Elementary Facts About the Distribution of Primes

Manuel Eberl

October 13, 2025

Abstract

This entry is a formalisation of Chapter 4 (and parts of Chapter 3) of Apostol's *Introduction to Analytic Number Theory*. The main topics that are addressed are properties of the distribution of prime numbers that can be shown in an elementary way (i. e. without the Prime Number Theorem), the various equivalent forms of the PNT (which imply each other in elementary ways), and consequences that follow from the PNT in elementary ways. The latter include bounds for the number of distinct prime factors of n, the divisor function $d(n)$, Euler's totient function $\varphi(n)$, and $\text{lcm}(1, \ldots, n)$.

Contents

1 Auxiliary material

theory *Prime-Distribution-Elementary-Library*
imports
Zeta-Function.Zeta-Function
Prime-Number-Theorem.Prime-Counting-Functions
Stirling-Formula.Stirling-Formula
begin

lemma *divisor-count-pos* [*intro*]: $n > 0 \Longrightarrow$ *divisor-count* $n > 0$
by (*auto simp*: *divisor-count-def intro*!: *Nat.gr0I*)

lemma *divisor-count-eq-0-iff* [*simp*]: *divisor-count* $n = 0 \longleftrightarrow n = 0$
by (*cases* $n = 0$) *auto*

lemma *divisor-count-pos-iff* [*simp*]: *divisor-count* $n > 0 \longleftrightarrow n > 0$
by (*cases* $n = 0$) *auto*

lemma *smallest-prime-beyond-eval*:
prime $n \Longrightarrow$ *smallest-prime-beyond* $n = n$
$\neg$*prime* $n \Longrightarrow$ *smallest-prime-beyond* $n =$ *smallest-prime-beyond* (*Suc* n)
proof −
assume *prime* n
thus *smallest-prime-beyond* $n = n$
by (*rule smallest-prime-beyond-eq*) *auto*
next
assume $\neg$*prime* n
show *smallest-prime-beyond* $n =$ *smallest-prime-beyond* (*Suc* n)
proof (*rule antisym*)
show *smallest-prime-beyond* $n \leq$ *smallest-prime-beyond* (*Suc* n)
by (*rule smallest-prime-beyond-smallest*)
(*auto intro*: *order.trans*[*OF - smallest-prime-beyond-le*])
next
have *smallest-prime-beyond* $n \neq n$
using *prime-smallest-prime-beyond*[*of n*] ‹$\neg$*prime* n› **by** *metis*
hence *smallest-prime-beyond* $n > n$
using *smallest-prime-beyond-le*[*of n*] **by** *linarith*
thus *smallest-prime-beyond* $n \geq$ *smallest-prime-beyond* (*Suc* n)
by (*intro smallest-prime-beyond-smallest*) *auto*
qed
qed

lemma *nth-prime-numeral*:
nth-prime (*numeral* n) = *smallest-prime-beyond* (*Suc* (*nth-prime* (*pred-numeral* n)))
by (*subst nth-prime-Suc*[*symmetric*]) *auto*

lemmas *nth-prime-eval* = *smallest-prime-beyond-eval nth-prime-Suc nth-prime-numeral*

```
lemma nth-prime-1 [simp]: nth-prime (Suc 0) = 3
  by (simp add: nth-prime-eval)

lemma nth-prime-2 [simp]: nth-prime 2 = 5
  by (simp add: nth-prime-eval)

lemma nth-prime-3 [simp]: nth-prime 3 = 7
  by (simp add: nth-prime-eval)


lemma strict-mono-sequence-partition:
  assumes strict-mono (f :: nat ⇒ 'a :: {linorder, no-top})
  assumes x ≥ f 0
  assumes filterlim f at-top at-top
  shows   ∃k. x ∈ {f k..<f (Suc k)}
proof −
  define k where k = (LEAST k. f (Suc k) > x)
  {
    obtain n where x ≤ f n
      using assms by (auto simp: filterlim-at-top eventually-at-top-linorder)
    also have f n < f (Suc n)
      using assms by (auto simp: strict-mono-Suc-iff)
    finally have ∃n. f (Suc n) > x by auto
  }
  from LeastI-ex[OF this] have x < f (Suc k)
    by (simp add: k-def)
  moreover have f k ≤ x
  proof (cases k)
    case (Suc k')
    have k ≤ k' if f (Suc k') > x
      using that unfolding k-def by (rule Least-le)
    with Suc show f k ≤ x by (cases f k ≤ x) (auto simp: not-le)
  qed (use assms in auto)
  ultimately show ?thesis by auto
qed

lemma nth-prime-partition:
  assumes x ≥ 2
  shows   ∃k. x ∈ {nth-prime k..<nth-prime (Suc k)}
 using strict-mono-sequence-partition[OF strict-mono-nth-prime, of x] assms nth-prime-at-top
  by simp

lemma nth-prime-partition':
  assumes x ≥ 2
  shows   ∃k. x ∈ {real (nth-prime k)..<real (nth-prime (Suc k))}
  by (rule strict-mono-sequence-partition)
     (auto simp: strict-mono-Suc-iff assms
           intro!: filterlim-real-sequentially filterlim-compose[OF - nth-prime-at-top])
```

lemma *between-nth-primes-imp-nonprime*:
assumes $n > \textit{nth-prime}\ k\ n < \textit{nth-prime}\ (\textit{Suc}\ k)$
shows $\neg \textit{prime}\ n$
using *assms* **by** (*metis Suc-leI not-le nth-prime-Suc smallest-prime-beyond-smallest*)

lemma *nth-prime-partition''*:
includes *prime-counting-syntax*
assumes $x \geq (2 :: \textit{real})$
shows $x \in \{\textit{real}\ (\textit{nth-prime}\ (\textit{nat}\ \lfloor \pi\ x \rfloor - 1))..<\textit{real}\ (\textit{nth-prime}\ (\textit{nat}\ \lfloor \pi\ x \rfloor))\}$
proof −
obtain n **where** n: $x \in \{\textit{nth-prime}\ n..<\textit{nth-prime}\ (\textit{Suc}\ n)\}$
using *nth-prime-partition' assms* **by** *auto*
have $\pi\ (\textit{nth-prime}\ n) = \pi\ x$
unfolding *π-def* **using** *between-nth-primes-imp-nonprime n*
by (*intro prime-sum-upto-eqI*) (*auto simp*: *le-nat-iff le-floor-iff*)
hence $\textit{real}\ n = \pi\ x - 1$
by *simp*
hence *n-eq*: $n = \textit{nat}\ \lfloor \pi\ x \rfloor - 1\ \textit{Suc}\ n = \textit{nat}\ \lfloor \pi\ x \rfloor$
by *linarith+*
with n **show** *?thesis*
by *simp*
qed

lemma *asymp-equivD-strong*:
assumes $f \sim[F]\ g$ *eventually* $(\lambda x.\ f\ x \neq 0 \vee g\ x \neq 0)\ F$
shows $((\lambda x.\ f\ x\ /\ g\ x) \longrightarrow 1)\ F$
proof −
from *assms*(*1*) **have** $((\lambda x.\ \textit{if}\ f\ x = 0 \wedge g\ x = 0\ \textit{then}\ 1\ \textit{else}\ f\ x\ /\ g\ x) \longrightarrow 1)\ F$
by (*rule asymp-equivD*)
also have *?this* $\longleftrightarrow$ *?thesis*
by (*intro filterlim-cong eventually-mono*[*OF assms*(*2*)]) *auto*
finally show *?thesis* **.**
qed

lemma *hurwitz-zeta-shift*:
fixes $s :: \textit{complex}$
assumes $a > 0$ **and** $s \neq 1$
shows *hurwitz-zeta* $(a + \textit{real}\ n)\ s = \textit{hurwitz-zeta}\ a\ s - (\sum k<n.\ (a + \textit{real}\ k)\ \textit{powr}\ -s)$
proof (*rule analytic-continuation-open*[**where** $f = \lambda s.\ \textit{hurwitz-zeta}\ (a + \textit{real}\ n)\ s$])
fix s **assume** s: $s \in \{s.\ \textit{Re}\ s > 1\}$
have $(\lambda k.\ (a + \textit{of-nat}\ (k + n))\ \textit{powr}\ -s)\ \textit{sums}\ \textit{hurwitz-zeta}\ (a + \textit{real}\ n)\ s$
using *sums-hurwitz-zeta*[*of a + real n s*] *s assms* **by** (*simp add*: *add-ac*)
moreover have $(\lambda k.\ (a + \textit{of-nat}\ k)\ \textit{powr}\ -s)\ \textit{sums}\ \textit{hurwitz-zeta}\ a\ s$
using *sums-hurwitz-zeta*[*of a s*] *s assms* **by** (*simp add*: *add-ac*)
hence $(\lambda k.\ (a + \textit{of-nat}\ (k + n))\ \textit{powr}\ -s)\ \textit{sums}$

(*hurwitz-zeta a s* − (∑ *k*<*n*. (*a* + *of-nat k*) *powr* −*s*))
by (*rule sums-split-initial-segment*)
ultimately show *hurwitz-zeta* (*a* + *real n*) *s* = *hurwitz-zeta a s* − (∑ *k*<*n*. (*a* + *real k*) *powr* −*s*)
by (*simp add*: *sums-iff*)
next
show *connected* (−{*1*::*complex*})
by (*rule connected-punctured-universe*) *auto*
qed (*use assms* **in** ‹*auto intro*!: *holomorphic-intros open-halfspace-Re-gt exI*[*of* - *2*]›)

lemma *pbernpoly-bigo*: *pbernpoly n* ∈ *O*(λ-. *1*)
proof −
from *bounded-pbernpoly*[*of n*] **obtain** *c* **where** ⋀*x*. *norm* (*pbernpoly n x*) ≤ *c*
by *auto*
thus *?thesis* **by** (*intro bigoI*[*of* - *c*]) *auto*
qed

lemma *harm-le*: *n* ≥ *1* ⟹ *harm n* ≤ *ln n* + *1*
using *euler-mascheroni-sequence-decreasing*[*of 1 n*]
by (*simp add*: *harm-expand*)

lemma *sum-upto-1* [*simp*]: *sum-upto f 1* = *f 1*
proof −
have {*0*<..*Suc 0*} = {*1*} **by** *auto*
thus *?thesis* **by** (*simp add*: *sum-upto-altdef*)
qed

lemma *sum-upto-cong'* [*cong*]:
(⋀*n*. *n* > *0* ⟹ *real n* ≤ *x* ⟹ *f n* = *f' n*) ⟹ *x* = *x'* ⟹ *sum-upto f x* = *sum-upto f' x'*
unfolding *sum-upto-def* **by** (*intro sum.cong*) *auto*

lemma *finite-primes-le*: *finite* {*p*. *prime p* ∧ *real p* ≤ *x*}
by (*rule finite-subset*[*of* - {..*nat* ⌊*x*⌋}]) (*auto simp*: *le-nat-iff le-floor-iff*)

lemma *frequently-filtermap*: *frequently P* (*filtermap f F*) = *frequently* (λ*n*. *P* (*f n*)) *F*
by (*auto simp*: *frequently-def eventually-filtermap*)

lemma *frequently-mono-filter*: *frequently P F* ⟹ *F* ≤ *F'* ⟹ *frequently P F'*
using *filter-leD*[*of F F' λx*. ¬*P x*] **by** (*auto simp*: *frequently-def*)

lemma *π-at-top*: *filterlim primes-pi at-top at-top*
unfolding *filterlim-at-top*
proof *safe*
fix *C* :: *real*
define *x0* **where** *x0* = *real* (*nth-prime* (*nat* ⌈*max 0 C*⌉))

```
  show eventually (λx. primes-pi x ≥ C) at-top
    using eventually-ge-at-top
  proof eventually-elim
    fix x assume x ≥ x0
    have C ≤ real (nat ⌈max 0 C⌉ + 1) by linarith
    also have real (nat ⌈max 0 C⌉ + 1) = primes-pi x0
      unfolding x0-def by simp
    also have ... ≤ primes-pi x by (rule π-mono) fact
    finally show primes-pi x ≥ C .
  qed
qed

lemma sum-upto-ln-stirling-weak-bigo: (λx. sum-upto ln x − x * ln x + x) ∈ O(ln)
proof −
  let ?f = λx. x * ln x − x + ln (2 * pi * x) / 2
  have ln (fact n) − (n * ln n − n + ln (2 * pi * n) / 2) ∈ {0..1/(12*n)} if n
> 0 for n :: nat
    using ln-fact-bounds[OF that] by (auto simp: algebra-simps)
  hence (λn. ln (fact n) − ?f n) ∈ O(λn. 1 / real n)
    by (intro bigoI[of - 1/12] eventually-mono[OF eventually-gt-at-top[of 0]]) auto
  hence (λx. ln (fact (nat ⌊x⌋)) − ?f (nat ⌊x⌋)) ∈ O(λx. 1 / real (nat ⌊x⌋))
    by (rule landau-o.big.compose)
     (intro filterlim-compose[OF filterlim-nat-sequentially] filterlim-floor-sequentially)
  also have (λx. 1 / real (nat ⌊x⌋)) ∈ O(λx::real. ln x) by real-asymp
  finally have (λx. ln (fact (nat ⌊x⌋)) − ?f (nat ⌊x⌋) + (?f (nat ⌊x⌋) − ?f x)) ∈
O(λx. ln x)
    by (rule sum-in-bigo) real-asymp
  hence (λx. ln (fact (nat ⌊x⌋)) − ?f x) ∈ O(λx. ln x)
    by (simp add: algebra-simps)
  hence (λx. ln (fact (nat ⌊x⌋)) − ?f x + ln (2 * pi * x) / 2) ∈ O(λx. ln x)
    by (rule sum-in-bigo) real-asymp
  thus ?thesis by (simp add: sum-upto-ln-conv-ln-fact algebra-simps)
qed
```

1.1 Various facts about Dirichlet series

```
lemma fds-mangoldt':
  fds mangoldt = fds-zeta * fds-deriv (fds moebius-mu)
proof −
  have fds mangoldt = (fds moebius-mu * fds (λn. of-real (ln (real n)) :: 'a))
    (is - = ?f) by (subst fds-mangoldt) auto
  also have ... = fds-zeta * fds-deriv (fds moebius-mu)
  proof (intro fds-eqI)
    fix n :: nat assume n: n > 0
    have fds-nth ?f n = (∑ d | d dvd n. moebius-mu d * of-real (ln (real (n div
d))))
      by (auto simp: fds-eq-iff fds-nth-mult dirichlet-prod-def)
    also have ... = (∑ d | d dvd n. moebius-mu d * of-real (ln (real n / real d)))
      by (intro sum.cong) (auto elim!: dvdE simp: ln-mult split: if-splits)
```

```
    also have ... = (∑ d | d dvd n. moebius-mu d * of-real (ln n - ln d))
      using n by (intro sum.cong refl) (subst ln-div, auto elim!: dvdE)
    also have ... = of-real (ln n) * (∑ d | d dvd n. moebius-mu d) -
                    (∑ d | d dvd n. of-real (ln d) * moebius-mu d)
      by (simp add: sum-subtractf sum-distrib-left sum-distrib-right algebra-simps)
    also have of-real (ln n) * (∑ d | d dvd n. moebius-mu d) = 0
      by (subst sum-moebius-mu-divisors') auto
    finally show fds-nth ?f n = fds-nth (fds-zeta * fds-deriv (fds moebius-mu) :: 'a
fds) n
      by (simp add: fds-nth-mult dirichlet-prod-altdef1 fds-nth-deriv sum-negf scaleR-conv-of-real)
  qed
  finally show ?thesis .
qed

lemma sum-upto-divisor-sum1:
  sum-upto (λn. ∑ d | d dvd n. f d :: real) x = sum-upto (λn. f n * floor (x / n))
x
proof -
  have sum-upto (λn. ∑ d | d dvd n. f d :: real) x =
        sum-upto (λn. f n * real (nat (floor (x / n)))) x
    using sum-upto-dirichlet-prod[of f λ-. 1 x]
    by (simp add: dirichlet-prod-def sum-upto-altdef)
  also have ... = sum-upto (λn. f n * floor (x / n)) x
    unfolding sum-upto-def by (intro sum.cong) auto
  finally show ?thesis .
qed

lemma sum-upto-divisor-sum2:
  sum-upto (λn. ∑ d | d dvd n. f d :: real) x = sum-upto (λn. sum-upto f (x / n))
x
  using sum-upto-dirichlet-prod[of λ-. 1 f x] by (simp add: dirichlet-prod-altdef1)

lemma sum-upto-moebius-times-floor-linear:
  sum-upto (λn. moebius-mu n * ⌊x / real n⌋) x = (if x ≥ 1 then 1 else 0)
proof -
  have real-of-int (sum-upto (λn. moebius-mu n * ⌊x / real n⌋) x) =
        sum-upto (λn. moebius-mu n * of-int ⌊x / real n⌋) x
    by (simp add: sum-upto-def)
  also have ... = sum-upto (λn. ∑ d | d dvd n. moebius-mu d :: real) x
    using sum-upto-divisor-sum1[of moebius-mu x] by auto
  also have ... = sum-upto (λn. if n = 1 then 1 else 0) x
    by (intro sum-upto-cong sum-moebius-mu-divisors' refl)
  also have ... = real-of-int (if x ≥ 1 then 1 else 0)
    by (auto simp: sum-upto-def)
  finally show ?thesis unfolding of-int-eq-iff .
qed

lemma ln-fact-conv-sum-mangoldt:
  sum-upto (λn. mangoldt n * ⌊x / real n⌋) x = ln (fact (nat ⌊x⌋))
```

proof −
  have *sum-upto* (λ*n*. *mangoldt n* ∗ *of-int* ⌊*x* / *real n*⌋) *x* =
        sum-upto (λ*n*. ∑ *d* | *d dvd n*. *mangoldt d* :: *real*) *x*
    using *sum-upto-divisor-sum1*[*of mangoldt x*] **by** *auto*
  also have ... = *sum-upto* (λ*n*. *of-real* (*ln* (*real n*))) *x*
    by (*intro sum-upto-cong mangoldt-sum refl*) *auto*
  also have ... = (∑ *n*∈{*0*<..*nat* ⌊*x*⌋}. *ln n*)
    by (*simp add*: *sum-upto-altdef*)
  also have ... = *ln* (∏ {*0*<..*nat* ⌊*x*⌋})
    unfolding *of-nat-prod* **by** (*subst ln-prod*) *auto*
  also have {*0*<..*nat* ⌊*x*⌋} = {*1*..*nat* ⌊*x*⌋} **by** *auto*
  also have ∏ ... = *fact* (*nat* ⌊*x*⌋)
    by (*simp add*: *fact-prod*)
  finally show *?thesis* **by** *simp*
qed

1.2 Facts about prime-counting functions

lemma *abs-π* [*simp*]: |*primes-pi x*| = *primes-pi x*
  by (*subst abs-of-nonneg*) *auto*

lemma *π-less-self*:
  includes *prime-counting-syntax*
  assumes *x* > *0*
  shows *π x* < *x*
proof −
  have *π x* ≤ (∑ *n*∈{*1*<..*nat* ⌊*x*⌋}. *1*)
    unfolding *π-def prime-sum-upto-altdef2* **by** (*intro sum-mono2*) (*auto dest*: *prime-gt-1-nat*)
  also have ... = *real* (*nat* ⌊*x*⌋ − *1*)
    using *assms* **by** *simp*
  also have ... < *x* **using** *assms* **by** *linarith*
  finally show *?thesis* **.**
qed

lemma *π-le-self'*:
  includes *prime-counting-syntax*
  assumes *x* ≥ *1*
  shows *π x* ≤ *x* − *1*
proof −
  have *π x* ≤ (∑ *n*∈{*1*<..*nat* ⌊*x*⌋}. *1*)
    unfolding *π-def prime-sum-upto-altdef2* **by** (*intro sum-mono2*) (*auto dest*: *prime-gt-1-nat*)
  also have ... = *real* (*nat* ⌊*x*⌋ − *1*)
    using *assms* **by** *simp*
  also have ... ≤ *x* − *1* **using** *assms* **by** *linarith*
  finally show *?thesis* **.**
qed

lemma *π-le-self*:
includes *prime-counting-syntax*
assumes $x \geq 0$
shows $\pi\ x \leq x$
using *π-less-self*[*of x*] *assms* **by** (*cases* $x = 0$) *auto*

1.3 Strengthening 'Big-O' bounds

The following two statements are crucial: They allow us to strengthen a 'Big-O' statement for $n \to \infty$ or $x \to \infty$ to a bound for *all* $n \geq n_0$ or all $x \geq x_0$ under some mild conditions.

This allows us to use all the machinery of asymptotics in Isabelle and still get a bound that is applicable over the full domain of the function in the end. This is important because Newman often shows that $f(x) \in O(g(x))$ and then writes

$$\sum_{n \leq x} f(\frac{x}{n}) = \sum_{n \leq x} O(g(\frac{x}{n}))$$

which is not easy to justify otherwise.

lemma *natfun-bigoE*:
fixes $f :: nat \Rightarrow$ -
assumes *bigo*: $f \in O(g)$ **and** *nz*: $\bigwedge n.\ n \geq n0 \Longrightarrow g\ n \neq 0$
obtains c **where** $c > 0\ \bigwedge n.\ n \geq n0 \Longrightarrow norm\ (f\ n) \leq c * norm\ (g\ n)$
proof −
from *bigo* **obtain** c **where** c: $c > 0$ *eventually* $(\lambda n.\ norm\ (f\ n) \leq c * norm\ (g\ n))$ *at-top*
by (*auto elim*: *landau-o.bigE*)
then obtain $n0'$ **where** $n0'$: $\bigwedge n.\ n \geq n0' \Longrightarrow norm\ (f\ n) \leq c * norm\ (g\ n)$
by (*auto simp*: *eventually-at-top-linorder*)
define c' **where** $c' = Max\ ((\lambda n.\ norm\ (f\ n)\ /\ norm\ (g\ n))\ `\ (insert\ n0\ \{n0..<n0'\}))$
have $norm\ (f\ n) \leq max\ 1\ (max\ c\ c') * norm\ (g\ n)$ **if** $n \geq n0$ **for** n
proof (*cases* $n \geq n0'$)
case *False*
with *that* **have** $norm\ (f\ n)\ /\ norm\ (g\ n) \leq c'$
unfolding *c'-def* **by** (*intro Max.coboundedI*) *auto*
also have $\ldots \leq max\ 1\ (max\ c\ c')$ **by** *simp*
finally show *?thesis* **using** *nz*[*of n*] *that* **by** (*simp add*: *field-simps*)
next
case *True*
hence $norm\ (f\ n) \leq c * norm\ (g\ n)$ **by** (*rule n0'*)
also have $\ldots \leq max\ 1\ (max\ c\ c') * norm\ (g\ n)$
by (*intro mult-right-mono*) *auto*
finally show *?thesis* **.**
qed
with *that*[*of max 1* (*max c c'*)] **show** *?thesis* **by** *auto*
qed

lemma *bigoE-bounded-real-fun*:

```
  fixes f g :: real ⇒ real
  assumes f ∈ O(g)
  assumes ⋀x. x ≥ x0 ⟹ |g x| ≥ cg cg > 0
  assumes ⋀b. b ≥ x0 ⟹ bounded (f ‘ {x0..b})
  shows   ∃ c>0. ∀ x≥x0. |f x| ≤ c * |g x|
proof −
  from assms(1) obtain c where c: c > 0 eventually (λx. |f x| ≤ c * |g x|) at-top
    by (elim landau-o.bigE) auto
  then obtain b where b: ⋀x. x ≥ b ⟹ |f x| ≤ c * |g x|
    by (auto simp: eventually-at-top-linorder)
  have bounded (f ‘ {x0..max x0 b}) by (intro assms) auto
  then obtain C where C: ⋀x. x ∈ {x0..max x0 b} ⟹ |f x| ≤ C
    unfolding bounded-iff by fastforce

  define c' where c' = max c (C / cg)
  have |f x| ≤ c' * |g x| if x ≥ x0 for x
  proof (cases x ≥ b)
    case False
    then have |f x| ≤ C
      using C that by auto
    with False have |f x| / |g x| ≤ C / cg
      by (meson abs-ge-zero assms frac-le landau-omega.R-trans that)
    also have ... ≤ c' by (simp add: c'-def)
    finally show |f x| ≤ c' * |g x|
      using that False assms(2)[of x] assms(3) by (auto simp add: divide-simps
split: if-splits)
  next
    case True
    hence |f x| ≤ c * |g x| by (intro b) auto
    also have ... ≤ c' * |g x| by (intro mult-right-mono) (auto simp: c'-def)
    finally show ?thesis .
  qed
  moreover from c(1) have c' > 0 by (auto simp: c'-def)
  ultimately show ?thesis by blast
qed

lemma sum-upto-asymptotics-lift-nat-real-aux:
  fixes f :: nat ⇒ real and g :: real ⇒ real
  assumes bigo: (λn. (∑ k=1..n. f k) − g (real n)) ∈ O(λn. h (real n))
  assumes g-bigo-self: (λn. g (real n) − g (real (Suc n))) ∈ O(λn. h (real n))
  assumes h-bigo-self: (λn. h (real n)) ∈ O(λn. h (real (Suc n)))
  assumes h-pos: ⋀x. x ≥ 1 ⟹ h x > 0
  assumes mono-g: mono-on {1..} g ∨ mono-on {1..} (λx. − g x)
  assumes mono-h: mono-on {1..} h ∨ mono-on {1..} (λx. − h x)
  shows   ∃ c>0. ∀ x≥1. sum-upto f x − g x ≤ c * h x
proof −
  have h-nz: h (real n) ≠ 0 if n ≥ 1 for n
    using h-pos[of n] that by simp
```

```
  from natfun-bigoE[OF bigo h-nz] obtain c1 where
    c1: c1 > 0 ⋀n. n ≥ 1 ⟹ norm ((∑ k=1..n. f k) − g (real n)) ≤ c1 * norm
(h (real n))
    by auto
  from natfun-bigoE[OF g-bigo-self h-nz] obtain c2 where
    c2: c2 > 0 ⋀n. n ≥ 1 ⟹ norm (g (real n) − g (real (Suc n))) ≤ c2 * norm
(h (real n))
    by auto
  from natfun-bigoE[OF h-bigo-self h-nz] obtain c3 where
    c3: c3 > 0 ⋀n. n ≥ 1 ⟹ norm (h (real n)) ≤ c3 * norm (h (real (Suc n)))
    by auto

  {
    fix x :: real assume x: x ≥ 1
    define n where n = nat ⌊x⌋
    from x have n: n ≥ 1 unfolding n-def by linarith

    have (∑ k = 1..n. f k) − g x ≤ (c1 + c2) * h (real n) using mono-g
    proof
      assume mono: mono-on {1..} (λx. −g x)
      from x have x ≤ real (Suc n)
        unfolding n-def by linarith
      hence (∑ k=1..n. f k) − g x ≤ (∑ k=1..n. f k) − g n + (g n − g (Suc n))
        using mono-onD[OF mono, of x real (Suc n)] x by auto
      also have ... ≤ norm ((∑ k=1..n. f k) − g n) + norm (g n − g (Suc n))
        by simp
      also have ... ≤ c1 * norm (h n) + c2 * norm (h n)
        using n by (intro add-mono c1 c2) auto
      also have ... = (c1 + c2) * h n
        using h-pos[of real n] n by (simp add: algebra-simps)
      finally show ?thesis .
    next
      assume mono: mono-on {1..} g
      have (∑ k=1..n. f k) − g x ≤ (∑ k=1..n. f k) − g n
        using x by (intro diff-mono mono-onD[OF mono]) (auto simp: n-def)
      also have ... ≤ c1 * h (real n)
        using c1(2)[of n] n h-pos[of n] by simp
      also have ... ≤ (c1 + c2) * h (real n)
        using c2 h-pos[of n] n by (intro mult-right-mono) auto
      finally show ?thesis .
    qed
    also have (c1 + c2) * h (real n) ≤ (c1 + c2) * (1 + c3) * h x
      using mono-h
    proof
      assume mono: mono-on {1..} (λx. −h x)
      have (c1 + c2) * h (real n) ≤ (c1 + c2) * (c3 * h (real (Suc n)))
        using c3(2)[of n] n h-pos[of n] h-pos[of Suc n] c1(1) c2(1)
        by (intro mult-left-mono) (auto)
      also have ... = (c1 + c2) * c3 * h (real (Suc n))
```

```
    by (simp add: mult-ac)
  also have ... ≤ (c1 + c2) * (1 + c3) * h (real (Suc n))
  using c1(1) c2(1) c3(1) h-pos[of Suc n] by (intro mult-left-mono mult-right-mono)
auto
  also from x have x ≤ real (Suc n)
    unfolding n-def by linarith
  hence (c1 + c2) * (1 + c3) * h (real (Suc n)) ≤ (c1 + c2) * (1 + c3) * h x
    using c1(1) c2(1) c3(1) mono-onD[OF mono, of x real (Suc n)] x
    by (intro mult-left-mono) (auto simp: n-def)
  finally show (c1 + c2) * h (real n) ≤ (c1 + c2) * (1 + c3) * h x .
 next
  assume mono: mono-on {1..} h
  have (c1 + c2) * h (real n) = 1 * ((c1 + c2) * h (real n)) by simp
  also have ... ≤ (1 + c3) * ((c1 + c2) * h (real n))
    using c1(1) c2(1) c3(1) h-pos[of n] x n by (intro mult-right-mono) auto
  also have ... = (1 + c3) * (c1 + c2) * h (real n)
    by (simp add: mult-ac)
  also have ... ≤ (1 + c3) * (c1 + c2) * h x
    using x c1(1) c2(1) c3(1) h-pos[of n] n
    by (intro mult-left-mono mono-onD[OF mono]) (auto simp: n-def)
  finally show (c1 + c2) * h (real n) ≤ (c1 + c2) * (1 + c3) * h x
    by (simp add: mult-ac)
 qed
 also have (∑ k = 1..n. f k) = sum-upto f x
   unfolding sum-upto-altdef n-def by (intro sum.cong) auto
 finally have sum-upto f x − g x ≤ (c1 + c2) * (1 + c3) * h x .
}
moreover have (c1 + c2) * (1 + c3) > 0
  using c1(1) c2(1) c3(1) by (intro mult-pos-pos add-pos-pos) auto
ultimately show ?thesis by blast
qed

lemma sum-upto-asymptotics-lift-nat-real:
  fixes f :: nat ⇒ real and g :: real ⇒ real
  assumes bigo: (λn. (∑ k=1..n. f k) − g (real n)) ∈ O(λn. h (real n))
  assumes g-bigo-self: (λn. g (real n) − g (real (Suc n))) ∈ O(λn. h (real n))
  assumes h-bigo-self: (λn. h (real n)) ∈ O(λn. h (real (Suc n)))
  assumes h-pos: ⋀x. x ≥ 1 ⟹ h x > 0
  assumes mono-g: mono-on {1..} g ∨ mono-on {1..} (λx. − g x)
  assumes mono-h: mono-on {1..} h ∨ mono-on {1..} (λx. − h x)
  shows   ∃ c>0. ∀ x≥1. |sum-upto f x − g x| ≤ c * h x
proof −
  have ∃ c>0. ∀ x≥1. sum-upto f x − g x ≤ c * h x
    by (intro sum-upto-asymptotics-lift-nat-real-aux assms)
  then obtain c1 where c1: c1 > 0 ⋀x. x ≥ 1 ⟹ sum-upto f x − g x ≤ c1 *
h x
    by auto

  have (λn. −(g (real n) − g (real (Suc n)))) ∈ O(λn. h (real n))
```

```
    by (subst landau-o.big.uminus-in-iff) fact
  also have (λn. −(g (real n) − g (real (Suc n)))) = (λn. g (real (Suc n)) − g
(real n))
    by simp
  finally have (λn. g (real (Suc n)) − g (real n)) ∈ O(λn. h (real n)) .
  moreover {
    have (λn. −((∑ k=1..n. f k) − g (real n))) ∈ O(λn. h (real n))
      by (subst landau-o.big.uminus-in-iff) fact
    also have (λn. −((∑ k=1..n. f k) − g (real n))) =
                (λn. (∑ k=1..n. −f k) + g (real n)) by (simp add: sum-negf)
    finally have (λn. (∑ k=1..n. − f k) + g (real n)) ∈ O(λn. h (real n)) .
  }
  ultimately have ∃ c>0. ∀ x≥1. sum-upto (λn. −f n) x − (−g x) ≤ c ∗ h x
using mono-g
   by (intro sum-upto-asymptotics-lift-nat-real-aux assms) (simp-all add: disj-commute)
  then obtain c2 where c2: c2 > 0 ⋀x. x ≥ 1 ⟹ sum-upto (λn. − f n) x + g
x ≤ c2 ∗ h x
    by auto

  {
    fix x :: real assume x: x ≥ 1
    have sum-upto f x − g x ≤ max c1 c2 ∗ h x
      using h-pos[of x] x by (intro order.trans[OF c1(2)] mult-right-mono) auto
    moreover have sum-upto (λn. −f n) x + g x ≤ max c1 c2 ∗ h x
      using h-pos[of x] x by (intro order.trans[OF c2(2)] mult-right-mono) auto
    hence −(sum-upto f x − g x) ≤ max c1 c2 ∗ h x
      by (simp add: sum-upto-def sum-negf)
    ultimately have |sum-upto f x − g x| ≤ max c1 c2 ∗ h x by linarith
  }
  moreover from c1(1) c2(1) have max c1 c2 > 0 by simp
  ultimately show ?thesis by blast
qed

lemma (in factorial-semiring) primepow-divisors-induct [case-names zero unit fac-
tor]:
  assumes P 0 ⋀x. is-unit x ⟹ P x
        ⋀p k x. prime p ⟹ k > 0 ⟹ ¬p dvd x ⟹ P x ⟹ P (p ^ k ∗ x)
  shows   P x
proof −
  have finite (prime-factors x) by simp
  thus ?thesis
  proof (induction prime-factors x arbitrary: x rule: finite-induct)
    case empty
    hence prime-factors x = {} by metis
    hence prime-factorization x = {#} by simp
    thus ?case using assms(1,2) by (auto simp: prime-factorization-empty-iff)
  next
    case (insert p A x)
    define k where k = multiplicity p x
```

have $k > 0$ **using** *insert.hyps*
by (*auto simp*: *prime-factors-multiplicity k-def*)
have p: $p \in$ *prime-factors* x **using** *insert.hyps* **by** *auto*
from p **have** $x \neq 0$ $\neg$*is-unit* p **by** (*auto simp*: *in-prime-factors-iff*)

from *multiplicity-decompose'*[*OF this*] **obtain** y **where** y: $x = p \mathbin{\hat{}} k * y$ $\neg p$ *dvd* y
by (*auto simp*: *k-def*)
have *prime-factorization* x = *replicate-mset* k p + *prime-factorization* y
using p ‹$k > 0$› y **unfolding** y
by (*subst prime-factorization-mult*)
(*auto simp*: *prime-factorization-prime-power in-prime-factors-iff*)
moreover from y p **have** $p \notin$ *prime-factors* y
by (*auto simp*: *in-prime-factors-iff*)
ultimately have *prime-factors* y = *prime-factors* $x - \{p\}$
by *auto*
also have ... = A
using *insert.hyps* **by** *auto*
finally have P y **using** *insert* **by** *auto*
thus P x
unfolding y **using** y ‹$k > 0$› p **by** (*intro assms*(*3*)) (*auto simp*: *in-prime-factors-iff*)
qed
qed

end

2 Miscellaneous material

theory *More-Dirichlet-Misc*
imports
Prime-Distribution-Elementary-Library
Prime-Number-Theorem.Prime-Counting-Functions
begin

2.1 Generalised Dirichlet products

definition *dirichlet-prod'* :: (*nat* ⇒ 'a :: *comm-semiring-1*) ⇒ (*real* ⇒ 'a) ⇒ *real* ⇒ 'a **where**
dirichlet-prod' f g x = *sum-upto* (λm. f m * g (x / *real* m)) x

lemma *dirichlet-prod'-one-left*:
dirichlet-prod' (λn. *if* n = *1 then 1 else 0*) f x = (*if* $x \geq$ *1 then* f x *else 0*)
proof −
have *dirichlet-prod'* (λn. *if* n = *1 then 1 else 0*) f x =
(∑ i | *0* < i ∧ *real* $i \leq x$. (*if* i = *Suc 0 then 1 else 0*) * f (x / *real* i))
by (*simp add*: *dirichlet-prod'-def sum-upto-def*)
also have ... = (∑ i∈(*if* $x \geq$ *1 then* {*1*::*nat*} *else* {}). f x)
by (*intro sum.mono-neutral-cong-right*) (*auto split*: *if-splits*)
also have ... = (*if* $x \geq$ *1 then* f x *else 0*)

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

lemma *dirichlet-prod'-cong*:
assumes $\bigwedge n.\ n > 0 \Longrightarrow real\ n \leq x \Longrightarrow f\ n = f'\ n$
assumes $\bigwedge y.\ y \geq 1 \Longrightarrow y \leq x \Longrightarrow g\ y = g'\ y$
assumes $x = x'$
shows *dirichlet-prod' f g x = dirichlet-prod' f' g' x'*
unfolding *dirichlet-prod'-def*
by (*intro sum-upto-cong' assms*, (*subst assms* | *simp add*: *assms field-simps*)+)

lemma *dirichlet-prod'-assoc*:
dirichlet-prod' f ($\lambda y.$ *dirichlet-prod' g h y*) *x = dirichlet-prod'* (*dirichlet-prod f g*) *h x*
proof −
have *dirichlet-prod' f* ($\lambda y.$ *dirichlet-prod' g h y*) *x* =
($\sum m \mid m > 0 \wedge real\ m \leq x.\ \sum n \mid n > 0 \wedge real\ n \leq x\ /\ m.\ f\ m * g\ n * h\ (x\ /\ (m * n))$)
by (*simp add*: *algebra-simps dirichlet-prod'-def dirichlet-prod-def sum-upto-def sum-distrib-left sum-distrib-right*)
also have ... = ($\sum$ (*m*,*n*)∈(*SIGMA m*:{*m*. *m* > 0 ∧ *real m* ≤ *x*}. {*n*. *n* > 0 ∧ *real n* ≤ *x* / *m*}).
f m * *g n* * *h* (*x* / (*m* * *n*)))
by (*subst sum.Sigma*) *auto*
also have ... = ($\sum$ (*mn*, *m*)∈(*SIGMA mn*:{*mn*. *mn* > 0 ∧ *real mn* ≤ *x*}. {*m*. *m dvd mn*}).
f m * *g* (*mn div m*) * *h* (*x* / *mn*))
by (*rule sum.reindex-bij-witness*[*of* - λ(*mn*, *m*). (*m*, *mn div m*) λ(*m*, *n*). (*m* * *n*, *m*)])
(*auto simp*: *case-prod-unfold field-simps dest*: *dvd-imp-le*)
also have ... = *dirichlet-prod'* (*dirichlet-prod f g*) *h x*
by (*subst sum.Sigma* [*symmetric*])
(*simp-all add*: *dirichlet-prod'-def dirichlet-prod-def sum-upto-def algebra-simps sum-distrib-left sum-distrib-right*)
finally show *?thesis* **.**
qed

lemma *dirichlet-prod'-inversion1*:
assumes ∀ *x*≥*1*. *g x = dirichlet-prod' a f x x* ≥ *1*
dirichlet-prod a ainv = (λ*n*. *if n = 1 then 1 else 0*)
shows *f x = dirichlet-prod' ainv g x*
proof −
have *dirichlet-prod' ainv g x = dirichlet-prod' ainv* (*dirichlet-prod' a f*) *x*
using *assms* **by** (*intro dirichlet-prod'-cong*) *auto*
also have ... = *dirichlet-prod'* (λ*n*. *if n = 1 then 1 else 0*) *f x*
using *assms* **by** (*simp add*: *dirichlet-prod'-assoc dirichlet-prod-commutes*)

also have $\ldots = f\ x$
using *assms* **by** (*subst dirichlet-prod'-one-left*) *auto*
finally show *?thesis* **..**
qed

lemma *dirichlet-prod'-inversion2*:
assumes $\forall\, x{\geq}1.\ f\ x = \textit{dirichlet-prod'}\ \textit{ainv}\ g\ x\ x \geq 1$
dirichlet-prod a ainv $= (\lambda n.\ \textit{if}\ n = 1\ \textit{then}\ 1\ \textit{else}\ 0)$
shows $g\ x = \textit{dirichlet-prod'}\ a\ f\ x$
proof −
have *dirichlet-prod' a f x* $=$ *dirichlet-prod' a* (*dirichlet-prod' ainv g*) x
using *assms* **by** (*intro dirichlet-prod'-cong*) *auto*
also have $\ldots$ $=$ *dirichlet-prod'* $(\lambda n.\ \textit{if}\ n = 1\ \textit{then}\ 1\ \textit{else}\ 0)\ g\ x$
using *assms* **by** (*simp add*: *dirichlet-prod'-assoc dirichlet-prod-commutes*)
also have $\ldots = g\ x$
using *assms* **by** (*subst dirichlet-prod'-one-left*) *auto*
finally show *?thesis* **..**
qed

lemma *dirichlet-prod'-inversion*:
assumes *dirichlet-prod a ainv* $= (\lambda n.\ \textit{if}\ n = 1\ \textit{then}\ 1\ \textit{else}\ 0)$
shows $(\forall\, x{\geq}1.\ g\ x = \textit{dirichlet-prod'}\ a\ f\ x) \longleftrightarrow (\forall\, x{\geq}1.\ f\ x = \textit{dirichlet-prod'}\ \textit{ainv}\ g\ x)$
using *dirichlet-prod'-inversion1*[*of g a f - ainv*] *dirichlet-prod'-inversion2*[*of f ainv g - a*]
assms **by** *blast*

lemma *dirichlet-prod'-inversion'*:
assumes $a\ 1 * y = 1$
defines $\textit{ainv} \equiv \textit{dirichlet-inverse}\ a\ y$
shows $(\forall\, x{\geq}1.\ g\ x = \textit{dirichlet-prod'}\ a\ f\ x) \longleftrightarrow (\forall\, x{\geq}1.\ f\ x = \textit{dirichlet-prod'}\ \textit{ainv}\ g\ x)$
unfolding *ainv-def*
by (*intro dirichlet-prod'-inversion dirichlet-prod-inverse assms*)

lemma *dirichlet-prod'-floor-conv-sum-upto*:
dirichlet-prod' f $(\lambda x.\ \textit{real-of-int}\ (\textit{floor}\ x))\ x = \textit{sum-upto}\ (\lambda n.\ \textit{sum-upto}\ f\ (x\ /\ n))\ x$
proof −
have [*simp*]: *sum-upto* $(\lambda\text{-}.\ 1 :: \textit{real})\ x = \textit{real}\ (\textit{nat}\ \lfloor x \rfloor)$ **for** x
by (*simp add*: *sum-upto-altdef*)
show *?thesis*
using *sum-upto-dirichlet-prod*[*of* $\lambda n.\ 1{::}\textit{real}\ f$] *sum-upto-dirichlet-prod*[*of f* $\lambda n.\ 1{::}\textit{real}$]
by (*simp add*: *dirichlet-prod'-def dirichlet-prod-commutes*)
qed

lemma (**in** *completely-multiplicative-function*) *dirichlet-prod-self*:

*dirichlet-prod f f n = f n * of-nat (divisor-count n)*
proof *(cases n = 0)*
case *False*
have *dirichlet-prod f f n = (∑ (r, d) | r * d = n. f (r * d))*
by *(simp add: dirichlet-prod-altdef2 mult)*
also have *... = (∑ (r, d) | r * d = n. f n)*
by *(intro sum.cong) auto*
also have *... = f n * of-nat (card {(r, d). r * d = n})*
by *(simp add: mult.commute)*
also have *bij-betw fst {(r, d). r * d = n} {r. r dvd n}*
by *(rule bij-betwI[of - - - λr. (r, n div r)]) (use False* **in** *auto)*
hence *card {(r, d). r * d = n} = card {r. r dvd n}*
by *(rule bij-betw-same-card)*
also have *... = divisor-count n*
by *(simp add: divisor-count-def)*
finally show *?thesis* **.**
qed *auto*

lemma *completely-multiplicative-imp-moebius-mu-inverse*:
fixes *f :: nat ⇒ 'a :: {comm-ring-1}*
assumes *completely-multiplicative-function f*
shows *dirichlet-prod f (λn. moebius-mu n * f n) n = (if n = 1 then 1 else 0)*
proof −
interpret *completely-multiplicative-function f* **by** *fact*
have [*simp*]: *fds f ≠ 0* **by** *(auto simp: fds-eq-iff)*
have *dirichlet-prod f (λn. moebius-mu n * f n) n =*
*(∑ (r, d) | r * d = n. moebius-mu r * f (r * d))*
by *(subst dirichlet-prod-commutes)*
(simp add: fds-eq-iff fds-nth-mult fds-nth-fds dirichlet-prod-altdef2 mult-ac mult)
also have *... = (∑ (r, d) | r * d = n. moebius-mu r * f n)*
by *(intro sum.cong) auto*
also have *... = dirichlet-prod moebius-mu (λ-. 1) n * f n*
by *(simp add: dirichlet-prod-altdef2 sum-distrib-right case-prod-unfold mult)*
also have *dirichlet-prod moebius-mu (λ-. 1) n = fds-nth (fds moebius-mu * fds-zeta) n*
by *(simp add: fds-nth-mult)*
also have *fds moebius-mu * fds-zeta = 1*
by *(simp add: mult-ac fds-zeta-times-moebius-mu)*
also have *fds-nth 1 n * f n = fds-nth 1 n*
by *(auto simp: fds-eq-iff fds-nth-one)*
finally show *?thesis* **by** *(simp add: fds-nth-one)*
qed

lemma *dirichlet-prod-inversion-completely-multiplicative*:
fixes *a :: nat ⇒ 'a :: comm-ring-1*
assumes *completely-multiplicative-function a*
shows *(∀ x≥1. g x = dirichlet-prod' a f x) ⟷*

