Bernoulli Numbers

Lukas Bulwahn and Manuel Eberl

May 21, 2019

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*: $a \mathbin{\hat{}} \mathit{numeral}\ n = a * a \mathbin{\hat{}} \mathit{pred\text{-}numeral}\ n$
  by (*simp only*: *numeral-eq-Suc power-Suc*)

lemma *fact-diff-Suc*: $n < \mathit{Suc}\ m \Longrightarrow \mathit{fact}\ (\mathit{Suc}\ m - n) = \mathit{of\text{-}nat}\ (\mathit{Suc}\ m - n) * \mathit{fact}\ (m - n)$
  by (*subst fact-reduce*) *auto*

lemma *of-nat-binomial-Suc*:
  assumes $k \leq n$
  shows $(\mathit{of\text{-}nat}\ (\mathit{Suc}\ n\ \mathit{choose}\ k) :: 'a :: \mathit{field\text{-}char\text{-}0}) =$
    $\mathit{of\text{-}nat}\ (\mathit{Suc}\ n)\ /\ \mathit{of\text{-}nat}\ (\mathit{Suc}\ n - k) * \mathit{of\text{-}nat}\ (n\ \mathit{choose}\ k)$
  using *assms* **by** (*simp add*: *binomial-fact divide-simps fact-diff-Suc of-nat-diff del*: *of-nat-Suc*)

lemma *integrals-eq*:
  assumes $f\ 0 = g\ 0$
  assumes $\bigwedge x.\ ((\lambda x.\ f\ x - g\ x)\ \mathit{has\text{-}real\text{-}derivative}\ 0)\ (\mathit{at}\ x)$
  shows $f\ x = g\ x$
proof −
  show $f\ x = g\ x$
  proof (*cases* $x \neq 0$)
    case *True*
    from *assms DERIV-const-ratio-const*[*OF this*, *of* $\lambda x.\ f\ x - g\ x\ 0$]
    show *?thesis* **by** *auto*
  qed (*simp add*: *assms*)
qed

lemma *sum-diff*: $((\sum i{\leq}n{::}\mathit{nat}.\ f\ (i + 1) - f\ i){::}'a{::}\mathit{field}) = f\ (n + 1) - f\ 0$
  by (*induct n*) (*auto simp add*: *field-simps*)

lemma *Rats-sum*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}) \Longrightarrow \mathit{sum}\ f\ A \in \mathbb{Q}$
  by (*induction A rule*: *infinite-finite-induct*) *simp-all*

1.2 Bernoulli Numbers and Bernoulli Polynomials

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

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

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* **by** (*simp add*: *bernoulli-Suc*)
lemma *bernoulli-Suc-0* [*simp*]: *bernoulli* (*Suc 0*) = *−1/2* **by** (*simp add*: *bernoulli-Suc*)

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* **by** (*simp add*: *bernoulli'-def*)

lemma *bernoulli'-1* [*simp*]: *bernoulli'* (*Suc 0*) = *1/2*
by (*simp add*: *bernoulli'-def*)

lemma *bernoulli-conv-bernoulli'*: *n* ≠ *1* ⟹ *bernoulli n* = *bernoulli' n*
by (*simp add*: *bernoulli'-def*)

lemma *bernoulli'-conv-bernoulli*: *n* ≠ *1* ⟹ *bernoulli' n* = *bernoulli n*
by (*simp add*: *bernoulli'-def*)

lemma *bernoulli-conv-bernoulli'-if*:
n ≠ *1* ⟹ *bernoulli n* = (*if n* = *1 then −1/2 else bernoulli' n*)
by (*simp add*: *bernoulli'-def*)

lemma *bernoulli-in-Rats*: *bernoulli n* ∈ ℚ
proof (*induction n rule*: *less-induct*)
case (*less n*)
thus *?case*
by (*cases n*) (*auto simp*: *bernoulli-Suc intro*!: *Rats-sum Rats-divide*)
qed

lemma *bernoulli'-in-Rats*: *bernoulli' n* ∈ ℚ
by (*simp add*: *bernoulli'-def bernoulli-in-Rats*)

