedge c'. ereal c' > c \implies \exists s. s \cdot 1 = c' \wedge fds-converges f s$   
**shows**  $conv-abscissa (f :: 'a fds) \leq c$   
**proof** (*rule ccontr*)  
**assume**  $\neg conv-abscissa f \leq c$   
**hence**  $c < conv-abscissa f$  **by** *simp*  
**from** *ereal-dense2[OF this] obtain c' where c < ereal c' c' < conv-abscissa f*  
**by** *auto*  
**moreover from** *assms[OF this(1)] obtain s where s \cdot 1 = c' fds-converges f s*  
**by** *blast*  
**ultimately show** *False using fds-diverges[of s f] by auto*  
**qed**

**lemma** *abs-conv-abscissa-geI*:  
**assumes**  $\bigwedge c'. ereal c' < c \implies \exists s. s \cdot 1 = c' \wedge \neg fds-abs-converges f s$   
**shows**  $abs-conv-abscissa (f :: 'a fds) \geq c$   
**proof** (*rule ccontr*)  
**assume**  $\neg abs-conv-abscissa f \geq c$   
**hence**  $c > abs-conv-abscissa f$  **by** *simp*  
**from** *ereal-dense2[OF this] obtain c' where c > ereal c' c' > abs-conv-abscissa f*  
**by** *auto*

moreover from *assms*[*OF this*(1)] obtain *s* where  $s \cdot 1 = c' \neg \text{fds-abs-converges } f \ s$  by *blast*  
ultimately show *False* using *fds-abs-converges*[*of f s*] by *auto*  
qed

**lemma** *abs-conv-abscissa-leI*:

assumes  $\bigwedge c'. \text{ereal } c' > c \implies \exists s. s \cdot 1 = c' \wedge \text{fds-abs-converges } f \ s$   
shows  $\text{abs-conv-abscissa } (f :: 'a \text{ fds}) \leq c$

**proof** (rule *ccontr*)

assume  $\neg \text{abs-conv-abscissa } f \leq c$

hence  $c < \text{abs-conv-abscissa } f$  by *simp*

from *ereal-dense2*[*OF this*] obtain *c'* where  $c < \text{ereal } c' \ c' < \text{abs-conv-abscissa } f$  by *auto*

moreover from *assms*[*OF this*(1)] obtain *s* where  $s \cdot 1 = c' \text{fds-abs-converges } f \ s$  by *blast*

ultimately show *False* using *fds-abs-diverges*[*of s f*] by *auto*  
qed

**lemma** *conv-abscissa-leI-weak*:

assumes  $\bigwedge x. \text{ereal } x > d \implies \text{fds-converges } f \ (\text{of-real } x)$

shows  $\text{conv-abscissa } (f :: 'a \text{ fds}) \leq d$

**proof** (rule *conv-abscissa-leI*)

fix *x* assume  $d < \text{ereal } x$

from *assms*[*OF this*] show  $\exists s. s \cdot 1 = x \wedge \text{fds-converges } f \ s$

by (*intro exI*[*of - of-real x*]) *auto*

qed

**lemma** *abs-conv-abscissa-leI-weak*:

assumes  $\bigwedge x. \text{ereal } x > d \implies \text{fds-abs-converges } f \ (\text{of-real } x)$

shows  $\text{abs-conv-abscissa } (f :: 'a \text{ fds}) \leq d$

**proof** (rule *abs-conv-abscissa-leI*)

fix *x* assume  $d < \text{ereal } x$

from *assms*[*OF this*] show  $\exists s. s \cdot 1 = x \wedge \text{fds-abs-converges } f \ s$

by (*intro exI*[*of - of-real x*]) *auto*

qed

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

$\text{conv-abscissa } (\text{fds-truncate } m \ (f :: 'a \text{ fds})) = -\infty$

by (*auto simp: conv-abscissa-MInf-iff*)

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

$\text{abs-conv-abscissa } (\text{fds-truncate } m \ (f :: 'a \text{ fds})) = -\infty$

by (*auto simp: abs-conv-abscissa-MInf-iff*)

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

**proof** (rule *abs-conv-abscissa-leI*)

fix *c* assume *less: conv-abscissa f + 1 < eréal c*

**define**  $c'$  **where**  $c' = (\text{if } \text{conv-abscissa } f = -\infty \text{ then } c - 2$   
 $\quad \text{else } (c - 1 + \text{real-of-ereal } (\text{conv-abscissa } f)) / 2)$   
**from**  $\text{less}$  **have**  $c'$ :  $\text{conv-abscissa } f < \text{ereal } c' \wedge c' < c - 1$   
**by** ( $\text{cases } \text{conv-abscissa } f$ ) ( $\text{simp-all add: } c'\text{-def field-simps}$ )

**from**  $c'$  **have**  $\text{fds-converges } f$  ( $\text{of-real } c'$ )  
**by** ( $\text{intro } \text{fds-converges}$ ) ( $\text{simp-all add: } \text{inner-diff-left dot-square-norm}$ )  
**hence**  $\text{fds-abs-converges } f$  ( $\text{of-real } c$ )  
**by** ( $\text{rule } \text{fds-converges-imp-abs-converges}$ ) ( $\text{insert } c', \text{ simp-all}$ )  
**thus**  $\exists s. s \cdot 1 = c \wedge \text{fds-abs-converges } f s$   
**by** ( $\text{intro } \text{exI}[\text{of - of-real } c]$ )  $\text{auto}$   
**qed**

**lemma**  $\text{uniformly-convergent-eval-fds}$ :  
**assumes**  $B$ :  $\text{compact } B \wedge z. z \in B \implies z \cdot 1 > \text{conv-abscissa } (f :: 'a \text{ fds})$   
**shows**  $\text{uniformly-convergent-on } B (\lambda N z. \sum n \leq N. \text{fds-nth } f n / \text{nat-power } n z)$   
**proof** ( $\text{cases } B = \{\}$ )  
**case**  $\text{False}$   
**define**  $\sigma$  **where**  $\sigma = \text{Inf } ((\lambda s. s \cdot 1) \text{ ` } B)$   
**have**  $\sigma\text{-le}$ :  $s \cdot 1 \geq \sigma$  **if**  $s \in B$  **for**  $s$   
**unfolding**  $\sigma\text{-def}$  **using**  $\text{that}$   
**by** ( $\text{intro } \text{cInf-lower bounded-inner-imp-bdd-below compact-imp-bounded } B$ )  $\text{auto}$   
**have**  $\sigma \in ((\lambda s. s \cdot 1) \text{ ` } B)$   
**unfolding**  $\sigma\text{-def}$  **using**  $B \langle B \neq \{\} \rangle$   
**by** ( $\text{intro } \text{closed-contains-Inf bounded-inner-imp-bdd-below compact-imp-bounded } B$   
 $\text{compact-imp-closed compact-continuous-image continuous-intros}$ )  $\text{auto}$   
**with**  $B(2)$  **have**  $\sigma\text{-gt}$ :  $\sigma > \text{conv-abscissa } f$  **by**  $\text{auto}$   
**define**  $s$  **where**  $s = (\text{if } \text{conv-abscissa } f = -\infty \text{ then } \sigma - 1 \text{ else}$   
 $\quad (\sigma + \text{real-of-ereal } (\text{conv-abscissa } f)) / 2)$   
**from**  $\sigma\text{-gt}$  **have**  $s$ :  $\text{conv-abscissa } f < s \wedge s < \sigma$   
**by** ( $\text{cases } \text{conv-abscissa } f$ ) ( $\text{auto simp: } s\text{-def}$ )  
**show**  $?thesis$  **using**  $s \langle \text{compact } B \rangle$   
**by** ( $\text{intro } \text{uniformly-convergent-eval-fds-aux}'[\text{of } f \text{ of-real } s] \text{ fds-converges}$ )  
 $(\text{auto dest: } \sigma\text{-le})$   
**qed**  $\text{auto}$

**corollary**  $\text{uniformly-convergent-eval-fds}'$ :  
**assumes**  $B$ :  $\text{compact } B \wedge z. z \in B \implies z \cdot 1 > \text{conv-abscissa } (f :: 'a \text{ fds})$   
**shows**  $\text{uniformly-convergent-on } B (\lambda N z. \sum n < N. \text{fds-nth } f n / \text{nat-power } n z)$   
**proof** –  
**from**  $\text{uniformly-convergent-eval-fds}[OF \text{ assms}]$  **obtain**  $l$  **where**  
 $\text{uniform-limit } B (\lambda N z. \sum n \leq N. \text{fds-nth } f n / \text{nat-power } n z) l \text{ at-top}$   
**by** ( $\text{auto simp: } \text{uniformly-convergent-on-def}$ )  
**also** **have**  $(\lambda N z. \sum n \leq N. \text{fds-nth } f n / \text{nat-power } n z) =$   
 $(\lambda N z. \sum n < \text{Suc } N. \text{fds-nth } f n / \text{nat-power } n z)$   
**by** ( $\text{simp only: } \text{lessThan-Suc-atMost}$ )  
**finally** **have**  $\text{uniform-limit } B (\lambda N z. \sum n < N. \text{fds-nth } f n / \text{nat-power } n z) l$

at-top

**unfolding** *uniform-limit-iff* **by** (*subst (asm) eventually-sequentially-Suc*)  
**thus** *?thesis* **by** (*auto simp: uniformly-convergent-on-def*)  
**qed**

## 12.2 Derivative of a Dirichlet series

**lemma** *fds-converges-deriv-aux*:

**assumes** *conv: fds-converges f (s0 :: 'a)* **and** *gt: s · 1 > s0 · 1*

**shows** *fds-converges (fds-deriv f) s*

**proof** –

**have** *Cauchy* ( $\lambda n. \sum k \leq n. (-\ln (\text{real } k) *_{\mathbb{R}} \text{fds-nth } f k) / \text{nat-power } k s$ )

**proof** (*rule CauchyI'*, *goal-cases*)

**case** ( $1 \ \varepsilon$ )

**define**  $\delta$  **where**  $\delta = s \cdot 1 - s0 \cdot 1$

**define**  $\delta'$  **where**  $\delta' = \delta / 2$

**from** *gt* **have**  $\delta\text{-pos}$ :  $\delta > 0$  **by** (*simp add: \delta-def*)

**define** *A* **where**  $A = \text{sum-upto } (\lambda k. \text{fds-nth } f k / \text{nat-power } k s0)$

**from** *conv* **have** *convergent* ( $\lambda n. \sum k \leq n. \text{fds-nth } f k / \text{nat-power } k s0$ )

**by** (*simp add: fds-converges-def summable-iff-convergent'*)

**hence** *Bseq* ( $\lambda n. \sum k \leq n. \text{fds-nth } f k / \text{nat-power } k s0$ ) **by** (*rule convergent-imp-Bseq*)

**then obtain** *C-aux* **where** *C-aux*:  $\bigwedge n. \text{norm } (\sum k \leq n. \text{fds-nth } f k / \text{nat-power } k s0) \leq C\text{-aux}$

**by** (*auto simp: Bseq-def*)

**define** *C* **where**  $C = \max C\text{-aux } 1$

**have** *C-pos*:  $C > 0$  **by** (*simp add: C-def*)

**have** *C*:  $\text{norm } (A x) \leq C$  **for** *x*

**proof** –

**have**  $A x = (\sum k \leq \text{nat } \lfloor x \rfloor. \text{fds-nth } f k / \text{nat-power } k s0)$

**unfolding** *A-def sum-upto-altdef* **by** (*intro sum.mono-neutral-left*) *auto*

**also have**  $\text{norm } \dots \leq C\text{-aux}$  **by** (*rule C-aux*)

**also have**  $\dots \leq C$  **by** (*simp add: C-def*)

**finally show** *?thesis* .

**qed**

**define** *C'* **where**  $C' = 2 * C + C * (\text{norm } (s0 - s) * (1 + 1 / \delta) + 1) / \delta$

**have** ( $\lambda m. C' * \text{real } m \text{ powr } (-\delta')$ )  $\longrightarrow 0$  **unfolding**  $\delta'\text{-def}$  **using** *gt*  $\delta\text{-pos}$

**by** (*intro tendsto-mult-right-zero tendsto-neg-powr filterlim-real-sequentially*)

*simp-all*

**from** *order-tendstoD(2)[OF this \(\varepsilon > 0\)]* **obtain** *M1* **where**

*M1*:  $\bigwedge m. m \geq M1 \implies C' * \text{real } m \text{ powr } -\delta' < \varepsilon$

**by** (*auto simp: eventually-at-top-linorder*)

**have** ( $(\lambda x. \ln (\text{real } x) / \text{real } x \text{ powr } \delta') \longrightarrow 0$ ) *at-top* **using**  $\delta\text{-pos}$

**by** (*intro lim-ln-over-power*) (*simp-all add: \delta'-def*)

**from** *order-tendstoD(2)[OF this zero-less-one]* *eventually-gt-at-top[of 1::nat]*

**have** *eventually* ( $\lambda n. \ln (\text{real } n) \leq n \text{ powr } \delta'$ ) *at-top* **by** *eventually-elim*  
*simp-all*

**then obtain** *M2* **where** *M2*:  $\bigwedge n. n \geq M2 \implies \ln (\text{real } n) \leq n \text{ powr } \delta'$

```

    by (auto simp: eventually-at-top-linorder)
  let ?f' =  $\lambda k. -\ln (\text{real } k) *_R \text{fds-nth } f \ k$ 

  show ?case
  proof (intro exI[of - max (max M1 M2) 1] allI impI, goal-cases)
    case (1 m n)
    hence mn:  $m \geq M1 \ m \geq M2 \ m > 0 \ m < n$  by simp-all
    define g ::  $\text{real} \Rightarrow 'a$  where  $g = (\lambda t. \text{real-power } t \ (s0 - s) * \text{of-real } (\ln t))$ 
    define g' ::  $\text{real} \Rightarrow 'a$ 
      where  $g' = (\lambda t. \text{real-power } t \ (s0 - s - 1) * ((s0 - s) * \text{of-real } (\ln t) + 1))$ 
    define norm-g' ::  $\text{real} \Rightarrow \text{real}$ 
      where  $\text{norm-g}' = (\lambda t. t \ \text{powr } (-\delta - 1) * (\text{norm } (s0 - s) * \ln t + 1))$ 
    define norm-g ::  $\text{real} \Rightarrow \text{real}$ 
      where  $\text{norm-g} = (\lambda t. -(t \ \text{powr } -\delta) * (\text{norm } (s0 - s) * (\delta * \ln t + 1) + \delta) / \delta^{\wedge}2)$ 
    have g-g': ( $g$  has-vector-derivative  $g'$   $t$ ) (at  $t$ ) if  $t \in \{\text{real } m.. \text{real } n\}$  for  $t$ 
      using mn that by (auto simp: g-def g'-def real-power-diff field-simps
    real-power-add
      intro!: derivative-eq-intros)
    have [continuous-intros]: continuous-on  $\{\text{real } m.. \text{real } n\}$   $g$  using mn
      by (auto simp: g-def intro!: continuous-intros)

    let ?S =  $\sum_{k \in \text{real} - \{ \text{real } m <.. \text{real } n \}. \text{fds-nth } f \ k / \text{nat-power } k \ s0 * g \ k}$ 
    have dist ( $\sum_{k \leq m. ?f' \ k / \text{nat-power } k \ s}$ ) ( $\sum_{k \leq n. ?f' \ k / \text{nat-power } k \ s} =$ 
      norm ( $\sum_{k \in \{..n\} - \{..m\}. \text{fds-nth } f \ k / \text{nat-power } k \ s * \text{of-real } (\ln (\text{real } k))$ 
    k)))
      using mn by (subst sum-diff)
      (simp-all add: dist-norm norm-minus-commute sum-negf scaleR-conv-of-real
    mult-ac)
    also have  $\{..n\} - \{..m\} = \text{real} - \{ \text{real } m <.. \text{real } n \}$  by auto
    also have ( $\sum_{k \in \dots} \text{fds-nth } f \ k / \text{nat-power } k \ s * \text{of-real } (\ln (\text{real } k))$ ) =
      ( $\sum_{k \in \dots} \text{fds-nth } f \ k / \text{nat-power } k \ s0 * g \ k$ ) using mn unfolding g-def
      by (intro sum.cong refl) (auto simp: real-power-nat-power field-simps
    nat-power-diff)
    also have *: (( $\lambda t. A \ t * g' \ t$ ) has-integral
      ( $A \ (\text{real } n) * g \ n - A \ (\text{real } m) * g \ m - ?S$ ))
      { $\text{real } m.. \text{real } n$ } (is (?h has-integral -) -) unfolding A-def using
    mn
      by (intro partial-summation-strong[of {}])
      (auto intro!: g-g' simp: field-simps continuous-intros)
    hence ?S =  $A \ (\text{real } n) * g \ n - A \ (\text{real } m) * g \ m - \text{integral } \{\text{real } m.. \text{real } n\}$ 
    ?h
      using mn by (simp add: has-integral-iff field-simps)
    also have norm ...  $\leq$  norm ( $A \ (\text{real } n) * g \ n$ ) + norm ( $A \ (\text{real } m) * g \ m$ )
    +
      norm (integral  $\{\text{real } m.. \text{real } n\}$  ?h)
      by (intro order.trans[OF norm-triangle-ineq4] add-right-mono order.refl)
    also have norm ( $A \ (\text{real } n) * g \ n$ )  $\leq$   $C * \text{norm } (g \ n)$ 
      unfolding norm-mult using mn C-pos by (intro mult-mono C) auto

```

**also have**  $\text{norm } (g \ n) \leq n \ \text{powr } -\delta * n \ \text{powr } \delta'$  **using**  $mn \ M2[\text{of } n]$   
**by** (*simp add: g-def norm-real-power norm-mult  $\delta$ -def inner-diff-left*)  
**also have**  $\dots = n \ \text{powr } -\delta'$  **using**  $mn$   
**by** (*simp add:  $\delta'$ -def powr-minus field-simps powr-add [symmetric]*)  
**also have**  $\text{norm } (A \ (\text{real } m) * g \ m) \leq C * \text{norm } (g \ m)$   
**unfolding**  $\text{norm-mult}$  **using**  $mn \ C\text{-pos}$  **by** (*intro mult-mono C*) **auto**  
**also have**  $\text{norm } (g \ m) \leq m \ \text{powr } -\delta * m \ \text{powr } \delta'$  **using**  $mn \ M2[\text{of } m]$   
**by** (*simp add: g-def norm-real-power norm-mult  $\delta$ -def inner-diff-left*)  
**also have**  $\dots = m \ \text{powr } -\delta'$  **using**  $mn$   
**by** (*simp add:  $\delta'$ -def powr-minus field-simps powr-add [symmetric]*)  
**also have**  $C * \text{real } n \ \text{powr } -\delta' \leq C * \text{real } m \ \text{powr } -\delta'$  **using**  $\delta\text{-pos } mn$   
*C-pos*  
**by** (*intro mult-left-mono powr-mono2'*) (*simp-all add:  $\delta'$ -def*)  
**also have**  $\dots + \dots = 2 * \dots$  **by** *simp*  
**also have**  $\text{norm } (\text{integral } \{m..n\} \ ?h) \leq \text{integral } \{m..n\} \ (\lambda t. C * \text{norm-g}' \ t)$   
**proof** (*intro integral-norm-bound-integral ballI, goal-cases*)  
**case** 1  
**with**  $*$  **show**  $\ ?case$  **by** (*simp add: has-integral-iff*)  
**next**  
**case** 2  
**from**  $mn$  **show**  $\ ?case$   
**by** (*auto intro!: integrable-continuous-real continuous-intros simp: norm-g'-def*)  
**next**  
**case** (3  $t$ )  
**have**  $\text{norm } (g' \ t) \leq \text{norm-g}' \ t$  **unfolding**  $g'\text{-def } \text{norm-g}'\text{-def}$  **using** 3  $mn$   
**unfolding**  $\text{norm-mult}$   
**by** (*intro mult-mono order.trans[OF norm-triangle-ineq]*)  
*(auto simp: norm-real-power inner-diff-left dot-square-norm norm-mult*  
 *$\delta$ -def*  
*intro!: mult-left-mono)*  
**thus**  $\ ?case$  **unfolding**  $\text{norm-mult}$  **using**  $C\text{-pos } mn$   
**by** (*intro mult-mono C*) *simp-all*  
**qed**  
**also have**  $\dots = C * \text{integral } \{m..n\} \ \text{norm-g}'$   
**unfolding**  $\text{norm-g}'\text{-def}$  **by** (*simp add: norm-g'-def  $\delta$ -def inner-diff-left*)  
**also** {  
**have** ( $\text{norm-g}' \ \text{has-integral } (\text{norm-g } n - \text{norm-g } m) \ \{m..n\}$ )  
**unfolding**  $\text{norm-g}'\text{-def } \text{norm-g}\text{-def } \text{power2-eq-square}$  **using**  $mn \ \delta\text{-pos}$   
**by** (*intro fundamental-theorem-of-calculus*)  
*(auto simp: has-real-derivative-iff-has-vector-derivative [symmetric]*  
*field-simps powr-diff intro!: derivative-eq-intros)*  
**hence**  $\text{integral } \{m..n\} \ \text{norm-g}' = \text{norm-g } n - \text{norm-g } m$  **by** (*simp add:*  
*has-integral-iff*)  
**also have**  $\text{norm-g } n \leq 0$  **unfolding**  $\text{norm-g}\text{-def}$  **using**  $\delta\text{-pos } mn$   
**by** (*intro divide-nonpos-pos mult-nonpos-nonneg add-nonneg-nonneg*  
*mult-nonneg-nonneg*)  
*simp-all*  
**hence**  $\text{norm-g } n - \text{norm-g } m \leq -\text{norm-g } m$  **by** *simp*  
**also have**  $\dots = \text{real } m \ \text{powr } -\delta * \ln (\text{real } m) * (\text{norm } (s0 - s)) / \delta +$

```

      real m powr  $-\delta * ((\text{norm } (s0 - s) / \delta + 1) / \delta)$  using  $\delta$ -pos
    by (simp add: field-simps norm-g-def power2-eq-square)
  also {
    have  $\ln (\text{real } m) \leq \text{real } m \text{ powr } \delta'$  using  $M2[\text{of } m]$  mn by simp
    also have  $\text{real } m \text{ powr } -\delta * \dots = \text{real } m \text{ powr } -\delta'$ 
      by (simp add: powr-add [symmetric]  $\delta'$ -def)
    finally have  $\text{real } m \text{ powr } -\delta * \ln (\text{real } m) * (\text{norm } (s0 - s)) / \delta \leq$ 
       $\dots * (\text{norm } (s0 - s)) / \delta$  using  $\delta$ -pos
  }
  by (intro divide-right-mono mult-right-mono) (simp-all add: mult-left-mono)
}
also have  $\text{real } m \text{ powr } -\delta * ((\text{norm } (s0 - s) / \delta + 1) / \delta) \leq$ 
   $\text{real } m \text{ powr } -\delta' * ((\text{norm } (s0 - s) / \delta + 1) / \delta)$  using mn  $\delta$ -pos
  by (intro mult-right-mono powr-mono) (simp-all add:  $\delta'$ -def)
also have  $\text{real } m \text{ powr } -\delta' * \text{norm } (s0 - s) / \delta + \dots =$ 
   $\text{real } m \text{ powr } -\delta' * (\text{norm } (s0 - s) * (1 + 1 / \delta) + 1) / \delta$  using
 $\delta$ -pos
  by (simp add: field-simps power2-eq-square)
finally have  $\text{integral } \{\text{real } m.. \text{real } n\} \text{ norm-g}' \leq$ 
   $\text{real } m \text{ powr } -\delta' * (\text{norm } (s0 - s) * (1 + 1 / \delta) + 1) / \delta$  by
- simp-all
}
also have  $2 * (C * m \text{ powr } -\delta') + C * (m \text{ powr } -\delta' * (\text{norm } (s0 - s) * (1 + 1 / \delta) + 1) / \delta) =$ 
   $C' * m \text{ powr } -\delta'$  by (simp add: algebra-simps  $C'$ -def)
also have  $\dots < \varepsilon$  using  $M1[\text{of } m]$  mn by simp
finally show ?case using  $C$ -pos by - simp-all
qed
qed
from Cauchy-convergent[OF this]
show ?thesis by (simp add: summable-iff-convergent' fds-converges-def fds-nth-deriv)
qed

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  $\text{has-field-derivative-eval-fds } [\text{derivative-intros}]:$ 
     $(\text{eval-fds } f \text{ has-field-derivative } \text{eval-fds } (\text{fds-deriv } f) s)$  (at  $s$  within  $A$ )

proof -
  define  $s1 :: \text{real}$  where
     $s1 = (\text{if } \text{conv-abscissa } f = -\infty \text{ then } s \cdot 1 - 2 \text{ else } (s \cdot 1 * 1 / 3 + \text{real-of-ereal } (\text{conv-abscissa } f) * 2 / 3))$ 
  define  $s2 :: \text{real}$  where
     $s2 = (\text{if } \text{conv-abscissa } f = -\infty \text{ then } s \cdot 1 - 1 \text{ else } (s \cdot 1 * 2 / 3 + \text{real-of-ereal } (\text{conv-abscissa } f) * 1 / 3))$ 
  from assms have  $s: \text{conv-abscissa } f < s1 \wedge s1 < s2 \wedge s2 < s \cdot 1$ 
  by (cases conv-abscissa f) (auto simp: s1-def s2-def field-simps)
  from s have  $*$ :  $\text{fds-converges } f$  (of-real s1) by (intro fds-converges) simp-all
  thus  $\text{conv}'$ :  $\text{fds-converges } (\text{fds-deriv } f) s$ 
  by (rule fds-converges-deriv-aux) (insert s, simp-all)

```

```

from * have conv: fds-converges (fds-deriv f) (of-real s2)
  by (rule fds-converges-deriv-aux) (insert s, simp-all)

define  $\delta :: \text{real}$  where  $\delta = (s \cdot 1 - s^2) / 2$ 
from s have  $\delta\text{-pos}$ :  $\delta > 0$  by (simp add:  $\delta\text{-def}$ )

have uniformly-convergent-on (cball s  $\delta$ )
  ( $\lambda n s. \sum k \leq n. \text{fds-nth } (\text{fds-deriv } f) k / \text{nat-power } k s$ )
proof (intro uniformly-convergent-eval-fds-aux'[OF conv])
  fix  $s'' :: 'a$  assume  $s'' : s'' \in \text{cball } s \delta$ 
  have  $\text{dist } (s \cdot 1) (s'' \cdot 1) \leq \text{dist } s s''$ 
    by (intro Euclidean-dist-upper) (simp-all add: one-in-Basis)
  also from  $s''$  have  $\dots \leq \delta$  by simp
  finally show  $s'' \cdot 1 > (\text{of-real } s^2 :: 'a) \cdot 1$  using s
    by (auto simp:  $\delta\text{-def}$  dist-real-def abs-if split: if-splits)
qed (insert  $\delta\text{-pos}$ , auto)
then obtain l where
  uniform-limit (cball s  $\delta$ ) ( $\lambda n s. \sum k \leq n. \text{fds-nth } (\text{fds-deriv } f) k / \text{nat-power } k$ 
s) l at-top
  by (auto simp: uniformly-convergent-on-def)
  also have ( $\lambda n s. \sum k \leq n. \text{fds-nth } (\text{fds-deriv } f) k / \text{nat-power } k s$ ) =
    ( $\lambda n s. \sum k < \text{Suc } n. \text{fds-nth } (\text{fds-deriv } f) k / \text{nat-power } k s$ )
  by (simp only: lessThan-Suc-atMost)
  finally have uniform-limit (cball s  $\delta$ ) ( $\lambda n s. \sum k < n. \text{fds-nth } (\text{fds-deriv } f) k /$ 
nat-power k s)
    l at-top
  unfolding uniform-limit-iff by (subst (asm) eventually-sequentially-Suc)
hence *: uniformly-convergent-on (cball s  $\delta$ )
  ( $\lambda n s. \sum k < n. \text{fds-nth } (\text{fds-deriv } f) k / \text{nat-power } k s$ )
  unfolding uniformly-convergent-on-def by blast

have (eval-fds f has-field-derivative eval-fds (fds-deriv f) s) (at s)
  unfolding eval-fds-def
proof (rule has-field-derivative-series'(2)[OF - - *])
  show  $s \in \text{cball } s \delta$   $s \in \text{interior } (\text{cball } s \delta)$  using s by (simp-all add:  $\delta\text{-def}$ )
  show summable ( $\lambda n. \text{fds-nth } f n / \text{nat-power } n s$ )
    using assms fds-converges[of f s] by (simp add: fds-converges-def)
next
  fix  $s' :: 'a$  and  $n :: \text{nat}$ 
  show ( $(\lambda s. \text{fds-nth } f n / \text{nat-power } n s)$  has-field-derivative
     $\text{fds-nth } (\text{fds-deriv } f) n / \text{nat-power } n s'$ ) (at  $s'$  within cball s  $\delta$ )
    by (cases  $n = 0$ )
    (simp, auto intro!: derivative-eq-intros simp: fds-nth-deriv field-simps)
qed (auto simp: fds-nth-deriv intro!: derivative-eq-intros)
thus (eval-fds f has-field-derivative eval-fds (fds-deriv f) s) (at s within A)
  by (rule has-field-derivative-at-within)
qed

lemmas has-field-derivative-eval-fds' [derivative-intros] =

```

*DERIV-chain2[OF has-field-derivative-eval-fds]*

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

**assumes**  $s \cdot 1 > \text{conv-abscissa } f$

**shows** *continuous (at s within A) (eval-fds (f :: 'a :: dirichlet-series fds))*

**proof** –

**have** *isCont (eval-fds f) s*

**by** (*rule has-field-derivative-eval-fds DERIV-isCont assms*)+

**thus** *?thesis* **by** (*rule continuous-within-subset*) *auto*

**qed**

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

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

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

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

**by** (*rule continuous-within-compose3[OF - assms(1)] continuous-intros assms*)+

**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)*

**by** (*rule DERIV-continuous-on derivative-intros*) + (*insert assms, auto*)

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

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

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

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

**by** (*rule continuous-on-compose2[OF continuous-on-eval-fds assms(1)]*)  
(*insert assms, auto simp: image-iff*)

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

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

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

**proof** (*rule conv-abscissa-leI*)

**fix**  $c' :: \text{real}$

**assume**  $\text{ereal } c' > \text{conv-abscissa } f$

**thus**  $\exists s. s \cdot 1 = c' \wedge \text{fds-converges } (f\text{ds-deriv } f) s$

**by** (*intro exI[of - of-real c']*) (*auto simp: fds-converges-deriv*)

**qed**

**lemma** *abs-conv-abscissa-integral*:

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

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

**proof** (*rule antisym*)

**show**  $\text{abs-conv-abscissa } (f\text{ds-integral } a \ f) \leq \text{abs-conv-abscissa } f$

**proof** (*rule abs-conv-abscissa-leI, goal-cases*)

**case** (1 c)

**have**  $\text{fds-abs-converges } (f\text{ds-integral } a \ f) \text{ (of-real } c)$

**unfolding** *fds-abs-converges-def*

```

proof (rule summable-comparison-test-ev)
  from 1 have fds-abs-converges f (of-real c)
    by (intro fds-abs-converges) auto
  thus summable ( $\lambda n. \text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ (\text{of-real } c))$ )
    by (simp add: fds-abs-converges-def)
next
  show  $\forall_F n \text{ in sequentially. norm } (\text{norm } (\text{fds-nth } (\text{fds-integral } a \ f) \ n / \text{nat-power } n \ (\text{of-real } c))) \leq$ 
     $\text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ (\text{of-real } c))$ 
    using eventually-gt-at-top[of 3]
  proof eventually-elim
    case (elim n)
    from elim and exp-le have  $\ln (\text{exp } 1) \leq \ln (\text{real } n)$ 
      by (subst ln-le-cancel-iff) auto
    hence  $1 * \text{norm } (\text{fds-nth } f \ n) \leq \ln (\text{real } n) * \text{norm } (\text{fds-nth } f \ n)$ 
      by (intro mult-right-mono) auto
    with elim show ?case
      by (simp add: norm-divide norm-nat-power fds-integral-def field-simps)
    qed
  qed
  thus ?case by (intro exI[of - of-real c]) auto
qed
next
show abs-conv-abscissa f  $\leq$  abs-conv-abscissa (fds-integral a f) (is -  $\leq$  ?s0)
proof (cases abs-conv-abscissa (fds-integral a f) =  $\infty$ )
  case False
  show ?thesis
proof (rule abs-conv-abscissa-leI)
  fix c :: real
  define  $\varepsilon$  where  $\varepsilon = (\text{if } ?s0 = -\infty \text{ then } 1 \text{ else } (c - \text{real-of-ereal } ?s0) / 2)$ 
  assume  $\text{ereal } c > ?s0$ 
  with False have  $\varepsilon: \varepsilon > 0 \ c - \varepsilon > ?s0$ 
    by (cases ?s0; force simp:  $\varepsilon$ -def field-simps)+

  have fds-abs-converges f (of-real c)
    unfolding fds-abs-converges-def
  proof (rule summable-comparison-test-ev)
    from  $\varepsilon$  have fds-abs-converges (fds-integral a f) (of-real (c -  $\varepsilon$ ))
      by (intro fds-abs-converges) (auto simp: algebra-simps)
    thus summable ( $\lambda n. \text{norm } (\text{fds-nth } (\text{fds-integral } a \ f) \ n / \text{nat-power } n \ (\text{of-real } (c - \varepsilon)))$ )
      by (simp add: fds-abs-converges-def)
  next
  have  $\forall_F n \text{ in at-top. } \ln (\text{real } n) / \text{real } n \text{ powr } \varepsilon < 1$ 
    by (rule order-tendstoD lim-ln-over-power  $\langle \varepsilon > 0 \rangle$  zero-less-one)+
  thus  $\forall_F n \text{ in sequentially. norm } (\text{norm } (\text{fds-nth } f \ n / \text{nat-power } n \ (\text{of-real } c)))$ 
     $\leq \text{norm } (\text{fds-nth } (\text{fds-integral } a \ f) \ n / \text{nat-power } n \ (\text{of-real } (c - \varepsilon)))$ 
    using eventually-gt-at-top[of 1]

