Bernoulli Numbers

Lukas Bulwahn and Manuel Eberl

May 26, 2024

Abstract

Bernoulli numbers were first discovered in the closed-form expansion of the sum $1^m + 2^m + \ldots + n^m$ for a fixed m and appear in many other places. This entry provides three different definitions for them: a recursive one, an explicit one, and one through their exponential generating function.

In addition, we prove some basic facts, e. g. their relation to sums of powers of integers and that all odd Bernoulli numbers except the first are zero. We also prove the correctness of the Akiyama–Tanigawa algorithm [2] for computing Bernoulli numbers with reasonable efficiency, and we define the periodic Bernoulli polynomials (which appear e. g. in the Euler–MacLaurin summation formula and the expansion of the log-Gamma function) and prove their basic properties.

Contents

1 Bernoulli numbers

theory *Bernoulli*
imports *Complex-Main*
begin

1.1 Preliminaries

lemma *power-numeral-reduce*: $a \mathbin{\hat{}} numeral\ n = a * a \mathbin{\hat{}} pred\text{-}numeral\ n$
⟨*proof*⟩

lemma *fact-diff-Suc*: $n < Suc\ m \Longrightarrow fact\ (Suc\ m - n) = of\text{-}nat\ (Suc\ m - n) * fact\ (m - n)$
⟨*proof*⟩

lemma *of-nat-binomial-Suc*:
assumes $k \le n$
shows $(of\text{-}nat\ (Suc\ n\ choose\ k) :: 'a :: field\text{-}char\text{-}0) = of\text{-}nat\ (Suc\ n)\ /\ of\text{-}nat\ (Suc\ n - k) * of\text{-}nat\ (n\ choose\ k)$
⟨*proof*⟩

lemma *integrals-eq*:
assumes $f\ 0 = g\ 0$
assumes $\bigwedge x.\ ((\lambda x.\ f\ x - g\ x)\ has\text{-}real\text{-}derivative\ 0)\ (at\ x)$
shows $f\ x = g\ x$
⟨*proof*⟩

lemma *sum-diff*: $((\sum i{\le}n{::}nat.\ f\ (i + 1) - f\ i){::}'a{::}field) = f\ (n + 1) - f\ 0$
⟨*proof*⟩

lemma *Rats-sum*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}) \Longrightarrow sum\ f\ A \in \mathbb{Q}$
⟨*proof*⟩

1.2 Bernoulli Numbers and Bernoulli Polynomials

declare *sum.cong* [*fundef-cong*]

fun *bernoulli* :: $nat \Rightarrow real$
where
$bernoulli\ 0 = (1{::}real)$
$|\ bernoulli\ (Suc\ n) = (-1\ /\ (n + 2)) * (\sum k \le n.\ ((n + 2\ choose\ k) * bernoulli\ k))$

declare *bernoulli.simps*[*simp del*]

lemmas *bernoulli-0* [*simp*] = *bernoulli.simps*(*1*)
lemmas *bernoulli-Suc* = *bernoulli.simps*(*2*)
lemma *bernoulli-1* [*simp*]: $bernoulli\ 1 = -1/2$ ⟨*proof*⟩
lemma *bernoulli-Suc-0* [*simp*]: $bernoulli\ (Suc\ 0) = -1/2$ ⟨*proof*⟩

The "normal" Bernoulli numbers are the negative Bernoulli numbers B_n^- we just defined (so called because $B_1^- = -\frac{1}{2}$). There is also another convention, the positive Bernoulli numbers B_n^+, which differ from the negative ones only in that $B_1^+ = \frac{1}{2}$. Both conventions have their justification, since a number of theorems are easier to state with one than the other.

definition *bernoulli′* **where**
bernoulli′ n = (if n = 1 then 1/2 else bernoulli n)

lemma *bernoulli′-0* [*simp*]: *bernoulli′ 0 = 1* ⟨*proof*⟩

lemma *bernoulli′-1* [*simp*]: *bernoulli′ (Suc 0) = 1/2*
⟨*proof*⟩

lemma *bernoulli-conv-bernoulli′*: *n ≠ 1* ⟹ *bernoulli n = bernoulli′ n*
⟨*proof*⟩

lemma *bernoulli′-conv-bernoulli*: *n ≠ 1* ⟹ *bernoulli′ n = bernoulli n*
⟨*proof*⟩

lemma *bernoulli-conv-bernoulli′-if*:
n ≠ 1 ⟹ *bernoulli n = (if n = 1 then −1/2 else bernoulli′ n)*
⟨*proof*⟩

lemma *bernoulli-in-Rats*: *bernoulli n ∈ ℚ*
⟨*proof*⟩

lemma *bernoulli′-in-Rats*: *bernoulli′ n ∈ ℚ*
⟨*proof*⟩

definition *bernpoly* :: *nat ⇒ ′a ⇒ ′a* :: *real-algebra-1* **where**
bernpoly n = (λx. ∑ k ≤ n. of-nat (n choose k) ∗ of-real (bernoulli k) ∗ x ^ (n − k))

lemma *bernpoly-altdef*:
bernpoly n = (λx. ∑ k≤n. of-nat (n choose k) ∗ of-real (bernoulli (n − k)) ∗ x ^ k)
⟨*proof*⟩

lemma *bernoulli-Suc′*:
bernoulli (Suc n) = −1/(real n + 2) ∗ (∑ k≤n. real (n + 2 choose (k + 2)) ∗ bernoulli (n − k))
⟨*proof*⟩

1.3 Basic Observations on Bernoulli Polynomials

lemma *bernpoly-0* [*simp*]: *bernpoly n 0 = (of-real (bernoulli n) :: ′a :: real-algebra-1)*
⟨*proof*⟩

lemma *continuous-on-bernpoly* [*continuous-intros*]:

continuous-on A (*bernpoly n* :: *'a* ⇒ *'a* :: *real-normed-algebra-1*)
⟨*proof*⟩

lemma *isCont-bernpoly* [*continuous-intros*]:
isCont (*bernpoly n* :: *'a* ⇒ *'a* :: *real-normed-algebra-1*) *x*
⟨*proof*⟩

lemma *has-field-derivative-bernpoly*:
(*bernpoly* (*Suc n*) *has-field-derivative*
(*of-nat* (*n* + *1*) ∗ *bernpoly n x* :: *'a* :: *real-normed-field*)) (*at x*)
⟨*proof*⟩

lemmas *has-field-derivative-bernpoly'* [*derivative-intros*] =
DERIV-chain'[*OF - has-field-derivative-bernpoly*]

lemma *sum-binomial-times-bernoulli*:
(∑ *k*≤*n*. ((*Suc n*) *choose k*) ∗ *bernoulli k*) = (*if n* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *sum-binomial-times-bernoulli'*:
(∑ *k*<*n*. *real* (*n choose k*) ∗ *bernoulli k*) = (*if n* = *1 then 1 else 0*)
⟨*proof*⟩

lemma *binomial-unroll*:
n > *0* ⟹ (*n choose k*) = (*if k* = *0 then 1 else* (*n* − *1*) *choose* (*k* − *1*) + ((*n* − *1*) *choose k*))
⟨*proof*⟩

lemma *sum-unroll*:
(∑ *k*≤*n*::*nat*. *f k*) = (*if n* = *0 then f 0 else f n* + (∑ *k*≤*n* − *1*. *f k*))
⟨*proof*⟩