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

fix $x :: 'a$
have *bernpoly n x* = $(\sum k{\le}n.$ *of-nat* (*n choose* (*n* − *k*)) ∗
of-real (*bernoulli* (*n* − *k*)) ∗ *x* ^ (*n* − (*n* − *k*)))
unfolding *bernpoly-def* **by** (*rule sum.reindex-bij-witness*[*of* - λ*k*. *n* − *k* λ*k*. *n* − *k*]) *simp-all*
also have ... = $(\sum k{\le}n.$ *of-nat* (*n choose k*) ∗ *of-real* (*bernoulli* (*n* − *k*)) ∗ *x* ^ *k*)
by (*intro sum.cong refl*) (*simp-all add*: *binomial-symmetric* [*symmetric*])
finally show *bernpoly n x* = ... **.**
qed

lemma *bernoulli-Suc'*:
bernoulli (*Suc n*) = −*1*/(*real n* + *2*) ∗ $(\sum k{\le}n.$ *real* (*n* + *2 choose* (*k* + *2*)) ∗ *bernoulli* (*n* − *k*))
proof −
have *bernoulli* (*Suc n*) = − *1* / (*real n* + *2*) ∗ $(\sum k{\le}n.$ *real* (*n* + *2 choose k*) ∗ *bernoulli k*)
unfolding *bernoulli.simps* **..**
also have $(\sum k{\le}n.$ *real* (*n* + *2 choose k*) ∗ *bernoulli k*) =
$(\sum k{\le}n.$ *real* (*n* + *2 choose* (*n* − *k*)) ∗ *bernoulli* (*n* − *k*))
by (*rule sum.reindex-bij-witness*[*of* - λ*k*. *n* − *k* λ*k*. *n* − *k*]) *simp-all*
also have ... = $(\sum k{\le}n.$ *real* (*n* + *2 choose* (*k* + *2*)) ∗ *bernoulli* (*n* − *k*))
by (*intro sum.cong refl*, *subst binomial-symmetric*) *simp-all*
finally show *?thesis* **.**
qed

1.3 Basic Observations on Bernoulli Polynomials

lemma *bernpoly-0* [*simp*]: *bernpoly n 0* = (*of-real* (*bernoulli n*) :: *'a* :: *real-algebra-1*)
proof (*cases n*)
case *0*
then show *bernpoly n 0* = *of-real* (*bernoulli n*)
unfolding *bernpoly-def bernoulli.simps* **by** *auto*
next
case (*Suc n'*)
have $(\sum k{\le}n'.$ *of-nat* (*Suc n' choose k*) ∗ *of-real* (*bernoulli k*) ∗ *0* ^ (*Suc n'* − *k*)) = (*0*::*'a*)
proof (*intro sum.neutral ballI*)
fix *k* **assume** $k \in \{..n'\}$
thus *of-nat* (*Suc n' choose k*) ∗ *of-real* (*bernoulli k*) ∗ (*0*::*'a*) ^ (*Suc n'* − *k*) = *0*
by (*cases Suc n'* − *k*) *auto*
qed
with *Suc* **show** *?thesis*
unfolding *bernpoly-def* **by** *simp*
qed

lemma *continuous-on-bernpoly* [*continuous-intros*]:
continuous-on A (*bernpoly n* :: *'a* ⇒ *'a* :: *real-normed-algebra-1*)

unfolding *bernpoly-def* **by** (*auto intro*!: *continuous-intros*)

lemma *isCont-bernpoly* [*continuous-intros*]:
isCont (*bernpoly n* :: *'a* ⇒ *'a* :: *real-normed-algebra-1*) *x*
unfolding *bernpoly-def* **by** (*auto intro*!: *continuous-intros*)

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 −
have (*bernpoly* (*Suc n*) *has-field-derivative*
(∑ *k*≤*n*. *of-nat* (*Suc n* − *k*) ∗ *x* ^ (*n* − *k*) ∗ (*of-nat* (*Suc n choose k*) ∗
of-real (*bernoulli k*)))) (*at x*) (**is** (- *has-field-derivative ?D*) -)
unfolding *bernpoly-def* **by** (*rule DERIV-cong*) (*fast intro*!: *derivative-intros*, *simp*)
also have *?D* = *of-nat* (*n* + *1*) ∗ *bernpoly n x* **unfolding** *bernpoly-def*
by (*subst sum-distrib-left*, *intro sum.cong refl*, *subst of-nat-binomial-Suc*) *simp-all*
ultimately show *?thesis* **by** (*auto simp del*: *of-nat-Suc One-nat-def*)
qed