$(\forall x{\geq}1.\ f\ x = \textit{dirichlet-prod}'\ (\lambda n.\ \textit{moebius-mu}\ n * a\ n)\ g\ x)$
by (*intro dirichlet-prod'-inversion ext completely-multiplicative-imp-moebius-mu-inverse assms*)

lemma *divisor-sigma-conv-dirichlet-prod*:
divisor-sigma x n = dirichlet-prod ($\lambda n.$ *real n powr x*) (λ-. *1*) *n*
proof (*cases n = 0*)
case *False*
have *fds* (*divisor-sigma x*) = *fds-shift x fds-zeta * fds-zeta*
using *fds-divisor-sigma*[*of x*] **by** (*simp add*: *mult-ac*)
thus *?thesis* **using** *False* **by** (*auto simp*: *fds-eq-iff fds-nth-mult*)
qed *simp-all*

2.2 Legendre's identity

definition *legendre-aux* :: *real* ⇒ *nat* ⇒ *nat* **where**
legendre-aux x p = (*if prime p then* ($\sum m \mid m > 0 \wedge$ *real* ($p \mathbin{\hat{}} m$) $\leq x$. *nat* $\lfloor x / p \mathbin{\hat{}} m \rfloor$) *else 0*)

lemma *legendre-aux-not-prime* [*simp*]: ¬*prime p* ⟹ *legendre-aux x p = 0*
by (*simp add*: *legendre-aux-def*)

lemma *legendre-aux-eq-0*:
assumes *real p > x*
shows *legendre-aux x p = 0*
proof (*cases prime p*)
case *True*
have [*simp*]: ¬*real p* $\mathbin{\hat{}}$ *m* ≤ *x* **if** *m > 0* **for** *m*
proof −
have *x < real p* $\mathbin{\hat{}}$ *1* **using** *assms* **by** *simp*
also have ... ≤ *real p* $\mathbin{\hat{}}$ *m*
using *prime-gt-1-nat*[*OF True*] *that* **by** (*intro power-increasing*) *auto*
finally show *?thesis* **by** *auto*
qed
from *assms* **have** ∗: {*m. m > 0* ∧ *real* ($p \mathbin{\hat{}} m$) ≤ *x*} = {}
using *prime-gt-1-nat*[*OF True*] **by** *auto*
show *?thesis* **unfolding** *legendre-aux-def*
by (*subst* ∗) *auto*
qed (*auto simp*: *legendre-aux-def*)

lemma *legendre-aux-posD*:
assumes *legendre-aux x p > 0*
shows *prime p real p ≤ x*
proof −
show *real p ≤ x* **using** *legendre-aux-eq-0*[*of x p*] *assms*
by (*cases real p ≤ x*) *auto*
qed (*use assms* **in** ‹*auto simp*: *legendre-aux-def split*: *if-splits*›)

lemma *exponents-le-finite*:

```
  assumes p > (1 :: nat) k > 0
  shows   finite {i. real (p ^ (k * i + l)) ≤ x}
proof (rule finite-subset)
  show {i. real (p ^ (k * i + l)) ≤ x} ⊆ {..nat ⌊x⌋}
  proof safe
    fix i assume i: real (p ^ (k * i + l)) ≤ x
    have i < 2 ^ i
      by (rule less-exp)
    also from assms have i ≤ k * i + l by (cases k) auto
    hence 2 ^ i ≤ (2 ^ (k * i + l) :: nat)
      using assms by (intro power-increasing) auto
    also have ... ≤ p ^ (k * i + l) using assms by (intro power-mono) auto
    also have real ... ≤ x using i by simp
    finally show i ≤ nat ⌊x⌋ by linarith
  qed
qed auto

lemma finite-sum-legendre-aux:
  assumes prime p
  shows   finite {m. m > 0 ∧ real (p ^ m) ≤ x}
  by (rule finite-subset[OF - exponents-le-finite[where k = 1 and l = 0 and p =
p]])
     (use assms prime-gt-1-nat[of p] in auto)

lemma legendre-aux-set-eq:
  assumes prime p x ≥ 1
  shows   {m. m > 0 ∧ real (p ^ m) ≤ x} = {0<..nat ⌊log (real p) x⌋}
  using prime-gt-1-nat[OF assms(1)] assms
  by (auto simp: le-nat-iff le-log-iff le-floor-iff powr-realpow)

lemma legendre-aux-altdef1:
  legendre-aux x p = (if prime p ∧ x ≥ 1 then
                        (∑ m∈{0<..nat ⌊log (real p) x⌋}. nat ⌊x / p ^ m⌋) else 0)
proof (cases prime p ∧ x < 1)
  case False
  thus ?thesis using legendre-aux-set-eq[of p x] by (auto simp: legendre-aux-def)
next
  case True
  have [simp]: ¬(real p ^ m ≤ x) for m
  proof −
    have x < real 1 using True by simp
    also have real 1 ≤ real (p ^ m)
      unfolding of-nat-le-iff by (intro one-le-power) (use prime-gt-1-nat[of p] True
in auto)
    finally show ¬(real p ^ m ≤ x) by auto
  qed
  have {m. m > 0 ∧ real (p ^ m) ≤ x} = {} by simp
  with True show ?thesis by (simp add: legendre-aux-def)
qed
```

lemma *legendre-aux-altdef2*:
assumes $x \geq 1$ *prime p real p* $\hat{}$ *Suc k* $> x$
shows *legendre-aux x p* $= (\sum m \in \{0<..k\}.\ nat\ \lfloor x\ /\ p\ \hat{}\ m \rfloor)$
proof −
have *legendre-aux x p* $= (\sum m \mid m > 0 \wedge real\ (p\ \hat{}\ m) \leq x.\ nat\ \lfloor x\ /\ p\ \hat{}\ m \rfloor)$
using *assms* **by** (*simp add*: *legendre-aux-def*)
also have ... $= (\sum m \in \{0<..k\}.\ nat\ \lfloor x\ /\ p\ \hat{}\ m \rfloor)$
proof (*intro sum.mono-neutral-left*)
show $\{m.\ 0 < m \wedge real\ (p\ \hat{}\ m) \leq x\} \subseteq \{0<..k\}$
proof *safe*
fix *m* **assume** $m > 0$ *real* $(p\ \hat{}\ m) \leq x$
hence *real p* $\hat{}$ *m* $\leq x$ **by** *simp*
also note ‹$x <$ *real p* $\hat{}$ *Suc k*›
finally show $m \in \{0<..k\}$ **using** ‹$m > 0$›
using *prime-gt-1-nat*[*OF* ‹*prime p*›] **by** (*subst* (*asm*) *power-strict-increasing-iff*) *auto*
qed
qed (*use prime-gt-0-nat*[*of p*] *assms* **in** ‹*auto simp*: *field-simps*›)
finally show *?thesis* **.**
qed

theorem *legendre-identity*:
sum-upto ln x $=$ *prime-sum-upto* $(\lambda p.$ *legendre-aux x p* $*$ *ln p*$)$ *x*
proof −
define *S* **where** $S = (SIGMA\ p{:}\{p.\ prime\ p \wedge real\ p \leq x\}.\ \{i.\ i > 0 \wedge real\ (p\ \hat{}\ i) \leq x\})$

have *prime-power-leD*: *real p* $\leq x$ **if** *real p* $\hat{}$ *i* $\leq x$ *prime p i* > 0 **for** *p i*
proof −
have *real p* $\hat{}$ $1 \leq$ *real p* $\hat{}$ *i*
using *that prime-gt-1-nat*[*of p*] **by** (*intro power-increasing*) *auto*
also have ... $\leq x$ **by** *fact*
finally show *real p* $\leq x$ **by** *simp*
qed

have *sum-upto ln x* $=$ *sum-upto* $(\lambda n.$ *mangoldt n* $*$ *real* $(nat\ \lfloor x\ /$ *real n*$\rfloor))$ *x*
by (*rule sum-upto-ln-conv-sum-upto-mangoldt*)
also have ... $= (\sum (p,\ i) \mid prime\ p \wedge 0 < i \wedge real\ (p\ \hat{}\ i) \leq x.$
ln p $*$ *real* $(nat\ \lfloor x\ /$ *real* $(p\ \hat{}\ i)\rfloor))$
by (*subst sum-upto-primepows*[**where** $g = \lambda p\ i.$ *ln p* $*$ *real* $(nat\ \lfloor x\ /$ *real* $(p\ \hat{}\ i)\rfloor)$]$)$
(*auto simp*: *mangoldt-non-primepow*)
also have ... $= (\sum (p,i) \in S.$ *ln p* $*$ *real* $(nat\ \lfloor x\ /\ p\ \hat{}\ i \rfloor))$
using *prime-power-leD* **by** (*intro sum.cong refl*) (*auto simp*: *S-def*)
also have ... $= (\sum p \mid prime\ p \wedge real\ p \leq x.\ \sum i \mid i > 0 \wedge real\ (p\ \hat{}\ i) \leq x.$
ln p $*$ *real* $(nat\ \lfloor x\ /\ p\ \hat{}\ i \rfloor))$
proof (*unfold S-def*, *subst sum.Sigma*)

```
    have {p. prime p ∧ real p ≤ x} ⊆ {..nat ⌊x⌋}
      by (auto simp: le-nat-iff le-floor-iff)
    thus finite {p. prime p ∧ real p ≤ x}
      by (rule finite-subset) auto
  next
    show ∀p∈{p. prime p ∧ real p ≤ x}. finite {i. 0 < i ∧ real (p ^ i) ≤ x}
      by (intro ballI finite-sum-legendre-aux) auto
  qed auto
  also have ... = (∑ p | prime p ∧ real p ≤ x. ln p *
                    real (∑ i | i > 0 ∧ real (p ^ i) ≤ x. (nat ⌊x / p ^ i⌋)))
    by (simp add: sum-distrib-left)
  also have ... = (∑ p | prime p ∧ real p ≤ x. ln p * real (legendre-aux x p))
    by (intro sum.cong refl) (auto simp: legendre-aux-def)
  also have ... = prime-sum-upto (λp. ln p * real (legendre-aux x p)) x
    by (simp add: prime-sum-upto-def)
  finally show ?thesis by (simp add: mult-ac)
qed

lemma legendre-identity':
  fact (nat ⌊x⌋) = (∏ p | prime p ∧ real p ≤ x. p ^ legendre-aux x p)
proof −
  have fin: finite {p. prime p ∧ real p ≤ x}
    by (rule finite-subset[of - {..nat ⌊x⌋}]) (auto simp: le-nat-iff le-floor-iff)
  have real (fact (nat ⌊x⌋)) = exp (sum-upto ln x)
    by (subst sum-upto-ln-conv-ln-fact) auto
  also have sum-upto ln x = prime-sum-upto (λp. legendre-aux x p * ln p) x
    by (rule legendre-identity)
  also have exp ... = (∏ p | prime p ∧ real p ≤ x. exp (ln (real p) * legendre-aux
x p))
      unfolding prime-sum-upto-def using fin by (subst exp-sum) (auto simp:
mult-ac)
  also have ... = (∏ p | prime p ∧ real p ≤ x. real (p ^ legendre-aux x p))
  proof (intro prod.cong refl)
    fix p assume p ∈ {p. prime p ∧ real p ≤ x}
    hence p > 0 using prime-gt-0-nat[of p] by auto
     from ‹p > 0› have exp (ln (real p) * legendre-aux x p) = real p powr real
(legendre-aux x p)
      by (simp add: powr-def)
    also from ‹p > 0› have ... = real (p ^ legendre-aux x p)
      by (subst powr-realpow) auto
    finally show exp (ln (real p) * legendre-aux x p) = ... .
  qed
  also have ... = real (∏ p | prime p ∧ real p ≤ x. p ^ legendre-aux x p)
    by simp
  finally show ?thesis unfolding of-nat-eq-iff .
qed
```

2.3 A weighted sum of the Möbius μ function

context
fixes $M :: real \Rightarrow real$
defines $M \equiv (\lambda x.\ sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n\ /\ n)\ x)$
begin

lemma *abs-sum-upto-moebius-mu-over-n-less*:
assumes x: $x \geq 2$
shows $|M\ x| < 1$
proof −
have $x * sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n\ /\ n)\ x - sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * frac\ (x\ /\ n))\ x =$
$sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * (x\ /\ n - frac\ (x\ /\ n)))\ x$
by (*subst mult.commute*[*of x*])
(*simp add*: *sum-upto-def sum-distrib-right sum-subtractf ring-distribs*)
also have $(\lambda n.\ x\ /\ real\ n - frac\ (x\ /\ real\ n)) = (\lambda n.\ of\text{-}int\ (floor\ (x\ /\ real\ n)))$
by (*simp add*: *frac-def*)
also have $sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * real\text{-}of\text{-}int\ \lfloor x\ /\ real\ n\rfloor)\ x =$
$real\text{-}of\text{-}int\ (sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * \lfloor x\ /\ real\ n\rfloor)\ x)$
by (*simp add*: *sum-upto-def*)
also have $\ldots = 1$
using x **by** (*subst sum-upto-moebius-times-floor-linear*) *auto*
finally have *eq*: $x * M\ x = 1 + sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * frac\ (x\ /\ n))\ x$
by (*simp add*: *M-def*)

have $x * |M\ x| = |x * M\ x|$
using x **by** (*simp add*: *abs-mult*)
also note *eq*
also have $|1 + sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * frac\ (x\ /\ n))\ x| \leq$
$1 + |sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * frac\ (x\ /\ n))\ x|$
by *linarith*
also have $|sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * frac\ (x\ /\ n))\ x| \leq$
$sum\text{-}upto\ (\lambda n.\ |moebius\text{-}mu\ n * frac\ (x\ /\ n)|)\ x$
unfolding *sum-upto-def* **by** (*rule sum-abs*)
also have $\ldots \leq sum\text{-}upto\ (\lambda n.\ frac\ (x\ /\ n))\ x$
unfolding *sum-upto-def* **by** (*intro sum-mono*) (*auto simp*: *moebius-mu-def abs-mult*)
also have $\ldots = (\sum n \in \{0<..nat\ \lfloor x\rfloor\}.\ frac\ (x\ /\ n))$
by (*simp add*: *sum-upto-altdef*)
also have $\{0<..nat\ \lfloor x\rfloor\} = insert\ 1\ \{1<..nat\ \lfloor x\rfloor\}$
using x **by** (*auto simp*: *le-nat-iff le-floor-iff*)
also have $(\sum n \in \ldots.\ frac\ (x\ /\ n)) = frac\ x + (\sum n \in \{1<..nat\ \lfloor x\rfloor\}.\ frac\ (x\ /\ n))$
by (*subst sum.insert*) *auto*
also have $(\sum n \in \{1<..nat\ \lfloor x\rfloor\}.\ frac\ (x\ /\ n)) < (\sum n \in \{1<..nat\ \lfloor x\rfloor\}.\ 1)$
using x **by** (*intro sum-strict-mono frac-lt-1*) *auto*
also have $\ldots = nat\ \lfloor x\rfloor - 1$ **by** *simp*
also have $1 + (frac\ x + real\ (nat\ \lfloor x\rfloor - 1)) = x$
using x **by** (*subst of-nat-diff*) (*auto simp*: *le-nat-iff le-floor-iff frac-def*)

finally have $x * |M\ x| < x * 1$ **by** *simp*
with x **show** $|M\ x| < 1$
by (*subst* (*asm*) *mult-less-cancel-left-pos*) *auto*
qed

lemma *sum-upto-moebius-mu-over-n-eq*:
assumes $x < 2$
shows $M\ x = (\mathit{if}\ x \geq 1\ \mathit{then}\ 1\ \mathit{else}\ 0)$
proof (*cases* $x \geq 1$)
case *True*
have $M\ x = (\sum n{\in}\{n.\ n > 0 \wedge \mathit{real}\ n \leq x\}.\ \mathit{moebius\text{-}mu}\ n\ /\ n)$
by (*simp add*: *M-def sum-upto-def*)
also from *assms True* **have** $\{n.\ n > 0 \wedge \mathit{real}\ n \leq x\} = \{1\}$
by *auto*
thus *?thesis* **using** *True* **by** (*simp add*: *M-def sum-upto-def*)
next
case *False*
have $M\ x = (\sum n{\in}\{n.\ n > 0 \wedge \mathit{real}\ n \leq x\}.\ \mathit{moebius\text{-}mu}\ n\ /\ n)$
by (*simp add*: *M-def sum-upto-def*)
also from *False* **have** $\{n.\ n > 0 \wedge \mathit{real}\ n \leq x\} = \{\}$
by *auto*
finally show *?thesis* **using** *False* **by** (*simp add*: *M-def*)
qed

lemma *abs-sum-upto-moebius-mu-over-n-le*: $|M\ x| \leq 1$
using *sum-upto-moebius-mu-over-n-eq*[*of x*] *abs-sum-upto-moebius-mu-over-n-less*[*of x*]
by (*cases* $x < 2$) *auto*

end

end

3 The Prime ω function

theory *Primes-Omega*
imports *Dirichlet-Series.Dirichlet-Series Dirichlet-Series.Divisor-Count*
begin

The prime ω function $\omega(n)$ counts the number of distinct prime factors of n.

definition *primes-omega* :: *nat* ⇒ *nat* **where**
primes-omega n = *card* (*prime-factors n*)

lemma *primes-omega-prime* [*simp*]: *prime p* ⟹ *primes-omega p* = *1*
by (*simp add*: *primes-omega-def prime-factorization-prime*)

lemma *primes-omega-0* [*simp*]: *primes-omega 0* = *0*
by (*simp add*: *primes-omega-def*)

lemma *primes-omega-1* [*simp*]: *primes-omega 1 = 0*
by (*simp add*: *primes-omega-def*)

lemma *primes-omega-Suc-0* [*simp*]: *primes-omega* (*Suc 0*) = *0*
by (*simp add*: *primes-omega-def*)

lemma *primes-omega-power* [*simp*]: *n* > *0* ⟹ *primes-omega* (*x* ^ *n*) = *primes-omega x*
by (*simp add*: *primes-omega-def prime-factors-power*)

lemma *primes-omega-primepow* [*simp*]: *primepow n* ⟹ *primes-omega n = 1*
by (*auto simp*: *primepow-def*)

lemma *primes-omega-eq-0-iff*: *primes-omega n = 0* ⟷ *n = 0* ∨ *n = 1*
by (*auto simp*: *primes-omega-def prime-factorization-empty-iff*)

lemma *primes-omega-pos* [*simp*, *intro*]: *n* > *1* ⟹ *primes-omega n* > *0*
by (*cases primes-omega n* > *0*) (*auto simp*: *primes-omega-eq-0-iff*)

lemma *primes-omega-mult-coprime*:
assumes *coprime x y x* > *0* ∨ *y* > *0*
shows *primes-omega* (*x* ∗ *y*) = *primes-omega x* + *primes-omega y*
proof (*cases x = 0* ∨ *y = 0*)
case *False*
hence *prime-factors* (*x* ∗ *y*) = *prime-factors x* ∪ *prime-factors y*
by (*subst prime-factorization-mult*) *auto*
also **{**
have *prime-factors x* ∩ *prime-factors y = set-mset* (*prime-factorization* (*gcd x y*))
using *False* **by** (*subst prime-factorization-gcd*) *auto*
also have *gcd x y = 1* **using** ‹*coprime x y*› **by** *auto*
finally have *card* (*prime-factors x* ∪ *prime-factors y*) = *primes-omega x* + *primes-omega y*
unfolding *primes-omega-def* **by** (*intro card-Un-disjoint*) (*use False* **in** *auto*)
}
finally show *?thesis* **by** (*simp add*: *primes-omega-def*)
qed (*use assms* **in** *auto*)

lemma *divisor-count-squarefree*:
assumes *squarefree n n* > *0*
shows *divisor-count n = 2* ^ *primes-omega n*
proof −
have *divisor-count n* = (∏ *p*∈*prime-factors n*. *Suc* (*multiplicity p n*))
using *assms* **by** (*subst divisor-count.prod-prime-factors′*) *auto*
also have ... = (∏ *p*∈*prime-factors n*. *2*)
using *assms assms* **by** (*intro prod.cong*) (*auto simp*: *squarefree-factorial-semiring′*)
finally show *?thesis* **by** (*simp add*: *primes-omega-def*)
qed

end

4 The Primorial function

theory *Primorial*
  imports *Prime-Distribution-Elementary-Library Primes-Omega*
begin

4.1 Definition and basic properties

definition *primorial* :: *real* ⇒ *nat* **where**
  primorial x = ∏ {*p*. *prime p* ∧ *real p* ≤ *x*}

lemma *primorial-mono*: *x* ≤ *y* ⟹ *primorial x* ≤ *primorial y*
  unfolding *primorial-def*
  by (*intro dvd-imp-le prod-dvd-prod-subset*)
    (*auto intro*!: *prod-pos finite-primes-le dest*: *prime-gt-0-nat*)

lemma *prime-factorization-primorial*:
  prime-factorization (*primorial x*) = *mset-set* {*p*. *prime p* ∧ *real p* ≤ *x*}
proof (*intro multiset-eqI*)
  fix *p* :: *nat*
  note *fin* = *finite-primes-le*[*of x*]
  show *count* (*prime-factorization* (*primorial x*)) *p* =
      count (*mset-set* {*p*. *prime p* ∧ *real p* ≤ *x*}) *p*
  proof (*cases prime p*)
    case *True*
    hence *count* (*prime-factorization* (*primorial x*)) *p* =
        sum (*multiplicity p*) {*p*. *prime p* ∧ *real p* ≤ *x*}
      unfolding *primorial-def count-prime-factorization* **using** *fin*
      by (*subst prime-elem-multiplicity-prod-distrib*) *auto*
    also from *True* **have** ... = *sum* (λ-. *1*) (*if p* ≤ *x then* {*p*} *else* {}) **using** *fin*
      by (*intro sum.mono-neutral-cong-right*) (*auto simp*: *prime-multiplicity-other*
split: *if-splits*)
    also have ... = *count* (*mset-set* {*p*. *prime p* ∧ *real p* ≤ *x*}) *p*
      using *True fin* **by** *auto*
    finally show *?thesis* **.**
  qed *auto*
qed

lemma *prime-factors-primorial* [*simp*]:
  prime-factors (*primorial x*) = {*p*. *prime p* ∧ *real p* ≤ *x*}
  unfolding *prime-factorization-primorial* **using** *finite-primes-le*[*of x*] **by** *simp*

lemma *primorial-pos* [*simp*, *intro*]: *primorial x* > *0*
  unfolding *primorial-def* **by** (*intro prod-pos*) (*auto dest*: *prime-gt-0-nat*)

lemma *primorial-neq-zero* [*simp*]: *primorial x* ≠ *0*

by *auto*

lemma *of-nat-primes-omega-primorial* [*simp*]: *real* (*primes-omega* (*primorial x*)) = *primes-pi x*
by (*simp add*: *primes-omega-def primes-pi-def prime-sum-upto-def*)

lemma *primes-omega-primorial*: *primes-omega* (*primorial x*) = *nat* ⌊*primes-pi x*⌋
by (*simp add*: *primes-omega-def primes-pi-def prime-sum-upto-def*)

lemma *prime-dvd-primorial-iff*: *prime p* ⟹ *p dvd primorial x* ⟷ *p* ≤ *x*
using *finite-primes-le*[*of x*]
by (*auto simp*: *primorial-def prime-dvd-prod-iff dest*: *primes-dvd-imp-eq*)

lemma *squarefree-primorial* [*intro*]: *squarefree* (*primorial x*)
unfolding *primorial-def*
by (*intro squarefree-prod-coprime*) (*auto simp*: *squarefree-prime intro*: *primes-coprime*)

lemma *primorial-ge*: *primorial x* ≥ *2 powr primes-pi x*
proof −
have *2 powr primes-pi x* = *real* (∏ *p* | *prime p* ∧ *real p* ≤ *x*. *2*)
by (*simp add*: *primes-pi-def prime-sum-upto-def powr-realpow*)
also have (∏ *p* | *prime p* ∧ *real p* ≤ *x*. *2*) ≤ (∏ *p* | *prime p* ∧ *real p* ≤ *x*. *p*)
by (*intro prod-mono*) (*auto dest*: *prime-gt-1-nat*)
also have ... = *primorial x* **by** (*simp add*: *primorial-def*)
finally show *?thesis* **by** *simp*
qed

lemma *primorial-at-top*: *filterlim primorial at-top at-top*
proof −
have *filterlim* (λ*x*. *real* (*primorial x*)) *at-top at-top*
proof (*rule filterlim-at-top-mono*)
show *eventually* (λ*x*. *primorial x* ≥ *2 powr primes-pi x*) *at-top*
by (*intro always-eventually primorial-ge allI*)
have *filterlim* (λ*x*. *exp* (*ln 2* ∗ *primes-pi x*)) *at-top at-top*
by (*intro filterlim-compose*[*OF exp-at-top*]
filterlim-tendsto-pos-mult-at-top[*OF tendsto-const*] *π-at-top*) *auto*
thus *filterlim* (λ*x*. *2 powr primes-pi x*) *at-top at-top*
by (*simp add*: *powr-def mult-ac*)
qed
thus *?thesis* **unfolding** *filterlim-sequentially-iff-filterlim-real* [*symmetric*] **.**
qed

lemma *totient-primorial*:
real (*totient* (*primorial x*)) =
real (*primorial x*) ∗ (∏ *p* | *prime p* ∧ *real p* ≤ *x*. *1* − *1* / *real p*) **for** *x*
proof −
have *real* (*totient* (*primorial x*)) =
primorial x ∗ (∏ *p*∈*prime-factors* (*primorial x*). *1* − *1* / *p*)
by (*rule totient-formula2*)

thus *?thesis* **by** *simp*
qed

lemma *ln-primorial*: *ln* (*primorial x*) = *primes-theta x*
proof −
have *finite* {*p*. *prime p* ∧ *real p* ≤ *x*}
by (*rule finite-subset*[*of* - {..*nat* ⌊*x*⌋}]) (*auto simp*: *le-nat-iff le-floor-iff*)
thus *?thesis* **unfolding** *of-nat-prod primorial-def*
by (*subst ln-prod*) (*auto dest*: *prime-gt-0-nat simp*: *ϑ-def prime-sum-upto-def*)
qed

lemma *divisor-count-primorial*: *divisor-count* (*primorial x*) = *2 powr primes-pi x*
proof −
have *divisor-count* (*primorial x*) = (∏ *p* | *prime p* ∧ *real p* ≤ *x*. *divisor-count p*)
unfolding *primorial-def*
by (*subst divisor-count.prod-coprime*) (*auto simp*: *primes-coprime*)
also have ... = (∏ *p* | *prime p* ∧ *real p* ≤ *x*. *2*)
by (*intro prod.cong divisor-count.prime*) *auto*
also have ... = *2 powr primes-pi x*
by (*simp add*: *primes-pi-def prime-sum-upto-def powr-realpow*)
finally show *?thesis* **.**
qed

4.2 An alternative view on primorials

The following function is an alternative representation of primorials; instead of taking the product of all primes up to a given real bound x, it takes the product of the first k primes. This is sometimes more convenient.

definition *primorial′* :: *nat* ⇒ *nat* **where**
primorial′ n = (∏ *k*<*n*. *nth-prime k*)

lemma *primorial′-0* [*simp*]: *primorial′ 0* = *1*
and *primorial′-1* [*simp*]: *primorial′ 1* = *2*
and *primorial′-2* [*simp*]: *primorial′ 2* = *6*
and *primorial′-3* [*simp*]: *primorial′ 3* = *30*
by (*simp-all add*: *primorial′-def lessThan-nat-numeral*)

lemma *primorial′-Suc*: *primorial′* (*Suc n*) = *nth-prime n* ∗ *primorial′ n*
by (*simp add*: *primorial′-def*)

lemma *primorial′-pos* [*intro*]: *primorial′ n* > *0*
unfolding *primorial′-def* **by** (*auto intro*: *prime-gt-0-nat*)

lemma *primorial′-neq-0* [*simp*]: *primorial′ n* ≠ *0*
by *auto*

lemma *strict-mono-primorial′*: *strict-mono primorial′*
unfolding *strict-mono-Suc-iff*
proof

```
  fix n :: nat
  have primorial' n * 1 < primorial' n * nth-prime n
    using prime-gt-1-nat[OF prime-nth-prime[of n]]
    by (intro mult-strict-left-mono) auto
  thus primorial' n < primorial' (Suc n)
    by (simp add: primorial'-Suc)
qed

lemma prime-factorization-primorial':
  prime-factorization (primorial' k) = mset-set (nth-prime ` {..<k})
proof -
  have prime-factorization (primorial' k) = (∑ i<k. prime-factorization (nth-prime
i))
    unfolding primorial'-def by (subst prime-factorization-prod) (auto intro: prime-gt-0-nat)
  also have ... = (∑ i<k. {#nth-prime i#})
    by (intro sum.cong prime-factorization-prime) auto
  also have ... = (∑ p∈nth-prime ` {..<k}. {#p#})
    by (subst sum.reindex) (auto intro: inj-onI)
  also have ... = mset-set (nth-prime ` {..<k})
    by simp
  finally show ?thesis .
qed

lemma prime-factors-primorial': prime-factors (primorial' k) = nth-prime ` {..<k}
  by (simp add: prime-factorization-primorial')

lemma primes-omega-primorial' [simp]: primes-omega (primorial' k) = k
  unfolding primes-omega-def prime-factors-primorial' by (subst card-image) (auto
intro: inj-onI)

lemma squarefree-primorial' [intro]: squarefree (primorial' x)
  unfolding primorial'-def
  by (intro squarefree-prod-coprime) (auto intro: squarefree-prime intro: primes-coprime)

lemma divisor-count-primorial' [simp]: divisor-count (primorial' k) = 2 ^ k
  by (subst divisor-count-squarefree) auto

lemma totient-primorial':
  totient (primorial' k) = primorial' k * (∏ i<k. 1 - 1 / nth-prime i)
  unfolding totient-formula2 prime-factors-primorial'
  by (subst prod.reindex) (auto intro: inj-onI)

lemma primorial-conv-primorial': primorial x = primorial' (nat ⌊primes-pi x⌋)
  unfolding primorial-def primorial'-def
proof (rule prod.reindex-bij-witness[of - nth-prime λp. nat ⌊primes-pi (real p)⌋ -
1])
  fix p assume p: p ∈ {p. prime p ∧ real p ≤ x}
  have [simp]: primes-pi 2 = 1 by (auto simp: eval-π)
  have primes-pi p ≥ 1
```

```
    using p π-mono[of 2 real p] by (auto dest!: prime-gt-1-nat)
  with p show nth-prime (nat ⌊primes-pi p⌋ − 1) = p
    using π-pos[of real p]
    by (intro nth-prime-eqI'') (auto simp: le-nat-iff le-floor-iff primes-pi-def prime-sum-upto-def of-nat-diff)
  from p have nat ⌊primes-pi (real p)⌋ ≤ nat ⌊primes-pi x⌋
    by (intro nat-mono floor-mono π-mono) auto
  hence nat ⌊primes-pi (real p)⌋ − 1 < nat ⌊primes-pi x⌋
    using ‹primes-pi p ≥ 1› by linarith
  thus  nat ⌊primes-pi (real p)⌋ − 1 ∈ {..<nat ⌊primes-pi x⌋} by simp
  show nth-prime (nat ⌊primes-pi (real p)⌋ − 1) = p
    using p ‹primes-pi p ≥ 1›
   by (intro nth-prime-eqI'') (auto simp: le-nat-iff primes-pi-def prime-sum-upto-def)
next
  fix k assume k: k ∈ {..<nat ⌊primes-pi x⌋}
  thus ∗: nat ⌊primes-pi (real (nth-prime k))⌋ − 1 = k by auto
  from k have ¬(x < 2) by (intro notI) auto
  hence x ≥ 2 by simp
  have real (nth-prime k) ≤ real (nth-prime (nat ⌊primes-pi x⌋ − 1))
    using k by simp
  also have ... ≤ x
    using nth-prime-partition''[of x] ‹x ≥ 2› by auto
  finally show nth-prime k ∈ {p. prime p ∧ real p ≤ x}
    by auto
qed

lemma primorial'-conv-primorial:
  assumes n > 0
  shows   primorial' n = primorial (nth-prime (n − 1))
proof −
  have primorial (nth-prime (n − 1)) = (∏k<nat (int (n − 1) + 1). nth-prime
k)
    by (simp add: primorial-conv-primorial' primorial'-def)
  also have nat (int (n − 1) + 1) = n
    using assms by auto
  finally show ?thesis by (simp add: primorial'-def)
qed
```

4.3 Maximal compositeness of primorials

Primorials are maximally composite, i. e. any number with k distinct prime factors is as least as big as the primorial with k distinct prime factors, and and number less than a primorial has strictly less prime factors.

```
lemma nth-prime-le-prime-sequence:
  fixes p :: nat ⇒ nat
  assumes strict-mono-on {..<n} p and ⋀k. k < n ⟹ prime (p k) and k < n
  shows   nth-prime k ≤ p k
  using assms(3)
proof (induction k)
```

```
  case 0
  hence prime (p 0) by (intro assms)
  hence p 0 ≥ 2 by (auto dest: prime-gt-1-nat)
  thus ?case by simp
next
  case (Suc k)
  have IH: Suc (nth-prime k) ≤ Suc (p k) using Suc by simp
  have nth-prime (Suc k) = smallest-prime-beyond (Suc (nth-prime k))
    by (simp add: nth-prime-Suc)
  also {
    have Suc (nth-prime k) ≤ Suc (p k) using Suc by simp
    also have ... ≤ smallest-prime-beyond (Suc (p k))
      by (rule smallest-prime-beyond-le)
   finally have smallest-prime-beyond (Suc (nth-prime k)) ≤ smallest-prime-beyond
(Suc (p k))
      by (rule smallest-prime-beyond-smallest[OF prime-smallest-prime-beyond])
  }
  also have p k < p (Suc k)
    using Suc by (intro strict-mono-onD[OF assms(1)]) auto
  hence smallest-prime-beyond (Suc (p k)) ≤ p (Suc k)
    using Suc.prems by (intro smallest-prime-beyond-smallest assms) auto
  finally show ?case .
qed

theorem primorial'-primes-omega-le:
  fixes n :: nat
  assumes n: n > 0
  shows primorial' (primes-omega n) ≤ n
proof (cases n = 1)
  case True
  thus ?thesis by simp
next
  case False
  with assms have n > 1 by simp
  define m where m = primes-omega n
  define P where P = {p. prime p ∧ real p ≤ primes-pi n}
  define ps where ps = sorted-list-of-set (prime-factors n)
  have set-ps: set ps = prime-factors n by (simp add: ps-def)
  have [simp]: length ps = m by (simp add: ps-def m-def primes-omega-def)
  have sorted ps distinct ps by (simp-all add: ps-def)
  hence mono: strict-mono-on {..<m} (λk. ps ! k)
   by (intro strict-mono-onI le-neq-trans) (auto simp: sorted-nth-mono distinct-conv-nth)
  from ‹n > 1› have m > 0
    by (auto simp: m-def prime-factorization-empty-iff intro!: Nat.gr0I)

  have primorial' m = (∏k<m. nth-prime k)
    using ‹m > 0› by (simp add: of-nat-diff primorial'-def m-def)
  also have (∏k<m. nth-prime k) ≤ (∏k<m. ps ! k ^ multiplicity (ps ! k) n)
  proof (intro prod-mono conjI)
```

```
    fix i assume i: i ∈ {..<m}
    hence p: ps ! i ∈ prime-factors n
      using set-ps by (auto simp: set-conv-nth)
    with i set-ps have nth-prime i ≤ ps ! i
      by (intro nth-prime-le-prime-sequence[where n = m] mono) (auto simp:
set-conv-nth)
    also have ... ≤ ps ! i ^ multiplicity (ps ! i) n
      using p by (intro self-le-power) (auto simp: prime-factors-multiplicity dest:
prime-gt-1-nat)
    finally show nth-prime i ≤ ... .
  qed auto
  also have ... = (∏ p∈(λi. ps ! i) ` {..<m}. p ^ multiplicity p n)
    using ‹distinct ps› by (subst prod.reindex) (auto intro!: inj-onI simp: dis-
tinct-conv-nth)
  also have (λi. ps ! i) ` {..<m} = set ps
    by (auto simp: set-conv-nth)
  also have set ps = prime-factors n
    by (simp add: set-ps)
  also have (∏ p∈prime-factors n. p ^ multiplicity p n) = n
    using ‹n > 1› by (intro prime-factorization-nat [symmetric]) auto
  finally show primorial' m ≤ n .
qed

lemma primes-omega-less-primes-omega-primorial:
  fixes n :: nat
  assumes n: n > 0 and n < primorial x
  shows primes-omega n < primes-omega (primorial x)
proof (cases n > 1)
  case False
  have [simp]: primes-pi 2 = 1 by (simp add: eval-π)
  from False assms have [simp]: n = 1 by auto
  from assms have ¬(x < 2)
    by (intro notI) (auto simp: primorial-conv-primorial')
  thus ?thesis using assms π-mono[of 2 x] by auto
next
  case True
  define m where m = primes-omega n
  have le: primorial' m ≤ n
   using primorial'-primes-omega-le[of n] ‹n > 1› by (simp add: m-def primes-omega-def)
  also have ... < primorial x by fact
  also have ... = primorial' (nat ⌊primes-pi x⌋)
    by (simp add: primorial-conv-primorial')
  finally have m < nat ⌊primes-pi x⌋
    using strict-mono-less[OF strict-mono-primorial'] by simp
  hence m < primes-pi x by linarith
  also have ... = primes-omega (primorial x) by simp
  finally show ?thesis unfolding m-def of-nat-less-iff .
qed
```

lemma *primes-omega-le-primes-omega-primorial*:
fixes n :: *nat*
assumes $n \leq$ *primorial* x
shows *primes-omega* $n \leq$ *primes-omega* (*primorial* x)
proof −
consider $n = 0 \mid n > 0\ n =$ *primorial* $x \mid n > 0\ n \neq$ *primorial* x **by** *force*
thus *?thesis*
by *cases* (*use primes-omega-less-primes-omega-primorial*[*of* n x] *assms* **in** *auto*)
qed

end

5 The LCM of the first n natural numbers

theory *Lcm-Nat-Upto*
imports *Prime-Number-Theorem.Prime-Counting-Functions*
begin

In this section, we examine *Lcm* {*1*..*n*}. In particular, we will show that it is equal to $e^{\psi(n)}$ and thus (by the PNT) $e^{n+o(n)}$.

lemma *multiplicity-Lcm*:
fixes A :: $'a$ :: {*semiring-Gcd*, *factorial-semiring-gcd*} *set*
assumes *finite* A $A \neq \{\}$ *prime* p $0 \notin A$
shows *multiplicity* p (*Lcm* A) = *Max* (*multiplicity* p ‘ A)
using *assms*
proof (*induction* A *rule*: *finite-ne-induct*)
case (*insert* x A)
have *Lcm* (*insert* x A) = *lcm* x (*Lcm* A) **by** *simp*
also have *multiplicity* p ... = *Max* (*multiplicity* p ‘ *insert* x A)
using *insert* **by** (*subst multiplicity-lcm*) (*auto simp*: *Lcm-0-iff*)
finally show *?case* **by** *simp*
qed *auto*

The multiplicity of any prime p in *Lcm* {*1*..*n*} differs from that in *Lcm* {*1*..*n* − *1*} iff n is a power of p, in which case it is greater by 1.

lemma *multiplicity-Lcm-atLeast1AtMost-Suc*:
fixes p n :: *nat*
assumes p: *prime* p **and** n: $n > 0$
shows *multiplicity* p (*Lcm* {*1*..*Suc* n}) =
(*if* ∃ k. *Suc* $n = p \mathbin{\hat{}} k$ *then 1 else 0*) + *multiplicity* p (*Lcm* {*1*..n})
proof −
define k **where** k = *Max* (*multiplicity* p ‘ {*1*..n})
define l **where** l = *multiplicity* p (*Suc* n)
have *eq*: {*1*..*Suc* n} = *insert* (*Suc* n) {*1*..n} **by** *auto*
from ‹*prime* p› **have** $p > 1$ **by** (*auto dest*: *prime-gt-1-nat*)

have *multiplicity* p (*Lcm* {*1*..*Suc* n}) = *Max* (*multiplicity* p ‘ {*1*..*Suc* n})
using *assms* **by** (*subst multiplicity-Lcm*) *auto*

```
  also have multiplicity p ‘ {1..Suc n} =
              insert (multiplicity p (Suc n)) (multiplicity p ‘ {1..n})
    by (simp only: eq image-insert)
  also have Max ... = max l k
    unfolding l-def k-def using assms by (subst Max.insert) auto
  also have ... = (if ∃k. Suc n = p ^ k then 1 else 0) + k
  proof (cases ∃k. Suc n = p ^ k)
    case False
    have p ^ l dvd Suc n
      unfolding l-def by (intro multiplicity-dvd)
    hence p ^ l ≤ Suc n
      unfolding l-def by (intro dvd-imp-le multiplicity-dvd) auto
    moreover have Suc n ≠ p ^ l using False by blast
    ultimately have p ^ l < Suc n by linarith
    moreover have p ^ l > 0 using ‹p > 1› by (intro zero-less-power) auto
    ultimately have l = multiplicity p (p ^ l) and p ^ l ∈ {1..n}
      using ‹prime p› by auto
    hence l ≤ k unfolding k-def by (intro Max.coboundedI) auto
    with False show ?thesis by (simp add: l-def k-def)
  next
    case True
    then obtain x where x: Suc n = p ^ x by blast
    from x and ‹n > 0› have x > 0 by (intro Nat.gr0I) auto
    from x have [simp]: l = x using ‹prime p› by (simp add: l-def)
    have x = k + 1
    proof (intro antisym)
      have p ^ (x − 1) < Suc n
       using ‹x > 0› ‹p > 1› unfolding x by (intro power-strict-increasing) auto
      moreover have p ^ (x − 1) > 0
        using ‹p > 1› by (intro zero-less-power) auto
      ultimately have multiplicity p (p ^ (x − 1)) = x − 1 and p ^ (x − 1) ∈
{1..n}
        using ‹prime p› by auto
      hence x − 1 ≤ k
        unfolding k-def by (intro Max.coboundedI) force+
      thus x ≤ k + 1 by linarith
    next
      have multiplicity p y < x if y ∈ {1..n} for y
      proof −
        have p ^ multiplicity p y ≤ y
          using that by (intro dvd-imp-le multiplicity-dvd) auto
        also have ... < Suc n using that by simp
        also have ... = p ^ x by (fact x)
        finally show multiplicity p y < x
          using ‹p > 1› by (subst (asm) power-strict-increasing-iff)
      qed
      hence k < x
        using ‹n > 0› unfolding k-def by (subst Max-less-iff) auto
      thus k + 1 ≤ x by simp
```

```
    qed
    thus ?thesis using True by simp
  qed
  also have k = multiplicity p (Lcm {1..n})
    unfolding k-def using ‹n > 0› and ‹prime p› by (subst multiplicity-Lcm)
auto
  finally show ?thesis .
qed
```

Consequently, $Lcm\ \{1..n\}$ differs from $Lcm\ \{1..n-1\}$ iff n is of the form p^k for some prime p, in which case it is greater by a factor of p.

```
lemma Lcm-atLeast1AtMost-Suc:
  Lcm {1..Suc n} = Lcm {1..n} * (if primepow (Suc n) then aprimedivisor (Suc
n) else 1)
proof (cases n > 0)
  case True
  show ?thesis
  proof (rule multiplicity-eq-nat)
    fix p :: nat assume prime p
    define x where x = (if primepow (Suc n) then aprimedivisor (Suc n) else 1)
    have x > 0
      using ‹n > 0› by (auto simp: x-def intro!: aprimedivisor-pos-nat)

    have multiplicity p (Lcm {1..n} * x) = multiplicity p (Lcm {1..n}) + multi-
plicity p x
      using ‹prime p› ‹x > 0› by (subst prime-elem-multiplicity-mult-distrib) auto
    also consider ∃k. Suc n = p ^ k | primepow (Suc n) ¬(∃k. Suc n = p ^ k)
            | ¬primepow (Suc n) by blast
    hence multiplicity p x = (if ∃k. Suc n = p ^ k then 1 else 0)
    proof cases
      assume ∃k. Suc n = p ^ k
      thus ?thesis using ‹prime p› ‹n > 0›
        by (auto simp: x-def aprimedivisor-prime-power intro!: Nat.gr0I)
    next
      assume *: primepow (Suc n) ∄k. Suc n = p ^ k
      then obtain q k where qk: prime q Suc n = q ^ k k > 0 q ≠ p
        by (auto simp: primepow-def)
      thus ?thesis using ‹prime p›
      by (subst *) (auto simp: x-def aprimedivisor-prime-power prime-multiplicity-other)
    next
      assume *: ¬primepow (Suc n)
      hence **: ∄k. Suc n = p ^ k using ‹prime p› ‹n > 0› by auto
      from * show ?thesis unfolding x-def
        by (subst **) auto
    qed
    also have multiplicity p (Lcm {1..n}) + ... = multiplicity p (Lcm {1..Suc n})
      using ‹prime p› ‹n > 0› by (subst multiplicity-Lcm-atLeast1AtMost-Suc)
(auto simp: x-def)
    finally show multiplicity p (Lcm {1..Suc n}) = multiplicity p (Lcm {1..n} *
```

x) **..**
  qed (*use* ‹$n > 0$› **in** ‹*auto intro*!: *Nat.gr0I dest*: *aprimedivisor-pos-nat*›)
qed *auto*