lemma *bernoulli-unroll*:
n > *0* ⟹ *bernoulli n* = − *1* / (*real n* + *1*) ∗ (∑ *k*≤*n* − *1*. *real* (*n* + *1 choose k*) ∗ *bernoulli k*)
⟨*proof*⟩

lemmas *bernoulli-unroll-all* = *binomial-unroll bernoulli-unroll sum-unroll bernpoly-def*

lemma *bernpoly-1-1*: *bernpoly 1 1* = *of-real* (*1*/*2*)
⟨*proof*⟩

1.4 Sum of Powers with Bernoulli Polynomials

lemma *diff-bernpoly*:
fixes *x* :: *real*
shows *bernpoly n* (*x* + *1*) − *bernpoly n x* = *of-nat n* ∗ *x* ^ (*n* − *1*)
⟨*proof*⟩

lemma *bernpoly-of-real*: *bernpoly n* (*of-real x*) = *of-real* (*bernpoly n x*)
⟨*proof*⟩

lemma *bernpoly-1*:
assumes $n \neq 1$
shows *bernpoly n 1* = *of-real* (*bernoulli n*)
⟨*proof*⟩

lemma *bernpoly-1'*: *bernpoly n 1* = *of-real* (*bernoulli' n*)
⟨*proof*⟩

theorem *sum-of-powers*:
($\sum k \leq n$::*nat*. (*real k*) ^ *m*) = (*bernpoly* (*Suc m*) (*n* + *1*) − *bernpoly* (*Suc m*) *0*) / (*m* + *1*)
⟨*proof*⟩

lemma *sum-of-powers-nat-aux*:
assumes *real a* = *b* / *c* *real b'* = *b* *real c'* = *c*
shows *a* = *b' div c'*
⟨*proof*⟩

1.5 Instances for Square And Cubic Numbers

theorem *sum-of-squares*: *real* ($\sum k \leq n$::*nat*. *k* ^ *2*) = *real* (*2* ∗ *n* ^ *3* + *3* ∗ *n* ^ *2* + *n*) / *6*
⟨*proof*⟩

corollary *sum-of-squares-nat*: ($\sum k \leq n$::*nat*. *k* ^ *2*) = (*2* ∗ *n* ^ *3* + *3* ∗ *n* ^ *2* + *n*) *div 6*
⟨*proof*⟩

theorem *sum-of-cubes*: *real* ($\sum k \leq n$::*nat*. *k* ^ *3*) = *real* (*n* ^ *2* + *n*) ^ *2* / *4*
⟨*proof*⟩

corollary *sum-of-cubes-nat*: ($\sum k \leq n$::*nat*. *k* ^ *3*) = (*n* ^ *2* + *n*) ^ *2 div 4*
⟨*proof*⟩

end

2 Periodic Bernoulli polynomials

theory *Periodic-Bernpoly*
imports
Bernoulli
HOL−Library.Periodic-Fun
begin

Given the n-th Bernoulli polynomial $B_n(x)$, one can define the periodic function $P_n(x) = B_n(x - \lfloor x \rfloor)$, which shares many of the interesting properties of the Bernoulli polynomials. In particular, all $P_n(x)$ with $n \neq 1$ are continuous and if $n \geq 3$, they are continuously differentiable with $P_n'(x) = nP_{n-1}(x)$ just like the Bernoully polynomials themselves.

These functions occur e. g. in the Euler–MacLaurin summation formula and Stirling's approximation for the logarithmic Gamma function.

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

lemma *frac-eq-id*: *x* ∈ {*0*..<*1*} ⟹ *frac x* = *x*
⟨*proof*⟩

lemma *periodic-continuous-onI*:
fixes *f* :: *real* ⇒ *real*
assumes *periodic*: ⋀*x*. *f* (*x* + *p*) = *f x p* > *0*
assumes *cont*: *continuous-on* {*a*..*a*+*p*} *f*
shows *continuous-on UNIV f*
⟨*proof*⟩

lemma *has-field-derivative-at-within-union*:
assumes (*f has-field-derivative D*) (*at x within A*)
(*f has-field-derivative D*) (*at x within B*)
shows (*f has-field-derivative D*) (*at x within* (*A* ∪ *B*))
⟨*proof*⟩

lemma *has-field-derivative-cong-ev′*:
assumes *x* = *y*
and ∗: *eventually* (λ*x*. *x* ∈ *s* ⟶ *f x* = *g x*) (*nhds x*)
and *u* = *v s* = *t f x* = *g y*
shows (*f has-field-derivative u*) (*at x within s*) = (*g has-field-derivative v*) (*at y within t*)
⟨*proof*⟩

interpretation *frac*: *periodic-fun-simple′ frac*
⟨*proof*⟩

lemma *tendsto-frac-at-right-0*:
(*frac* ⟶ *0*) (*at-right* (*0* :: ′*a* :: {*floor-ceiling*,*order-topology*}))
⟨*proof*⟩

lemma *tendsto-frac-at-left-1*:
(*frac* ⟶ *1*) (*at-left* (*1* :: ′*a* :: {*floor-ceiling*,*order-topology*}))
⟨*proof*⟩

lemma *continuous-on-frac* [*THEN continuous-on-subset*, *continuous-intros*]:
continuous-on {*0*::′*a*::{*floor-ceiling*,*order-topology*}..<*1*} *frac*
⟨*proof*⟩

lemma *isCont-frac* [*continuous-intros*]:
  assumes (x :: $'a$ :: {*floor-ceiling*,*order-topology*,*t2-space*}) $\in$ {*0*<..<*1*}
  shows *isCont frac x*
⟨*proof*⟩

lemma *has-field-derivative-frac*:
  assumes (x::*real*) $\notin \mathbb{Z}$
  shows (*frac has-field-derivative 1*) (*at x*)
⟨*proof*⟩

lemmas *has-field-derivative-frac′* [*derivative-intros*] =
  DERIV-chain′[*OF - has-field-derivative-frac*]

lemma *continuous-on-compose-fracI*:
  fixes f :: *real* $\Rightarrow$ *real*
  assumes *cont1*: *continuous-on* {*0*..*1*} *f*
  assumes *cont2*: f *0* = f *1*
  shows *continuous-on UNIV* (λx. f (*frac x*))
⟨*proof*⟩

definition *pbernpoly* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
  pbernpoly n x = *bernpoly n* (*frac x*)

lemma *pbernpoly-0* [*simp*]: *pbernpoly n 0* = *bernoulli n*
  ⟨*proof*⟩

lemma *pbernpoly-eq-bernpoly*: $x \in$ {*0*..<*1*} $\Longrightarrow$ *pbernpoly n x* = *bernpoly n x*
  ⟨*proof*⟩

interpretation *pbernpoly*: *periodic-fun-simple′ pbernpoly n*
  ⟨*proof*⟩

lemma *continuous-on-pbernpoly* [*continuous-intros*]:
  assumes $n \neq$ *1*
  shows *continuous-on A* (*pbernpoly n*)
⟨*proof*⟩

lemma *continuous-on-pbernpoly′* [*continuous-intros*]:
  assumes $n \neq$ *1* *continuous-on A f*
  shows *continuous-on A* (λx. *pbernpoly n* (f x))
  ⟨*proof*⟩