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 (*cases n*)
case (*Suc m*)
then show *?thesis*
by (*simp add*: *bernoulli-Suc*)
(*simp add*: *field-simps add-2-eq-Suc'*[*symmetric*] *del*: *add-2-eq-Suc add-2-eq-Suc'*)
qed *simp-all*

lemma *sum-binomial-times-bernoulli'*:
(∑ *k*<*n*. *real* (*n choose k*) ∗ *bernoulli k*) = (*if n* = *1 then 1 else 0*)
proof (*cases n*)
case (*Suc m*)
have (∑ *k*<*n*. *real* (*n choose k*) ∗ *bernoulli k*) =
(∑ *k*≤*m*. *real* (*Suc m choose k*) ∗ *bernoulli k*)
unfolding *Suc lessThan-Suc-atMost* **..**
also have ... = (*if n* = *1 then 1 else 0*)
by (*subst sum-binomial-times-bernoulli*) (*simp add*: *Suc*)
finally show *?thesis* **.**
qed *simp-all*

lemma *binomial-unroll*:
n > *0* ⟹ (*n choose k*) = (*if k* = *0 then 1 else* (*n* − *1*) *choose* (*k* − *1*) + ((*n* − *1*) *choose k*))
by (*auto simp add*: *gr0-conv-Suc*)

lemma *sum-unroll*:
(∑ k≤n::*nat*. *f k*) = (*if n* = *0 then f 0 else f n* + (∑ k≤*n* − *1*. *f k*))
by (*cases n*) (*simp-all add*: *add-ac*)

lemma *bernoulli-unroll*:
n > *0* ⟹ *bernoulli n* = − *1* / (*real n* + *1*) ∗ (∑ k≤*n* − *1*. *real* (*n* + *1 choose k*) ∗ *bernoulli k*)
by (*cases n*) (*simp add*: *bernoulli-Suc*)+

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

lemma *bernpoly-1-1*: *bernpoly 1 1* = *of-real* (*1*/*2*)
proof −
have ∗: (*1* :: *'a*) = *of-real 1* **by** *simp*
have *bernpoly 1* (*1*::*'a*) = *1* − *of-real* (*1* / *2*)
by (*simp add*: *bernoulli-unroll-all*)
also have ... = *of-real* (*1* − *1* / *2*)
by (*simp only*: ∗ *of-real-diff*)
also have *1* − *1* / *2* = (*1* / *2* :: *real*)
by *simp*
finally show *?thesis* **.**
qed

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 (*induct n arbitrary*: *x*)
case *0*
show *?case* **unfolding** *bernpoly-def* **by** *auto*
next
case (*Suc n*)
have *bernpoly* (*Suc n*) (*0* + *1*) − *bernpoly* (*Suc n*) (*0* :: *real*) =
(∑ k≤*n*. *of-real* (*real* (*Suc n choose k*) ∗ *bernoulli k*))
unfolding *bernpoly-0* **unfolding** *bernpoly-def* **by** *simp*
also have ... = *of-nat* (*Suc n*) ∗ *0* ^ *n*
by (*simp only*: *of-real-sum* [*symmetric*] *sum-binomial-times-bernoulli*) *simp*
finally have *const*: *bernpoly* (*Suc n*) (*0* + *1*) − *bernpoly* (*Suc n*) *0* = ...
by *simp*

have *hyps'*: *of-nat* (*Suc n*) ∗ *bernpoly n* (*x* + *1*) −
of-nat (*Suc n*) ∗ *bernpoly n x* =
of-nat n ∗ *of-nat* (*Suc n*) ∗ *x* ^ (*n* − *Suc 0*) **for** *x* :: *real*
unfolding *right-diff-distrib*[*symmetric*]
by (*subst Suc*) (*simp-all add*: *algebra-simps*)
have ((λ*x*. *bernpoly* (*Suc n*) (*x* + *1*) − *bernpoly* (*Suc n*) *x* − *of-nat* (*Suc n*) ∗ *x* ^ *n*)
has-field-derivative 0) (*at x*) **for** *x* :: *real*

```
    by (rule derivative-eq-intros refl)+ (insert hyps'[of x], simp add: algebra-simps)
  from integrals-eq[OF const this] show ?case by simp
qed

lemma bernpoly-of-real: bernpoly n (of-real x) = of-real (bernpoly n x)
  by (simp add: bernpoly-def)

lemma bernpoly-1:
  assumes n ≠ 1
  shows   bernpoly n 1 = of-real (bernoulli n)
proof −
  have bernpoly n 1 = bernoulli n
  proof (cases n ≥ 2)
    case False
    with assms have n = 0 by auto
    thus ?thesis by (simp add: bernpoly-def)
  next
    case True
    with diff-bernpoly[of n 0] show ?thesis
      by (simp add: power-0-left bernpoly-0)
  qed
  hence bernpoly n (of-real 1) = of-real (bernoulli n)
    by (simp only: bernpoly-of-real)
  thus ?thesis by simp
qed

lemma bernpoly-1': bernpoly n 1 = of-real (bernoulli' n)
  using bernpoly-1-1 [where ?'a = 'a]
  by (cases n = 1) (simp-all add: bernpoly-1 bernoulli'-def)

theorem sum-of-powers:
  (∑k≤n::nat. (real k) ^ m) = (bernpoly (Suc m) (n + 1) − bernpoly (Suc m)
0) / (m + 1)
proof −
  from diff-bernpoly[of Suc m, simplified] have (m + (1::real)) * (∑k≤n. (real
k) ^ m) = (∑k≤n. bernpoly (Suc m) (real k + 1) − bernpoly (Suc m) (real k))
    by (auto simp add: sum-distrib-left intro!: sum.cong)
  also have ... = (∑k≤n. bernpoly (Suc m) (real (k + 1)) − bernpoly (Suc m)
(real k))
    by (simp add: add-ac)
  also have ... = bernpoly (Suc m) (n + 1) − bernpoly (Suc m) 0
    by (simp only: sum-diff [where f=λk. bernpoly (Suc m) (real k)]) simp
  finally show ?thesis by (auto simp add: field-simps intro!: eq-divide-imp)
qed

lemma sum-of-powers-nat-aux:
  assumes real a = b / c real b' = b real c' = c
  shows   a = b' div c'
proof (cases c = 0)
```