It follows by induction that Lcm $\{1..n\} = e^{\psi(n)}$.

lemma *Lcm-atLeast1AtMost-conv-ψ*:
  includes *prime-counting-syntax*
  shows *real* (*Lcm* {*1*..*n*}) = *exp* (*ψ* (*real n*))
proof (*induction n*)
  case (*Suc n*)
  have *real* (*Lcm* {*1*..*Suc n*}) =
        real (*Lcm* {*1*..*n*}) ∗ (*if primepow* (*Suc n*) *then aprimedivisor* (*Suc n*) *else 1*)
    by (*subst Lcm-atLeast1AtMost-Suc*) *auto*
  also **{**
    assume *primepow* (*Suc n*)
    hence *Suc n* > *Suc 0* **by** (*rule primepow-gt-Suc-0*)
    hence *aprimedivisor* (*Suc n*) > *0* **by** (*intro aprimedivisor-pos-nat*) *auto*
  }
  hence (*if primepow* (*Suc n*) *then aprimedivisor* (*Suc n*) *else 1*) = *exp* (*mangoldt* (*Suc n*))
    by (*simp add*: *mangoldt-def*)
  also have *Lcm* {*1*..*n*} ∗ ... = *exp* (*ψ* (*real n* + *1*))
    using *Suc.IH* **by** (*simp add*: *primes-psi-def sum-upto-plus1 exp-add*)
  finally show *?case* **by** (*simp add*: *add-ac*)
qed *simp-all*

lemma *Lcm-upto-real-conv-ψ*:
  includes *prime-counting-syntax*
  shows *real* (*Lcm* {*1*..*nat* ⌊*x*⌋}) = *exp* (*ψ x*)
 by (*subst Lcm-atLeast1AtMost-conv-ψ*) (*simp add*: *primes-psi-def sum-upto-altdef*)

end

6 Shapiro's Tauberian Theorem

theory *Shapiro-Tauberian*
imports
  More-Dirichlet-Misc
  Prime-Number-Theorem.Prime-Counting-Functions
  Prime-Distribution-Elementary-Library
begin

6.1 Proof

Given an arithmeticla function $a(n)$, Shapiro's Tauberian theorem relates the sum $\sum_{n\leq x} a(n)$ to the weighted sums $\sum_{n\leq x} a(n)\lfloor\frac{x}{n}\rfloor$ and $\sum_{n\leq x} a(n)/n$. More precisely, it shows that if $\sum_{n\leq x} a(n)\lfloor\frac{x}{n}\rfloor = x\ln x + O(x)$, then:

- $\sum_{n \leq x} \frac{a(n)}{n} = \ln x + O(1)$
- $\sum_{n \leq x} a(n) \leq Bx$ for some constant $B \geq 0$ and all $x \geq 0$
- $\sum_{n \leq x} a(n) \geq Cx$ for some constant $C > 0$ and all $x \geq 1/C$

locale *shapiro-tauberian* =
  fixes $a :: nat \Rightarrow real$ **and** $A\ S\ T :: real \Rightarrow real$
  defines $A \equiv sum\text{-}upto\ (\lambda n.\ a\ n\ /\ n)$
  defines $S \equiv sum\text{-}upto\ a$
  defines $T \equiv (\lambda x.\ dirichlet\text{-}prod'\ a\ floor\ x)$
  assumes *a-nonneg*: $\bigwedge n.\ n > 0 \Longrightarrow a\ n \geq 0$
  assumes *a-asymptotics*: $(\lambda x.\ T\ x - x * ln\ x) \in O(\lambda x.\ x)$
begin

lemma *fin*: *finite* X **if** $X \subseteq \{n.\ real\ n \leq x\}$ **for** $X\ x$
  by (*rule finite-subset*[*of* - {..*nat* ⌊*x*⌋}]) (*use this* **in** ‹*auto simp*: *le-nat-iff le-floor-iff*›)

lemma *S-mono*: $S\ x \leq S\ y$ **if** $x \leq y$ **for** $x\ y$
  unfolding *S-def sum-upto-def* **using** *that* **by** (*intro sum-mono2 fin*[*of* - *y*] *a-nonneg*) *auto*

lemma *split*:
  fixes $f :: nat \Rightarrow real$
  assumes $\alpha \in \{0..1\}$
  shows *sum-upto f x* = *sum-upto f* (α∗*x*) + ($\sum n \mid n > 0 \wedge real\ n \in \{\alpha*x<..x\}.$ *f n*)
proof (*cases* $x > 0$)
  case *False*
  hence ∗: $\{n.\ n > 0 \wedge real\ n \leq x\} = \{\}$ $\{n.\ n > 0 \wedge real\ n \in \{\alpha*x<..x\}\} = \{\}$
    using *mult-right-mono*[*of* α *1 x*] *assms* **by** *auto*
  have $\alpha * x \leq 0$
    using *False assms* **by** (*intro mult-nonneg-nonpos*) *auto*
  hence ∗∗: $\{n.\ n > 0 \wedge real\ n \leq \alpha * x\} = \{\}$
    by *auto*
  show *?thesis*
    unfolding *sum-upto-def* ∗ ∗∗ **by** *auto*
next
  case *True*
  have *sum-upto f x* = ($\sum n \mid n > 0 \wedge real\ n \leq x.$ *f n*)
    by (*simp add*: *sum-upto-def*)
  also have $\{n.\ n > 0 \wedge real\ n \leq x\} =$
      $\{n.\ n > 0 \wedge real\ n \leq \alpha*x\} \cup \{n.\ n > 0 \wedge real\ n \in \{\alpha*x<..x\}\}$
    using *assms True mult-right-mono*[*of* α *1 x*] **by** (*force intro*: *order-trans*)
  also have ($\sum n \in \ldots.$ *f n*) = *sum-upto f* (α∗*x*) + ($\sum n \mid n > 0 \wedge real\ n \in \{\alpha*x<..x\}.$ *f n*)
    by (*subst sum.union-disjoint*) (*auto intro*: *fin simp*: *sum-upto-def*)
  finally show *?thesis* **.**
qed

lemma *S-diff-T-diff*: $S\ x - S\ (x\ /\ 2) \leq T\ x - 2 * T\ (x\ /\ 2)$
proof −
note *fin* = *fin*[*of* - *x*]
have *T-diff-eq*:
$T\ x - 2 * T\ (x\ /\ 2) = \textit{sum-upto}\ (\lambda n.\ a\ n * (\lfloor x\ /\ n \rfloor - 2 * \lfloor x\ /\ (2 * n) \rfloor))\ (x\ /\ 2) +$
$(\sum n \mid n > 0 \wedge \textit{real}\ n \in \{x/2<..x\}.\ a\ n * \lfloor x\ /\ n \rfloor)$
unfolding *T-def dirichlet-prod'-def*
by (*subst split*[**where** $\alpha = 1/2$])
(*simp-all add*: *sum-upto-def sum-subtractf ring-distribs*
sum-distrib-left sum-distrib-right mult-ac)

have $S\ x - S\ (x\ /\ 2) = (\sum n \mid n > 0 \wedge \textit{real}\ n \in \{x/2<..x\}.\ a\ n)$
unfolding *S-def* **by** (*subst split*[**where** $\alpha = 1\ /\ 2$]) (*auto simp*: *sum-upto-def*)
also have $\ldots = (\sum n \mid n > 0 \wedge \textit{real}\ n \in \{x/2<..x\}.\ a\ n * \lfloor x\ /\ n \rfloor)$
proof (*intro sum.cong*)
fix *n* **assume** $n \in \{n.\ n > 0 \wedge \textit{real}\ n \in \{x/2<..x\}\}$
hence $x\ /\ n \geq 1\ x\ /\ n < 2$ **by** (*auto simp*: *field-simps*)
hence $\lfloor x\ /\ n \rfloor = 1$ **by** *linarith*
thus $a\ n = a\ n * \lfloor x\ /\ n \rfloor$ **by** *simp*
qed *auto*
also have $\ldots = 0 + \ldots$ **by** *simp*
also have $0 \leq \textit{sum-upto}\ (\lambda n.\ a\ n * (\lfloor x\ /\ n \rfloor - 2 * \lfloor x\ /\ (2 * n) \rfloor))\ (x\ /\ 2)$
unfolding *sum-upto-def*
proof (*intro sum-nonneg mult-nonneg-nonneg a-nonneg*)
fix *n* **assume** $n \in \{n.\ n > 0 \wedge \textit{real}\ n \leq x\ /\ 2\}$
hence $x\ /\ \textit{real}\ n \geq 2$ **by** (*auto simp*: *field-simps*)
thus *real-of-int* $(\lfloor x\ /\ n \rfloor - 2 * \lfloor x\ /\ (2 * n) \rfloor) \geq 0$
using *le-mult-floor*[*of* $2\ x\ /\ (2 * n)$] **by** (*simp add*: *mult-ac*)
qed *auto*
also have $\ldots + (\sum n \mid n > 0 \wedge \textit{real}\ n \in \{x/2<..x\}.\ a\ n * \lfloor x\ /\ n \rfloor) = T\ x - 2 * T\ (x\ /\ 2)$
using *T-diff-eq* **..**
finally show $S\ x - S\ (x\ /\ 2) \leq T\ x - 2 * T\ (x\ /\ 2)$ **by** *simp*
qed

lemma
shows *diff-bound-strong*: $\exists\, c{\geq}0.\ \forall\, x{\geq}0.\ x * A\ x - T\ x \in \{0..c{*}x\}$
and *asymptotics*: $(\lambda x.\ A\ x - \ln x) \in O(\lambda\text{-}.\ 1)$
and *upper*: $\exists\, c{\geq}0.\ \forall\, x{\geq}0.\ S\ x \leq c * x$
and *lower*: $\exists\, c{>}0.\ \forall\, x{\geq}1/c.\ S\ x \geq c * x$
and *bigtheta*: $S \in \Theta(\lambda x.\ x)$
proof −
— We first prove the third case, i. e. the upper bound for *S*.
have $(\lambda x.\ S\ x - S\ (x\ /\ 2)) \in O(\lambda x.\ T\ x - 2 * T\ (x\ /\ 2))$
proof (*rule le-imp-bigo-real*)
show *eventually* $(\lambda x.\ S\ x - S\ (x\ /\ 2) \geq 0)$ *at-top*
using *eventually-ge-at-top*[*of* 0]

```
  proof eventually-elim
    case (elim x)
    thus ?case using S-mono[of x / 2 x] by simp
  qed
next
  show eventually (λx. S x − S (x / 2) ≤ 1 ∗ (T x − 2 ∗ T (x / 2))) at-top
    using S-diff-T-diff by simp
qed auto
also have (λx. T x − 2 ∗ T (x / 2)) ∈ O(λx. x)
proof −
  have (λx. T x − 2 ∗ T (x / 2)) =
        (λx. (T x − x ∗ ln x) − 2 ∗ (T (x / 2) − (x / 2) ∗ ln (x / 2))
          + x ∗ (ln x − ln (x / 2))) by (simp add: algebra-simps)
  also have ... ∈ O(λx. x)
  proof (rule sum-in-bigo, rule sum-in-bigo)
    show (λx. T x − x ∗ ln x) ∈ O(λx. x) by (rule a-asymptotics)
  next
    have (λx. T (x / 2) − (x / 2) ∗ ln (x / 2)) ∈ O(λx. x / 2)
      using a-asymptotics by (rule landau-o.big.compose) real-asymp+
    thus (λx. 2 ∗ (T (x / 2) − x / 2 ∗ ln (x / 2))) ∈ O(λx. x)
      unfolding cmult-in-bigo-iff by (subst (asm) landau-o.big.cdiv) auto
  qed real-asymp+
  finally show ?thesis .
qed
finally have S-diff-bigo: (λx. S x − S (x / 2)) ∈ O(λx. x) .

obtain c1 where c1: c1 ≥ 0 ⋀x. x ≥ 0 ⟹ S x ≤ c1 ∗ x
proof −
  from S-diff-bigo have (λn. S (real n) − S (real n / 2)) ∈ O(λn. real n)
    by (rule landau-o.big.compose) real-asymp
  from natfun-bigoE[OF this, of 1] obtain c
    where c > 0 ∀ n≥1. |S (real n) − S (real n / 2)| ≤ c ∗ real n by auto
  hence c: S (real n) − S (real n / 2) ≤ c ∗ real n if n ≥ 1 for n
    using S-mono[of real n 2 ∗ real n] that by auto
  have c-twopow: S (2 ^ Suc n / 2) − S (2 ^ n / 2) ≤ c ∗ 2 ^ n for n
    using c[of 2 ^ n] by simp

  have S-twopow-le: S (2 ^ k) ≤ 2 ∗ c ∗ 2 ^ k for k
  proof −
    have [simp]: {0<..Suc 0} = {1} by auto
    have (∑ r<Suc k. S (2 ^ Suc r / 2) − S (2 ^ r / 2)) ≤ (∑ r<Suc k. c ∗ 2
^ r)
      by (intro sum-mono c-twopow)
    also have (∑ r<Suc k. S (2 ^ Suc r / 2) − S (2 ^ r / 2)) = S (2 ^ k)
      by (subst sum-lessThan-telescope) (auto simp: S-def sum-upto-altdef)
    also have (∑ r<Suc k. c ∗ 2 ^ r) = c ∗ (∑ r<Suc k. 2 ^ r)
      unfolding sum-distrib-left ..
    also have (∑ r<Suc k. 2 ^ r :: real) = 2^Suc k − 1
      by (subst geometric-sum) auto
```

```
    also have c * ... ≤ c * 2 ^ Suc k
      using ‹c > 0› by (intro mult-left-mono) auto
    finally show S (2 ^ k) ≤ 2 * c * 2 ^ k by simp
  qed

  have S-le: S x ≤ 4 * c * x if x ≥ 0 for x
  proof (cases x ≥ 1)
    case False
    with that have x ∈ {0..<1} by auto
    thus ?thesis using ‹c > 0› by (auto simp: S-def sum-upto-altdef)
  next
    case True
    hence x: x ≥ 1 by simp
    define n where n = nat ⌊log 2 x⌋
    have 2 powr real n ≤ 2 powr (log 2 x)
      unfolding n-def using x by (intro powr-mono) auto
    hence ge: 2 ^ n ≤ x using x by (subst (asm) powr-realpow) auto

    have 2 powr real (Suc n) > 2 powr (log 2 x)
      unfolding n-def using x by (intro powr-less-mono) linarith+
    hence less: 2 ^ (Suc n) > x using x by (subst (asm) powr-realpow) auto

    have S x ≤ S (2 ^ Suc n)
      using less by (intro S-mono) auto
    also have ... ≤ 2 * c * 2 ^ Suc n
      by (intro S-twopow-le)
    also have ... = 4 * c * 2 ^ n
      by simp
    also have ... ≤ 4 * c * x
      by (intro mult-left-mono ge) (use x ‹c > 0› in auto)
    finally show S x ≤ 4 * c * x .
  qed

  with that[of 4 * c] and ‹c > 0› show ?thesis by auto
qed
thus ∃c≥0. ∀x≥0. S x ≤ c * x by auto

— The asymptotics of A follows from this immediately:
have a-strong: x * A x - T x ∈ {0..c1 * x} if x: x ≥ 0 for x
proof -
 have sum-upto (λn. a n * frac (x / n)) x ≤ sum-upto (λn. a n * 1) x unfolding
sum-upto-def
   by (intro sum-mono mult-left-mono a-nonneg) (auto intro: less-imp-le frac-lt-1)
  also have ... = S x unfolding S-def by simp
  also from x have ... ≤ c1 * x by (rule c1)
  finally have sum-upto (λn. a n * frac (x / n)) x ≤ c1 * x .
  moreover have sum-upto (λn. a n * frac (x / n)) x ≥ 0
   unfolding sum-upto-def by (intro sum-nonneg mult-nonneg-nonneg a-nonneg)
auto
```

```
    ultimately have sum-upto (λn. a n * frac (x / n)) x ∈ {0..c1*x} by auto
    also have sum-upto (λn. a n * frac (x / n)) x = x * A x - T x
      by (simp add: T-def A-def sum-upto-def sum-subtractf frac-def algebra-simps
                    sum-distrib-left sum-distrib-right dirichlet-prod'-def)
    finally show ?thesis .
  qed
  thus ∃ c≥0. ∀ x≥0. x * A x - T x ∈ {0..c*x}
    using ‹c1 ≥ 0› by (intro exI[of - c1]) auto
  hence (λx. x * A x - T x) ∈ O(λx. x)
    using a-strong ‹c1 ≥ 0›
   by (intro le-imp-bigo-real[of c1] eventually-mono[OF eventually-ge-at-top[of 1]])
auto
  from this and a-asymptotics have (λx. (x * A x - T x) + (T x - x * ln x)) ∈
O(λx. x)
    by (rule sum-in-bigo)
  hence (λx. x * (A x - ln x)) ∈ O(λx. x * 1)
    by (simp add: algebra-simps)
  thus bigo: (λx. A x - ln x) ∈ O(λx. 1)
    by (subst (asm) landau-o.big.mult-cancel-left) auto

  — It remains to show the lower bound for S.
  define R where R = (λx. A x - ln x)
  obtain M where M: ⋀x. x ≥ 1 ⟹ |R x| ≤ M
  proof -
    have (λn. R (real n)) ∈ O(λ-. 1)
      using bigo unfolding R-def by (rule landau-o.big.compose) real-asymp
    from natfun-bigoE[OF this, of 0] obtain M where M: M > 0 ⋀n. |R (real
n)| ≤ M
      by auto

    have |R x| ≤ M + ln 2 if x: x ≥ 1 for x
    proof -
      define n where n = nat ⌊x⌋
      have |R x - R (real n)| = ln (x / n)
        using x by (simp add: R-def A-def sum-upto-altdef n-def ln-divide-pos)
      also {
        have x ≤ real n + 1
          unfolding n-def by linarith
        also have 1 ≤ real n
          using x unfolding n-def by simp
        finally have ln (x / n) ≤ ln 2
          using x by (simp add: field-simps)
      }
      finally have |R x| ≤ |R (real n)| + ln 2
        by linarith
      also have |R (real n)| ≤ M
        by (rule M)
      finally show |R x| ≤ M + ln 2 by simp
    qed
```

```
  with that[of M + ln 2] show ?thesis by blast
qed
have M ≥ 0 using M[of 1] by simp

have A-diff-ge: A x − A (α∗x) ≥ −ln α − 2 ∗ M
  if α: α ∈ {0<..<1} and x ≥ 1 / α for x α :: real
proof −
  from that have 1 < inverse α ∗ 1 by (simp add: field-simps)
  also have ... ≤ inverse α ∗ (α ∗ x)
    using ‹x ≥ 1 / α› and α by (intro mult-left-mono) (auto simp: field-simps)
  also from α have ... = x by simp
  finally have x > 1 .
  note x = this ‹x >= 1 / α›

  have −ln α − M − M ≤ −ln α − |R x| − |R (α∗x)|
    using x α by (intro diff-mono M) (auto simp: field-simps)
  also have ... ≤ −ln α + R x − R (α∗x)
    by linarith
  also have ... = A x − A (α∗x)
    using α x by (simp add: R-def ln-mult)
  finally show A x − A (α∗x) ≥ −ln α − 2 ∗ M by simp
qed

define α where α = exp (−2∗M−1)
have α ∈ {0<..<1}
  using ‹M ≥ 0› by (auto simp: α-def)

have S-ge: S x ≥ α ∗ x if x: x ≥ 1 / α for x
proof −
  have 1 = −ln α − 2 ∗ M
    by (simp add: α-def)
  also have ... ≤ A x − A (α∗x)
    by (intro A-diff-ge) fact+
  also have ... = (∑ n | n > 0 ∧ real n ∈ {α∗x<..x}. a n / n)
    unfolding A-def using ‹α ∈ {0<..<1}› by (subst split[where α = α]) auto
  also have ... ≤ (∑ n | n > 0 ∧ real n ∈ {α∗x<..x}. a n / (α∗x))
   using x ‹α ∈ {0<..<1}› by (intro sum-mono divide-left-mono a-nonneg) auto
  also have ... = (∑ n | n > 0 ∧ real n ∈ {α∗x<..x}. a n) / (α∗x)
    by (simp add: sum-divide-distrib)
  also have ... ≤ S x / (α∗x)
    using x ‹α ∈ {0<..<1}› unfolding S-def sum-upto-def
    by (intro divide-right-mono sum-mono2 a-nonneg) (auto simp: field-simps)
  finally show S x ≥ α ∗ x
    using ‹α ∈ {0<..<1}› x by (simp add: field-simps)
qed
thus ∃ c>0. ∀ x≥1/c. S x ≥ c ∗ x
  using ‹α ∈ {0<..<1}› by (intro exI[of - α]) auto

have S-nonneg: S x ≥ 0 for x
```

unfolding *S-def sum-upto-def* **by** (*intro sum-nonneg a-nonneg*) *auto*
have *eventually* (λ*x*. |*S x*| ≥ α ∗ |*x*|) *at-top*
using *eventually-ge-at-top*[*of max 0* (*1* / α)]
proof *eventually-elim*
case (*elim x*)
with *S-ge*[*of x*] *elim* **show** *?case* **by** *auto*
qed
hence *S* ∈ Ω(λ*x*. *x*)
using ‹α ∈ {*0*<..<*1*}› **by** (*intro landau-omega.bigI*[*of* α]) *auto*
moreover have *S* ∈ *O*(λ*x*. *x*)
proof (*intro bigoI eventually-mono*[*OF eventually-ge-at-top*[*of 0*]])
fix *x* :: *real* **assume** *x* ≥ *0*
thus *norm* (*S x*) ≤ *c1* ∗ *norm x*
using *c1*(*2*)[*of x*] **by** (*auto simp*: *S-nonneg*)
qed
ultimately show *S* ∈ Θ(λ*x*. *x*)
by (*intro bigthetaI*)
qed

end

6.2 Applications to the Chebyshev functions

We can now apply Shapiro's Tauberian theorem to ψ and ϑ.

lemma *dirichlet-prod-mangoldt1-floor-bigo*:
includes *prime-counting-syntax*
shows (λ*x*. *dirichlet-prod′* (λ*n*. *ind prime n* ∗ *ln n*) *floor x* − *x* ∗ *ln x*) ∈ *O*(λ*x*. *x*)
proof −
— This is a perhaps somewhat roundabout way of proving this statement. We show this using the asymptotics of $\mathfrak{M}$: $\mathfrak{M}(x) = \ln x + O(1)$
We proved this before (which was a bit of work, but not that much). Apostol, on the other hand, shows the following statement first and then deduces the asymptotics of $\mathfrak{M}$ with Shapiro's Tauberian theorem instead. This might save a bit of work, but it is probably negligible.
define *R* **where** *R* = (λ*x*. *sum-upto* (λ*i*. *ind prime i* ∗ *ln i* ∗ *frac* (*x* / *i*)) *x*)
have ∗: *R x* ∈ {*0*..*ln 4* ∗ *x*} **if** *x* ≥ *1* **for** *x*
proof −
have *R x* ≤ ϑ *x*
unfolding *R-def prime-sum-upto-altdef1 sum-upto-def* ϑ*-def*
by (*intro sum-mono*) (*auto simp*: *ind-def less-imp-le*[*OF frac-lt-1*] *dest*!: *prime-gt-1-nat*)
also have ... < *ln 4* ∗ *x*
by (*rule* ϑ*-upper-bound*) *fact*+
finally have *R x* ≤ *ln 4* ∗ *x* **by** *auto*
moreover have *R x* ≥ *0* **unfolding** *R-def sum-upto-def*
by (*intro sum-nonneg mult-nonneg-nonneg*) (*auto simp*: *ind-def*)
ultimately show *?thesis* **by** *auto*
qed

have *eventually* $(\lambda x.\ |R\ x| \leq ln\ 4 * |x|)$ *at-top*
using *eventually-ge-at-top*[*of 1*] **by** *eventually-elim* (*use* $*$ **in** *auto*)
hence $R \in O(\lambda x.\ x)$ **by** (*intro landau-o.bigI*[*of ln 4*]) *auto*

have $(\lambda x.\ dirichlet\text{-}prod'\ (\lambda n.\ ind\ prime\ n * ln\ n)\ floor\ x - x * ln\ x) =$
$(\lambda x.\ x * (\mathfrak{M}\ x - ln\ x) - R\ x)$
by (*auto simp*: *primes-M-def dirichlet-prod'-def prime-sum-upto-altdef1 sum-upto-def frac-def sum-subtractf sum-distrib-left sum-distrib-right algebra-simps R-def*)
also have $\ldots \in O(\lambda x.\ x)$
proof (*rule sum-in-bigo*)
have $(\lambda x.\ x * (\mathfrak{M}\ x - ln\ x)) \in O(\lambda x.\ x * 1)$
by (*intro landau-o.big.mult mertens-bounded*) *auto*
thus $(\lambda x.\ x * (\mathfrak{M}\ x - ln\ x)) \in O(\lambda x.\ x)$ **by** *simp*
qed *fact*+
finally show *?thesis* **.**
qed

lemma *dirichlet-prod'-mangoldt-floor-asymptotics*:
$(\lambda x.\ dirichlet\text{-}prod'\ mangoldt\ floor\ x - x * ln\ x + x) \in O(ln)$
proof −
have *dirichlet-prod' mangoldt floor* $= (\lambda x.\ sum\text{-}upto\ ln\ x)$
unfolding *sum-upto-ln-conv-sum-upto-mangoldt dirichlet-prod'-def*
by (*intro sum-upto-cong' ext*) *auto*
hence $(\lambda x.\ dirichlet\text{-}prod'\ mangoldt\ floor\ x - x * ln\ x + x) = (\lambda x.\ sum\text{-}upto\ ln\ x - x * ln\ x + x)$
by *simp*
also have $\ldots \in O(ln)$
by (*rule sum-upto-ln-stirling-weak-bigo*)
finally show $(\lambda x.\ dirichlet\text{-}prod'\ mangoldt\ (\lambda x.\ real\text{-}of\text{-}int\ \lfloor x \rfloor)\ x - x * ln\ x + x) \in O(ln)$ **.**
qed

interpretation ψ: *shapiro-tauberian mangoldt sum-upto* $(\lambda n.\ mangoldt\ n\ /\ n)$ *primes-psi*
dirichlet-prod' mangoldt floor
proof *unfold-locales*
have *dirichlet-prod' mangoldt floor* $= (\lambda x.\ sum\text{-}upto\ ln\ x)$
unfolding *sum-upto-ln-conv-sum-upto-mangoldt dirichlet-prod'-def*
by (*intro sum-upto-cong' ext*) *auto*
hence $(\lambda x.\ dirichlet\text{-}prod'\ mangoldt\ floor\ x - x * ln\ x + x) = (\lambda x.\ sum\text{-}upto\ ln\ x - x * ln\ x + x)$
by *simp*
also have $\ldots \in O(ln)$
by (*rule sum-upto-ln-stirling-weak-bigo*)
also have $ln \in O(\lambda x{::}real.\ x)$ **by** *real-asymp*
finally have $(\lambda x.\ dirichlet\text{-}prod'\ mangoldt\ (\lambda x.\ real\text{-}of\text{-}int\ \lfloor x \rfloor)\ x - x * ln\ x + x - x)$

$\in O(\lambda x.\ x)$ **by** (*rule sum-in-bigo*) *auto*
thus $(\lambda x.\ dirichlet\text{-}prod'\ mangoldt\ (\lambda x.\ real\text{-}of\text{-}int\ \lfloor x \rfloor)\ x - x * ln\ x) \in O(\lambda x.\ x)$
by *simp*
qed (*simp-all add*: *primes-psi-def mangoldt-nonneg*)

thm *ψ.asymptotics ψ.upper ψ.lower*

interpretation *ϑ*: *shapiro-tauberian* $\lambda n.\ ind\ prime\ n * ln\ n$
sum-upto $(\lambda n.\ ind\ prime\ n * ln\ n\ /\ n)$ *primes-theta dirichlet-prod'* $(\lambda n.\ ind\ prime\ n * ln\ n)$ *floor*
proof *unfold-locales*
fix *n* :: *nat* **show** $ind\ prime\ n * ln\ n \geq 0$
by (*auto simp*: *ind-def dest*: *prime-gt-1-nat*)
next
show $(\lambda x.\ dirichlet\text{-}prod'\ (\lambda n.\ ind\ prime\ n * ln\ n)\ floor\ x - x * ln\ x) \in O(\lambda x.\ x)$
by (*rule dirichlet-prod-mangoldt1-floor-bigo*)
qed (*simp-all add*: *primes-theta-def mangoldt-nonneg prime-sum-upto-altdef1* [*abs-def*])

thm *ϑ.asymptotics ϑ.upper ϑ.lower*

lemma *sum-upto-ψ-x-over-n-asymptotics*:
$(\lambda x.\ sum\text{-}upto\ (\lambda n.\ primes\text{-}psi\ (x\ /\ n))\ x - x * ln\ x + x) \in O(ln)$
and *sum-upto-ϑ-x-over-n-asymptotics*:
$(\lambda x.\ sum\text{-}upto\ (\lambda n.\ primes\text{-}theta\ (x\ /\ n))\ x - x * ln\ x) \in O(\lambda x.\ x)$
using *dirichlet-prod-mangoldt1-floor-bigo dirichlet-prod'-mangoldt-floor-asymptotics*
by (*simp-all add*: *dirichlet-prod'-floor-conv-sum-upto primes-theta-def*
primes-psi-def prime-sum-upto-altdef1)

end

7 Bounds on partial sums of the ζ function

theory *Partial-Zeta-Bounds*
imports
Euler-MacLaurin.Euler-MacLaurin-Landau
Zeta-Function.Zeta-Function
Prime-Number-Theorem.Prime-Number-Theorem-Library
Prime-Distribution-Elementary-Library
begin

We employ Euler–MacLaurin's summation formula to obtain asymptotic estimates for the partial sums of the Riemann $\zeta(s)$ function for fixed real a,

i. e. the function

$$f(n) = \sum_{k=1}^{n} k^{-s} \ .$$

We distinguish various cases. The case $s = 1$ is simply the Harmonic numbers and is treated apart from the others.

```
lemma harm-asymp-equiv: sum-upto (λn. 1 / n) ∼[at-top] ln
proof −
  have sum-upto (λn. n powr −1) ∼[at-top] (λx. ln (⌊x⌋ + 1))
  proof (rule asymp-equiv-sandwich)
    have eventually (λx. sum-upto (λn. n powr −1) x ∈ {ln (⌊x⌋ + 1)..ln ⌊x⌋ +
1}) at-top
      using eventually-ge-at-top[of 1]
    proof eventually-elim
      case (elim x)
      have sum-upto (λn. real n powr −1) x = harm (nat ⌊x⌋)
          unfolding sum-upto-altdef harm-def by (intro sum.cong) (auto simp:
field-simps powr-minus)
      also have ... ∈ {ln (⌊x⌋ + 1)..ln ⌊x⌋ + 1}
        using elim harm-le[of nat ⌊x⌋] ln-le-harm[of nat ⌊x⌋]
        by (auto simp: le-nat-iff le-floor-iff)
      finally show ?case by simp
    qed
    thus eventually (λx. sum-upto (λn. n powr −1) x ≥ ln (⌊x⌋ + 1)) at-top
        eventually (λx. sum-upto (λn. n powr −1) x ≤ ln ⌊x⌋ + 1) at-top
      by (eventually-elim; simp)+
  qed real-asymp+
  also have ... ∼[at-top] ln by real-asymp
  finally show ?thesis by (simp add: powr-minus field-simps)
qed

lemma
  fixes s :: real
  assumes s: s > 0 s ≠ 1
  shows   zeta-partial-sum-bigo-pos:
          (λn. (∑ k=1..n. real k powr −s) − real n powr (1 − s) / (1 − s) − Re
(zeta s))
              ∈ O(λx. real x powr −s)
    and   zeta-partial-sum-bigo-pos′:
          (λn. ∑ k=1..n. real k powr −s) =o
              (λn. real n powr (1 − s) / (1 − s) + Re (zeta s)) +o O(λx. real x
powr −s)
proof −
  define F where F = (λx. x powr (1 − s) / (1 − s))
  define f where f = (λx. x powr −s)
  define f′ where f′ = (λx. −s ∗ x powr (−s−1))
  define z where z = Re (zeta s)

  interpret euler-maclaurin-nat′ F f (!) [f, f′] 1 0 z {}
```

```
  proof
    have (λb. (∑ k=1..b. real k powr −s) − real b powr (1 − s) / (1 − s) − real
b powr −s / 2)
            ⟶ Re (zeta s) − 0
    proof (intro tendsto-diff)
      let ?g = λb. (∑ i<b. complex-of-real (real i + 1) powr − complex-of-real s)
−
                          of-nat b powr (1 − complex-of-real s) / (1 − complex-of-real
s)
      have ∀F b in at-top. Re (?g b) = (∑ k=1..b. real k powr −s) − real b powr
(1 − s) / (1 − s)
        using eventually-ge-at-top[of 1]
      proof eventually-elim
        case (elim b)
        have (∑ k=1..b. real k powr −s) = (∑ k<b. real (Suc k) powr −s)
          by (intro sum.reindex-bij-witness[of - Suc λn. n − 1]) auto
        also have ... − real b powr (1 − s) / (1 − s) = Re (?g b)
          by (auto simp: powr-Reals-eq add-ac)
        finally show ?case ..
      qed
      moreover have (λb. Re (?g b)) ⟶ Re (zeta s)
        using hurwitz-zeta-critical-strip[of of-real s 1] s
        by (intro tendsto-intros) (simp add: zeta-def)
      ultimately show (λb. (∑ k=1..b. real k powr −s) − real b powr (1 − s) /
(1 − s)) ⟶ Re (zeta s)
        by (blast intro: Lim-transform-eventually)
    qed (use s in real-asymp)
    thus (λb. (∑ k = 1..b. f (real k)) − F (real b) −
            (∑ i<2 * 0 + 1. (bernoulli' (Suc i) / fact (Suc i)) *R ([f, f'] ! i) (real
b)))
          ⟶ z by (simp add: f-def F-def z-def)
  qed (use ‹s ≠ 1› in
        ‹auto intro!: derivative-eq-intros continuous-intros
            simp flip: has-real-derivative-iff-has-vector-derivative
            simp: F-def f-def f'-def nth-Cons split: nat.splits›)

  {
    fix n :: nat assume n: n ≥ 1
    have (∑ k=1..n. real k powr −s) =
          n powr (1 − s) / (1 − s) + z + 1/2 * n powr −s −
          EM-remainder 1 f' (int n)
      using euler-maclaurin-strong-nat'[of n] n by (simp add: F-def f-def)
  } note * = this

  have (λn. (∑ k=1..n. real k powr −s) − n powr (1 − s) / (1 − s) − z) ∈
        Θ(λn. 1/2 * n powr −s − EM-remainder 1 f' (int n))
    using * by (intro bigthetaI-cong eventually-mono[OF eventually-ge-at-top[of
1]]) auto
  also have (λn. 1/2 * n powr −s − EM-remainder 1 f' (int n)) ∈ O(λn. n powr
```

```
−s)
  proof (intro sum-in-bigo)
    have (λx. norm (EM-remainder 1 f' (int x))) ∈ O(λx. real x powr − s)
    proof (rule EM-remainder-strong-bigo-nat[where a = 1 and Y = {}])
      fix x :: real assume x ≥ 1
      show norm (f' x) ≤ s * x powr (−s−1)
        using s by (simp add: f'-def)
    next
      from s show ((λx. x powr −s) ⟶ 0) at-top by real-asymp
    qed (auto simp: f'-def intro!: continuous-intros derivative-eq-intros)
    thus (λx. EM-remainder 1 f' (int x)) ∈ O(λx. real x powr −s)
      by simp
  qed real-asymp+
  finally show (λn. (∑ k=1..n. real k powr −s) − real n powr (1 − s) / (1 − s)
− z)
                 ∈ O(λx. real x powr −s) .
  thus(λn. ∑ k=1..n. real k powr −s) =o
         (λn. real n powr (1 − s) / (1 − s) + z) +o O(λx. real x powr −s)
    by (subst set-minus-plus [symmetric]) (simp-all add: fun-diff-def algebra-simps)
qed

lemma zeta-tail-bigo:
  fixes s :: real
  assumes s: s > 1
  shows   (λn. Re (hurwitz-zeta (real n + 1) s)) ∈ O(λx. real x powr (1 − s))
proof −
  have [simp]: complex-of-real (Re (zeta s)) = zeta s
    using zeta-real[of s] by (auto elim!: Reals-cases)

  from s have s': s > 0 s ≠ 1 by auto
  have (λn. −Re (hurwitz-zeta (real n + 1) s) − real n powr (1 − s) / (1 − s)
            + real n powr (1 − s) / (1 − s))
         ∈ O(λx. real x powr (1 − s))
  proof (rule sum-in-bigo)
    have (λn. −Re (hurwitz-zeta (real n + 1) s) − real n powr (1 − s) / (1 − s))
=
           (λn. (∑ k=1..n. real k powr −s) − real n powr (1 − s) / (1 − s) − Re
(zeta s))
      (is ?lhs = ?rhs)
    proof
      fix n :: nat
      have hurwitz-zeta (1 + real n) s = zeta s − (∑ k<n. real (Suc k) powr −s)
      by (subst hurwitz-zeta-shift) (use assms in ‹auto simp: zeta-def powr-Reals-eq›)
      also have (∑ k<n. real (Suc k) powr −s) = (∑ k=1..n. real k powr −s)
        by (rule sum.reindex-bij-witness[of - λk. k − 1 Suc]) auto
      finally show ?lhs n = ?rhs n by (simp add: add-ac)
    qed
    also have ... ∈ O(λx. real x powr (−s))
      by (rule zeta-partial-sum-bigo-pos) (use s in auto)
```

```
  also have (λx. real x powr (−s)) ∈ O(λx. real x powr (1−s))
    by real-asymp
  finally show (λn. −Re (hurwitz-zeta (real n + 1) s) − real n powr (1 − s) /
(1 − s)) ∈ ... .
 qed (use s in real-asymp)
 thus ?thesis by simp
qed

lemma zeta-tail-bigo′:
 fixes s :: real
 assumes s: s > 1
 shows   (λn. Re (hurwitz-zeta (real n) s)) ∈ O(λx. real x powr (1 − s))
proof −
 have (λn. Re (hurwitz-zeta (real n) s)) ∈ Θ(λn. Re (hurwitz-zeta (real (n − 1)
+ 1) s))
   by (intro bigthetaI-cong eventually-mono[OF eventually-ge-at-top[of 1]])
     (auto simp: of-nat-diff)
 also have (λn. Re (hurwitz-zeta (real (n − 1) + 1) s)) ∈ O(λx. real (x − 1)
powr (1 − s))
   by (rule landau-o.big.compose[OF zeta-tail-bigo[OF assms]]) real-asymp
 also have (λx. real (x − 1) powr (1 − s)) ∈ O(λx. real x powr (1 − s))
   by real-asymp
 finally show ?thesis .
qed

lemma
 fixes s :: real
 assumes s: s > 0
 shows   zeta-partial-sum-bigo-neg:
         (λn. (∑ i=1..n. real i powr s) − n powr (1 + s) / (1 + s)) ∈ O(λn. n
powr s)
   and   zeta-partial-sum-bigo-neg′:
         (λn. (∑ i=1..n. real i powr s)) =o (λn. n powr (1 + s) / (1 + s)) +o
O(λn. n powr s)
proof −
 define F where F = (λx. x powr (1 + s) / (1 + s))
 define f where f = (λx. x powr s)
 define f′ where f′ = (λx. s ∗ x powr (s − 1))

 have (∑ i=1..n. f (real i)) − F n =
       1 / 2 − F 1 + f n / 2 + EM-remainder′ 1 f′ 1 (real n) if n: n ≥ 1 for n
 proof −
   have (∑ i∈{1<..n}. f (real i)) − integral {real 1..real n} f =
         (∑ k<1. (bernoulli′ (Suc k) / fact (Suc k)) ∗R (([f, f′] ! k) (real n) −
           ([f, f′] ! k) (real 1))) + EM-remainder′ 1 ([f, f′] ! 1) (real 1) (real n)
     by (rule euler-maclaurin-strong-raw-nat[where Y = {}])
       (use ‹s > 0› ‹n ≥ 1› in
         ‹auto intro!: derivative-eq-intros continuous-intros
              simp flip: has-real-derivative-iff-has-vector-derivative
```

```
        simp: F-def f-def f'-def nth-Cons split: nat.splits›)
  also have (∑ i∈{1<..n}. f (real i)) = (∑ i∈insert 1 {1<..n}. f (real i)) − f
1
    using n by (subst sum.insert) auto
  also from n have insert 1 {1<..n} = {1..n} by auto
  finally have (∑ i=1..n. f (real i)) − F n = f 1 + (integral {1..real n} f − F
n) +
                (f (real n) − f 1) / 2 + EM-remainder' 1 f' 1 (real n) by simp
  hence (∑ i=1..n. f (real i)) − F n = 1 / 2 + (integral {1..real n} f − F n)
+
          f (real n) / 2 + EM-remainder' 1 f' 1 (real n)
    using s by (simp add: f-def diff-divide-distrib)
  also have (f has-integral (F (real n) − F 1)) {1..real n} using assms n
    by (intro fundamental-theorem-of-calculus)
      (auto simp flip: has-real-derivative-iff-has-vector-derivative simp: F-def f-def
          intro!: derivative-eq-intros continuous-intros)
  hence integral {1..real n} f − F n = − F 1
    by (simp add: has-integral-iff)
  also have 1 / 2 + (−F 1) + f (real n) / 2 = 1 / 2 − F 1 + f n / 2
    by simp
  finally show ?thesis .
 qed

 hence (λn. (∑ i=1..n. f (real i)) − F n) ∈
       Θ(λn. 1 / 2 − F 1 + f n / 2 + EM-remainder' 1 f' 1 (real n))
   by (intro bigthetaI-cong eventually-mono[OF eventually-ge-at-top[of 1]])
 also have (λn. 1 / 2 − F 1 + f n / 2 + EM-remainder' 1 f' 1 (real n))
           ∈ O(λn. real n powr s)
   unfolding F-def f-def
 proof (intro sum-in-bigo)
   have (λx. integral {1..real x} (λt. pbernpoly 1 t *R f' t)) ∈ O(λn. 1 / s * real
n powr s)
   proof (intro landau-o.big.compose[OF integral-bigo])
     have (λx. pbernpoly 1 x * f' x) ∈ O(λx. 1 * x powr (s − 1))
       by (intro landau-o.big.mult pbernpoly-bigo) (auto simp: f'-def)
     thus (λx. pbernpoly 1 x *R f' x) ∈ O(λx. x powr (s − 1))
       by simp
     from s show filterlim (λa. 1 / s * a powr s) at-top at-top by real-asymp
   next
     fix a' x :: real assume a' ≥ 1 a' ≤ x
     thus (λa. pbernpoly 1 a *R f' a) integrable-on {a'..x}
       by (intro integrable-EM-remainder') (auto intro!: continuous-intros simp:
f'-def)
    qed (use s in ‹auto intro!: filterlim-real-sequentially continuous-intros deriva-
tive-eq-intros›)
   thus (λx. EM-remainder' 1 f' 1 (real x)) ∈ O(λn. real n powr s)
     using ‹s > 0› by (simp add: EM-remainder'-def)
 qed (use ‹s > 0› in real-asymp)+
 finally show (λn. (∑ i=1..n. real i powr s) − n powr (1 + s) / (1 + s)) ∈
```