lemma *isCont-pbernpoly* [*continuous-intros*]: $n \neq$ *1* $\Longrightarrow$ *isCont* (*pbernpoly n*) *x*
  ⟨*proof*⟩

lemma *has-field-derivative-pbernpoly-Suc*:
  assumes $n \geq 2 \lor x \notin \mathbb{Z}$

shows $(pbernpoly\ (Suc\ n)\ has\text{-}field\text{-}derivative\ real\ (Suc\ n) * pbernpoly\ n\ x)\ (at\ x)$
⟨*proof*⟩

lemmas *has-field-derivative-pbernpoly-Suc′* =
DERIV-chain′[*OF - has-field-derivative-pbernpoly-Suc*]

lemma *bounded-pbernpoly*: **obtains** *c* **where** $\bigwedge x.\ norm\ (pbernpoly\ n\ x) \leq c$
⟨*proof*⟩

end

3 Connection of Bernoulli numbers to formal power series

theory *Bernoulli-FPS*
imports
Bernoulli
HOL−Computational-Algebra.Computational-Algebra
HOL−Combinatorics.Stirling
HOL−Number-Theory.Number-Theory
begin

3.1 Preliminaries

context *factorial-semiring*
begin

lemma *multiplicity-prime-prime*:
$prime\ p \Longrightarrow prime\ q \Longrightarrow multiplicity\ p\ q = (if\ p = q\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *prime-prod-dvdI*:
fixes $f :: 'b \Rightarrow 'a$
assumes *finite A*
assumes $\bigwedge x.\ x \in A \Longrightarrow prime\ (f\ x)$
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x\ dvd\ y$
assumes *inj-on f A*
shows *prod f A dvd y*
⟨*proof*⟩

end

context *semiring-gcd*
begin

lemma *gcd-add-dvd-right1*: $a\ dvd\ b \Longrightarrow gcd\ a\ (b + c) = gcd\ a\ c$
⟨*proof*⟩

lemma *gcd-add-dvd-right2*: $a\ dvd\ c \Longrightarrow gcd\ a\ (b + c) = gcd\ a\ b$
⟨*proof*⟩

lemma *gcd-add-dvd-left1*: $a\ dvd\ b \Longrightarrow gcd\ (b + c)\ a = gcd\ c\ a$
⟨*proof*⟩

lemma *gcd-add-dvd-left2*: $a\ dvd\ c \Longrightarrow gcd\ (b + c)\ a = gcd\ b\ a$
⟨*proof*⟩

end

context *ring-gcd*
begin

lemma *gcd-diff-dvd-right1*: $a\ dvd\ b \Longrightarrow gcd\ a\ (b - c) = gcd\ a\ c$
⟨*proof*⟩

lemma *gcd-diff-dvd-right2*: $a\ dvd\ c \Longrightarrow gcd\ a\ (b - c) = gcd\ a\ b$
⟨*proof*⟩

lemma *gcd-diff-dvd-left1*: $a\ dvd\ b \Longrightarrow gcd\ (b - c)\ a = gcd\ c\ a$
⟨*proof*⟩

lemma *gcd-diff-dvd-left2*: $a\ dvd\ c \Longrightarrow gcd\ (b - c)\ a = gcd\ b\ a$
⟨*proof*⟩

end

lemma *cong-int*: $[a = b]\ (mod\ m) \Longrightarrow [int\ a = int\ b]\ (mod\ m)$
⟨*proof*⟩

lemma *Rats-int-div-natE*:
assumes $(x :: 'a :: field\text{-}char\text{-}0) \in \mathbb{Q}$
obtains $m :: int$ **and** $n :: nat$ **where** $n > 0$ **and** $x = of\text{-}int\ m\ /\ of\text{-}nat\ n$ **and** *coprime m n*
⟨*proof*⟩

lemma *sum-in-Ints*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Z}) \Longrightarrow sum\ f\ A \in \mathbb{Z}$
⟨*proof*⟩

lemma *Ints-real-of-nat-divide*: $b\ dvd\ a \Longrightarrow real\ a\ /\ real\ b \in \mathbb{Z}$
⟨*proof*⟩

lemma *product-dvd-fact*:
assumes $a > 1$ $b > 1$ $a = b \longrightarrow a > 2$

shows $(a * b)$ *dvd fact* $(a * b - 1)$
⟨*proof*⟩

lemma *composite-imp-factors-nat*:
assumes $m > 1$ $\neg$*prime* $(m{::}nat)$
shows $\exists n\ k.\ m = n * k \wedge 1 < n \wedge n < m \wedge 1 < k \wedge k < m$
⟨*proof*⟩

This lemma describes what the numerator and denominator of a finite subseries of the harmonic series are when it is written as a single fraction.

lemma *sum-inverses-conv-fraction*:
fixes $f :: 'a \Rightarrow 'b :: field$
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \neq 0$ *finite A*
shows $(\sum x{\in}A.\ 1\ /\ f\ x) = (\sum x{\in}A.\ \prod y{\in}A-\{x\}.\ f\ y)\ /\ (\prod x{\in}A.\ f\ x)$
⟨*proof*⟩

If all terms in the subseries are primes, this fraction is automatically on lowest terms.

lemma *sum-prime-inverses-fraction-coprime*:
fixes $f :: 'a \Rightarrow nat$
assumes *finite A* **and** *primes*: $\bigwedge x.\ x \in A \Longrightarrow prime\ (f\ x)$ **and** *inj*: *inj-on f A*
defines $a \equiv (\sum x{\in}A.\ \prod y{\in}A-\{x\}.\ f\ y)$
shows *coprime a* $(\prod x{\in}A.\ f\ x)$
⟨*proof*⟩

In the following, we will prove the correctness of the Akiyama–Tanigawa algorithm [2], which is a simple algorithm for computing Bernoulli numbers that was discovered by Akiyama and Tanigawa [1] essentially as a by-product of their studies of the Euler–Zagier multiple zeta function. The algorithm is based on a number triangle (similar to Pascal's triangle) in which the Bernoulli numbers are the leftmost diagonal.

While the algorithm itself is quite simple, proving it correct is not entirely trivial. We will use generating functions and Stirling numbers, mostly following the presentation by Kaneko [2].