case *False*
with *assms* **have** *real* $(a * c') = real\ b'$ **by** (*simp add*: *field-simps*)
hence $b' = a * c'$ **by** (*subst* (*asm*) *of-nat-eq-iff*) *simp*
with *False assms* **show** *?thesis* **by** *simp*
qed (*insert assms*, *simp-all*)

1.5 Instances for Square And Cubic Numbers

theorem *sum-of-squares*: *real* $(\sum k{\leq}n{::}nat.\ k\ \hat{}\ 2) = real\ (2 * n\ \hat{}\ 3 + 3 * n\ \hat{}\ 2 + n)\ /\ 6$
by (*simp only*: *of-nat-sum of-nat-power sum-of-powers*)
(*simp add*: *bernoulli-unroll-all field-simps power2-eq-square power-numeral-reduce*)

corollary *sum-of-squares-nat*: $(\sum k{\leq}n{::}nat.\ k\ \hat{}\ 2) = (2 * n\ \hat{}\ 3 + 3 * n\ \hat{}\ 2 + n)\ div\ 6$
by (*rule sum-of-powers-nat-aux*[*OF sum-of-squares*]) *simp-all*

theorem *sum-of-cubes*: *real* $(\sum k{\leq}n{::}nat.\ k\ \hat{}\ 3) = real\ (n\ \hat{}\ 2 + n)\ \hat{}\ 2\ /\ 4$
by (*simp only*: *of-nat-sum of-nat-power sum-of-powers*)
(*simp add*: *bernoulli-unroll-all field-simps power2-eq-square power-numeral-reduce*)

corollary *sum-of-cubes-nat*: $(\sum k{\leq}n{::}nat.\ k\ \hat{}\ 3) = (n\ \hat{}\ 2 + n)\ \hat{}\ 2\ div\ 4$
by (*rule sum-of-powers-nat-aux*[*OF sum-of-cubes*]) *simp-all*

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* **by** (*simp add*: *frac-def*)

lemma *frac-eq-id*: $x \in \{0..<1\} \Longrightarrow frac\ x = x$
by (*simp add*: *frac-eq*)

lemma *periodic-continuous-onI*:
fixes $f :: real \Rightarrow real$

assumes *periodic*: $\bigwedge x.\ f\ (x + p) = f\ x\ p > 0$
assumes *cont*: *continuous-on* $\{a..a+p\}\ f$
shows *continuous-on UNIV f*
unfolding *continuous-on-def*
proof *safe*
fix x :: *real*
interpret f: *periodic-fun-simple f p* **by** *unfold-locales* (*rule periodic*)

have *continuous-on* $\{a-p..a\}\ (f \circ (\lambda x.\ x + p))$
by (*intro continuous-on-compose*) (*auto intro*!: *continuous-intros cont*)
also have $f \circ (\lambda x.\ x + p) = f$ **by** (*rule ext*) (*simp add*: *f.periodic-simps*)
finally have *continuous-on* $(\{a-p..a\} \cup \{a..a+p\})\ f$ **using** *cont*
by (*intro continuous-on-closed-Un*) *simp-all*
also have $\{a-p..a\} \cup \{a..a+p\} = \{a-p..a+p\}$ **by** *auto*
finally have *continuous-on* $\{a-p..a+p\}\ f$ **.**
hence *cont*: *continuous-on* $\{a-p<..<a+p\}\ f$ **by** (*rule continuous-on-subset*) *auto*