$O(\lambda n.\ n\ powr\ s)$
by (*simp add*: *f-def F-def*)
thus $(\lambda n.\ (\sum i{=}1..n.\ real\ i\ powr\ s)) =o (\lambda n.\ n\ powr\ (1\ +\ s)\ /\ (1\ +\ s)) +o O(\lambda n.\ n\ powr\ s)$
by (*subst set-minus-plus* [*symmetric*]) (*simp-all add*: *fun-diff-def algebra-simps*)
qed

lemma *zeta-partial-sum-le-pos*:
assumes $s > 0\ s \neq 1$
defines $z \equiv Re\ (zeta\ (complex\text{-}of\text{-}real\ s))$
shows $\exists c{>}0.\ \forall x{\geq}1.\ |sum\text{-}upto\ (\lambda n.\ n\ powr\ {-}s)\ x - (x\ powr\ (1{-}s)\ /\ (1{-}s) + z)| \leq c * x\ powr\ {-}s$
proof (*rule sum-upto-asymptotics-lift-nat-real*)
show $(\lambda n.\ (\sum k = 1..n.\ real\ k\ powr\ -\ s) - (real\ n\ powr\ (1\ -\ s)\ /\ (1\ -\ s) + z))$
$\in O(\lambda n.\ real\ n\ powr\ -\ s)$
using *zeta-partial-sum-bigo-pos*[*OF assms*(*1*,*2*)] **unfolding** *z-def*
by (*simp add*: *algebra-simps*)

from *assms* **have** $s < 1 \vee s > 1$ **by** *linarith*
thus $(\lambda n.\ real\ n\ powr\ (1\ -\ s)\ /\ (1\ -\ s) + z - (real\ (Suc\ n)\ powr\ (1\ -\ s)\ /\ (1\ -\ s) + z))$
$\in O(\lambda n.\ real\ n\ powr\ -\ s)$
by *standard* (*use* ‹$s > 0$› **in** ‹*real-asymp*+›)
show $(\lambda n.\ real\ n\ powr\ -\ s) \in O(\lambda n.\ real\ (Suc\ n)\ powr\ -\ s)$
by *real-asymp*
show *mono-on* $\{1..\}$ $(\lambda a.\ a\ powr\ -\ s) \vee$ *mono-on* $\{1..\}$ $(\lambda x.\ -\ (x\ powr\ -\ s))$
using *assms* **by** (*intro disjI2*) (*auto intro*!: *mono-onI powr-mono2'*)

from *assms* **have** $s < 1 \vee s > 1$ **by** *linarith*
hence *mono-on* $\{1..\}$ $(\lambda a.\ a\ powr\ (1\ -\ s)\ /\ (1\ -\ s) + z)$
proof
assume $s < 1$
thus *?thesis* **using** ‹$s > 0$›
by (*intro mono-onI powr-mono2 divide-right-mono add-right-mono*) *auto*
next
assume $s > 1$
thus *?thesis*
by (*intro mono-onI le-imp-neg-le add-right-mono divide-right-mono-neg powr-mono2'*) *auto*
qed
thus *mono-on* $\{1..\}$ $(\lambda a.\ a\ powr\ (1\ -\ s)\ /\ (1\ -\ s) + z) \vee$
mono-on $\{1..\}$ $(\lambda x.\ -\ (x\ powr\ (1\ -\ s)\ /\ (1\ -\ s) + z))$ **by** *blast*
qed *auto*

lemma *zeta-partial-sum-le-pos'*:
assumes $s > 0\ s \neq 1$
defines $z \equiv Re\ (zeta\ (complex\text{-}of\text{-}real\ s))$
shows $\exists c{>}0.\ \forall x{\geq}1.\ |sum\text{-}upto\ (\lambda n.\ n\ powr\ {-}s)\ x - x\ powr\ (1{-}s)\ /\ (1{-}s)|$

$\leq c$
proof −
have $\exists\, c{>}0.\ \forall\, x{\geq}1.\ |\mathit{sum\text{-}upto}\ (\lambda n.\ n\ \mathit{powr}\ {-s})\ x - x\ \mathit{powr}\ (1{-s})\ /\ (1{-s})| \leq c * 1$
proof (*rule sum-upto-asymptotics-lift-nat-real*)
have $(\lambda n.\ (\sum k = 1..n.\ \mathit{real}\ k\ \mathit{powr} - s) - (\mathit{real}\ n\ \mathit{powr}\ (1 - s)\ /\ (1 - s) + z))$
$\in O(\lambda n.\ \mathit{real}\ n\ \mathit{powr} - s)$
using *zeta-partial-sum-bigo-pos*[*OF assms*(*1*,*2*)] **unfolding** *z-def*
by (*simp add*: *algebra-simps*)
also have $(\lambda n.\ \mathit{real}\ n\ \mathit{powr}\ {-s}) \in O(\lambda n.\ 1)$
using *assms* **by** *real-asymp*
finally have $(\lambda n.\ (\sum k = 1..n.\ \mathit{real}\ k\ \mathit{powr} - s) - \mathit{real}\ n\ \mathit{powr}\ (1 - s)\ /\ (1 - s) - z)$
$\in O(\lambda n.\ 1)$ **by** (*simp add*: *algebra-simps*)
hence $(\lambda n.\ (\sum k = 1..n.\ \mathit{real}\ k\ \mathit{powr} - s) - \mathit{real}\ n\ \mathit{powr}\ (1 - s)\ /\ (1 - s) - z + z) \in O(\lambda n.\ 1)$
by (*rule sum-in-bigo*) *auto*
thus $(\lambda n.\ (\sum k = 1..n.\ \mathit{real}\ k\ \mathit{powr} - s) - (\mathit{real}\ n\ \mathit{powr}\ (1 - s)\ /\ (1 - s))) \in O(\lambda n.\ 1)$
by *simp*

from *assms* **have** $s < 1 \vee s > 1$ **by** *linarith*
thus $(\lambda n.\ \mathit{real}\ n\ \mathit{powr}\ (1 - s)\ /\ (1 - s) - (\mathit{real}\ (\mathit{Suc}\ n)\ \mathit{powr}\ (1 - s)\ /\ (1 - s))) \in O(\lambda n.\ 1)$
by *standard* (*use* ‹$s > 0$› **in** ‹*real-asymp*+›)
show *mono-on* $\{1..\}$ $(\lambda a.\ 1) \vee$ *mono-on* $\{1..\}$ $(\lambda x{::}\mathit{real}.\ {-1} :: \mathit{real})$
using *assms* **by** (*intro disjI2*) (*auto intro*!: *mono-onI powr-mono2'*)

from *assms* **have** $s < 1 \vee s > 1$ **by** *linarith*
hence *mono-on* $\{1..\}$ $(\lambda a.\ a\ \mathit{powr}\ (1 - s)\ /\ (1 - s))$
proof
assume $s < 1$
thus *?thesis* **using** ‹$s > 0$›
by (*intro mono-onI powr-mono2 divide-right-mono add-right-mono*) *auto*
next
assume $s > 1$
thus *?thesis*
by (*intro mono-onI le-imp-neg-le add-right-mono divide-right-mono-neg powr-mono2'*) *auto*
qed
thus *mono-on* $\{1..\}$ $(\lambda a.\ a\ \mathit{powr}\ (1 - s)\ /\ (1 - s)) \vee$
mono-on $\{1..\}$ $(\lambda x.\ -(x\ \mathit{powr}\ (1 - s)\ /\ (1 - s)))$ **by** *blast*
qed *auto*
thus *?thesis* **by** *simp*
qed

lemma *zeta-partial-sum-le-pos''*:
assumes $s > 0$ $s \neq 1$

shows $\exists\, c{>}0.\ \forall\, x{\geq}1.\ |\mathit{sum\text{-}upto}\ (\lambda n.\ n\ \mathit{powr}\ {-s})\ x| \leq c * x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s)$
proof −
from *zeta-partial-sum-le-pos'*[*OF assms*] **obtain** *c* **where**
c: $c > 0\ \bigwedge x.\ x \geq 1 \Longrightarrow |\mathit{sum\text{-}upto}\ (\lambda x.\ \mathit{real}\ x\ \mathit{powr} - s)\ x - x\ \mathit{powr}\ (1 - s) / (1 - s)| \leq c$
by *auto*

{
fix *x* :: *real* **assume** *x*: $x \geq 1$
have $|\mathit{sum\text{-}upto}\ (\lambda x.\ \mathit{real}\ x\ \mathit{powr} - s)\ x| \leq |x\ \mathit{powr}\ (1 - s) / (1 - s)| + c$
using *c*(*1*) *c*(*2*)[*OF x*] *x* **by** *linarith*
also have $|x\ \mathit{powr}\ (1 - s) / (1 - s)| = x\ \mathit{powr}\ (1 - s) / |1 - s|$
using *assms* **by** *simp*
also have $\ldots \leq x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s) / |1 - s|$
using *x* **by** (*intro divide-right-mono powr-mono*) *auto*
also have $c = c * x\ \mathit{powr}\ 0$ **using** *x* **by** *simp*
also have $c * x\ \mathit{powr}\ 0 \leq c * x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s)$
using *c*(*1*) *x* **by** (*intro mult-left-mono powr-mono*) *auto*
also have $x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s) / |1 - s| + c * x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s) = (1 / |1 - s| + c) * x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s)$
by (*simp add*: *algebra-simps*)
finally have $|\mathit{sum\text{-}upto}\ (\lambda x.\ \mathit{real}\ x\ \mathit{powr} - s)\ x| \leq (1 / |1 - s| + c) * x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s)$
by *simp*
}
moreover have $(1 / |1 - s| + c) > 0$
using *c assms* **by** (*intro add-pos-pos divide-pos-pos*) *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *zeta-partial-sum-le-pos-bigo*:
assumes $s > 0\ s \neq 1$
shows $(\lambda x.\ \mathit{sum\text{-}upto}\ (\lambda n.\ n\ \mathit{powr}\ {-s})\ x) \in O(\lambda x.\ x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s))$
proof −
from *zeta-partial-sum-le-pos''*[*OF assms*] **obtain** *c*
where $\forall\, x{\geq}1.\ |\mathit{sum\text{-}upto}\ (\lambda n.\ n\ \mathit{powr}\ {-s})\ x| \leq c * x\ \mathit{powr}\ \mathit{max}\ 0\ (1 - s)$ **by** *auto*
thus *?thesis*
by (*intro bigoI*[*of - c*] *eventually-mono*[*OF eventually-ge-at-top*[*of 1*]]) *auto*
qed

lemma *zeta-partial-sum-01-asymp-equiv*:
assumes $s \in \{0{<}..{<}1\}$
shows *sum-upto* $(\lambda n.\ n\ \mathit{powr}\ {-s}) \sim[\mathit{at\text{-}top}]\ (\lambda x.\ x\ \mathit{powr}\ (1 - s) / (1 - s))$
proof −
from *zeta-partial-sum-le-pos'*[*of s*] *assms* **obtain** *c* **where**
c: $c > 0\ \forall\, x{\geq}1.\ |\mathit{sum\text{-}upto}\ (\lambda x.\ \mathit{real}\ x\ \mathit{powr}\ {-s})\ x - x\ \mathit{powr}\ (1 - s) / (1 - s)| \leq c$ **by** *auto*

hence ($\lambda x.$ *sum-upto* ($\lambda x.$ *real x powr* $-s$) $x - x$ *powr* $(1 - s)$ / $(1 - s)$) $\in$ $O(\lambda\text{-}.\ 1)$
by (*intro bigoI*[*of* - *c*] *eventually-mono*[*OF eventually-ge-at-top*[*of 1*]]) *auto*
also have $(\lambda\text{-}.\ 1) \in o(\lambda x.\ x$ *powr* $(1 - s)$ / $(1 - s))$
using *assms* **by** *real-asymp*
finally show *?thesis*
by (*rule smallo-imp-asymp-equiv*)
qed

lemma *zeta-partial-sum-gt-1-asymp-equiv*:
fixes s :: *real*
assumes $s > 1$
defines $\zeta \equiv$ *Re* (*zeta s*)
shows *sum-upto* $(\lambda n.\ n$ *powr* $-s) \sim[$*at-top*$]$ $(\lambda x.\ \zeta)$
proof −
have [*simp*]: $\zeta \neq 0$
using *assms zeta-Re-gt-1-nonzero*[*of s*] *zeta-real*[*of s*] **by** (*auto elim*!: *Reals-cases*)
from *zeta-partial-sum-le-pos*[*of s*] *assms* **obtain** c **where**
c: $c > 0$ $\forall x{\geq}1.$ $|$*sum-upto* $(\lambda x.$ *real x powr* $-s)$ $x - (x$ *powr* $(1 - s)$ / $(1 - s) + \zeta)| \leq$
$c *$ *x powr* $-s$ **by** (*auto simp*: ζ*-def*)
hence ($\lambda x.$ *sum-upto* ($\lambda x.$ *real x powr* $-s$) $x - \zeta - x$ *powr* $(1 - s)$ / $(1 - s)$) $\in$ $O(\lambda x.\ x$ *powr* $-s)$
by (*intro bigoI*[*of* - *c*] *eventually-mono*[*OF eventually-ge-at-top*[*of 1*]]) *auto*
also have $(\lambda x.\ x$ *powr* $-s) \in o(\lambda\text{-}.\ 1)$
using ‹$s > 1$› **by** *real-asymp*
finally have ($\lambda x.$ *sum-upto* ($\lambda x.$ *real x powr* $-s$) $x - \zeta - x$ *powr* $(1 - s)$ / $(1 - s) +$
x *powr* $(1 - s)$ / $(1 - s)) \in o(\lambda\text{-}.\ 1)$
by (*rule sum-in-smallo*) (*use* ‹$s > 1$› **in** *real-asymp*)
thus *?thesis* **by** (*simp add*: *smallo-imp-asymp-equiv*)
qed

lemma *zeta-partial-sum-pos-bigtheta*:
assumes $s > 0$ $s \neq 1$
shows *sum-upto* $(\lambda n.\ n$ *powr* $-s) \in \Theta(\lambda x.\ x$ *powr max* 0 $(1 - s))$
proof (*cases* $s > 1$)
case *False*
thus *?thesis*
using *asymp-equiv-imp-bigtheta*[*OF zeta-partial-sum-01-asymp-equiv*[*of s*]] *assms*
by (*simp add*: *max-def*)
next
case *True*
have [*simp*]: *Re* (*zeta s*) $\neq 0$
using *True zeta-Re-gt-1-nonzero*[*of s*] *zeta-real*[*of s*] **by** (*auto elim*!: *Reals-cases*)
show *?thesis*
using *True asymp-equiv-imp-bigtheta*[*OF zeta-partial-sum-gt-1-asymp-equiv*[*of s*]]

by (*simp add*: *max-def*)
qed

lemma *zeta-partial-sum-le-neg*:
assumes $s > 0$
shows $\exists c{>}0.\ \forall x{\geq}1.\ |\mathit{sum\text{-}upto}\ (\lambda n.\ n\ \mathit{powr}\ s)\ x - x\ \mathit{powr}\ (1 + s)\ /\ (1 + s)| \leq c * x\ \mathit{powr}\ s$
proof (*rule sum-upto-asymptotics-lift-nat-real*)
show $(\lambda n.\ (\sum k = 1..n.\ \mathit{real}\ k\ \mathit{powr}\ s) - (\mathit{real}\ n\ \mathit{powr}\ (1 + s)\ /\ (1 + s))) \in O(\lambda n.\ \mathit{real}\ n\ \mathit{powr}\ s)$
using *zeta-partial-sum-bigo-neg*[*OF assms*(*1*)] **by** (*simp add*: *algebra-simps*)

show $(\lambda n.\ \mathit{real}\ n\ \mathit{powr}\ (1 + s)\ /\ (1 + s) - (\mathit{real}\ (\mathit{Suc}\ n)\ \mathit{powr}\ (1 + s)\ /\ (1 + s))) \in O(\lambda n.\ \mathit{real}\ n\ \mathit{powr}\ s)$
using *assms* **by** *real-asymp*
show $(\lambda n.\ \mathit{real}\ n\ \mathit{powr}\ s) \in O(\lambda n.\ \mathit{real}\ (\mathit{Suc}\ n)\ \mathit{powr}\ s)$
by *real-asymp*
show *mono-on* $\{1..\}$ $(\lambda a.\ a\ \mathit{powr}\ s) \vee \mathit{mono\text{-}on}\ \{1..\}\ (\lambda x.\ -\ (x\ \mathit{powr}\ s))$
using *assms* **by** (*intro disjI1*) (*auto intro*!: *mono-onI powr-mono2*)
show *mono-on* $\{1..\}$ $(\lambda a.\ a\ \mathit{powr}\ (1 + s)\ /\ (1 + s)) \vee \mathit{mono\text{-}on}\ \{1..\}\ (\lambda x.\ -\ (x\ \mathit{powr}\ (1 + s)\ /\ (1 + s)))$
using *assms* **by** (*intro disjI1 divide-right-mono powr-mono2 mono-onI*) *auto*
qed *auto*

lemma *zeta-partial-sum-neg-asymp-equiv*:
assumes $s > 0$
shows $\mathit{sum\text{-}upto}\ (\lambda n.\ n\ \mathit{powr}\ s) \sim[\mathit{at\text{-}top}]\ (\lambda x.\ x\ \mathit{powr}\ (1 + s)\ /\ (1 + s))$
proof −
from *zeta-partial-sum-le-neg*[*of s*] *assms* **obtain** c **where**
c: $c > 0$ $\forall x{\geq}1.\ |\mathit{sum\text{-}upto}\ (\lambda x.\ \mathit{real}\ x\ \mathit{powr}\ s)\ x - x\ \mathit{powr}\ (1 + s)\ /\ (1 + s)| \leq c * x\ \mathit{powr}\ s$
by *auto*
hence $(\lambda x.\ \mathit{sum\text{-}upto}\ (\lambda x.\ \mathit{real}\ x\ \mathit{powr}\ s)\ x - x\ \mathit{powr}\ (1 + s)\ /\ (1 + s)) \in O(\lambda x.\ x\ \mathit{powr}\ s)$
by (*intro bigoI*[*of* - *c*] *eventually-mono*[*OF eventually-ge-at-top*[*of 1*]]) *auto*
also have $(\lambda x.\ x\ \mathit{powr}\ s) \in o(\lambda x.\ x\ \mathit{powr}\ (1 + s)\ /\ (1 + s))$
using *assms* **by** *real-asymp*
finally show *?thesis*
by (*rule smallo-imp-asymp-equiv*)
qed

end

8 The summatory Möbius μ function

theory *Moebius-Mu-Sum*
imports
More-Dirichlet-Misc

Dirichlet-Series.Partial-Summation
Prime-Number-Theorem.Prime-Counting-Functions
Dirichlet-Series.Arithmetic-Summatory-Asymptotics
Shapiro-Tauberian
Partial-Zeta-Bounds
Prime-Number-Theorem.Prime-Number-Theorem-Library
Prime-Distribution-Elementary-Library
begin

In this section, we shall examine the summatory Möbius μ function $M(x) := \sum_{n \leq x} \mu(n)$. The main result is that $M(x) \in o(x)$ is equivalent to the Prime Number Theorem.

context
includes *prime-counting-syntax*
fixes *M H* :: *real* ⇒ *real*
defines *M* ≡ *sum-upto moebius-mu*
defines *H* ≡ *sum-upto* (λ*n*. *moebius-mu n* ∗ *ln n*)
begin

lemma *sum-upto-moebius-mu-integral*: *x* > *1* ⟹ ((λ*t*. *M t* / *t*) *has-integral M x* ∗ *ln x* − *H x*) {*1*..*x*}
and *sum-upto-moebius-mu-integrable*: *a* ≥ *1* ⟹ (λ*t*. *M t* / *t*) *integrable-on* {*a*..*b*}
proof −
{
fix *a b* :: *real*
assume *ab*: *a* ≥ *1 a* < *b*
have ((λ*t*. *M t* ∗ (*1* / *t*)) *has-integral M b* ∗ *ln b* − *M a* ∗ *ln a* −
(∑ *n*∈*real* −‘ {*a*<..*b*}. *moebius-mu n* ∗ *ln* (*real n*))) {*a*..*b*}
unfolding *M-def* **using** *ab*
by (*intro partial-summation-strong* [**where** *X* = {}])
(*auto intro*!: *derivative-eq-intros continuous-intros*
simp flip: *has-real-derivative-iff-has-vector-derivative*)
} note ∗ = *this*
{
fix *x* :: *real* **assume** *x*: *x* > *1*
have (∑ *n*∈*real* −‘ {*1*<..*x*}. *moebius-mu n* ∗ *ln* (*real n*)) = *H x*
unfolding *H-def sum-upto-def* **by** (*intro sum.mono-neutral-cong-left*) (*use x* **in** *auto*)
thus ((λ*t*. *M t* / *t*) *has-integral M x* ∗ *ln x* − *H x*) {*1*..*x*} **using** ∗[*of 1 x*] *x* **by** *simp*
}
{
fix *a b* :: *real* **assume** *ab*: *a* ≥ *1*
show (λ*t*. *M t* / *t*) *integrable-on* {*a*..*b*}
using ∗[*of a b*] *ab*
by (*cases a b rule*: *linorder-cases*) (*auto intro*: *integrable-negligible*)
}
qed

lemma *sum-moebius-mu-bound*:
assumes $x \geq 0$
shows $|M\ x| \leq x$
proof −
have $|M\ x| \leq$ *sum-upto* $(\lambda n.\ |$*moebius-mu* $n|)\ x$
unfolding *M-def sum-upto-def* **by** (*rule sum-abs*)
also have $\ldots \leq$ *sum-upto* $(\lambda n.\ 1)\ x$
unfolding *sum-upto-def* **by** (*intro sum-mono*) (*auto simp*: *moebius-mu-def*)
also have $\ldots \leq x$ **using** *assms*
by (*simp add*: *sum-upto-altdef*)
finally show *?thesis* **.**
qed

lemma *sum-moebius-mu-aux1*: $(\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x)) \in O(\lambda x.\ 1\ /\ ln\ x)$
proof −
define R **where** $R = (\lambda x.$ *integral* $\{1..x\}\ (\lambda t.\ M\ t\ /\ t))$
have *eventually* $(\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x) = R\ x\ /\ (x * ln\ x))$ *at-top*
using *eventually-gt-at-top*[*of 1*]
proof *eventually-elim*
case (*elim x*)
thus *?case*
using *sum-upto-moebius-mu-integral*[*of x*] **by** (*simp add*: *R-def has-integral-iff field-simps*)
qed
hence $(\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x)) \in \Theta(\lambda x.\ R\ x\ /\ (x * ln\ x))$
by (*intro bigthetaI-cong*)
also have $(\lambda x.\ R\ x\ /\ (x * ln\ x)) \in O(\lambda x.\ x\ /\ (x * ln\ x))$
proof (*intro landau-o.big.divide-right*)
have $M \in O(\lambda x.\ x)$
using *sum-moebius-mu-bound*
by (*intro bigoI*[**where** $c = 1$] *eventually-mono*[*OF eventually-ge-at-top*[*of 0*]]) *auto*
hence $(\lambda t.\ M\ t\ /\ t) \in O(\lambda t.\ 1)$
by (*simp add*: *landau-divide-simps*)
thus $R \in O(\lambda x.\ x)$
unfolding *R-def*
by (*intro integral-bigo*[**where** $g' = \lambda$-. *1*])
(*auto simp*: *filterlim-ident has-integral-iff intro*!: *sum-upto-moebius-mu-integrable*)
qed (*intro eventually-mono*[*OF eventually-gt-at-top*[*of 1*]], *auto*)
also have $(\lambda x{::}real.\ x\ /\ (x * ln\ x)) \in \Theta(\lambda x.\ 1\ /\ ln\ x)$
by *real-asymp*
finally show *?thesis* **.**
qed

lemma *sum-moebius-mu-aux2*: $((\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x)) \longrightarrow 0)$ *at-top*
proof −
have $(\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x)) \in O(\lambda x.\ 1\ /\ ln\ x)$

```
    by (rule sum-moebius-mu-aux1)
  also have (λx. 1 / ln x) ∈ o(λ-. 1 :: real)
    by real-asymp
  finally show ?thesis by (auto dest!: smalloD-tendsto)
qed

lemma sum-moebius-mu-ln-eq: H = (λx. − dirichlet-prod' moebius-mu ψ x)
proof
  fix x :: real
  have fds mangoldt = (fds-deriv (fds moebius-mu) ∗ fds-zeta :: real fds)
    using fds-mangoldt' by (simp add: mult-ac)
  hence eq: fds-deriv (fds moebius-mu) = fds moebius-mu ∗ (fds mangoldt :: real
fds)
    by (subst (asm) fds-moebius-inversion [symmetric])
  have −H x = sum-upto (λn. −ln n ∗ moebius-mu n) x
    by (simp add: H-def sum-upto-def sum-negf mult-ac)
  also have ... = sum-upto (λn. dirichlet-prod moebius-mu mangoldt n) x
   using eq by (intro sum-upto-cong) (auto simp: fds-eq-iff fds-nth-deriv fds-nth-mult)
  also have ... = dirichlet-prod' moebius-mu ψ x
   by (subst sum-upto-dirichlet-prod) (simp add: primes-psi-def dirichlet-prod'-def)
  finally show H x = −dirichlet-prod' moebius-mu ψ x
    by simp
qed


theorem PNT-implies-sum-moebius-mu-sublinear:
  assumes ψ ∼[at-top] (λx. x)
  shows   M ∈ o(λx. x)
proof −
  have ((λx. H x / (x ∗ ln x)) ⟶ 0) at-top
  proof (rule tendstoI)
    fix ε' :: real assume ε': ε' > 0
    define ε where ε = ε' / 2
    from ε' have ε: ε > 0 by (simp add: ε-def)
    from assms have ((λx. ψ x / x) ⟶ 1) at-top
        by (elim asymp-equivD-strong) (auto intro!: eventually-mono[OF eventu-
ally-gt-at-top[of 0]])
    from tendstoD[OF this ε] have eventually (λx. |ψ x / x − 1| < ε) at-top
      by (simp add: dist-norm)
    hence eventually (λx. |ψ x − x| < ε ∗ x) at-top
     using eventually-gt-at-top[of 0] by eventually-elim (auto simp: abs-if field-simps)
    then obtain A' where A': ⋀x. x ≥ A' ⟹ |ψ x − x| < ε ∗ x
      by (auto simp: eventually-at-top-linorder)
    define A where A = max 2 A'
    from A' have A: A ≥ 2 ⋀x. x ≥ A ⟹ |ψ x − x| < ε ∗ x
      by (auto simp: A-def)

    have H-bound: |H x| / (x ∗ ln x) ≤ (1 + ε + ψ A) / ln x + ε if x ≥ A for x
    proof −
```

from ‹$x \geq A$› **have** $x \geq 2$ **using** $A(1)$ **by** *linarith*
note $x = ‹x \geq A› \; ‹x \geq 2›$

define y **where** $y = nat \lfloor floor \; (x \; / \; A) \rfloor$
have $real \; y = real\text{-}of\text{-}int \lfloor x \; / \; A \rfloor$ **using** $A \; x$ **by** (*simp add*: *y-def*)
also have $real\text{-}of\text{-}int \lfloor x \; / \; A \rfloor \leq x \; / \; A$ **by** *linarith*
also have $\ldots \leq x$ **using** $x \; A(1)$ **by** (*simp add*: *field-simps*)
finally have $y \leq x$ **.**
have $y \geq 1$ **using** $x \; A(1)$ **by** (*auto simp*: *y-def le-nat-iff le-floor-iff*)
note $y = ‹y \geq 1› \; ‹y \leq x›$

define *S1* **where** [*simp*]: $S1 = sum\text{-}upto \; (\lambda m. \; moebius\text{-}mu \; m * \psi \; (x \; / \; m)) \; y$
define *S2* **where** [*simp*]: $S2 = (\sum m \mid m > y \wedge real \; m \leq x. \; moebius\text{-}mu \; m * \psi \; (x \; / \; m))$

have *fin*: $finite \; \{m. \; y < m \wedge real \; m \leq x\}$
by (*rule finite-subset*[*of* - {..*nat* $\lfloor x \rfloor$}]) (*auto simp*: *le-nat-iff le-floor-iff*)
have $H \; x = -dirichlet\text{-}prod' \; moebius\text{-}mu \; \psi \; x$
by (*simp add*: *sum-moebius-mu-ln-eq*)
also have $dirichlet\text{-}prod' \; moebius\text{-}mu \; \psi \; x =$
$(\sum m \mid m > 0 \wedge real \; m \leq x. \; moebius\text{-}mu \; m * \psi \; (x \; / \; m))$
unfolding *dirichlet-prod'-def sum-upto-def* **..**
also have $\{m. \; m > 0 \wedge real \; m \leq x\} = \{0<..y\} \cup \{m. \; y < m \wedge real \; m \leq x\}$
using $x \; y \; A(1)$ **by** *auto*
also have $(\sum m \in \ldots. \; moebius\text{-}mu \; m * \psi \; (x \; / \; m)) = S1 + S2$
unfolding *dirichlet-prod'-def sum-upto-def S1-def S2-def* **using** *fin*
by (*subst sum.union-disjoint*) (*auto intro*: *sum.cong*)
finally have *abs-H-eq*: $|H \; x| = |S1 + S2|$ **by** *simp*

define *S1-1* **where** [*simp*]: $S1\text{-}1 = sum\text{-}upto \; (\lambda m. \; moebius\text{-}mu \; m \; / \; m) \; y$
define *S1-2* **where** [*simp*]: $S1\text{-}2 = sum\text{-}upto \; (\lambda m. \; moebius\text{-}mu \; m * (\psi \; (x \; / \; m) - x \; / \; m)) \; y$

have $|S1| = |x * S1\text{-}1 + S1\text{-}2|$
by (*simp add*: *sum-upto-def sum-distrib-left sum-distrib-right*
mult-ac sum-subtractf ring-distribs)
also have $\ldots \leq x * |S1\text{-}1| + |S1\text{-}2|$
by (*rule order.trans*[*OF abs-triangle-ineq*]) (*use* x **in** ‹*simp add*: *abs-mult*›)
also have $\ldots \leq x * 1 + \varepsilon * x * (ln \; x + 1)$
proof (*intro add-mono mult-left-mono*)
show $|S1\text{-}1| \leq 1$
using *abs-sum-upto-moebius-mu-over-n-le*[*of* y] **by** *simp*
next
have $|S1\text{-}2| \leq sum\text{-}upto \; (\lambda m. \; |moebius\text{-}mu \; m * (\psi \; (x \; / \; m) - x \; / \; m)|) \; y$
unfolding *S1-2-def sum-upto-def* **by** (*rule sum-abs*)
also have $\ldots \leq sum\text{-}upto \; (\lambda m. \; 1 * (\varepsilon * (x \; / \; m))) \; y$
unfolding *abs-mult sum-upto-def*
proof (*intro sum-mono mult-mono less-imp-le*[*OF A(2)*])
fix m **assume** m: $m \in \{i. \; 0 < i \wedge real \; i \leq real \; y\}$

hence *real m* ≤ *real y* **by** *simp*
also from *x A(1)* **have** ... = *of-int* ⌊*x* / *A*⌋ **by** (*simp add*: *y-def*)
also have ... ≤ *x* / *A* **by** *linarith*
finally show *A* ≤ *x* / *real m* **using** *A(1) m* **by** (*simp add*: *field-simps*)
qed (*auto simp*: *moebius-mu-def field-simps*)
also have ... = ε * *x* * (∑ *i*∈{*0*<..*y*}. *inverse* (*real i*))
by (*simp add*: *sum-upto-altdef sum-distrib-left divide-simps*)
also have (∑ *i*∈{*0*<..*y*}. *inverse* (*real i*)) = *harm* (*nat* ⌊*y*⌋)
unfolding *harm-def* **by** (*intro sum.cong*) *auto*
also have ... ≤ *ln* (*nat* ⌊*y*⌋) + *1*
by (*rule harm-le*) (*use y* **in** *auto*)
also have *ln* (*nat* ⌊*y*⌋) ≤ *ln x*
using *y* **by** *simp*
finally show |*S1-2*| ≤ ε * *x* * (*ln x* + *1*) **using** ε *x* **by** *simp*
qed (*use x* **in** *auto*)
finally have *S1-bound*: |*S1*| ≤ *x* + ε * *x* * *ln x* + ε * *x*
by (*simp add*: *algebra-simps*)

have |*S2*| ≤ (∑ *m* | *y* < *m* ∧ *real m* ≤ *x*. |*moebius-mu m* * ψ (*x* / *m*)|)
unfolding *S2-def* **by** (*rule sum-abs*)
also have ... ≤ (∑ *m* | *y* < *m* ∧ *real m* ≤ *x*. *1* * ψ *A*)
unfolding *abs-mult* **using** *y*
proof (*intro sum-mono mult-mono*)
fix *m* **assume** *m*: *m* ∈ {*m*. *y* < *m* ∧ *real m* ≤ *x*}
hence *y* < *m* **by** *simp*
moreover have *y* = *of-int* ⌊*x* / *A*⌋ **using** *x A(1)* **by** (*simp add*: *y-def*)
ultimately have ⌊*x* / *A*⌋ < *m* **by** *simp*
hence *x* / *A* ≤ *real m* **by** *linarith*
hence ψ (*x* / *real m*) ≤ ψ *A*
using *m A(1)* **by** (*intro ψ-mono*) (*auto simp*: *field-simps*)
thus |ψ (*x* / *real m*)| ≤ ψ *A*
by (*simp add*: *ψ-nonneg*)
qed (*auto simp*: *moebius-mu-def ψ-nonneg field-simps intro*!: *ψ-mono*)
also have ... ≤ *sum-upto* (λ-. *1* * ψ *A*) *x*
unfolding *sum-upto-def* **by** (*intro sum-mono2*) *auto*
also have ... = *real* (*nat* ⌊*x*⌋) * ψ *A*
by (*simp add*: *sum-upto-altdef*)
also have ... ≤ *x* * ψ *A*
using *x* **by** (*intro mult-right-mono*) *auto*
finally have *S2-bound*: |*S2*| ≤ *x* * ψ *A* **.**

have |*H x*| ≤ |*S1*| + |*S2*| **using** *abs-H-eq* **by** *linarith*
also have ... ≤ *x* + ε * *x* * *ln x* + ε * *x* + *x* * ψ *A*
by (*intro add-mono S1-bound S2-bound*)
finally have |*H x*| ≤ (*1* + ε + ψ *A*) * *x* + ε * *x* * *ln x*
by (*simp add*: *algebra-simps*)
thus |*H x*| / (*x* * *ln x*) ≤ (*1* + ε + ψ *A*) / *ln x* + ε
using *x* **by** (*simp add*: *field-simps*)
qed

have *eventually* $(\lambda x.\ |H\ x|\ /\ (x * ln\ x) \leq (1 + \varepsilon + \psi\ A)\ /\ ln\ x + \varepsilon)$ *at-top*
using *eventually-ge-at-top*[*of A*] **by** *eventually-elim* (*use H-bound* **in** *auto*)
moreover have *eventually* $(\lambda x.\ (1 + \varepsilon + \psi\ A)\ /\ ln\ x + \varepsilon < \varepsilon')$ *at-top*
unfolding ε*-def* **using** ε' **by** *real-asymp*
moreover have *eventually* $(\lambda x.\ |H\ x|\ /\ (x * ln\ x) = |H\ x\ /\ (x * ln\ x)|)$ *at-top*
using *eventually-gt-at-top*[*of 1*] **by** *eventually-elim* (*simp add*: *abs-mult*)
ultimately have *eventually* $(\lambda x.\ |H\ x\ /\ (x * ln\ x)| < \varepsilon')$ *at-top*
by *eventually-elim simp*
thus *eventually* $(\lambda x.\ dist\ (H\ x\ /\ (x * ln\ x))\ 0 < \varepsilon')$ *at-top*
by (*simp add*: *dist-norm*)
qed
hence $(\lambda x.\ H\ x\ /\ (x * ln\ x)) \in o(\lambda\text{-}.\ 1)$
by (*intro smalloI-tendsto*) *auto*
hence $(\lambda x.\ H\ x\ /\ (x * ln\ x) + (M\ x\ /\ x - H\ x\ /\ (x * ln\ x))) \in o(\lambda\text{-}.\ 1)$
proof (*rule sum-in-smallo*)
have $(\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x)) \in O(\lambda x.\ 1\ /\ ln\ x)$
by (*rule sum-moebius-mu-aux1*)
also have $(\lambda x{::}real.\ 1\ /\ ln\ x) \in o(\lambda\text{-}.\ 1)$
by *real-asymp*
finally show $(\lambda x.\ M\ x\ /\ x - H\ x\ /\ (x * ln\ x)) \in o(\lambda\text{-}.\ 1)$ **.**
qed
thus *?thesis* **by** (*simp add*: *landau-divide-simps*)
qed

theorem *sum-moebius-mu-sublinear-imp-PNT*:
assumes $M \in o(\lambda x.\ x)$
shows $\psi \sim[at\text{-}top]\ (\lambda x.\ x)$
proof −
define σ :: *nat* ⇒ *real* **where** [*simp*]: $\sigma = (\lambda n.\ real\ (divisor\text{-}count\ n))$
define C **where** [*simp*]: $C = (euler\text{-}mascheroni :: real)$
define f :: *nat* ⇒ *real* **where** $f = (\lambda n.\ \sigma\ n - ln\ n - 2 * C)$
define F **where** [*simp*]: $F = sum\text{-}upto\ f$
write *moebius-mu* (‹μ›)

— The proof is based on the fact that $\psi(x) - x$ can be approximated fairly well by the Dirichlet product $\sum_{n \leq x} \sum_{d|n} \mu(d) f(n/d)$:
have *eq*: $\psi\ x - x = -sum\text{-}upto\ (dirichlet\text{-}prod\ \mu\ f)\ x - frac\ x - 2 * C$ **if** *x*: $x \geq 1$ **for** x
proof −
have $\lfloor x \rfloor - \psi\ x - 2 * C =$
$sum\text{-}upto\ (\lambda\text{-}.\ 1)\ x - sum\text{-}upto\ mangoldt\ x - sum\text{-}upto\ (\lambda n.\ if\ n = 1\ then\ 2 * C\ else\ 0)\ x$
using *x* **by** (*simp add*: *sum-upto-altdef* ψ*-def le-nat-iff le-floor-iff*)
also have $\ldots = sum\text{-}upto\ (\lambda n.\ 1 - mangoldt\ n - (if\ n = 1\ then\ 2 * C\ else\ 0))\ x$
by (*simp add*: *sum-upto-def sum-subtractf*)
also have $\ldots = sum\text{-}upto\ (dirichlet\text{-}prod\ \mu\ f)\ x$

by (*intro sum-upto-cong refl moebius-inversion*)
(*auto simp*: *divisor-count-def sum-subtractf mangoldt-sum f-def*)
finally show $\psi\ x - x = -sum\text{-}upto\ (dirichlet\text{-}prod\ \mu\ f)\ x - frac\ x - 2 * C$
by (*simp add*: *algebra-simps frac-def*)
qed

— We now obtain a bound of the form $|F\ x| \leq B * sqrt\ x$.
have $F \in O(sqrt)$
proof −
have $F \in \Theta(\lambda x.\ (sum\text{-}upto\ \sigma\ x - (x * ln\ x + (2 * C - 1) * x)) -$
$(sum\text{-}upto\ ln\ x - x * ln\ x + x) + 2 * C * frac\ x)$ (**is** - $\in \Theta(?rhs)$)
by (*intro bigthetaI-cong eventually-mono*[*OF eventually-ge-at-top*[*of 1*]])
(*auto simp*: *sum-upto-altdef sum-subtractf f-def frac-def algebra-simps sum.distrib*)
also have $?rhs \in O(sqrt)$
proof (*rule sum-in-bigo*, *rule sum-in-bigo*)
show $(\lambda x.\ sum\text{-}upto\ \sigma\ x - (x * ln\ x + (2 * C - 1) * x)) \in O(sqrt)$
unfolding *C-def* σ*-def* **by** (*rule summatory-divisor-count-asymptotics*)
show $(\lambda x.\ sum\text{-}upto\ (\lambda x.\ ln\ (real\ x))\ x - x * ln\ x + x) \in O(sqrt)$
by (*rule landau-o.big.trans*[*OF sum-upto-ln-stirling-weak-bigo*]) *real-asymp*
qed (*use euler-mascheroni-pos* **in** *real-asymp*)
finally show *?thesis* **.**
qed
hence $(\lambda n.\ F\ (real\ n)) \in O(sqrt)$
by (*rule landau-o.big.compose*) *real-asymp*
from *natfun-bigoE*[*OF this*, *of 1*] **obtain** $B :: real$
where B: $B > 0\ \bigwedge n.\ n \geq 1 \Longrightarrow |F\ (real\ n)| \leq B * sqrt\ (real\ n)$
by *auto*
have B': $|F\ x| \leq B * sqrt\ x$ **if** $x \geq 1$ **for** x
proof −
have $|F\ x| \leq B * sqrt\ (nat\ \lfloor x \rfloor)$
using $B(2)$[*of nat* $\lfloor x \rfloor$] *that* **by** (*simp add*: *sum-upto-altdef le-nat-iff le-floor-iff*)
also have $\ldots \leq B * sqrt\ x$
using $B(1)$ *that* **by** (*intro mult-left-mono*) *auto*
finally show *?thesis* **.**
qed

— Next, we obtain a good bound for $\sum_{n \leq x} \frac{1}{\sqrt{n}}$.
from *zeta-partial-sum-le-pos''*[*of 1 / 2*] **obtain** A
where A: $A > 0\ \bigwedge x.\ x \geq 1 \Longrightarrow |sum\text{-}upto\ (\lambda n.\ 1\ /\ sqrt\ n)\ x| \leq A * sqrt\ x$
by (*auto simp*: *max-def powr-half-sqrt powr-minus field-simps*)