```

**proof** *eventually-elim*  
**case** (*elim n*)  
**hence**  $\ln (\text{real } n) * \text{norm } (\text{fds-nth } f \ n) \leq \text{real } n \ \text{powr } \varepsilon * \text{norm } (\text{fds-nth } f \ n)$   
**by** (*intro mult-right-mono*) *auto*  
**with** *elim show ?case*  
**by** (*simp add: norm-divide norm-nat-power field-simps*  
*powr-diff inner-diff-left fds-integral-def*)  
**qed**  
**qed**  
**thus**  $\exists s. s \cdot 1 = c \wedge \text{fds-abs-converges } f \ s$   
**by** (*intro exI[of - of-real c]*) *auto*  
**qed**  
**qed** *auto*  
**qed**

**lemma** *abs-conv-abscissa-ln*:  
 $\text{abs-conv-abscissa } (\text{fds-ln } l \ (f :: 'a :: \text{dirichlet-series } \text{fds})) =$   
 $\text{abs-conv-abscissa } (\text{fds-deriv } f / f)$   
**by** (*simp add: fds-ln-def abs-conv-abscissa-integral*)

**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** –  
**have**  $\text{abs-conv-abscissa } (\text{fds-deriv } f) =$   
 $\text{abs-conv-abscissa } (\text{fds-integral } (\text{fds-nth } f \ 1) \ (\text{fds-deriv } f))$   
**by** (*rule abs-conv-abscissa-integral [symmetric]*)  
**also have**  $\text{fds-integral } (\text{fds-nth } f \ 1) \ (\text{fds-deriv } f) = f$   
**by** (*rule fds-integral-fds-deriv*)  
**finally show** *?thesis .*  
**qed**

**lemma** *abs-conv-abscissa-higher-deriv*:  
 $\text{abs-conv-abscissa } ((\text{fds-deriv } \widehat{\sim} \ n) \ f) = \text{abs-conv-abscissa } (f :: 'a :: \text{dirichlet-series } \text{fds})$   
**by** (*induction n*) (*simp-all add: abs-conv-abscissa-deriv*)

**lemma** *conv-abscissa-higher-deriv-le*:  
 $\text{conv-abscissa } ((\text{fds-deriv } \widehat{\sim} \ n) \ f) \leq \text{conv-abscissa } (f :: 'a :: \text{dirichlet-series } \text{fds})$   
**by** (*induction n*) (*auto intro: order.trans[OF conv-abscissa-deriv-le]*)

**lemma** *abs-conv-abscissa-restrict*:  
 $\text{abs-conv-abscissa } (\text{fds-subseries } P \ f) \leq \text{abs-conv-abscissa } f$   
**by** (*rule abs-conv-abscissa-mono*) *auto*

**lemma** *eval-fds-deriv*:  
**fixes**  $f :: 'a \ \text{fds}$   
**assumes**  $s \cdot 1 > \text{conv-abscissa } f$

**shows**  $eval-fds (fds-deriv f) s = deriv (eval-fds f) s$   
**by** (*intro DERIV-imp-deriv [symmetric] derivative-intros assms*)

**lemma** *eval-fds-higher-deriv*:

**assumes**  $(s :: 'a :: dirichlet-series) \cdot 1 > conv-abscissa f$   
**shows**  $eval-fds ((fds-deriv \hat{\sim} n) f) s = (deriv \hat{\sim} n) (eval-fds f) s$   
**using** *assms*

**proof** (*induction n arbitrary: f s*)

**case** (*Suc n f s*)

**have** *ev*: *eventually*  $(\lambda s. s \in \{s. s \cdot 1 > conv-abscissa f\}) (nhds s)$   
**using** *Suc.premis open-halfspace-gt[of - 1::'a]*

**by** (*intro eventually-nhds-in-open, cases conv-abscissa f*)  
*(auto simp: open-halfspace-gt inner-commute)*

**have**  $eval-fds ((fds-deriv \hat{\sim} Suc n) f) s = eval-fds ((fds-deriv \hat{\sim} n) (fds-deriv f)) s$

**by** (*subst funpow-Suc-right*) *simp*

**also have**  $\dots = (deriv \hat{\sim} n) (eval-fds (fds-deriv f)) s$

**by** (*intro Suc.IH le-less-trans[OF conv-abscissa-deriv-le] Suc.premis*)

**also have**  $\dots = (deriv \hat{\sim} n) (deriv (eval-fds f)) s$

**by** (*intro higher-deriv-cong-ev refl eventually-mono[OF ev] eval-fds-deriv*) *auto*

**also have**  $\dots = (deriv \hat{\sim} Suc n) (eval-fds f) s$

**by** (*subst funpow-Suc-right*) *simp*

**finally show** *?case* .

**qed** *auto*

**end**

### 12.3 Multiplication of two series

**lemma**

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

**fixes**  $s :: 'a$

**assumes** [*simp*]:  $f 0 = 0 \ g 0 = 0$

**assumes** *summable*: *summable*  $(\lambda n. norm (f n / nat-power n s))$   
*summable*  $(\lambda n. norm (g n / nat-power n s))$

**shows** *summable-dirichlet-prod*: *summable*  $(\lambda n. norm (dirichlet-prod f g n / nat-power n s))$

**and** *suminf-dirichlet-prod*:

$$\left(\sum n. dirichlet-prod f g n / nat-power n s\right) = \left(\sum n. f n / nat-power n s\right) * \left(\sum n. g n / nat-power n s\right)$$

**proof** –

**have** *summable'*:  $(\lambda n. f n / nat-power n s)$  *abs-summable-on A*

$(\lambda n. g n / nat-power n s)$  *abs-summable-on A* **for A**

**by** (*(rule abs-summable-on-subset[OF - subset-UNIV], insert summable, simp add: abs-summable-on-nat-iff')*; *fail*)+

**have** *f-g*:  $f a / nat-power a s * (g b / nat-power b s) =$

$$f a * g b / nat-power (a * b) s \text{ for } a b$$

**by** (*cases a \* b = 0*) (*auto simp: nat-power-mult-distrib*)

**have**  $eq: (\sum_{a(m,n) \in \{(m,n). m * n = x\}} f m * g n / nat-power x s) =$   
 $dirichlet-prod f g x / nat-power x s$  **for**  $x :: nat$   
**proof** (*cases*  $x > 0$ )  
  **case** *False*  
  **hence**  $(\sum_{a(m,n) \mid m * n = x} f m * g n / nat-power x s) = (\sum_{a(m,n) \mid m * n = x} 0)$   
  **by** (*intro infsetsum-cong*) *auto*  
  **with** *False* **show** *?thesis* **by** *simp*  
**next**  
  **case** *True*  
  **from** *finite-divisors-nat* [*OF this*] **show** *?thesis*  
  **by** (*simp add: dirichlet-prod-altdef2 case-prod-unfold sum-divide-distrib*)  
**qed**

**have**  $(\lambda(m,n). (f m / nat-power m s) * (g n / nat-power n s))$  *abs-summable-on*  
 $UNIV \times UNIV$   
  **using** *summable'* **by** (*intro abs-summable-on-product*) *auto*  
**also have** *?this*  $\longleftrightarrow (\lambda(m,n). f m * g n / nat-power (m*n) s)$  *abs-summable-on*  
 $UNIV$   
  **using** *f-g* **by** (*intro abs-summable-on-cong*) *auto*  
**also have**  $\dots \longleftrightarrow (\lambda(x,(m,n)). f m * g n / nat-power (m*n) s)$  *abs-summable-on*

$(SIGMA x:UNIV. \{(m,n). m * n = x\})$   
  **unfolding** *case-prod-unfold*  
  **by** (*rule abs-summable-on-reindex-bij-betw* [*symmetric*])  
  (*auto simp: bij-betw-def inj-on-def image-iff*)  
**also have**  $\dots \longleftrightarrow (\lambda(x,(m,n)). f m * g n / nat-power x s)$  *abs-summable-on*  
   $(SIGMA x:UNIV. \{(m,n). m * n = x\})$   
  **by** (*intro abs-summable-on-cong*) *auto*  
**finally have** *summable''*:  $\dots$  .  
**from** *abs-summable-on-Sigma-project1* [*OF this*]  
  **show** *summable'''*: *summable*  $(\lambda n. norm (dirichlet-prod f g n / nat-power n s))$   
  **by** (*simp add: eq abs-summable-on-nat-iff'*)

**have**  $(\sum n. f n / nat-power n s) * (\sum n. g n / nat-power n s) =$   
 $(\sum_{a n}. f n / nat-power n s) * (\sum_{a n}. g n / nat-power n s)$   
  **using** *summable'* **by** (*simp add: infsetsum-nat'*)  
**also have**  $\dots = (\sum_{a(m,n)}. (f m / nat-power m s) * (g n / nat-power n s))$   
  **using** *summable'* **by** (*subst infsetsum-product* [*symmetric*]) *simp-all*  
**also have**  $\dots = (\sum_{a(m,n)}. f m * g n / nat-power (m * n) s)$   
  **using** *f-g* **by** (*intro infsetsum-cong refl*) *auto*  
**also have**  $\dots = (\sum_{a(x,(m,n)) \in (SIGMA x:UNIV. \{(m,n). m * n = x\})} f m * g n / nat-power (m * n) s)$   
  **unfolding** *case-prod-unfold*  
  **by** (*rule infsetsum-reindex-bij-betw* [*symmetric*]) (*auto simp: bij-betw-def inj-on-def image-iff*)  
**also have**  $\dots = (\sum_{a(x,(m,n)) \in (SIGMA x:UNIV. \{(m,n). m * n = x\})} f m * g n / nat-power x s)$



**lemma** *abs-conv-abcissa-mult-leI*:

*abs-conv-abcissa* ( $f :: 'a :: \text{dirichlet-series fds}$ )  $\leq d \implies$   
*abs-conv-abcissa*  $g \leq d \implies \text{abs-conv-abcissa } (f * g) \leq d$   
**using** *abs-conv-abcissa-mult-le*[*of f g*] **by** (*auto simp add: le-max-iff-disj*)

**lemma** *abs-conv-abcissa-shift* [*simp*]:

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

**proof** –

**have** *abs-conv-abcissa* (*fds-shift*  $c f$ )  $\leq \text{abs-conv-abcissa } f + c \cdot 1$  **for**  $c :: 'a$   
**and**  $f$

**proof** (*rule abs-conv-abcissa-leI*)

**fix**  $d$  **assume** *abs-conv-abcissa*  $f + c \cdot 1 < \text{ereal } d$

**hence** *abs-conv-abcissa*  $f < \text{ereal } (d - c \cdot 1)$  **by** (*cases abs-conv-abcissa f*)

*auto*

**hence** *fds-abs-converges* (*fds-shift*  $c f$ ) (*of-real d*)

**by** (*auto intro!: fds-abs-converges-shift fds-abs-converges simp: algebra-simps*)

**thus**  $\exists s. s \cdot 1 = d \wedge \text{fds-abs-converges } (f \text{ds-shift } c f) s$

**by** (*auto intro!: exI[of - of-real d]*)

**qed**

**note**  $*$  = *this[of c f] this[of -c fds-shift c f]*

**show** *?thesis* **by** (*cases abs-conv-abcissa (fds-shift c f); cases abs-conv-abcissa f*)

(*insert \*, auto intro!: antisym*)

**qed**

**lemma** *eval-fds-mult*:

**fixes**  $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$

**assumes** *fds-abs-converges*  $f s$  *fds-abs-converges*  $g s$

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

**using** *suminf-dirichlet-prod*[*OF - - assms[unfolded fds-abs-converges-def]*]

**by** (*simp-all add: eval-fds-def fds-nth-mult*)

**lemma** *eval-fds-power*:

**fixes**  $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$

**assumes** *fds-abs-converges*  $f s$

**shows** *eval-fds* ( $f \wedge n$ )  $s = \text{eval-fds } f s \wedge n$

**using** *assms* **by** (*induction n*) (*simp-all add: eval-fds-mult fds-abs-converges-power*)

**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** *eval-fds* (*prod f A*)  $s = (\prod_{x \in A}. \text{eval-fds } (f x) s)$  **using** *assms*

**by** (*induction A rule: infinite-finite-induct*) (*auto simp: eval-fds-mult fds-abs-converges-prod*)

**lemma** *eval-fds-inverse*:

**fixes**  $s :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$

**assumes** *fds-abs-converges*  $f s$  *fds-abs-converges* (*inverse f*)  $s$  *fds-nth f 1*  $\neq 0$

**shows**  $eval-fds (inverse f) s = inverse (eval-fds f s)$   
**proof** –  
**have**  $eval-fds (inverse f * f) s = eval-fds (inverse f) s * eval-fds f s$   
**by**  $(intro\ eval-fds-mult\ assms)$   
**also have**  $inverse f * f = 1$  **by**  $(intro\ fds-left-inverse\ assms)$   
**also have**  $eval-fds 1 s = 1$  **by**  $simp$   
**finally show**  $?thesis$  **by**  $(auto\ simp:\ divide-simps)$   
**qed**

**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** –  
**have**  $conv-abscissa (fds-integral c f) \leq abs-conv-abscissa (fds-integral c f)$   
**by**  $(rule\ conv-le-abs-conv-abscissa)$   
**also from**  $assms$  **have**  $\dots < ereal (s \cdot 1)$  **by**  $(simp\ add:\ abs-conv-abscissa-integral)$   
**finally have**  $(eval-fds (fds-integral c f) has-field-derivative$   
 $eval-fds (fds-deriv (fds-integral c f)) s) (at s)$   
**by**  $(intro\ derivative-eq-intros)\ auto$   
**also from**  $assms$  **have**  $fds-deriv (fds-integral c f) = f$   
**by**  $simp$   
**finally show**  $?thesis$  .  
**qed**

**lemma**  $holomorphic-fds-eval$   $[holomorphic-intros]$ :  
 $A \subseteq \{z. Re\ z > conv-abscissa f\} \implies eval-fds f\ holomorphic-on\ A$   
**unfolding**  $holomorphic-on-def\ field-differentiable-def$   
**by**  $(rule\ ballI\ exI\ derivative-intros)+\ auto$

**lemma**  $analytic-fds-eval$   $[holomorphic-intros]$ :  
**assumes**  $A \subseteq \{z. Re\ z > conv-abscissa f\}$   
**shows**  $eval-fds f\ analytic-on\ A$   
**proof** –  
**have**  $eval-fds f\ analytic-on\ \{z. Re\ z > conv-abscissa f\}$   
**proof**  $(subst\ analytic-on-open)$   
**show**  $open\ \{z. Re\ z > conv-abscissa f\}$   
**by**  $(cases\ conv-abscissa\ f)\ (simp-all\ add:\ open-halfspace-Re-gt)$   
**qed**  $(intro\ holomorphic-intros,\ simp-all)$   
**from**  $analytic-on-subset[OF\ this\ assms]$  **show**  $?thesis$  .  
**qed**

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

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

**by** (auto simp: abs-conv-abscissa-MInf-iff)

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

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

**by** (auto simp: conv-abscissa-MInf-iff)

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

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

**by** (auto simp: abs-conv-abscissa-MInf-iff)

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

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

**by** (auto simp: conv-abscissa-MInf-iff)

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

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

**by** (auto simp: abs-conv-abscissa-MInf-iff)

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

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

**by** (auto simp: numeral-fds)

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

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

**by** (auto simp: numeral-fds)

**lemma** abs-conv-abscissa-power-le:

$abs-conv-abscissa (f \wedge n :: 'a :: dirichlet-series fds) \leq abs-conv-abscissa f$

**by** (induction n) (auto intro!: order.trans[OF abs-conv-abscissa-mult-le])

**lemma** abs-conv-abscissa-power-leI:

$abs-conv-abscissa (f :: 'a :: dirichlet-series fds) \leq d \implies abs-conv-abscissa (f \wedge n) \leq d$

**by** (rule order.trans[OF abs-conv-abscissa-power-le])

**lemma** abs-conv-abscissa-prod-le:

**assumes**  $\bigwedge x. x \in A \implies abs-conv-abscissa (f x :: 'a :: dirichlet-series fds) \leq d$

**shows**  $abs-conv-abscissa (prod f A) \leq d$  **using** *assms*

**by** (induction A rule: infinite-finite-induct) (auto intro!: abs-conv-abscissa-mult-leI)

**lemma** conv-abscissa-add-le:

$conv-abscissa (f + g :: 'a :: dirichlet-series fds) \leq \max (conv-abscissa f) (conv-abscissa g)$

**by** (rule conv-abscissa-leI-weak) (auto intro!: fds-converges-add intro: fds-converges)

**lemma** conv-abscissa-add-leI:

$conv-abscissa (f :: 'a :: dirichlet-series fds) \leq d \implies conv-abscissa g \leq d \implies$

$conv-abscissa (f + g) \leq d$

**using** conv-abscissa-add-le[of f g] **by** (auto simp: le-max-iff-disj)

**lemma** *conv-abscissa-sum-leI*:

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

**shows**  $\text{conv-abscissa } (\text{sum } f A) \leq d$  **using** *assms*

**by** (*induction A rule: infinite-finite-induct*) (*auto intro!: conv-abscissa-add-leI*)

**lemma** *abs-conv-abscissa-add-le*:

$\text{abs-conv-abscissa } (f + g :: 'a :: \text{dirichlet-series fds}) \leq \max (\text{abs-conv-abscissa } f)$   
 $(\text{abs-conv-abscissa } g)$

**by** (*rule abs-conv-abscissa-leI-weak*) (*auto intro!: fds-abs-converges-add intro: fds-abs-converges*)

**lemma** *abs-conv-abscissa-add-leI*:

$\text{abs-conv-abscissa } (f :: 'a :: \text{dirichlet-series fds}) \leq d \implies \text{abs-conv-abscissa } g \leq d$   
 $\implies$

$\text{abs-conv-abscissa } (f + g) \leq d$

**using** *abs-conv-abscissa-add-le*[*of f g*] **by** (*auto simp: le-max-iff-disj*)

**lemma** *abs-conv-abscissa-sum-leI*:

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

**shows**  $\text{abs-conv-abscissa } (\text{sum } f A) \leq d$  **using** *assms*

**by** (*induction A rule: infinite-finite-induct*) (*auto intro!: abs-conv-abscissa-add-leI*)

**lemma** *fds-converges-cmult-left* [*intro*]:

**assumes** *fds-converges f s*

**shows** *fds-converges (fds-const c \* f) s*

**proof** –

**from** *assms* **have** *summable*  $(\lambda n. c * (\text{fds-nth } f n / \text{nat-power } n s))$

**by** (*intro summable-mult*) (*auto simp: fds-converges-def*)

**thus** *?thesis* **by** (*simp add: fds-converges-def mult-ac*)

**qed**

**lemma** *fds-converges-cmult-right* [*intro*]:

**assumes** *fds-converges f s*

**shows** *fds-converges (f \* fds-const c) s*

**using** *fds-converges-cmult-left*[*OF assms*] **by** (*simp add: mult-ac*)

**lemma** *conv-abscissa-cmult-left* [*simp*]:

**fixes**  $c :: 'a :: \text{dirichlet-series}$  **assumes**  $c \neq 0$

**shows**  $\text{conv-abscissa } (\text{fds-const } c * f) = \text{conv-abscissa } f$

**proof** –

**have** *fds-converges (fds-const c \* f) s*  $\iff$  *fds-converges f s* **for**  $s$

**proof**

**assume** *fds-converges (fds-const c \* f) s*

**hence** *fds-converges (fds-const (inverse c) \* (fds-const c \* f)) s*

**by** (*rule fds-converges-cmult-left*)

**also have**  $\text{fds-const } (\text{inverse } c) * (\text{fds-const } c * f) = \text{fds-const } (\text{inverse } c * c)$

$* f$

**by** *simp*

**also have**  $\text{inverse } c * c = 1$   
**using** *assms* **by** *simp*  
**finally show**  $\text{fds-converges } f \text{ s}$  **by** *simp*  
**qed** *auto*  
**thus** *?thesis* **by** (*simp add: conv-abscissa-def*)  
**qed**

**lemma** *conv-abscissa-cmult-right* [*simp*]:  
**fixes**  $c :: 'a :: \text{dirichlet-series}$  **assumes**  $c \neq 0$   
**shows**  $\text{conv-abscissa } (f * \text{fds-const } c) = \text{conv-abscissa } f$   
**using** *assms* **by** (*subst mult.commute*) *auto*

**lemma** *abs-conv-abscissa-cmult*:  
**fixes**  $c :: 'a :: \text{dirichlet-series}$  **assumes**  $c \neq 0$   
**shows**  $\text{abs-conv-abscissa } (\text{fds-const } c * f) = \text{abs-conv-abscissa } f$   
**proof** (*intro antisym*)  
**have**  $\text{abs-conv-abscissa } (\text{fds-const } (\text{inverse } c) * (\text{fds-const } c * f)) \leq$   
 $\text{abs-conv-abscissa } (\text{fds-const } c * f)$   
**using** *abs-conv-abscissa-mult-le*[*of*  $\text{fds-const } (\text{inverse } c) \text{ fds-const } c * f$ ]  
**by** (*auto simp: max-def*)  
**also have**  $\text{fds-const } (\text{inverse } c) * (\text{fds-const } c * f) = \text{fds-const } (\text{inverse } c * c) * f$   
**by** (*simp add: mult-ac*)  
**also have**  $\text{inverse } c * c = 1$  **using** *assms* **by** *simp*  
**finally show**  $\text{abs-conv-abscissa } f \leq \text{abs-conv-abscissa } (\text{fds-const } c * f)$  **by** *simp*  
**qed** (*insert abs-conv-abscissa-mult-le*[*of*  $\text{fds-const } c f$ ], *auto simp: max-def*)

**lemma** *conv-abscissa-minus* [*simp*]:  
**fixes**  $f :: 'a :: \text{dirichlet-series}$  *fds*  
**shows**  $\text{conv-abscissa } (-f) = \text{conv-abscissa } f$   
**using** *conv-abscissa-cmult-left*[*of*  $-1 f$ ] **by** *simp*

**lemma** *abs-conv-abscissa-minus* [*simp*]:  
**fixes**  $f :: 'a :: \text{dirichlet-series}$  *fds*  
**shows**  $\text{abs-conv-abscissa } (-f) = \text{abs-conv-abscissa } f$   
**using** *abs-conv-abscissa-cmult*[*of*  $-1 f$ ] **by** *simp*

**lemma** *conv-abscissa-diff-le*:  
 $\text{conv-abscissa } (f - g :: 'a :: \text{dirichlet-series } \text{fds}) \leq \max (\text{conv-abscissa } f) (\text{conv-abscissa } g)$   
**using** *conv-abscissa-add-le*[*of*  $f -g$ ] **by** *simp*

**lemma** *conv-abscissa-diff-leI*:  
 $\text{conv-abscissa } (f :: 'a :: \text{dirichlet-series } \text{fds}) \leq d \implies \text{conv-abscissa } g \leq d \implies$   
 $\text{conv-abscissa } (f - g) \leq d$   
**using** *conv-abscissa-add-le*[*of*  $f -g$ ] **by** (*auto simp: le-max-iff-disj*)

**lemma** *abs-conv-abscissa-diff-le*:  
 $\text{abs-conv-abscissa } (f - g :: 'a :: \text{dirichlet-series } \text{fds}) \leq$   
 $\max (\text{abs-conv-abscissa } f) (\text{abs-conv-abscissa } g)$

**using** *abs-conv-abcissa-add-le*[*of f - g*] **by** *simp*

**lemma** *abs-conv-abcissa-diff-leI*:  
*abs-conv-abcissa (f :: 'a :: dirichlet-series fds) ≤ d ⇒ abs-conv-abcissa g ≤ d*  
 $\Rightarrow$   
*abs-conv-abcissa (f - g) ≤ d*  
**using** *abs-conv-abcissa-add-le*[*of f - g*] **by** (*auto simp: le-max-iff-disj*)

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

**lemma** *abs-conv-abcissa-completely-multiplicative-log-deriv*:  
**fixes** *f :: 'a :: dirichlet-series fds*  
**assumes** *completely-multiplicative-function (fds-nth f) fds-nth f 1 ≠ 0*  
**shows** *abs-conv-abcissa (fds-deriv f / f) ≤ abs-conv-abcissa f*  
**proof** -  
**have** *fds-deriv f = - fds (λn. fds-nth f n \* mangoldt n) \* f*  
**using** *assms* **by** (*subst completely-multiplicative-fds-deriv'*) *simp-all*  
**also have**  $\dots / f = - \text{fds } (\lambda n. \text{fds-nth } f \ n * \text{mangoldt } n) * (f * \text{inverse } f)$   
**by** (*simp add: divide-fds-def*)  
**also have**  $f * \text{inverse } f = 1$  **using** *assms* **by** (*intro fds-right-inverse*)  
**finally have** *fds-deriv f / f = - fds (λn. fds-nth f n \* mangoldt n)* **by** *simp*  
**also have** *abs-conv-abcissa \dots =*  
*abs-conv-abcissa (fds (λn. fds-nth f n \* mangoldt n))*  
*(is - = abs-conv-abcissa ?f)* **by** (*rule abs-conv-abcissa-minus*)  
**also have**  $\dots \leq \text{abs-conv-abcissa } f$   
**proof** (*rule abs-conv-abcissa-leI, goal-cases*)  
**case** (*1 c*)  
**have** *fds-abs-converges ?f (of-real c)* **unfolding** *fds-abs-converges-def*  
**proof** (*rule summable-comparison-test-ev*)  
**from** *1* **have** *fds-abs-converges (fds-deriv f) (of-real c)*  
**by** (*intro fds-abs-converges*) (*auto simp: abs-conv-abcissa-deriv*)  
**thus** *summable (λn. |ln (real n)| \* norm (fds-nth f n) / norm (nat-power n*  
*(of-real c :: 'a)))*  
**by** (*simp add: fds-abs-converges-def fds-deriv-def fds-nth-fds'*  
*scaleR-conv-of-real powr-minus norm-mult norm-divide*  
*norm-nat-power*)  
**next**  
**show**  $\forall_F n$  *in sequentially.*  
*norm (norm (fds-nth (fds (λn. fds-nth f n \* mangoldt n)) n /*  
*nat-power n (of-real c)))*  
 $\leq | \ln (\text{real } n) | * \text{norm } (\text{fds-nth } f \ n) / \text{norm } (\text{nat-power } n \ (\text{of-real } c)) ::$   
*'a)*  
**using** *eventually-gt-at-top*[*of 0*]  
**proof** *eventually-elim*  
**case** (*elim n*)  
**have** *norm (norm (fds-nth (fds (λn. fds-nth f n \* mangoldt n)) n /*  
*nat-power n (of-real c))) =*  
*norm (fds-nth f n) \* mangoldt n / real n powr c*

```

    using elim by (simp add: fds-nth-fds' norm-mult norm-divide
                      norm-nat-power abs-mult mangoldt-nonneg)
  also have ... ≤ norm (fds-nth f n) * ln n / real n powr c using elim
    by (intro mult-left-mono divide-right-mono mangoldt-le)
      (simp-all add: mangoldt-def)
  finally show ?case using elim by (simp add: norm-nat-power algebra-simps)
qed
qed
thus ?case by (intro exI[of - of-real c]) auto
qed
finally show ?thesis .
qed

```

## 12.4 Uniqueness

context

assumes SORT-CONSTRAINT ('a :: dirichlet-series)

begin

lemma norm-dirichlet-series-cutoff-le:

assumes fds-abs-converges f (s0 :: 'a) N > 0 s · 1 ≥ c c ≥ s0 · 1

shows summable (λn. fds-nth f (n + N) / nat-power (n + N) s)

summable (λn. norm (fds-nth f (n + N)) / nat-power (n + N) c)

and norm (∑ n. fds-nth f (n + N) / nat-power (n + N) s) ≤

(∑ n. norm (fds-nth f (n + N)) / nat-power (n + N) c) / nat-power

N (s · 1 - c)

proof -

from assms have fds-abs-converges f (of-real c)

using fds-abs-converges-Re-le[of f s0 of-real c] by auto

hence summable (λn. norm (fds-nth f (n + N) / nat-power (n + N) (of-real c)))

unfolding fds-abs-converges-def by (rule summable-ignore-initial-segment)

also have ?this ↔ summable (λn. norm (fds-nth f (n + N)) / nat-power (n + N) c)

by (intro summable-cong eventually-mono[OF eventually-gt-at-top[of 0::nat]])

(auto simp: norm-divide norm-nat-power)

finally show \*: summable (λn. norm (fds-nth f (n + N)) / nat-power (n + N) c) .

from assms have fds-abs-converges f s using fds-abs-converges-Re-le[of f s0 s]

by auto

hence \*\*: summable (λn. norm (fds-nth f (n + N) / nat-power (n + N) s))

unfolding fds-abs-converges-def by (rule summable-ignore-initial-segment)

thus summable (λn. fds-nth f (n + N) / nat-power (n + N) s)

by (rule summable-norm-cancel)

have norm (∑ n. fds-nth f (n + N) / nat-power (n + N) s)

≤ (∑ n. norm (fds-nth f (n + N) / nat-power (n + N) s))

by (intro summable-norm \*\*)