define n :: *int* **where** $n = \lceil (a - x) / p \rceil$
have $(a - x) / p \leq n\ n < (a - x) / p + 1$ **unfolding** *n-def* **by** *linarith*+
with ‹$p > 0$› **have** $x + n * p \in \{a-p<..<a + p\}$ **by** (*simp add*: *field-simps*)
with *cont* **have** *isCont f* $(x + n * p)$
by (*subst* (*asm*) *continuous-on-eq-continuous-at*) *auto*
hence $*$: $f \longrightarrow x+n*p\rightarrow f\ (x+n*p)$ **by** (*simp add*: *isCont-def f.periodic-simps*)
have $(\lambda x.\ f\ (x + n*p)) \longrightarrow x\rightarrow f\ (x+n*p)$
by (*intro tendsto-compose*[*OF* $*$] *tendsto-intros*)
thus $f \longrightarrow x\rightarrow f\ x$ **by** (*simp add*: *f.periodic-simps*)
qed

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* $(A \cup B)$)
proof −
from *assms* **have** $((\lambda y.\ (f\ y - f\ x) / (y - x)) \longrightarrow D)$ (*sup* (*at x within A*) (*at x within B*))
unfolding *has-field-derivative-iff* **by** (*rule filterlim-sup*)
also have *sup* (*at x within A*) (*at x within B*) = *at x within* $(A \cup B)$
using *at-within-union* **..**
finally show *?thesis* **unfolding** *has-field-derivative-iff* **.**
qed

lemma *has-field-derivative-cong-ev*′:
assumes $x = y$
and $*$: *eventually* $(\lambda x.\ x \in s \longrightarrow f\ x = g\ x)$ (*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 −

have (*f has-field-derivative u*) (*at x within* (*s* ∪ {*x*})) =
(*g has-field-derivative v*) (*at y within* (*s* ∪ {*x*})) **using** *assms*
by (*intro has-field-derivative-cong-ev*) (*auto elim*!: *eventually-mono*)
also from *assms* **have** *at x within* (*s* ∪ {*x*}) = *at x within s* **by** (*simp add*: *at-within-def*)
also from *assms* **have** *at y within* (*s* ∪ {*x*}) = *at y within t* **by** (*simp add*: *at-within-def*)
finally show *?thesis* **.**
qed

interpretation *frac*: *periodic-fun-simple′ frac*
by *unfold-locales* (*simp add*: *frac-def*)

lemma *tendsto-frac-at-right-0*:
(*frac* ⟶ *0*) (*at-right* (*0* :: *′a* :: {*floor-ceiling*,*order-topology*}))
proof −
have ∗: *eventually* (λ*x*. *x* = *frac x*) (*at-right* (*0*::*′a*))
by (*intro eventually-at-rightI*[*of 0 1*]) (*simp-all add*: *frac-eq eq-commute*[*of - frac x* **for** *x*])
moreover have ∗∗: ((λ*x*::*′a*. *x*) ⟶ *0*) (*at-right 0*)
by (*rule tendsto-ident-at*)
ultimately show *?thesis* **by** (*rule Lim-transform-eventually*)
qed

lemma *tendsto-frac-at-left-1*:
(*frac* ⟶ *1*) (*at-left* (*1* :: *′a* :: {*floor-ceiling*,*order-topology*}))
proof −
have ∗: *eventually* (λ*x*. *x* = *frac x*) (*at-left* (*1*::*′a*))
by (*intro eventually-at-leftI*[*of 0*]) (*simp-all add*: *frac-eq eq-commute*[*of - frac x* **for** *x*])
moreover have ∗∗: ((λ*x*::*′a*. *x*) ⟶ *1*) (*at-left 1*)
by (*rule tendsto-ident-at*)
ultimately show *?thesis* **by** (*rule Lim-transform-eventually*)
qed

lemma *continuous-on-frac* [*THEN continuous-on-subset*, *continuous-intros*]:
continuous-on {*0*::*′a*::{*floor-ceiling*,*order-topology*}..<*1*} *frac*
proof (*subst continuous-on-cong*[*OF refl*])
fix *x* :: *′a* **assume** *x* ∈ {*0*..<*1*}
thus *frac x* = *x* **by** (*simp add*: *frac-eq*)
qed (*auto intro*: *continuous-intros*)