— Finally, we show that $\sum_{n \leq x} \sum_{d|n} \mu(d) f(n/d) \in o(x)$.
have $sum\text{-}upto\ (dirichlet\text{-}prod\ \mu\ f) \in o(\lambda x.\ x)$
proof (*rule landau-o.smallI*)
fix $\varepsilon :: real$
assume ε: $\varepsilon > 0$

have $*$: $eventually\ (\lambda x.\ |sum\text{-}upto\ (dirichlet\text{-}prod\ \mu\ f)\ x| \leq \varepsilon * x)\ at\text{-}top$

```
    if b: b ≥ 1 A * B / sqrt b ≤ ε / 3 B / sqrt b ≤ ε / 3 for b
  proof −
    define K :: real where K = sum-upto (λn. |f n| / n) b
    have C ≠ (1 / 2) using euler-mascheroni-gt-19-over-33 by auto
    hence K: K > 0 unfolding K-def f-def sum-upto-def
      by (intro sum-pos2[where i = 1]) (use ‹b ≥ 1› in auto)
    have eventually (λx. |M x / x| < ε / 3 / K) at-top
     using smalloD-tendsto[OF assms] ε K by (auto simp: tendsto-iff dist-norm)
    then obtain c' where c': ⋀x. x ≥ c' ⟹ |M x / x| < ε / 3 / K
      by (auto simp: eventually-at-top-linorder)
    define c where c = max 1 c'
    have c: |M x| < ε / 3 / K * x if x ≥ c for x
      using c'[of x] that by (simp add: c-def field-simps)

    show eventually (λx. |sum-upto (dirichlet-prod μ f) x| ≤ ε * x) at-top
        using eventually-ge-at-top[of b * c] eventually-ge-at-top[of 1] eventu-
ally-ge-at-top[of b]
    proof eventually-elim
      case (elim x)
      define a where a = x / b
      from elim ‹b ≥ 1› have ab: a ≥ 1 b ≥ 1 a * b = x
        by (simp-all add: a-def field-simps)
      from ab have a * 1 ≤ a * b by (intro mult-mono) auto
      hence a ≤ x by (simp add: ab(3))
       from ab have a * 1 ≤ a * b and 1 * b ≤ a * b by (intro mult-mono;
simp)+
      hence a ≤ x b ≤ x by (simp-all add: ab(3))
      have a = x / b b = x / a using ab by (simp-all add: field-simps)

      have sum-upto (dirichlet-prod μ f) x =
            sum-upto (λn. μ n * F (x / n)) a + sum-upto (λn. M (x / n) * f n)
b − M a * F b
        unfolding M-def F-def by (rule hyperbola-method) (use ab in auto)
      also have |...| ≤ ε / 3 * x + ε / 3 * x + ε / 3 * x
    proof (rule order.trans[OF abs-triangle-ineq4] order.trans[OF abs-triangle-ineq]
add-mono)+
        have |sum-upto (λn. μ n * F (x / real n)) a| ≤ sum-upto (λn. |μ n * F
(x / real n)|) a
          unfolding sum-upto-def by (rule sum-abs)
        also have ... ≤ sum-upto (λn. 1 * (B * sqrt (x / real n))) a
          unfolding sum-upto-def abs-mult using ‹a ≤ x›
          by (intro sum-mono mult-mono B') (auto simp: moebius-mu-def)
        also have ... = B * sqrt x * sum-upto (λn. 1 / sqrt n) a
          by (simp add: sum-upto-def sum-distrib-left real-sqrt-divide)
        also have ... ≤ B * sqrt x * |sum-upto (λn. 1 / sqrt n) a|
          using B(1) ‹x ≥ 1› by (intro mult-left-mono) auto
        also have ... ≤ B * sqrt x * (A * sqrt a)
          using ‹a ≥ 1› B(1) ‹x ≥ 1› by (intro mult-left-mono A) auto
        also have ... = A * B / sqrt b * x
```

```
      using ab ‹x ≥ 1›‹x ≥ 1› by (subst ‹a = x / b›) (simp-all add: field-simps
real-sqrt-divide)
      also have ... ≤ ε / 3 * x using ‹x ≥ 1› by (intro mult-right-mono b)
auto
      finally show |sum-upto (λn. μ n * F (x / n)) a| ≤ ε / 3 * x .
    next
      have |sum-upto (λn. M (x / n) * f n) b| ≤ sum-upto (λn. |M (x / n) * f
n|) b
        unfolding sum-upto-def by (rule sum-abs)
      also have ... ≤ sum-upto (λn. ε / 3 / K * (x / n) * |f n|) b
        unfolding sum-upto-def abs-mult
      proof (intro sum-mono mult-right-mono)
        fix n assume n: n ∈ {n. n > 0 ∧ real n ≤ b}
        have c ≥ 0 by (simp add: c-def)
        with n have c * n ≤ c * b by (intro mult-left-mono) auto
        also have ... ≤ x using ‹b * c ≤ x› by (simp add: algebra-simps)
        finally show |M (x / real n)| ≤ ε / 3 / K * (x / real n)
          by (intro less-imp-le[OF c]) (use n in ‹auto simp: field-simps›)
      qed auto
      also have ... = ε / 3 * x / K * sum-upto (λn. |f n| / n) b
        by (simp add: sum-upto-def sum-distrib-left)
      also have ... = ε / 3 * x
        unfolding K-def [symmetric] using K by simp
      finally show |sum-upto (λn. M (x / real n) * f n) b| ≤ ε / 3 * x .
    next
      have |M a * F b| ≤ a * (B * sqrt b)
      unfolding abs-mult using ab by (intro mult-mono sum-moebius-mu-bound
B') auto
      also have ... = B / sqrt b * x
        using ab(1,2) by (simp add: real-sqrt-mult ‹b = x / a› real-sqrt-divide
field-simps)
      also have ... ≤ ε / 3 * x using ‹x ≥ 1› by (intro mult-right-mono b)
auto
      finally show |M a * F b| ≤ ε / 3 * x .
    qed
    also have ... = ε * x by simp
    finally show ?case .
  qed
qed

have eventually (λb::real. b ≥ 1 ∧ A * B / sqrt b ≤ ε / 3 ∧ B / sqrt b ≤ ε /
3) at-top
  using ε by (intro eventually-conj; real-asymp)
then obtain b where b ≥ 1 A * B / sqrt b ≤ ε / 3 B / sqrt b ≤ ε / 3
  by (auto simp: eventually-at-top-linorder)
from *[OF this] have eventually (λx. |sum-upto (dirichlet-prod μ f) x| ≤ ε *
x) at-top .
thus eventually (λx. norm (sum-upto (dirichlet-prod μ f) x) ≤ ε * norm x)
at-top
```

using *eventually-ge-at-top*[*of 0*] **by** *eventually-elim simp*
qed

have $(\lambda x.\ \psi\ x - x) \in \Theta(\lambda x.\ -(\mathit{sum\text{-}upto}\ (\mathit{dirichlet\text{-}prod}\ \mu\ f)\ x + (\mathit{frac}\ x + 2 * C)))$
by (*intro bigthetaI-cong eventually-mono*[*OF eventually-ge-at-top*[*of 1*]], *subst eq*) *auto*
hence $(\lambda x.\ \psi\ x - x) \in \Theta(\lambda x.\ \mathit{sum\text{-}upto}\ (\mathit{dirichlet\text{-}prod}\ \mu\ f)\ x + (\mathit{frac}\ x + 2 * C))$
by (*simp only*: *landau-theta.uminus*)
also have $(\lambda x.\ \mathit{sum\text{-}upto}\ (\mathit{dirichlet\text{-}prod}\ \mu\ f)\ x + (\mathit{frac}\ x + 2 * C)) \in o(\lambda x.\ x)$
using ‹$\mathit{sum\text{-}upto}\ (\mathit{dirichlet\text{-}prod}\ \mu\ f) \in o(\lambda x.\ x)$› **by** (*rule sum-in-smallo*) *real-asymp*+
finally show *?thesis* **by** (*rule smallo-imp-asymp-equiv*)
qed

We now turn to a related fact: For the weighted sum $A(x) := \sum_{n \leq x} \mu(n)/n$, the asymptotic relation $A(x) \in o(1)$ is also equivalent to the Prime Number Theorem. Like Apostol, we only show one direction, namely that $A(x) \in o(1)$ implies the PNT.

context
fixes A **defines** $A \equiv \mathit{sum\text{-}upto}\ (\lambda n.\ \mathit{moebius\text{-}mu}\ n\ /\ n)$
begin

lemma *sum-upto-moebius-mu-integral'*: $x > 1 \Longrightarrow (A\ \mathit{has\text{-}integral}\ x * A\ x - M\ x)\ \{1..x\}$
and *sum-upto-moebius-mu-integrable'*: $a \geq 1 \Longrightarrow A\ \mathit{integrable\text{-}on}\ \{a..b\}$
proof −
{
fix $a\ b :: \mathit{real}$
assume *ab*: $a \geq 1\ a < b$
have $((\lambda t.\ A\ t * 1)\ \mathit{has\text{-}integral}\ A\ b * b - A\ a * a - (\sum n{\in}\mathit{real} -` \{a<..b\}.\ \mathit{moebius\text{-}mu}\ n\ /\ n * n))\ \{a..b\}$
unfolding *M-def A-def* **using** *ab*
by (*intro partial-summation-strong* [**where** $X = \{\}$])
(*auto intro*!: *derivative-eq-intros continuous-intros*
simp flip: *has-real-derivative-iff-has-vector-derivative*)
} **note** $* = \mathit{this}$
{
fix $x :: \mathit{real}$ **assume** *x*: $x > 1$
have [*simp*]: $A\ 1 = 1$ **by** (*simp add*: *A-def*)
have $(\sum n{\in}\mathit{real} -` \{1<..x\}.\ \mathit{moebius\text{-}mu}\ n\ /\ n * n) = (\sum n{\in}\mathit{insert}\ 1\ (\mathit{real} -` \{1<..x\}).\ \mathit{moebius\text{-}mu}\ n\ /\ n * n) - 1$
using *finite-vimage-real-of-nat-greaterThanAtMost*[*of 1 x*] **by** (*subst sum.insert*) *auto*
also have $\mathit{insert}\ 1\ (\mathit{real} -` \{1<..x\}) = \{n.\ n > 0 \wedge \mathit{real}\ n \leq x\}$
using *x* **by** *auto*
also have $(\sum n \mid 0 < n \wedge \mathit{real}\ n \leq x.\ \mathit{moebius\text{-}mu}\ n\ /\ \mathit{real}\ n * \mathit{real}\ n) = M\ x$
unfolding *M-def sum-upto-def* **by** (*intro sum.cong*) *auto*

finally show (*A has-integral x * A x - M x*) {*1..x*} **using** *[of 1 x] x* **by** (*simp add*: *mult-ac*)
}
{
fix *a b* :: *real* **assume** *ab*: *a* ≥ *1*
show *A integrable-on* {*a..b*}
using *[of a b] ab*
by (*cases a b rule*: *linorder-cases*) (*auto intro*: *integrable-negligible*)
}
qed

theorem *sum-moebius-mu-div-n-smallo-imp-PNT*:
assumes *smallo*: *A* ∈ *o*(λ-. *1*)
shows *M* ∈ *o*(λ*x*. *x*) **and** ψ ∼[*at-top*] (λ*x*. *x*)
proof −
have *eventually* (λ*x*. *M x* = *x* * *A x* − *integral* {*1..x*} *A*) *at-top*
using *eventually-gt-at-top*[*of 1*]
by *eventually-elim* (*use sum-upto-moebius-mu-integral′* **in** ‹*simp add*: *has-integral-iff*›)
hence *M* ∈ Θ(λ*x*. *x* * *A x* − *integral* {*1..x*} *A*)
by (*rule bigthetaI-cong*)
also have (λ*x*. *x* * *A x* − *integral* {*1..x*} *A*) ∈ *o*(λ*x*. *x*)
proof (*intro sum-in-smallo*)
from *smallo* **show** (λ*x*. *x* * *A x*) ∈ *o*(λ*x*. *x*)
by (*simp add*: *landau-divide-simps*)
show (λ*x*. *integral* {*1..x*} *A*) ∈ *o*(λ*x*. *x*)
by (*intro integral-smallo*[*OF smallo*] *sum-upto-moebius-mu-integrable′*)
(*auto intro*!: *derivative-eq-intros filterlim-ident*)
qed
finally show *M* ∈ *o*(λ*x*. *x*) .
thus ψ ∼[*at-top*] (λ*x*. *x*)
by (*rule sum-moebius-mu-sublinear-imp-PNT*)
qed

end

end

end

9 Elementary bounds on $\pi(x)$ and p_n

theory *Elementary-Prime-Bounds*
imports
Prime-Number-Theorem.Prime-Counting-Functions
Prime-Distribution-Elementary-Library
More-Dirichlet-Misc
begin

In this section, we will follow Apostol and give elementary proofs of Chebyshev-type lower and upper bounds for $\pi(x)$, i. e. $c_1 x/\ln x < \pi(x) < c_2 x/\ln x$. From this, similar bounds for p_n follow as easy corollaries.

9.1 Preliminary lemmas

The following two estimates relating the central Binomial coefficient to powers of 2 and 4 form the starting point for Apostol's elementary bounds for $\pi(x)$:

```
lemma twopow-le-central-binomial: 2 ^ n ≤ ((2 * n) choose n)
proof −
  have 2 ^ n * fact n ^ 2 ≤ (fact (2 * n) :: nat)
  proof (induction n)
    case (Suc n)
    have (fact (2 * Suc n) :: nat) = (2 * n + 1) * (2 * n + 2) * fact (2 * n)
      by (simp add: algebra-simps)

    have 2 ^ Suc n * fact (Suc n) ^ 2 = 2 ^ n * fact n ^ 2 * 2 * (n + 1) ^ 2
      by (simp add: algebra-simps power2-eq-square)
    also have ... ≤ fact (2 * n) * 2 * (n + 1) ^ 2
      by (intro mult-right-mono Suc.IH) auto
    also have ... = fact (2 * n) * (2 * (n + 1) ^ 2)
      by (simp add: mult-ac)
    also have ... ≤ fact (2 * n) * ((2 * n + 1) * (2 * n + 2))
      by (intro mult-left-mono) (auto simp: power2-eq-square)
    also have ... = fact (2 * Suc n)
      by (simp add: algebra-simps)
    finally show ?case .
  qed simp-all
  also have ... = (2 * n choose n) * fact n ^ 2
   using binomial-fact-lemma[of n 2 * n] by (simp add: power2-eq-square mult-ac)
  finally show ?thesis by simp
qed

lemma fourpow-gt-central-binomial:
  assumes n > 0
  shows   4 ^ n > ((2 * n) choose n)
proof −
  have (∑ i∈{..2*n}−{n}. ((2 * n) choose i)) > 0
    using assms by (intro sum-pos) (auto simp: subset-iff)
  hence ((2 * n) choose n) < (∑ i∈{..2*n}−{n}. ((2 * n) choose i)) + ((2 * n)
choose n)
    by simp
  also have ... = (∑ i ∈ insert n ({..2*n} − {n}). ((2 * n) choose i))
    by (subst sum.insert) auto
  also have insert n ({..2*n} − {n}) = {..2*n} by auto
  also have (∑ i≤2*n. ((2 * n) choose i)) = (1 + 1) ^ (2 * n)
    by (subst binomial) simp-all
```

also have ... = $4 \hat{} n$
by (*subst power-mult*) (*simp add*: *eval-nat-numeral*)
finally show *?thesis* **.**
qed

9.2 Lower bound for $\pi(x)$

context
includes *prime-counting-syntax*
fixes $S :: nat \Rightarrow nat \Rightarrow int$
defines $S \equiv (\lambda n\ p.\ (\sum m \in \{0<..nat\ \lfloor log\ p\ (2*n) \rfloor\}.\ \lfloor 2*n/p\hat{}m \rfloor - 2 * \lfloor n/p\hat{}m \rfloor))$
begin

We now first prove the bound $\pi(x) \geq \frac{1}{6} x / \ln x$ for $x \geq 2$. The constant could probably be improved for starting points greater than 2; this is true for most of the constants in this section.

The first step is to show a slightly stronger bound for even numbers, where the constant is $\frac{1}{2} \ln 2 \approx 0.347$:

lemma
fixes $n :: nat$
assumes n: $n \geq 1$
shows π-*bounds-aux*: *ln* (*fact* (2 * *n*)) − 2 * *ln* (*fact n*) =
prime-sum-upto (λ*p*. *S n p* * *ln p*) (2 * *n*)
and π-*lower-bound-ge-strong*: π (2 * *n*) ≥ *ln* 2 / 2 * (2 * *n*) / *ln* (2 * *n*)
proof −
define $L :: real \Rightarrow nat \Rightarrow nat$ **where** $L = (\lambda x\ p.\ legendre\text{-}aux\ x\ p)$
have *ln* (*fact* (2 * *n*)) − 2 * *ln* (*fact n*) = *sum-upto ln* (2 * *n*) − 2 * *sum-upto ln n*
by (*simp add*: *ln-fact-conv-sum-upto*)
also have ... = *prime-sum-upto* (λ*p*. *L* (2 * *n*) *p* * *ln p*) (2 * *n*) −
2 * *prime-sum-upto* (λ*p*. *L n p* * *ln p*) *n*
by (*subst* (*1 2*) *legendre-identity*) (*auto simp*: *L-def*)
also have *prime-sum-upto* (λ*p*. *L n p* * *ln p*) *n* = *prime-sum-upto* (λ*p*. *L n p* * *ln p*) (2 * *n*)
unfolding *prime-sum-upto-altdef2*
by (*intro sum.mono-neutral-left*[*OF finite-subset*[*of* - {..2*n}]])
(*auto dest*: *prime-gt-0-nat legendre-aux-posD*
simp: *legendre-aux-eq-0 L-def le-nat-iff le-floor-iff*)
also have *prime-sum-upto* (λ*p*. *L* (2*n) *p* * *ln p*) (2*n) −
2 * *prime-sum-upto* (λ*p*. *L n p* * *ln p*) (2 * *n*) =
prime-sum-upto (λ*p*. (*real* (*L* (2*n) *p*) − 2 * *real* (*L n p*)) * *ln p*) (2 * *n*)
by (*simp add*: *ring-distribs sum-subtractf sum-distrib-left mult.assoc prime-sum-upto-def*)
also have ... = *prime-sum-upto* (λ*p*. *of-int* (*S n p*) * *ln p*) (2*n)
unfolding *prime-sum-upto-def*
proof (*intro sum.cong refl*, *goal-cases*)
case (*1 p*)
define *ub* **where** *ub* = *nat* ⌊*log p* (2*n)⌋
from *1* **have** *p*: *prime p p* > *1 p* ≤ 2 * *n*

```
    using prime-gt-1-nat[of p] by auto
  have L (2 * n) p = (∑ m∈{0<..ub}. nat ⌊2*n/p^m⌋)
    unfolding L-def legendre-aux-altdef1 using p 1 by (auto simp: ub-def)
  moreover have L n p = (∑ m∈{0<..ub}. nat ⌊n/p^m⌋) unfolding L-def
  proof (intro legendre-aux-altdef2)
    have real n = real p powr log p n
      using n p by simp
    also have log (real p) 2 > 0 using p by auto
    hence log p n < 1 + of-int ⌊log p 2 + log p n⌋ by linarith
    hence real p powr log p n < real p powr Suc ub
     unfolding ub-def using n p by (intro powr-less-mono) (auto simp: log-mult)
    also have ... = p ^ Suc ub
      using p by (subst powr-realpow) auto
    finally show real n < real p ^ Suc ub by simp
  qed (use n p in auto)
  ultimately have real (L (2 * n) p) - 2 * real (L n p) =
                  (∑ m∈{0<..ub}. real (nat ⌊2*n/p^m⌋) - 2 * real (nat ⌊n/p^m⌋))
    by (simp add: sum-subtractf sum-distrib-left)
  also have ... = of-int (∑ m∈{0<..ub}. ⌊2*n/p^m⌋ - 2 * ⌊n/p^m⌋)
    unfolding of-int-sum by (intro sum.cong) auto
  finally show ?case by (simp add: ub-def S-def)
qed
finally show eq: ln (fact (2 * n)) - 2 * ln (fact n) =
                   prime-sum-upto (λp. S n p * ln p) (2 * n) .

have S-nonneg: S n p ≥ 0 for p
  unfolding S-def by (intro sum-nonneg) linarith
have S-le: S n p ≤ ⌊log p (2*n)⌋ if prime p for p
proof -
  have S n p ≤ (∑ m∈{0<..nat ⌊log p (2*n)⌋}. 1)
    unfolding S-def of-nat-mult of-nat-numeral by (intro sum-mono) linarith
  thus ?thesis using prime-gt-1-nat[of p] that n by auto
qed

have n * ln 2 = ln (real (2 ^ n))
  by (simp add: ln-realpow)
also have ... ≤ ln (real ((2*n) choose n))
  using twopow-le-central-binomial[of n]
  by (subst ln-le-cancel-iff; (unfold of-nat-le-iff)?) auto
also have ... = ln (fact (2 * n)) - 2 * ln (fact n)
  by (simp add: binomial-fact ln-div ln-mult)
also have ... = prime-sum-upto (λp. S n p * ln p) (2 * n)
  by (fact eq)
also have ... ≤ prime-sum-upto (λp. ⌊log p (2*n)⌋ * ln p) (2 * n)
  unfolding prime-sum-upto-def using S-le
  by (intro sum-mono mult-right-mono) (auto dest: prime-gt-0-nat)
also have ... ≤ prime-sum-upto (λp. ln (2 * n)) (2 * n)
  unfolding prime-sum-upto-def
proof (intro sum-mono)
```

```
    fix p assume p ∈ {p. prime p ∧ real p ≤ real (2 * n)}
    hence p: p > 1 using prime-gt-1-nat[of p] by auto
    have real-of-int ⌊log p (2 * n)⌋ = real-of-int ⌊ln (2 * n) / ln p⌋
      using p n by (simp add: log-def ln-mult)
    also have ... ≤ ln (2 * n) / ln p
      by linarith
    also have ... * ln p = ln (2 * n)
      using p by (simp add: field-simps)
    finally show real-of-int ⌊log p (2 * n)⌋ * ln p ≤ ln (2 * n)
      using p by simp
  qed
  also have ... = ln (2 * n) * π (2 * n)
    by (simp add: π-def prime-sum-upto-def)
  finally show π (2 * n) ≥ (ln 2 / 2) * (2 * n) / ln (2 * n)
    using n by (simp add: field-simps)
qed

lemma ln-2-ge-56-81: ln 2 ≥ (56 / 81 :: real)
  using ln-approx-bounds[of 2 2, simplified, simplified eval-nat-numeral, simplified]
by simp
```

The bound for any real number $x \geq 2$ follows fairly easily, although some ugly accounting for error terms has to be done.

```
theorem π-lower-bound:
  fixes x :: real
  assumes x: x ≥ 2
  shows   π x > (1 / 6) * (x / ln x)
proof (cases even (nat ⌊x⌋))
  case True
  define n where n = nat ⌊x⌋
  from True assms have n: n ≥ 2 even n
    by (auto simp: n-def le-nat-iff le-floor-iff)
  have (1 / 6) * (x / ln x) < (ln 2 / 4) * (x / ln x)
    using ln-2-ge-56-81 x by (intro mult-strict-right-mono) auto
  also have ln 2 / 4 * (x / ln x) = (1 / 2) * (ln 2 / 2 * x / ln x)
    by simp
  also have ... ≤ (1 - 1 / x) * (ln 2 / 2 * x / ln x)
    by (intro mult-right-mono) (use assms in ‹auto simp: field-simps›)
  also have (1 - 1 / x) * (ln 2 / 2 * x / ln x) = ln 2 / 2 * (x - 1) / ln x
    using assms by (simp add: field-simps)
  also have ln 2 / 2 * (x - 1) / ln x ≤ ln 2 / 2 * n / ln n
    using x by (intro frac-le mult-mono mult-nonneg-nonneg) (auto simp: n-def)
  also have ln 2 / 2 * n / ln n ≤ π n
    using π-lower-bound-ge-strong[of n div 2] ‹even n› n by simp
  also have π n = π x by (simp add: n-def)
  finally show ?thesis .
next
  case False
  define n where n = nat ⌊x⌋
```

```
  from False assms have n: n ≥ 2 odd n
    by (auto simp: n-def le-nat-iff le-floor-iff)
  then obtain k where [simp]: n = 2 * k + 1
    by (auto elim!: oddE)
  from n have k: k > 0 by simp

  from k have 3 ≤ real n by simp
  also have real n ≤ x unfolding n-def using x by linarith
  finally have x ≥ 3 .

  have (1 / 6) * (x / ln x) = 1 / 6 * x / ln x
    using x by (simp add: field-simps)
  also have 1 / 6 * x / ln x < ln 2 / 2 * (2 * k) / ln (2 * k)
  proof (intro frac-less)
    have x < real n + 1 unfolding n-def by linarith
    hence 1 / 6 * x < 1 / 6 * (n + 1) by simp
    also {
      have *: (3 * ln 2 - 1 :: real) ≥ 1
        using ln-2-ge-56-81 by simp
      hence 1 / (3 * ln 2 - 1 :: real) ≤ 1 by simp
      also have 1 ≤ real k using k by simp
      finally have 1 / 6 * (n + 1) ≤ ln 2 / 2 * real (2 * k)
        using * by (simp add: field-simps)
    }
    finally show 1 / 6 * x < ln 2 / 2 * real (2 * k) .
  next
    have real (2 * k) ≤ real n by simp
    also have ... ≤ x using x unfolding n-def by linarith
    finally show ln (real (2 * k)) ≤ ln x using k by simp
  qed (use k x in auto)
  also have ln 2 / 2 * (2 * k) / ln (2 * k) ≤ π (2 * k)
    by (rule π-lower-bound-ge-strong) (use ‹k > 0› in auto)
  also have π (2 * k) ≤ π n
    by (rule π-mono) auto
  also have ... = π x unfolding n-def by simp
  finally show ?thesis .
qed

lemma π-at-top: filterlim primes-pi at-top at-top
proof (rule filterlim-at-top-mono)
  show eventually (λx. primes-pi x ≥ 1 / 6 * (x / ln x)) at-top
   using eventually-gt-at-top[of 2] by eventually-elim (intro less-imp-le π-lower-bound)
qed real-asymp
```

9.3 Upper bound for $\vartheta(x)$

In this section, we prove a linear upper bound for ϑ. This is somewhat unnecessary because we already have a considerably better bound on $\vartheta(x)$ using a proof that has roughly the same complexity as this one and also

only uses elementary means. Nevertheless, here is the proof from Apostol's book; it is quite nice and it would be a shame not to formalise it.

The idea is to first show a bound for $\vartheta(2n) - \vartheta(n)$ and then deduce one for $\vartheta(2^n)$ from this by telescoping, which then yields one for general x by monotonicity.

```
lemma ϑ-double-less:
  fixes n :: nat
  assumes n: n > 0
  shows ϑ (2 * real n) - ϑ (real n) < real n * ln 4
proof (cases n ≥ 2)
  case False
  with assms have n = 1 by force
  moreover have ϑ 2 = ln 2 by (simp add: eval-ϑ)
  ultimately show ?thesis by auto
next
  define P where P = (λn::nat. {p∈{0<..n}. prime p})
  have ϑ-eq: ϑ n = (∑p∈P n. ln p) for n
    unfolding ϑ-def prime-sum-upto-def
    by (intro sum.cong) (auto simp: P-def dest: prime-gt-0-nat)

  have ϑ (2 * n) - ϑ n = (∑p ∈ P (2*n) - P n. ln p)
    unfolding ϑ-eq by (rule Groups-Big.sum-diff [symmetric]) (auto simp: P-def)

  also have (∑p∈P (2*n) - P n. ln p) =
        (∑p∈P (2*n) - P n. (⌊2*n/p⌋ - 2 * ⌊n/p⌋) * ln p)
  proof (intro sum.cong refl)
    fix p assume p: p ∈ P (2*n) - P n
    hence *: real n / real p < 1 real n / real p ≥ 1 / 2 by (auto simp: P-def)
    from * have ⌊real n / real p⌋ = 0 by linarith
    moreover from * have ⌊2 * real n / real p⌋ = 1
      by linarith
    ultimately show ln p = (⌊2*n/p⌋ - 2 * ⌊n/p⌋) * ln p
      by simp
  qed

  also have (∑p∈P (2*n) - P n. (⌊2*n/p⌋ - 2 * ⌊n/p⌋) * ln p) ≤
        (∑p∈P (2*n). (⌊2*n/p⌋ - 2 * ⌊n/p⌋) * ln p)
  proof (intro sum-mono2)
    fix p assume p: p ∈ P (2 * n) - (P (2 * n) - P n)
    have 2 * ⌊real n / real p⌋ ≤ ⌊2 * (real n / real p)⌋
      by linarith
    thus 0 ≤ real-of-int (⌊real (2 * n) / real p⌋ - 2 * ⌊real n / real p⌋) * ln (real
p)
      using p by (intro mult-nonneg-nonneg) (auto simp: P-def)
  qed (auto simp: P-def)

  also have *: 2 * real-of-int ⌊real n / real p ^ m⌋ ≤ 2 * real n / real p ^ m for
p m
```

```
    by linarith
  have (∑ m∈{1}. ⌊2 * real n / real p ^ m⌋ - 2 * ⌊n / real p^m⌋) ≤ S n p
    if prime p p ≤ 2 * n for p :: nat
    unfolding S-def using prime-gt-1-nat[OF that(1)] that(2) n *[of p]
    by (intro sum-mono2) (auto dest: prime-gt-1-nat simp: le-nat-iff le-floor-iff)
  hence (∑ p∈P (2*n). (∑ m∈{1}. ⌊2*n/p^m⌋ - 2 * ⌊n/p^m⌋) * ln p) ≤ (∑ p∈P
(2*n). S n p * ln p)
    by (intro sum-mono mult-right-mono; (unfold of-int-le-iff)?)
      (auto dest: prime-gt-1-nat simp: P-def)
  hence (∑ p∈P (2*n). (⌊2*n/p⌋ - 2 * ⌊n/p⌋) * ln p) ≤ (∑ p∈P (2*n). S n p
* ln p)
    by simp

  also have (∑ p ∈ P (2*n). S n p * ln p) = prime-sum-upto (λp. S n p * ln p)
(2 * n)
    unfolding P-def prime-sum-upto-def by (intro sum.cong) (auto simp: P-def
dest: prime-gt-0-nat)
  also have ... = ln (fact (2 * n)) - 2 * ln (fact n)
    by (rule π-bounds-aux [symmetric]) (use n in auto)
  also have ... = ln (real ((2*n) choose n))
    by (simp add: binomial-fact ln-div ln-mult)
  also have ... < ln (real (4 ^ n))
    by (subst ln-less-cancel-iff; (unfold of-nat-le-iff)?)
      (use fourpow-gt-central-binomial[of n] n in auto)
  also have ... = n * ln 4
    by (simp add: ln-realpow)
  finally show ?thesis by simp
qed

lemma ϑ-twopow-less: ϑ (2 ^ r) < 2 ^ (r + 1) * ln 2
proof -
  have ϑ-twopow-diff: ϑ (2 ^ Suc k) - ϑ (2 ^ k) < 2 ^ Suc k * ln 2 for k
    using ϑ-double-less[of 2 ^ k] ln-realpow[of 2 2] by simp
  show ϑ (2 ^ r) < 2 ^ (r + 1) * ln 2
  proof (cases r > 0)
    case True
    have (∑ k<r. ϑ (2 ^ Suc k) - ϑ (2 ^ k)) < (∑ k<r. 2 ^ Suc k * ln 2)
      by (intro sum-strict-mono ϑ-twopow-diff) (use ‹r > 0› in auto)
    also have (∑ k<r. ϑ (2 ^ Suc k) - ϑ (2 ^ k)) = ϑ (2 ^ r)
      by (subst sum-lessThan-telescope) auto
    also have (∑ k<r. 2 ^ Suc k * ln 2 :: real) = (∑ k<r. 2 ^ k) * 2 * ln 2
      by (simp add: sum-distrib-right sum-distrib-left mult-ac)
    also have (∑ k<r. 2 ^ k :: real) = 2 ^ r - 1
      using geometric-sum[of 2 :: real] by simp
    also have ... ≤ 2 ^ r by simp
    finally show ϑ (2 ^ r) < 2 ^ (r + 1) * ln 2
      by simp
  qed auto
qed
```

theorem *ϑ-upper-bound-weak*:
fixes *n* :: *nat*
assumes *n*: *n* > *0*
shows *ϑ n* < *4* ∗ *ln 2* ∗ *n*
proof −
define *r* **where** *r* = *floor-log n*
have *ϑ n* ≤ *ϑ* (*real* (*2* ^ *Suc r*))
unfolding *r-def* **using** *floor-log-exp2-ge*[*of n*] **by** (*intro ϑ-mono*, *unfold of-nat-le-iff*) *auto*
also have ... < *4* ∗ *ln 2* ∗ *real* (*2* ^ *r*)
using *ϑ-twopow-less*[*of r* + *1*] **by** (*simp add*: *mult-ac*)
also have ... ≤ *4* ∗ *ln 2* ∗ *real n* **unfolding** *r-def*
by (*intro mult-left-mono*, *unfold of-nat-le-iff*, *intro floor-log-exp2-le*) (*use n* **in** *auto*)
finally show *ϑ n* < *4* ∗ *ln 2* ∗ *n* **by** *simp*
qed

9.4 Upper bound for $\pi(x)$

We use our upper bound for $\vartheta(x)$ (the strong one, not the one from the previous section) to derive an upper bound for $\pi(x)$.

As a first step, we show the following lemma about the global maximum of the function $\ln x/x^c$ for $c > 0$:

lemma *π-upper-bound-aux*:
fixes *c* :: *real*
assumes *c* > *0*
defines *f* ≡ (*λx*. *x powr* (−*c*) ∗ *ln x*)
assumes *x*: *x* > *0*
shows *f x* ≤ *1* / (*c* ∗ *exp 1*)
proof −
define *f′* **where** *f′* = (*λx*. *x powr* (−*c* − *1*) ∗ (*1* − *c* ∗ *ln x*))
define *z* **where** *z* = *exp* (*1* / *c*)
have *z* > *0* **by** (*simp add*: *z-def*)
have *deriv*: (*f has-real-derivative f′ t*) (*at t*) **if** *t* > *0* **for** *t*
unfolding *f-def f′-def* **using** *that*
by (*auto intro*!: *derivative-eq-intros simp*: *field-simps powr-diff powr-minus*)
have [*simp*]: *f z* = *1* / (*c* ∗ *exp 1*)
by (*simp add*: *z-def f-def powr-def exp-minus field-simps*)

show *?thesis*
proof (*cases x z rule*: *linorder-cases*)
assume *x*: *x* < *z*
from *x assms* **have** *t*: ∃ *t*. *t* > *x* ∧ *t* < *z* ∧ *f z* − *f x* = (*z* − *x*) ∗ *f′ t*
by (*intro MVT2 deriv*) *auto*
then obtain *t* **where** *t*: *t* > *x t* < *z f z* − *f x* = (*z* − *x*) ∗ *f′ t*
by *blast*
hence *ln t* < *ln z* **using** *assms* **by** *simp*

```
    also have ln z = 1 / c by (simp add: z-def)
    finally have 0 ≤ (z − x) * f' t
      unfolding f'-def using ‹c > 0› x
      by (intro mult-nonneg-nonneg) (auto simp: z-def field-simps)
    also from t have ... = f z − f x by (simp add: algebra-simps)
    finally show ?thesis by simp
  next
    assume x: x > z
    from x assms ‹z > 0› have t: ∃ t. t > z ∧ t < x ∧ f x − f z = (x − z) * f' t
      by (intro MVT2 deriv) auto
    then obtain t where t: t > z t < x f x − f z = (x − z) * f' t
      by blast
    hence ln z < ln t using ‹z > 0› assms by simp
    also have ln z = 1 / c by (simp add: z-def)
    finally have 0 ≤ (z − x) * f' t
      unfolding f'-def using ‹c > 0› x
    by (intro mult-nonpos-nonpos mult-nonneg-nonpos) (auto simp: z-def field-simps)
    also from t have ... = f z − f x by (simp add: algebra-simps)
    finally show ?thesis by simp
  qed auto
qed
```

Following Apostol, we first show a generic bound depending on some real-valued parameter α:

```
lemma π-upper-bound-strong:
  fixes α :: real and n :: nat
  assumes n: n ≥ 2 and α: α ∈ {0<..<1}
  shows π n < (1 / ((1 − α) * exp 1) + ln 4 / α) * n / ln n
proof −
  have real n powr α ≤ real n powr 1
    using assms n by (intro powr-mono) auto
  hence n': real n powr α ≤ real n using n by simp

  define P where P = (λx. {p. prime p ∧ real p ≤ x})
  define Q where Q = {p. prime p ∧ real p ∈ {n powr α<..n}}

  have finite-P [intro]: finite (P x) for x
  proof (cases x ≥ 0)
    case True
    hence P x ⊆ {..nat ⌊x⌋}
      by (auto simp: le-nat-iff le-floor-iff P-def)
    thus ?thesis by (rule finite-subset) auto
  qed (auto simp: P-def)

  have P-subset: P x ⊆ P y if x ≤ y for x y
    using that by (auto simp: P-def)

  have Q = P n − P (n powr α) by (auto simp: Q-def P-def)
  also have card ... = card (P n) − card (P (n powr α))
```

```
    by (intro card-Diff-subset finite-P P-subset n')
  also have real ... = π n - π (n powr α)
    by (subst of-nat-diff[OF card-mono[OF - P-subset]])
       (use n' in ‹auto simp: π-def prime-sum-upto-def P-def›)
  finally have card-Q: real (card Q) = π n - π (n powr α) .

  have (π n - π (n powr α)) * ln (n powr α) = (∑p∈Q. ln (n powr α))
    using card-Q by simp
  also have ... ≤ (∑p∈Q. ln p)
    using n α by (intro sum-mono, subst ln-le-cancel-iff) (auto simp: Q-def dest:
prime-gt-0-nat)
  also have ... ≤ ϑ n
    unfolding ϑ-def prime-sum-upto-def by (intro sum-mono2) (auto simp: Q-def
dest: prime-gt-1-nat)
  also have ... < ln 4 * real n
    by (rule ϑ-upper-bound) (use n in auto)
  finally have ineq: (π n - π (n powr α)) * ln (n powr α) < ln 4 * n .

  with n assms have π n < π (n powr α) + (ln 4 / α) * n / ln n
    by (simp add: field-simps ln-powr
          del: div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1)
  also have π (n powr α) ≤ n powr α
    by (rule π-le-self) auto
  also have n powr α + ln 4 / α * n / ln n =
            (n powr (-(1 - α)) * ln n + ln 4 / α) * n / ln n
    using n α by (simp add: field-simps powr-diff
                  del: div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1)
  also have n powr (-(1 - α)) * ln n ≤ 1 / ((1 - α) * exp 1)
    by (intro π-upper-bound-aux) (use α n in auto)
  hence (n powr (-(1 - α)) * ln n + ln 4 / α) * n / ln n ≤
        (1 / ((1 - α) * exp 1) + ln 4 / α) * n / ln n
    using n α by (intro divide-right-mono mult-right-mono add-mono) auto
  finally show π n < (1 / ((1 - α) * exp 1) + ln 4 / α) * n / ln n
    by simp
qed
```

The choice $\alpha := \frac{2}{3}$ then leads to the upper bound $\pi(x) < cx/\ln x$ with $c = 3(e^{-1} + \ln 2) \approx 3.183$. This is considerably stronger than Apostol's bound.

```
theorem π-upper-bound:
  fixes x :: real
  assumes x ≥ 2
  shows   π x < 3 * (exp (-1) + ln 2) * x / ln x
proof (cases x ≥ 3)
  case False
  have π x = π (nat ⌊x⌋) by simp
  also from False and assms have nat ⌊x⌋ = 2
    by linarith
  finally have π x = 1 by (simp add: eval-π)
```

```
  also have ... < 3 * (exp (-1) + ln 2) * exp 1
    by (simp add: exp-minus field-simps add-pos-pos
          del: div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1)
  also have ... ≤ 3 * (exp (-1) + ln 2) * (x / ln x)
    using π-upper-bound-aux[of 1 x]
    by (intro mult-left-mono) (use assms in ‹auto simp: field-simps powr-minus›)
  finally show ?thesis
    using assms by (simp add: field-simps)
next
  case True
  define n where n = nat ⌊x⌋
  from True have n: n ≥ 3 by (simp add: n-def le-nat-iff le-floor-iff)
  have π x = π n
    by (simp add: n-def)
  also have π n < 3 * (exp (-1) + ln 2) * (n / ln n)
    using π-upper-bound-strong[of n 2 / 3] ln-realpow[of 2 2] n
    by (simp add: field-simps exp-minus
          del: div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1)
  also have ... ≤ 3 * (exp (-1) + ln 2) * (x / ln x)
    using n True by (intro mult-left-mono divide-ln-mono) (auto simp: n-def)
  finally show ?thesis by (simp add: divide-simps)
qed

corollary π-upper-bound':
  fixes x :: real
  assumes x ≥ 2
  shows   π x < 443 / 139 * (x / ln x)
proof -
  have 2.71828 ≤ 5837465777 / 2147483648 - inverse (2 ^ 32 :: real)
    by simp
  also have ... ≤ exp (1 :: real)
    using e-approx-32 by linarith
  finally have exp 1 ≥ (2.71828 :: real) .
  hence e-m1: exp (-1) ≤ (10^5 / 271828 :: real) by (simp add: field-simps
exp-minus)

  from assms have π x < 3 * (exp (-1) + ln 2) * (x / ln x)
    using π-upper-bound[of x] by (simp add: field-simps)
  also have ... ≤ 443 / 139 * (x / ln x)
  proof (intro mult-right-mono)
    have 3 * (exp (-1) + ln 2 :: real) ≤ 3 * (10^5 / 271828 + 25 / 36)
      using e-m1 by (intro mult-left-mono add-mono ln2-le-25-over-36)
                    (auto simp: exp-minus field-simps abs-if split: if-splits)
    also have ... ≤ 443 / 139 by simp
    finally show 3 * (exp (- 1) + ln 2 :: real) ≤ 443 / 139 by simp
  qed (use assms in auto)
  finally show ?thesis .
qed
```

corollary *π-upper-bound''*:
fixes *x* :: *real*
assumes *x* ≥ *2*
shows *π x* < *4* ∗ (*x* / *ln x*)
by (*rule less-le-trans*[*OF π-upper-bound'*[*OF assms*] *mult-right-mono*]) (*use assms* **in** *auto*)

In particular, we have now shown a weak version of the Prime Number Theorem, namely that $\pi(x) \in \Theta(x/\ln x)$:

lemma *π-bigtheta*: *π* ∈ Θ(λ*x*. *x* / *ln x*)
proof
have *eventually* (λ*x*. |*π x*| ≤ *3* ∗ (*exp* (− *1*) + *ln 2*) ∗ |*x* / *ln x*|) *at-top*
using *eventually-ge-at-top*[*of 2*]
by *eventually-elim* (*use π-upper-bound* **in** ‹*auto intro*!: *less-imp-le*›)
thus *π* ∈ *O*(λ*x*. *x* / *ln x*)
by (*intro bigoI*[**where** *c* = *3* ∗ (*exp* (− *1*) + *ln 2*)]) *auto*
next
have *eventually* (λ*x*. |*π x*| ≥ *1* / *6* ∗ |*x* / *ln x*|) *at-top*
using *eventually-ge-at-top*[*of 2*]
by *eventually-elim* (*use π-lower-bound* **in** ‹*auto intro*!: *less-imp-le*›)
thus *π* ∈ Ω(λ*x*. *x* / *ln x*)
by (*intro landau-omega.bigI*[**where** *c* = *1* / *6*]) *auto*
qed

9.5 Bounds for p_n

By some rearrangements, the lower and upper bounds for $\pi(x)$ give rise to analogous bounds for p_n:

lemma *nth-prime-lower-bound-gen*:
assumes *c*: *c* > *0* **and** *n*: *n* > *0*
assumes ⋀*n*. *n* ≥ *2* ⟹ *π* (*real n*) < (*1* / *c*) ∗ (*real n* / *ln* (*real n*))
shows *nth-prime* (*n* − *1*) ≥ *c* ∗ (*real n* ∗ *ln* (*real n*))
proof −
define *p* **where** *p* = *nth-prime* (*n* − *1*)
have *p* ≥ *2*
by (*simp add*: *p-def nth-prime-ge-2*)
have *p* ≥ *n* **using** *nth-prime-lower-bound*[*of n* − *1*] **by** (*simp add*: *p-def*)

have *c* ∗ (*n* ∗ *ln n*) ≤ *c* ∗ (*n* ∗ *ln p*)
using *n c* ‹*p* ≥ *n*› **by** (*intro mult-left-mono*) *auto*
also **{**
from ‹*p* ≥ *2*› **have** *π* (*real p*) < (*1* / *c*) ∗ (*real p* / *ln* (*real p*))
by (*rule assms*)
also from *n* **have** *π* (*real p*) = *n*
by (*simp add*: *p-def*)
finally have *c* ∗ (*n* ∗ *ln p*) < *p*
using *c* ‹*p* ≥ *2*› *n* **by** (*simp add*: *field-simps*)
}

```
  finally show nth-prime (n − 1) ≥ c * (real n * ln (real n))
    using c n by (simp add: p-def)
qed

corollary nth-prime-lower-bound:
  n > 0 ⟹ nth-prime (n − 1) ≥ (139 / 443) * (n * ln n)
  using π-upper-bound' by (intro nth-prime-lower-bound-gen) auto

corollary nth-prime-upper-bound:
  assumes n: n > 0
  shows   nth-prime (n − 1) < 12 * (n * ln n + n * ln (12 / exp 1))
proof −
  define p where p = nth-prime (n − 1)
  have p ≥ 2
    by (simp add: p-def nth-prime-ge-2)

  have (1 / 6) * (p / ln p) < π p
    by (intro π-lower-bound) (use ‹p ≥ 2› in auto)
  also have ... = n
    using n by (simp add: p-def)
  finally have less: p < 6 * n * ln p
    using ‹p ≥ 2› by (simp add: field-simps)

  also have ln p ≤ (2 / exp 1) * sqrt p
    using π-upper-bound-aux[of 1 / 2 p] ‹p ≥ 2›
    by (simp add: field-simps powr-minus powr-half-sqrt)
  finally have sqrt p * sqrt p < 12 / exp 1 * n * sqrt p
    using n by simp
  hence sqrt p < 12 / exp 1 * n
    by (subst (asm) mult-less-cancel-right) (use ‹p ≥ 2› in auto)
  hence ln (sqrt p) < ln (12 / exp 1 * n)
    using n ‹p ≥ 2› by (subst ln-less-cancel-iff) auto
  also have ln (sqrt p) = ln p / 2
    using ‹p ≥ 2› by (simp add: ln-sqrt)
  also have ln (12 / exp 1 * n) = ln n + ln (12 / exp 1)
    using n by (simp add: ln-div ln-mult)
  finally have ln-less: ln p ≤ 2 * ln n + 2 * ln (12 / exp 1)
    by simp

  have p < 6 * n * ln p by (fact less)
  also have ... ≤ 6 * n * (2 * ln n + 2 * ln (12 / exp 1))
    by (intro mult-left-mono ln-less) auto
  also have ... = 12 * (n * ln n + n * ln (12 / exp 1))
    by (simp add: algebra-simps)
  finally show ?thesis unfolding p-def .
qed
```

We can thus also conclude that $p_n \sim n \ln n$:

```
corollary nth-prime-bigtheta: nth-prime ∈ Θ(λn. n * ln n)
```

proof
have *eventually* (λ*n*. |*nth-prime n*| ≤
12 ∗ |(*n* + *1*) ∗ *ln* (*n* + *1*) + (*n* + *1*) ∗ *ln* (*12* / *exp 1*)|) *at-top*
using *eventually-ge-at-top*[*of 2*]
proof *eventually-elim*
case (*elim n*)
with *nth-prime-upper-bound*[*of n* + *1*] **show** *?case* **by** (*auto simp*: *add-ac*)
qed
hence *nth-prime* ∈ *O*(λ*n*. (*n* + *1*) ∗ *ln* (*n* + *1*) + (*n* + *1*) ∗ *ln* (*12* / *exp 1*))
by (*intro bigoI*[**where** *c* = *12*]) *auto*
also have (λ*n*. (*n* + *1*) ∗ *ln* (*n* + *1*) + (*n* + *1*) ∗ *ln* (*12* / *exp 1*)) ∈ *O*(λ*n*::*nat*. *n* ∗ *ln n*)
by *real-asymp*
finally show *nth-prime* ∈ *O*(λ*n*. *n* ∗ *ln n*) **.**
next
have *eventually* (λ*n*. |*nth-prime n*| ≥ *139* / *443* ∗ |(*n* + *1*) ∗ *ln* (*n* + *1*)|) *at-top*
using *eventually-ge-at-top*[*of 2*]
proof *eventually-elim*
case (*elim n*)
with *nth-prime-lower-bound*[*of n* + *1*] **show** *?case* **by** (*auto simp*: *add-ac*)
qed
hence *nth-prime* ∈ Ω(λ*n*::*nat*. *real* (*n* + *1*) ∗ *ln* (*real n* + *1*))
by (*intro landau-omega.bigI*[**where** *c* = *139* / *443*]) (*auto simp*: *add-ac*)
also have (λ*n*::*nat*. *real* (*n* + *1*) ∗ *ln* (*real n* + *1*)) ∈ Ω(λ*n*. *n* ∗ *ln n*)
by *real-asymp*
finally show *nth-prime* ∈ Ω(λ*n*. *n* ∗ *ln n*) **.**
qed

end

end

10 The asymptotics of the summatory divisor σ function

theory *Summatory-Divisor-Sigma-Bounds*
imports *Partial-Zeta-Bounds More-Dirichlet-Misc*
begin

In this section, we analyse the asymptotic behaviour of the summatory divisor functions $\sum_{n\leq x} \sigma_\alpha(n)$ for real α. This essentially tells us what the average value of these functions is for large x.