**also have**  $\dots \leq (\sum n. \text{norm } (\text{fds-nth } f (n + N)) / \text{nat-power } (n + N) c / \text{nat-power } N (s \cdot 1 - c))$   
**proof** (intro suminf-le \*\* summable-divide allI)  
**fix**  $n :: \text{nat}$   
**have**  $\text{real } N \text{ powr } (s \cdot 1 - c) \leq \text{real } (n + N) \text{ powr } (s \cdot 1 - c)$   
**using** *assms* **by** (intro powr-mono2) simp-all  
**also have**  $\text{real } (n + N) \text{ powr } c * \dots = \text{real } (n + N) \text{ powr } (s \cdot 1)$   
**by** (simp add: powr-diff)  
**finally have**  $\text{norm } (\text{fds-nth } f (n + N)) / \text{real } (n + N) \text{ powr } (s \cdot 1) \leq$   
 $\text{norm } (\text{fds-nth } f (n + N)) / (\text{real } (n + N) \text{ powr } c * \text{real } N \text{ powr } (s \cdot 1 - c))$   
**using**  $\langle N > 0 \rangle$  **by** (intro divide-left-mono) (simp-all add: mult-left-mono)  
**thus**  $\text{norm } (\text{fds-nth } f (n + N) / \text{nat-power } (n + N) s) \leq$   
 $\text{norm } (\text{fds-nth } f (n + N)) / \text{nat-power } (n + N) c / \text{nat-power } N (s \cdot 1 - c)$   
**using**  $\langle N > 0 \rangle$  **by** (simp add: norm-divide norm-nat-power )  
**qed**  
**also have**  $\dots = (\sum n. \text{norm } (\text{fds-nth } f (n + N)) / \text{nat-power } (n + N) c) / \text{nat-power } N (s \cdot 1 - c)$   
**using** \* **by** (rule suminf-divide)  
**finally show**  $\text{norm } (\sum n. \text{fds-nth } f (n + N) / \text{nat-power } (n + N) s) \leq \dots$   
**qed**

**lemma** *eval-fds-zeroD-aux*:

**fixes**  $h :: 'a \text{ fds}$   
**assumes** *conv*: *fds-abs-converges*  $h (s0 :: 'a)$   
**assumes** *freq*: *frequently*  $(\lambda s. \text{eval-fds } h s = 0)$   $((\lambda s. s \cdot 1) \text{ going-to at-top})$   
**shows**  $h = 0$   
**proof** (rule ccontr)  
**assume**  $h \neq 0$   
**hence** *ex*:  $\exists n > 0. \text{fds-nth } h n \neq 0$  **by** (auto simp: fds-eq-iff)  
**define**  $N :: \text{nat}$  **where**  $N = (\text{LEAST } n. n > 0 \wedge \text{fds-nth } h n \neq 0)$   
**have**  $N: N > 0 \text{ fds-nth } h N \neq 0$   
**using** *LeastI-ex*[*OF ex, folded N-def*] **by** auto  
**have** *less-N*:  $\text{fds-nth } h n = 0$  **if**  $n < N$  **for**  $n$   
**using** *Least-le*[*of*  $\lambda n. n > 0 \wedge \text{fds-nth } h n \neq 0$ , *folded N-def*] **that**  
**by** (cases  $n = 0$ ) (auto simp: not-less)

**define**  $c$  **where**  $c = s0 \cdot 1$

**define** *remainder* **where**  $\text{remainder} = (\lambda s. (\sum n. \text{fds-nth } h (n + \text{Suc } N) / \text{nat-power } (n + \text{Suc } N) s))$

**define**  $A$  **where**  $A = (\sum n. \text{norm } (\text{fds-nth } h (n + \text{Suc } N)) / \text{nat-power } (n + \text{Suc } N) c) * \text{nat-power } (\text{Suc } N) c$

**have** *eq*:  $\text{fds-nth } h N = \text{nat-power } N s * \text{eval-fds } h s - \text{nat-power } N s * \text{remainder } s$

**if**  $s \cdot 1 \geq c$  **for**  $s :: 'a$

**proof** –

**from** *conv* **and** *that* **have** *conv'*: *fds-abs-converges h s*  
**unfolding** *c-def* **by** (*rule fds-abs-converges-Re-le*)  
**hence** *conv''*: *fds-converges h s* **by** *blast*  
**from** *conv''* **have**  $(\lambda n. \text{fds-nth } h \ n / \text{nat-power } n \ s)$  *sums eval-fds h s*  
**by** (*simp add: fds-converges-iff*)  
**hence**  $(\lambda n. \text{fds-nth } h \ (n + \text{Suc } N) / \text{nat-power } (n + \text{Suc } N) \ s)$  *sums*  
 $(\text{eval-fds } h \ s - (\sum n < \text{Suc } N. \text{fds-nth } h \ n / \text{nat-power } n \ s))$   
**by** (*rule sums-split-initial-segment*)  
**also have**  $(\sum n < \text{Suc } N. \text{fds-nth } h \ n / \text{nat-power } n \ s) =$   
 $(\sum n < \text{Suc } N. \text{if } n = N \text{ then } \text{fds-nth } h \ N / \text{nat-power } N \ s \text{ else } 0)$   
**by** (*intro sum.cong refl*) (*auto simp: less-N*)  
**also have**  $\dots = \text{fds-nth } h \ N / \text{nat-power } N \ s$  **by** (*subst sum.delta*) *auto*  
**finally show** *?thesis* **unfolding** *remainder-def* **using**  $\langle N > 0 \rangle$  **by** (*auto simp:*  
*sums-iff field-simps*)  
**qed**

**have** *remainder-bound*:  $\text{norm } (\text{remainder } s) \leq A / \text{real } (\text{Suc } N) \ \text{powr } (s \cdot 1)$   
**if**  $s \cdot 1 \geq c$  **for**  $s :: 'a$

**proof** –

**note**  $*$  = *norm-dirichlet-series-cutoff-le*[*of h s0 Suc N c s, folded remainder-def*]  
**have**  $\text{norm } (\text{remainder } s) \leq (\sum n. \text{norm } (\text{fds-nth } h \ (n + \text{Suc } N))) /$   
 $\text{nat-power } (n + \text{Suc } N) \ c / \text{nat-power } (\text{Suc } N) \ (s \cdot 1 - c)$   
**using** *that* *assms* **unfolding** *remainder-def* **by** (*intro \**) (*simp-all add: c-def*)  
**also have**  $\dots = A / \text{real } (\text{Suc } N) \ \text{powr } (s \cdot 1)$  **by** (*simp add: A-def powr-diff*)  
**finally show** *?thesis* .

**qed**

**from** *freq* **have**  $\forall c. \exists s. s \cdot 1 \geq c \wedge \text{eval-fds } h \ s = 0$

**unfolding** *frequently-def* **by** (*auto simp: eventually-going-to-at-top-linorder*)

**hence**  $\forall k. \exists s. s \cdot 1 \geq \text{real } k \wedge \text{eval-fds } h \ s = 0$  **by** *blast*

**then obtain** *S* **where**  $S: \bigwedge k. S \ k \cdot 1 \geq \text{real } k \wedge \text{eval-fds } h \ (S \ k) = 0$

**by** *metis*

**have** *S-limit*: *filterlim*  $(\lambda k. S \ k \cdot 1)$  *at-top* *sequentially*

**by** (*rule filterlim-at-top-mono*[*OF filterlim-real-sequentially*]) (*use S in auto*)

**have** *eventually*  $(\lambda k. \text{real } k \geq c)$  *sequentially* **by** *real-asymp*

**hence** *eventually*  $(\lambda k. \text{norm } (\text{fds-nth } h \ N) \leq$   
 $(\text{real } N / \text{real } (\text{Suc } N)) \ \text{powr } (S \ k \cdot 1) * A)$  *sequentially*

**proof** *eventually-elim*

**case** (*elim k*)

**hence**  $\text{norm } (\text{fds-nth } h \ N) = \text{real } N \ \text{powr } (S \ k \cdot 1) * \text{norm } (\text{remainder } (S \ k))$

(*is - = - \* ?X*) **using**  $\langle N > 0 \rangle$  *S*[*of k*] *eq*[*of S k*]

**by** (*auto simp: norm-mult norm-nat-power c-def*)

**also have**  $\text{norm } (\text{remainder } (S \ k)) \leq A / \text{real } (\text{Suc } N) \ \text{powr } (S \ k \cdot 1)$

**using** *elim S*[*of k*] **by** (*intro remainder-bound*) (*simp-all add: c-def*)

**finally show** *?case*

**using** *N* **by** (*simp add: mult-left-mono powr-divide field-simps del: of-nat-Suc*)

**qed**

**moreover have**  $((\lambda k. (\text{real } N / \text{real } (\text{Suc } N)) \ \text{powr } (S \ k \cdot 1) * A) \longrightarrow 0)$

*sequentially*

**by** (rule *filterlim-compose*[*OF - S-limit*]) (use  $\langle N > 0 \rangle$  **in** *real-asymp*)  
**ultimately have**  $((\lambda s. \text{fds-nth } h \ N) \longrightarrow 0)$  *sequentially*  
**by** (rule *Lim-null-comparison*)  
**hence**  $\text{fds-nth } h \ N = 0$  **by** (*simp add: tendsto-const-iff*)  
**with**  $\langle \text{fds-nth } h \ N \neq 0 \rangle$  **show** *False* **by** *contradiction*  
**qed**

**lemma** *eval-fds-zeroD*:

**fixes**  $h :: 'a \text{ fds}$   
**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** –

**have** [*simp*]:  $2 \cdot (1 :: 'a) = 2$   
**using** *of-real-inner-1*[*of 2*] **unfolding** *of-real-numeral* **by** *simp*  
**from** *conv* **obtain**  $s$  **where** *fds-converges*  $h \ s$   
**by** *auto*  
**hence** *fds-abs-converges*  $h \ (s + 2)$   
**by** (rule *fds-converges-imp-abs-converges*) (*auto simp: algebra-simps*)  
**from** *this* *assms*(2–) **show** *?thesis* **by** (rule *eval-fds-zeroD-aux*)

**qed**

**lemma** *eval-fds-eqD*:

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

**have** *conv'*: *conv-abscissa*  $(f - g) < \infty$   
**using** *assms* **by** (*intro le-less-trans*[*OF conv-abscissa-diff-le*]) (*auto simp: max-def*)  
  
**have** *max* (*conv-abscissa*  $f$ ) (*conv-abscissa*  $g$ )  $< \infty$   
**using** *conv* **by** (*auto simp: max-def*)  
**from** *ereal-dense2*[*OF this*] **obtain**  $c$  **where**  $c : \text{max} (\text{conv-abscissa } f) (\text{conv-abscissa } g) < \text{ereal } c$   
**by** *auto*

**have** *frequently*  $(\lambda s. \text{eval-fds } f \ s = \text{eval-fds } g \ s \wedge s \cdot 1 \geq c)$   $((\lambda s. s \cdot 1)$  *going-to at-top*)

**using** *eq* **by** (rule *frequently-eventually-frequently*) *auto*  
**hence**  $*$ : *frequently*  $(\lambda s. \text{eval-fds } (f - g) \ s = 0)$   $((\lambda s. s \cdot 1)$  *going-to at-top*)  
**proof** (rule *frequently-mono* [*rotated*], *safe*, *goal-cases*)  
**case** (1  $s$ )  
**thus** *?case* **using**  $c$   
**by** (*subst eval-fds-diff*) (*auto intro!: fds-converges intro: less-le-trans*)  
**qed**

**have**  $f - g = 0$  **by** (rule eval-fds-zeroD fds-abs-converges-diff assms \* conv')+  
**thus** ?thesis **by** simp  
**qed**  
**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** –  
**from**  $c$  **have**  $\text{fds-abs-converges } f$  (of-real  $c$ ) **by** (intro fds-abs-converges) simp-all  
**also have** ?this  $\longleftrightarrow$  summable  $(\lambda n. \text{norm } (\text{fds-nth } f n) / \text{real } n \text{ powr } c)$   
**unfolding** fds-abs-converges-def  
**by** (intro summable-cong eventually-mono[OF eventually-gt-at-top[of 0::nat]])  
(auto simp: norm-divide norm-nat-power norm-powr-real-powr)  
**finally have** summable-c: ... .  
  
**note**  $c$   
**also from**  $s$  **have**  $\text{ereal } c \leq \text{ereal } (s \cdot 1)$  **by** simp  
**finally have**  $\text{fds-abs-converges } f s$  **by** (intro fds-abs-converges) auto  
**hence** summable: summable  $(\lambda n. \text{norm } (\text{fds-nth } f n / \text{nat-power } n s))$   
**by** (simp add: fds-abs-converges-def)  
**from** summable-norm-cancel[OF this]  
**have**  $(\lambda n. \text{fds-nth } f n / \text{nat-power } n s)$  sums eval-fds  $f s$   
**by** (simp add: eval-fds-def sums-iff)  
**from** sums-split-initial-segment[OF this, of Suc (Suc 0)]  
**have**  $\text{norm } (\text{eval-fds } f s - \text{fds-nth } f 1) = \text{norm } (\sum n. \text{fds-nth } f (n+2) / \text{nat-power } (n+2) s)$   
**by** (auto simp: sums-iff)  
**also have**  $\dots \leq (\sum n. \text{norm } (\text{fds-nth } f (n+2) / \text{nat-power } (n+2) s))$   
**by** (intro summable-norm summable-ignore-initial-segment summable)  
**also have**  $\dots \leq (\sum n. \text{norm } (\text{fds-nth } f (n+2)) / \text{real } (n+2) \text{ powr } c / 2 \text{ powr } (s \cdot 1 - c))$   
**proof** (intro suminf-le allI)  
**fix**  $n :: \text{nat}$   
**have**  $\text{norm } (\text{fds-nth } f (n + 2) / \text{nat-power } (n + 2) s) =$   
 $\text{norm } (\text{fds-nth } f (n + 2)) / \text{real } (n+2) \text{ powr } c / \text{real } (n+2) \text{ powr } (s \cdot 1 - c)$   
**by** (simp add: field-simps powr-diff norm-divide norm-nat-power)  
**also have**  $\dots \leq \text{norm } (\text{fds-nth } f (n + 2)) / \text{real } (n+2) \text{ powr } c / 2 \text{ powr } (s \cdot 1 - c)$  **using**  $s$   
**by** (intro divide-left-mono divide-nonneg-pos powr-mono2 mult-pos-pos) simp-all  
**finally show**  $\text{norm } (\text{fds-nth } f (n + 2) / \text{nat-power } (n + 2) s) \leq \dots$  .  
**qed** (intro summable-ignore-initial-segment summable summable-divide summable-c)+

**also have**  $\dots = (\sum n. \text{norm } (f_{ds-nth} f (n+2)) / \text{real } (n+2) \text{ powr } c) / 2 \text{ powr } (s \cdot 1 - c)$   
**by** *(intro suminf-divide summable-ignore-initial-segment summable-c)*  
**also have**  $\dots = B / 2 \text{ powr } (s \cdot 1)$  **by** *(simp add: B-def powr-diff)*  
**finally show** *?thesis* .  
**qed**

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

**assumes** *conv-abscissa*  $(f :: 'a :: \text{dirichlet-series } f_{ds}) \neq \infty$   
**assumes** *lim: filterlim*  $(\lambda x. S x \cdot 1)$  *at-top*  $F$   
**shows**  $((\lambda x. \text{eval-fds } f (S x)) \longrightarrow f_{ds-nth} f 1) F$   
**proof** –  
**from** *assms(1)* **have** *abs-conv-abscissa*  $f < \infty$   
**using** *abs-conv-le-conv-abscissa-plus-1*[*of f*] **by** *auto*  
**from** *ereal-dense2*[*OF this*] **obtain**  $c$  **where**  $c: \text{abs-conv-abscissa } f < \text{ereal } c$  **by** *auto*  
**define**  $B$  **where**  $B = (\sum n. \text{norm } (f_{ds-nth} f (n+2)) / \text{real } (n+2) \text{ powr } c) * 2 \text{ powr } c$

**have**  $*$ : *norm*  $(\text{eval-fds } f s - f_{ds-nth} f 1) \leq B / 2 \text{ powr } (s \cdot 1)$  **if**  $s: s \cdot 1 \geq c$   
**for**  $s$   
**using** *eval-fds-at-top-tail-bound*[*of f c s*] **that**  $c$  **by** *(simp add: B-def)*  
**moreover from** *lim* **have** *eventually*  $(\lambda x. S x \cdot 1 \geq c) F$  **by** *(auto simp: filter-lim-at-top)*  
**ultimately have** *eventually*  $(\lambda x. \text{norm } (\text{eval-fds } f (S x) - f_{ds-nth} f 1) \leq B / 2 \text{ powr } (S x \cdot 1)) F$  **by** *(auto elim!: eventually-mono)*  
**moreover have**  $((\lambda x. B / 2 \text{ powr } (S x \cdot 1)) \longrightarrow 0) F$   
**using** *filterlim-tendsto-pos-mult-at-top*[*OF tendsto-const*][*of ln 2*] - *lim*  
**by** *(intro real-tendsto-divide-at-top*[*OF tendsto-const*])  
*(auto simp: powr-def mult-ac intro!: filterlim-compose*[*OF exp-at-top*])  
**ultimately have**  $((\lambda x. \text{eval-fds } f (S x) - f_{ds-nth} f 1) \longrightarrow 0) F$   
**by** *(rule Lim-null-comparison)*  
**thus** *?thesis* **by** *(subst (asm) Lim-null [symmetric])*  
**qed**

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

**assumes** *conv-abscissa*  $(f :: \text{complex } f_{ds}) \neq \infty$   
**shows** *uniform-limit UNIV*  $(\lambda \sigma t. \text{eval-fds } f (\text{of-real } \sigma + \text{of-real } t * i)) (\lambda \cdot . f_{ds-nth} f 1)$  *at-top*  
**proof** –  
**from** *assms(1)* **have** *abs-conv-abscissa*  $f < \infty$   
**using** *abs-conv-le-conv-abscissa-plus-1*[*of f*] **by** *auto*  
**from** *ereal-dense2*[*OF this*] **obtain**  $c$  **where**  $c: \text{abs-conv-abscissa } f < \text{ereal } c$  **by** *auto*  
**define**  $B$  **where**  $B \equiv (\sum n. \text{norm } (f_{ds-nth} f (n+2)) / \text{real } (n+2) \text{ powr } c) * 2 \text{ powr } c$

**show** *?thesis*  
**unfolding** *uniform-limit-iff*

```

proof safe
  fix  $\varepsilon :: \text{real}$  assume  $\varepsilon > 0$ 
  hence eventually  $(\lambda\sigma. B / 2 \text{ powr } \sigma < \varepsilon)$  at-top
    by real-asymp
  thus eventually  $(\lambda\sigma. \forall t \in \text{UNIV.}$ 
     $\text{dist } (\text{eval-fds } f \text{ (of-real } \sigma + \text{of-real } t * i)) \text{ (fds-nth } f \ 1) < \varepsilon)$  at-top
    using eventually-ge-at-top[of c]
  proof eventually-elim
    case (elim  $\sigma$ )
    show ?case
    proof
      fix  $t :: \text{real}$ 
      have  $\text{dist } (\text{eval-fds } f \text{ (of-real } \sigma + \text{of-real } t * i)) \text{ (fds-nth } f \ 1) \leq B / 2 \text{ powr } \sigma$ 
        using eval-fds-at-top-tail-bound[of f c of-real } \sigma + \text{of-real } t * i] elim c
        by (simp add: dist-norm B-def)
      also have  $\dots < \varepsilon$  by fact
      finally show  $\text{dist } (\text{eval-fds } f \text{ (of-real } \sigma + \text{of-real } t * i)) \text{ (fds-nth } f \ 1) < \varepsilon .$ 
    qed
  qed
qed
qed

```

```

lemma tendsto-eval-fds-Re-going-to-at-top:
  assumes conv-abscissa  $(f :: 'a :: \text{dirichlet-series fds}) \neq \infty$ 
  shows  $((\lambda s. \text{eval-fds } f \ s) \longrightarrow \text{fds-nth } f \ 1) ((\lambda s. s \cdot 1) \text{ going-to at-top})$ 
  using assms by (rule tendsto-eval-fds-Re-at-top) auto

```

```

lemma tendsto-eval-fds-Re-going-to-at-top':
  assumes conv-abscissa  $(f :: \text{complex fds}) \neq \infty$ 
  shows  $((\lambda s. \text{eval-fds } f \ s) \longrightarrow \text{fds-nth } f \ 1) (\text{Re going-to at-top})$ 
  using assms by (rule tendsto-eval-fds-Re-at-top) auto