lemma *isCont-frac* [*continuous-intros*]:
assumes (*x* :: *′a* :: {*floor-ceiling*,*order-topology*,*t2-space*}) ∈ {*0*<..<*1*}
shows *isCont frac x*
proof −
have *continuous-on* {*0*<..<(*1*::*′a*)} *frac* **by** (*rule continuous-on-frac*) *auto*
with *assms* **show** *?thesis*

```
    by (subst (asm) continuous-on-eq-continuous-at) auto
qed

lemma has-field-derivative-frac:
  assumes (x::real) ∉ ℤ
  shows   (frac has-field-derivative 1) (at x)
proof −
  have ((λt. t − of-int ⌊x⌋) has-field-derivative 1) (at x)
    by (auto intro!: derivative-eq-intros)
  also have ?this ⟷ ?thesis
    using eventually-floor-eq[OF filterlim-ident assms]
    by (intro DERIV-cong-ev refl) (auto elim!: eventually-mono simp: frac-def)
  finally show ?thesis .
qed

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

lemma continuous-on-compose-fracI:
  fixes f :: real ⇒ real
  assumes cont1: continuous-on {0..1} f
  assumes cont2: f 0 = f 1
  shows   continuous-on UNIV (λx. f (frac x))
proof (rule periodic-continuous-onI)
  have cont: continuous-on {0..1} (λx. f (frac x))
    unfolding continuous-on-def
  proof safe
    fix x :: real assume x: x ∈ {0..1}
    show ((λx. f (frac x)) ⟶ f (frac x)) (at x within {0..1})
    proof (cases x = 1)
      case False
      with x have [simp]: frac x = x by (simp add: frac-eq)
      from x False have eventually (λx. x ∈ {..<1}) (nhds x)
        by (intro eventually-nhds-in-open) auto
      hence eventually (λx. frac x = x) (at x within {0..1})
        by (auto simp: eventually-at-filter frac-eq elim!: eventually-mono)
      hence eventually (λx. f x = f (frac x)) (at x within {0..1})
        by eventually-elim simp
      moreover from cont1 x have (f ⟶ f (frac x)) (at x within {0..1})
        by (simp add: continuous-on-def)
      ultimately show ((λx. f (frac x)) ⟶ f (frac x)) (at x within {0..1})
        by (rule Lim-transform-eventually)
    next
      case True
        from cont1 have ∗∗: (f ⟶ f 1) (at 1 within {0..1}) by (simp add:
continuous-on-def)
      moreover have ∗: filterlim frac (at 1 within {0..1}) (at 1 within {0..1})
      proof (subst filterlim-cong[OF refl refl])
        show eventually (λx. frac x = x) (at 1 within {0..1})
```

```
      by (auto simp: eventually-at-filter frac-eq)
    qed (simp add: filterlim-ident)
    ultimately have ((λx. f (frac x)) ⟶ f 1) (at 1 within {0..1})
      by (rule filterlim-compose)
    thus ?thesis by (simp add: True cont2 frac-def)
  qed
 qed
 thus continuous-on {0..0+1} (λx. f (frac x)) by simp
qed (simp-all add: frac.periodic-simps)



definition pbernpoly :: nat ⇒ real ⇒ real where
 pbernpoly n x = bernpoly n (frac x)

lemma pbernpoly-0 [simp]: pbernpoly n 0 = bernoulli n
 by (simp add: pbernpoly-def)

lemma pbernpoly-eq-bernpoly: x ∈ {0..<1} ⟹ pbernpoly n x = bernpoly n x
 by (simp add: pbernpoly-def frac-eq-id)

interpretation pbernpoly: periodic-fun-simple' pbernpoly n
 by unfold-locales (simp add: pbernpoly-def frac.periodic-simps)

lemma continuous-on-pbernpoly [continuous-intros]:
 assumes n ≠ 1
 shows   continuous-on A (pbernpoly n)
proof (cases n = 0)
 case True
 thus ?thesis by (auto intro: continuous-intros simp: pbernpoly-def bernpoly-def)
next
 case False
 with assms have n: n ≥ 2 by auto
 have continuous-on UNIV (pbernpoly n) unfolding pbernpoly-def [abs-def]
   by (rule continuous-on-compose-fracI)
      (insert n, auto intro!: continuous-intros simp: bernpoly-0 bernpoly-1)
 thus ?thesis by (rule continuous-on-subset) simp-all
qed

lemma continuous-on-pbernpoly' [continuous-intros]:
 assumes n ≠ 1 continuous-on A f
 shows   continuous-on A (λx. pbernpoly n (f x))
 using continuous-on-compose[OF assms(2) continuous-on-pbernpoly[OF assms(1)]]
 by (simp add: o-def)

lemma isCont-pbernpoly [continuous-intros]: n ≠ 1 ⟹ isCont (pbernpoly n) x
 using continuous-on-pbernpoly[of n UNIV] by (simp add: continuous-on-eq-continuous-at)
```