The case $\alpha = 0$ is not treated here since σ_0 is simply the divisor function, for which precise asymptotics are already available in the AFP.

10.1 Case 1: $\alpha = 1$

If $\alpha = 1$, $\sigma_\alpha(n)$ is simply the sum of all divisors of n. Here, the asymptotics is

$$\sum_{n \leq x} \sigma_1(n) = \frac{\pi^2}{12} x^2 + O(x \ln x) \ .$$

theorem *summatory-divisor-sum-asymptotics*:
sum-upto divisor-sum =o (λx. *pi*^{2} / *12* $*$ *x* ^ *2*) *+o* $O(\lambda x.\ x * \ln x)$
proof −
define ζ **where** ζ = *Re* (*zeta 2*)
define *R1* **where** *R1* = (λx. *sum-upto real x* − *x*^{2} / *2*)
define *R2* **where** *R2* = (λx. *sum-upto* (λd. *1* / *d*^{2}) *x* − (ζ − *1* / *x*))
obtain *c1* **where** *c1*: *c1* > *0* $\bigwedge x.\ x \geq 1 \Longrightarrow |R1\ x| \leq c1 * x$
using *zeta-partial-sum-le-neg*[*of 1*] **by** (*auto simp*: *R1-def*)
obtain *c2* **where** *c2*: *c2* > *0* $\bigwedge x.\ x \geq 1 \Longrightarrow |R2\ x| \leq c2$ / *x*^{2}
using *zeta-partial-sum-le-pos*[*of 2*]
by (*auto simp*: ζ*-def R2-def powr-minus field-simps*
simp del: *div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1*)

have *le*: |*sum-upto divisor-sum x* − ζ / *2* $*$ *x*^{2}| ≤ *c2* / *2* + *x* / *2* + *c1* $*$ *x* $*$ (*ln x* + *1*)
if *x*: *x* ≥ *1* **for** *x*
proof −
have *div-le*: *real* (*a div b*) ≤ *x* **if** *a* ≤ *x* **for** *a b* :: *nat*
by (*rule order.trans*[*OF - that*(*1*)]) *auto*

have *real* (*sum-upto divisor-sum x*) = *sum-upto* (*dirichlet-prod real* (λ-. *1*)) *x*
by (*simp add*: *divisor-sigma-conv-dirichlet-prod* [*abs-def*]
sum-upto-def divisor-sigma-1-left [*symmetric*])
also have ... = *sum-upto* (λn. $\sum d \mid d\ dvd\ n$. *real d*) *x*
by (*simp add*: *dirichlet-prod-def*)
also have ... = ($\sum$ (*n*, *d*) ∈ (*SIGMA n*:{*n*. *n* > *0* ∧ *real n* ≤ *x*}. {*d*. *d dvd n*}). *real d*)
unfolding *sum-upto-def* **by** (*subst sum.Sigma*) *auto*
also have ... = ($\sum$ (*d*, *q*) ∈ (*SIGMA d*:{*d*. *d* > *0* ∧ *real d* ≤ *x*}. {*q*. *q* > *0* ∧ *real q* ≤ *x* / *d*}). *real q*)
by (*rule sum.reindex-bij-witness*[*of* - λ(*d*, *q*). (*d* $*$ *q*, *q*) λ(*n*, *d*). (*n div d*, *d*)])
(*use div-le* **in** ‹*auto simp*: *field-simps*›)
also have ... = *sum-upto* (λd. *sum-upto real* (*x* / *d*)) *x*
by (*simp add*: *sum-upto-def sum.Sigma*)
also have ... = *x*^{2} $*$ *sum-upto* (λd. *1* / *d*^{2}) *x* / *2* + *sum-upto* (λd. *R1* (*x* / *d*)) *x*
by (*simp add*: *R1-def sum-upto-def sum.distrib sum-subtractf sum-divide-distrib*
power-divide sum-distrib-left)
also have *sum-upto* (λd. *1* / *d*^{2}) *x* = ζ − *1* / *x* + *R2 x*
by (*simp add*: *R2-def*)
finally have *eq*: *real* (*sum-upto divisor-sum x*) =
x^2 $*$ (ζ − *1* / *x* + *R2 x*) / *2* + *sum-upto* (λd. *R1* (*x* / *real d*))

```
x .

    have real (sum-upto divisor-sum x) − ζ / 2 ∗ x² =
          x² / 2 ∗ R2 x − x / 2 + sum-upto (λd. R1 (x / real d)) x using x
      by (subst eq)
        (simp add: field-simps power2-eq-square del: div-diff div-add
            del: div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1)
    also have |...| ≤ c2 / 2 + x / 2 + c1 ∗ x ∗ (ln x + 1)
   proof (rule order.trans[OF abs-triangle-ineq] order.trans[OF abs-triangle-ineq4]
add-mono)+
      have |x² / 2 ∗ R2 x| = x² / 2 ∗ |R2 x|
        using x by (simp add: abs-mult)
      also have ... ≤ x² / 2 ∗ (c2 / x²)
        using x by (intro mult-left-mono c2) auto
      finally show |x² / 2 ∗ R2 x| ≤ c2 / 2
        using x by simp
    next
      have |sum-upto (λd. R1 (x / real d)) x| ≤ sum-upto (λd. |R1 (x / real d)|) x
        unfolding sum-upto-def by (rule sum-abs)
      also have ... ≤ sum-upto (λd. c1 ∗ (x / real d)) x
        unfolding sum-upto-def by (intro sum-mono c1) auto
      also have ... = c1 ∗ x ∗ sum-upto (λd. 1 / real d) x
        by (simp add: sum-upto-def sum-distrib-left)
      also have sum-upto (λd. 1 / real d) x = harm (nat ⌊x⌋)
          unfolding sum-upto-altdef harm-def by (intro sum.cong) (auto simp:
field-simps)
      also have ... ≤ ln (nat ⌊x⌋) + 1
        by (rule harm-le) (use x in ‹auto simp: le-nat-iff›)
      also have ln (nat ⌊x⌋) ≤ ln x using x by simp
      finally show |sum-upto (λd. R1 (x / real d)) x| ≤ c1 ∗ x ∗ (ln x + 1)
        using c1(1) x by simp
    qed (use x in auto)
    finally show |sum-upto divisor-sum x − ζ / 2 ∗ x²| ≤ c2 / 2 + x / 2 + c1
∗ x ∗ (ln x + 1) .
  qed

  have eventually (λx. |sum-upto divisor-sum x − ζ / 2 ∗ x²| ≤
                        c2 / 2 + x / 2 + c1 ∗ x ∗ (ln x + 1)) at-top
    using eventually-ge-at-top[of 1] by eventually-elim (use le in auto)
  hence eventually (λx. |sum-upto divisor-sum x − ζ / 2 ∗ x²| ≤
                          |c2 / 2 + x / 2 + c1 ∗ x ∗ (ln x + 1)|) at-top
    by eventually-elim linarith
  hence (λx. sum-upto divisor-sum x − ζ / 2 ∗ x²) ∈ O(λx. c2 / 2 + x / 2 + c1
∗ x ∗ (ln x + 1))
    by (intro landau-o.bigI[of 1]) auto
  also have (λx. c2 / 2 + x / 2 + c1 ∗ x ∗ (ln x + 1)) ∈ O(λx. x ∗ ln x)
    by real-asymp
  finally show ?thesis
    by (subst set-minus-plus [symmetric])
```

(*simp-all add*: *fun-diff-def algebra-simps ζ-def zeta-even-numeral*)
qed

10.2 Case 2: $\alpha > 0$, $\alpha \neq 1$

Next, we consider the case $\alpha > 0$ and $\alpha \neq 1$. We then have:

$$\sum_{n \leq x} \sigma_\alpha(n) = \frac{\zeta(\alpha+1)}{\alpha+1} x^{\alpha+1} + O\left(x^{\max(1,\alpha)}\right)$$

theorem *summatory-divisor-sigma-asymptotics-pos*:
fixes α :: *real*
assumes α: $\alpha > 0$ $\alpha \neq 1$
defines $\zeta \equiv Re\ (zeta\ (\alpha + 1))$
shows *sum-upto* (*divisor-sigma* α) =*o*
(λ*x*. ζ / (α + 1) ∗ *x powr* (α + 1)) +*o* *O*(λ*x*. *x powr max* 1 α)
proof −
define *R1* **where** *R1* = (λ*x*. *sum-upto* (λ*d*. *real d powr* α) *x* − *x powr* (α + 1) / (α + 1))
define *R2* **where** *R2* = (λ*x*. *sum-upto* (λ*d*. *d powr* (−α − 1)) *x* − (ζ − *x powr* −α / α))
define *R3* **where** *R3* = (λ*x*. *sum-upto* (λ*d*. *d powr* −α) *x* − *x powr* (1 − α) / (1 − α))
obtain *c1* **where** *c1*: *c1* > 0 ⋀*x*. *x* ≥ 1 ⟹ |*R1 x*| ≤ *c1* ∗ *x powr* α
using *zeta-partial-sum-le-neg*[*of* α] α **by** (*auto simp*: *R1-def add-ac*)
obtain *c2* **where** *c2*: *c2* > 0 ⋀*x*. *x* ≥ 1 ⟹ |*R2 x*| ≤ *c2* ∗ *x powr* (−α−1)
using *zeta-partial-sum-le-pos*[*of* α + 1] α **by** (*auto simp*: *ζ-def R2-def*)
obtain *c3* **where** *c3*: *c3* > 0 ⋀*x*. *x* ≥ 1 ⟹ |*R3 x*| ≤ *c3*
using *zeta-partial-sum-le-pos'*[*of* α] α **by** (*auto simp*: *R3-def*)
define *ub* :: *real* ⇒ *real* **where**
ub = (λ*x*. *x* / (α ∗ (α + 1)) + *c2* / (α + 1) + *c1* ∗ (1 / (1 − α) ∗ *x* + *c3* ∗ *x powr* α))

have *le*: |*sum-upto* (*divisor-sigma* α) *x* − ζ / (α + 1) ∗ *x powr* (α + 1)| ≤ *ub x*
if *x*: *x* ≥ 1 **for** *x*
proof −
have *div-le*: *real* (*a div b*) ≤ *x* **if** *a* ≤ *x* **for** *a b* :: *nat*
by (*rule order.trans*[*OF* - *that*(1)]) *auto*

have *sum-upto* (*divisor-sigma* α) *x* =
sum-upto (*dirichlet-prod* (λ*n*. *real n powr* α) (λ-. 1)) *x*
by (*simp add*: *divisor-sigma-conv-dirichlet-prod* [*abs-def*]
sum-upto-def divisor-sigma-1-left [*symmetric*])
also have ... = *sum-upto* (λ*n*. ∑ *d* | *d dvd n*. *real d powr* α) *x*
by (*simp add*: *dirichlet-prod-def*)
also have ... = (∑ (*n*, *d*) ∈ (*SIGMA n*:{*n*. *n* > 0 ∧ *real n* ≤ *x*}. {*d*. *d dvd n*}). *real d powr* α)
unfolding *sum-upto-def* **by** (*subst sum.Sigma*) *auto*

also have ... = ($\sum$ (*d*, *q*) ∈ (*SIGMA d*:{*d*. *d* > 0 ∧ *real d* ≤ *x*}. {*q*. *q* > 0 ∧ *real q* ≤ *x* / *d*}). *real q powr* α)
by (*rule sum.reindex-bij-witness*[*of* - λ(*d*, *q*). (*d* * *q*, *q*) λ(*n*, *d*). (*n div d*, *d*)])
(*use div-le* **in** ‹*auto simp*: *field-simps*›)
also have ... = *sum-upto* (λ*d*. *sum-upto* (λ*q*. *q powr* α) (*x* / *d*)) *x*
by (*simp add*: *sum-upto-def sum.Sigma*)
also have ... = *x powr* (α + *1*) * *sum-upto* (λ*d*. *1* / *d powr* (α + *1*)) *x* / (α + *1*) +
sum-upto (λ*d*. *R1* (*x* / *d*)) *x*
by (*simp add*: *R1-def sum-upto-def sum.distrib sum-subtractf sum-divide-distrib*
powr-divide sum-distrib-left)
also have *sum-upto* (λ*d*. *1* / *d powr* (α + *1*)) *x* = ζ − *x powr* −α / α + *R2 x*
by (*simp add*: *R2-def powr-minus field-simps powr-diff powr-add*)
finally have *eq*: *sum-upto* (*divisor-sigma* α) *x* =
x powr (α + *1*) * (ζ − *x powr* −α / α + *R2 x*) / (α + *1*) + *sum-upto* (λ*d*. *R1* (*x* / *d*)) *x* **.**

have *sum-upto* (*divisor-sigma* α) *x* − ζ / (α + *1*) * *x powr* (α + *1*) =
−*x* / (α * (α + *1*)) + *x powr* (α + *1*) / (α + *1*) * *R2 x* + *sum-upto* (λ*d*. *R1* (*x* / *d*)) *x*
using *x* α
by (*subst eq*, *simp add*: *divide-simps*
del: *div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1*)
(*simp add*: *field-simps power2-eq-square powr-add powr-minus del*: *div-diff div-add*
del: *div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1*)
also have |...| ≤ *ub x* **unfolding** *ub-def*
proof (*rule order.trans*[*OF abs-triangle-ineq*] *order.trans*[*OF abs-triangle-ineq4*] *add-mono*)+
have |*x powr* (α + *1*) / (α + *1*) * *R2 x*| = *x powr* (α + *1*) / (α + *1*) * |*R2 x*|
using *x* α **by** (*simp add*: *abs-mult*)
also have ... ≤ *x powr* (α + *1*) / (α + *1*) * (*c2* * *x powr* (−α−*1*))
using *x* α **by** (*intro mult-left-mono c2*) *auto*
also have ... = *c2* / (α + *1*)
using α *x* **by** (*simp add*: *field-simps powr-diff powr-minus powr-add*)
finally show |*x powr* (α + *1*) / (α + *1*) * *R2 x*| ≤ *c2* / (α + *1*) **.**
next
have |*sum-upto* (λ*d*. *R1* (*x* / *real d*)) *x*| ≤ *sum-upto* (λ*d*. |*R1* (*x* / *real d*)|) *x*
unfolding *sum-upto-def* **by** (*rule sum-abs*)
also have ... ≤ *sum-upto* (λ*d*. *c1* * (*x* / *real d*) *powr* α) *x*
unfolding *sum-upto-def* **by** (*intro sum-mono c1*) *auto*
also have ... = *c1* * *x powr* α * *sum-upto* (λ*d*. *1* / *real d powr* α) *x*
by (*simp add*: *sum-upto-def sum-distrib-left powr-divide*)
also have *sum-upto* (λ*d*. *1* / *real d powr* α) *x* = *x powr* (*1*−α) / (*1*−α) + *R3 x*
using *x* **by** (*simp add*: *R3-def powr-minus field-simps*)
also have *c1* * *x powr* α * (*x powr* (*1* − α) / (*1* − α) + *R3 x*) =
c1 / (*1* − α) * *x* + *c1* * *x powr* α * *R3 x*

using x **by** (*simp add*: *powr-diff divide-simps*
del: *div-mult-self3 div-mult-self4 div-mult-self2 div-mult-self1*)
(*simp add*: *field-simps*)
also have $c1 * x\ powr\ \alpha * R3\ x \leq c1 * x\ powr\ \alpha * c3$
using x *c1*(*1*) *c3*(*2*)[*of x*] **by** (*intro mult-left-mono*) *auto*
finally show $|sum\text{-}upto\ (\lambda d.\ R1\ (x\ /\ d))\ x| \leq c1 * (1\ /\ (1 - \alpha) * x + c3 * x\ powr\ \alpha)$
by (*simp add*: *field-simps*)
qed (*use* α x **in** *simp-all*)
finally show $|sum\text{-}upto\ (divisor\text{-}sigma\ \alpha)\ x - \zeta\ /\ (\alpha + 1) * x\ powr\ (\alpha + 1)| \leq ub\ x$ **.**
qed

have *eventually* $(\lambda x.\ |sum\text{-}upto\ (divisor\text{-}sigma\ \alpha)\ x - \zeta\ /\ (\alpha+1) * x\ powr\ (\alpha+1)| \leq ub\ x)$ *at-top*
using *eventually-ge-at-top*[*of 1*] **by** *eventually-elim* (*use le* **in** *auto*)
hence *eventually* $(\lambda x.\ |sum\text{-}upto\ (divisor\text{-}sigma\ \alpha)\ x - \zeta/(\alpha+1) * x\ powr\ (\alpha+1)| \leq |ub\ x|)$ *at-top*
by *eventually-elim linarith*
hence $(\lambda x.\ sum\text{-}upto\ (divisor\text{-}sigma\ \alpha)\ x - \zeta/(\alpha+1) * x\ powr\ (\alpha+1)) \in O(ub)$
by (*intro landau-o.bigI*[*of 1*]) *auto*
also have $ub \in O(\lambda x.\ x\ powr\ max\ 1\ \alpha)$
using α **unfolding** *ub-def* **by** (*cases* $\alpha \geq 1$; *real-asymp*)
finally show *?thesis*
by (*subst set-minus-plus* [*symmetric*])
(*simp-all add*: *fun-diff-def algebra-simps* ζ*-def zeta-even-numeral*)
qed

10.3 Case 3: $\alpha < 0$

Last, we consider the case of a negative exponent. We have for $\alpha > 0$:

$$\sum_{n \leq x} \sigma_{-\alpha}(n) = \zeta(\alpha + 1)x + O(R(x))$$

where $R(x) = \ln x$ if $\alpha = 1$ and $R(x) = x^{\max(0,1-\alpha)}$ otherwise.

theorem *summatory-divisor-sigma-asymptotics-neg*:
fixes α :: *real*
assumes α: $\alpha > 0$
defines $\delta \equiv max\ 0\ (1 - \alpha)$
defines $\zeta \equiv Re\ (zeta\ (\alpha + 1))$
shows $sum\text{-}upto\ (divisor\text{-}sigma\ (-\alpha)) =o\ (if\ \alpha = 1\ then\ (\lambda x.\ pi^2/6 * x) +o O(ln)$
$else\ (\lambda x.\ \zeta * x) +o O(\lambda x.\ x\ powr\ \delta))$
proof −
define *Ra* **where** $Ra = (\lambda x.\ -sum\text{-}upto\ (\lambda d.\ d\ powr\ (-\alpha) * frac\ (x\ /\ d))\ x)$
define *R1* **where** $R1 = (\lambda x.\ sum\text{-}upto\ (\lambda d.\ real\ d\ powr\ (-\alpha))\ x - (x\ powr\ (1 - \alpha)\ /\ (1 - \alpha) + \zeta))$

define *R2* **where** *R2* = (λ*x*. *sum-upto* (λ*d*. *d powr* (−α − *1*)) *x* − (ζ − *x powr* −α / α))
define *R3* **where** *R3* = (λ*x*. *sum-upto* (λ*d*. *d powr* −α) *x* − *x powr* (*1* − α) / (*1* − α))
obtain *c2* **where** *c2*: *c2* > *0* ⋀*x*. *x* ≥ *1* ⟹ |*R2 x*| ≤ *c2* ∗ *x powr* (−α−*1*)
using *zeta-partial-sum-le-pos*[*of* α + *1*] α **by** (*auto simp*: ζ*-def R2-def*)
define *ub* :: *real* ⇒ *real* **where**
ub = (λ*x*. *x powr* (*1* − α) / α + *c2* ∗ *x powr* − α + |*Ra x*|)

have *le*: |*sum-upto* (*divisor-sigma* (−α)) *x* − ζ ∗ *x*| ≤ *ub x*
if *x*: *x* ≥ *1* **for** *x*
proof −
have *div-le*: *real* (*a div b*) ≤ *x* **if** *a* ≤ *x* **for** *a b* :: *nat*
by (*rule order.trans*[*OF* - *that*(*1*)]) *auto*

have *sum-upto* (*divisor-sigma* (−α)) *x* =
sum-upto (*dirichlet-prod* (λ*n*. *real n powr* (−α)) (λ-. *1*)) *x*
by (*simp add*: *divisor-sigma-conv-dirichlet-prod* [*abs-def*]
sum-upto-def divisor-sigma-1-left [*symmetric*])
also have ... = *sum-upto* (λ*n*. ∑ *d* | *d dvd n*. *real d powr* (−α)) *x*
by (*simp add*: *dirichlet-prod-def*)
also have ... = (∑ (*n*, *d*) ∈ (*SIGMA n*:{*n*. *n* > *0* ∧ *real n* ≤ *x*}. {*d*. *d dvd n*}). *real d powr* (−α))
unfolding *sum-upto-def* **by** (*subst sum.Sigma*) *auto*
also have ... = (∑ (*d*, *q*) ∈ (*SIGMA d*:{*d*. *d* > *0* ∧ *real d* ≤ *x*}. {*q*. *q* > *0* ∧ *real q* ≤ *x* / *d*}).
real d powr (−α))
by (*rule sum.reindex-bij-witness*[*of* - λ(*d*, *q*). (*d* ∗ *q*, *d*) λ(*n*, *d*). (*d*, *n div d*)])
(*use div-le* **in** ‹*auto simp*: *field-simps dest*: *dvd-imp-le*›)
also have ... = *sum-upto* (λ*d*. *sum-upto* (λ*q*. *d powr* (−α)) (*x* / *d*)) *x*
by (*simp add*: *sum-upto-def sum.Sigma* [*symmetric*])
also have ... = *sum-upto* (λ*d*. *d powr* (−α) ∗ ⌊*x* / *d*⌋) *x*
using *x* **by** (*simp add*: *sum-upto-altdef mult-ac*)
also have ... = *x* ∗ *sum-upto* (λ*d*. *d powr* (−α) / *d*) *x* + *Ra x*
by (*simp add*: *frac-def sum-distrib-left sum-distrib-right*
sum-subtractf sum-upto-def algebra-simps Ra-def)
also have *sum-upto* (λ*d*. *d powr* (−α) / *d*) *x* = *sum-upto* (λ*d*. *d powr* (−α − *1*)) *x*
by (*simp add*: *powr-diff powr-minus powr-add field-simps*)
also have ... = ζ − *x powr* − α / α + *R2 x*
by (*simp add*: *R2-def*)
finally have *sum-upto* (*divisor-sigma* (−α)) *x* − ζ ∗ *x* = −(*x powr* (*1* − α) / α) + *x* ∗ *R2 x* + *Ra x*
using *x* α **by** (*simp add*: *powr-diff powr-minus field-simps*)

also have |...| ≤ *x powr* (*1* − α) / α + *c2* ∗ *x powr* −α + |*Ra x*|
proof (*rule order.trans*[*OF abs-triangle-ineq*] *order.trans*[*OF abs-triangle-ineq4*] *add-mono*)+
from *x* **have** |*x* ∗ *R2 x*| ≤ *x* ∗ |*R2 x*|

```
    by (simp add: abs-mult)
  also from x have ... ≤ x * (c2 * x powr (−α − 1))
    by (intro mult-left-mono c2) auto
  also have ... = c2 * x powr −α
    using x by (simp add: field-simps powr-minus powr-diff)
  finally show |x * R2 x| ≤ ... .
 qed (use x α in auto)
 finally show |sum-upto (divisor-sigma (− α)) x − ζ * x| ≤ ub x
  by (simp add: ub-def)
qed

have eventually (λx. |sum-upto (divisor-sigma (−α)) x − ζ * x| ≤ ub x) at-top
 using eventually-ge-at-top[of 1] by eventually-elim (use le in auto)
hence eventually (λx. |sum-upto (divisor-sigma (−α)) x − ζ * x| ≤ |ub x|) at-top
 by eventually-elim linarith
hence bigo: (λx. sum-upto (divisor-sigma (−α)) x − ζ * x) ∈ O(ub)
 by (intro landau-o.bigI[of 1]) auto

define ub' :: real ⇒ real where ub' = sum-upto (λn. real n powr − α)
have |Ra x| ≤ |ub' x| if x ≥ 1 for x
proof −
 have |Ra x| ≤ sum-upto (λn. |real n powr −α * frac (x / n)|) x
  unfolding Ra-def abs-minus sum-upto-def by (rule sum-abs)
 also have ... ≤ sum-upto (λn. real n powr −α * 1) x
  unfolding abs-mult sum-upto-def
  by (intro sum-mono mult-mono) (auto intro: less-imp-le[OF frac-lt-1])
 finally show ?thesis by (simp add: ub'-def)
qed
hence Ra ∈ O(ub')
 by (intro bigoI[of - 1] eventually-mono[OF eventually-ge-at-top[of 1]]) auto
also have ub' ∈ O(λx. if α = 1 then ln x else x powr δ)
proof (cases α = 1)
 case [simp]: True
 have sum-upto (λn. 1 / n) ∈ O(ln)
  by (intro asymp-equiv-imp-bigo harm-asymp-equiv)
 thus ?thesis by (simp add: ub'-def powr-minus field-simps)
next
 case False
 have sum-upto (λn. real n powr − α) ∈ O(λx. x powr δ)
   using assms False unfolding δ-def by (intro zeta-partial-sum-pos-bigtheta
bigthetaD1)
 thus ?thesis
  using zeta-partial-sum-neg-asymp-equiv[of α] α False by (simp add: ub'-def)
qed
finally have Ra-bigo: Ra ∈ ... .

show ?thesis
proof (cases α = 1)
 case [simp]: True
```

with *Ra-bigo* **have** *Ra*: $(\lambda x.\ |Ra\ x|) \in O(ln)$ **by** *simp*
note *bigo*
also have $ub \in O(\lambda x.\ ln\ x)$
unfolding *ub-def* **by** (*intro sum-in-bigo Ra*) *real-asymp*+
finally have *sum-upto* (*divisor-sigma* $(-\alpha)$) $=o$ $(\lambda x.\ (pi^2\ /\ 6) * x) +o O(ln)$
by (*subst set-minus-plus* [*symmetric*])
(*simp-all add*: *fun-diff-def algebra-simps* ζ*-def zeta-even-numeral*)
thus *?thesis* **by** (*simp only*: *True refl if-True*)
next
case *False*
with *Ra-bigo* **have** *Ra*: $(\lambda x.\ |Ra\ x|) \in O(\lambda x.\ x\ powr\ \delta)$ **by** *simp*
have $*$: $(\lambda x.\ x\ powr\ (1 - \alpha)\ /\ \alpha) \in O(\lambda x.\ x\ powr\ \delta)$
$(\lambda x.\ c2 * x\ powr - \alpha) \in O(\lambda x.\ x\ powr\ \delta)$
unfolding δ*-def* **using** α *False* **by** (*cases* $\alpha > 1$; *real-asymp*)+

note *bigo*
also have $ub \in O(\lambda x.\ x\ powr\ \delta)$
unfolding *ub-def* **using** α *False* **by** (*intro sum-in-bigo Ra* $*$)
finally have *sum-upto* (*divisor-sigma* $(-\alpha)$) $=o$ $(\lambda x.\ \zeta * x) +o O(\lambda x.\ x\ powr\ \delta)$
by (*subst set-minus-plus* [*symmetric*])
(*simp-all add*: *fun-diff-def algebra-simps* ζ*-def zeta-even-numeral*)
thus *?thesis* **by** (*simp only*: *False refl if-False*)
qed
qed

end

11 Selberg's asymptotic formula

theory *Selberg-Asymptotic-Formula*
imports
More-Dirichlet-Misc
Prime-Number-Theorem.Prime-Counting-Functions
Shapiro-Tauberian
Euler-MacLaurin.Euler-MacLaurin-Landau
Partial-Zeta-Bounds
begin

Following Apostol, we first show an inversion formula: Consider a function $f(x)$ for $x \in \mathbb{R}_{>0}$. Define $g(x) := \ln x \cdot \sum_{n \le x} f(x/n)$. Then:

$$f(x) \ln x + \sum_{n \le x} \Lambda(n) f(x/n) = \sum_{n \le x} \mu(n) g(x/n)$$

locale *selberg-inversion* =
fixes $F\ G$ :: *real* $\Rightarrow$ $'a$ :: {*real-algebra-1*, *comm-ring-1*}
defines $G \equiv (\lambda x.\ of\text{-}real\ (ln\ x) * sum\text{-}upto\ (\lambda n.\ F\ (x\ /\ n))\ x)$

begin

lemma *eq*:
assumes $x \geq 1$
shows $F\ x * \textit{of-real}\ (\textit{ln}\ x) + \textit{dirichlet-prod}'\ \textit{mangoldt}\ F\ x = \textit{dirichlet-prod}'\ \textit{moebius-mu}\ G\ x$
proof −
have $F\ x * \textit{of-real}\ (\textit{ln}\ x) =$
$\textit{dirichlet-prod}'\ (\lambda n.\ \textit{if}\ n = 1\ \textit{then}\ 1\ \textit{else}\ 0)\ (\lambda x.\ F\ x * \textit{of-real}\ (\textit{ln}\ x))\ x$
by (*subst dirichlet-prod'-one-left*) (*use* ‹$x \geq 1$› **in** *auto*)
also have $\ldots = \textit{dirichlet-prod}'\ (\lambda n.\ \sum d \mid d\ \textit{dvd}\ n.\ \textit{moebius-mu}\ d)\ (\lambda x.\ F\ x * \textit{of-real}\ (\textit{ln}\ x))\ x$
by (*intro dirichlet-prod'-cong refl*, *subst sum-moebius-mu-divisors'*) *auto*
finally have *eq1*: $F\ x * \textit{of-real}\ (\textit{ln}\ x) = \ldots$ **.**

have *eq2*: $\textit{dirichlet-prod}'\ \textit{mangoldt}\ F\ x =$
$\textit{dirichlet-prod}'\ (\textit{dirichlet-prod}\ \textit{moebius-mu}\ (\lambda n.\ \textit{of-real}\ (\textit{ln}\ (\textit{real}\ n))))\ F\ x$
proof (*intro dirichlet-prod'-cong refl*)
fix $n :: \textit{nat}$ **assume** n: $n > 0$
thus $\textit{mangoldt}\ n = \textit{dirichlet-prod}\ \textit{moebius-mu}\ (\lambda n.\ \textit{of-real}\ (\textit{ln}\ (\textit{real}\ n)) :: 'a)\ n$
by (*intro moebius-inversion mangoldt-sum* [*symmetric*]) *auto*
qed

have $F\ x * \textit{of-real}\ (\textit{ln}\ x) + \textit{dirichlet-prod}'\ \textit{mangoldt}\ F\ x =$
$\textit{sum-upto}\ (\lambda n.\ F\ (x\ /\ n) * (\sum d \mid d\ \textit{dvd}\ n.$
$\textit{moebius-mu}\ d * \textit{of-real}\ (\textit{ln}\ (x\ /\ n) + \textit{ln}\ (n\ \textit{div}\ d))))\ x$
unfolding *eq1 eq2* **unfolding** *dirichlet-prod'-def sum-upto-def*
by (*simp add*: *algebra-simps sum.distrib dirichlet-prod-def sum-distrib-left sum-distrib-right*)
also have $\ldots = \textit{sum-upto}\ (\lambda n.\ F\ (x\ /\ n) * (\sum d \mid d\ \textit{dvd}\ n.\ \textit{moebius-mu}\ d * \textit{of-real}\ (\textit{ln}\ (x\ /\ d))))\ x$
using ‹$x \geq 1$› **by** (*intro sum-upto-cong refl arg-cong2*[**where** $f = \lambda x\ y.\ x * y$] *sum.cong*)
(*auto elim*!: *dvdE simp*: *ln-div ln-mult*)
also have $\ldots = \textit{sum-upto}\ (\lambda n.\ \sum d \mid d\ \textit{dvd}\ n.\ \textit{moebius-mu}\ d * \textit{of-real}\ (\textit{ln}\ (x\ /\ d)) * F\ (x\ /\ n))\ x$
by (*simp add*: *sum-distrib-left sum-distrib-right mult-ac*)
also have $\ldots = (\sum (n,d) \in (\textit{SIGMA}\ n : \{n.\ n > 0 \wedge \textit{real}\ n \leq x\}.\ \{d.\ d\ \textit{dvd}\ n\}).$
$\textit{moebius-mu}\ d * \textit{of-real}\ (\textit{ln}\ (x\ /\ d)) * F\ (x\ /\ n))$
unfolding *sum-upto-def* **by** (*subst sum.Sigma*) (*auto simp*: *case-prod-unfold*)
also have $\ldots = (\sum (d,q) \in (\textit{SIGMA}\ d : \{d.\ d > 0 \wedge \textit{real}\ d \leq x\}.\ \{q.\ q > 0 \wedge \textit{real}\ q \leq x\ /\ d\}).$
$\textit{moebius-mu}\ d * \textit{of-real}\ (\textit{ln}\ (x\ /\ d)) * F\ (x\ /\ (q * d)))$
by (*rule sum.reindex-bij-witness*[*of* - $\lambda(d,q).\ (d * q,\ d)$ $\lambda(n,d).\ (d,\ n\ \textit{div}\ d)$])
(*auto simp*: *Real.real-of-nat-div field-simps dest*: *dvd-imp-le*)
also have $\ldots = \textit{sum-upto}\ (\lambda d.\ \textit{moebius-mu}\ d * \textit{of-real}\ (\textit{ln}\ (x\ /\ d)) * \textit{sum-upto}\ (\lambda q.\ F\ (x\ /\ (q * d)))\ (x\ /\ d))\ x$
by (*subst sum.Sigma* [*symmetric*]) (*auto simp*: *sum-upto-def sum-distrib-left*)
also have $\ldots = \textit{dirichlet-prod}'\ \textit{moebius-mu}\ G\ x$

```
    by (simp add: dirichlet-prod'-def G-def mult-ac)
  finally show ?thesis .
qed

end
```

We can now show Selberg's formula

$$\psi(x)\ln x + \sum_{n\leq x} \Lambda(n)\psi(x/n) = 2x\ln x + O(x)\ .$$

```
theorem selberg-asymptotic-formula:
  includes prime-counting-syntax
  shows   (λx. ψ x * ln x + dirichlet-prod' mangoldt ψ x) =o
          (λx. 2 * x * ln x) +o O(λx. x)
proof −
  define C :: real where [simp]: C = euler-mascheroni
  define F2 :: real ⇒ real where [simp]: F2 = (λx. x − C − 1)
  define G1 where G1 = (λx. ln x * sum-upto (λn. ψ (x / n)) x)
  define G2 where G2 = (λx. ln x * sum-upto (λn. F2 (x / n)) x)

  interpret F1: selberg-inversion ψ G1
    by unfold-locales (simp-all add: G1-def)
  interpret F2: selberg-inversion F2 G2
    by unfold-locales (simp-all add: G2-def)

  have G1-bigo: (λx. G1 x − (x * ln x ^ 2 − x * ln x)) ∈ O(λx. ln x ^ 2)
  proof −
    have (λx. ln x * (sum-upto (λn. ψ (x / n)) x − x * ln x + x)) ∈ O(λx. ln x *
ln x)
      by (intro landau-o.big.mult-left sum-upto-ψ-x-over-n-asymptotics)
    thus ?thesis by (simp add: power2-eq-square G1-def algebra-simps)
  qed

  have G2-bigo: (λx. G2 x − (x * ln x ^ 2 − x * ln x)) ∈ O(ln)
  proof −
  define R1 :: real ⇒ real where R1 = (λx. x * ln x * (harm (nat ⌊x⌋) − (ln x
+ C)))
  define R2 :: real ⇒ real where R2 = (λx. (C + 1) * ln x * frac x)
    have (λx. G2 x − (x * ln x ^ 2 − x * ln x)) ∈ Θ(λx. R1 x + R2 x)
    proof (intro bigthetaI-cong eventually-mono[OF eventually-ge-at-top[of 1]])
      fix x :: real assume x: x ≥ 1
      have G2 x = x * ln x * sum-upto (λn. 1 / n) x − (C + 1) * ⌊x⌋ * ln x
        using x by (simp add: G2-def sum-upto-altdef sum-subtractf
                              sum-distrib-left sum-distrib-right algebra-simps)
      also have sum-upto (λn. 1 / n) x = harm (nat ⌊x⌋)
        using x unfolding sum-upto-def harm-def
        by (intro sum.cong) (auto simp: field-simps le-nat-iff le-floor-iff)
      also have x * ln x * harm (nat ⌊x⌋) − (C + 1) * ⌊x⌋ * ln x =
                  x * ln x ^ 2 − x * ln x + R1 x + R2 x
```

by (*simp add*: *R1-def R2-def algebra-simps frac-def power2-eq-square*)
finally show *G2 x* − (*x* ∗ *ln x* ^ *2* − *x* ∗ *ln x*) = *R1 x* + *R2 x* **by** *simp*
qed
also have (λ*x*. *R1 x* + *R2 x*) ∈ *O*(*ln*)
proof (*intro sum-in-bigo*)
have (λ*x*::*real*. *ln x* − *ln* (*nat* ⌊*x*⌋)) ∈ *O*(λ*x*. *ln x* − *ln* (*x* − *1*))
proof (*intro bigoI*[*of - 1*] *eventually-mono*[*OF eventually-ge-at-top*[*of 2*]])
fix *x* :: *real* **assume** *x*: *x* ≥ *2*
thus *norm* (*ln x* − *ln* (*nat* ⌊*x*⌋)) ≤ *1* ∗ *norm* (*ln x* − *ln* (*x* − *1*)) **by** *auto*
qed
also have (λ*x*::*real*. *ln x* − *ln* (*x* − *1*)) ∈ *O*(λ*x*. *1* / *x*) **by** *real-asymp*
finally have *bigo-ln-floor*: (λ*x*::*real*. *ln x* − *ln* (*nat* ⌊*x*⌋)) ∈ *O*(λ*x*. *1* / *x*) **.**

have (λ*x*. *harm* (*nat* ⌊*x*⌋) − (*ln* (*nat* ⌊*x*⌋) + *C*)) ∈ *O*(λ*x*. *1* / *nat* ⌊*x*⌋)
unfolding *C-def* **using** *harm-expansion-bigo-simple2*
by (*rule landau-o.big.compose*)
(*auto intro*!: *filterlim-compose*[*OF filterlim-nat-sequentially filterlim-floor-sequentially*])
also have (λ*x*. *1* / *nat* ⌊*x*⌋) ∈ *O*(λ*x*. *1* / *x*) **by** *real-asymp*
finally have (λ*x*. *harm* (*nat* ⌊*x*⌋) − (*ln* (*nat* ⌊*x*⌋) + *C*) − (*ln x* − *ln* (*nat* ⌊*x*⌋)))
∈ *O*(λ*x*. *1* / *x*) **by** (*rule sum-in-bigo*[*OF -bigo-ln-floor*])
hence (λ*x*. *harm* (*nat* ⌊*x*⌋) − (*ln x* + *C*)) ∈ *O*(λ*x*. *1* / *x*) **by** (*simp add*: *algebra-simps*)
hence (λ*x*. *x* ∗ *ln x* ∗ (*harm* (*nat* ⌊*x*⌋) − (*ln x* + *C*))) ∈ *O*(λ*x*. *x* ∗ *ln x* ∗ (*1* / *x*))
by (*intro landau-o.big.mult-left*)
thus *R1* ∈ *O*(*ln*) **by** (*simp add*: *landau-divide-simps R1-def*)
next
have *R2* ∈ *O*(λ*x*. *1* ∗ *ln x* ∗ *1*)
unfolding *R2-def* **by** (*intro landau-o.big.mult landau-o.big-refl*) *real-asymp*+
thus *R2* ∈ *O*(*ln*) **by** (*simp add*: *R2-def*)
qed
finally show (λ*x*. *G2 x* − (*x* ∗ (*ln x*)^2 − *x* ∗ *ln x*)) ∈ *O*(*ln*) **.**
qed
hence *G2-bigo′*: (λ*x*. *G2 x* − (*x* ∗ (*ln x*)^2 − *x* ∗ *ln x*)) ∈ *O*(λ*x*. *ln x* ^ *2*)
by (*rule landau-o.big.trans*) *real-asymp*+

— Now things become a bit hairy. In order to show that the 'Big-O' bound is actually valid for all *x* ≥ *1*, we need to show that *G1 x* − *G2 x* is bounded on any compact interval starting at 1.
have ∃ *c*>*0*. ∀ *x*≥*1*. |*G1 x* − *G2 x*| ≤ *c* ∗ |*sqrt x*|
proof (*rule bigoE-bounded-real-fun*)
have (λ*x*. *G1 x* − *G2 x*) ∈ *O*(λ*x*. *ln x* ^ *2*)
using *sum-in-bigo*(*2*)[*OF G1-bigo G2-bigo′*] **by** *simp*
also have (λ*x*::*real*. *ln x* ^ *2*) ∈ *O*(*sqrt*) **by** *real-asymp*
finally show (λ*x*. *G1 x* − *G2 x*) ∈ *O*(*sqrt*) **.**
next
fix *x* :: *real* **assume** *x* ≥ *1*
thus |*sqrt x*| ≥ *1* **by** *simp*

next
fix b :: *real* **assume** b: $b \geq 1$
show *bounded* $((\lambda x.\ G1\ x - G2\ x)\ `\ \{1..b\})$
proof (*rule boundedI*, *safe*)
fix x **assume** x: $x \in \{1..b\}$
have $|G1\ x - G2\ x| = |ln\ x * sum\text{-}upto\ (\lambda n.\ \psi\ (x\ /\ n) - F2\ (x\ /\ n))\ x|$
by (*simp add*: *G1-def G2-def sum-upto-def sum-distrib-left ring-distribs sum-subtractf*)
also have $\ldots = ln\ x * |sum\text{-}upto\ (\lambda n.\ \psi\ (x\ /\ n) - F2\ (x\ /\ n))\ x|$
using x b **by** (*simp add*: *abs-mult*)
also have $|sum\text{-}upto\ (\lambda n.\ \psi\ (x\ /\ n) - F2\ (x\ /\ n))\ x| \leq$
$sum\text{-}upto\ (\lambda n.\ |\psi\ (x\ /\ n) - F2\ (x\ /\ n)|)\ x$
unfolding *sum-upto-def* **by** (*rule sum-abs*)
also have $\ldots \leq sum\text{-}upto\ (\lambda n.\ \psi\ x + (x + C + 1))\ x$
unfolding *sum-upto-def*
proof (*intro sum-mono*)
fix n **assume** n: $n \in \{n.\ n > 0 \wedge real\ n \leq x\}$
hence *le*: $x\ /\ n \leq x\ /\ 1$ **by** (*intro divide-left-mono*) *auto*
thus $|\psi\ (x\ /\ n) - F2\ (x\ /\ n)| \leq \psi\ x + (x + C + 1)$
unfolding *F2-def* **using** *euler-mascheroni-pos x le* ψ*-nonneg* ψ*-mono*[*of x / n x*]
by (*intro order.trans*[*OF abs-triangle-ineq*]
order.trans[*OF abs-triangle-ineq4*] *add-mono*) *auto*
qed
also have $\ldots = (\psi\ x + (x + C + 1)) * \lfloor x \rfloor$
using x **by** (*simp add*: *sum-upto-altdef*)
also have $ln\ x * ((\psi\ x + (x + C + 1)) * real\text{-}of\text{-}int\ \lfloor x \rfloor) \leq$
$ln\ b * ((\psi\ b + (b + C + 1)) * real\text{-}of\text{-}int\ \lfloor b \rfloor)$
using *euler-mascheroni-pos x*
by (*intro mult-mono add-mono order.refl* ψ*-mono add-nonneg-nonneg mult-nonneg-nonneg*
ψ*-nonneg*) (*auto intro*: *floor-mono*)
finally show $norm\ (G1\ x - G2\ x) \leq ln\ b * ((\psi\ b + (b + C + 1)) * real\text{-}of\text{-}int\ \lfloor b \rfloor)$
using x **by** (*simp add*: *mult-left-mono*)
qed
qed *auto*
then obtain A **where** A: $A > 0\ \bigwedge x.\ x \geq 1 \Longrightarrow |G1\ x - G2\ x| \leq A * sqrt\ x$ **by** *auto*