The following operator is a variant of the *fps-XD* operator where the multiplication is not with *fps-X*, but with an arbitrary formal power series. It is not quite clear if this operator has a less ad-hoc meaning than the fashion in which we use it; it is, however, very useful for proving the relationship between Stirling numbers and Bernoulli numbers.

context
includes *fps-notation*
begin

definition *fps-XD'* **where** *fps-XD'* $a = (\lambda b.\ a *$ *fps-deriv* $b)$

lemma *fps-XD'-0* [*simp*]: *fps-XD'* $a\ 0 = 0$ ⟨*proof*⟩

lemma *fps-XD′-1* [*simp*]: *fps-XD′ a 1* = *0* ⟨*proof*⟩
lemma *fps-XD′-fps-const* [*simp*]: *fps-XD′ a* (*fps-const b*) = *0* ⟨*proof*⟩
lemma *fps-XD′-fps-of-nat* [*simp*]: *fps-XD′ a* (*of-nat b*) = *0* ⟨*proof*⟩
lemma *fps-XD′-fps-of-int* [*simp*]: *fps-XD′ a* (*of-int b*) = *0* ⟨*proof*⟩
lemma *fps-XD′-fps-numeral* [*simp*]: *fps-XD′ a* (*numeral b*) = *0* ⟨*proof*⟩

lemma *fps-XD′-add* [*simp*]: *fps-XD′ a* (*b* + *c* :: ′*a* :: *comm-ring-1 fps*) = *fps-XD′ a b* + *fps-XD′ a c*
⟨*proof*⟩

lemma *fps-XD′-minus* [*simp*]: *fps-XD′ a* (*b* − *c* :: ′*a* :: *comm-ring-1 fps*) = *fps-XD′ a b* − *fps-XD′ a c*
⟨*proof*⟩

lemma *fps-XD′-prod*: *fps-XD′ a* (*b* ∗ *c* :: ′*a* :: *comm-ring-1 fps*) = *fps-XD′ a b* ∗ *c* + *b* ∗ *fps-XD′ a c*
⟨*proof*⟩

lemma *fps-XD′-power*: *fps-XD′ a* (*b* ^ *n* :: ′*a* :: *idom fps*) = *of-nat n* ∗ *b* ^ (*n* − *1*) ∗ *fps-XD′ a b*
⟨*proof*⟩

lemma *fps-XD′-power-Suc*: *fps-XD′ a* (*b* ^ *Suc n* :: ′*a* :: *idom fps*) = *of-nat* (*Suc n*) ∗ *b* ^ *n* ∗ *fps-XD′ a b*
⟨*proof*⟩

lemma *fps-XD′-sum*: *fps-XD′ a* (*sum f A*) = *sum* (λ*x*. *fps-XD′* (*a* :: ′*a* :: *comm-ring-1 fps*) (*f x*)) *A*
⟨*proof*⟩

lemma *fps-XD′-funpow-affine*:
fixes *G H* :: *real fps*
assumes [*simp*]: *fps-deriv G* = *1*
defines *S* ≡ λ*n i*. *fps-const* (*real* (*Stirling n i*))
shows (*fps-XD′ G* ^^ *n*) *H* =
(∑ *m*≤*n*. *S n m* ∗ *G* ^ *m* ∗ (*fps-deriv* ^^ *m*) *H*)
⟨*proof*⟩

3.2 Generating function of Stirling numbers

lemma *Stirling-n-0*: *Stirling n 0* = (*if n* = *0 then 1 else 0*)
⟨*proof*⟩

The generating function of Stirling numbers w. r. t. their first argument:

$$\sum_{n=0}^{\infty} \left\{ {n \atop m} \right\} \frac{x^n}{n!} = \frac{(e^x - 1)^m}{m!}$$

definition *Stirling-fps* :: *nat* ⇒ *real fps* **where**

Stirling-fps m = fps-const (1 / fact m) ∗ (fps-exp 1 − 1) ^ m

theorem *sum-Stirling-binomial*:
Stirling (Suc n) (Suc m) = (∑ i = 0..n. Stirling i m ∗ (n choose i))
⟨*proof*⟩

lemma *Stirling-fps-aux*: (*fps-exp 1 − 1*) ^ *m $ n ∗ fact n = fact m ∗ real* (*Stirling n m*)
⟨*proof*⟩

lemma *Stirling-fps-nth*: *Stirling-fps m $ n = Stirling n m / fact n*
⟨*proof*⟩

theorem *Stirling-fps-altdef*: *Stirling-fps m = Abs-fps* (*λn. Stirling n m / fact n*)
⟨*proof*⟩

theorem *Stirling-closed-form*:
real (Stirling n k) = (∑ j≤k. (−1)^(k − j) ∗ real (k choose j) ∗ real j ^ n) / fact k
⟨*proof*⟩

3.3 Generating function of Bernoulli numbers

We will show that the negative and positive Bernoulli numbers are the coefficients of the exponential generating function $\frac{x}{e^x-1}$ (resp. $\frac{x}{1-e^{-x}}$), i. e.

$$\sum_{n=0}^{\infty} B_n^- \frac{x^n}{n!} = \frac{x}{e^x - 1}$$

$$\sum_{n=0}^{\infty} B_n^+ \frac{x^n}{n!} = \frac{x}{1 - e^{-1}}$$

definition *bernoulli-fps* :: *'a* :: *real-normed-field fps*
where *bernoulli-fps = fps-X / (fps-exp 1 − 1)*
definition *bernoulli'-fps* :: *'a* :: *real-normed-field fps*
where *bernoulli'-fps = fps-X / (1 − (fps-exp (−1)))*

lemma *bernoulli-fps-altdef*: *bernoulli-fps = Abs-fps* (*λn. of-real* (*bernoulli n*) / *fact n* :: *'a*)
and *bernoulli-fps-aux*: *bernoulli-fps ∗ (fps-exp 1 − 1 :: 'a :: real-normed-field fps) = fps-X*
⟨*proof*⟩

theorem *fps-nth-bernoulli-fps* [*simp*]:
fps-nth bernoulli-fps n = of-real (bernoulli n) / fact n
⟨*proof*⟩

lemma *bernoulli'-fps-aux*:

(*fps-exp 1* − *1*) ∗ *Abs-fps* (λ*n*. *of-real* (*bernoulli′ n*) / *fact n* :: ′*a*) = *fps-exp 1* ∗ *fps-X*
and *bernoulli′-fps-aux′*:
(*1* − *fps-exp* (−*1*)) ∗ *Abs-fps* (λ*n*. *of-real* (*bernoulli′ n*) / *fact n* :: ′*a*) = *fps-X*
and *bernoulli′-fps-altdef*:
bernoulli′-fps = *Abs-fps* (λ*n*. *of-real* (*bernoulli′ n*) / *fact n* :: ′*a* :: *real-normed-field*)
⟨*proof*⟩

theorem *fps-nth-bernoulli′-fps* [*simp*]:
fps-nth bernoulli′-fps n = *of-real* (*bernoulli′ n*) / *fact n*
⟨*proof*⟩

lemma *bernoulli-fps-conv-bernoulli′-fps*: *bernoulli-fps* = *bernoulli′-fps* − *fps-X*
⟨*proof*⟩

lemma *bernoulli′-fps-conv-bernoulli-fps*: *bernoulli′-fps* = *bernoulli-fps* + *fps-X*
⟨*proof*⟩

theorem *bernoulli-odd-eq-0*:
assumes *n* ≠ *1* **and** *odd n*
shows *bernoulli n* = *0*
⟨*proof*⟩

lemma *bernoulli′-odd-eq-0*: *n* ≠ *1* ⟹ *odd n* ⟹ *bernoulli′ n* = *0*
⟨*proof*⟩

The following simplification rule takes care of rewriting *bernoulli n* to 0 for any odd numeric constant greater than 1:

lemma *bernoulli-odd-numeral-eq-0* [*simp*]: *bernoulli* (*numeral* (*Num.Bit1 n*)) = *0*
⟨*proof*⟩

lemma *bernoulli′-odd-numeral-eq-0* [*simp*]: *bernoulli′* (*numeral* (*Num.Bit1 n*)) = *0*
⟨*proof*⟩