```

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 conv-abscissa  $f < \infty$   $f \neq 0$ 
  shows 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 –
    from ereal-dense2[OF assms(1)] obtain  $c$  where  $c: \text{conv-abscissa } f < \text{ereal } c$ 
    by auto
    have eventually  $(\lambda s::'a. s \cdot 1 > c) ((\lambda s. s \cdot 1) \text{ going-to at-top})$ 
      using eventually-gt-at-top[of c] by auto
    hence eventually  $(\lambda s. \text{fds-converges } f \ s) ((\lambda s. s \cdot 1) \text{ going-to at-top})$ 
      by eventually-elim (use c in <auto intro!>: fds-converges simp: less-le-trans)
    moreover have eventually  $(\lambda s. \text{eval-fds } f \ s \neq 0) ((\lambda s. s \cdot 1) \text{ going-to at-top})$ 
      using eval-fds-zeroD[OF assms(1)] assms(2) by (auto simp: frequently-def)
    ultimately show ?thesis by (rule eventually-conj)

```

qed

## 12.6 Normed series

**lemma** *fds-converges-norm-iff* [simp]:  
fixes  $s :: 'a :: \{\text{nat-power-normed-field}, \text{banach}\}$   
shows  $\text{fds-converges } (\text{fds-norm } f) (s \cdot 1) \longleftrightarrow \text{fds-abs-converges } f s$   
unfolding *fds-converges-def* *fds-abs-converges-def*  
by (rule *summable-cong* [OF *eventually-mono* [OF *eventually-gt-at-top* [of 0]]])  
(simp add: *fds-abs-converges-def* *fds-norm-def* *fds-nth-fds'* *norm-divide* *norm-nat-power*)

**lemma** *fds-abs-converges-norm-iff* [simp]:  
fixes  $s :: 'a :: \{\text{nat-power-normed-field}, \text{banach}\}$   
shows  $\text{fds-abs-converges } (\text{fds-norm } f) (s \cdot 1) \longleftrightarrow \text{fds-abs-converges } f s$   
unfolding *fds-abs-converges-def*  
by (rule *summable-cong* [OF *eventually-mono* [OF *eventually-gt-at-top* [of 0]]])  
(simp add: *fds-abs-converges-def* *fds-norm-def* *fds-nth-fds'* *norm-divide* *norm-nat-power*)

**lemma** *fds-converges-norm-iff'*:  
fixes  $f :: 'a :: \{\text{nat-power-normed-field}, \text{banach}\}$  *fds*  
shows  $\text{fds-converges } (\text{fds-norm } f) s \longleftrightarrow \text{fds-abs-converges } f (\text{of-real } s)$   
unfolding *fds-converges-def* *fds-abs-converges-def*  
by (rule *summable-cong* [OF *eventually-mono* [OF *eventually-gt-at-top* [of 0]]])  
(simp add: *fds-abs-converges-def* *fds-norm-def* *fds-nth-fds'* *norm-divide* *norm-nat-power*)

**lemma** *fds-abs-converges-norm-iff'*:  
fixes  $f :: 'a :: \{\text{nat-power-normed-field}, \text{banach}\}$  *fds*  
shows  $\text{fds-abs-converges } (\text{fds-norm } f) s \longleftrightarrow \text{fds-abs-converges } f (\text{of-real } s)$   
unfolding *fds-abs-converges-def*  
by (rule *summable-cong* [OF *eventually-mono* [OF *eventually-gt-at-top* [of 0]]])  
(simp add: *fds-abs-converges-def* *fds-norm-def* *fds-nth-fds'* *norm-divide* *norm-nat-power*)

**lemma** *abs-conv-abscissa-norm* [simp]:  
fixes  $f :: 'a :: \text{dirichlet-series } \text{fds}$   
shows  $\text{abs-conv-abscissa } (\text{fds-norm } f) = \text{abs-conv-abscissa } f$   
**proof** (rule *antisym*)  
show  $\text{abs-conv-abscissa } f \leq \text{abs-conv-abscissa } (\text{fds-norm } f)$   
**proof** (rule *abs-conv-abscissa-leI-weak*)  
fix  $x$  **assume**  $\text{abs-conv-abscissa } (\text{fds-norm } f) < \text{ereal } x$   
hence  $\text{fds-abs-converges } (\text{fds-norm } f) (\text{of-real } x)$  **by** (*intro* *fds-abs-converges*)  
*auto*  
thus  $\text{fds-abs-converges } f (\text{of-real } x)$  **by** (*simp* add: *fds-abs-converges-norm-iff'*)  
**qed**  
**qed** (*auto* *intro!*: *abs-conv-abscissa-leI-weak* *simp*: *fds-abs-converges-norm-iff'* *fds-abs-converges*)

**lemma** *conv-abscissa-norm* [simp]:  
fixes  $f :: 'a :: \text{dirichlet-series } \text{fds}$   
shows  $\text{conv-abscissa } (\text{fds-norm } f) = \text{abs-conv-abscissa } f$   
**proof** (rule *antisym*)

**show**  $\text{abs-conv-abcissa } f \leq \text{conv-abcissa } (\text{fds-norm } f)$   
**proof** (rule *abs-conv-abcissa-leI-weak*)  
    **fix**  $x$  **assume**  $\text{conv-abcissa } (\text{fds-norm } f) < \text{ereal } x$   
    **hence**  $\text{fds-converges } (\text{fds-norm } f) (\text{of-real } x)$  **by** (intro *fds-converges*) *auto*  
    **thus**  $\text{fds-abs-converges } f (\text{of-real } x)$  **by** (simp add: *fds-converges-norm-iff'*)  
**qed**  
**qed** (auto intro!: *conv-abcissa-leI-weak* simp: *fds-abs-converges*)

**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**  $\text{fds-abs-converges-norm-mult: fds-abs-converges } (\text{fds-norm } (f * g)) s$   
    **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** –  
    **show**  $\text{conv: fds-abs-converges } (\text{fds-norm } (f * g)) s$  **unfolding** *fds-abs-converges-def*  
    **proof** (rule *summable-comparison-test-ev*)  
    **have**  $\text{fds-abs-converges } (\text{fds-norm } f * \text{fds-norm } g) s$  **by** (rule *fds-abs-converges-mult* *assms*)  
    **thus**  $\text{summable } (\lambda n. \text{norm } (\text{fds-nth } (\text{fds-norm } f * \text{fds-norm } g) n) / \text{nat-power } n s)$   
    **by** (simp add: *fds-abs-converges-def*)  
**qed** (auto intro!: *always-eventually divide-right-mono order.trans[OF fds-nth-norm-mult-le]*)

        (simp: *norm-divide*)  
**have**  $\text{conv': fds-abs-converges } (\text{fds-norm } f * \text{fds-norm } g) s$   
    **by** (intro *fds-abs-converges-mult* *assms*)  
**hence**  $\text{eval-fds } (\text{fds-norm } (f * g)) s \leq \text{eval-fds } (\text{fds-norm } f * \text{fds-norm } g) s$   
    **using** *conv* **unfolding** *eval-fds-def* *fds-abs-converges-def* *norm-divide*  
    **by** (intro *suminf-le allI divide-right-mono*) (simp-all add: *norm-mult fds-nth-norm-mult-le*)  
**also have**  $\dots = \text{eval-fds } (\text{fds-norm } f) s * \text{eval-fds } (\text{fds-norm } g) s$   
    **by** (intro *eval-fds-mult* *assms*)  
**finally show**  $\text{eval-fds } (\text{fds-norm } (f * g)) s \leq \text{eval-fds } (\text{fds-norm } f) s * \text{eval-fds } (\text{fds-norm } g) s$ .  
**qed**

**lemma** *eval-fds-norm-nonneg*:  
    **assumes**  $\text{fds-abs-converges } (\text{fds-norm } f) s$   
    **shows**  $\text{eval-fds } (\text{fds-norm } f) s \geq 0$   
    **using** *assms* **unfolding** *eval-fds-def* *fds-abs-converges-def*  
    **by** (intro *suminf-nonneg*) *auto*

**lemma**  
    **fixes**  $f :: 'a :: \text{dirichlet-series fds}$   
    **assumes**  $\text{fds-abs-converges } (\text{fds-norm } f) s$   
    **shows**  $\text{fds-abs-converges-norm-power: fds-abs-converges } (\text{fds-norm } (f \wedge n)) s$   
    **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** –  
**show** \*:  $\text{fds-abs-converges } (\text{fds-norm } (f \wedge n)) \text{ } s \text{ for } n$   
**by** (*induction n*) (*auto intro!*:  $\text{fds-abs-converges-norm-mult } \text{assms}$ )  
**show**  $\text{eval-fds } (\text{fds-norm } (f \wedge n)) \text{ } s \leq \text{eval-fds } (\text{fds-norm } f) \text{ } s \wedge n$   
**by** (*induction n*) (*auto intro!*:  $\text{order.trans}[OF \text{eval-fds-norm-mult-le}] \text{assms} *$   
 $\text{mult-left-mono eval-fds-norm-nonneg}$ )  
**qed**

## 12.7 Logarithms of Dirichlet series

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

**lemma** *eval-fds-log-deriv*:  
**fixes**  $s :: 'a :: \text{dirichlet-series}$   
**assumes**  $\text{fds-nth } f \text{ } 1 \neq 0$   $s \cdot 1 > \text{abs-conv-abscissa } f$   
 $s \cdot 1 > \text{abs-conv-abscissa } (\text{fds-deriv } f / f)$   
**assumes**  $\text{eval-fds } f \text{ } s \neq 0$   
**shows**  $\text{eval-fds } (\text{fds-deriv } f / f) \text{ } s = \text{eval-fds } (\text{fds-deriv } f) \text{ } s / \text{eval-fds } f \text{ } s$   
**proof** –  
**have**  $\text{eval-fds } (\text{fds-deriv } f / f * f) \text{ } s = \text{eval-fds } (\text{fds-deriv } f / f) \text{ } s * \text{eval-fds } f \text{ } s$   
**using** *assms* **by** (*intro eval-fds-mult fds-abs-converges*) *auto*  
**also have**  $\text{fds-deriv } f / f * f = \text{fds-deriv } f * (f * \text{inverse } f)$   
**by** (*simp add: divide-fds-def algebra-simps*)  
**also have**  $f * \text{inverse } f = 1$  **using** *assms* **by** (*intro fds-right-inverse*)  
**finally show** *?thesis* **using** *assms* **by** *simp*  
**qed**

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**  $\text{nz: } \bigwedge s. \text{Re } s > s0 \implies \text{eval-fds } f \text{ } s \neq 0$   $\text{fds-nth } f \text{ } 1 \neq 0$   
**assumes**  $l: \text{exp } l = \text{fds-nth } f \text{ } 1$  ( $(g \circ \text{of-real}) \longrightarrow l$ ) *at-top*  
**assumes**  $g: \bigwedge s. \text{Re } s > s0 \implies \text{exp } (g \text{ } s) = \text{eval-fds } f \text{ } s$   
**assumes** *holo-g*:  $g$  *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**  $\text{eval-fds } (\text{fds-ln } l \text{ } f) \text{ } s = g \text{ } s$   
**proof** –  
**let**  $?s0 = \text{abs-conv-abscissa } f$  **and**  $?s1 = \text{abs-conv-abscissa } (\text{inverse } f)$   
**let**  $?h = \lambda s. \text{eval-fds } (\text{fds-ln } l \text{ } f) \text{ } s - g \text{ } s$   
**let**  $?A = \{s. \text{Re } s > s0\}$   
**have** *open-A*: *open*  $?A$  **by** (*cases s0*) (*auto simp: open-halfspace-Re-gt*)  
  
**have**  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$  **by** (*rule conv-le-abs-conv-abscissa*)  
**moreover from** *assms* **have**  $\dots \neq \infty$  **by** *auto*  
**ultimately have**  $\text{conv-abscissa } f \neq \infty$  **by** *auto*

**have**  $\text{conv-abscissa } (f\text{ds-ln } l f) \leq \text{abs-conv-abscissa } (f\text{ds-ln } l f)$   
**by** (rule *conv-le-abs-conv-abscissa*)  
**also have**  $\dots \leq \text{abs-conv-abscissa } (f\text{ds-deriv } f / f)$   
**unfolding** *fds-ln-def* **by** (*simp add: abs-conv-abscissa-integral*)  
**finally have**  $\text{conv-abscissa } (f\text{ds-ln } l f) \neq \infty$   
**using** *assms* **by** (*auto simp: max-def abs-conv-abscissa-deriv split: if-splits*)

**have** *deriv-g* [*derivative-intros*]:  
*(g has-field-derivative eval-fds (fds-deriv f) s / eval-fds f s) (at s within B)*  
**if**  $s: \text{Re } s > s0$  **for**  $s \in B$

**proof** –

**have**  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$  **by** (rule *conv-le-abs-conv-abscissa*)  
**also have**  $\dots \leq s0$  **using** *assms* **by** *simp*  
**also have**  $\dots < \text{Re } s$  **by** *fact*  
**finally have**  $s': \text{Re } s > \text{conv-abscissa } f$  .

**have** *deriv-g*: (*g has-field-derivative deriv g s*) (*at s*)  
**using** *holomorphic-derivI*[*OF holo-g open-A, of s*]  $s$   
**by** (*auto simp: at-within-open*[*OF - open-A*])  
**have** ( $(\lambda s. \text{exp } (g s))$  *has-field-derivative eval-fds f s \* deriv g s*) (*at s*) (**is** *?P*)  
**by** (rule *derivative-eq-intros deriv-g s*) + (*insert s, simp-all add: g*)  
**also from**  $s$  **have** *ev: eventually* ( $\lambda t. t \in ?A$ ) (*nhds s*)  
**by** (*intro eventually-nhds-in-open open-A*) *auto*  
**have**  $?P \iff (\text{eval-fds } f \text{ has-field-derivative eval-fds f s * deriv g s})$  (*at s*)  
**by** (*intro DERIV-cong-ev refl eventually-mono*[*OF ev*]) (*auto simp: g*)  
**finally have** ( $\text{eval-fds } f \text{ has-field-derivative eval-fds f s * deriv g s}$ ) (*at s*) .  
**moreover have** ( $\text{eval-fds } f \text{ has-field-derivative eval-fds } (f\text{ds-deriv } f) s$ ) (*at s*)  
**using**  $s'$  *assms* **by** (*intro derivative-intros*) *auto*  
**ultimately have**  $\text{eval-fds } f s * \text{deriv } g s = \text{eval-fds } (f\text{ds-deriv } f) s$   
**by** (rule *DERIV-unique*)  
**hence**  $\text{deriv } g s = \text{eval-fds } (f\text{ds-deriv } f) s / \text{eval-fds } f s$   
**using**  $s \text{ nz}$  **by** (*simp add: field-simps*)  
**with** *deriv-g* **show** *?thesis* **by** (*auto intro: has-field-derivative-at-within*)

**qed**

**have**  $\exists c. \forall z \in \{z. \text{Re } z > s0\}. ?h z = c$   
**proof** (rule *has-field-derivative-zero-constant, goal-cases*)

**case** 1

**show** *?case* **using** *convex-halfspace-gt*[*of - 1::complex*]  
**by** (*cases s0*) *auto*

**next**

**case** (2  $z$ )

**have**  $\text{conv-abscissa } (f\text{ds-ln } l f) \leq \text{abs-conv-abscissa } (f\text{ds-ln } l f)$   
**by** (rule *conv-le-abs-conv-abscissa*)  
**also have**  $\dots \leq \text{abs-conv-abscissa } (f\text{ds-deriv } f / f)$   
**by** (*simp add: abs-conv-abscissa-ln*)  
**also have**  $\dots < \text{Re } z$  **using** 2 *assms* **by** (*auto simp: abs-conv-abscissa-deriv*)  
**finally have**  $s1: \text{conv-abscissa } (f\text{ds-ln } l f) < \text{ereal } (\text{Re } z)$  .

**have**  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$   
**by** (rule conv-le-abs-conv-abscissa)  
**also have**  $\dots < \text{Re } z$  **using** 2 *assms* **by** *auto*  
**finally have**  $s_2: \text{conv-abscissa } f < \text{ereal } (\text{Re } z)$  .

**from**  $l$  **have**  $\text{fds-nth } f \ 1 \neq 0$  **by** *auto*  
**with** 2 *assms* **have**  $*$ :  $\text{eval-fds } (\text{fds-deriv } f / f) \ z = \text{eval-fds } (\text{fds-deriv } f) \ z /$   
( $\text{eval-fds } f \ z$ )  
**by** (*auto simp: eval-fds-log-deriv*)  
**have**  $\text{eval-fds } f \ z \neq 0$  **using** 2 *assms* **by** *auto*  
**show**  $?case$  **using**  $s_1 \ s_2 \ 2 \ nz$   
**by** (*auto intro!: derivative-eq-intros simp: \* field-simps*)  
**qed**  
**then obtain**  $c$  **where**  $c: \bigwedge z. \text{Re } z > s_0 \implies ?h \ z = c$  **by** *blast*

**have** ( $\text{at-top} :: \text{real filter}$ )  $\neq \text{bot}$  **by** *simp*  
**moreover from** *assms* **have**  $s_0 \neq \infty$  **by** *auto*  
**have** *eventually*  $(\lambda x. c = (?h \circ \text{of-real}) \ x)$  *at-top*  
**using** *eventually-gt-ereal-at-top*[ $OF \ \langle s_0 \neq \infty \rangle$ ] **by** *eventually-elim (simp add:*  
*c)*  
**hence**  $(?h \circ \text{of-real}) \longrightarrow c$  *at-top*  
**by** (*force intro: Lim-transform-eventually*)  
**moreover have**  $(?h \circ \text{of-real}) \longrightarrow \text{fds-nth } (\text{fds-ln } l \ f) \ 1 - l$  *at-top*  
**using**  $\langle \text{conv-abscissa } (\text{fds-ln } l \ f) \neq \infty \rangle$  **and**  $l$  **unfolding** *o-def*  
**by** (*intro tendsto-intros tendsto-eval-fds-Re-at-top (auto simp: filterlim-ident)*)  
**ultimately have**  $c = \text{fds-nth } (\text{fds-ln } l \ f) \ 1 - l$   
**by** (*rule tendsto-unique*)  
**with**  $c[OF \ \langle \text{Re } s > s_0 \rangle]$  **and**  $l$  **and**  $nz$  **show**  $?thesis$   
**by** (*simp add: exp-minus field-simps*)  
**qed**

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**  $s_0 :: \text{ereal}$   
**assumes**  $\text{ereal } (\text{Re } s) > s_0$   
**assumes**  $s_0 \geq \text{abs-conv-abscissa } f$  **and**  $s_0 \geq \text{abs-conv-abscissa } (\text{fds-deriv } f / f)$   
**and**  $nz: \bigwedge s. \text{Re } s > s_0 \implies \text{eval-fds } f \ s \neq 0 \ \text{fds-nth } f \ 1 \neq 0$   
**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** –  
**let**  $?s_0 = \text{abs-conv-abscissa } f$  **and**  $?s_1 = \text{abs-conv-abscissa } (\text{inverse } f)$   
**let**  $?h = \lambda s. \text{eval-fds } f \ s * \text{exp } (-\text{eval-fds } (\text{fds-ln } l \ f) \ s)$   
  
**have**  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$  **by** (*rule conv-le-abs-conv-abscissa*)  
**moreover from** *assms* **have**  $\dots \neq \infty$  **by** *auto*  
**ultimately have**  $\text{conv-abscissa } f \neq \infty$  **by** *auto*

**have**  $\text{conv-abscissa } (f\text{ds-ln } l f) \leq \text{abs-conv-abscissa } (f\text{ds-ln } l f)$   
**by** (rule *conv-le-abs-conv-abscissa*)  
**also have**  $\dots \leq \text{abs-conv-abscissa } (f\text{ds-deriv } f / f)$   
**unfolding** *fds-ln-def* **by** (simp add: *abs-conv-abscissa-integral*)  
**finally have**  $\text{conv-abscissa } (f\text{ds-ln } l f) \neq \infty$   
**using** *assms* **by** (auto simp: *max-def abs-conv-abscissa-deriv split: if-splits*)

**have**  $\exists c. \forall z \in \{z. \text{Re } z > s0\}. ?h z = c$   
**proof** (rule *has-field-derivative-zero-constant, goal-cases*)  
**case** 1  
**show** *?case* **using** *convex-halfspace-gt[of - 1::complex]*  
**by** (cases *s0*) auto  
**next**  
**case** (2 *z*)  
**have**  $\text{conv-abscissa } (f\text{ds-ln } l f) \leq \text{abs-conv-abscissa } (f\text{ds-ln } l f)$   
**by** (rule *conv-le-abs-conv-abscissa*)  
**also have**  $\dots \leq \text{abs-conv-abscissa } (f\text{ds-deriv } f / f)$   
**unfolding** *fds-ln-def* **by** (simp add: *abs-conv-abscissa-integral*)  
**also have**  $\dots < \text{Re } z$  **using** 2 *assms* **by** (auto simp: *abs-conv-abscissa-deriv*)  
**finally have** *s1*:  $\text{conv-abscissa } (f\text{ds-ln } l f) < \text{ereal } (\text{Re } z)$  .

**have**  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$   
**by** (rule *conv-le-abs-conv-abscissa*)  
**also have**  $\dots < \text{Re } z$  **using** 2 *assms* **by** auto  
**finally have** *s2*:  $\text{conv-abscissa } f < \text{ereal } (\text{Re } z)$  .

**from** *l* **have** *fds-nth* *f* 1  $\neq 0$  **by** auto  
**with** 2 *assms* **have**  $*$ :  $\text{eval-fds } (f\text{ds-deriv } f / f) z = \text{eval-fds } (f\text{ds-deriv } f) z /$   
(*eval-fds* *f* *z*)  
**by** (subst *eval-fds-log-deriv*) auto  
**have**  $\text{eval-fds } f z \neq 0$  **using** 2 *assms* **by** auto  
**thus** *?case* **using** *s1 s2*  
**by** (auto intro!: *derivative-eq-intros simp: \**)

**qed**  
**then obtain** *c* **where**  $c: \bigwedge z. \text{Re } z > s0 \implies ?h z = c$  **by** *blast*

**have** (*at-top* :: *real filter*)  $\neq \text{bot}$  **by** *simp*  
**moreover from** *assms* **have** *s0*  $\neq \infty$  **by** auto  
**have** *eventually* ( $\lambda x. c = (?h \circ \text{of-real}) x$ ) *at-top*  
**using** *eventually-gt-ereal-at-top[OF <s0  $\neq$   $\infty$ >]* **by** *eventually-elim (simp add:*  
*c)*  
**hence** ((*?h*  $\circ$  *of-real*)  $\longrightarrow c$ ) *at-top*  
**by** (force intro: *Lim-transform-eventually*)  
**moreover have** ((*?h*  $\circ$  *of-real*)  $\longrightarrow f\text{ds-nth } f 1 * \text{exp } (-f\text{ds-nth } (f\text{ds-ln } l f) 1)$ )  
*at-top*  
**unfolding** *o-def* **using**  $\langle \text{conv-abscissa } (f\text{ds-ln } l f) \neq \infty \rangle$  **and**  $\langle \text{conv-abscissa } f$   
 $\neq \infty \rangle$   
**by** (intro *tendsto-intros tendsto-eval-fds-Re-at-top*) (auto simp: *filterlim-ident*)

ultimately have  $c = \text{fds-nth } f \ 1 * \exp (-\text{fds-nth } (\text{fds-ln } l \ f) \ 1)$   
 by (rule tendsto-unique)  
 with  $c[OF \langle Re \ s > \ s0 \rangle]$  and  $l$  and  $nz$  show ?thesis  
 by (simp add: exp-minus field-simps)  
 qed

**lemma** *fds-ln-completely-multiplicative*:  
 fixes  $f :: 'a :: \text{dirichlet-series } \text{fds}$   
 assumes *completely-multiplicative-function* ( $\text{fds-nth } f$ )  
 assumes  $\text{fds-nth } f \ 1 \neq 0$   
 shows  $\text{fds-ln } l \ f = \text{fds } (\lambda n. \text{if } n = 1 \text{ then } l \ \text{else } \text{fds-nth } f \ n * \text{mangoldt } n /_R \text{ln } n)$   
**proof** –  
 have  $\text{fds-ln } l \ f = \text{fds-integral } l \ (\text{fds-deriv } f / f)$   
 by (simp add: fds-ln-def)  
 also have  $\text{fds-deriv } f = -\text{fds } (\lambda n. \text{fds-nth } f \ n * \text{mangoldt } n) * f$   
 by (intro completely-multiplicative-fds-deriv' assms)  
 also have  $\dots / f = -\text{fds } (\lambda n. \text{fds-nth } f \ n * \text{mangoldt } n) * (f * \text{inverse } f)$   
 by (simp add: divide-fds-def)  
 also from assms have  $f * \text{inverse } f = 1$   
 by (simp add: fds-right-inverse)  
 also have  $\text{fds-integral } l \ (-\text{fds } (\lambda n. \text{fds-nth } f \ n * \text{mangoldt } n) * 1) =$   
 $\text{fds } (\lambda n. \text{if } n = 1 \text{ then } l \ \text{else } \text{fds-nth } f \ n * \text{mangoldt } n /_R \text{ln } n)$   
 by (simp add: fds-integral-def cong: if-cong)  
 finally show ?thesis .  
 qed

**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  $(\lambda(p,k). (\text{fds-nth } f \ p / \text{nat-power } p \ s) \wedge \text{Suc } k * g \ (p \wedge \text{Suc } k) / \text{of-nat } (\text{Suc } k))$   
 $\text{abs-summable-on } (\{p. \text{prime } p\} \times \text{UNIV})$  (is ?th1)  
 and  $\text{eval-fds } h \ s = l * g \ 1 + (\sum_{a(p,k) \in \{p. \text{prime } p\} \times \text{UNIV}} (\text{fds-nth } f \ p / \text{nat-power } p \ s) \wedge \text{Suc } k * g \ (p \wedge \text{Suc } k) / \text{of-nat } (\text{Suc } k))$   
 (is ?th2)  
**proof** –  
 let  $?P = \{p :: \text{nat}. \text{prime } p\}$   
 interpret  $f$ : *completely-multiplicative-function*  $\text{fds-nth } f$  by fact  
 from assms have  $*$ :  $(\lambda n. \text{fds-nth } h \ n / \text{nat-power } n \ s) \text{abs-summable-on } \text{UNIV}$   
 by (auto simp: abs-summable-on-nat-iff' fds-abs-converges-def)  
 have eq:  $h = \text{fds } (\lambda n. \text{if } n = 1 \text{ then } l * g \ 1 \ \text{else } \text{fds-nth } f \ n * g \ n * \text{mangoldt } n /_R \text{ln } (\text{real } n))$   
 using *fds-ln-completely-multiplicative* [OF assms(3), of  $l$ ]  
 by (simp add: h-def fds-eq-iff)

**note** \*

**also have**  $(\lambda n. \text{fds-nth } h \ n \ / \ \text{nat-power } n \ s) \text{ abs-summable-on } UNIV \longleftrightarrow$   
 $(\lambda x. \text{if } x = \text{Suc } 0 \text{ then } l * g \ 1 \ \text{else } \text{fds-nth } f \ x * g \ x * \text{mangoldt } x \ /_R \ \ln$   
 $(\text{real } x) \ /$   
 $\text{nat-power } x \ s) \text{ abs-summable-on } \{1\} \cup \text{Collect primepow}$   
**using** *eq* **by**  $(\text{intro abs-summable-on-cong-neutral}) \ (\text{auto simp: fds-nth-fds mangoldt-def})$   
**finally have** *sum1*:  $(\lambda x. \text{if } x = \text{Suc } 0 \text{ then } l * g \ 1 \ \text{else}$   
 $\text{fds-nth } f \ x * g \ x * \text{mangoldt } x \ /_R \ \ln \ (\text{real } x) \ / \ \text{nat-power } x \ s)$   
 $\text{abs-summable-on } \text{Collect primepow}$   
**by**  $(\text{rule abs-summable-on-subset}) \ \text{auto}$   
**also have** *?this*  $\longleftrightarrow (\lambda x. \text{fds-nth } f \ x * g \ x * \text{mangoldt } x \ /_R \ \ln \ (\text{real } x) \ / \ \text{nat-power}$   
 $x \ s)$   
 $\text{abs-summable-on } \text{Collect primepow}$   
**by**  $(\text{intro abs-summable-on-cong}) \ (\text{insert primepow-gt-Suc-0}, \ \text{auto})$   
**also have**  $\dots \longleftrightarrow (\lambda(p,k). \text{fds-nth } f \ (p \ \wedge \ \text{Suc } k) * g \ (p \ \wedge \ \text{Suc } k) * \text{mangoldt } (p$   
 $\wedge \ \text{Suc } k)$   
 $\ /_R \ \ln \ (\text{real } (p \ \wedge \ \text{Suc } k)) \ / \ \text{nat-power } (p \ \wedge \ \text{Suc } k) \ s) \ \text{abs-summable-on}$   
 $(?P \times UNIV)$   
**using** *bij-betw-primewords unfolding case-prod-unfold*  
**by**  $(\text{intro abs-summable-on-reindex-bij-betw } [\text{symmetric}])$   
**also have**  $\dots \longleftrightarrow \text{?th1}$   
**by**  $(\text{intro abs-summable-on-cong})$   
 $(\text{auto simp: f.mult f.power mangoldt-def aprimedivisor-prime-power ln-realpow}$   
 $\text{prime-gt-0-nat}$   
 $\text{nat-power-power-left divide-simps scaleR-conv-of-real simp del: power-Suc})$   
**finally show** *?th1* .

**have** *eval-fds*  $h \ s = (\sum_a n. \text{fds-nth } h \ n \ / \ \text{nat-power } n \ s)$   
**using** *\** **unfolding** *eval-fds-def* **by**  $(\text{subst infsetsum-nat'}) \ \text{auto}$   
**also have**  $\dots = (\sum_a n \in \{1\} \cup \{n. \text{primepow } n\}.$   
 $\text{if } n = 1 \text{ then } l * g \ 1 \ \text{else } \text{fds-nth } f \ n * g \ n * \text{mangoldt } n \ /_R \ \ln \ (\text{real } n) \ /$   
 $\text{nat-power } n \ s)$   
**by**  $(\text{intro infsetsum-cong-neutral}) \ (\text{auto simp: eq fds-nth-fds mangoldt-def})$   
**also have**  $\dots = l * g \ 1 + (\sum_a n \mid \text{primepow } n.$   
 $\text{if } n = 1 \text{ then } l * g \ 1 \ \text{else } \text{fds-nth } f \ n * g \ n * \text{mangoldt } n \ /_R \ \ln \ (\text{real } n) \ /$   
 $\text{nat-power } n \ s)$   
 $(\text{is } - = - + ?x) \ \text{using } \text{sum1 primepow-gt-Suc-0} \ \text{by } (\text{subst infsetsum-Un-disjoint})$   
 $\text{auto}$   
**also have** *?x* =  
 $(\sum_a n \in \text{Collect primepow}. \text{fds-nth } f \ n * g \ n * \text{mangoldt } n \ /_R \ \ln \ (\text{real } n) \ /$   
 $\text{nat-power } n \ s)$   
 $(\text{is } - = \text{infsetsum } ?f -) \ \text{by } (\text{intro infsetsum-cong refl}) \ (\text{insert primepow-gt-Suc-0},$   
 $\text{auto})$   
**also have**  $\dots = (\sum_a (p,k) \in (?P \times UNIV). \text{fds-nth } f \ (p \ \wedge \ \text{Suc } k) * g \ (p \ \wedge \ \text{Suc } k)$   
 $*$   
 $\text{mangoldt } (p \ \wedge \ \text{Suc } k) \ /_R \ \ln \ (p \ \wedge \ \text{Suc } k) \ / \ \text{nat-power } (p \ \wedge \ \text{Suc } k) \ s)$   
**using** *bij-betw-primewords unfolding case-prod-unfold*  
**by**  $(\text{intro infsetsum-reindex-bij-betw } [\text{symmetric}])$   
**also have**  $\dots = (\sum_a (p,k) \in (?P \times UNIV).$

$(\text{fds-nth } f \ p \ / \ \text{nat-power } p \ s) \wedge \text{Suc } k * g \ (p \wedge \text{Suc } k) \ / \ \text{of-nat}$   
 $(\text{Suc } k)$   
**by**  $(\text{intro infsetsum-cong})$   
 $(\text{auto simp: } f.\text{mult } f.\text{power mangoldt-def aprimedivisor-prime-power ln-realpow}$   
 $\text{prime-gt-0-nat}$   
 $\text{nat-power-power-left divide-simps scaleR-conv-of-real simp del: power-Suc})$   
**finally show**  $?th2$  .  
**qed**

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

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

**proof** –

**from** *assms* **have** *fds-abs-converges*  $(\text{fds-ln } l \ f) \ s$   
**by**  $(\text{intro } \text{fds-abs-converges-ln})$   $(\text{auto intro!: } \text{fds-abs-converges-mult intro: } \text{fds-abs-converges})$   
**hence** *fds-abs-converges*  $(\text{fds } (\lambda n. \text{fds-nth } (\text{fds-ln } l \ f) \ n * 1)) \ s$   
**by** *simp*  
**from** *eval-fds-ln-completely-multiplicative-strong*  $[OF \ \text{this } \text{assms}(1,2)]$  **show**  $?th1$   
 $?th2$   
**by** *simp-all*  
**qed**

## 12.8 Exponential and logarithm

**lemma** *summable-fds-exp-aux:*

**assumes**  $\text{fds-nth } f' \ 1 = (0 :: 'a :: \text{real-normed-algebra-1})$   
**shows** *summable*  $(\lambda k. \text{fds-nth } (f' \wedge k) \ n \ /_{\mathbb{R}} \ \text{fact } k)$

**proof**  $(\text{rule summable-finite})$

**fix**  $k$  **assume**  $k \notin \{..n\}$

**hence**  $n < k$  **by** *simp*

**also have**  $\dots < 2 \wedge k$

**by**  $(\text{rule less-exp})$

**finally have**  $\text{fds-nth } (f' \wedge k) \ n = 0$

**using** *assms* **by**  $(\text{intro } \text{fds-nth-power-eq-0})$  *auto*

**thus**  $\text{fds-nth } (f' \wedge k) \ n \ /_{\mathbb{R}} \ \text{fact } k = 0$  **by** *simp*

**qed** *auto*

**lemma**

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

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

**shows** *fds-abs-converges-exp: fds-abs-converges*  $(\text{fds-exp } f) \ s$

**and** *eval-fds-exp: eval-fds*  $(\text{fds-exp } f) \ s = \text{exp } (\text{eval-fds } f \ s)$

**proof** –

```

have conv: fds-abs-converges (fds-exp f) s and ev: eval-fds (fds-exp f) s = exp
(eval-fds f s)
if fds-abs-converges f s and [simp]: fds-nth f (Suc 0) = 0 for f
proof -
have [simp]: fds (λn. if n = Suc 0 then 0 else fds-nth f n) = f
by (intro fds-eqI) simp-all
have (λ(k,n). fds-nth (f ^ k) n / fact k / nat-power n s) abs-summable-on
(UNIV × {1..})
proof (subst abs-summable-on-Sigma-iff, safe, goal-cases)
case (∃ k)
from that have fds-abs-converges (f ^ k) s by (intro fds-abs-converges-power)
hence (λn. fds-nth (f ^ k) n / nat-power n s * inverse (fact k)) abs-summable-on
{1..}
unfolding fds-abs-converges-altdef by (intro abs-summable-on-cmult-left)
thus ?case by (simp add: field-simps)
next
case 4
show ?case unfolding abs-summable-on-nat-iff'
proof (rule summable-comparison-test-ev[OF always-eventually[OF allI]])
fix k :: nat
from that have *: fds-abs-converges (fds-norm (f ^ k)) (s · 1)
by (auto simp: fds-abs-converges-power)
have (∑a n ∈ {1..}. norm (fds-nth (f ^ k) n / fact k / nat-power n s)) =
(∑a n ∈ {1..}. fds-nth (fds-norm (f ^ k)) n / nat-power n (s · 1) / fact
k)
(is ?S = -) by (intro infsetsum-cong) (simp-all add: norm-divide norm-mult
norm-nat-power)
also have ... = (∑a n ∈ {1..}. fds-nth (fds-norm (f ^ k)) n / nat-power n
(s · 1)) /R fact k
(is - = ?S' /R -) using * unfolding fds-abs-converges-altdef
by (subst infsetsum-cdiv) (auto simp: abs-summable-on-nat-iff scaleR-conv-of-real
divide-simps)
also have ?S' = eval-fds (fds-norm (f ^ k)) (s · 1)
using * unfolding fds-abs-converges-altdef eval-fds-def
by (subst infsetsum-nat) (auto intro!: suminf-cong)
finally have eq: ?S = ... /R fact k .
note eq
also have ?S ≥ 0 by (intro infsetsum-nonneg) auto
hence ?S = norm (norm ?S) by simp
also have eval-fds (fds-norm (f ^ k)) (s · 1) ≤ eval-fds (fds-norm f) (s · 1)
^ k
using that by (intro eval-fds-norm-power-le) auto
finally show norm (norm (∑a n ∈ {1..}. norm (fds-nth (f ^ k) n / fact k /
nat-power n s))) ≤
eval-fds (fds-norm f) (s · 1) ^ k /R fact k
by (simp add: divide-right-mono)
next
from exp-converges[of eval-fds (fds-norm f) (s · 1)]
show summable (λx. eval-fds (fds-norm f) (s · 1) ^ x /R fact x)

```

by (simp add: sums-iff)  
 qed  
 qed auto  
 hence summable:  
 $(\lambda(n,k). \text{fds-nth } (f \wedge k) \ n / \text{fact } k / \text{nat-power } n \ s) \text{ abs-summable-on } \{1..\} \times$   
 UNIV  
 by (subst abs-summable-on-Times-swap) (simp add: case-prod-unfold)

have summable':  $(\lambda k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) \text{ abs-summable-on UNIV for } n$   
 using abs-summable-on-cmult-left[of nat-power n s,  
 OF abs-summable-on-Sigma-project2 [OF summable, of n]] by (cases n  
 = 0) simp-all

have  $(\lambda n. \sum_a k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k / \text{nat-power } n \ s) \text{ abs-summable-on}$   
 $\{1..\}$   
 using summable by (rule abs-summable-on-Sigma-project1') auto  
 also have ?this  $\longleftrightarrow (\lambda n. (\sum k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) * \text{inverse } (\text{nat-power}$   
 $n \ s))$   
 $\text{abs-summable-on } \{1..\}$

proof (intro abs-summable-on-cong refl, goal-cases)  
 case (1 n)  
 hence  $(\sum_a k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k / \text{nat-power } n \ s) =$   
 $(\sum_a k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) * \text{inverse } (\text{nat-power } n \ s)$   
 using summable'[of n]  
 by (subst infsetsum-cmult-left [symmetric]) (auto simp: field-simps)  
 also have  $(\sum_a k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) = (\sum k. \text{fds-nth } (f \wedge k) \ n / \text{fact}$   
 $k)$   
 using summable'[of n] 1 by (intro abs-summable-on-cong refl infsetsum-nat')  
 auto

finally show ?case .  
 qed  
 finally show fds-abs-converges (fds-exp f) s  
 by (simp add: fds-exp-def fds-nth-fds' abs-summable-on-Sigma-iff scaleR-conv-of-real  
 $\text{fds-abs-converges-altdef field-simps})$

have eval-fds (fds-exp f) s =  $(\sum n. (\sum k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) / \text{nat-power}$   
 $n \ s)$   
 by (simp add: fds-exp-def eval-fds-def fds-nth-fds')  
 also have ... =  $(\sum n. (\sum_a k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) / \text{nat-power } n \ s)$   
 proof (intro suminf-cong, goal-cases)  
 case (1 n)  
 show ?case  
 proof (cases n = 0)  
 case False  
 have  $(\sum k. \text{fds-nth } (f \wedge k) \ n / \text{fact } k) = (\sum_a k. \text{fds-nth } (f \wedge k) \ n / \text{fact}$   
 $k)$   
 using summable'[of n] False  
 by (intro infsetsum-nat' [symmetric]) (auto simp: scaleR-conv-of-real