lemma *has-field-derivative-pbernpoly-Suc*:
assumes $n \geq 2 \vee x \notin \mathbb{Z}$
shows (*pbernpoly* (*Suc n*) *has-field-derivative real* (*Suc n*) $*$ *pbernpoly n x*) (*at x*)
using *assms*
proof (*cases* $x \in \mathbb{Z}$)
assume $x \notin \mathbb{Z}$
with *assms* **show** *?thesis* **unfolding** *pbernpoly-def*
by (*auto intro*!: *derivative-eq-intros simp del*: *of-nat-Suc*)
next
case *True*
from *True* **obtain** *k* **where** *k*: $x =$ *of-int k* **by** (*auto elim*: *Ints-cases*)
have (*pbernpoly* (*Suc n*) *has-field-derivative real* (*Suc n*) $*$ *pbernpoly n x*)
(*at x within* ($\{..<x\} \cup \{x<..\}$))
proof (*rule has-field-derivative-at-within-union*)
have ((λx. *bernpoly* (*Suc n*) (*x* − *of-int* (*k*−*1*))) *has-field-derivative*
real (*Suc n*) $*$ *bernpoly n* (*x* − *of-int* (*k*−*1*))) (*at-left x*)
by (*auto intro*!: *derivative-eq-intros*)
also have *?this* $\longleftrightarrow$ (*pbernpoly* (*Suc n*) *has-field-derivative*
real (*Suc n*) $*$ *pbernpoly n x*) (*at-left x*) **using** *assms*
proof (*intro has-field-derivative-cong-ev*′ *refl*)
have $\forall_F$ *y in nhds x*. $y \in \{x - 1<..<x + 1\}$ **by** (*intro eventually-nhds-in-open*) *simp-all*
thus $\forall_F$ *t in nhds x*. $t \in \{..<x\} \longrightarrow$ *bernpoly* (*Suc n*) (*t* − *real-of-int* (*k* − *1*)) =
pbernpoly (*Suc n*) *t*
proof (*elim eventually-mono*, *safe*)
fix *t* **assume** $t < x$ $t \in \{x-1<..<x+1\}$
hence *frac t* = *t* − *real-of-int* (*k* − *1*) **using** *k*
by (*subst frac-unique-iff*) *auto*
thus *bernpoly* (*Suc n*) (*t* − *real-of-int* (*k* − *1*)) = *pbernpoly* (*Suc n*) *t*
by (*simp add*: *pbernpoly-def*)
qed
qed (*insert k*, *auto simp*: *pbernpoly-def bernpoly-1*)
finally show (*pbernpoly* (*Suc n*) *has-real-derivative*
real (*Suc n*) $*$ *pbernpoly n x*) (*at-left x*) **.**
next
have ((λx. *bernpoly* (*Suc n*) (*x* − *of-int k*)) *has-field-derivative*
real (*Suc n*) $*$ *bernpoly n* (*x* − *of-int k*)) (*at-right x*)
by (*auto intro*!: *derivative-eq-intros*)
also have *?this* $\longleftrightarrow$ (*pbernpoly* (*Suc n*) *has-field-derivative*
real (*Suc n*) $*$ *pbernpoly n x*) (*at-right x*) **using** *assms*
proof (*intro has-field-derivative-cong-ev*′ *refl*)
have $\forall_F$ *y in nhds x*. $y \in \{x - 1<..<x + 1\}$ **by** (*intro eventually-nhds-in-open*) *simp-all*
thus $\forall_F$ *t in nhds x*. $t \in \{x<..\} \longrightarrow$ *bernpoly* (*Suc n*) (*t* − *real-of-int k*) =
pbernpoly (*Suc n*) *t*
proof (*elim eventually-mono*, *safe*)
fix *t* **assume** $t > x$ $t \in \{x-1<..<x+1\}$

```
      hence frac t = t − real-of-int k using k
        by (subst frac-unique-iff) auto
      thus bernpoly (Suc n) (t − real-of-int k) = pbernpoly (Suc n) t
        by (simp add: pbernpoly-def)
    qed
  qed (insert k, auto simp: pbernpoly-def bernpoly-1)
  finally show (pbernpoly (Suc n) has-real-derivative
                real (Suc n) * pbernpoly n x) (at-right x) .
 qed
 also have {..<x} ∪ {x<..} = UNIV − {x} by auto
 also have at x within ... = at x by (simp add: at-within-def)
 finally show ?thesis .
qed

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

lemma bounded-pbernpoly: obtains c where ⋀x. norm (pbernpoly n x) ≤ c
proof −
 have ∃x∈{0..1}. ∀y∈{0..1}. norm (bernpoly n y :: real) ≤ norm (bernpoly n x
:: real)
   by (intro continuous-attains-sup) (auto intro!: continuous-intros)
 then obtain x where x:
   ⋀y. y ∈ {0..1} ⟹ norm (bernpoly n y :: real) ≤ norm (bernpoly n x :: real)
   by blast
 have norm (pbernpoly n y) ≤ norm (bernpoly n x :: real) for y
   unfolding pbernpoly-def using frac-lt-1[of y] by (intro x) simp-all
 thus ?thesis by (rule that)
qed

end
```