The following explicit formula for Bernoulli numbers can also derived reasonably easily using the generating functions of Stirling numbers and Bernoulli numbers. The proof follows an answer by Marko Riedel on the Mathematics StackExchange [3].

theorem *bernoulli-altdef*:
bernoulli n = (∑*m*≤*n*. ∑*k*≤*m*. (−*1*)^*k* ∗ *real* (*m choose k*) ∗ *real k*^*n* / *real* (*Suc m*))
⟨*proof*⟩

corollary *bernoulli-conv-Stirling*:
bernoulli n = (∑*k*≤*n*. (−*1*) ^ *k* ∗ *fact k* / *real* (*k* + *1*) ∗ *Stirling n k*)
proof −
have (∑*k*≤*n*. (−*1*) ^ *k* ∗ *fact k* / (*k* + *1*) ∗ *Stirling n k*) =

$(\sum k \le n. \sum i \le k. (-1) \verb|^| i * (k\ choose\ i) * i \verb|^| n / real\ (k + 1))$
proof (*intro sum.cong*, *goal-cases*)
case (*2 k*)
have $(-1) \verb|^| k * fact\ k / (k + 1) * Stirling\ n\ k =$
$(\sum j \le k. (-1) \verb|^| k * (-1) \verb|^|(k - j) * (k\ choose\ j) * j \verb|^| n / (k + 1))$
by (*simp add*: *Stirling-closed-form sum-distrib-left sum-divide-distrib mult-ac*)
also have ... $= (\sum j \le k. (-1) \verb|^| j * (k\ choose\ j) * j \verb|^| n / (k + 1))$
by (*intro sum.cong*) (*auto simp*: *uminus-power-if split*: *if-splits*)
finally show *?case* **.**
qed *auto*
also have ... = *bernoulli n*
by (*simp add*: *bernoulli-altdef*)
finally show *?thesis* **..**
qed

3.4 Von Staudt–Clausen Theorem

lemma *vonStaudt-Clausen-lemma*:
assumes $n > 0$ **and** *prime p*
shows $[(\sum m<p. (-1) \verb|^| m * ((p - 1)\ choose\ m) * m \verb|^|(2*n)) =$
$(if\ (p - 1)\ dvd\ (2 * n)\ then\ -1\ else\ 0)]\ (mod\ p)$
⟨*proof*⟩

The Von Staudt–Clausen theorem states that for $n > 0$,

$$B_{2n} + \sum_{p-1|2n} \frac{1}{p}$$

is an integer.

theorem *vonStaudt-Clausen*:
assumes $n > 0$
shows $bernoulli\ (2 * n) + (\sum p \mid prime\ p \wedge (p - 1)\ dvd\ (2 * n).\ 1 / real\ p)$
$\in \mathbb{Z}$
(**is** - + *?P* $\in \mathbb{Z}$)
⟨*proof*⟩

3.5 Denominators of Bernoulli numbers

A consequence of the Von Staudt–Clausen theorem is that the denominator of B_{2n} for $n > 0$ is precisely the product of all prime numbers p such that $p - 1$ divides $2n$. Since the denominator is obvious in all other cases, this fully characterises the denominator of Bernoulli numbers.

definition *bernoulli-denom* :: *nat* ⇒ *nat* **where**
bernoulli-denom n =
$(if\ n = 1\ then\ 2\ else\ if\ n = 0 \vee odd\ n\ then\ 1\ else \prod\{p.\ prime\ p \wedge (p - 1)$
$dvd\ n\})$

definition *bernoulli-num* :: *nat* ⇒ *int* **where**

bernoulli-num n = ⌊*bernoulli n* ∗ *bernoulli-denom n*⌋

lemma *finite-bernoulli-denom-set*: *n* > (*0* :: *nat*) ⟹ *finite* {*p*. *prime p* ∧ (*p* − *1*) *dvd n*}
⟨*proof*⟩

lemma *bernoulli-denom-0* [*simp*]: *bernoulli-denom 0* = *1*
and *bernoulli-denom-1* [*simp*]: *bernoulli-denom 1* = *2*
and *bernoulli-denom-Suc-0* [*simp*]: *bernoulli-denom* (*Suc 0*) = *2*
and *bernoulli-denom-odd* [*simp*]: *n* ≠ *1* ⟹ *odd n* ⟹ *bernoulli-denom n* = *1*
and *bernoulli-denom-even*:
n > *0* ⟹ *even n* ⟹ *bernoulli-denom n* = ∏{*p*. *prime p* ∧ (*p* − *1*) *dvd n*}
⟨*proof*⟩

lemma *bernoulli-denom-pos*: *bernoulli-denom n* > *0*
⟨*proof*⟩

lemma *bernoulli-denom-nonzero* [*simp*]: *bernoulli-denom n* ≠ *0*
⟨*proof*⟩

lemma *bernoulli-denom-code* [*code*]:
bernoulli-denom n =
(*if n* = *1 then 2 else if n* = *0* ∨ *odd n then 1*
else prod-list (*filter* (λ*p*. (*p* − *1*) *dvd n*) (*primes-upto* (*n* + *1*)))) (**is** - = *?rhs*)
⟨*proof*⟩

corollary *bernoulli-denom-correct*:
obtains *a* :: *int*
where *coprime a* (*bernoulli-denom m*)
bernoulli m = *of-int a* / *of-nat* (*bernoulli-denom m*)
proof −
consider *m* = *0* | *m* = *1* | *odd m m* ≠ *1* | *even m m* > *0*
by *auto*
thus *?thesis*
proof *cases*
assume *m* = *0*
thus *?thesis* **by** (*intro that*[*of 1*]) (*auto simp*: *bernoulli-denom-def*)
next
assume *m* = *1*
thus *?thesis* **by** (*intro that*[*of* −*1*]) (*auto simp*: *bernoulli-denom-def*)
next
assume *odd m m* ≠ *1*
thus *?thesis* **by** (*intro that*[*of 0*]) (*auto simp*: *bernoulli-denom-def bernoulli-odd-eq-0*)
next
assume *even m m* > *0*
define *n* **where** *n* = *m div 2*
have [*simp*]: *m* = *2* ∗ *n* **and** *n*: *n* > *0*
using ‹*even m*› ‹*m* > *0*› **by** (*auto simp*: *n-def intro*!: *Nat.gr0I*)

obtain *a b* **where** *ab*: *bernoulli* (*2* ∗ *n*) = *a* / *b coprime a* (*int b*) *b* > *0*
  using *Rats-int-div-natE*[*OF bernoulli-in-Rats*] **by** *metis*
define *P* **where** *P* = {*p*. *prime p* ∧ (*p* − *1*) *dvd* (*2* ∗ *n*)}
have *finite P* **unfolding** *P-def*
  using *n* **by** (*intro finite-bernoulli-denom-set*) *auto*
from *vonStaudt-Clausen*[*of n*] **obtain** *k* **where** *k*: *bernoulli* (*2* ∗ *n*) + (∑ *p*∈*P*. *1*/*p*) = *of-int k*
  using ‹*n* > *0*› **by** (*auto simp*: *P-def Ints-def*)