```

field-simps)
  thus ?thesis by simp
qed simp-all
qed
also have ... = (∑a n. (∑a k. fds-nth (f ^ k) n /R fact k) / nat-power n s)
proof (intro infsetsum-nat' [symmetric], goal-cases)
  case 1
  have *: UNIV - {Suc 0..} = {0} by auto
  have (λx. ∑a y. fds-nth (f ^ y) x / fact y / nat-power x s) abs-summable-on
{1..}
  by (intro abs-summable-on-Sigma-project1' [OF summable]) auto
  also have ?this ↔ (λx. (∑a y. fds-nth (f ^ y) x / fact y) * inverse (nat-power
x s))
    abs-summable-on {1..}
  using summable' by (intro abs-summable-on-cong refl, subst infsetsum-cmult-left
[symmetric])
    (auto simp: field-simps)
  also have ... ↔ (λx. (∑a y. fds-nth (f ^ y) x /R fact y) / (nat-power x s))
    abs-summable-on {1..} by (simp add: field-simps scaleR-conv-of-real)
  finally show ?case by (rule abs-summable-on-finite-diff) (use * in auto)
qed
also have ... = (∑a n. (∑a k. fds-nth (f ^ k) n /R fact k * inverse (nat-power
n s)))
  using summable' by (subst infsetsum-cmult-left) (auto simp: field-simps
scaleR-conv-of-real)
  also have ... = (∑a n ∈ {1..}. (∑a k. fds-nth (f ^ k) n /R fact k * inverse
(nat-power n s)))
  by (intro infsetsum-cong-neutral) (auto simp: Suc-le-eq)
  also have ... = (∑a k. ∑a n ∈ {1..}. fds-nth (f ^ k) n / nat-power n s /R fact
k) using summable
  by (subst infsetsum-swap) (auto simp: field-simps scaleR-conv-of-real case-prod-unfold)
  also have ... = (∑a k. (∑a n ∈ {1..}. fds-nth (f ^ k) n / nat-power n s) /R
fact k)
  by (subst infsetsum-scaleR-right) simp
  also have ... = (∑a k. eval-fds f s ^ k /R fact k)
proof (intro infsetsum-cong refl, goal-cases)
  case (1 k)
  have *: fds-abs-converges (f ^ k) s by (intro fds-abs-converges-power that)
  have (∑a n ∈ {1..}. fds-nth (f ^ k) n / nat-power n s) =
    (∑a n. fds-nth (f ^ k) n / nat-power n s)
  by (intro infsetsum-cong-neutral) (auto simp: Suc-le-eq)
  also have ... = eval-fds (f ^ k) s using * unfolding eval-fds-def
  by (intro infsetsum-nat') (auto simp: fds-abs-converges-def abs-summable-on-nat-iff')
  also from that have ... = eval-fds f s ^ k by (simp add: eval-fds-power)
  finally show ?case by simp
qed
also have ... = (∑a k. eval-fds f s ^ k /R fact k)
  using exp-converges[of norm (eval-fds f s)]

```

by (intro infsetsum-nat') (auto simp: abs-summable-on-nat-iff' sums-iff field-simps norm-power)

also have ... = exp (eval-fds f s) by (simp add: exp-def)

finally show eval-fds (fds-exp f) s = exp (eval-fds f s) .

qed

define f' where f' = f - fds-const (fds-nth f 1)

have \*: fds-abs-converges (fds-exp f') s

by (auto simp: f'-def intro!: fds-abs-converges-diff conv assms)

have fds-abs-converges (fds-const (exp (fds-nth f 1)) \* fds-exp f') s

unfolding f'-def

by (intro fds-abs-converges-mult conv fds-abs-converges-diff assms) auto

thus fds-abs-converges (fds-exp f) s unfolding f'-def

by (simp add: fds-exp-times-fds-nth-0)

have eval-fds (fds-exp f) s = eval-fds (fds-const (exp (fds-nth f 1)) \* fds-exp f')

s

by (simp add: f'-def fds-exp-times-fds-nth-0)

also have ... = exp (fds-nth f (Suc 0)) \* eval-fds (fds-exp f') s using \*

using assms by (subst eval-fds-mult) (simp-all)

also have ... = exp (eval-fds f s) using ev[of f'] assms unfolding f'-def

by (auto simp: fds-abs-converges-diff eval-fds-diff fds-abs-converges-imp-converges exp-diff)

finally show eval-fds (fds-exp f) s = exp (eval-fds f s) .

qed

lemma fds-exp-add:

fixes f :: 'a :: dirichlet-series fds

shows fds-exp (f + g) = fds-exp f \* fds-exp g

proof (rule fds-eqI-truncate)

fix m :: nat assume m: m > 0

let ?T = fds-truncate m

have ?T (fds-exp (f + g)) = ?T (fds-exp (?T f + ?T g))

by (simp add: fds-truncate-exp fds-truncate-add-strong [symmetric])

also have fds-exp (?T f + ?T g) = fds-exp (?T f) \* fds-exp (?T g)

proof (rule eval-fds-eqD)

have fds-abs-converges (fds-exp (?T f + ?T g)) 0

by (intro fds-abs-converges-exp fds-abs-converges-add) auto

thus conv-abscissa (fds-exp (?T f + ?T g)) < ∞

using conv-abscissa-PInf-iff by blast

hence fds-abs-converges (fds-exp (fds-truncate m f) \* fds-exp (fds-truncate m g)) 0

by (intro fds-abs-converges-mult fds-abs-converges-exp) auto

thus conv-abscissa (fds-exp (fds-truncate m f) \* fds-exp (fds-truncate m g)) <

∞

using conv-abscissa-PInf-iff by blast

show frequently (λs. eval-fds (fds-exp (fds-truncate m f + fds-truncate m g)) s

=

eval-fds (fds-exp (fds-truncate m f) \* fds-exp (fds-truncate m

g)) s)

$((\lambda s. s \cdot 1) \text{ going-to at-top})$   
**by** (*auto simp: eval-fds-add eval-fds-mult eval-fds-exp fds-abs-converges-add*  
*fds-abs-converges-exp exp-add*)  
**qed**  
**also have**  $?T \dots = ?T (fds\text{-exp } f * fds\text{-exp } g)$   
**by** (*subst fds-truncate-mult [symmetric], subst (1 2) fds-truncate-exp*)  
*(simp add: fds-truncate-mult)*  
**finally show**  $?T (fds\text{-exp } (f + g)) = \dots$  .  
**qed**

**lemma** *fds-exp-minus:*  
**fixes**  $f :: 'a :: \text{dirichlet-series } fds$   
**shows**  $fds\text{-exp } (-f) = \text{inverse } (fds\text{-exp } f)$   
**proof** (*rule fds-right-inverse-unique*)  
**have**  $fds\text{-exp } f * fds\text{-exp } (-f) = fds\text{-exp } (f + (-f))$   
**by** (*subst fds-exp-add simp-all*)  
**also have**  $f + (-f) = 0$  **by** *simp*  
**also have**  $fds\text{-exp } \dots = 1$  **by** *simp*  
**finally show**  $fds\text{-exp } f * fds\text{-exp } (-f) = 1$  .  
**qed**

**lemma** *abs-conv-abscissa-exp:*  
**fixes**  $f :: 'a :: \text{dirichlet-series } fds$   
**shows**  $\text{abs-conv-abscissa } (fds\text{-exp } f) \leq \text{abs-conv-abscissa } f$   
**by** (*intro abs-conv-abscissa-mono fds-abs-converges-exp*)

**lemma** *fds-deriv-exp [simp]:*  
**fixes**  $f :: 'a :: \text{dirichlet-series } fds$   
**shows**  $fds\text{-deriv } (fds\text{-exp } f) = fds\text{-exp } f * fds\text{-deriv } f$   
**proof** (*rule fds-eqI-truncate*)  
**fix**  $m :: \text{nat}$  **assume**  $m: m > 0$   
**let**  $?T = \text{fds-truncate } m$   
**have**  $\text{abs-conv-abscissa } (fds\text{-deriv } (?T f)) = -\infty$   
**by** (*simp add: abs-conv-abscissa-deriv*)

**have**  $?T (fds\text{-deriv } (fds\text{-exp } f)) = ?T (fds\text{-deriv } (fds\text{-exp } (?T f)))$   
**by** (*simp add: fds-truncate-deriv fds-truncate-exp*)  
**also have**  $fds\text{-deriv } (fds\text{-exp } (?T f)) = fds\text{-exp } (?T f) * fds\text{-deriv } (?T f)$   
**proof** (*rule eval-fds-eqD*)  
**note**  $\text{abscissa} = \text{conv-le-abs-conv-abscissa abs-conv-abscissa-exp}$   
**note**  $\text{abscissa}' = \text{abscissa}[THEN \text{le-less-trans}]$   
**have**  $fds\text{-abs-converges } (fds\text{-deriv } (fds\text{-exp } (fds\text{-truncate } m f))) 0$   
**by** (*intro fds-abs-converges*)  
*(auto simp: abs-conv-abscissa-deriv intro: le-less-trans[OF abs-conv-abscissa-exp])*  
**thus**  $\text{conv-abscissa } (fds\text{-deriv } (fds\text{-exp } (fds\text{-truncate } m f))) < \infty$   
**using** *conv-abscissa-PInf-iff* **by** *blast*  
**have**  $fds\text{-abs-converges } (fds\text{-exp } (fds\text{-truncate } m f) * fds\text{-deriv } (fds\text{-truncate } m$   
 $f)) 0$   
**by** (*intro fds-abs-converges-mult fds-abs-converges-exp*)

```

      (auto intro: fds-abs-converges simp add: fds-truncate-deriv [symmetric])
    thus conv-abscissa (fds-exp (fds-truncate m f) * fds-deriv (fds-truncate m f))
  < ∞
    using conv-abscissa-PInf-iff by blast
  show ∃F s in (λs. s · 1) going-to at-top.
    eval-fds (fds-deriv (fds-exp (?T f))) s =
    eval-fds (fds-exp (?T f) * fds-deriv (?T f)) s
  proof (intro always-eventually eventually-frequently allI, goal-cases)
    case (2 s)
    have eval-fds (fds-deriv (fds-exp (?T f))) s =
      deriv (eval-fds (fds-exp (?T f))) s
    by (auto simp: eval-fds-exp eval-fds-mult fds-abs-converges-mult fds-abs-converges-exp
      fds-abs-converges eval-fds-deriv abscissa')
    also have eval-fds (fds-exp (?T f)) = (λs. exp (eval-fds (?T f) s))
      by (intro ext eval-fds-exp) auto
    also have deriv ... = (λs. exp (eval-fds (?T f) s) * deriv (eval-fds (?T f))
s)
      by (auto intro!: DERIV-imp-deriv derivative-eq-intros simp: eval-fds-deriv)
    also have ... = eval-fds (fds-exp (?T f) * fds-deriv (?T f))
    by (auto simp: eval-fds-exp eval-fds-mult fds-abs-converges-mult fds-abs-converges-exp
      fds-abs-converges eval-fds-deriv abs-conv-abscissa-deriv)
    finally show ?case .
  qed auto
  qed
  also have ?T ... = ?T (fds-exp f * fds-deriv f)
    by (subst fds-truncate-mult [symmetric])
      (simp add: fds-truncate-exp fds-truncate-deriv [symmetric], simp add: fds-truncate-mult)
  finally show ?T (fds-deriv (fds-exp f)) = ... .
  qed

```

**lemma** *fds-exp-ln-strong*:

```

  fixes f :: 'a :: dirichlet-series fds
  assumes fds-nth f (Suc 0) ≠ 0
  shows fds-exp (fds-ln l f) = fds-const (exp l / fds-nth f (Suc 0)) * f
  proof -
    let ?c = exp l / fds-nth f (Suc 0)
    have f * fds-const ?c = f * (fds-exp (-fds-ln l f) * fds-exp (fds-ln l f)) * fds-const
    ?c
      (is - = - * (?g * ?h) * -) by (subst fds-exp-add [symmetric]) simp
    also have ... = fds-const ?c * (f * ?g) * ?h by (simp add: mult-ac)
    also have f * ?g = fds-const (inverse ?c)
  proof (rule fds-deriv-eq-imp-eq)
    have fds-deriv (f * fds-exp (-fds-ln l f)) =
      fds-exp (-fds-ln l f) * fds-deriv f * (1 - f / f)
    by (simp add: divide-fds-def algebra-simps)
    also from assms have f / f = 1 by (simp add: divide-fds-def fds-right-inverse)
    finally show fds-deriv (f * fds-exp (-fds-ln l f)) = fds-deriv (fds-const (inverse
    ?c))
      by simp
  
```

**qed** (*insert assms, auto simp: exp-minus field-simps*)  
**also have**  $\text{fds-const } ?c * \text{fds-const } (\text{inverse } ?c) = 1$   
**using** *assms* **by** (*subst fds-const-mult [symmetric]*) (*simp add: divide-simps*)  
**finally show** *?thesis* **by** (*simp add: mult-ac*)  
**qed**

**lemma** *fds-exp-ln [simp]*:  
**fixes**  $f :: 'a :: \text{dirichlet-series fds}$   
**assumes**  $\text{exp } l = \text{fds-nth } f (\text{Suc } 0)$   
**shows**  $\text{fds-exp } (\text{fds-ln } l f) = f$   
**using** *assms* **by** (*subst fds-exp-ln-strong*) *auto*

**lemma** *fds-ln-exp [simp]*:  
**fixes**  $f :: 'a :: \text{dirichlet-series fds}$   
**assumes**  $l = \text{fds-nth } f (\text{Suc } 0)$   
**shows**  $\text{fds-ln } l (\text{fds-exp } f) = f$   
**proof** (*rule fds-deriv-eq-imp-eq*)  
**have**  $\text{fds-deriv } (\text{fds-ln } l (\text{fds-exp } f)) = \text{fds-deriv } f * (\text{fds-exp } f / \text{fds-exp } f)$   
**by** (*simp add: algebra-simps divide-fds-def*)  
**also have**  $\text{fds-exp } f / \text{fds-exp } f = 1$  **by** (*simp add: divide-fds-def fds-right-inverse*)  
**finally show**  $\text{fds-deriv } (\text{fds-ln } l (\text{fds-exp } f)) = \text{fds-deriv } f$  **by** *simp*  
**qed** (*insert assms, auto simp: field-simps*)

## 12.9 Euler products

**lemma** *fds-euler-product-LIMSEQ*:  
**fixes**  $f :: 'a :: \{\text{nat-power, real-normed-field, banach, second-countable-topology}\}$   
*fds*  
**assumes** *multiplicative-function (fds-nth f)* **and** *fds-abs-converges f s*  
**shows**  $(\lambda n. \prod_{p \leq n. \text{if prime } p \text{ then } \sum i. \text{fds-nth } f (p \wedge i) / \text{nat-power } (p \wedge i)}$   
 $s \text{ else } 1) \longrightarrow$   
 $\text{eval-fds } f s$   
**unfolding** *eval-fds-def*  
**proof** (*rule euler-product-LIMSEQ*)  
**show** *multiplicative-function*  $(\lambda n. \text{fds-nth } f n / \text{nat-power } n s)$   
**by** (*rule multiplicative-function-divide-nat-power*) *fact+*  
**qed** (*insert assms, auto simp: fds-abs-converges-def*)

**lemma** *fds-euler-product-LIMSEQ'*:  
**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**  $(\lambda n. \prod_{p \leq n. \text{if prime } p \text{ then } \text{inverse } (1 - \text{fds-nth } f p / \text{nat-power } p s)}$   
 $\text{else } 1) \longrightarrow$   
 $\text{eval-fds } f s$   
**unfolding** *eval-fds-def*  
**proof** (*rule euler-product-LIMSEQ'*)  
**show** *completely-multiplicative-function*  $(\lambda n. \text{fds-nth } f n / \text{nat-power } n s)$   
**by** (*rule completely-multiplicative-function-divide-nat-power*) *fact+*

**qed** (*insert assms, auto simp: fds-abs-converges-def*)

**lemma** *fds-abs-convergent-euler-product*:

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

**assumes** *multiplicative-function* (*fds-nth*  $f$ ) **and** *fds-abs-converges*  $f$   $s$

**shows** *abs-convergent-prod*

$(\lambda p. \text{if prime } p \text{ then } \sum i. \text{fds-nth } f (p \wedge i) / \text{nat-power } (p \wedge i) s \text{ else } 1)$

**unfolding** *eval-fds-def*

**proof** (*rule abs-convergent-euler-product*)

**show** *multiplicative-function*  $(\lambda n. \text{fds-nth } f n / \text{nat-power } n s)$

**by** (*rule multiplicative-function-divide-nat-power*) *fact+*

**qed** (*insert assms, auto simp: fds-abs-converges-def*)

**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*

$(\lambda p. \text{if prime } p \text{ then inverse } (1 - \text{fds-nth } f p / \text{nat-power } p s) \text{ else } 1)$

**unfolding** *eval-fds-def*

**proof** (*rule abs-convergent-euler-product'*)

**show** *completely-multiplicative-function*  $(\lambda n. \text{fds-nth } f n / \text{nat-power } n s)$

**by** (*rule completely-multiplicative-function-divide-nat-power*) *fact+*

**qed** (*insert assms, auto simp: fds-abs-converges-def*)

**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** *eval-fds*  $f$   $s = 0 \iff (\exists p. \text{prime } p \wedge \text{fds-nth } f p = \text{nat-power } p s)$

**proof** –

**let**  $?g = \lambda p. \text{if prime } p \text{ then inverse } (1 - \text{fds-nth } f p / \text{nat-power } p s) \text{ else } 1$

**have** *lim*:  $(\lambda n. \prod_{p \leq n}. ?g p) \longrightarrow \text{eval-fds } f s$

**by** (*intro fds-euler-product-LIMSEQ'* *assms*)

**have** *conv*: *convergent-prod*  $?g$

**by** (*intro abs-convergent-prod-imp-convergent-prod fds-abs-convergent-euler-product'* *assms*)

{

**assume** *eval-fds*  $f$   $s = 0$

**from** *convergent-prod-to-zero-iff*[*OF conv*] **and** *this* **and** *lim*

**have**  $\exists p. \text{prime } p \wedge \text{fds-nth } f p = \text{nat-power } p s$

**by** (*auto split: if-splits*)

} **moreover** {

**assume**  $\exists p. \text{prime } p \wedge \text{fds-nth } f p = \text{nat-power } p s$

**then obtain**  $p$  **where** *prime*  $p$  *fds-nth*  $f p = \text{nat-power } p s$  **by** *blast*

**moreover from** *this* **have** *nat-power*  $p s \neq 0$

```

    by (intro nat-power-nonzero) (auto simp: prime-gt-0-nat)
  ultimately have  $(\lambda n. \prod_{p \leq n} ?g p) \longrightarrow 0$ 
    using convergent-prod-to-zero-iff[OF conv]
    by (auto intro!: exI[of - p] split: if-splits)
  from tendsto-unique[OF - lim this] have eval-fds  $f s = 0$ 
    by simp
}
ultimately show ?thesis by blast
qed

```

lemma

```

fixes s :: 'a :: {nat-power-normed-field, banach, euclidean-space}
assumes s · 1 > 1
shows euler-product-fds-zeta:
   $(\lambda n. \prod_{p \leq n} \text{if prime } p \text{ then inverse } (1 - 1 / \text{nat-power } p s) \text{ else } 1)$ 
   $\longrightarrow \text{eval-fds fds-zeta } s$  (is ?th1)
and eval-fds-zeta-nonzero:  $\text{eval-fds fds-zeta } s \neq 0$ 
proof -
  have *: completely-multiplicative-function (fds-nth fds-zeta)
    by standard auto
  have lim:  $(\lambda n. \prod_{p \leq n} \text{if prime } p \text{ then inverse } (1 - \text{fds-nth fds-zeta } p / \text{nat-power } p s) \text{ else } 1)$ 
     $\longrightarrow \text{eval-fds fds-zeta } s$  (is filterlim ?g -)
    using assms by (intro fds-euler-product-LIMSEQ' * fds-abs-summable-zeta)
  also have ?g =  $(\lambda n. \prod_{p \leq n} \text{if prime } p \text{ then inverse } (1 - 1 / \text{nat-power } p s) \text{ else } 1)$ 
    by (intro ext prod.cong refl) (auto simp: fds-zeta-def fds-nth-fds)
  finally show ?th1 .

```

```

{
  fix p :: nat assume prime p
  from this have  $p > 1$  by (simp add: prime-gt-Suc-0-nat)
  hence norm (nat-power p s) = real p powr (s · 1)
    by (simp add: norm-nat-power)
  also have ... > real p powr 0 using assms and  $\langle p > 1 \rangle$ 
    by (intro powr-less-mono) auto
  finally have nat-power p s  $\neq 1$ 
    using  $\langle p > 1 \rangle$  by auto
}
hence **:  $\nexists p. \text{prime } p \wedge \text{fds-nth fds-zeta } p = \text{nat-power } p s$ 
  by (auto simp: fds-zeta-def fds-nth-fds)
show eval-fds fds-zeta s  $\neq 0$ 
  using assms * ** by (subst fds-abs-convergent-zero-iff) simp-all
qed

```

lemma fds-primepow-subseries-euler-product-cm:

```

fixes f :: 'a :: dirichlet-series fds
assumes completely-multiplicative-function (fds-nth f) prime p
assumes s · 1 > abs-conv-abscissa f

```

**shows**  $eval\_fds (fds\_primepow\_subseries\ p\ f)\ s = 1 / (1 - fds\_nth\ f\ p / nat\_power\ p\ s)$   
**proof** –  
**let**  $?f = (\lambda n. \prod_{pa \leq n. \text{if prime } pa \text{ then inverse } (1 - fds\_nth (fds\_primepow\_subseries\ p\ f)\ pa / nat\_power\ pa\ s) \text{ else } 1)$   
**have**  $sequentially \neq \text{bot}$  **by**  $simp$   
**moreover have**  $?f \longrightarrow eval\_fds (fds\_primepow\_subseries\ p\ f)\ s$   
**by**  $(intro\ fds\_euler\_product\_LIMSEQ' \text{ completely-multiplicative-function-only-pows } assms\ fds\_abs\_converges\_subseries) (insert\ assms, auto\ intro!: fds\_abs\_converges)$   
**moreover have**  $eventually (\lambda n. ?f\ n = 1 / (1 - fds\_nth\ f\ p / nat\_power\ p\ s))$   
*at-top*  
**using**  $eventually\_ge\_at\_top[of\ p]$   
**proof**  $eventually\_elim$   
**case**  $(elim\ n)$   
**have**  $(\prod_{pa \leq n. \text{if prime } pa \text{ then inverse } (1 - fds\_nth (fds\_primepow\_subseries\ p\ f)\ pa / nat\_power\ pa\ s) \text{ else } 1) = (\prod_{q \leq n. \text{if } q = p \text{ then inverse } (1 - fds\_nth\ f\ p / nat\_power\ p\ s) \text{ else } 1)$   
**using**  $\langle prime\ p \rangle$   
**by**  $(intro\ prod.cong) (auto\ simp: fds\_nth\_subseries\ prime\_prime\_factors)$   
**also have**  $\dots = 1 / (1 - fds\_nth\ f\ p / nat\_power\ p\ s)$   
**using**  $elim$  **by**  $(subst\ prod.delta) (auto\ simp: divide\_simps)$   
**finally show**  $?case$  .  
**qed**  
**hence**  $?f \longrightarrow 1 / (1 - fds\_nth\ f\ p / nat\_power\ p\ s)$  **by**  $(rule\ tendsto\_eventually)$   
**ultimately show**  $?thesis$  **by**  $(rule\ tendsto\_unique)$   
**qed**

## 12.10 Non-negative Dirichlet series

**lemma**  $nonneg\_Reals\_sum: (\bigwedge x. x \in A \implies f\ x \in \mathbb{R}_{\geq 0}) \implies sum\ f\ A \in \mathbb{R}_{\geq 0}$   
**by**  $(induction\ A\ rule: infinite\_finite\_induct) auto$

**locale**  $nonneg\_dirichlet\_series =$   
**fixes**  $f :: 'a :: dirichlet\_series\ fds$   
**assumes**  $nonneg\_coeffs\_aux: n > 0 \implies fds\_nth\ f\ n \in \mathbb{R}_{\geq 0}$   
**begin**

**lemma**  $nonneg\_coeffs: fds\_nth\ f\ n \in \mathbb{R}_{\geq 0}$   
**using**  $nonneg\_coeffs\_aux[of\ n]$  **by**  $(cases\ n = 0) auto$

**end**

**lemma**  $nonneg\_dirichlet\_series-0$   $[simp, intro]: nonneg\_dirichlet\_series\ 0$   
**by**  $standard (auto\ simp: zero\_fds\_def)$

**lemma**  $nonneg\_dirichlet\_series-1$   $[simp, intro]: nonneg\_dirichlet\_series\ 1$

by *standard* (*auto simp: one-fds-def*)

**lemma** *nonneg-dirichlet-series-const* [*simp,intro*]:  
 $c \in \mathbb{R}_{\geq 0} \implies \text{nonneg-dirichlet-series } (\text{fds-const } c)$   
by *standard* (*auto simp: fds-const-def*)

**lemma** *nonneg-dirichlet-series-add* [*intro*]:  
**assumes** *nonneg-dirichlet-series* *f nonneg-dirichlet-series g*  
**shows** *nonneg-dirichlet-series* (*f + g*)  
**proof** –  
**interpret** *f: nonneg-dirichlet-series f by fact*  
**interpret** *g: nonneg-dirichlet-series g by fact*  
**show** *?thesis*  
by *standard* (*auto intro!: nonneg-Reals-add-I f.nonneg-coeffs g.nonneg-coeffs*)  
**qed**

**lemma** *nonneg-dirichlet-series-mult* [*intro*]:  
**assumes** *nonneg-dirichlet-series f nonneg-dirichlet-series g*  
**shows** *nonneg-dirichlet-series* (*f \* g*)  
**proof** –  
**interpret** *f: nonneg-dirichlet-series f by fact*  
**interpret** *g: nonneg-dirichlet-series g by fact*  
**show** *?thesis*  
by *standard* (*auto intro!: nonneg-Reals-sum nonneg-Reals-mult-I f.nonneg-coeffs g.nonneg-coeffs*)  
*simp: fds-nth-mult dirichlet-prod-def*  
**qed**

**lemma** *nonneg-dirichlet-series-power* [*intro*]:  
**assumes** *nonneg-dirichlet-series f*  
**shows** *nonneg-dirichlet-series* (*f ^ n*)  
**using** *assms by (induction n) auto*

**context** *nonneg-dirichlet-series*  
**begin**

**lemma** *nonneg-exp* [*intro*]: *nonneg-dirichlet-series* (*fds-exp f*)  
**proof**  
**fix** *n :: nat* **assume** *n > 0*  
**define** *c* **where** *c = exp (fds-nth f (Suc 0))*  
**define** *f'* **where** *f' = fds (λn. if n = Suc 0 then 0 else fds-nth f n)*  
**from** *nonneg-coeffs[of 1]* **obtain** *c'* **where** *fds-nth f (Suc 0) = of-real c'*  
by (*auto elim!: nonneg-Reals-cases*)  
**hence** *c = of-real (exp c')* **by** (*simp add: c-def exp-of-real*)  
**hence** *c: c ∈ ℝ<sub>≥0</sub>* **by** *simp*  
**have** *less: n < 2 ^ k if n < k for k*  
**proof** –  
**have** *n < k* **by** *fact*  
**also have** *... < 2 ^ k*

```

    by (rule less-exp)
  finally show ?thesis .
qed
have nonneg-power: fds-nth (f' ^ k) n ∈ ℝ≥0 for k
proof -
  have nonneg-dirichlet-series f'
  by standard (insert nonneg-coeffs, auto simp: f'-def)
  interpret nonneg-dirichlet-series f' ^ k
  by (intro nonneg-dirichlet-series-power) fact+
  from nonneg-coeffs[of n] show ?thesis .
qed
hence fds-nth (fds-exp f) n = c * (∑ k. fds-nth (f' ^ k) n /R fact k)
  by (simp add: fds-exp-def fds-nth-fds' f'-def c-def)
also have (∑ k. fds-nth (f' ^ k) n /R fact k) = (∑ k ≤ n. fds-nth (f' ^ k) n /R
fact k)
  by (intro suminf-finite) (auto intro!: fds-nth-power-eq-0 less simp: f'-def not-le)
also have c * ... ∈ ℝ≥0 unfolding scaleR-conv-of-real
  by (intro nonneg-Reals-mult-I nonneg-Reals-sum nonneg-power, unfold non-
neg-Reals-of-real-iff )
  (auto simp: c)
  finally show fds-nth (fds-exp f) n ∈ ℝ≥0 .
qed

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 -
  from assms have nonneg-dirichlet-series (fds-exp (fds-ln l f))
  by (intro nonneg-dirichlet-series.nonneg-exp)
  thus ?thesis using assms by simp
qed

context nonneg-dirichlet-series
begin

lemma fds-of-real-norm: fds-of-real (fds-norm f) = f
proof (rule fds-eqI)
  fix n :: nat assume n: n > 0
  show fds-nth (fds-of-real (fds-norm f)) n = fds-nth f n
  using nonneg-coeffs[of n] by (auto elim!: nonneg-Reals-cases)
qed

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  $\exists x. x < c \wedge \text{fds-abs-converges } f \ (\text{of-real } x)$ 
proof –
interpret nonneg-dirichlet-series f by fact
define a where  $a = 1 + c$ 

define g' where  $g' = (\lambda s. \text{if } s \in \{s. \text{Re } s > c\} \text{ then eval-fds } f \ s \ \text{else } g \ s)$ 

```

— We can find some  $\varepsilon > 0$  such that the Dirichlet series can be continued analytically in a ball of radius  $1 + \varepsilon$  around  $a$ .

```

from  $\langle \text{open } A \rangle \langle c \in A \rangle$  obtain  $\delta$  where  $\delta > 0$  ball c \delta \subseteq A
by (auto simp: open-contains-ball)
define  $\varepsilon$  where  $\varepsilon = \text{sqrt } (1 + \delta^2) - 1$ 
from  $\delta$  have  $\varepsilon > 0$  by (simp add: \varepsilon-def)

```

```

have ball-a-subset:  $\text{ball } a \ (1 + \varepsilon) \subseteq \{s. \text{Re } s > c\} \cup A$ 

```

```

proof (intro subsetI)

```

```

fix  $s :: \text{complex}$  assume  $s \in \text{ball } a \ (1 + \varepsilon)$ 

```

```

define  $x \ y$  where  $x = \text{Re } s$  and  $y = \text{Im } s$ 

```

```

have [simp]:  $s = x + i * y$  by (simp add: complex-eq-iff x-def y-def)

```

```

show  $s \in \{s. \text{Re } s > c\} \cup A$ 

```

```

proof (cases Re s \le c)

```

```

case True

```

```

hence  $(c - x)^2 + y^2 \leq (1 + c - x)^2 + y^2 - 1$ 

```

```

by (simp add: power2-eq-square algebra-simps)

```

```

also from  $s$  have  $(1 + c - x)^2 + y^2 - 1 < \delta^2$ 

```

```

by (auto simp: dist-norm cmod-def a-def \varepsilon-def)

```

```

finally have  $\text{sqrt } ((c - x)^2 + y^2) < \delta$  using  $\delta$ 

```

```

by (intro real-less-lsqr) auto

```

```

hence  $s \in \text{ball } c \ \delta$  by (auto simp: dist-norm cmod-def)

```

```

also have  $\dots \subseteq A$  by fact

```

```

finally show ?thesis ..

```

```

qed auto

```

```

qed

```

```

have holo:  $g'$  holomorphic-on ball a (1 + \varepsilon) unfolding g'-def

```

```

proof (intro holomorphic-on-subset[OF - ball-a-subset] holomorphic-on-If-Un)

```

```

have conv-abscissa f \le abs-conv-abscissa f by (rule conv-le-abs-conv-abscissa)

```