— The rest of the proof now consists merely of combining some asymptotic estimates.
have $(\lambda x.\ (\psi\ x - F2\ x) * ln\ x + sum\text{-}upto\ (\lambda n.\ mangoldt\ n * (\psi\ (x\ /\ n) - F2\ (x\ /\ n)))\ x)$
$\in \Theta(\lambda x.\ sum\text{-}upto\ (\lambda n.\ moebius\text{-}mu\ n * (G1\ (x\ /\ n) - G2\ (x\ /\ n)))\ x)$
proof (*intro bigthetaI-cong eventually-mono*[*OF eventually-ge-at-top*[*of 1*]])
fix x :: *real* **assume** x: $x \geq 1$
have $(\psi\ x - F2\ x) * ln\ x + sum\text{-}upto\ (\lambda n.\ mangoldt\ n * (\psi\ (x\ /\ n) - F2\ (x\ /\ n)))\ x =$

$(\psi\ x * \textit{of-real}\ (\ln x) + \textit{dirichlet-prod}'\ \textit{mangoldt}\ \psi\ x) -$
$(F2\ x * \textit{of-real}\ (\ln x) + \textit{dirichlet-prod}'\ \textit{mangoldt}\ F2\ x)$
by (*simp add*: *algebra-simps dirichlet-prod'-def sum-upto-def sum-subtractf sum.distrib*)
also have $\ldots = \textit{sum-upto}\ (\lambda n.\ \textit{moebius-mu}\ n * (G1\ (x\ /\ n) - G2\ (x\ /\ n)))\ x$
unfolding *F1.eq*[*OF x*] *F2.eq*[*OF x*]
by (*simp add*: *dirichlet-prod'-def sum-upto-def sum-subtractf sum.distrib algebra-simps*)
finally show $(\psi\ x - F2\ x) * \ln x + \textit{sum-upto}\ (\lambda n.\ \textit{mangoldt}\ n * (\psi\ (x/n) - F2\ (x/n)))\ x = \ldots$ **.**
qed
also have $(\lambda x.\ \textit{sum-upto}\ (\lambda n.\ \textit{moebius-mu}\ n * (G1\ (x\ /\ n) - G2\ (x\ /\ n)))\ x) \in O(\lambda x.\ A * \textit{sqrt}\ x * \textit{sum-upto}\ (\lambda x.\ x\ \textit{powr}\ (-1/2))\ x)$
proof (*intro bigoI eventually-mono*[*OF eventually-ge-at-top*[*of 1*]])
fix $x :: \textit{real}$ **assume** x: $x \geq 1$
have $|\textit{sum-upto}\ (\lambda n.\ \textit{moebius-mu}\ n * (G1\ (x\ /\ n) - G2\ (x\ /\ n)))\ x| \leq \textit{sum-upto}\ (\lambda n.\ |\textit{moebius-mu}\ n * (G1\ (x\ /\ n) - G2\ (x\ /\ n))|)\ x$
unfolding *sum-upto-def* **by** (*rule sum-abs*)
also have $\ldots \leq \textit{sum-upto}\ (\lambda n.\ 1 * (A * \textit{sqrt}\ (x\ /\ n)))\ x$
unfolding *sum-upto-def abs-mult* **by** (*intro A sum-mono mult-mono*) (*auto simp*: *moebius-mu-def*)
also have $\ldots = A * \textit{sqrt}\ x * \textit{sum-upto}\ (\lambda x.\ x\ \textit{powr}\ (-1/2))\ x$
using x **by** (*simp add*: *sum-upto-def powr-minus powr-half-sqrt sum-distrib-left sum-distrib-right real-sqrt-divide field-simps*)
also have $\ldots \leq |A * \textit{sqrt}\ x * \textit{sum-upto}\ (\lambda x.\ x\ \textit{powr}\ (-1/2))\ x|$ **by** *simp*
finally show $\textit{norm}\ (\textit{sum-upto}\ (\lambda n.\ \textit{moebius-mu}\ n * (G1\ (x\ /\ n) - G2\ (x\ /\ n)))\ x) \leq 1 * \textit{norm}\ (A * \textit{sqrt}\ x * \textit{sum-upto}\ (\lambda x.\ x\ \textit{powr}\ (-1/2))\ x)$ **by** *simp*
qed
also have $(\lambda x.\ A * \textit{sqrt}\ x * \textit{sum-upto}\ (\lambda x.\ x\ \textit{powr}\ (-1/2))\ x) \in O(\lambda x.\ 1 * \textit{sqrt}\ x * x\ \textit{powr}\ (1/2))$
using *zeta-partial-sum-le-pos-bigo*[*of 1 / 2*]
by (*intro landau-o.big.mult*) (*auto simp*: *max-def*)
also have $(\lambda x{::}\textit{real}.\ 1 * \textit{sqrt}\ x * x\ \textit{powr}\ (1/2)) \in O(\lambda x.\ x)$
by *real-asymp*
finally have *bigo*: $(\lambda x.\ (\psi\ x - F2\ x) * \ln x + \textit{sum-upto}\ (\lambda n.\ \textit{mangoldt}\ n * (\psi\ (x/n) - F2\ (x/n)))\ x) \in O(\lambda x.\ x)$ (**is** *?h* $\in$ -) **.**

let *?R* $= \lambda x.\ \textit{sum-upto}\ (\lambda n.\ \textit{mangoldt}\ n\ /\ n)\ x$
let *?lhs* $= \lambda x.\ \psi\ x * \ln x + \textit{dirichlet-prod}'\ \textit{mangoldt}\ \psi\ x$

note *bigo*
also have *?h* $= (\lambda x.\ ?lhs\ x - (x * \ln x - (C + 1) * (\ln x + \psi\ x)) - x * ?R\ x)$
by (*rule ext*) (*simp add*: *algebra-simps dirichlet-prod'-def sum-distrib-right* ψ*-def sum-upto-def sum-subtractf sum.distrib sum-distrib-left*)
finally have $(\lambda x.\ ?lhs\ x - (x * \ln x - (C + 1) * (\ln x + \psi\ x)) - x * ?R\ x + x * (?R\ x - \ln x))$

$\in O(\lambda x.\ x)$ (**is** $?h' \in$ -)
proof (*rule sum-in-bigo*)
have $(\lambda x.\ x * (\textit{sum-upto}\ (\lambda n.\ \textit{mangoldt}\ n\ /\ \textit{real}\ n)\ x - \textit{ln}\ x)) \in O(\lambda x.\ x * 1)$
by (*intro landau-o.big.mult-left* ψ*.asymptotics*)
thus $(\lambda x.\ x * (\textit{sum-upto}\ (\lambda n.\ \textit{mangoldt}\ n\ /\ \textit{real}\ n)\ x - \textit{ln}\ x)) \in O(\lambda x.\ x)$ **by** *simp*
qed
also have $?h' = (\lambda x.\ ?\textit{lhs}\ x - (2 * x * \textit{ln}\ x - (C + 1) * (\textit{ln}\ x + \psi\ x)))$
by (*simp add*: *fun-eq-iff algebra-simps*)
finally have $(\lambda x.\ ?\textit{lhs}\ x - (2{*}x{*}\textit{ln}\ x - (C{+}1) * (\textit{ln}\ x + \psi\ x)) - (C{+}1) * (\textit{ln}\ x + \psi\ x)) \in O(\lambda x.\ x)$
proof (*rule sum-in-bigo*)
have $(\lambda x.\ \textit{ln}\ x + \psi\ x) \in O(\lambda x.\ x)$
by (*intro sum-in-bigo bigthetaD1*[*OF* ψ*.bigtheta*]) *real-asymp+*
thus $(\lambda x.\ (C + 1) * (\textit{ln}\ x + \psi\ x)) \in O(\lambda x.\ x)$ **by** *simp*
qed
also have $(\lambda x.\ ?\textit{lhs}\ x - (2{*}x{*}\textit{ln}\ x - (C{+}1) * (\textit{ln}\ x + \psi\ x)) - (C{+}1) * (\textit{ln}\ x + \psi\ x)) =$
$(\lambda x.\ ?\textit{lhs}\ x - 2 * x * \textit{ln}\ x)$ **by** (*simp add*: *algebra-simps*)
finally show *?thesis*
by (*subst set-minus-plus* [*symmetric*]) (*simp-all add*: *fun-diff-def algebra-simps*)
qed

end

12 Consequences of the Prime Number Theorem

theory *PNT-Consequences*
imports
Elementary-Prime-Bounds
Prime-Number-Theorem.Mertens-Theorems
Prime-Number-Theorem.Prime-Counting-Functions
Moebius-Mu-Sum
Lcm-Nat-Upto
Primorial
Primes-Omega
begin

In this section, we will define a locale that assumes the Prime Number Theorem in order to explore some of its elementary consequences.

locale *prime-number-theorem* =
assumes *prime-number-theorem* [*asymp-equiv-intros*]: $\pi \sim$[*at-top*] $(\lambda x.\ x\ /\ \textit{ln}\ x)$
begin

corollary ϑ*-asymptotics* [*asymp-equiv-intros*]: $\vartheta \sim$[*at-top*] $(\lambda x.\ x)$
using *prime-number-theorem* **by** (*rule PNT1-imp-PNT4*)

corollary ψ*-asymptotics* [*asymp-equiv-intros*]: $\psi \sim$[*at-top*] $(\lambda x.\ x)$

using *ϑ-asymptotics PNT4-imp-PNT5* **by** *simp*

corollary *ln-π-asymptotics* [*asymp-equiv-intros*]: $(\lambda x.\ ln\ (\pi\ x)) \sim[at\text{-}top]\ ln$
using *prime-number-theorem PNT1-imp-PNT1'* **by** *simp*

corollary *π-ln-π-asymptotics*: $(\lambda x.\ \pi\ x * ln\ (\pi\ x)) \sim[at\text{-}top]\ (\lambda x.\ x)$
using *prime-number-theorem PNT1-imp-PNT2* **by** *simp*

corollary *nth-prime-asymptotics* [*asymp-equiv-intros*]:
$(\lambda n.\ real\ (nth\text{-}prime\ n)) \sim[at\text{-}top]\ (\lambda n.\ real\ n * ln\ (real\ n))$
using *π-ln-π-asymptotics PNT2-imp-PNT3* **by** *simp*

corollary *moebius-mu-smallo*: *sum-upto moebius-mu* $\in o(\lambda x.\ x)$
using *PNT-implies-sum-moebius-mu-sublinear ψ-asymptotics* **by** *simp*

lemma *ln-ϑ-asymptotics*:
includes *prime-counting-syntax*
shows $(\lambda x.\ ln\ (\vartheta\ x) - ln\ x) \in o(\lambda\text{-}.\ 1)$
proof −
have [*simp*]: $\vartheta\ 2 = ln\ 2$
by (*simp add*: *eval-ϑ*)
have *ϑ-pos*: $\vartheta\ x > 0$ **if** $x \geq 2$ **for** x
proof −
have $0 < ln\ (2 :: real)$ **by** *simp*
also have $\ldots \leq \vartheta\ x$
using *ϑ-mono*[*OF that*] **by** *simp*
finally show *?thesis* **.**
qed

have *nz*: *eventually* $(\lambda x.\ \vartheta\ x \neq 0 \vee x \neq 0)$ *at-top*
using *eventually-gt-at-top*[*of 0*] **by** *eventually-elim auto*
have *filterlim* $(\lambda x.\ \vartheta\ x\ /\ x)$ (*nhds 1*) *at-top*
using *asymp-equivD-strong*[*OF ϑ-asymptotics nz*] **.**
hence *filterlim* $(\lambda x.\ ln\ (\vartheta\ x\ /\ x))$ (*nhds* (*ln 1*)) *at-top*
by (*rule tendsto-ln*) *auto*
also have *?this* $\longleftrightarrow$ *filterlim* $(\lambda x.\ ln\ (\vartheta\ x) - ln\ x)$ (*nhds 0*) *at-top*
by (*intro filterlim-cong eventually-mono*[*OF eventually-ge-at-top*[*of 2*]])
(*auto simp*: *ln-divide-pos ϑ-pos*)
finally show $(\lambda x.\ ln\ (\vartheta\ x) - ln\ x) \in o(\lambda x.\ 1)$
by (*intro smalloI-tendsto*) *auto*
qed

lemma *ln-ϑ-asymp-equiv* [*asymp-equiv-intros*]:
includes *prime-counting-syntax*
shows $(\lambda x.\ ln\ (\vartheta\ x)) \sim[at\text{-}top]\ ln$
proof (*rule smallo-imp-asymp-equiv*)
have $(\lambda x.\ ln\ (\vartheta\ x) - ln\ x) \in o(\lambda\text{-}.\ 1)$ **by** (*rule ln-ϑ-asymptotics*)
also have $(\lambda\text{-}.\ 1) \in O(\lambda x{::}real.\ ln\ x)$ **by** *real-asymp*
finally show $(\lambda x.\ ln\ (\vartheta\ x) - ln\ x) \in o(ln)$ **.**

qed

lemma *ln-nth-prime-asymptotics*:
(λ*n*. *ln* (*nth-prime n*) − (*ln n* + *ln* (*ln n*))) ∈ *o*(λ-. *1*)
proof −
have *filterlim* (λ*n*. *ln* (*nth-prime n* / (*n* ∗ *ln n*))) (*nhds* (*ln 1*)) *at-top*
by (*intro tendsto-ln asymp-equivD-strong*[*OF nth-prime-asymptotics*])
(*auto intro*!: *eventually-mono*[*OF eventually-gt-at-top*[*of 1*]])
also have *?this* ⟷ *filterlim* (λ*n*. *ln* (*nth-prime n*) − (*ln n* + *ln* (*ln n*))) (*nhds 0*) *at-top*
using *prime-gt-0-nat*[*OF prime-nth-prime*]
by (*intro filterlim-cong refl eventually-mono*[*OF eventually-gt-at-top*[*of 1*]])
(*auto simp*: *field-simps ln-mult ln-div*)
finally show *?thesis* **by** (*intro smalloI-tendsto*) *auto*
qed

lemma *ln-nth-prime-asymp-equiv* [*asymp-equiv-intros*]:
(λ*n*. *ln* (*nth-prime n*)) ∼[*at-top*] *ln*
proof −
have (λ*n*. *ln* (*nth-prime n*) − (*ln n* + *ln* (*ln n*))) ∈ *o*(*ln*)
using *ln-nth-prime-asymptotics* **by** (*rule landau-o.small.trans*) *real-asymp*
hence (λ*n*. *ln* (*nth-prime n*) − (*ln n* + *ln* (*ln n*)) + *ln* (*ln n*)) ∈ *o*(*ln*)
by (*rule sum-in-smallo*) *real-asymp*
thus *?thesis* **by** (*intro smallo-imp-asymp-equiv*) *auto*
qed

The following versions use a little less notation.

corollary *prime-number-theorem′*: ((λ*x*. π *x* / (*x* / *ln x*)) ⟶ *1*) *at-top*
using *prime-number-theorem*
by (*rule asymp-equivD-strong*[*OF - eventually-mono*[*OF eventually-gt-at-top*[*of 1*]]]) *auto*

corollary *prime-number-theorem″*:
(λ*x*. *card* {*p*. *prime p* ∧ *real p* ≤ *x*}) ∼[*at-top*] (λ*x*. *x* / *ln x*)
proof −
have π = (λ*x*. *card* {*p*. *prime p* ∧ *real p* ≤ *x*})
by (*intro ext*) (*simp add*: π*-def prime-sum-upto-def*)
with *prime-number-theorem* **show** *?thesis* **by** *simp*
qed

corollary *prime-number-theorem‴*:
(λ*n*. *card* {*p*. *prime p* ∧ *p* ≤ *n*}) ∼[*at-top*] (λ*n*. *real n* / *ln* (*real n*))
proof −
have (λ*n*. *card* {*p*. *prime p* ∧ *real p* ≤ *real n*}) ∼[*at-top*] (λ*n*. *real n* / *ln* (*real n*))
using *prime-number-theorem″*
by (*rule asymp-equiv-compose′*) (*simp add*: *filterlim-real-sequentially*)
thus *?thesis* **by** *simp*
qed

end

12.1 Existence of primes in intervals

For fixed ε, The interval $(x; \varepsilon x]$ contains a prime number for any sufficiently large x. This proof was taken from A. J. Hildebrand's lecture notes [2].

lemma (**in** *prime-number-theorem*) *prime-in-interval-exists*:
fixes $c :: real$
assumes $c > 1$
shows *eventually* $(\lambda x.\ \exists p.\ prime\ p \wedge real\ p \in \{x<..c*x\})$ *at-top*
proof −
from ‹$c > 1$› **have** $(\lambda x.\ \pi\ (c * x)\ /\ \pi\ x) \sim[at\text{-}top]\ (\lambda x.\ ((c * x)\ /\ ln\ (c * x))\ /\ (x\ /\ ln\ x))$
by (*intro asymp-equiv-intros asymp-equiv-compose'*[*OF prime-number-theorem*]) *real-asymp*+
also have ... $\sim[at\text{-}top]\ (\lambda x.\ c)$
using ‹$c > 1$› **by** *real-asymp*
finally have $(\lambda x.\ \pi\ (c * x)\ /\ \pi\ x) \sim[at\text{-}top]\ (\lambda a.\ c)$ **by** *simp*
hence $((\lambda x.\ \pi\ (c * x)\ /\ \pi\ x) \longrightarrow c)$ *at-top*
by (*rule asymp-equivD-const*)
from *this* **and** ‹$c > 1$› **have** *eventually* $(\lambda x.\ \pi\ (c * x)\ /\ \pi\ x > 1)$ *at-top*
by (*rule order-tendstoD*)
moreover have *eventually* $(\lambda x.\ \pi\ x > 0)$ *at-top*
using π*-at-top* **by** (*auto simp*: *filterlim-at-top-dense*)
ultimately show *?thesis* **using** *eventually-gt-at-top*[*of 1*]
proof *eventually-elim*
case (*elim x*)
define P **where** $P = \{p.\ prime\ p \wedge real\ p \in \{x<..c*x\}\}$
from *elim* **and** ‹$c > 1$› **have** $1 * x < c * x$ **by** (*intro mult-strict-right-mono*) *auto*
hence $x < c * x$ **by** *simp*
have $P = \{p.\ prime\ p \wedge real\ p \leq c * x\} - \{p.\ prime\ p \wedge real\ p \leq x\}$
by (*auto simp*: *P-def*)
also have *card* ... = *card* $\{p.\ prime\ p \wedge real\ p \leq c * x\}$ − *card* $\{p.\ prime\ p \wedge real\ p \leq x\}$
using ‹$x < c * x$› **by** (*subst card-Diff-subset*) (*auto intro*: *finite-primes-le*)
also have *real* ... = $\pi\ (c * x) - \pi\ x$
using π*-mono*[*of x c * x*] ‹$x < c * x$›
by (*subst of-nat-diff*) (*auto simp*: *primes-pi-def prime-sum-upto-def*)
finally have *real* (*card P*) = $\pi\ (c * x) - \pi\ x$ **by** *simp*
moreover have $\pi\ (c * x) - \pi\ x > 0$
using *elim* **by** (*auto simp*: *field-simps*)
ultimately have *real* (*card P*) > 0 **by** *linarith*
hence *card P* > 0 **by** *simp*
hence $P \neq \{\}$ **by** (*intro notI*) *simp*
thus *?case* **by** (*auto simp*: *P-def*)
qed
qed

The set of rationals whose numerator and denominator are primes is dense in $\mathbb{R}_{>0}$.

lemma (**in** *prime-number-theorem*) *prime-fractions-dense*:
fixes α ε :: *real*
assumes $\alpha > 0$ **and** $\varepsilon > 0$
obtains *p q* :: *nat* **where** *prime p* **and** *prime q* **and** *dist* (*real p* / *real q*) $\alpha < \varepsilon$
proof −
define ε' **where** $\varepsilon' = \varepsilon$ / 2
from *assms* **have** $\varepsilon' > 0$ **by** (*simp add*: ε'-*def*)
have *eventually* ($\lambda x.\ \exists p.\ prime\ p \wedge real\ p \in \{x<..(1 + \varepsilon' / \alpha) * x\}$) *at-top*
using *assms* ‹$\varepsilon' > 0$› **by** (*intro prime-in-interval-exists*) (*auto simp*: *field-simps*)
then obtain *x0* **where** *x0*: $\bigwedge x.\ x \geq x0 \Longrightarrow \exists p.\ prime\ p \wedge real\ p \in \{x<..(1 + \varepsilon' / \alpha) * x\}$
by (*auto simp*: *eventually-at-top-linorder*)

have $\exists q.\ prime\ q \wedge q > nat\ \lfloor x0 / \alpha \rfloor$ **by** (*rule bigger-prime*)
then obtain *q* **where** *prime q q* > *nat* $\lfloor x0 / \alpha \rfloor$ **by** *blast*
hence *real q* ≥ *x0* / α **by** *linarith*
with ‹$\alpha > 0$› **have** $\alpha * real\ q \geq x0$ **by** (*simp add*: *field-simps*)
hence $\exists p.\ prime\ p \wedge real\ p \in \{\alpha * real\ q<..(1 + \varepsilon' / \alpha) * (\alpha * real\ q)\}$
by (*intro x0*)
then obtain *p* **where** *p*: *prime p real p* > α * *real q real p* ≤ (1 + ε' / α) * (α * *real q*)
using *assms* **by** *auto*

from *p* ‹*prime q*› **have** *real p* / *real q* ≤ (1 + ε' / α) * α
using *assms* **by** (*auto simp*: *field-simps dest*: *prime-gt-0-nat*)
also have ... = $\alpha + \varepsilon'$
using *assms* **by** (*simp add*: *field-simps*)
finally have *real p* / *real q* ≤ $\alpha + \varepsilon'$ **.**
moreover from *p* ‹*prime q*› **have** *real p* / *real q* > α *real p* / *real q* ≤ (1 + ε' / α) * α
using *assms* **by** (*auto simp*: *field-simps dest*: *prime-gt-0-nat*)
ultimately have *dist* (*real p* / *real q*) $\alpha \leq \varepsilon'$
by (*simp add*: *dist-norm*)
also have ... < ε
using ‹$\varepsilon > 0$› **by** (*simp add*: ε'-*def*)
finally show *?thesis* **using** ‹*prime p*› ‹*prime q*› *that*[*of p q*] **by** *blast*
qed

12.2 The logarithm of the primorial

The PNT directly implies the asymptotics of the logarithm of the primorial function:

context *prime-number-theorem*
begin

lemma *ln-primorial-asymp-equiv* [*asymp-equiv-intros*]:

$(\lambda x.\ ln\ (primorial\ x)) \sim[at\text{-}top]\ (\lambda x.\ x)$
by (*auto simp*: *ln-primorial ϑ-asymptotics*)

lemma *ln-ln-primorial-asymp-equiv* [*asymp-equiv-intros*]:
$(\lambda x.\ ln\ (ln\ (primorial\ x))) \sim[at\text{-}top]\ (\lambda x.\ ln\ x)$
by (*auto simp*: *ln-primorial ln-ϑ-asymp-equiv*)

lemma *ln-primorial′-asymp-equiv* [*asymp-equiv-intros*]:
$(\lambda k.\ ln\ (primorial'\ k)) \sim[at\text{-}top]\ (\lambda k.\ k * ln\ k)$
and *ln-ln-primorial′-asymp-equiv* [*asymp-equiv-intros*]:
$(\lambda k.\ ln\ (ln\ (primorial'\ k))) \sim[at\text{-}top]\ (\lambda k.\ ln\ k)$
and *ln-over-ln-ln-primorial′-asymp-equiv*:
$(\lambda k.\ ln\ (primorial'\ k)\ /\ ln\ (ln\ (primorial'\ k))) \sim[at\text{-}top]\ (\lambda k.\ k)$
proof −
have *lim1*: *filterlim* $(\lambda k.\ real\ (nth\text{-}prime\ (k - 1)))$ *at-top at-top*
by (*rule filterlim-compose*[*OF filterlim-real-sequentially*]
filterlim-compose[*OF nth-prime-at-top*])+ *real-asymp*
have *lim2*: *filterlim* $(\lambda k{::}nat.\ k - 1)$ *at-top at-top*
by *real-asymp*

have $(\lambda k.\ ln\ (primorial'\ k)) \sim[at\text{-}top]\ (\lambda n.\ ln\ (primorial\ (nth\text{-}prime\ (n - 1))))$
by (*intro asymp-equiv-refl-ev eventually-mono*[*OF eventually-gt-at-top*[*of 0*]])
(*auto simp*: *primorial′-conv-primorial*)
also have ... $\sim[at\text{-}top]\ (\lambda n.\ nth\text{-}prime\ (n - 1))$
by (*intro asymp-equiv-compose′*[*OF - lim1*] *asymp-equiv-intros*)
also have ... $\sim[at\text{-}top]\ (\lambda n.\ real\ (n - 1) * ln\ (real\ (n - 1)))$
by (*intro asymp-equiv-compose′*[*OF - lim2*] *asymp-equiv-intros*)
also have ... $\sim[at\text{-}top]\ (\lambda n.\ n * ln\ n)$ **by** *real-asymp*
finally show *1*: $(\lambda k.\ ln\ (primorial'\ k)) \sim[at\text{-}top]\ (\lambda k.\ k * ln\ k)$ **.**

have $(\lambda k.\ ln\ (ln\ (primorial'\ k))) \sim[at\text{-}top]\ (\lambda n.\ ln\ (ln\ (primorial\ (nth\text{-}prime\ (n - 1)))))$
by (*intro asymp-equiv-refl-ev eventually-mono*[*OF eventually-gt-at-top*[*of 0*]])
(*auto simp*: *primorial′-conv-primorial*)
also have ... $\sim[at\text{-}top]\ (\lambda n.\ ln\ (nth\text{-}prime\ (n - 1)))$
by (*intro asymp-equiv-compose′*[*OF - lim1*] *asymp-equiv-intros*)
also have ... $\sim[at\text{-}top]\ (\lambda n.\ ln\ (real\ (n - 1)))$
by (*intro asymp-equiv-compose′*[*OF - lim2*] *asymp-equiv-intros*)
also have ... $\sim[at\text{-}top]\ (\lambda n.\ ln\ n)$ **by** *real-asymp*
finally show *2*: $(\lambda k.\ ln\ (ln\ (primorial'\ k))) \sim[at\text{-}top]\ (\lambda k.\ ln\ k)$ **.**

have $(\lambda k.\ ln\ (primorial'\ k)\ /\ ln\ (ln\ (primorial'\ k))) \sim[at\text{-}top]\ (\lambda k.\ (k * ln\ k)\ /\ ln\ k)$
by (*intro asymp-equiv-intros 1 2*)
also have ... $\sim[at\text{-}top]\ (\lambda k.\ k)$ **by** *real-asymp*
finally show $(\lambda k.\ ln\ (primorial'\ k)\ /\ ln\ (ln\ (primorial'\ k))) \sim[at\text{-}top]\ (\lambda k.\ k)$ **.**
qed

end

12.3 Consequences of the asymptotics of ψ and ϑ

Next, we will show some consequences of $\psi(x) \sim x$ and $\vartheta(x) \sim x$. To this end, we first show generically that any function $g = e^{x+o(x)}$ is $o(c^n)$ if $c > e$ and $\omega(c^n)$ if $c < e$.

locale *exp-asymp-equiv-linear* =
  fixes *f g* :: *real* ⇒ *real*
  assumes *f-asymp-equiv*: *f* ∼[*at-top*] (λ*x*. *x*)
  assumes *g*: *eventually* (λ*x*. *g x* = *exp* (*f x*)) *F*
begin

lemma
  fixes ε :: *real* **assumes** ε > *0*
  shows *smallo*: *g* ∈ *o*(λ*x*. *exp* ((*1* + ε) ∗ *x*))
    and *smallomega*: *g* ∈ ω(λ*x*. *exp* ((*1* − ε) ∗ *x*))
proof −
  have (λ*x*. *exp* (*f x*) / *exp* ((*1* + ε) ∗ *x*)) ∈ Θ(λ*x*. *exp* (((*f x* − *x*) / *x* − ε) ∗ *x*))
    by (*intro bigthetaI-cong eventually-mono*[*OF eventually-gt-at-top*[*of 1*]])
      (*simp-all add*: *divide-simps ring-distribs flip*: *exp-add exp-diff*)
  also have ((λ*x*. *exp* (((*f x* − *x*) / *x* − ε) ∗ *x*)) ⟶ *0*) *at-top*
  proof (*intro filterlim-compose*[*OF exp-at-bot*] *filterlim-tendsto-neg-mult-at-bot*)
    have *smallo*: (λ*x*. *f x* − *x*) ∈ *o*(λ*x*. *x*)
      using *f-asymp-equiv* **by** (*rule asymp-equiv-imp-diff-smallo*)
    show ((λ*x*. (*f x* − *x*) / *x* − ε) ⟶ *0* − ε) *at-top*
      by (*intro tendsto-diff smalloD-tendsto*[*OF smallo*] *tendsto-const*)
  qed (*use* ‹ε > *0*› **in** ‹*auto simp*: *filterlim-ident*›)
  hence (λ*x*. *exp* (((*f x* − *x*) / *x* − ε) ∗ *x*)) ∈ *o*(λ-. *1*)
    by (*intro smalloI-tendsto*) *auto*
  finally have (λ*x*. *exp* (*f x*)) ∈ *o*(λ*x*. *exp* ((*1* + ε) ∗ *x*))
    by (*simp add*: *landau-divide-simps*)
  also have *?this* ⟷ *g* ∈ *o*(λ*x*. *exp* ((*1* + ε) ∗ *x*))
    using *g* **by** (*intro landau-o.small.in-cong*) (*simp add*: *eq-commute*)
  finally show *g* ∈ *o*(λ*x*. *exp* ((*1* + ε) ∗ *x*)) **.**
next
  have (λ*x*. *exp* (*f x*) / *exp* ((*1* − ε) ∗ *x*)) ∈ Θ(λ*x*. *exp* (((*f x* − *x*) / *x* + ε) ∗ *x*))
    by (*intro bigthetaI-cong eventually-mono*[*OF eventually-gt-at-top*[*of 1*]])
      (*simp add*: *ring-distribs flip*: *exp-add exp-diff*)
  also have *filterlim* (λ*x*. *exp* (((*f x* − *x*) / *x* + ε) ∗ *x*)) *at-top at-top*
  proof (*intro filterlim-compose*[*OF exp-at-top*] *filterlim-tendsto-pos-mult-at-top*)
    have *smallo*: (λ*x*. *f x* − *x*) ∈ *o*(λ*x*. *x*)
      using *f-asymp-equiv* **by** (*rule asymp-equiv-imp-diff-smallo*)
    show ((λ*x*. (*f x* − *x*) / *x* + ε) ⟶ *0* + ε) *at-top*
      by (*intro tendsto-add smalloD-tendsto*[*OF smallo*] *tendsto-const*)
  qed (*use* ‹ε > *0*› **in** ‹*auto simp*: *filterlim-ident*›)
  hence (λ*x*. *exp* (((*f x* − *x*) / *x* + ε) ∗ *x*)) ∈ ω(λ-. *1*)
    by (*simp add*: *filterlim-at-top-iff-smallomega*)
  finally have (λ*x*. *exp* (*f x*)) ∈ ω(λ*x*. *exp* ((*1* − ε) ∗ *x*))
    by (*simp add*: *landau-divide-simps*)
  also have *?this* ⟷ *g* ∈ ω(λ*x*. *exp* ((*1* − ε) ∗ *x*))

```
    using g by (intro landau-omega.small.in-cong) (simp add: eq-commute)
  finally show g ∈ ω(λx. exp ((1 − ε) ∗ x)) .
qed

lemma smallo′:
  fixes c :: real assumes c > exp 1
  shows g ∈ o(λx. c powr x)
proof −
  have c > 0 by (rule le-less-trans[OF - assms]) auto
  from ‹c > 0› assms have exp 1 < exp (ln c)
    by (subst exp-ln) auto
  hence ln c > 1 by (subst (asm) exp-less-cancel-iff)
  hence g ∈ o(λx. exp ((1 + (ln c − 1)) ∗ x))
    using assms by (intro smallo) auto
  also have (λx. exp ((1 + (ln c − 1)) ∗ x)) = (λx. c powr x)
    using ‹c > 0› by (simp add: powr-def mult-ac)
  finally show ?thesis .
qed

lemma smallomega′:
  fixes c :: real assumes c ∈ {0<..<exp 1}
  shows g ∈ ω(λx. c powr x)
proof −
  from assms have exp 1 > exp (ln c)
    by (subst exp-ln) auto
  hence ln c < 1 by (subst (asm) exp-less-cancel-iff)
  hence g ∈ ω(λx. exp ((1 − (1 − ln c)) ∗ x))
    using assms by (intro smallomega) auto
  also have (λx. exp ((1 − (1 − ln c)) ∗ x)) = (λx. c powr x)
    using assms by (simp add: powr-def mult-ac)
  finally show ?thesis .
qed

end
```

The primorial fulfils $x\# = e^{\vartheta(x)}$ and is therefore one example of this.

```
context prime-number-theorem
begin

sublocale primorial: exp-asymp-equiv-linear ϑ λx. real (primorial x)
  using ϑ-asymptotics by unfold-locales (simp-all add: ln-primorial [symmetric])

end
```

The LCM of the first n natural numbers is equal to $e^{\psi(n)}$ and is therefore another example.

```
context prime-number-theorem
begin
```

sublocale *Lcm-upto*: *exp-asymp-equiv-linear* ψ $\lambda x.$ *real* (*Lcm* {*1*..*nat* $\lfloor x \rfloor$})
using ψ*-asymptotics* **by** *unfold-locales* (*simp-all flip*: *Lcm-upto-real-conv-*ψ)

end

12.4 Bounds on the prime ω function

Next, we will examine the asymptotic behaviour of the prime ω function $\omega(n)$, i. e. the number of distinct prime factors of n. These proofs are again taken from A. J. Hildebrand's lecture notes [2].

lemma *ln-gt-1*:
assumes $x > (3 :: real)$
shows *ln x* > *1*
proof −
have $x > exp\ 1$
using *exp-le assms* **by** *linarith*
hence *ln x* > *ln* (*exp 1*) **using** *assms* **by** (*subst ln-less-cancel-iff*) *auto*
thus *?thesis* **by** *simp*
qed

lemma (**in** *prime-number-theorem*) *primes-omega-primorial'-asymp-equiv*:
($\lambda k.$ *primes-omega* (*primorial'* *k*)) ∼[*at-top*]
($\lambda k.$ *ln* (*primorial'* *k*) / *ln* (*ln* (*primorial'* *k*)))
using *ln-over-ln-ln-primorial'-asymp-equiv* **by** (*simp add*: *asymp-equiv-sym*)

The number of distinct prime factors of n has maximal order $\ln n / \ln \ln n$:

theorem (**in** *prime-number-theorem*)
limsup-primes-omega: *limsup* ($\lambda n.$ *primes-omega n* / (*ln n* / *ln* (*ln n*))) = *1*
proof (*intro antisym*)
have ($\lambda k.$ *primes-omega* (*primorial'* *k*) / (*ln* (*primorial'* *k*) / *ln* (*ln* (*primorial'* *k*)))) ⟶ *1*
using *primes-omega-primorial'-asymp-equiv*
by (*intro asymp-equivD-strong eventually-mono*[*OF eventually-gt-at-top*[*of 1*]]) *auto*
hence *limsup* (($\lambda n.$ *primes-omega n* / (*ln n* / *ln* (*ln n*))) ∘ *primorial'*) = *ereal 1*
by (*intro lim-imp-Limsup tendsto-ereal*) *simp-all*
hence *1* = *limsup* (($\lambda n.$ *ereal* (*primes-omega n* / (*ln n* / *ln* (*ln n*)))) ∘ *primorial'*)
by (*simp add*: *o-def*)
also have ... ≤ *limsup* ($\lambda n.$ *primes-omega n* / (*ln n* / *ln* (*ln n*)))
using *strict-mono-primorial'* **by** (*rule limsup-subseq-mono*)
finally show *limsup* ($\lambda n.$ *primes-omega n* / (*ln n* / *ln* (*ln n*))) ≥ *1* .
next
show *limsup* ($\lambda n.$ *primes-omega n* / (*ln n* / *ln* (*ln n*))) ≤ *1*
unfolding *Limsup-le-iff*
proof *safe*
fix $C' :: ereal$ **assume** C': $C' > 1$
from *ereal-dense2*[*OF this*] **obtain** *C* **where** *C*: *C* > *1* *ereal C* < *C'* **by** *auto*

have (λk. *primes-omega* (*primorial'* *k*) / (*ln* (*primorial'* *k*) / *ln* (*ln* (*primorial'* *k*)))) $\longrightarrow$ *1*
(**is** *filterlim ?f - -*) **using** *primes-omega-primorial'-asymp-equiv*
by (*intro asymp-equivD-strong eventually-mono*[*OF eventually-gt-at-top*[*of 1*]]) *auto*
from *order-tendstoD*(*2*)[*OF this C*(*1*)]
have *eventually* (λk. *?f k* < *C*) *at-top* **.**
then obtain *k0* **where** *k0*: $\bigwedge k$. *k* $\geq$ *k0* $\Longrightarrow$ *?f k* < *C* **by** (*auto simp*: *eventually-at-top-linorder*)

have *eventually* (λn::*nat*. *max 3 k0* / (*ln n* / *ln* (*ln n*)) < *C*) *at-top*
using ‹*C* > *1*› **by** *real-asymp*
hence *eventually* (λn. *primes-omega n* / (*ln n* / *ln* (*ln n*)) $\leq$ *C*) *at-top*
using *eventually-gt-at-top*[*of primorial'* (*max k0 3*)]
proof *eventually-elim*
case (*elim n*)
define *k* **where** *k* = *primes-omega n*
define *m* **where** *m* = *primorial' k*
have *primorial' 3* $\leq$ *primorial'* (*max k0 3*)
by (*subst strict-mono-less-eq*[*OF strict-mono-primorial'*]) *auto*
also have ... < *n* **by** *fact*
finally have *n* > *30* **by** *simp*

show *?case*
proof (*cases k* $\geq$ *max 3 k0*)
case *True*
hence *m* $\geq$ *30*
using *strict-mono-less-eq*[*OF strict-mono-primorial'*, *of 3 k*] **by** (*simp add*: *m-def k-def*)
have *exp 1* ^ *3* $\leq$ (*3* ^ *3* :: *real*)
using *exp-le* **by** (*intro power-mono*) *auto*
also have ... < *m* **using** ‹*m* $\geq$ *30*› **by** *simp*
finally have *ln* (*exp 1* ^ *3*) < *ln m*
using ‹*m* $\geq$ *30*› **by** (*subst ln-less-cancel-iff*) *auto*
hence *ln m* > *3* **by** (*subst* (*asm*) *ln-realpow*) *auto*

have *primorial'* (*primes-omega n*) $\leq$ *n*
using ‹*n* > *30*› **by** (*intro primorial'-primes-omega-le*) *auto*
hence *m* $\leq$ *n* **unfolding** *m-def k-def* **using** *elim*
by (*auto simp*: *max-def*)
hence *primes-omega n* / (*ln n* / *ln* (*ln n*)) $\leq$ *k* / (*ln m* / *ln* (*ln m*))
unfolding *k-def* **using** *elim* ‹*m* $\geq$ *30*› *ln-gt-1*[*of n*] ‹*ln m* > *3*›
by (*intro frac-le*[*of primes-omega n*] *divide-ln-mono mult-pos-pos divide-pos-pos*) *auto*
also have ... = *?f k*
by (*simp add*: *k-def m-def*)
also have ... < *C*
by (*intro k0*) (*use True* **in** ‹*auto simp*: *k-def*›)
finally show *?thesis* **by** *simp*

```
    next
      case False
      hence primes-omega n / (ln n / ln (ln n)) ≤ max 3 k0 / (ln n / ln (ln n))
        using elim ln-gt-1[of n] ‹n > 30›
        by (intro divide-right-mono divide-nonneg-pos) (auto simp: k-def)
      also have ... < C
        using elim by simp
      finally show ?thesis by simp
    qed
  qed
  thus eventually (λn. ereal (primes-omega n / (ln n / ln (ln n))) < C') at-top
    by eventually-elim (rule le-less-trans[OF - C(2)], auto)
 qed
qed
```

12.5 Bounds on the divisor function

In this section, we shall examine the growth of the divisor function $\sigma_0(n)$. In particular, we will show that $\sigma_0(n) < 2^{c \ln n / \ln \ln n}$ for all sufficiently large n if $c > 1$ and $\sigma_0(n) > 2^{c \ln n / \ln \ln n}$ for infinitely many n if $c < 1$.

An equivalent statement is that $\ln(\sigma_0(n))$ has maximal order $\ln 2 \cdot \ln n / \ln \ln n$.