define *c* **where** *c* = (∑ *p*∈*P*. ∏ (*P*−{*p*}))
from ‹*finite P*› **have** (∑ *p*∈*P*. *1* / *p*) = *c* / ∏ *P*
  by (*subst sum-inverses-conv-fraction*) (*auto simp*: *P-def prime-gt-0-nat c-def*)
moreover have *P-nz*: *prod real P* > *0*
  using *prime-gt-0-nat* **by** (*auto simp*: *P-def intro*!: *prod-pos*)
ultimately have *eq*: *bernoulli* (*2* ∗ *n*) = (*k* ∗ ∏ *P* − *c*) / ∏ *P*
  using *ab P-nz* **by** (*simp add*: *field-simps k* [*symmetric*])

have *gcd* (*k* ∗ ∏ *P* − *int c*) (∏ *P*) = *gcd* (*int c*) (∏ *P*)
  by (*simp add*: *gcd-diff-dvd-left1*)
also have ... = *int* (*gcd c* (∏ *P*))
  by (*simp flip*: *gcd-int-int-eq*)
also have *coprime c* (∏ *P*)
  unfolding *c-def* **using** ‹*finite P*›
  by (*intro sum-prime-inverses-fraction-coprime*) (*auto simp*: *P-def*)
hence *gcd c* (∏ *P*) = *1*
  by *simp*
finally have *coprime*: *coprime* (*k* ∗ ∏ *P* − *int c*) (∏ *P*)
  by (*simp only*: *coprime-iff-gcd-eq-1*)

have *eq'*: ∏ *P* = *bernoulli-denom* (*2* ∗ *n*)
  using *n* **by** (*simp add*: *bernoulli-denom-def P-def*)
show *?thesis*
  by (*rule that*[*of k* ∗ ∏ *P* − *int c*]) (*use eq eq' coprime* **in** *simp-all*)
qed
qed

lemma *bernoulli-conv-num-denom*: *bernoulli n* = *bernoulli-num n* / *bernoulli-denom n* (**is** *?th1*)
and *coprime-bernoulli-num-denom*: *coprime* (*bernoulli-num n*) (*bernoulli-denom n*) (**is** *?th2*)
⟨*proof*⟩

Two obvious consequences from this are that the denominators of all odd Bernoulli numbers except for the first one are squarefree and multiples of 6:

lemma *six-divides-bernoulli-denom*:
assumes *even n n* > *0*
shows *6 dvd bernoulli-denom n*
⟨*proof*⟩

lemma *squarefree-bernoulli-denom*: *squarefree* (*bernoulli-denom n*)
⟨*proof*⟩

Furthermore, the denominator of B_n divides $2(2^n - 1)$. This also gives us an upper bound on the denominators.

lemma *bernoulli-denom-dvd*: *bernoulli-denom n dvd* (*2* ∗ (*2* ^ *n* − *1*))
⟨*proof*⟩

corollary *bernoulli-bound*:
assumes *n* > *0*
shows *bernoulli-denom n* ≤ *2* ∗ (*2* ^ *n* − *1*)
⟨*proof*⟩

It can also be shown fairly easily from the von Staudt–Clausen theorem that if p is prime and $2p + 1$ is not, then $B_{2p} \equiv \frac{1}{6} \pmod 1$ or, equivalently, the denominator of B_{2p} is 6 and the numerator is of the form $6k + 1$.

This is the case e. g. for any primes of the form $3k + 1$ or $5k + 2$.

lemma *bernoulli-denom-prime-nonprime*:
assumes *prime p* ¬*prime* (*2* ∗ *p* + *1*)
shows *bernoulli* (*2* ∗ *p*) − *1* / *6* ∈ ℤ
[*bernoulli-num* (*2* ∗ *p*) = *1*] (*mod 6*)
bernoulli-denom (*2* ∗ *p*) = *6*
⟨*proof*⟩

3.6 Akiyama–Tanigawa algorithm

First, we define the Akiyama–Tanigawa number triangle as shown by Kaneko [2]. We define this generically, parametrised by the first row. This makes the proofs a little bit more modular.

fun *gen-akiyama-tanigawa* :: (*nat* ⇒ *real*) ⇒ *nat* ⇒ *nat* ⇒ *real* **where**
gen-akiyama-tanigawa f 0 m = *f m*
| *gen-akiyama-tanigawa f* (*Suc n*) *m* =
real (*Suc m*) ∗ (*gen-akiyama-tanigawa f n m* − *gen-akiyama-tanigawa f n* (*Suc m*))

lemma *gen-akiyama-tanigawa-0* [*simp*]: *gen-akiyama-tanigawa f 0* = *f*
⟨*proof*⟩

The "regular" Akiyama–Tanigawa triangle is the one that is used for reading off Bernoulli numbers:

definition *akiyama-tanigawa* **where**
akiyama-tanigawa = *gen-akiyama-tanigawa* (λ*n*. *1* / *real* (*Suc n*))

context
begin

private definition *AT-fps* :: (*nat* ⇒ *real*) ⇒ *nat* ⇒ *real fps* **where**
AT-fps f n = (*fps-X* − *1*) ∗ *Abs-fps* (*gen-akiyama-tanigawa f n*)

private lemma *AT-fps-Suc*: *AT-fps f* (*Suc n*) = (*fps-X* − *1*) ∗ *fps-deriv* (*AT-fps f n*)
⟨*proof*⟩ **lemma** *AT-fps-altdef*:
AT-fps f n =
(∑ *m*≤*n*. *fps-const* (*real* (*Stirling n m*)) ∗ (*fps-X* − *1*)^*m* ∗ (*fps-deriv* ^^ *m*) (*AT-fps f 0*))
⟨*proof*⟩ **lemma** *AT-fps-0-nth*: *AT-fps f 0* $ *n* = (*if n* = *0 then* −*f 0 else f* (*n* − *1*) − *f n*)
⟨*proof*⟩

The following fact corresponds to Proposition 1 in Kaneko's proof:

lemma *gen-akiyama-tanigawa-n-0*:
gen-akiyama-tanigawa f n 0 =
(∑ *k*≤*n*. (− *1*) ^ *k* ∗ *fact k* ∗ *real* (*Stirling* (*Suc n*) (*Suc k*)) ∗ *f k*)
⟨*proof*⟩

The following lemma states that for $A(x) := \sum_{k=0}^{\infty} a_{0,k} x^k$, we have

$$\sum_{n=0}^{\infty} a_{n,0} \frac{x^n}{n!} = e^x A(1 - e^x)$$

which correspond's to Kaneko's remark at the end of Section 2. This seems to be easier to formalise than his actual proof of his Theorem 1, since his proof contains an infinite sum of formal power series, and it was unclear to us how to capture this formally.

lemma *gen-akiyama-tanigawa-fps*:
Abs-fps (λ*n*. *gen-akiyama-tanigawa f n 0* / *fact n*) = *fps-exp 1* ∗ *fps-compose* (*Abs-fps f*) (*1* − *fps-exp 1*)
⟨*proof*⟩

As Kaneko notes in his afore-mentioned remark, if we let $a_{0,k} = \frac{1}{k+1}$, we obtain

$$A(z) = \sum_{k=0}^{\infty} \frac{x^k}{k+1} = -\frac{\ln(1-x)}{x}$$

and therefore

$$\sum_{n=0}^{\infty} a_{n,0} \frac{x^n}{n!} = \frac{xe^x}{e^x - 1} = \frac{x}{1 - e^{-x}},$$

which immediately gives us the connection to the positive Bernoulli numbers.