```

also have  $\dots \leq \text{ereal } c$  by fact

```

```

finally have*:  $\text{conv-abscissa } f \leq \text{ereal } c$  .

```

```

show  $\text{eval-fds } f$  holomorphic-on \{s. c < Re s\}

```

```

by (intro holomorphic-intros) (auto intro: le-less-trans[OF *])

```

```

qed (insert assms, auto intro!: holomorphic-intros open-halfspace-Re-gt)

```

```

define  $f'$  where  $f' = \text{fds-norm } f$ 

```

```

have  $f \cdot f'$ :  $f = \text{fds-of-real } f'$  by (simp add: f'-def fds-of-real-norm)

```

**have**  $f'$ -nonneg:  $\text{fds-nth } f' n \geq 0$  **for**  $n$   
**using** *nonneg-coeffs[of n]* **by** (*auto elim!:* *nonneg-Reals-cases simp: f'-def*)

**have** *deriv*:  $(\lambda n. (\text{deriv } \sim n) g' a) = (\lambda n. \text{eval-fds } ((\text{fds-deriv } \sim n) f) a)$   
**proof**  
**fix**  $n :: \text{nat}$   
**have** *ev*: *eventually*  $(\lambda s. s \in \{s. \text{Re } s > c\})$  (*nhds (complex-of-real a)*)  
**by** (*intro eventually-nhds-in-open open-halfspace-Re-gt*) (*auto simp: a-def*)  
**have**  $(\text{deriv } \sim n) g' a = (\text{deriv } \sim n) (\text{eval-fds } f) a$   
**by** (*intro higher-deriv-cong-ev refl eventually-mono[OF ev]*) (*auto simp: g'-def*)  
**also have**  $\dots = \text{eval-fds } ((\text{fds-deriv } \sim n) f) a$   
**proof** (*intro eval-fds-higher-deriv [symmetric]*)  
**have**  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$  **by** (*rule conv-le-abs-conv-abscissa*)  
**also have**  $\dots \leq \text{ereal } c$  **by** (*rule assms*)  
**also have**  $\dots < a$  **by** (*simp add: a-def*)  
**finally show**  $\text{conv-abscissa } f < \text{ereal } (\text{complex-of-real } a \cdot 1)$  **by** *simp*  
**qed**  
**finally show**  $(\text{deriv } \sim n) g' a = \text{eval-fds } ((\text{fds-deriv } \sim n) f) a$  .  
**qed**

**have** *nth-deriv-conv*:  $\text{fds-abs-converges } ((\text{fds-deriv } \sim n) f)$  (*of-real a*) **for**  $n$   
**by** (*intro fds-abs-converges*)  
*(auto simp: abs-conv-abscissa-higher-deriv a-def intro!: le-less-trans[OF abscissa])*

**have** *nth-deriv-eq*:  $(\text{fds-deriv } \sim n) f = \text{fds } (\lambda k. (-1) \wedge n * \text{fds-nth } f k * \ln (\text{real } k) \wedge n)$  **for**  $n$   
**proof** –  
**have**  $\text{fds-nth } ((\text{fds-deriv } \sim n) f) k = (-1) \wedge n * \text{fds-nth } f k * \ln (\text{real } k) \wedge n$   
**for**  $k$   
**by** (*induction n*) (*simp-all add: fds-deriv-def fds-eq-iff fds-nth-fds' scaleR-conv-of-real*)  
**thus** *?thesis* **by** (*intro fds-eqI*) *simp-all*  
**qed**

**have** *deriv'*:  $(\lambda n. \text{eval-fds } ((\text{fds-deriv } \sim n) f) (\text{complex-of-real } a)) =$   
 $(\lambda n. (-1) \wedge n * \text{complex-of-real } (\sum_{a k. \text{fds-nth } f' k * \ln (\text{real } k) \wedge n} / \text{real } k$   
*powr a*))  
**proof**  
**fix**  $n$   
**have**  $\text{eval-fds } ((\text{fds-deriv } \sim n) f) (\text{of-real } a) =$   
 $(\sum_{a k. \text{fds-nth } ((\text{fds-deriv } \sim n) f) k} / \text{of-nat } k \text{ powr } \text{complex-of-real } a)$   
**using** *nth-deriv-conv* **by** (*subst eval-fds-altdef*) *auto*  
**hence**  $\text{eval-fds } ((\text{fds-deriv } \sim n) f) (\text{of-real } a) =$   
 $(\sum_{a k. (-1) \wedge n *_{\mathbb{R}} (\text{fds-nth } f k * \ln (\text{real } k) \wedge n} / k \text{ powr } a))$   
**by** (*simp add: nth-deriv-eq fds-nth-fds' powr-Reals-eq scaleR-conv-of-real algebra-simps*)  
**also have**  $\dots = (-1) \wedge n * (\sum_{a k. \text{of-real } (\text{fds-nth } f' k * \ln (\text{real } k) \wedge n} / k$   
*powr a*))

by (subst infsetsum-scaleR-right) (simp-all add: scaleR-conv-of-real f-f')  
 also have ... =  $(-1)^{\wedge n} * \text{of-real} (\sum_{a k. \text{fds-nth } f' k * \ln (\text{real } k)^{\wedge n} / k}$   
 powr a)  
 by (subst infsetsum-of-real) (rule refl)  
 finally show eval-fds ((fds-deriv  $\widehat{\wedge} n$ ) f) (complex-of-real a) =  
 $(-1)^{\wedge n} * \text{complex-of-real} (\sum_{a k. \text{fds-nth } f' k * \ln (\text{real } k)^{\wedge n} / \text{real } k}$   
 powr a) .  
 qed

define s :: complex where s = c - ε / 2  
 have s: Re s < c using assms δ by (simp-all add: s-def ε-def field-simps)  
 have s ∈ ball a (1 + ε) using s by (simp add: a-def dist-norm cmod-def s-def)  
 from holomorphic-power-series[OF holo this]  
 have sums: (λn. (deriv  $\widehat{\wedge} n$ ) g' a / fact n \* (s - a)^{\wedge n}) sums g' s by simp  
 also note deriv  
 also have s - a = -of-real (1 + ε / 2) by (simp add: s-def a-def)  
 also have (λn. ...^{\wedge} n) = (λn. of-real ((-1)^{\wedge} n \* (1 + ε / 2)^{\wedge} n))  
 by (intro ext) (subst power-minus, auto)  
 also have (λn. eval-fds ((fds-deriv  $\widehat{\wedge} n$ ) f) a / fact n \* ...^{\wedge} n) =  
 (λn. of-real ((-1)^{\wedge} n \* eval-fds ((fds-deriv  $\widehat{\wedge} n$ ) f') a / fact n \*  
 (1+ε/2)^{\wedge} n))  
 using nth-deriv-conv by (simp add: f-f' fds-abs-converges-imp-converges mult-ac)  
 finally have summable ... by (simp add: sums-iff)  
 hence summable: summable (λn. (-1)^{\wedge} n \* eval-fds ((fds-deriv  $\widehat{\wedge} n$ ) f') a / fact  
 n \* (1+ε/2)^{\wedge} n)  
 by (subst (asm) summable-of-real-iff)

have (λ(n,k). (-1)^{\wedge} n \* fds-nth f k \* ln (real k)^{\wedge} n / (real k powr a) \* ((s-a)  
 ^{\wedge} n / fact n))  
 abs-summable-on (UNIV × UNIV)  
 proof (subst abs-summable-on-Sigma-iff, safe, goal-cases)  
 case (3 n)  
 from nth-deriv-conv[of n] show ?case  
 unfolding fds-abs-converges-altdef'  
 by (intro abs-summable-on-cmult-left) (simp add: nth-deriv-eq fds-nth-fds'  
 powr-Reals-eq)  
 next  
 case 4  
 have nth-deriv-f-f': (fds-deriv  $\widehat{\wedge} n$ ) f = fds-of-real ((fds-deriv  $\widehat{\wedge} n$ ) f') for n  
 by (induction n) (auto simp: f'-def fds-of-real-norm)  
 have norm-nth-deriv-f: norm (fds-nth ((fds-deriv  $\widehat{\wedge} n$ ) f) k) =  
 (-1)^{\wedge} n \* of-real (fds-nth ((fds-deriv  $\widehat{\wedge} n$ ) f') k) for  
 n k  
 proof (induction n)  
 case (Suc n)  
 thus ?case by (cases k) (auto simp: f-f' fds-nth-deriv scaleR-conv-of-real  
 norm-mult)  
 qed (auto simp: f'-nonneg f-f')

**note summable**  
**also have**  $(\lambda n. (-1)^{\wedge n} * \text{eval-fds } ((\text{fds-deriv } \sim n) f') a / \text{fact } n * (1 + \varepsilon / 2)^{\wedge n})$   
 $=$   
 $(\lambda n. \sum_a k. \text{norm } ((-1)^{\wedge n} * \text{fds-nth } f k * \ln (\text{real } k)^{\wedge n} /$   
 $(\text{real } k \text{ powr } a) * ((s - a)^{\wedge n} / \text{fact } n)))$  (**is - = ?h**)  
**proof** (*rule ext, goal-cases*)  
**case** ( $1\ n$ )  
**have**  $(\sum_a k. \text{norm } ((-1)^{\wedge n} * \text{fds-nth } f k * \ln (\text{real } k)^{\wedge n} /$   
 $(\text{real } k \text{ powr } a) * ((s - a)^{\wedge n} / \text{fact } n))) =$   
 $(\text{norm } ((s - a)^{\wedge n} / \text{fact } n) * (-1)^{\wedge n}) *_{\mathbb{R}}$   
 $(\sum_a k. (-1)^{\wedge n} * \text{norm } (\text{fds-nth } ((\text{fds-deriv } \sim n) f) k / \text{real } k \text{ powr } a))$  (**is - = - \*<sub>R</sub> ?S**)  
**by** (*subst infsetsum-scaleR-right [symmetric]*)  
*(auto simp: norm-mult norm-divide norm-power mult-ac nth-deriv-eq*  
*fds-nth-fds')*  
**also have**  $?S = (\sum_a k. \text{fds-nth } ((\text{fds-deriv } \sim n) f') k / \text{real } k \text{ powr } a)$   
**by** (*intro infsetsum-cong*) (*auto simp: norm-mult norm-divide norm-power*  
*norm-nth-deriv-f*)  
**also have**  $\dots = \text{eval-fds } ((\text{fds-deriv } \sim n) f') a$   
**using** *nth-deriv-conv[of n]* **by** (*subst eval-fds-altdef*) (*auto simp: f'-def*  
*nth-deriv-f-f')*  
**also have**  $(\text{norm } ((s - a)^{\wedge n} / \text{fact } n) * (-1)^{\wedge n}) *_{\mathbb{R}} \text{eval-fds } ((\text{fds-deriv } \sim n) f') a =$   
 $(-1)^{\wedge n} * \text{eval-fds } ((\text{fds-deriv } \sim n) f') a / \text{fact } n * \text{norm } (s - a)^{\wedge n}$   
**by** (*simp add: norm-divide norm-power*)  
**also have**  $s - a = -\text{of-real } (1 + \varepsilon / 2)$  **by** (*simp add: s-def a-def*)  
**have**  $\text{norm } (s - a) = 1 + \varepsilon / 2$  **unfolding**  $s - a$  *norm-minus-cancel norm-of-real*  
**using**  $\varepsilon$  **by** *simp*  
**finally show**  $?case ..$   
**qed**  
**also have**  $?h\ n \geq 0$  **for**  $n$  **by** (*intro infsetsum-nonneg*) *auto*  
**hence**  $?h = (\lambda n. \text{norm } (?h\ n))$  **by** *simp*  
**finally show**  $?case$  **unfolding** *abs-summable-on-nat-iff'* .  
**qed** *auto*  
**hence**  $(\lambda(k,n). (-1)^{\wedge n} * \text{fds-nth } f k * \ln (\text{real } k)^{\wedge n} / (\text{real } k \text{ powr } a) * ((s - a)^{\wedge n} / \text{fact } n))$   
*abs-summable-on (UNIV × UNIV)*  
**by** (*subst (asm) abs-summable-on-Times-swap*) (*simp add: case-prod-unfold*)  
**hence**  $(\lambda k. \sum_a n. (-1)^{\wedge n} * \text{fds-nth } f k * \ln (\text{real } k)^{\wedge n} / (k \text{ powr } a) *$   
 $((s - a)^{\wedge n} / \text{fact } n))$  *abs-summable-on UNIV* (**is**  $?h$  *abs-summable-on -*)  
**by** (*rule abs-summable-on-Sigma-project1'*) *auto*  
**also have**  $?this \iff (\lambda k. \text{fds-nth } f k / \text{nat-power } k\ s)$  *abs-summable-on UNIV*  
**proof** (*intro abs-summable-on-cong refl, goal-cases*)  
**case** ( $1\ k$ )  
**have**  $?h\ k = (\text{fds-nth } f' k / k \text{ powr } a) *_{\mathbb{R}} (\sum_a n. (-\ln (\text{real } k) * (s - a))^{\wedge n} / \text{fact } n)$   
**by** (*subst infsetsum-scaleR-right [symmetric], rule infsetsum-cong*)  
*(simp-all add: scaleR-conv-of-real f-f' power-minus' power-mult-distrib*

*divide-simps*)  
**also have**  $(\sum_a n. (-\ln (\text{real } k) * (s - a)) ^ n / \text{fact } n) = \text{exp } (-\ln (\text{real } k) * (s - a))$   
**using** *exp-converges[of  $-\ln k * (s - a)$ ] exp-converges[of norm  $(-\ln k * (s - a))$ ]*  
**by** (*subst infsetsum-nat'*) (*auto simp: abs-summable-on-nat-iff' sums-iff scaleR-conv-of-real*)  
*divide-simps norm-divide norm-mult norm-power*)  
**also have**  $(\text{fds-nth } f' k / k \text{ powr } a) *_{\mathbb{R}} \dots = \text{fds-nth } f k / \text{nat-power } k s$   
**by** (*auto simp: scaleR-conv-of-real f-f' powr-def exp-minus field-simps exp-of-real [symmetric] exp-diff*)  
**finally show** ?case .  
**qed**  
**finally have** *fds-abs-converges f s*  
**by** (*simp add: fds-abs-converges-def abs-summable-on-nat-iff'*)  
**thus** ?thesis **by** (*intro exI[of  $-(c - \varepsilon / 2)$ ] (auto simp: s-def a-def  $\varepsilon$ )*)  
**qed**

**theorem** *pringsheim-landau*:

**fixes**  $c :: \text{real}$  **and**  $f :: \text{complex fds}$   
**assumes** *nonneg-dirichlet-series f*  
**assumes** *abscissa: 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 ∈ A*  
**shows** *False*

**proof** –

**have**  $\exists x < c. \text{fds-abs-converges } f (\text{complex-of-real } x)$   
**by** (*rule pringsheim-landau-aux[where  $g = g$  and  $A = A$ ] (insert assms, auto)*)  
**then obtain**  $x$  **where**  $x: x < c \text{ fds-abs-converges } f (\text{complex-of-real } x)$  **by** *blast*  
**hence** *abs-conv-abscissa f ≤ complex-of-real x · 1*  
**unfolding** *abs-conv-abscissa-def*  
**by** (*intro Inf-lower*) (*auto simp: image-iff intro!: exI[of - of-real x]*)  
**also have**  $\dots < \text{abs-conv-abscissa } f$  **using** *assms x* **by** *simp*  
**finally show** *False* **by** *simp*

**qed**

**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** *abs-conv-abscissa f =  $-\infty$*

**proof** (*rule ccontr*)

**assume** *abs-conv-abscissa f ≠  $-\infty$*   
**with**  $c$  **obtain**  $a$  **where** *abscissa [simp]: abs-conv-abscissa f = ereal a*  
**by** (*cases abs-conv-abscissa f*) *auto*  
**show** *False*  
**proof** (*rule pringsheim-landau[OF assms(1) abscissa - holo]*)  
**fix**  $s$  **assume**  $s: \text{Re } s > a$

```

show  $g\ s = \text{eval-fds}\ f\ s$ 
proof (rule sym, rule analytic-continuation-open[of - - - g])
  show  $g$  holomorphic-on  $\{s. \text{Re } s > a\}$  by (rule holomorphic-on-subset[OF
holo]) auto
  from assms show  $\{s. \text{Re } s > c\} \subseteq \{s. \text{Re } s > a\}$  by auto
next
  have  $\text{conv-abscissa } f \leq \text{abs-conv-abscissa } f$  by (rule conv-le-abs-conv-abscissa)
  also have  $\dots = \text{ereal } a$  by simp
  finally show  $\text{eval-fds } f$  holomorphic-on  $\{s. \text{Re } s > a\}$ 
    by (intro holomorphic-intros) (auto intro: le-less-trans)
qed (insert assms s, auto intro!: exI[of - of-real (c + 1)]
open-halfspace-Re-gt convex-connected convex-halfspace-Re-gt)
qed auto
qed

```

## 12.11 Convergence of the $\zeta$ and Möbius $\mu$ series

```

lemma fds-abs-summable-zeta-real-iff [simp]:
   $\text{fds-abs-converges } \text{fds-zeta } s \longleftrightarrow s > (1 :: \text{real})$ 
proof -
  have  $\text{fds-abs-converges } \text{fds-zeta } s \longleftrightarrow \text{summable } (\lambda n. \text{real } n \text{ powr } -s)$ 
    unfolding fds-abs-converges-def
    by (intro summable-cong always-eventually)
    (auto simp: fds-nth-zeta powr-minus divide-simps)
  also have  $\dots \longleftrightarrow s > 1$  by (simp add: summable-real-powr-iff)
  finally show ?thesis .
qed

```

```

lemma fds-abs-summable-zeta-real:  $s > (1 :: \text{real}) \implies \text{fds-abs-converges } \text{fds-zeta } s$ 
by simp

```

```

lemma fds-abs-converges-moebius-mu-real:
  assumes  $s > (1 :: \text{real})$ 
  shows  $\text{fds-abs-converges } (\text{fds } \text{moebius-mu})\ s$ 
  unfolding fds-abs-converges-def
proof (rule summable-comparison-test, intro exI allI impI)
  fix  $n :: \text{nat}$ 
  show  $\text{norm } (\text{norm } (\text{fds-nth } (\text{fds } \text{moebius-mu})\ n / \text{nat-power } n\ s)) \leq n \text{ powr } (-s)$ 
    by (simp add: powr-minus divide-simps abs-moebius-mu-le)
next
  from assms show  $\text{summable } (\lambda n. \text{real } n \text{ powr } -s)$  by (simp add: summable-real-powr-iff)
qed

```

## 12.12 Application to the Möbius $\mu$ function

```

lemma inverse-squares-sums':  $(\lambda n. 1 / \text{real } n^2) \text{ sums } (\pi^2 / 6)$ 
  using inverse-squares-sums sums-Suc-iff[of  $\lambda n. 1 / \text{real } n^2$   $\pi^2 / 6$ ] by simp

```

```

lemma norm-summable-moebius-over-square:

```

```

    summable (λn. norm (moebius-mu n / real n ^ 2))
proof (subst summable-Suc-iff [symmetric], rule summable-comparison-test)
  show summable (λn. 1 / real (Suc n) ^ 2)
    using inverse-squares-sums by (simp add: sums-iff)
qed (auto simp del: of-nat-Suc simp: field-simps abs-moebius-mu-le)

lemma summable-moebius-over-square:
  summable (λn. moebius-mu n / real n ^ 2)
  using norm-summable-moebius-over-square by (rule summable-norm-cancel)

lemma moebius-over-square-sums: (λn. moebius-mu n / n2) sums (6 / pi2)
proof –
  have 1 = eval-fds (1 :: real fds) 2 by simp
  also have (1 :: real fds) = fds-zeta * fds moebius-mu
    by (rule fds-zeta-times-moebius-mu [symmetric])
  also have eval-fds ... 2 = eval-fds fds-zeta 2 * eval-fds (fds moebius-mu) 2
    by (intro eval-fds-mult fds-abs-converges-moebius-mu-real) simp-all
  also have ... = pi ^ 2 / 6 * (∑ n. moebius-mu n / (real n)2)
    using inverse-squares-sums' by (simp add: eval-fds-at-numeral suminf-fds-zeta-aux
sums-iff)
  finally have (∑ n. moebius-mu n / (real n)2) = 6 / pi ^ 2 by (simp add:
field-simps)
  with summable-moebius-over-square show ?thesis by (simp add: sums-iff)
qed

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 (λn. 1 / (real (Suc n) + d) ^ 2)

(∑ n. 1 / (real (Suc n) + d) ^ 2) ≤ 1 / d

**proof** –

**show** \*: summable (λn. 1 / (real (Suc n) + d) ^ 2)

**proof** (rule summable-comparison-test, intro allI exI impI)

**fix** n :: nat

**from** assms **show** norm (1 / (real (Suc n) + d) ^ 2) ≤ 1 / real (Suc n) ^ 2

**unfolding** norm-divide norm-one norm-power

```

    by (intro divide-left-mono power-mono) simp-all
  qed (insert inverse-squares-sums, simp add: sums-iff)
  show  $(\sum n. 1 / (\text{real } (\text{Suc } n) + d) ^ 2) \leq 1 / d$ 
  proof (rule sums-le)
    fix n have  $1 / (\text{real } (\text{Suc } n) + d) ^ 2 \leq 1 / ((\text{real } n + d) * (\text{real } (\text{Suc } n) + d))$ 
  next
    unfolding power2-eq-square using assms
    by (intro divide-left-mono mult-mono mult-pos-pos add-nonneg-pos) simp-all
    also have  $\dots = 1 / (\text{real } n + d) - 1 / (\text{real } (\text{Suc } n) + d)$ 
    using assms by (simp add: divide-simps)
    finally show  $1 / (\text{real } (\text{Suc } n) + d) ^ 2 \leq 1 / (\text{real } n + d) - 1 / (\text{real } (\text{Suc } n) + d)$  .
  next
    show  $(\lambda n. 1 / (\text{real } (\text{Suc } n) + d) ^ 2)$  sums  $(\sum n. 1 / (\text{real } (\text{Suc } n) + d) ^ 2)$ 
    using * by (simp add: sums-iff)
  next
    have  $(\lambda n. 1 / (\text{real } n + d) - 1 / (\text{real } (\text{Suc } n) + d))$  sums  $(1 / (\text{real } 0 + d) - 0)$ 
    by (intro telescope-sums' real-tendsto-divide-at-top[OF tendsto-const],
        subst add.commute, rule filterlim-tendsto-add-at-top[OF tendsto-const]
        filterlim-real-sequentially)
    thus  $(\lambda n. 1 / (\text{real } n + d) - 1 / (\text{real } (\text{Suc } n) + d))$  sums  $(1 / d)$  by simp
  qed
qed