Following Apostol's somewhat didactic approach, we first show a generic bounding lemma for σ_0 that depends on some function f that we will specify later.

```
lemma divisor-count-bound-gen:
 fixes f :: nat ⇒ real
 assumes eventually (λn. f n ≥ 2) at-top
 defines c ≡ (8 / ln 2 :: real)
 defines g ≡ (λn. (ln n + c * f n * ln (ln n)) / (ln (f n)))
 shows eventually (λn. divisor-count n < 2 powr g n) at-top
proof −
 include prime-counting-syntax
 have eventually (λn::nat. 1 + log 2 n ≤ ln n ^ 2) at-top  by real-asymp
 thus eventually (λn. divisor-count n < 2 powr g n) at-top
   using eventually-gt-at-top[of 2] assms(1)
 proof eventually-elim
   fix n :: nat
   assume n: n > 2 and f n ≥ 2 and 1 + log 2 n ≤ ln n ^ 2

   define Pr where [simp]: Pr = prime-factors n
   define Pr1 where [simp]: Pr1 = {p∈Pr. p < f n}
   define Pr2 where [simp]: Pr2 = {p∈Pr. p ≥ f n}

   have exp 1 < real n
     using e-less-272 ‹n > 2› by linarith
   hence ln (exp 1) < ln (real n)
     using ‹n > 2› by (subst ln-less-cancel-iff) auto
   hence ln (ln n) > ln (ln (exp 1))
```

```
    by (subst ln-less-cancel-iff) auto
  hence ln (ln n) > 0 by simp

  define S2 where S2 = (∑ p∈Pr2. multiplicity p n)
  have f n ^ S2 = (∏ p∈Pr2. f n ^ multiplicity p n)
    by (simp add: S2-def power-sum)
  also have ... ≤ (∏ p∈Pr2. real p ^ multiplicity p n)
    using ‹f n ≥ 2› by (intro prod-mono conjI power-mono) auto
  also from ‹n > 2› have ... ≤ (∏ p∈Pr. real p ^ multiplicity p n)
      by (intro prod-mono2 one-le-power) (auto simp: in-prime-factors-iff dest:
prime-gt-0-nat)
  also have ... = n
    using ‹n > 2› by (subst prime-factorization-nat[of n]) auto
  finally have f n ^ S2 ≤ n .
  hence ln (f n ^ S2) ≤ ln n
    using n ‹f n ≥ 2› by (subst ln-le-cancel-iff) auto
  hence S2 ≤ ln n / ln (f n)
    using ‹f n ≥ 2› by (simp add: field-simps ln-realpow)

  have le-twopow: Suc a ≤ 2 ^ a for a :: nat by (induction a) auto
  have (∏ p∈Pr2. Suc (multiplicity p n)) ≤ (∏ p∈Pr2. 2 ^ multiplicity p n)
    by (intro prod-mono conjI le-twopow) auto
  also have ... = 2 ^ S2
    by (simp add: S2-def power-sum)
  also have ... = 2 powr real S2
    by (subst powr-realpow) auto
  also have ... ≤ 2 powr (ln n / ln (f n))
    by (intro powr-mono ‹S2 ≤ ln n / ln (f n)›) auto
  finally have bound2: real (∏ p∈Pr2. Suc (multiplicity p n)) ≤ 2 powr (ln n /
ln (f n))
    by simp

  have multiplicity-le: multiplicity p n ≤ log 2 n if p: p ∈ Pr for p
  proof −
    from p have 2 ^ multiplicity p n ≤ p ^ multiplicity p n
      by (intro power-mono) (auto simp: in-prime-factors-iff dest: prime-gt-1-nat)
    also have ... = (∏ p∈{p}. p ^ multiplicity p n) by simp
   also from p have (∏ p∈{p}. p ^ multiplicity p n) ≤ (∏ p∈Pr. p ^ multiplicity
p n)
      by (intro dvd-imp-le prod-dvd-prod-subset)
        (auto simp: in-prime-factors-iff dest: prime-gt-0-nat)
    also have ... = n
      using n by (subst prime-factorization-nat[of n]) auto
    finally have 2 ^ multiplicity p n ≤ n .
    hence log 2 (2 ^ multiplicity p n) ≤ log 2 n
      using n by (subst log-le-cancel-iff) auto
    thus multiplicity p n ≤ log 2 n
      by (subst (asm) log-nat-power) auto
  qed
```

have $(\prod p{\in}Pr1.\ Suc\ (multiplicity\ p\ n)) = exp\ (\sum p{\in}Pr1.\ ln\ (multiplicity\ p\ n + 1))$
by (*simp add*: *exp-sum*)
also have $(\sum p{\in}Pr1.\ ln\ (multiplicity\ p\ n + 1)) \leq (\sum p{\in}Pr1.\ 2 * ln\ (ln\ n))$
proof (*intro sum-mono*)
fix *p* **assume** *p*: $p \in Pr1$
have $ln\ (multiplicity\ p\ n + 1) \leq ln\ (1 + log\ 2\ n)$
using *p multiplicity-le*[*of p*] **by** (*subst ln-le-cancel-iff*) *auto*
also have $\ldots \leq ln\ (ln\ n\ \widehat{}\ 2)$
using ‹$n > 2$› ‹$1 + log\ 2\ n \leq ln\ n\ \widehat{}\ 2$›
by (*subst ln-le-cancel-iff*) (*auto intro*: *add-pos-nonneg*)
also have $\ldots = 2 * ln\ (ln\ n)$
using ‹$n > 2$› **by** (*simp add*: *ln-realpow*)
finally show $ln\ (multiplicity\ p\ n + 1) \leq 2 * ln\ (ln\ n)$ **.**
qed
also have $\ldots = 2 * ln\ (ln\ n) * card\ Pr1$
by *simp*
also have *finite* $\{p.\ prime\ p \wedge real\ p \leq f\ n\}$
by (*rule finite-subset*[*of* - $\{..nat\ \lfloor f\ n \rfloor\}$]) (*auto simp*: *le-nat-iff le-floor-iff*)
hence $card\ Pr1 \leq card\ \{p.\ prime\ p \wedge real\ p \leq f\ n\}$
by (*intro card-mono*) *auto*
also have *real* $\ldots = \pi\ (f\ n)$
by (*simp add*: *primes-pi-def prime-sum-upto-def*)
also have $\ldots < 4 * (f\ n\ /\ ln\ (f\ n))$
using ‹$f\ n \geq 2$› **by** (*intro* π*-upper-bound''*) *auto*
also have $exp\ (2 * ln\ (ln\ (real\ n)) * (4 * (f\ n\ /\ ln\ (f\ n)))) =$
$2\ powr\ (c * f\ n * ln\ (ln\ n)\ /\ ln\ (f\ n))$
by (*simp add*: *powr-def c-def*)
finally have *bound1*: $(\prod p{\in}Pr1.\ Suc\ (multiplicity\ p\ n)) <$
$2\ powr\ (c * f\ n * ln\ (ln\ (real\ n))\ /\ ln\ (f\ n))$
using ‹$ln\ (ln\ n) > 0$› **by** (*simp add*: *mult-strict-left-mono*)

have *divisor-count* $n = (\prod p{\in}Pr.\ Suc\ (multiplicity\ p\ n))$
using *n* **by** (*subst divisor-count.prod-prime-factors'*) *auto*
also have $Pr = Pr1 \cup Pr2$ **by** *auto*
also have *real* $(\prod p{\in}\ldots.\ Suc\ (multiplicity\ p\ n)) =$
$real\ ((\prod p{\in}Pr1.\ Suc\ (multiplicity\ p\ n)) * (\prod p{\in}Pr2.\ Suc\ (multiplicity\ p\ n)))$
by (*subst prod.union-disjoint*) *auto*
also have $\ldots < 2\ powr\ (c * f\ n * ln\ (ln\ (real\ n))\ /\ ln\ (f\ n)) * 2\ powr\ (ln\ n\ /\ ln\ (f\ n))$
unfolding *of-nat-mult*
by (*intro mult-less-le-imp-less bound1 bound2*) (*auto intro*!: *prod-nonneg prod-pos*)
also have $\ldots = 2\ powr\ g\ n$
by (*simp add*: *g-def add-divide-distrib powr-add*)
finally show *real* $(divisor\text{-}count\ n) < 2\ powr\ g\ n$ **.**

qed
qed

Now, Apostol explains that one can choose $f(n) := \ln n/(\ln \ln n)^2$ to obtain the desired bound.

proposition *divisor-count-upper-bound*:
fixes ε :: *real*
assumes $\varepsilon > 0$
shows *eventually* (λn. *divisor-count n* < *2 powr* ((*1* + ε) * *ln n* / *ln* (*ln n*))) *at-top*
proof −
define *c* :: *real* **where** *c* = *8* / *ln 2*
define *f* :: *nat* ⇒ *real* **where** *f* = (λn. *ln n* / (*ln* (*ln n*)) ^ *2*)
define *g* **where** *g* = (λn. (*ln n* + *c* * *f n* * *ln* (*ln n*)) / (*ln* (*f n*)))

have *eventually* (λn. *divisor-count n* < *2 powr g n*) *at-top*
unfolding *g-def c-def f-def* **by** (*rule divisor-count-bound-gen*) *real-asymp+*
moreover have *eventually* (λn. *2 powr g n* < *2 powr* ((*1* + ε) * *ln n* / *ln* (*ln n*))) *at-top*
using ‹$\varepsilon > 0$› **unfolding** *g-def c-def f-def* **by** *real-asymp*
ultimately show *eventually* (λn. *divisor-count n* < *2 powr* ((*1* + ε) * *ln n* / *ln* (*ln n*))) *at-top*
by *eventually-elim* (*rule less-trans*)
qed

Next, we will examine the 'worst case'. Since any prime factor of n with multiplicity k contributes a factor of $k+1$, it is intuitively clear that $\sigma_0(n)$ is largest w. r. t. n if it is a product of small distinct primes.

We show that indeed, if $n := x\#$ (where $x\#$ denotes the primorial), we have $\sigma_0(n) = 2^{\pi(x)}$, which, by the Prime Number Theorem, indeed exceeds $c \ln n / \ln \ln n$.

theorem (**in** *prime-number-theorem*) *divisor-count-primorial-gt*:
assumes $\varepsilon > 0$
defines $h \equiv$ *primorial*
shows *eventually* (λx. *divisor-count* (*h x*) > *2 powr* ((*1* − ε) * *ln* (*h x*) / *ln* (*ln* (*h x*)))) *at-top*
proof −
have (λx. (*1* − ε) * *ln* (*h x*) / *ln* (*ln* (*h x*))) ∼[*at-top*] (λx. (*1* − ε) * ϑ *x* / *ln* (ϑ *x*))
by (*simp add*: *h-def ln-primorial*)
also have ... ∼[*at-top*] (λx. (*1* − ε) * *x* / *ln x*)
by (*intro asymp-equiv-intros* ϑ*-asymptotics ln-*ϑ*-asymp-equiv*)
finally have ∗: (λx. (*1* − ε) * *ln* (*h x*) / *ln* (*ln* (*h x*))) ∼[*at-top*] (λx. (*1* − ε) * *x* / *ln x*)
by *simp*
have (λx. (*1* − ε) * *ln* (*h x*) / (*ln* (*ln* (*h x*))) − (*1* − ε) * *x* / *ln x*) ∈ *o*(λx. (*1* − ε) * *x* / *ln x*)
using *asymp-equiv-imp-diff-smallo*[*OF* ∗] **by** *simp*

```
  also have ?this ⟷ (λx. (1 − ε) ∗ ln (h x) / (ln (ln (h x))) − x / ln x + ε ∗ x
/ ln x)
              ∈ o(λx. (1 − ε) ∗ x / ln x)
    by (intro landau-o.small.in-cong eventually-mono[OF eventually-gt-at-top[of
1]])
      (auto simp: field-simps)
  also have (λx. (1 − ε) ∗ x / ln x) ∈ O(λx. x / ln x)
    by real-asymp
  finally have (λx. (1 − ε) ∗ ln (h x) / (ln (ln (h x))) − x / ln x + ε ∗ x / ln x)
                 ∈ o(λx. x / ln x) .
  hence (λx. (1 − ε) ∗ ln (h x) / (ln (ln (h x))) − x / ln x + ε ∗ x / ln x − (π x
− x / ln x))
          ∈ o(λx. x / ln x)
   by (intro sum-in-smallo[OF - asymp-equiv-imp-diff-smallo] prime-number-theorem)
  hence (λx. (1 − ε) ∗ ln (h x) / (ln (ln (h x))) − π x + ε ∗ (x / ln x)) ∈ o(λx.
ε ∗ (x / ln x))
    using ‹ε > 0› by (subst landau-o.small.cmult) (simp-all add: algebra-simps)
  hence (λx. (1 − ε) ∗ ln (h x) / (ln (ln (h x))) − π x) ∼[at-top] (λx. −ε ∗ (x /
ln x))
    by (intro smallo-imp-asymp-equiv) auto
  hence eventually (λx. (1 − ε) ∗ ln (h x) / (ln (ln (h x))) − π x < 0 ⟷
                      −ε ∗ (x / ln x) < 0) at-top
    by (rule asymp-equiv-eventually-neg-iff)
  moreover have eventually (λx. −ε ∗ (x / ln x) < 0) at-top
    using ‹ε > 0› by real-asymp
  ultimately have eventually (λx. (1 − ε) ∗ ln (h x) / ln (ln (h x)) < π x) at-top
    by eventually-elim simp
  thus eventually (λx. divisor-count (h x) > 2 powr ((1 − ε) ∗ ln (h x) / ln (ln
(h x)))) at-top
  proof eventually-elim
    case (elim x)
    hence 2 powr ((1 − ε) ∗ ln (h x) / ln (ln (h x))) < 2 powr π x
      by (intro powr-less-mono) auto
    thus ?case by (simp add: divisor-count-primorial h-def)
  qed
qed
```

Since $h(x) \longrightarrow \infty$, this gives us our infinitely many values of n that exceed the bound.

```
corollary (in prime-number-theorem) divisor-count-lower-bound:
  assumes ε > 0
  shows   frequently (λn. divisor-count n > 2 powr ((1 − ε) ∗ ln n / ln (ln n)))
at-top
proof −
  define h where h = primorial
  have eventually (λn. divisor-count n > 2 powr ((1 − ε) ∗ ln n / ln (ln n)))
         (filtermap h at-top)
    using divisor-count-primorial-gt[OF assms] by (simp add: eventually-filtermap
h-def)
```

hence *frequently* (λ*n*. *divisor-count n* > *2 powr* ((*1* − ε) ∗ *ln n* / *ln* (*ln n*)))
(*filtermap h at-top*)
by (*intro eventually-frequently*) (*auto simp*: *filtermap-bot-iff*)
moreover from *this* **and** *primorial-at-top*
have *filtermap h at-top* ≤ *at-top* **by** (*simp add*: *filterlim-def h-def*)
ultimately show *?thesis*
by (*rule frequently-mono-filter*)
qed

A different formulation that is not quite as tedious to prove is this one:

lemma (**in** *prime-number-theorem*) *ln-divisor-count-primorial′-asymp-equiv*:
(λ*k*. *ln* (*divisor-count* (*primorial′ k*))) ∼[*at-top*]
(λ*k*. *ln 2* ∗ *ln* (*primorial′ k*) / *ln* (*ln* (*primorial′ k*)))
proof −
have (λ*k*. *ln 2* ∗ (*ln* (*primorial′ k*) / *ln* (*ln* (*primorial′ k*)))) ∼[*at-top*] (λ*k*. *ln 2* ∗ *k*)
by (*intro asymp-equiv-intros ln-over-ln-ln-primorial′-asymp-equiv*)
also have . . . ∼[*at-top*] (λ*k*. *ln* (*divisor-count* (*primorial′ k*)))
by (*simp add*: *ln-realpow mult-ac*)
finally show *?thesis* **by** (*simp add*: *asymp-equiv-sym mult-ac*)
qed

It follows that the maximal order of the divisor function is $\ln 2 \cdot \ln n / \ln \ln n$.

theorem (**in** *prime-number-theorem*) *limsup-divisor-count*:
limsup (λ*n*. *ln* (*divisor-count n*) ∗ *ln* (*ln n*) / *ln n*) = *ln 2*
proof (*intro antisym*)
let *?h* = *primorial′*
have *2* ^ *k* = (*1* :: *real*) ⟷ *k* = *0* **for** *k* :: *nat*
using *power-eq-1-iff*[*of 2*::*real k*] **by** *auto*
hence (λ*k*. *ln* (*divisor-count* (*?h k*)) / (*ln 2* ∗ *ln* (*?h k*) / *ln* (*ln* (*?h k*)))) ⟶ *1*
using *ln-divisor-count-primorial′-asymp-equiv*
by (*intro asymp-equivD-strong eventually-mono*[*OF eventually-gt-at-top*[*of 1*]])
(*auto simp*: *power-eq-1-iff*)
hence (λ*k*. *ln* (*divisor-count* (*?h k*)) / (*ln 2* ∗ *ln* (*?h k*) / *ln* (*ln* (*?h k*))) ∗ *ln 2*) ⟶ *1* ∗ *ln 2*
by (*rule tendsto-mult*) *auto*
hence (λ*k*. *ln* (*divisor-count* (*?h k*)) / (*ln* (*?h k*) / *ln* (*ln* (*?h k*)))) ⟶ *ln 2*
by *simp*
hence *limsup* ((λ*n*. *ln* (*divisor-count n*) ∗ *ln* (*ln n*) / *ln n*) ∘ *primorial′*) = *ereal* (*ln 2*)
by (*intro lim-imp-Limsup tendsto-ereal*) *simp-all*
hence *ln 2* = *limsup* ((λ*n*. *ereal* (*ln* (*divisor-count n*) ∗ *ln* (*ln n*) / *ln n*)) ∘ *primorial′*)
by (*simp add*: *o-def*)
also have . . . ≤ *limsup* (λ*n*. *ln* (*divisor-count n*) ∗ *ln* (*ln n*) / *ln n*)
using *strict-mono-primorial′* **by** (*rule limsup-subseq-mono*)
finally show *limsup* (λ*n*. *ln* (*divisor-count n*) ∗ *ln* (*ln n*) / *ln n*) ≥ *ln 2* .
next

```
  show limsup (λn. ln (divisor-count n) * ln (ln n) / ln n) ≤ ln 2
    unfolding Limsup-le-iff
  proof safe
    fix C' assume C' > ereal (ln 2)
    from ereal-dense2[OF this] obtain C where C: C > ln 2 ereal C < C' by
auto
    define ε where ε = (C / ln 2) - 1
    from C have ε > 0 by (simp add: ε-def)

    have eventually (λn::nat. ln (ln n) > 0) at-top by real-asymp
    hence eventually (λn. ln (divisor-count n) * ln (ln n) / ln n < C) at-top
      using divisor-count-upper-bound[OF ‹ε > 0›] eventually-gt-at-top[of 1]
    proof eventually-elim
      case (elim n)
      hence ln (divisor-count n) < ln (2 powr ((1 + ε) * ln n / ln (ln n)))
        by (subst ln-less-cancel-iff) auto
      also have ... = (1 + ε) * ln 2 * ln n / ln (ln n)
        by (simp add: ln-powr)
      finally have ln (divisor-count n) * ln (ln n) / ln n < (1 + ε) * ln 2
        using elim by (simp add: field-simps)
      also have ... = C by (simp add: ε-def)
      finally show ?case .
    qed
    thus eventually (λn. ereal (ln (divisor-count n) * ln (ln n) / ln n) < C') at-top
      by eventually-elim (rule less-trans[OF - C(2)], auto)
  qed
qed
```

12.6 Mertens' Third Theorem

In this section, we will show that

$$\prod_{p \leq x} \left(1 - \frac{1}{p}\right) = \frac{C}{\ln x} + O\left(\frac{1}{\ln^2 x}\right)$$

with explicit bounds for the factor in the 'Big-O'. Here, C is the following constant:

```
definition third-mertens-const :: real where
  third-mertens-const =
     exp (-(∑ p::nat. if prime p then -ln (1 - 1 / real p) - 1 / real p else 0) -
meissel-mertens)
```

This constant is actually equal to $e^{-\gamma}$ where γ is the Euler–Mascheroni constant, but showing this is quite a bit of work, which we shall not do here.

```
lemma third-mertens-const-pos: third-mertens-const > 0
  by (simp add: third-mertens-const-def)

theorem
```

defines $C \equiv$ *third-mertens-const*
shows *mertens-third-theorem-strong*:
$eventually\ (\lambda x.\ |(\prod p \mid prime\ p \wedge real\ p \leq x.\ 1 - 1\ /\ p) - C\ /\ ln\ x| \leq 10 * C\ /\ ln\ x\ \hat{}\ 2)\ at\text{-}top$
and *mertens-third-theorem*:
$(\lambda x.\ (\prod p \mid prime\ p \wedge real\ p \leq x.\ 1 - 1\ /\ p) - C\ /\ ln\ x) \in O(\lambda x.\ 1\ /\ ln\ x\ \hat{}\ 2)$
proof −
define *Pr* **where** $Pr = (\lambda x.\ \{p.\ prime\ p \wedge real\ p \leq x\})$
define $a :: nat \Rightarrow real$
where $a = (\lambda p.\ if\ prime\ p\ then\ -ln\ (1 - 1\ /\ real\ p) - 1\ /\ real\ p\ else\ 0)$
define *B* **where** $B = suminf\ a$
have *C-eq*: $C = exp\ (-B - meissel\text{-}mertens)$
by (*simp add*: *B-def a-def third-mertens-const-def C-def*)
have *fin*: *finite* $(Pr\ x)$ **for** *x*
by (*rule finite-subset*[*of* - $\{..nat\ \lfloor x \rfloor\}$]) (*auto simp*: *Pr-def le-nat-iff le-floor-iff*)

define *S* **where** $S = (\lambda x.\ (\sum p{\in}Pr\ x.\ ln\ (1 - 1\ /\ p)))$
define *R* **where** $R = (\lambda x.\ S\ x + ln\ (ln\ x) + (B + meissel\text{-}mertens))$

have *exp-S*: $exp\ (S\ x) = (\prod p{\in}Pr\ x.\ (1 - 1\ /\ p))$ **for** *x*
proof −
have $exp\ (S\ x) = (\prod p{\in}Pr\ x.\ exp\ (ln\ (1 - 1\ /\ p)))$
by (*simp add*: *S-def exp-sum fin*)
also have $\ldots = (\prod p{\in}Pr\ x.\ (1 - 1\ /\ p))$
by (*intro prod.cong*) (*auto simp*: *Pr-def dest*: *prime-gt-1-nat*)
finally show *?thesis* **.**
qed

have *a-nonneg*: $a\ n \geq 0$ **for** *n*
proof (*cases prime n*)
case *True*
hence $ln\ (1 - 1\ /\ n) \leq -(1\ /\ n)$
by (*intro ln-one-minus-pos-upper-bound*) (*auto dest*: *prime-gt-1-nat*)
thus *?thesis* **by** (*auto simp*: *a-def*)
qed (*auto simp*: *a-def*)

have *summable a*
proof (*rule summable-comparison-test-bigo*)
have $a \in O(\lambda n.\ -ln\ (1 - 1\ /\ n) - 1\ /\ n)$
by (*intro bigoI*[*of* - *1*]) (*auto simp*: *a-def*)
also have $(\lambda n{::}nat.\ -ln\ (1 - 1\ /\ n) - 1\ /\ n) \in O(\lambda n.\ 1\ /\ n\ \hat{}\ 2)$
by *real-asymp*
finally show $a \in O(\lambda n.\ 1\ /\ real\ n\ \hat{}\ 2)$ **.**
next
show *summable* $(\lambda n.\ norm\ (1\ /\ real\ n\ \hat{}\ 2))$
using *inverse-power-summable*[*of 2*] **by** (*simp add*: *field-simps*)
qed

```
  have eventually (λn. a n ≤ 1 / n − 1 / Suc n) at-top
  proof −
    have eventually (λn::nat. −ln (1 − 1 / n) − 1 / n ≤ 1 / n − 1 / Suc n)
at-top
      by real-asymp
    thus ?thesis using eventually-gt-at-top[of 1]
      by eventually-elim (auto simp: a-def of-nat-diff field-simps)
  qed
  hence eventually (λm. ∀ n≥m. a n ≤ 1 / n − 1 / Suc n) at-top
    by (rule eventually-all-ge-at-top)
  hence eventually (λx. ∀ n≥nat ⌊x⌋. a n ≤ 1 / n − 1 / Suc n) at-top
    by (rule eventually-compose-filterlim)
     (intro filterlim-compose[OF filterlim-nat-sequentially] filterlim-floor-sequentially)
  hence eventually (λx. B − sum-upto a x ≤ 1 / x) at-top
    using eventually-ge-at-top[of 1::real]
  proof eventually-elim
    case (elim x)
    have a-le: a n ≤ 1 / real n − 1 / real (Suc n) if real n ≥ x for n
    proof −
      from that and ‹x ≥ 1› have n ≥ nat ⌊x⌋ by linarith
      with elim and that show ?thesis by auto
    qed
    define m where m = Suc (nat ⌊x⌋)
    have telescope: (λn. 1 / (n + m) − 1 / (Suc n + m)) sums (1 / real (0 + m)
− 0)
      by (intro telescope-sums′) real-asymp

    have B − (∑ n<m. a n) = (∑ n. a (n + m))
      unfolding B-def sum-upto-altdef m-def using ‹summable a›
      by (subst suminf-split-initial-segment[of - Suc (nat ⌊x⌋)]) auto
    also have (∑ n<m. a n) = sum-upto a x
      unfolding m-def sum-upto-altdef by (intro sum.mono-neutral-right) (auto
simp: a-def)
    also have (∑ n. a (n + m)) ≤ (∑ n. 1 / (n + m) − 1 / (Suc n + m))
    proof (intro suminf-le allI)
      show summable (λn. a (n + m))
        by (rule summable-ignore-initial-segment) fact+
    next
      show summable (λn. 1 / (n + m) − 1 / (Suc n + m))
        using telescope by (rule sums-summable)
    next
      fix n :: nat
      have x ≤ n + m unfolding m-def using ‹x ≥ 1› by linarith
      thus a (n + m) ≤ 1 / (n + m) − 1 / (Suc n + m)
        using a-le[of n + m] ‹x ≥ 1› by simp
    qed
    also have ... = 1 / m
      using telescope by (simp add: sums-iff)
    also have x ≤ m m > 0
```

```
    unfolding m-def using ‹x ≥ 1› by linarith+
  hence 1 / m ≤ 1 / x
    using ‹x ≥ 1› by (intro divide-left-mono) (auto simp: m-def)
  finally show ?case .
 qed

 moreover have eventually (λx::real. 1 / x ≤ 1 / ln x) at-top by real-asymp
 ultimately have eventually (λx. B − sum-upto a x ≤ 1 / ln x) at-top
  by eventually-elim (rule order.trans)

 hence eventually (λx. |R x| ≤ 5 / ln x) at-top
  using eventually-ge-at-top[of 2]
 proof eventually-elim
  case (elim x)
   have |(B − sum-upto a x) − (prime-sum-upto (λp. 1 / p) x − ln (ln x) −
meissel-mertens)| ≤
          1 / ln x + 4 / ln x
  proof (intro order.trans[OF abs-triangle-ineq4 add-mono])
    show |prime-sum-upto (λp. 1 / real p) x − ln (ln x) − meissel-mertens| ≤ 4
/ ln x
      by (intro mertens-second-theorem ‹x ≥ 2›)
    have sum-upto a x ≤ B
        unfolding B-def sum-upto-def by (intro sum-le-suminf ‹summable a›
a-nonneg ballI) auto
    thus |B − sum-upto a x| ≤ 1 / ln x
      using elim by linarith
  qed
  also have sum-upto a x = prime-sum-upto (λp. −ln (1 − 1 / p) − 1 / p) x
     unfolding sum-upto-def prime-sum-upto-altdef1 a-def by (intro sum.cong
allI) auto
  also have ... = −S x − prime-sum-upto (λp. 1 / p) x
    by (simp add: a-def S-def Pr-def prime-sum-upto-def sum-subtractf sum-negf)
  finally show |R x| ≤ 5 / ln x
    by (simp add: R-def)
 qed

 moreover have eventually (λx::real. |5 / ln x| < 1 / 2) at-top by real-asymp
 ultimately have eventually (λx. exp (R x) − 1 ∈ {−5 / ln x..10 / ln x}) at-top
  using eventually-gt-at-top[of 1]
 proof eventually-elim
  case (elim x)
  have exp (R x) ≤ exp (5 / ln x)
    using elim by simp
  also have ... ≤ 1 + 10 / ln x
     using real-exp-bound-lemma[of 5 / ln x] elim by (simp add: abs-if split:
if-splits)
  finally have le: exp (R x) ≤ 1 + 10 / ln x .

  have 1 + (−5 / ln x) ≤ exp (−5 / ln x)
```

```
    by (rule exp-ge-add-one-self)
  also have exp (−5 / ln x) ≤ exp (R x)
    using elim by simp
  finally have exp (R x) ≥ 1 − 5 / ln x by simp
  with le show ?case by simp
 qed

 thus eventually (λx. |(∏ p∈Pr x. 1 − 1 / real p) − C / ln x| ≤ 10 * C / ln x
^ 2) at-top
  using eventually-gt-at-top[of exp 1] eventually-gt-at-top[of 1]
 proof eventually-elim
  case (elim x)
  have |exp (R x) − 1| ≤ 10 / ln x
    using elim by (simp add: abs-if)
  from elim have |exp (S x) − C / ln x| = C / ln x * |exp (R x) − 1|
    by (simp add: R-def exp-add C-eq exp-diff exp-minus field-simps)
  also have ... ≤ C / ln x * (10 / ln x)
    using elim by (intro mult-left-mono) (auto simp: C-eq)
  finally show ?case by (simp add: exp-S power2-eq-square mult-ac)
 qed

 thus (λx. (∏ p∈Pr x. 1 − 1 / real p) − C / ln x) ∈ O(λx. 1 / ln x ^ 2)
  by (intro bigoI[of - 10 * C]) auto
qed

lemma mertens-third-theorem-asymp-equiv:
 (λx. (∏ p | prime p ∧ real p ≤ x. 1 − 1 / real p)) ∼[at-top]
   (λx. third-mertens-const / ln x)
 by (rule smallo-imp-asymp-equiv[OF landau-o.big-small-trans[OF mertens-third-theorem]])
   (use third-mertens-const-pos in real-asymp)
```

We now show an equivalent version where $\prod_{p \leq x}(1 - 1/p)$ is replaced by $\prod_{i=1}^{k}(1 - 1/p_i)$:

```
lemma mertens-third-convert:
 assumes n > 0
 shows (∏ k<n. 1 − 1 / real (nth-prime k)) =
        (∏ p | prime p ∧ p ≤ nth-prime (n − 1). 1 − 1 / p)
proof −
 have primorial′ n = primorial (nth-prime (n − 1))
  using assms by (simp add: primorial′-conv-primorial)
 also have real (totient ...) =
            primorial′ n * (∏ p | prime p ∧ p ≤ nth-prime (n − 1). 1 − 1 / p)
  using assms by (subst totient-primorial) (auto simp: primorial′-conv-primorial)
 finally show ?thesis
  by (simp add: totient-primorial′)
qed

lemma (in prime-number-theorem) mertens-third-theorem-asymp-equiv′:
 (λn. (∏ k<n. 1 − 1 / nth-prime k)) ∼[at-top] (λx. third-mertens-const / ln x)
```

```
proof −
  have lim: filterlim (λn. real (nth-prime (n − 1))) at-top at-top
    by (intro filterlim-compose[OF filterlim-real-sequentially]
          filterlim-compose[OF nth-prime-at-top]) real-asymp
  have (λn. (∏k<n. 1 − 1 / nth-prime k)) ∼[at-top]
        (λn. (∏p | prime p ∧ real p ≤ real (nth-prime (n − 1)). 1 − 1 / p))
    by (intro asymp-equiv-refl-ev eventually-mono[OF eventually-gt-at-top[of 0]])
      (subst mertens-third-convert, auto)
  also have ... ∼[at-top] (λn. third-mertens-const / ln (real (nth-prime (n − 1))))
    by (intro asymp-equiv-compose'[OF mertens-third-theorem-asymp-equiv lim])
  also have ... ∼[at-top] (λn. third-mertens-const / ln (real (n − 1)))
   by (intro asymp-equiv-intros asymp-equiv-compose'[OF ln-nth-prime-asymp-equiv])
real-asymp
  also have ... ∼[at-top] (λn. third-mertens-const / ln (real n))
    using third-mertens-const-pos by real-asymp
  finally show ?thesis .
qed
```

12.7 Bounds on Euler's totient function

Similarly to the divisor function, we will show that $\varphi(n)$ has minimal order $Cn/\ln\ln n$.

The first part is to show the lower bound:

```
theorem totient-lower-bound:
  fixes ε :: real
  assumes ε > 0
  defines C ≡ third-mertens-const
  shows eventually (λn. totient n > (1 − ε) * C * n / ln (ln n)) at-top
proof −
  include prime-counting-syntax
  define f :: nat ⇒ nat where f = (λn. card {p∈prime-factors n. p > ln n})

  define lb1 where lb1 = (λn::nat. (∏p | prime p ∧ real p ≤ ln n. 1 − 1 / p))
  define lb2 where lb2 = (λn::nat. (1 − 1 / ln n) powr (ln n / ln (ln n)))
  define lb1' where lb1' = (λn::nat. C / ln (ln n) − 10 * C / ln (ln n) ^ 2)

  have eventually (λn::nat. 1 + log 2 n ≤ ln n ^ 2) at-top  by real-asymp
  hence eventually (λn. totient n / n ≥ lb1 n * lb2 n) at-top
    using eventually-gt-at-top[of 2]
  proof eventually-elim
    fix n :: nat
    assume n: n > 2 and 1 + log 2 n ≤ ln n ^ 2

    define Pr where [simp]: Pr = prime-factors n
    define Pr1 where [simp]: Pr1 = {p∈Pr. p ≤ ln n}
    define Pr2 where [simp]: Pr2 = {p∈Pr. p > ln n}

    have exp 1 < real n
```

using *e-less-272* ‹$n > 2$› **by** *linarith*
hence $ln\ (exp\ 1) < ln\ (real\ n)$
using ‹$n > 2$› **by** (*subst ln-less-cancel-iff*) *auto*
hence $1 < ln\ n$ **by** *simp*
hence $ln\ (ln\ n) > ln\ (ln\ (exp\ 1))$
by (*subst ln-less-cancel-iff*) *auto*
hence $ln\ (ln\ n) > 0$ **by** *simp*

have $ln\ n\ \hat{}\ f\ n \leq (\prod p{\in}Pr2.\ ln\ n)$
by (*simp add*: *f-def*)
also have $\ldots \leq real\ (\prod p{\in}Pr2.\ p)$ **unfolding** *of-nat-prod*
by (*intro prod-mono*) (*auto dest*: *prime-gt-1-nat simp*: *in-prime-factors-iff*)
also **{**
have $(\prod p{\in}Pr2.\ p)\ dvd\ (\prod p{\in}Pr2.\ p\ \hat{}\ multiplicity\ p\ n)$
by (*intro prod-dvd-prod dvd-power*) (*auto simp*: *prime-factors-multiplicity*)
also have $\ldots\ dvd\ (\prod p{\in}Pr.\ p\ \hat{}\ multiplicity\ p\ n)$
by (*intro prod-dvd-prod-subset2*) *auto*
also have $\ldots = n$
using ‹$n > 2$› **by** (*subst* (*2*) *prime-factorization-nat*[*of n*]) *auto*
finally have $(\prod p{\in}Pr2.\ p) \leq n$
using ‹$n > 2$› **by** (*intro dvd-imp-le*) *auto*
}
finally have $ln\ (ln\ n\ \hat{}\ f\ n) \leq ln\ n$
using ‹$n > 2$› **by** (*subst ln-le-cancel-iff*) *auto*
also have $ln\ (ln\ n\ \hat{}\ f\ n) = f\ n * ln\ (ln\ n)$
using ‹$n > 2$› **by** (*subst ln-realpow*) *auto*
finally have *f-le*: $f\ n \leq ln\ n\ /\ ln\ (ln\ n)$
using ‹$ln\ (ln\ n) > 0$› **by** (*simp add*: *field-simps*)

have $(1 - 1\ /\ ln\ n)\ powr\ (ln\ n\ /\ ln\ (ln\ n)) \leq (1 - 1\ /\ ln\ n)\ powr\ (real\ (f\ n))$
using ‹$n > 2$› **and** ‹$ln\ n > 1$› **by** (*intro powr-mono' f-le*) *auto*
also have $\ldots = (\prod p{\in}Pr2.\ 1 - 1\ /\ ln\ n)$
using ‹$n > 2$› **and** ‹$ln\ n > 1$› **by** (*subst powr-realpow*) (*auto simp*: *f-def*)
also have $\ldots \leq (\prod p{\in}Pr2.\ 1 - 1\ /\ p)$
using ‹$n > 2$› **and** ‹$ln\ n > 1$›
by (*intro prod-mono conjI diff-mono divide-left-mono*) (*auto dest*: *prime-gt-1-nat*)
finally have *bound2*: $(\prod p{\in}Pr2.\ 1 - 1\ /\ p) \geq lb2\ n$ **unfolding** *lb2-def* **.**

have $(\prod p \mid prime\ p \wedge real\ p \leq ln\ n.\ inverse\ (1 - 1\ /\ p)) \geq (\prod p{\in}Pr1.\ inverse\ (1 - 1\ /\ p))$
using ‹$n > 2$› **by** (*intro prod-mono2*) (*auto intro*: *finite-primes-le dest*: *prime-gt-1-nat simp*: *in-prime-factors-iff field-simps*)
hence $inverse\ (\prod p \mid prime\ p \wedge real\ p \leq ln\ n.\ 1 - 1\ /\ p) \geq inverse\ (\prod p{\in}Pr1.\ 1 - 1\ /\ p)$
by (*subst* (*1 2*) *prod-inversef* [*symmetric*]) *auto*
hence *bound1*: $(\prod p{\in}Pr1.\ 1 - 1\ /\ p) \geq lb1\ n$ **unfolding** *lb1-def*
by (*rule inverse-le-imp-le*)
(*auto intro*!: *prod-pos simp*: *in-prime-factors-iff dest*: *prime-gt-1-nat*)

```
    have lb1 n * lb2 n ≤ (∏p∈Pr1. 1 - 1 / p) * (∏p∈Pr2. 1 - 1 / p)
      by (intro mult-mono bound1 bound2 prod-nonneg ballI)
         (auto simp: in-prime-factors-iff lb1-def lb2-def dest: prime-gt-1-nat)
    also have ... = (∏p∈Pr1 ∪ Pr2. 1 - 1 / p)
      by (subst prod.union-disjoint) auto
    also have Pr1 ∪ Pr2 = Pr by auto
    also have (∏p∈Pr. 1 - 1 / p) = totient n / n
      using n by (subst totient-formula2) auto
    finally show real (totient n) / real n ≥ lb1 n * lb2 n
      by (simp add: lb1-def lb2-def)
  qed

  moreover have eventually (λn. |lb1 n - C / ln (ln n)| ≤ 10 * C / ln (ln n) ^
2) at-top
    unfolding lb1-def C-def using mertens-third-theorem-strong
    by (rule eventually-compose-filterlim) real-asymp

  moreover have eventually (λn. (1 - ε) * C / ln (ln n) < lb1' n * lb2 n) at-top
     unfolding lb1'-def lb2-def C-def using third-mertens-const-pos ‹ε > 0› by
real-asymp

  ultimately show eventually (λn. totient n > (1 - ε) * C * n / ln (ln n)) at-top
    using eventually-gt-at-top[of 0]
  proof eventually-elim
    case (elim n)
    have (1 - ε) * C / ln (ln n) < lb1' n * lb2 n by fact
    also have lb1' n ≤ lb1 n
      unfolding lb1'-def using elim by linarith
    hence lb1' n * lb2 n ≤ lb1 n * lb2 n
      by (intro mult-right-mono) (auto simp: lb2-def)
    also have ... ≤ totient n / n by fact
    finally show totient n > (1 - ε) * C * n / (ln (ln n))
      using ‹n > 0› by (simp add: field-simps)
  qed
qed
```

Next, we examine the 'worst case' of $\varphi(n)$ where n is the primorial of x. In this case, we have $\varphi(n) < cn/\ln\ln n$ for any $c > C$ for all sufficiently large n.

```
theorem (in prime-number-theorem) totient-primorial-less:
  fixes ε :: real
  defines C ≡ third-mertens-const and h ≡ primorial
  assumes ε > 0
  shows   eventually (λx. totient (h x) < (1 + ε) * C * h x / ln (ln (h x))) at-top
proof -
  have C > 0 by (simp add: C-def third-mertens-const-pos)

  have (λx. (1 + ε) * C / ln (ln (h x))) ∼[at-top] (λx. (1 + ε) * C / ln (ϑ x))
```

```
    by (simp add: ln-primorial h-def)
  also have ... ∼[at-top] (λx. (1 + ε) * C / ln x)
    by (intro asymp-equiv-intros ln-ϑ-asymp-equiv)
  finally have (λx. (1 + ε) * C / ln (ln (h x)) − (1 + ε) * C / ln x) ∈ o(λx. (1
+ ε) * C / ln x)
    by (rule asymp-equiv-imp-diff-smallo)
  also have (λx. (1 + ε) * C / ln x) ∈ O(λx. 1 / ln x)
    by real-asymp
  also have (λx. (1 + ε) * C / ln (ln (h x)) − (1 + ε) * C / ln x) =
              (λx. (1 + ε) * C / ln (ln (h x)) − C / ln x − ε * C / ln x)
    by (simp add: algebra-simps fun-eq-iff add-divide-distrib)
  finally have (λx. (1 + ε) * C / ln (ln (h x)) − C / ln x − ε * C / ln x) ∈
o(λx. 1 / ln x) .
  hence (λx. (1 + ε) * C / ln (ln (h x)) − C / ln x − ε * C / ln x −
           ((∏p∈{p. prime p ∧ real p ≤ x}. 1 − 1 / real p) − C / ln x)) ∈ o(λx.
1 / ln x)
    unfolding C-def
   by (rule sum-in-smallo[OF - landau-o.big-small-trans[OF mertens-third-theorem]])
real-asymp+
  hence (λx. ((1 + ε) * C / ln (ln (h x)) − (∏p∈{p. prime p ∧ real p ≤ x}. 1 −
1 / real p)) −
               (ε * C / ln x)) ∈ o(λx. 1 / ln x)
    by (simp add: algebra-simps)
  also have (λx. 1 / ln x) ∈ O(λx. ε * C / ln x)
   using ‹ε > 0› by (simp add: landau-divide-simps C-def third-mertens-const-def)
  finally have (λx. (1 + ε) * C / ln (ln (h x)) − (∏p | prime p ∧ real p ≤ x. 1
− 1 / p))
                 ∼[at-top] (λx. ε * C / ln x)
    by (rule smallo-imp-asymp-equiv)

  hence eventually (λx. (1 + ε) * C / ln (ln (h x)) − (∏p | prime p ∧ real p ≤
x. 1 − 1 / p) > 0
                       ⟷ ε * C / ln x > 0) at-top
    by (rule asymp-equiv-eventually-pos-iff)
  moreover have eventually (λx. ε * C / ln x > 0) at-top
    using ‹ε > 0› ‹C > 0› by real-asymp
  ultimately have eventually (λx. (1 + ε) * C / ln (ln (h x)) >
                    (∏p | prime p ∧ real p ≤ x. 1 − 1 / p)) at-top
    by eventually-elim auto
  thus ?thesis
  proof eventually-elim
    case (elim x)
   have h x > 0 by (auto simp: h-def primorial-def intro!: prod-pos dest: prime-gt-0-nat)
    have h x * ((1 + ε) * C / ln (ln (h x))) > totient (h x)
      using elim primorial-pos[of x] unfolding h-def totient-primorial
      by (intro mult-strict-left-mono) auto
    thus ?case by (simp add: mult-ac)
  qed
qed
```

It follows that infinitely many values of n exceed $cn/\ln(\ln n)$ when c is chosen larger than C.

corollary (**in** *prime-number-theorem*) *totient-upper-bound*:
assumes $\varepsilon > 0$
defines $C \equiv$ *third-mertens-const*
shows *frequently* (λn. *totient* $n < (1 + \varepsilon) * C * n$ / *ln* (*ln* n)) *at-top*
proof −
define h **where** $h =$ *primorial*
have *eventually* (λn. *totient* $n < (1 + \varepsilon) * C * n$ / *ln* (*ln* n)) (*filtermap h at-top*)
using *totient-primorial-less*[*OF assms*(*1*)] **by** (*simp add*: *eventually-filtermap C-def h-def*)
hence *frequently* (λn. *totient* $n < (1 + \varepsilon) * C * n$ / *ln* (*ln* n)) (*filtermap h at-top*)
by (*intro eventually-frequently*) (*auto simp*: *filtermap-bot-iff*)
moreover from *this* **and** *primorial-at-top*
have *filtermap h at-top* $\leq$ *at-top* **by** (*simp add*: *filterlim-def h-def*)
ultimately show *?thesis*
by (*rule frequently-mono-filter*)
qed

Again, the following alternative formulation is somewhat nicer to prove:

lemma (**in** *prime-number-theorem*) *totient-primorial′-asymp-equiv*:
(λk. *totient* (*primorial′ k*)) ∼[*at-top*] (λk. *third-mertens-const* ∗ *primorial′ k* / *ln k*)
proof −
let *?C* = *third-mertens-const*
have (λk. *totient* (*primorial′ k*)) ∼[*at-top*] (λk. *primorial′ k* ∗ ($\prod i{<}k$. *1* − *1* / *nth-prime i*))
by (*subst totient-primorial′*) *auto*
also have ... ∼[*at-top*] (λk. *primorial′ k* ∗ (*?C* / *ln k*))
by (*intro asymp-equiv-intros mertens-third-theorem-asymp-equiv′*)
finally show *?thesis* **by** (*simp add*: *algebra-simps*)
qed

lemma (**in** *prime-number-theorem*) *totient-primorial′-asymp-equiv′*:
(λk. *totient* (*primorial′ k*)) ∼[*at-top*]
(λk. *third-mertens-const* ∗ *primorial′ k* / *ln* (*ln* (*primorial′ k*)))
proof −
let *?C* = *third-mertens-const*
have (λk. *totient* (*primorial′ k*)) ∼[*at-top*] (λk. *third-mertens-const* ∗ *primorial′ k* / *ln k*)
by (*rule totient-primorial′-asymp-equiv*)
also have ... ∼[*at-top*] (λk. *third-mertens-const* ∗ *primorial′ k* / *ln* (*ln* (*primorial′ k*)))
by (*intro asymp-equiv-intros asymp-equiv-symI*[*OF ln-ln-primorial′-asymp-equiv*])
finally show *?thesis* **.**
qed

All in all, $\varphi(n)$ has minimal order $cn/\ln\ln n$:

```
theorem (in prime-number-theorem)
  liminf-totient: liminf (λn. totient n * ln (ln n) / n) = third-mertens-const
    (is - = ereal ?c)
proof (intro antisym)
  have (λk. totient (primorial' k) / (?c * primorial' k / ln (ln (primorial' k))))
⟶ 1
    using totient-primorial'-asymp-equiv'
    by (intro asymp-equivD-strong eventually-mono[OF eventually-gt-at-top[of 1]])
auto
  hence (λk. totient (primorial' k) / (?c * primorial' k / ln (ln (primorial' k))) *
?c)
          ⟶ 1 * ?c by (intro tendsto-mult) auto
  hence (λk. totient (primorial' k) / (primorial' k / ln (ln (primorial' k)))) ⟶
?c
    using third-mertens-const-pos by simp
  hence liminf ((λn. totient n * ln (ln n) / n) ∘ primorial') = ereal ?c
    by (intro lim-imp-Liminf tendsto-ereal) simp-all
  hence ?c = liminf ((λn. ereal (totient n * ln (ln n) / n)) ∘ primorial')
    by (simp add: o-def)
  also have ... ≥ liminf (λn. totient n * ln (ln n) / n)
    using strict-mono-primorial' by (rule liminf-subseq-mono)
  finally show liminf (λn. totient n * ln (ln n) / n) ≤ ?c .
next
  show liminf (λn. totient n * ln (ln n) / n) ≥ ?c
    unfolding le-Liminf-iff
  proof safe
    fix C' assume C' < ereal ?c
   from ereal-dense2[OF this] obtain C where C: C < ?c ereal C > C' by auto
    define ε where ε = 1 - C / ?c
    from C have ε > 0 using third-mertens-const-pos by (simp add: ε-def)

    have eventually (λn::nat. ln (ln n) > 0) at-top by real-asymp
    hence eventually (λn. totient n * ln (ln n) / n > C) at-top
      using totient-lower-bound[OF ‹ε > 0›] eventually-gt-at-top[of 1]
    proof eventually-elim
      case (elim n)
      hence totient n * ln (ln n) / n > (1 - ε) * ?c
        by (simp add: field-simps)
      also have (1 - ε) * ?c = C
        using third-mertens-const-pos by (simp add: field-simps ε-def)
      finally show ?case .
    qed
    thus eventually (λn. ereal (totient n * ln (ln n) / n) > C') at-top
      by eventually-elim (rule less-trans[OF C(2)], auto)
  qed
qed

end
```

References

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

[2] A. Hildebrand. Introduction to Analytic Number Theory (lecture notes). hhttps://faculty.math.illinois.edu/~hildebr/ant/.