theorem *bernoulli′-conv-akiyama-tanigawa*: *bernoulli′ n* = *akiyama-tanigawa n 0*
⟨*proof*⟩

theorem *bernoulli-conv-akiyama-tanigawa*:
bernoulli n = *akiyama-tanigawa n 0* − (*if n* = *1 then 1 else 0*)

⟨*proof*⟩

end

end

3.7 Efficient code

We can now compute parts of the Akiyama–Tanigawa (and thereby Bernoulli numbers) with reasonable efficiency but iterating the recurrence row by row. We essentially start with some finite prefix of the zeroth row, say of length n, and then apply the recurrence one to get a prefix of the first row of length $n - 1$ etc.

fun *akiyama-tanigawa-step-aux* :: *nat* ⇒ *real list* ⇒ *real list* **where**
akiyama-tanigawa-step-aux m (*x* # *y* # *xs*) =
real m ∗ (*x* − *y*) # *akiyama-tanigawa-step-aux* (*Suc m*) (*y* # *xs*)
| *akiyama-tanigawa-step-aux m xs* = []

lemma *length-akiyama-tanigawa-step-aux* [*simp*]:
length (*akiyama-tanigawa-step-aux m xs*) = *length xs* − *1*
⟨*proof*⟩

lemma *akiyama-tanigawa-step-aux-eq-Nil-iff* [*simp*]:
akiyama-tanigawa-step-aux m xs = [] ⟷ *length xs* < *2*
⟨*proof*⟩

lemma *nth-akiyama-tanigawa-step-aux*:
n < *length xs* − *1* ⟹
akiyama-tanigawa-step-aux m xs ! *n* = *real* (*m* + *n*) ∗ (*xs* ! *n* − *xs* ! *Suc n*)
⟨*proof*⟩

definition *gen-akiyama-tanigawa-row* **where**
gen-akiyama-tanigawa-row f n l u = *map* (*gen-akiyama-tanigawa f n*) [*l*..<*u*]

lemma *length-gen-akiyama-tanigawa-row* [*simp*]: *length* (*gen-akiyama-tanigawa-row f n l u*) = *u* − *l*
⟨*proof*⟩

lemma *gen-akiyama-tanigawa-row-eq-Nil-iff* [*simp*]:
gen-akiyama-tanigawa-row f n l u = [] ⟷ *l* ≥ *u*
⟨*proof*⟩

lemma *nth-gen-akiyama-tanigawa-row*:
i < *u* − *l* ⟹ *gen-akiyama-tanigawa-row f n l u* ! *i* = *gen-akiyama-tanigawa f n* (*i* + *l*)
⟨*proof*⟩

lemma *gen-akiyama-tanigawa-row-0* [*code*]:

gen-akiyama-tanigawa-row f 0 l u = *map f* [*l*..<*u*]
⟨*proof*⟩

lemma *gen-akiyama-tanigawa-row-Suc* [*code*]:
gen-akiyama-tanigawa-row f (*Suc n*) *l u* =
akiyama-tanigawa-step-aux (*Suc l*) (*gen-akiyama-tanigawa-row f n l* (*Suc u*))
⟨*proof*⟩

lemma *gen-akiyama-tanigawa-row-numeral*:
gen-akiyama-tanigawa-row f (*numeral n*) *l u* =
akiyama-tanigawa-step-aux (*Suc l*) (*gen-akiyama-tanigawa-row f* (*pred-numeral n*) *l* (*Suc u*))
⟨*proof*⟩

lemma *gen-akiyama-tanigawa-code* [*code*]:
gen-akiyama-tanigawa f n k = *hd* (*gen-akiyama-tanigawa-row f n k* (*Suc k*))
⟨*proof*⟩

definition *akiyama-tanigawa-row* **where**
akiyama-tanigawa-row n l u = *map* (*akiyama-tanigawa n*) [*l*..<*u*]

lemma *length-akiyama-tanigawa-row* [*simp*]: *length* (*akiyama-tanigawa-row n l u*) = *u* − *l*
⟨*proof*⟩

lemma *akiyama-tanigawa-row-eq-Nil-iff* [*simp*]:
akiyama-tanigawa-row n l u = [] ⟷ *l* ≥ *u*
⟨*proof*⟩

lemma *nth-akiyama-tanigawa-row*:
i < *u* − *l* ⟹ *akiyama-tanigawa-row n l u* ! *i* = *akiyama-tanigawa n* (*i* + *l*)
⟨*proof*⟩

lemma *akiyama-tanigawa-row-0* [*code*]:
akiyama-tanigawa-row 0 l u = *map* (λ*n*. *inverse* (*real* (*Suc n*))) [*l*..<*u*]
⟨*proof*⟩

lemma *akiyama-tanigawa-row-Suc* [*code*]:
akiyama-tanigawa-row (*Suc n*) *l u* =
akiyama-tanigawa-step-aux (*Suc l*) (*akiyama-tanigawa-row n l* (*Suc u*))
⟨*proof*⟩

lemma *akiyama-tanigawa-row-numeral*:
akiyama-tanigawa-row (*numeral n*) *l u* =
akiyama-tanigawa-step-aux (*Suc l*) (*akiyama-tanigawa-row* (*pred-numeral n*) *l* (*Suc u*))
⟨*proof*⟩

lemma *akiyama-tanigawa-code* [*code*]:
akiyama-tanigawa n k = *hd* (*akiyama-tanigawa-row n k* (*Suc k*))
⟨*proof*⟩

lemma *bernoulli-code* [*code*]:
bernoulli n =
(*if n* = *0 then 1 else if n* = *1 then* −*1*/*2 else if odd n then 0 else akiyama-tanigawa n 0*)
⟨*proof*⟩

lemma *bernoulli′-code* [*code*]:
bernoulli′ n =
(*if n* = *0 then 1 else if n* = *1 then 1*/*2 else if odd n then 0 else akiyama-tanigawa n 0*)
⟨*proof*⟩

Evaluation with the simplifier is much slower than by reflection, but can still be done with much better efficiency than before:

lemmas *eval-bernoulli* =
akiyama-tanigawa-code akiyama-tanigawa-row-numeral
numeral-2-eq-2 [*symmetric*] *akiyama-tanigawa-row-Suc upt-conv-Cons*
akiyama-tanigawa-row-0 bernoulli-code[*of numeral n* **for** *n*]

lemmas *eval-bernoulli′* = *eval-bernoulli bernoulli′-code*[*of numeral n* **for** *n*]

lemmas *eval-bernpoly* =
bernpoly-def atMost-nat-numeral power-eq-if binomial-fact fact-numeral eval-bernoulli

lemma *bernoulli-upto-20* [*simp*]:
bernoulli 2 = *1* / *6*
bernoulli 4 = −(*1* / *30*)
bernoulli 6 = *1* / *42*
bernoulli 8 = − (*1* / *30*)
bernoulli 10 = *5* / *66*
bernoulli 12 = − (*691* / *2730*)
bernoulli 14 = *7* / *6*
bernoulli 16 = −(*3617* / *510*)
bernoulli 18 = *43867* / *798*
bernoulli 20 = −(*174611* / *330*)
⟨*proof*⟩

lemma *bernoulli′-upto-20* [*simp*]:
bernoulli′ 2 = *1* / *6*
bernoulli′ 4 = −(*1* / *30*)
bernoulli′ 6 = *1* / *42*
bernoulli′ 8 = − (*1* / *30*)
bernoulli′ 10 = *5* / *66*

bernoulli' 12 = − (*691* / *2730*)
bernoulli' 14 = *7* / *6*
bernoulli' 16 = −(*3617* / *510*)
bernoulli' 18 = *43867* / *798*
bernoulli' 20 = −(*174611* / *330*)
⟨*proof*⟩

end

4 Bernoulli numbers and the zeta function at positive integers

theory *Bernoulli-Zeta*
imports
HOL−Complex-Analysis.Complex-Analysis
Bernoulli-FPS
begin

lemma *joinpaths-cong*: $f = f' \Longrightarrow g = g' \Longrightarrow f +\!+\!+ g = f' +\!+\!+ g'$
⟨*proof*⟩

lemma *linepath-cong*: $a = a' \Longrightarrow b = b' \Longrightarrow$ *linepath a b* = *linepath a' b'*
⟨*proof*⟩