```

**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
  abs ?S ≤ -)
  proof -
    have *: summable  $(\lambda n. 1 / (\text{real } (\text{Suc } n + d) ^ 2))$ 
    by (insert sum-inverse-squares-tail-bound(1)[of real d] assms, simp-all add:
    add-ac)
    have **: summable  $(\lambda n. \text{abs } (\text{moebius-mu } (\text{Suc } n + d) / \text{real } (\text{Suc } n + d) ^ 2))$ 
    proof (rule summable-comparison-test, intro exI allI impI)
      fix n :: nat
      show norm  $(|\text{moebius-mu } (\text{Suc } n + d) / \text{real } (\text{Suc } n + d) ^ 2|) \leq$ 
         $1 / (\text{real } (\text{Suc } n + d) ^ 2)$ 
      unfolding real-norm-def abs-abs abs-divide power-abs abs-of-nat
      by (intro divide-right-mono abs-moebius-mu-le) simp-all
    qed (insert *)
    from ** have  $\text{abs } ?S \leq (\sum n. \text{abs } (\text{moebius-mu } (\text{Suc } n + d) / \text{real } (\text{Suc } n + d) ^ 2))$ 
  by (rule summable-rabs)
  also have  $\dots \leq (\sum n. 1 / (\text{real } (\text{Suc } n) + d) ^ 2)$ 
  proof (intro suminf-le allI)
    fix n :: nat
    show  $\text{abs } (\text{moebius-mu } (\text{Suc } n + d) / \text{real } (\text{Suc } n + d) ^ 2) \leq 1 / (\text{real } (\text{Suc } n) + \text{real } d) ^ 2$ 

```

**unfolding** *abs-divide abs-of-nat power-abs of-nat-add [symmetric]*  
**by** (*intro divide-right-mono abs-moebius-mu-le simp-all*)  
**qed** (*insert \* \*\*, simp-all add: add-ac*)  
**also from** *assms have ... ≤ 1 / d* **by** (*intro sum-inverse-squares-tail-bound*)  
*simp-all*  
**finally show** *?thesis .*  
**qed**

**lemma** *sum-upto-inverse-bound:*  
*sum-upto (λi. 1 / real i) x ≥ 0*  
*eventually (λx. sum-upto (λi. 1 / real i) x ≤ ln x + 13 / 22) at-top*  
**proof** –  
**show** *sum-upto (λi. 1 / real i) x ≥ 0*  
**by** (*simp add: sum-upto-def sum-nonneg*)  
**from** *order-tendstoD(2)[OF euler-mascheroni-LIMSEQ euler-mascheroni-less-13-over-22]*  
**obtain** *N where N: ∧n. n ≥ N ⇒ harm n – ln (real n) < 13 / 22*  
**unfolding** *eventually-at-top-linorder* **by** *blast*  
**show** *eventually (λx. sum-upto (λi. 1 / real i) x ≤ ln x + 13 / 22) at-top*  
**using** *eventually-ge-at-top[of max (real N) 1]*  
**proof** *eventually-elim*  
**case** (*elim x*)  
**have** *sum-upto (λi. 1 / real i) x = (∑ i∈{0<..nat ⌊x⌋}. 1 / real i)*  
**by** (*simp add: sum-upto-altdef*)  
**also have** *... = harm (nat ⌊x⌋)*  
**unfolding** *harm-def* **by** (*intro sum.cong refl*) (*auto simp: field-simps*)  
**also have** *... ≤ ln (real (nat ⌊x⌋)) + 13 / 22*  
**using** *N[of nat ⌊x⌋] elim* **by** (*auto simp: le-nat-iff le-floor-iff*)  
**also have** *ln (real (nat ⌊x⌋)) ≤ ln x* **using** *elim* **by** (*subst ln-le-cancel-iff*) *auto*  
**finally show** *?case* **by** – *simp*  
**qed**  
**qed**

**lemma** *sum-upto-inverse-bigo: sum-upto (λi. 1 / real i) ∈ O(λx. ln x)*  
**proof** –  
**have** *eventually (λx. norm (sum-upto (λi. 1 / real i) x) ≤ 1 \* norm (ln x + 13/22)) at-top*  
**using** *eventually-ge-at-top[of 1::real] sum-upto-inverse-bound(2)*  
**by** *eventually-elim (insert sum-upto-inverse-bound(1), simp-all)*  
**hence** *sum-upto (λi. 1 / real i) ∈ O(λx. ln x + 13/22)*  
**by** (*rule bigoI*)  
**also have** *(λx::real. ln x + 13/22) ∈ O(λx. ln x)* **by** *simp*  
**finally show** *?thesis .*  
**qed**

**lemma**  
**defines** *G ≡ (λx::real. (∑ n. moebius-mu (n + Suc (nat ⌊x⌋)) / (n + Suc (nat ⌊x⌋))) ^ 2) :: real*  
**shows** *moebius-sum-tail-bound': ∧t. t ≥ 2 ⇒ |G t| ≤ 1 / (t – 1)*  
**and** *moebius-sum-tail-bigo: G ∈ O(λt. 1 / t)*

**proof** –  
**show**  $|G t| \leq 1 / (t - 1)$  **if**  $t: t \geq 2$  **for**  $t$   
**proof** –  
**from**  $t$  **have**  $|G t| \leq 1 / \text{real } (\text{nat } \lfloor t \rfloor)$   
**unfolding**  $G\text{-def}$  **using**  $\text{moebius-sum-tail-bound}[\text{of nat } \lfloor t \rfloor]$  **by**  $\text{simp}$   
**also have**  $t \leq 1 + \text{real-of-int } \lfloor t \rfloor$  **by**  $\text{linarith}$   
**hence**  $1 / \text{real } (\text{nat } \lfloor t \rfloor) \leq 1 / (t - 1)$  **using**  $t$  **by**  $(\text{simp add: field-simps})$   
**finally show**  $?thesis$  .  
**qed**  
**hence**  $G \in O(\lambda t. 1 / (t - 1))$   
**by**  $(\text{intro bigO}I[\text{of } - 1] \text{eventually-mono}[OF \text{eventually-ge-at-top}[\text{of } 2::\text{real}]]) \text{auto}$   
**also have**  $(\lambda t::\text{real}. 1 / (t - 1)) \in \Theta(\lambda t. 1 / t)$  **by**  $\text{simp}$   
**finally show**  $G \in O(\lambda t. 1 / t)$  .  
**qed**

## 13.2 Summatory totient function

**theorem**  $\text{summatory-totient-asymptotics}$ :

$(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) x - 3 / \text{pi}^2 * x^2) \in O(\lambda x. x * \ln x)$

**proof** –

**define**  $H$  **where**  $H = (\lambda x. \text{of-int } (\text{floor } x) * (\text{of-int } (\text{floor } x) + 1) / 2 - x^2 / 2 :: \text{real})$

**define**  $H'$  **where**  $H' = (\lambda x. \text{sum-upto } (\lambda n. \text{moebius-mu } n * H (x / \text{real } n)) x)$

**have**  $H: \text{sum-upto real } x = x^2 / 2 + H x$  **if**  $x \geq 0$  **for**  $x$

**using that by**  $(\text{simp add: sum-upto-real } H\text{-def})$

**define**  $G$  **where**  $G = (\lambda x::\text{real}. (\sum n. \text{moebius-mu } (n + \text{Suc } (\text{nat } \lfloor x \rfloor)) / (n + \text{Suc } (\text{nat } \lfloor x \rfloor))^2))$

**have**  $H\text{-bound}: |H t| \leq t / 2$  **if**  $t \geq 0$  **for**  $t$

**proof** –

**have**  $H t - t / 2 = (-(t - \text{of-int } (\text{floor } t))) * (\text{floor } t + t + 1) / 2$

**by**  $(\text{simp add: } H\text{-def field-simps power2-eq-square})$

**also have**  $\dots \leq 0$  **using that by**  $(\text{intro mult-nonpos-nonneg divide-nonpos-nonneg})$   
 $\text{simp-all}$

**finally have**  $H t \leq t / 2$  **by**  $\text{simp}$

**have**  $-H t - t / 2 = (t - \text{of-int } (\text{floor } t) - 1) * (\text{of-int } (\text{floor } t) + t) / 2$

**by**  $(\text{simp add: } H\text{-def field-simps power2-eq-square})$

**also have**  $\dots \leq 0$  **using that**

**by**  $(\text{intro divide-nonpos-nonneg mult-nonpos-nonneg}) ((\text{simp}; \text{fail}) | \text{linarith})+$

**finally have**  $-H t \leq t / 2$  **by**  $\text{simp}$

**with**  $\langle H t \leq t / 2 \rangle$  **show**  $|H t| \leq t / 2$  **by**  $\text{simp}$

**qed**

**have**  $H'\text{-bound}: |H' t| \leq t / 2 * \text{sum-upto } (\lambda i. 1 / \text{real } i) t$  **if**  $t \geq 0$  **for**  $t$

**proof** –

**have**  $|H' t| \leq (\sum i | 0 < i \wedge \text{real } i \leq t. |\text{moebius-mu } i * H (t / \text{real } i)|)$

**unfolding**  $H'\text{-def sum-upto-def}$  **by**  $(\text{rule sum-abs})$

**also have**  $\dots \leq (\sum i | 0 < i \wedge \text{real } i \leq t. 1 * ((t / \text{real } i) / 2))$

**unfolding**  $\text{abs-mult}$  **using that**

by (intro sum-mono mult-mono abs-moebius-mu-le H-bound) simp-all  
 also have ... =  $t / 2 * \text{sum-upto } (\lambda i. 1 / \text{real } i) t$   
 by (simp add: sum-upto-def sum-distrib-left sum-distrib-right mult-ac)  
 finally show ?thesis .  
 qed  
 hence  $H' \in O(\lambda t. t * \text{sum-upto } (\lambda i. 1 / \text{real } i) t)$   
 using sum-upto-inverse-bound(1)  
 by (intro bigoI[of - 1/2] eventually-mono[OF eventually-ge-at-top[of 0::real]])  
 (auto elim!: eventually-mono simp: abs-mult)  
 also have  $(\lambda t. t * \text{sum-upto } (\lambda i. 1 / \text{real } i) t) \in O(\lambda t. t * \ln t)$   
 by (intro landau-o.big.mult sum-upto-inverse-bigo) simp-all  
 finally have  $H'$ -bigo:  $H' \in O(\lambda x. x * \ln x)$  .  
  
 {  
 fix  $x :: \text{real}$  assume  $x \geq 0$   
 have sum-upto  $(\lambda n. \text{real } (\text{totient } n)) x = \text{sum-upto } (\lambda n. \text{of-int } (\text{int } (\text{totient } n)))$   
 $x$   
 by simp  
 also have ... =  $\text{sum-upto } (\lambda n. \text{moebius-mu } n * \text{sum-upto real } (x / \text{real } n)) x$   
 by (subst totient-conv-moebius-mu) (simp add: sum-upto-dirichlet-prod of-int-dirichlet-prod)  
 also have ... =  $\text{sum-upto } (\lambda n. \text{moebius-mu } n * ((x / \text{real } n) ^ 2 / 2 + H (x$   
 $/ \text{real } n))) x$  using  $x$   
 by (intro sum-upto-cong) (simp-all add: H)  
 also have ... =  $x^2 / 2 * \text{sum-upto } (\lambda n. \text{moebius-mu } n / \text{real } n ^ 2) x + H'$   
 $x$   
 by (simp add: sum-upto-def H'-def sum.distrib ring-distrib  
 sum-distrib-left sum-distrib-right power-divide mult-ac)  
 also have  $\text{sum-upto } (\lambda n. \text{moebius-mu } n / \text{real } n ^ 2) x =$   
 $(\sum n \in \{.. < \text{Suc } (\text{nat } \lfloor x \rfloor\}) . \text{moebius-mu } n / \text{real } n ^ 2)$   
 unfolding sum-upto-altdef by (intro sum.mono-neutral-cong-left refl) auto  
 also have ... =  $6 / \text{pi} ^ 2 - G x$   
 using sums-split-initial-segment[OF moebius-over-square-sums, of Suc (nat  
 $\lfloor x \rfloor$ )]  
 by (auto simp: sums-iff algebra-simps G-def)  
 finally have  $\text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) x = 3 / \text{pi}^2 * x^2 - x^2 / 2 * G x$   
 $+ H' x$   
 by (simp add: algebra-simps)  
 }  
 hence  $(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) x - 3 / \text{pi}^2 * x^2) \in$   
 $\Theta(\lambda x. -(x^2) / 2) * G x + H' x$   
 by (intro bighetaI-cong eventually-mono[OF eventually-ge-at-top[of 0::real]])  
 (auto elim!: eventually-mono)  
 also have  $(\lambda x. -(x^2) / 2) * G x + H' x \in O(\lambda x. x * \ln x)$   
 proof (intro sum-in-bigo H'-bigo)  
 have  $(\lambda x. -(x^2) / 2) * G x \in O(\lambda x. x^2 * (1 / x))$   
 using moebius-sum-tail-bigo [folded G-def] by (intro landau-o.big.mult)  
 $\text{simp-all}$   
 also have  $(\lambda x :: \text{real}. x^2 * (1 / x)) \in O(\lambda x. x * \ln x)$  by simp  
 finally show  $(\lambda x. -(x^2) / 2) * G x \in O(\lambda x. x * \ln x)$  .

qed  
 finally show ?thesis .  
 qed

**theorem summatory-totient-asymptotics'**:  
 $(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) x) = o(\lambda x. 3 / \pi^2 * x^2) + o O(\lambda x. x * \ln x)$   
 using summatory-totient-asymptotics  
 by (subst set-minus-plus [symmetric]) (simp-all add: fun-diff-def)

**theorem summatory-totient-asymptotics''**:  
 $\text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) \sim_{[\text{at-top}]} (\lambda x. 3 / \pi^2 * x^2)$   
**proof** –  
 have  $(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) x - 3 / \pi^2 * x^2) \in O(\lambda x. x * \ln x)$   
 by (rule summatory-totient-asymptotics)  
 also have  $(\lambda x. x * \ln x) \in o(\lambda x. 3 / \pi^2 * x^2)$  by simp  
 finally show ?thesis by (simp add: asymp-equiv-altdef)  
 qed

### 13.3 Asymptotic distribution of squarefree numbers

**lemma le-sqrt-iff**:  $x \geq 0 \implies x \leq \text{sqrt } y \iff x^2 \leq y$   
 using real-sqrt-le-iff[of  $x^2$   $y$ ] by (simp del: real-sqrt-le-iff)

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

**proof** –  
**define**  $f :: \text{nat} \Rightarrow \text{real}$  **where**  $f = (\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } 1)$   
**define**  $g :: \text{nat} \Rightarrow \text{real}$  **where**  $g = \text{dirichlet-prod } (\text{ind squarefree}) \text{ moebius-mu}$

**interpret**  $g$ : multiplicative-function  $g$  **unfolding**  $g$ -def  
 by (intro multiplicative-dirichlet-prod squarefree.multiplicative-function-axioms  
 moebius-mu.multiplicative-function-axioms)

**interpret**  $g$ : multiplicative-function'  $g$   $\lambda p k. \text{if } k = 2 \text{ then } -1 \text{ else } 0$   $\lambda-. 0$

**proof**  
**interpret**  $g'$ : multiplicative-dirichlet-prod' ind squarefree moebius-mu  
 $\lambda p k. \text{if } 1 < k \text{ then } 0 \text{ else } 1$   $\lambda p k. \text{if } k = 1 \text{ then } -1 \text{ else } 0$   $\lambda-. 1$   $\lambda-. -1$   
 by (intro multiplicative-dirichlet-prod'.intro squarefree.multiplicative-function'-axioms

moebius-mu.multiplicative-function'-axioms)

**fix**  $p k :: \text{nat}$  **assume** prime  $p k > 0$   
**hence**  $g (p \wedge k) = (\sum_{i \in \{0 .. < k\}}. (\text{if } \text{Suc } 0 < i \text{ then } 0 \text{ else } 1) * (\text{if } k - i = \text{Suc } 0 \text{ then } -1 \text{ else } 0))$

by (auto simp:  $g'$ .prime-power  $g$ -def)

**also have**  $\dots = (\sum_{i \in \{0 .. < k\}}. (\text{if } k = 2 \text{ then } -1 \text{ else } 0))$

by (intro sum.cong refl) auto

**also from**  $\langle k > 0 \rangle$  **have**  $\dots = (\text{if } k = 2 \text{ then } -1 \text{ else } 0)$  by simp

**finally show**  $g (p \wedge k) = \dots$  .

qed simp-all

**have** mult-g-square: multiplicative-function  $(\lambda n. g (n \wedge 2))$

by standard (simp-all add: power-mult-distrib g.mult-coprime)

have g-square:  $g(m^2) = \text{moebius-mu } m$  for  $m$   
 using mult-g-square moebius-mu.multiplicative-function-axioms  
 proof (rule multiplicative-function-eqI)  
 fix  $p k :: \text{nat}$  assume \*: prime  $p k > 0$   
 have  $g((p^k)^2) = g(p^{2k})$  by (simp add: power-mult [symmetric] mult-ac)  
 also from \* have ... = (if  $k = 1$  then  $-1$  else  $0$ ) by (simp add: g.prime-power)  
 also from \* have ... = moebius-mu  $(p^k)$  by (simp add: moebius-mu.prime-power)  
 finally show  $g((p^k)^2) = \text{moebius-mu } (p^k)$ .  
 qed

have g-nonsquare:  $g m = 0$  if  $\neg \text{is-square } m$  for  $m$   
 proof (cases  $m = 0$ )  
 case False  
 from that False obtain  $p$  where  $p$ : prime  $p$  odd (multiplicity  $p m$ )  
 using is-nth-power-conv-multiplicity-nat[of  $2 m$ ] by auto  
 from  $p$  have multiplicity  $p m \neq 2$  by auto  
 moreover from  $p$  have  $p \in \text{prime-factors } m$   
 by (auto simp: prime-factors-multiplicity intro!: Nat.gr0I)  
 ultimately have  $(\prod_{p \in \text{prime-factors } m} \text{if multiplicity } p m = 2 \text{ then } -1 \text{ else } 0 :: \text{real}) = 0$   
 (is ?P = -) by auto  
 also have  $?P = g m$  using False by (subst g.prod-prime-factors') auto  
 finally show ?thesis .  
 qed auto

have abs-g-le:  $\text{abs } (g m) \leq 1$  for  $m$   
 by (cases is-square  $m$ )  
 (auto simp: g-square g-nonsquare abs-moebius-mu-le elim!: is-nth-powerE)

have fds-g:  $\text{fds } g = \text{fds-ind squarefree } * \text{fds moebius-mu}$   
 by (rule fds-eqI) (simp add: g-def fds-nth-mult)  
 have  $\text{fds } g * \text{fds-zeta} = \text{fds-ind squarefree } * (\text{fds-zeta} * \text{fds moebius-mu})$   
 by (simp add: fds-g mult-ac)  
 also have  $\text{fds-zeta} * \text{fds moebius-mu} = (1 :: \text{real fds})$   
 by (rule fds-zeta-times-moebius-mu)  
 finally have \*:  $\text{fds-ind squarefree} = \text{fds } g * \text{fds-zeta}$  by simp  
 have ind-squarefree:  $\text{ind squarefree} = \text{dirichlet-prod } g f$   
 proof  
 fix  $n :: \text{nat}$   
 from \* show  $\text{ind squarefree } n = \text{dirichlet-prod } g f n$   
 by (cases  $n = 0$ ) (simp-all add: fds-eq-iff fds-nth-mult f-def)  
 qed

define  $H :: \text{real} \Rightarrow \text{real}$   
 where  $H = (\lambda x. \text{sum-upto } (\lambda m. g(m^2) * (\text{real-of-int } \lfloor x / \text{real } (m^2) \rfloor - x / \text{real } (m^2))) (\text{sqrt } x))$

**define**  $J$  **where**  $J = (\lambda x :: \text{real}. (\sum n. \text{moebius-mu } (n + \text{Suc } (\text{nat } \lfloor x \rfloor)) / (n + \text{Suc } (\text{nat } \lfloor x \rfloor))^2))$

**have** *eventually*  $(\lambda x. \text{norm } (H x) \leq 1 * \text{norm } (\text{sqrt } x))$  *at-top*

**using** *eventually-ge-at-top*[*of*  $0 :: \text{real}$ ]

**proof** *eventually-elim*

**case** (*elim*  $x$ )

**have**  $\text{abs } (H x) \leq \text{sum-upto } (\lambda m. \text{abs } (g (m^2) * (\text{real-of-int } \lfloor x / \text{real } (m^2) \rfloor$

$x / \text{real } (m^2))) (\text{sqrt } x)$  (*is -  $\leq$  ?S*) **unfolding** *H-def*

*sum-upto-def*

**by** (*rule sum-abs*)

**also have**  $x / (\text{real } m)^2 - \text{real-of-int } \lfloor x / (\text{real } m)^2 \rfloor \leq 1$  **for**  $m$  **by** *linarith*

**hence**  $?S \leq \text{sum-upto } (\lambda m. 1 * 1) (\text{sqrt } x)$  **unfolding** *abs-mult sum-upto-def*

**by** (*intro sum-mono mult-mono abs-g-le simp-all*)

**also have**  $\dots = \text{of-int } \lfloor \text{sqrt } x \rfloor$  **using** *elim* **by** (*simp add: sum-upto-altdef*)

**also have**  $\dots \leq \text{sqrt } x$  **by** *linarith*

**finally show**  $?case$  **using** *elim* **by** *simp*

**qed**

**hence** *H-bigo*:  $H \in O(\lambda x. \text{sqrt } x)$  **by** (*rule bigoI*)

**let**  $?A = \lambda x. \text{card } \{n. \text{real } n \leq x \wedge \text{squarefree } n\}$

**have** *eventually*  $(\lambda x. ?A x - 6 / \text{pi}^2 * x = (-x) * J (\text{sqrt } x) + H x)$  *at-top*

**using** *eventually-ge-at-top*[*of*  $0 :: \text{real}$ ]

**proof** *eventually-elim*

**fix**  $x :: \text{real}$  **assume**  $x \geq 0$

**have**  $\{n. \text{real } n \leq x \wedge \text{squarefree } n\} = \{n. n > 0 \wedge \text{real } n \leq x \wedge \text{squarefree } n\}$

**by** (*auto intro!: Nat.gr0I*)

**also have**  $\text{card } \dots = \text{sum-upto } (\text{ind squarefree } :: \text{nat} \Rightarrow \text{real}) x$

**by** (*rule sum-upto-ind [symmetric]*)

**also have**  $\dots = \text{sum-upto } (\lambda d. g d * \text{sum-upto } f (x / \text{real } d)) x$  (*is - = ?S*)

**unfolding** *ind-squarefree* **by** (*rule sum-upto-dirichlet-prod*)

**also have**  $\text{sum } f \{0 < .. \text{nat } \lfloor x / \text{real } i \rfloor\} = \text{of-int } \lfloor x / \text{real } i \rfloor$  **if**  $i > 0$  **for**  $i$

**using**  $x$  **by** (*simp add: f-def*)

**hence**  $?S = \text{sum-upto } (\lambda d. g d * \text{of-int } \lfloor x / \text{real } d \rfloor) x$

**unfolding** *sum-upto-altdef* **by** (*intro sum.cong refl simp-all*)

**also have**  $\dots = \text{sum-upto } (\lambda m. g (m^2) * \text{of-int } \lfloor x / \text{real } (m^2) \rfloor) (\text{sqrt } x)$

**unfolding** *sum-upto-def*

**proof** (*intro sum.reindex-bij-betw-not-neutral [symmetric]*)

**show** *bij-betw power2*  $(\{i. 0 < i \wedge \text{real } i \leq \text{sqrt } x\} - \{\})$

$(\{i. 0 < i \wedge \text{real } i \leq x\} - \{i \in \{0 < .. \text{nat } \lfloor x \rfloor\}. \neg \text{is-square } i\})$

**by** (*auto simp: bij-betw-def inj-on-def power-eq-iff-eq-base le-sqrt-iff is-nth-power-def le-nat-iff le-floor-iff*)

**qed** (*auto simp: g-nonsquare*)

**also have**  $\dots = x * \text{sum-upto } (\lambda m. g (m^2) / \text{real } m^2) (\text{sqrt } x) + H x$

**by** (*simp add: H-def sum-upto-def sum.distrib ring-distrib sum-subtractf sum-distrib-left sum-distrib-right mult-ac*)

**also have**  $\text{sum-upto } (\lambda m. g (m^2) / \text{real } m^2) (\text{sqrt } x) =$

$sum\text{-upto } (\lambda m. moebius\text{-mu } m / real\ m \wedge 2) (sqrt\ x)$   
**unfolding**  $sum\text{-upto}\text{-altdef}$  **by** ( $intro\ sum.\text{cong}\ refl$ ) ( $simp\text{-all}\ add: g\text{-square}$ )  
**also have**  $sum\text{-upto } (\lambda m. moebius\text{-mu } m / (real\ m)^2) (sqrt\ x) =$   
 $(\sum m < Suc\ (nat\ \lfloor sqrt\ x \rfloor). moebius\text{-mu } m / (real\ m) \wedge 2)$   
**unfolding**  $sum\text{-upto}\text{-altdef}$  **by** ( $intro\ sum.\text{mono}\text{-neutral}\text{-cong}\text{-left}$ )  $auto$   
**also have**  $\dots = (6 / pi \wedge 2 - J\ (sqrt\ x))$   
**using**  $sums\text{-split}\text{-initial}\text{-segment}[OF\ moebius\text{-over}\text{-square}\text{-sums, of } Suc\ (nat\ \lfloor sqrt\ x \rfloor)]$   
**by** ( $auto\ simp: sums\text{-iff}\ algebra\text{-simps}\ J\text{-def}\ sum\text{-upto}\text{-altdef}$ )  
**finally show**  $?A\ x - 6 / pi^2 * x = (-x) * J\ (sqrt\ x) + H\ x$   
**by** ( $simp\ add: algebra\text{-simps}$ )  
**qed**  
**hence**  $(\lambda x. ?A\ x - 6 / pi^2 * x) \in \Theta(\lambda x. (-x) * J\ (sqrt\ x) + H\ x)$   
**by** ( $rule\ bighetaI\text{-cong}$ )  
**also have**  $(\lambda x. (-x) * J\ (sqrt\ x) + H\ x) \in O(\lambda x. sqrt\ x)$   
**proof** ( $intro\ sum\text{-in}\text{-bigo}\ H\text{-bigo}$ )  
**have**  $(\lambda x. J\ (sqrt\ x)) \in O(\lambda x. 1 / sqrt\ x)$  **unfolding**  $J\text{-def}$   
**using**  $moebius\text{-sum}\text{-tail}\text{-bigo}\ sqrt\text{-at}\text{-top}$  **by** ( $rule\ landau\text{-o}\text{-big}\text{-compose}$ )  
**hence**  $(\lambda x. (-x) * J\ (sqrt\ x)) \in O(\lambda x. x * (1 / sqrt\ x))$   
**by** ( $intro\ landau\text{-o}\text{-big}\text{-mult}$ )  $simp\text{-all}$   
**also have**  $(\lambda x::real. x * (1 / sqrt\ x)) \in \Theta(\lambda x. sqrt\ x)$   
**by** ( $intro\ bighetaI\text{-cong}\ eventually\text{-mono}[OF\ eventually\text{-gt}\text{-at}\text{-top}[of\ 0::real]]$ )  
 $(auto\ simp: field\text{-simps})$   
**finally show**  $(\lambda x. (-x) * J\ (sqrt\ x)) \in O(\lambda x. sqrt\ x)$  .  
**qed**  
**finally show**  $?thesis$  .  
**qed**

**theorem**  $squarefree\text{-asymptotics}'$ :

$(\lambda x. card\ \{n. real\ n \leq x \wedge squarefree\ n\}) = o(\lambda x. 6 / pi^2 * x) + o\ O(\lambda x. sqrt\ x)$   
**using**  $squarefree\text{-asymptotics}$   
**by** ( $subst\ set\text{-minus}\text{-plus}\ [symmetric]$ ) ( $simp\text{-all}\ add: fun\text{-diff}\text{-def}$ )

**theorem**  $squarefree\text{-asymptotics}''$ :

$(\lambda x. card\ \{n. real\ n \leq x \wedge squarefree\ n\}) \sim[at\text{-top}] (\lambda x. 6 / pi^2 * x)$

**proof** –

**have**  $(\lambda x. card\ \{n. real\ n \leq x \wedge squarefree\ n\} - 6 / pi^2 * x) \in O(\lambda x. sqrt\ x)$   
**by** ( $rule\ squarefree\text{-asymptotics}$ )  
**also have**  $(sqrt\ :: real \Rightarrow real) \in \Theta(\lambda x. x\ powr\ (1/2))$   
**by** ( $intro\ bighetaI\text{-cong}\ eventually\text{-mono}[OF\ eventually\text{-ge}\text{-at}\text{-top}[of\ 0::real]]$ )  
 $(auto\ simp: powr\text{-half}\text{-sqrt})$   
**also have**  $(\lambda x::real. x\ powr\ (1/2)) \in o(\lambda x. 6 / pi \wedge 2 * x)$  **by**  $simp$   
**finally show**  $?thesis$  **by** ( $simp\ add: asymp\text{-equiv}\text{-altdef}$ )  
**qed**

## 13.4 The hyperbola method

**lemma**  $hyperbola\text{-method}\text{-bigo}$ :

**fixes**  $f\ g :: nat \Rightarrow 'a :: real\text{-normed}\text{-field}$

**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** –  
**let**  $?A = \lambda x. (\text{sum-upto } (\lambda n. f\ n * \text{sum-upto } g\ (x / \text{real } n))\ (\text{sqrt } x) - R\ x) +$   
 $(\text{sum-upto } (\lambda n. \text{sum-upto } f\ (x / \text{real } n) * g\ n)\ (\text{sqrt } x) - S\ x) +$   
 $(-\text{sum-upto } f\ (\text{sqrt } x) * \text{sum-upto } g\ (\text{sqrt } x) - T\ x)$   
**have**  $(\lambda x. \text{sum-upto } (\text{dirichlet-prod } f\ g)\ x - (R\ x + S\ x - T\ x)) \in \Theta(?A)$   
**by**  $(\text{intro } \text{bigthetaI-cong } \text{eventually-mono}[OF\ \text{eventually-ge-at-top}[of\ 0::\text{real}]])$   
 $(\text{auto } \text{simp: } \text{hyperbola-method-sqrt})$   
**also from**  $\text{assms}$  **have**  $?A \in O(b)$   
**by**  $(\text{intro } \text{sum-in-bigo}(1))\ (\text{simp-all } \text{only: } \text{landau-o.big.uminus-in-iff})$   
**finally show**  $?thesis$  .  
**qed**

**lemma**  $\text{frac-le-1: } \text{frac } x \leq 1$   
**unfolding**  $\text{frac-def}$  **by**  $\text{linarith}$

**lemma**  $\text{ln-minus-ln-floor-bound:}$

**assumes**  $x \geq 2$   
**shows**  $\ln\ x - \ln\ (\text{floor } x) \in \{0..<1 / (x - 1)\}$   
**proof** –  
**from**  $\text{assms}$  **have**  $\ln\ (\text{floor } x) \geq \ln\ (x - 1)$  **by**  $(\text{subst } \text{ln-le-cancel-iff})\ \text{simp-all}$   
**hence**  $\ln\ x - \ln\ (\text{floor } x) \leq \ln\ ((x - 1) + 1) - \ln\ (x - 1)$  **by**  $\text{simp}$   
**also from**  $\text{assms}$  **have**  $\dots < 1 / (x - 1)$  **by**  $(\text{intro } \text{ln-diff-le-inverse})\ \text{simp-all}$   
**finally have**  $\ln\ x - \ln\ (\text{floor } x) < 1 / (x - 1)$  **by**  $\text{simp}$   
**moreover from**  $\text{assms}$  **have**  $\ln\ x \geq \ln\ (\text{of-int } \lfloor x \rfloor)$  **by**  $(\text{subst } \text{ln-le-cancel-iff})$   
 $\text{simp-all}$   
**ultimately show**  $?thesis$  **by**  $\text{simp}$   
**qed**

**lemma**  $\text{ln-minus-ln-floor-bigo:}$

$(\lambda x::\text{real. } \ln\ x - \ln\ (\text{floor } x)) \in O(\lambda x. 1 / x)$   
**proof** –  
**have**  $\text{eventually } (\lambda x. \text{norm } (\ln\ x - \ln\ (\text{floor } x)) \leq 1 * \text{norm } (1 / (x - 1)))$   
 $\text{at-top}$   
**using**  $\text{eventually-ge-at-top}[of\ 2::\text{real}]$   
**proof**  $\text{eventually-elim}$   
**case**  $(\text{elim } x)$   
**with**  $\text{ln-minus-ln-floor-bound}[OF\ \text{this}]$  **show**  $?case$  **by**  $\text{auto}$   
**qed**  
**hence**  $(\lambda x::\text{real. } \ln\ x - \ln\ (\text{floor } x)) \in O(\lambda x. 1 / (x - 1))$  **by**  $(\text{rule } \text{bigoI})$   
**also have**  $(\lambda x::\text{real. } 1 / (x - 1)) \in O(\lambda x. 1 / x)$  **by**  $\text{simp}$   
**finally show**  $?thesis$  .  
**qed**

**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** –  
**define**  $R$  **where**  $R = (\lambda x. \sum i \in \{0 <.. \text{nat } \lfloor \text{sqrt } x \rfloor\}. \text{frac } (x / \text{real } i))$   
**define**  $S$  **where**  $S = (\lambda x. \ln (\text{real } (\text{nat } \lfloor \text{sqrt } x \rfloor)) - \ln x / 2)$   
**have**  $R$ -bound:  $R x \in \{0.. \text{sqrt } x\}$  **if**  $x: x \geq 0$  **for**  $x$   
**proof** –  
**have**  $R x \leq (\sum i \in \{0 <.. \text{nat } \lfloor \text{sqrt } x \rfloor\}. 1)$  **unfolding**  $R$ -def **by** (*intro sum-mono frac-le-1*)  
**also from**  $x$  **have**  $\dots = \text{of-int } \lfloor \text{sqrt } x \rfloor$  **by** *simp*  
**also have**  $\dots \leq \text{sqrt } x$  **by** *simp*  
**finally have**  $R x \leq \text{sqrt } x$  .  
**moreover have**  $R x \geq 0$  **unfolding**  $R$ -def **by** (*intro sum-nonneg simp-all*)  
**ultimately show** *?thesis* **by** *simp*  
**qed**  
**have**  $R$ -bound':  $\text{norm } (R x) \leq 1 * \text{norm } (\text{sqrt } x)$  **if**  $x \geq 0$  **for**  $x$   
**using**  $R$ -bound[*OF that*] **that** **by** *simp*  
**have**  $R$ -bigo:  $R \in O(\text{sqrt})$  **using** *eventually-ge-at-top[of 0::real]*  
**by** (*intro bigoI[of - 1], elim eventually-mono*) (*rule R-bound'*)  
  
**have** *eventually*  $(\lambda x. \text{sum-upto } (\lambda n. \text{sum-upto } (\lambda-. 1 :: \text{real}) (x / \text{real } n)) (\text{sqrt } x) =$   
 $x * \text{harm } (\text{nat } \lfloor \text{sqrt } x \rfloor) - R x)$  *at-top*  
**using** *eventually-ge-at-top[of 0 :: real]*  
**proof** *eventually-elim*  
**case** (*elim x*)  
**have**  $\text{sum-upto } (\lambda n. \text{sum-upto } (\lambda-. 1 :: \text{real}) (x / \text{real } n)) (\text{sqrt } x) =$   
 $(\sum i \in \{0 <.. \text{nat } \lfloor \text{sqrt } x \rfloor\}. \text{of-int } \lfloor x / \text{real } i \rfloor)$  **using** *elim*  
**by** (*simp add: sum-upto-altdef*)  
**also have**  $\dots = x * (\sum i \in \{0 <.. \text{nat } \lfloor \text{sqrt } x \rfloor\}. 1 / \text{real } i) - R x$   
**by** (*simp add: sum-subtractf frac-def R-def sum-distrib-left*)  
**also have**  $\{0 <.. \text{nat } \lfloor \text{sqrt } x \rfloor\} = \{1.. \text{nat } \lfloor \text{sqrt } x \rfloor\}$  **by** *auto*  
**also have**  $(\sum i \in \dots. 1 / \text{real } i) = \text{harm } (\text{nat } \lfloor \text{sqrt } x \rfloor)$  **by** (*simp add: harm-def divide-simps*)  
**finally show** *?case* .  
**qed**  
**hence**  $(\lambda x. \text{sum-upto } (\lambda n. \text{sum-upto } (\lambda-. 1 :: \text{real}) (x / \text{real } n)) (\text{sqrt } x) - (x * \ln x / 2 + \text{euler-mascheroni} * x)) \in$   
 $\Theta(\lambda x. x * (\text{harm } (\text{nat } \lfloor \text{sqrt } x \rfloor) - (\ln (\text{nat } \lfloor \text{sqrt } x \rfloor) + \text{euler-mascheroni})))$   
 $- R x + x * S x)$   
**(is -  $\in \Theta(?A)$ )**  
**by** (*intro bigthetaI-cong*) (*elim eventually-mono, simp-all add: algebra-simps S-def*)  
**also have**  $?A \in O(\text{sqrt})$   
**proof** (*intro sum-in-bigo*)  
**have**  $(\lambda x. - S x) \in \Theta(\lambda x. \ln (\text{sqrt } x) - \ln (\text{of-int } \lfloor \text{sqrt } x \rfloor))$   
**by** (*intro bigthetaI-cong eventually-mono [OF eventually-ge-at-top[of 1::real]]*)

```

      (auto simp: S-def ln-sqrt)
    also have  $(\lambda x. \ln (\text{sqrt } x) - \ln (\text{of-int } \lfloor \text{sqrt } x \rfloor)) \in O(\lambda x. 1 / \text{sqrt } x)$ 
      by (rule landau-o.big.compose[OF ln-minus-ln-floor-bigo sqrt-at-top])
    finally have  $(\lambda x. x * S x) \in O(\lambda x. x * (1 / \text{sqrt } x))$  by (intro landau-o.big.mult)
simp-all
    also have  $(\lambda x::\text{real}. x * (1 / \text{sqrt } x)) \in \Theta(\lambda x. \text{sqrt } x)$ 
      by (intro bighetaI-cong eventually-mono [OF eventually-gt-at-top[of 0::real]])

      (auto simp: field-simps)
    finally show  $(\lambda x. x * S x) \in O(\text{sqrt})$  .
next
  let ?f =  $\lambda x::\text{real}. \text{harm } (\text{nat } \lfloor \text{sqrt } x \rfloor) - (\ln (\text{real } (\text{nat } \lfloor \text{sqrt } x \rfloor)) + \text{euler-mascheroni})$ 
  have ?f  $\in O(\lambda x. 1 / \text{real } (\text{nat } \lfloor \text{sqrt } x \rfloor))$ 
  proof (rule landau-o.big.compose[of - - -  $\lambda x. \text{nat } \lfloor \text{sqrt } x \rfloor$ ])
    show filterlim  $(\lambda x::\text{real}. \text{nat } \lfloor \text{sqrt } x \rfloor)$  at-top at-top
      by (intro filterlim-compose[OF filterlim-nat-sequentially]
          filterlim-compose[OF filterlim-floor-sequentially] sqrt-at-top)
  next
    show  $(\lambda a. \text{harm } a - (\ln (\text{real } a) + \text{euler-mascheroni})) \in O(\lambda a. 1 / \text{real } a)$ 
      by (rule harm-expansion-bigo-simple2)
  qed
  also have  $(\lambda x. 1 / \text{real } (\text{nat } \lfloor \text{sqrt } x \rfloor)) \in O(\lambda x. 1 / (\text{sqrt } x - 1))$ 
  proof (rule bigoI[of - 1], use eventually-ge-at-top[of 2] in eventually-elim)
    case (elim x)
    have  $\text{sqrt } x \leq 1 + \text{real-of-int } \lfloor \text{sqrt } x \rfloor$  by linarith
    with elim show ?case by (simp add: field-simps)
  qed
  also have  $(\lambda x::\text{real}. 1 / (\text{sqrt } x - 1)) \in O(\lambda x. 1 / \text{sqrt } x)$ 
    by (rule landau-o.big.compose[OF - sqrt-at-top]) simp-all
  finally have  $(\lambda x. x * ?f x) \in O(\lambda x. x * (1 / \text{sqrt } x))$ 
    by (intro landau-o.big.mult landau-o.big-refl)
  also have  $(\lambda x::\text{real}. x * (1 / \text{sqrt } x)) \in \Theta(\lambda x. \text{sqrt } x)$ 
    by (intro bighetaI-cong eventually-mono[OF eventually-gt-at-top[of 0::real]])
      (auto elim!: eventually-mono simp: field-simps)
  finally show  $(\lambda x. x * ?f x) \in O(\text{sqrt})$  .
qed fact+
finally show ?thesis .
qed

```

**lemma** *sum-upto-sqrt-bound*:

```

  assumes  $x \geq (0 :: \text{real})$ 
  shows  $\text{norm } ((\text{sum-upto } (\lambda-. 1) (\text{sqrt } x))^2 - x) \leq 2 * \text{norm } (\text{sqrt } x)$ 
proof -
  from x have  $0 \leq 2 * \text{sqrt } x * (1 - \text{frac } (\text{sqrt } x)) + \text{frac } (\text{sqrt } x) ^ 2$ 
    by (intro add-nonneg-nonneg mult-nonneg-nonneg) (simp-all add: frac-le-1)
  also from x have  $\dots = (\text{sqrt } x - \text{frac } (\text{sqrt } x)) ^ 2 - x + 2 * \text{sqrt } x$ 
    by (simp add: algebra-simps power2-eq-square)
  also have  $\text{sqrt } x - \text{frac } (\text{sqrt } x) = \text{of-int } \lfloor \text{sqrt } x \rfloor$  by (simp add: frac-def)

```

**finally have**  $(\text{of-int } \lfloor \text{sqrt } x \rfloor)^2 - x \geq -2 * \text{sqrt } x$  **by**  $(\text{simp add: algebra-simps})$   
**moreover from**  $x$  **have**  $\text{of-int } (\lfloor \text{sqrt } x \rfloor)^2 \leq \text{sqrt } x^2$   
**by**  $(\text{intro power-mono})$   $\text{simp-all}$   
**with**  $x$  **have**  $\text{of-int } (\lfloor \text{sqrt } x \rfloor)^2 - x \leq 0$  **by**  $\text{simp}$   
**ultimately have**  $\text{sum-upto } (\lambda-. 1) (\text{sqrt } x)^2 - x \in \{-2 * \text{sqrt } x..0\}$   
**using**  $x$  **by**  $(\text{simp add: sum-upto-altdef})$   
**with**  $x$  **show**  $?thesis$  **by**  $\text{simp}$   
**qed**

**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** –

**let**  $?f = \lambda x. x * \ln x / 2 + \text{euler-mascheroni} * x$

**have**  $(\lambda x. \text{sum-upto } (\text{dirichlet-prod } (\lambda-. 1 :: \text{real}) (\lambda-. 1)) x - (?f x + ?f x - x))$   
 $\in O(\text{sqrt})$

**(is**  $?g \in -)$

**proof**  $(\text{rule hyperbola-method-bigo})$

**have**  $\text{eventually } (\lambda x :: \text{real}. \text{norm } (\text{sum-upto } (\lambda-. 1) (\text{sqrt } x)^2 - x) \leq$   
 $2 * \text{norm } (\text{sqrt } x)) \text{ at-top}$

**using**  $\text{eventually-ge-at-top}[of 0 :: \text{real}]$  **by**  $\text{eventually-elim } (\text{rule sum-upto-sqrt-bound})$

**thus**  $(\lambda x :: \text{real}. \text{sum-upto } (\lambda-. 1) (\text{sqrt } x) * \text{sum-upto } (\lambda-. 1) (\text{sqrt } x) - x) \in$   
 $O(\text{sqrt})$

**by**  $(\text{intro bigoI}[of - 2])$   $(\text{simp-all add: power2-eq-square})$

**next**

**show**  $(\lambda x. \text{sum-upto } (\lambda n. 1 * \text{sum-upto } (\lambda-. 1) (x / \text{real } n)) (\text{sqrt } x) -$   
 $(x * \ln x / 2 + \text{euler-mascheroni} * x)) \in O(\text{sqrt})$

**using** *divisor-count-asymptotics-aux* **by**  $\text{simp}$

**next**

**show**  $(\lambda x. \text{sum-upto } (\lambda n. \text{sum-upto } (\lambda-. 1) (x / \text{real } n) * 1) (\text{sqrt } x) -$   
 $(x * \ln x / 2 + \text{euler-mascheroni} * x)) \in O(\text{sqrt})$

**using** *divisor-count-asymptotics-aux* **by**  $\text{simp}$

**qed**

**also have**  $\text{divisor-count } n = \text{dirichlet-prod } (\lambda-. 1) (\lambda-. 1) n$  **for**  $n$

**using** *fds-divisor-count*

**by**  $(\text{cases } n = 0)$   $(\text{simp-all add: fds-eq-iff power2-eq-square fds-nth-mult})$

**hence**  $?g = (\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{divisor-count } n)) x -$   
 $(x * \ln x + (2 * \text{euler-mascheroni} - 1) * x))$

**by**  $(\text{intro ext})$   $(\text{simp-all add: algebra-simps dirichlet-prod-def})$

**finally show**  $?thesis$  .

**qed**

**theorem** *summatory-divisor-count-asymptotics'*:

$(\lambda x. \text{sum-upto } (\lambda n. \text{real } (\text{divisor-count } n)) x) = o$

$(\lambda x. x * \ln x + (2 * \text{euler-mascheroni} - 1) * x) + o O(\lambda x. \text{sqrt } x)$

**using** *summatory-divisor-count-asymptotics*

**by**  $(\text{subst set-minus-plus } [\text{symmetric}])$   $(\text{simp-all add: fun-diff-def})$

**theorem** *summatory-divisor-count-asymptotics''*:

$sum\text{-}upto (\lambda n. real (divisor\text{-}count\ n)) \sim[at\text{-}top] (\lambda x. x * \ln x)$   
**proof** –  
**have**  $(\lambda x. sum\text{-}upto (\lambda n. real (divisor\text{-}count\ n)) x - (x * \ln x + (2 * euler\text{-}mascheroni - 1) * x)) \in O(sqrt)$   
**by**  $(rule\ summatory\text{-}divisor\text{-}count\text{-}asymptotics)$   
**also have**  $sqrt \in \Theta(\lambda x. x\ powr (1/2))$   
**by**  $(intro\ bighetaI\text{-}cong\ eventually\text{-}mono [OF\ eventually\text{-}ge\text{-}at\text{-}top[of\ 0::real]])$   
 $(auto\ elim!: eventually\text{-}mono\ simp: powr\text{-}half\text{-}sqrt)$   
**also have**  $(\lambda x::real. x\ powr (1/2)) \in o(\lambda x. x * \ln x + (2 * euler\text{-}mascheroni - 1) * x)$  **by**  $simp$   
**finally have**  $sum\text{-}upto (\lambda n. real (divisor\text{-}count\ n)) \sim[at\text{-}top] (\lambda x. x * \ln x + (2 * euler\text{-}mascheroni - 1) * x)$   
**by**  $(simp\ add: asymp\text{-}equiv\text{-}altdef)$   
**also have**  $\dots \sim[at\text{-}top] (\lambda x. x * \ln x)$  **by**  $(subst\ asymp\text{-}equiv\text{-}add\text{-}right)$   $simp\text{-}all$   
**finally show**  $?thesis$  .  
**qed**

**lemma** *summatory-divisor-eq*:

$sum\text{-}upto (\lambda n. real (divisor\text{-}count\ n)) (real\ m) = card \{(n,d). n \in \{0<..m\} \wedge d\ dvd\ n\}$   
**proof** –  
**have**  $sum\text{-}upto (\lambda n. real (divisor\text{-}count\ n)) m = card (SIGMA\ n:\{0<..m\}. \{d. d\ dvd\ n\})$   
**unfolding**  $sum\text{-}upto\text{-}altdef\ divisor\text{-}count\text{-}def$  **by**  $(subst\ card\text{-}SigmaI)$   $simp\text{-}all$   
**also have**  $(SIGMA\ n:\{0<..m\}. \{d. d\ dvd\ n\}) = \{(n,d). n \in \{0<..m\} \wedge d\ dvd\ n\}$  **by**  $auto$   
**finally show**  $?thesis$  .  
**qed**

**context**

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

**lemma** *mean-divisor-count-asymptotics*:

$(\lambda m. M\ m - (\ln\ m + 2 * euler\text{-}mascheroni - 1)) \in O(\lambda m. 1 / sqrt\ m)$   
**proof** –  
**have**  $(\lambda m. M\ m - (\ln\ m + 2 * euler\text{-}mascheroni - 1)) \in \Theta(\lambda m. (sum\text{-}upto (\lambda n. real (divisor\text{-}count\ n)) (real\ m) - (m * \ln\ m + (2 * euler\text{-}mascheroni - 1) * m)) / m)$   $(is - \in \Theta(?f))$   
**unfolding**  $M\text{-}def$   
**by**  $(intro\ bighetaI\text{-}cong\ eventually\text{-}mono [OF\ eventually\text{-}gt\text{-}at\text{-}top[of\ 0::nat]])$   
 $(auto\ simp: summatory\text{-}divisor\text{-}eq\ field\text{-}simps)$   
**also have**  $?f \in O(\lambda m. sqrt\ m / m)$   
**by**  $(intro\ landau\text{-}o.big.compose[OF - filterlim\text{-}real\text{-}sequentially] landau\text{-}o.big.divide\text{-}right\ summatory\text{-}divisor\text{-}count\text{-}asymptotics\ eventually\text{-}at\text{-}top\text{-}not\text{-}equal)$   
**also have**  $(\lambda m::nat. sqrt\ m / m) \in \Theta(\lambda m. 1 / sqrt\ m)$   
**by**  $(intro\ bighetaI\text{-}cong\ eventually\text{-}mono [OF\ eventually\text{-}gt\text{-}at\text{-}top[of\ 0::nat]])$   
 $(auto\ simp: field\text{-}simps)$

**finally show** *?thesis* .  
**qed**

**theorem** *mean-divisor-count-asymptotics'*:  
 $M = o(\lambda x. \ln x + 2 * euler-mascheroni - 1) + o(O(\lambda x. 1 / \text{sqrt } x))$   
**using** *mean-divisor-count-asymptotics*  
**by** (*subst set-minus-plus [symmetric]*) (*simp-all add: fun-diff-def*)

**theorem** *mean-divisor-count-asymptotics''*:  $M \sim[at-top] \ln$   
**proof** –  
**have**  $(\lambda x. M x - (\ln x + 2 * euler-mascheroni - 1)) \in O(\lambda x. 1 / \text{sqrt } x)$   
**by** (*rule mean-divisor-count-asymptotics*)  
**also have**  $(\lambda x. 1 / \text{sqrt } (\text{real } x)) \in \Theta(\lambda x. x \text{ powr } (-1/2))$   
**using** *eventually-gt-at-top[of 0::nat]*  
**by** (*intro bigthetaI-cong*)  
*(auto elim!: eventually-mono simp: powr-half-sqrt field-simps powr-minus)*  
**also have**  $(\lambda x::nat. x \text{ powr } (-1/2)) \in o(\lambda x. \ln x + 2 * euler-mascheroni - 1)$   
**by** (*intro smallo-real-nat-transfer*) *simp-all*  
**finally have**  $M \sim[at-top] (\lambda x. \ln x + 2 * euler-mascheroni - 1)$   
**by** (*simp add: asymp-equiv-altdef*)  
**also have**  $\dots = (\lambda x::nat. \ln x + (2 * euler-mascheroni - 1))$  **by** (*simp add: algebra-simps*)  
**also have**  $\dots \sim[at-top] (\lambda x::nat. \ln x)$  **by** (*subst asymp-equiv-add-right*) *auto*  
**finally show** *?thesis* .  
**qed**

**end**

### 13.5 The asymptotic ditribution of coprime pairs

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

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

**proof** –  
**define**  $C :: nat \Rightarrow (nat \times nat) \text{ set}$   
**where**  $C = (\lambda N. (\bigcup m \in \{1..N\}. (\lambda n. (m,n)) \text{ ‘ totatives } m))$   
**define**  $D :: nat \Rightarrow (nat \times nat) \text{ set}$   
**where**  $D = (\lambda N. (\bigcup n \in \{1..N\}. (\lambda m. (m,n)) \text{ ‘ totatives } n))$   
**have** *fin*: *finite*  $(C \ N)$  *finite*  $(D \ N)$  **for**  $N$  **unfolding** *C-def D-def*  
**by** (*intro finite-UN-I finite-imageI; simp*)**+**

**have**  $*$ :  $\text{card } (A \ N) = 2 * (\sum m \in \{0 <..N\}. \text{totient } m) - 1$  **if**  $N: N > 0$  **for**  $N$

**proof** –  
**have**  $A \ N = C \ N \cup D \ N$   
**by** (*auto simp add: A-def C-def D-def totatives-def image-iff ac-simps*)

**also have**  $\text{card } \dots = \text{card } (C N) + \text{card } (D N) - \text{card } (C N \cap D N)$   
**using** *card-Un-Int[OF fin[of N]]* **by** *arith*  
**also have**  $C N \cap D N = \{(1, 1)\}$  **using**  $N$  **by** (*auto simp: image-iff totatives-def C-def D-def*)  
**also have**  $D N = (\lambda(x,y). (y,x)) \text{ ` } C N$  **by** (*simp add: image-UN image-image C-def D-def*)  
**also have**  $\text{card } \dots = \text{card } (C N)$  **by** (*rule card-image*) (*simp add: inj-on-def C-def*)  
**also have**  $\text{card } (C N) = (\sum m \in \{1..N\}. \text{card } ((\lambda n. (m,n)) \text{ ` } \text{totatives } m))$   
**unfolding** *C-def* **by** (*intro card-UN-disjoint*) *auto*  
**also have**  $\dots = (\sum m \in \{1..N\}. \text{totient } m)$  **unfolding** *totient-def*  
**by** (*subst card-image*) (*auto simp: inj-on-def*)  
**also have**  $\dots = (\sum m \in \{0 <..N\}. \text{totient } m)$  **by** (*intro sum.cong refl*) *auto*  
**finally show**  $\text{card } (A N) = 2 * \dots - 1$  **by** *simp*  
**qed**  
**have** \*\*:  $(\sum m \in \{0 <..N\}. \text{totient } m) \geq 1$  **if**  $N \geq 1$  **for**  $N$   
**proof** –  
**have**  $1 \leq N$  **by** *fact*  
**also have**  $N = (\sum m \in \{0 <..N\}. 1)$  **by** *simp*  
**also have**  $(\sum m \in \{0 <..N\}. 1) \leq (\sum m \in \{0 <..N\}. \text{totient } m)$   
**by** (*intro sum-mono*) (*simp-all add: Suc-le-eq*)  
**finally show** *?thesis* .  
**qed**  
  
**have**  $(\lambda N. \text{real } (\text{card } (A N)) - 6 / \text{pi}^2 * (\text{real } N)^2) \in$   
 $\Theta(\lambda N. 2 * (\text{sum-upto } (\lambda m. \text{real } (\text{totient } m)) (\text{real } N) - (3 / \text{pi}^2 * (\text{real } N)^2)) - 1)$   
**(is**  $- \in \Theta(?f)$ ) **using** \* \*\*  
**by** (*intro bigthetaI-cong eventually-mono [OF eventually-gt-at-top[of 0::nat]]*)  
*(auto simp: of-nat-diff sum-upto-altdef)*  
**also have**  $?f \in O(\lambda N. \text{real } N * \ln (\text{real } N))$   
**proof** (*rule landau-o.big.compose[OF - filterlim-real-sequentially]*, *rule sum-in-bigo*)  
**show**  $(\lambda x. 2 * (\text{sum-upto } (\lambda m. \text{real } (\text{totient } m)) x - 3 / \text{pi}^2 * x^2)) \in O(\lambda x.$   
 $x * \ln x)$   
**by** (*subst landau-o.big.cmult-in-iff*, *simp*, *rule summatory-totient-asymptotics*)  
**qed** *simp-all*  
**finally show** *?thesis* .  
**qed**  
  
**theorem** *coprime-pairs-asymptotics'*:  
 $(\lambda N. \text{real } (\text{card } (A N))) = o(\lambda N. 6 / \text{pi}^2 * (\text{real } N)^2) + o O(\lambda N. \text{real } N * \ln (\text{real } N))$   
**using** *coprime-pairs-asymptotics*  
**by** (*subst set-minus-plus [symmetric]*) (*simp-all add: fun-diff-def*)  
  
**theorem** *coprime-pairs-asymptotics''*:  
 $(\lambda N. \text{real } (\text{card } (A N))) \sim_{\text{[at-top]}} (\lambda N. 6 / \text{pi}^2 * (\text{real } N)^2)$   
**proof** –  
**have**  $(\lambda N. \text{real } (\text{card } (A N)) - 6 / \text{pi}^2 * (\text{real } N) \wedge 2) \in O(\lambda N. \text{real } N * \ln$

(*real N*)  
 by (*rule coprime-pairs-asymptotics*)  
 also have ( $\lambda N. \text{real } N * \ln (\text{real } N) \in o(\lambda N. 6 / \pi^2 * \text{real } N^2)$ )  
 by (*rule landau-o.small.compose[OF - filterlim-real-sequentially]*) *simp*  
 finally show ?thesis by (*simp add: asymp-equiv-altdef*)  
 qed

**theorem** *coprime-probability-tendsto*:

( $\lambda N. \text{card } (A \ N) / \text{card } (\{1..N\} \times \{1..N\}) \longrightarrow 6 / \pi^2$ )

**proof** –

have ( $\lambda N. 6 / \pi^2 \sim_{[at-top]} (\lambda N. 6 / \pi^2 * \text{real } N^2 / \text{real } N^2)$ )

using *eventually-gt-at-top[of 0::nat]*

by (*intro asymp-equiv-refl-ev*) (*auto elim!: eventually-mono*)

also have ...  $\sim_{[at-top]} (\lambda N. \text{real } (\text{card } (A \ N)) / \text{real } N^2)$

by (*intro asymp-equiv-intros asymp-equiv-symI[OF coprime-pairs-asymptotics']*)

also have ...  $\sim_{[at-top]} (\lambda N. \text{real } (\text{card } (A \ N)) / \text{real } (\text{card } (\{1..N\} \times \{1..N\})))$

by (*simp add: power2-eq-square*)

finally have ...  $\sim_{[at-top]} (\lambda. 6 / \pi^2)$  by (*simp add: asymp-equiv-sym*)

thus ?thesis by (*rule asymp-equivD-const*)

qed

end

### 13.6 The asymptotics of the number of Farey fractions

**definition** *farey-fractions* :: *nat*  $\Rightarrow$  *rat set* **where**

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

**lemma** *Fract-eq-coprime*:

**assumes** *Rat.Fract* *a* *b* = *Rat.Fract* *c* *d* *b* > 0 *d* > 0 *coprime* *a* *b* *coprime* *c* *d*

**shows** *a* = *c* *b* = *d*

**proof** –

**from** *assms* **have** *a* \* *d* = *c* \* *b* **by** (*auto simp: eq-rat*)

**hence** *abs* (*a* \* *d*) = *abs* (*c* \* *b*) **by** (*simp only:*)

**hence** *abs* *a* \* *abs* *d* = *abs* *c* \* *abs* *b* **by** (*simp only: abs-mult*)

**also have** ?*this*  $\longleftrightarrow$  *abs* *a* = *abs* *c*  $\wedge$  *d* = *b*

**using** *assms* **by** (*subst coprime-crossproduct-int*) *simp-all*

**finally show** *b* = *d* **by** *simp*

**with**  $\langle a * d = c * b \rangle$  **and**  $\langle b > 0 \rangle$  **show** *a* = *c* **by** *simp*

qed

**lemma** *quotient-of-split*:

*P* (*quotient-of* *q*) = ( $\forall$  *a* *b*. *b* > 0  $\longrightarrow$  *coprime* *a* *b*  $\longrightarrow$  *q* = *Rat.Fract* *a* *b*  $\longrightarrow$  *P* (*a*, *b*))

**by** (*cases* *q*) (*auto simp: quotient-of-Fract dest: Fract-eq-coprime*)

**lemma** *quotient-of-split-asm*:

*P* (*Rat.quotient-of* *q*) = ( $\neg(\exists$  *a* *b*. *b* > 0  $\wedge$  *coprime* *a* *b*  $\wedge$  *q* = *Rat.Fract* *a* *b*  $\wedge$   $\neg P$  (*a*, *b*))

using *quotient-of-split*[of  $P$   $q$ ] by *blast*

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

**proof** (*rule* *bij-betwI*[of - - -  $\lambda q$ . *case* *quotient-of*  $q$  of  $(a, b) \Rightarrow (nat\ a, nat\ b)$ ],  
*goal-cases*)  
**case** 1  
**show** *?case*  
**by** (*auto simp*: *farey-fractions-def* *Rat.zero-less-Fract-iff* *Rat.Fract-le-one-iff*  
*Rat.quotient-of-Fract* *Rat.normalize-def* *gcd-int-def* *Let-def*)

**next**  
**case** 2  
**show** *?case*  
**by** (*auto simp add*: *farey-fractions-def* *Rat.Fract-le-one-iff* *Rat.zero-less-Fract-iff*  
*split*: *prod.splits* *quotient-of-split-asm*)  
(*simp add*: *coprime-int-iff* [*symmetric*])

**next**  
**case** (3  $x$ )  
**thus** *?case* **by** (*auto simp*: *Rat.quotient-of-Fract* *Rat.normalize-def* *Let-def* *gcd-int-def*)

**next**  
**case** (4  $x$ )  
**thus** *?case* **unfolding** *farey-fractions-def*  
**by** (*split* *quotient-of-split*) (*auto simp*: *Rat.zero-less-Fract-iff*)

**qed**

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

**proof** –  
**have** *card* (*farey-fractions*  $N$ ) = *card*  $\{(a,b) \mid a\ b.\ 0 < a \wedge a \leq b \wedge b \leq N \wedge \text{coprime } a\ b\}$   
**using** *farey-fractions-bij* **by** (*rule* *bij-betw-same-card* [*symmetric*])  
**also have**  $\{(a,b) \mid a\ b.\ 0 < a \wedge a \leq b \wedge b \leq N \wedge \text{coprime } a\ b\} =$   
 $(\bigcup_{b \in \{0 <..N\}}. (\lambda a. (a, b)) \text{ ‘ totatives } b)$   
**by** (*auto simp*: *totatives-def* *image-iff*)  
**also have** *card* ... =  $(\sum_{b \in \{0 <..N\}}. \text{card } ((\lambda a. (a, b)) \text{ ‘ totatives } b))$   
**by** (*intro* *card-UN-disjoint*) *auto*  
**also have** ... =  $(\sum_{b \in \{0 <..N\}}. \text{totient } b)$   
**unfolding** *totient-def* **by** (*intro* *sum.cong* *refl* *card-image*) (*auto simp*: *inj-on-def*)  
**finally show** *?thesis* .

**qed**

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

**proof** –  
**have**  $(\lambda N. \text{sum-upto } (\lambda n. \text{real } (\text{totient } n)) (\text{real } N) - 3 / \text{pi}^2 * (\text{real } N)^2)$   
 $\in O(\lambda N. \text{real } N * \ln (\text{real } N))$  (**is** *?f*  $\in$  -)  
**using** *summatory-totient-asymptotics* *filterlim-real-sequentially*  
**by** (*rule* *landau-o.big.compose*)

**also have**  $?f = (\lambda N. \text{real} (\text{card} (\text{farey-fractions } N)) - 3 / \text{pi}^2 * (\text{real } N)^2)$   
**by** (*intro ext*) (*simp add: sum-upto-altdef card-farey-fractions*)  
**finally show** *?thesis* .  
**qed**

**theorem** *card-farey-fractions-asymptotics'*:  
 $(\lambda N. \text{card} (\text{farey-fractions } N)) = o (\lambda N. 3 / \text{pi}^2 * N^2) + o O(\lambda N. N * \ln N)$   
**using** *card-farey-fractions-asymptotics*  
**by** (*subst set-minus-plus [symmetric]*) (*simp-all add: fun-diff-def*)

**theorem** *card-farey-fractions-asymptotics''*:  
 $(\lambda N. \text{real} (\text{card} (\text{farey-fractions } N))) \sim[\text{at-top}] (\lambda N. 3 / \text{pi}^2 * (\text{real } N)^2)$   
**proof** –  
**have**  $(\lambda N. \text{real} (\text{card} (\text{farey-fractions } N)) - 3 / \text{pi}^2 * (\text{real } N)^2) \in O(\lambda N. \text{real } N * \ln (\text{real } N))$   
**by** (*rule card-farey-fractions-asymptotics*)  
**also have**  $(\lambda N. \text{real } N * \ln (\text{real } N)) \in o(\lambda N. 3 / \text{pi}^2 * \text{real } N^2)$   
**by** (*rule landau-o.small.compose[OF - filterlim-real-sequentially]*) *simp*  
**finally show** *?thesis* **by** (*simp add: asymp-equiv-altdef*)  
**qed**

**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**

*prime-factorization-nat' n =* (  
*let ps = prime-factorization-nat n*  
*in map* ( $\lambda p. (p, \text{count-list } ps \ p - 1)$ ) (*remdups-adj (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** (*intro equalityI subsetI; clarify*)

**fix**  $p \ k :: \text{nat}$

**assume**  $pk: (p, k) \in \text{set} (\text{prime-factorization-nat}' \ n)$

**hence**  $p: p \in \text{prime-factors } n$

**by** (*auto simp: prime-factorization-nat'-def Let-def multiset-prime-factorization-nat-correct*)

**hence**  $p'$ : prime  $p$  **by** (*simp add: prime-factors-multiplicity*)  
**from**  $pk\ p'$  **have**  $k = \text{multiplicity } p\ n - 1$   
**by** (*auto simp: prime-factorization-nat'-def Let-def multiset-prime-factorization-nat-correct*  
*count-prime-factorization-prime [symmetric] count-mset*)  
**with**  $p$  **show**  $(p, k) \in (\lambda p. (p, \text{multiplicity } p\ n - 1))$  ‘prime-factors  $n$  **by** *auto*  
**next**  
**fix**  $p :: \text{nat}$   
**assume**  $p \in \text{prime-factors } n$   
**moreover from this have** prime  $p$  **by** (*simp add: prime-factors-multiplicity*)  
**ultimately show**  $(p, \text{multiplicity } p\ n - 1) \in \text{set } (\text{prime-factorization-nat}'\ n)$   
**by** (*auto simp: prime-factorization-nat'-def Let-def multiset-prime-factorization-nat-correct*  
*count-prime-factorization-prime [symmetric] count-mset*)  
**qed**

**lemma** *distinct-prime-factorization-nat'* [*simp*]: *distinct (prime-factorization-nat'*  
 $n)$   
**by** (*simp add: distinct-map inj-on-def prime-factorization-nat'-def Let-def*)

**lemmas** (**in** *multiplicative-function'*) *efficient-code'* =  
*efficient-code [of  $\lambda-. \text{prime-factorization-nat}'\ n\ n$  for  $n$ ,*  
*OF set-prime-factorization-nat' distinct-prime-factorization-nat']*

## 14.1 Möbius $\mu$ function

**definition** *moebius-mu-aux* ::  $\text{nat} \Rightarrow (\text{unit} \Rightarrow \text{nat list}) \Rightarrow \text{int}$  **where**  
*moebius-mu-aux*  $n$  =  
*(if  $n \neq 0 \wedge \neg 4 \text{ dvd } n \wedge \neg 9 \text{ dvd } n$  then*  
*(let  $ps = \text{ps } ()$  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**  $\text{mset } ps = \text{prime-factorization } n$   
**shows**  $\text{moebius-mu } n = \text{of-int } (\text{moebius-mu-aux } n\ qs)$   
**proof** (*cases  $n = 0 \vee 4 \text{ dvd } n \vee 9 \text{ dvd } n$* )  
**case** *False*  
**hence** [*simp*]:  $n > 0$  **by** *auto*  
**have**  $\text{set-mset } (\text{mset } ps) = \text{prime-factors } n$  **by** (*subst assms*) *simp*  
**hence** [*simp*]:  $\text{set } ps = \text{prime-factors } n$  **by** *simp*  
**show** *?thesis*  
**proof** (*cases distinct ps*)  
**case** *True*  
**have**  $\text{multiplicity } p\ n = 1$  **if**  $p: p \in \text{prime-factors } n$  **for**  $p$   
**proof** –  
**from**  $p$  **and** *True* **have**  $\text{count } (\text{mset } ps)\ p = 1$  **by** (*auto simp: distinct-count-atmost-1*)  
**also from assms and**  $p$  **have**  $\text{count } (\text{mset } ps)\ p = \text{multiplicity } p\ n$   
**by** (*simp add: prime-factors-multiplicity count-prime-factorization-prime*)

```

    finally show multiplicity p n = 1 .
  qed
  moreover from True have card (prime-factors n) = length ps
    by (simp only: assms [symmetric] set-mset-mset distinct-card)
  ultimately show ?thesis using False and True
    by (auto simp add: moebius-mu-def moebius-mu-aux-def ps-def
        Let-def squarefree-factorial-semiring')
  next
  case False
  then obtain p where count (mset ps) p ≠ (if p ∈ set ps then 1 else 0)
    by (subst (asm) distinct-count-atmost-1) auto
  moreover from this have p: p ∈ prime-factors n
    by (cases count (mset ps) p = 0) (auto split: if-splits)
  ultimately have count (mset ps) p > 1 by (cases count (mset ps) p) auto
  with p and assms have multiplicity p n > 1
    by (simp add: prime-factors-multiplicity count-prime-factorization-prime)
  with False and assms and p have ¬squarefree n
    by (auto simp: squarefree-factorial-semiring')
  with False and assms and p show ?thesis
    by (auto simp: moebius-mu-def moebius-mu-aux-def)
  qed
  next
  case True
  with not-squarefreeI[of 2 n] and not-squarefreeI[of 3 n] show ?thesis
    by (auto simp: moebius-mu-aux-def)
  qed

```

**lemma** *moebius-mu-code* [code]:

```

  moebius-mu n = of-int (moebius-mu-aux n (λ-. prime-factorization-nat n))
  by (rule moebius-mu-conv-moebius-mu-aux) (simp-all add: multiset-prime-factorization-nat-correct)

```

**value** *moebius-mu 12578972695257* :: int

## 14.2 Euler's $\phi$ function

```

primrec totient-aux1 :: nat ⇒ nat list ⇒ nat where
  totient-aux1 n [] = n
| totient-aux1 n (p # ps) = totient-aux1 (n - n div p) ps

```

**lemma** *of-nat-totient-aux1*:

```

  assumes  $\bigwedge p. p \in \text{set } ps \implies \text{prime } p \wedge p. p \in \text{set } ps \implies p \text{ dvd } n \text{ distinct } ps$ 
  shows  $\text{real } (\text{totient-aux1 } n \text{ ps}) = \text{real } n * (\prod_{p \in \text{set } ps}. 1 - 1 / \text{real } p)$ 
using assms
proof (induction ps arbitrary: n)
  case (Cons p ps n)
  from Cons.prem1 have p: prime p p dvd n by auto
  have  $\text{real } (\text{totient-aux1 } n \text{ (p \# ps)}) = \text{real } (\text{totient-aux1 } (n - n \text{ div } p) \text{ ps})$  by
simp
  also have  $\dots = \text{real } (n - n \text{ div } p) * (\prod_{p \in \text{set } ps}. 1 - 1 / \text{real } p)$ 

```

```

proof (rule Cons.IH)
  fix q assume q: q ∈ set ps
  define m where m = n div p
  from p have m: n = p * m by (simp add: m-def)
  from Cons.prem1 q have prime q q dvd n p ≠ q by auto
  hence q dvd m using primes-dvd-imp-eq[of q p] p by (auto simp add: m
prime-dvd-mult-iff)
  thus q dvd n - n div p unfolding m-def using p <q dvd n> by simp
  qed (insert Cons.prem1, auto)
  also have real (n - n div p) = real n * (1 - 1 / real p)
  by (simp add: of-nat-diff real-of-nat-div p field-simps)
  also have ... * (∏ p∈set ps. 1 - 1 / real p) = real n * (∏ p∈set (p#ps). 1 -
1 / real p)
  using Cons.prem1 by simp
  finally show ?case .
qed simp-all

```

```

lemma totient-conv-totient-aux1:
  assumes set ps = prime-factors n distinct ps
  shows totient n = totient-aux1 n ps
proof -
  from assms have real (totient-aux1 n ps) = real n * (∏ p∈set ps. 1 - 1 / real
p)
  by (intro of-nat-totient-aux1) auto
  also have set ps = prime-factors n by fact
  also have real n * (∏ p∈prime-factors n. 1 - 1 / real p) = real (totient n)
  by (rule totient-formula2 [symmetric])
  finally show ?thesis by (simp only: of-nat-eq-iff)
qed

```

```

definition prime-factors-nat :: nat ⇒ nat list where
  prime-factors-nat n = remdups-adj (sort (prime-factorization-nat n))

```

```

lemma set-prime-factors-nat [simp]: set (prime-factors-nat n) = prime-factors n
unfolding prime-factors-nat-def multiset-prime-factorization-nat-correct by simp

```

```

lemma distinct-prime-factors-nat [simp]: distinct (prime-factors-nat n)
by (simp add: prime-factors-nat-def)

```

```

definition totient-aux2 :: (nat × nat) list ⇒ nat where
  totient-aux2 xs = (∏ (p,k)←xs. p ^ k * (p - 1))

```

```

lemma totient-conv-totient-aux2:
  assumes n ≠ 0
  assumes set xs = (λp. (p, multiplicity p n - 1)) ` prime-factors n
  assumes distinct xs
  shows totient n = totient-aux2 xs
proof -

```

```

have totient-aux2 xs = (∏ (p,k)←xs. p ^ k * (p - 1)) by (fact totient-aux2-def)
also from assms have ... =
  (∏ x∈(λp. (p, multiplicity p n - 1)) ' prime-factors n. case x of (p, k) ⇒ p ^
k * (p - Suc 0))
  by (subst prod.distinct-set-conv-list [symmetric]) simp-all
also have ... = (∏ p∈prime-factors n. p ^ (multiplicity p n - 1) * (p - Suc
0))
  by (subst prod.reindex) (auto simp: inj-on-def)
also have ... = (∏ p∈prime-factors n. p ^ multiplicity p n - p ^ (multiplicity
p n - 1))
  by (intro prod.cong refl) (auto simp: prime-factors-multiplicity algebra-simps
power-Suc [symmetric] simp del: power-Suc)
also have ... = totient n using assms(1) by (subst totient.prod-prime-factors')
auto
finally show ?thesis ..
qed

```

```

lemma totient-code1: totient n = totient-aux1 n (prime-factors-nat n)
by (intro totient-conv-totient-aux1) simp-all

```

```

lemma totient-code2: totient n = (if n = 0 then 0 else totient-aux2 (prime-factorization-nat'
n))
by (simp-all add: set-prime-factorization-nat' totient-conv-totient-aux2 split: if-splits)

```

```

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 $\lambda$ function

```

lemma [code]: liouville-lambda n =

```

```

  (if n = 0 then 0 else if even (length (prime-factorization-nat n)) then 1 else -1)

```

```

  by (auto simp: liouville-lambda-def multiset-prime-factorization-nat-correct)

```

```

value liouville-lambda 1264785343674 :: int

```

```

end

```

## References

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