The analytic continuation of the exponential generating function of the Bernoulli numbers is $\frac{z}{e^z-1}$, which has simple poles at all $2ki\pi$ for $k \in \mathbb{Z} \setminus \{0\}$. We will need the residue at these poles:

lemma *residue-bernoulli*:
assumes $n \neq 0$
shows *residue* (λ*z*. *1* / (*z* ^ *m* ∗ (*exp z* − *1*))) (*2* ∗ *pi* ∗ *real-of-int n* ∗ i) =
1 / (*2* ∗ *pi* ∗ *real-of-int n* ∗ i) ^ *m*
⟨*proof*⟩

At positive integers greater than 1, the Riemann zeta function is simply the infinite sum $\zeta(n) = \sum_{k=1}^{\infty} k^{-n}$. For even n, this quantity can also be expressed in terms of Bernoulli numbers.

To show this, we employ a similar strategy as in the meromorphic asymptotics approach: We apply the Residue Theorem to the exponential generating function of the Bernoulli numbers:

$$\sum_{n=0}^{\infty} \frac{B_n}{n!} z^n = \frac{z}{e^z - 1}$$

Recall that this function has poles at $2ki\pi$ for $k \in \mathbb{Z} \setminus \{0\}$. In the meromorphic asymptotics case, we integrated along a circle of radius $3i\pi$ in order to get the dominant singularities $2i\pi$ and $-2i\pi$. Now, however, we will not use a fixed integration path, but we let the integration path become bigger and bigger. Because the integrand decays relatively quickly if $n > 1$, the integral vanishes in the limit and we obtain not just an asymptotic formula, but an exact representation of B_n as an infinite sum.

For odd n, we have $B_n = 0$, but for even n, the residues at $2ki\pi$ and $-2ki\pi$ combine nicely to $2 \cdot (-2k\pi)^{-n}$, and after some simplification we get the formula for B_n.

Another difference to the meromorphic asymptotics is that we now use a rectangle instead of a circle as the integration path. For the asymptotics, only a big-oh bound was needed for the integral over one fixed integration path, and the circular path was very convenient. However, now we need to explicitly bound the integral for a whole sequence of integration paths that grow in size, and bounding $e^z - 1$ for z on a circle is very tedious. On a rectangle, this term can be bounded much more easily. Still, we have to do this separately for all four edges of the rectangle, which will be a bit tedious.

theorem *nat-even-power-sums-complex*:
  assumes *n'*: *n' > 0*
  shows *(λk. 1 / of-nat (Suc k) ^ (2∗n') :: complex) sums*
  of-real ((−1) ^ Suc n' ∗ bernoulli (2∗n') ∗ (2 ∗ pi) ^ (2 ∗ n') / (2 ∗ fact (2∗n')))
⟨proof⟩

corollary *nat-even-power-sums-real*:
  assumes *n'*: *n' > 0*
  shows *(λk. 1 / real (Suc k) ^ (2∗n')) sums*
  ((−1) ^ Suc n' ∗ bernoulli (2∗n') ∗ (2 ∗ pi) ^ (2 ∗ n') / (2 ∗ fact (2∗n')))
  (**is** *?f sums ?L*)
⟨proof⟩

We can now also easily determine the signs of Bernoulli numbers: the above formula clearly shows that the signs of B_{2n} alternate as n increases, and we already know that $B_{2n+1} = 0$ for any positive n. A lot of other facts about the signs of Bernoulli numbers follow.

corollary *sgn-bernoulli-even*:
  assumes *n > 0*
  shows *sgn (bernoulli (2 ∗ n)) = (−1) ^ Suc n*
⟨proof⟩

corollary *bernoulli-even-nonzero*: *even n ⟹ bernoulli n ≠ 0*
  ⟨proof⟩

corollary *sgn-bernoulli*:
  sgn (bernoulli n) =
  (if n = 0 then 1 else if n = 1 then −1 else if odd n then 0 else (−1) ^ Suc (n div 2))

⟨*proof*⟩

corollary *bernoulli-zero-iff*: $bernoulli\ n = 0 \longleftrightarrow odd\ n \wedge n \neq 1$
⟨*proof*⟩

corollary *bernoulli′-zero-iff*: $(bernoulli'\ n = 0) \longleftrightarrow (n \neq 1 \wedge odd\ n)$
⟨*proof*⟩

corollary *bernoulli-pos-iff*: $bernoulli\ n > 0 \longleftrightarrow n = 0 \vee n\ mod\ 4 = 2$
⟨*proof*⟩

corollary *bernoulli-neg-iff*: $bernoulli\ n < 0 \longleftrightarrow n = 1 \vee n > 0 \wedge 4\ dvd\ n$
⟨*proof*⟩

We also get the solution of the Basel problem (the sum over all squares of positive integers) and any 'Basel-like' problem with even exponent. The case of odd exponents is much more complicated and no similarly nice closed form is known for these.

corollary *nat-squares-sums*: $(\lambda n.\ 1\ /\ (n+1)\ \hat{}\ 2)\ sums\ (pi\ \hat{}\ 2\ /\ 6)$
⟨*proof*⟩

corollary *nat-power4-sums*: $(\lambda n.\ 1\ /\ (n+1)\ \hat{}\ 4)\ sums\ (pi\ \hat{}\ 4\ /\ 90)$
⟨*proof*⟩

corollary *nat-power6-sums*: $(\lambda n.\ 1\ /\ (n+1)\ \hat{}\ 6)\ sums\ (pi\ \hat{}\ 6\ /\ 945)$
⟨*proof*⟩

corollary *nat-power8-sums*: $(\lambda n.\ 1\ /\ (n+1)\ \hat{}\ 8)\ sums\ (pi\ \hat{}\ 8\ /\ 9450)$
⟨*proof*⟩

end

References

[1] S. Akiyama and Y. Tanigawa. Multiple zeta values at non-positive integers. *The Ramanujan Journal*, 5(4):327–351, 2001.

[2] M. Kaneko. The Akiyama–Tanigawa algorithm for Bernoulli numbers. *Journal of Integer Sequences*, 3, 2000.

[3] M. Riedel. Bernoulli numbers explicit form. https://math.stackexchange.com/a/784156/67576, 2014.