3 Connection of Bernoulli numbers to formal power series

```
theory Bernoulli-FPS
 imports
   Bernoulli
   HOL−Computational-Algebra.Formal-Power-Series
   HOL−Library.Stirling
begin
```

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 = (λb. a * fps-deriv b)

lemma fps-XD'-0 [simp]: fps-XD' a 0 = 0 by (simp add: fps-XD'-def)
lemma fps-XD'-1 [simp]: fps-XD' a 1 = 0 by (simp add: fps-XD'-def)
lemma fps-XD'-fps-const [simp]: fps-XD' a (fps-const b) = 0 by (simp add:
fps-XD'-def)
lemma fps-XD'-fps-of-nat [simp]: fps-XD' a (of-nat b) = 0 by (simp add: fps-XD'-def)
lemma fps-XD'-fps-of-int [simp]: fps-XD' a (of-int b) = 0 by (simp add: fps-XD'-def)
lemma fps-XD'-fps-numeral [simp]: fps-XD' a (numeral b) = 0 by (simp add:
fps-XD'-def)

lemma fps-XD'-add [simp]: fps-XD' a (b + c :: 'a :: comm-ring-1 fps) = fps-XD'
a b + fps-XD' a c
  by (simp add: fps-XD'-def algebra-simps)

lemma fps-XD'-minus [simp]: fps-XD' a (b - c :: 'a :: comm-ring-1 fps) =
fps-XD' a b - fps-XD' a c
  by (simp add: fps-XD'-def algebra-simps)

lemma fps-XD'-prod: fps-XD' a (b * c :: 'a :: comm-ring-1 fps) = fps-XD' a b *
c + b * fps-XD' a c
  by (simp add: fps-XD'-def algebra-simps)

lemma fps-XD'-power: fps-XD' a (b ^ n :: 'a :: idom fps) = of-nat n * b ^ (n -
1) * fps-XD' a b
proof (cases n = 0)
  case False
  have b * fps-XD' a (b ^ n) = of-nat n * b ^ n * fps-XD' a b
    by (induction n) (simp-all add: fps-XD'-prod algebra-simps)
  also have ... = b * (of-nat n * b ^ (n - 1) * fps-XD' a b)
    by (cases n) (simp-all add: algebra-simps)
  finally show ?thesis using False
    by (subst (asm) mult-cancel-left) (auto simp: power-0-left)
qed simp-all
```

lemma *fps-XD'-power-Suc*: *fps-XD' a (b ^ Suc n :: 'a :: idom fps) = of-nat (Suc n) * b ^ n * fps-XD' a b*
by (*subst fps-XD'-power*) *simp-all*

lemma *fps-XD'-sum*: *fps-XD' a (sum f A) = sum (λx. fps-XD' (a :: 'a :: comm-ring-1 fps) (f x)) A*
by (*induction A rule*: *infinite-finite-induct*) *simp-all*

lemma *fps-XD'-funpow-affine*:
fixes *G H* :: *real fps*
assumes [*simp*]: *fps-deriv G = 1*
defines *S* ≡ *λn 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 (*induction n arbitrary*: *H*)
case *0*
thus *?case* **by** (*simp add*: *S-def*)
next
case (*Suc n H*)
have *(∑ m≤Suc n. S (Suc n) m * G ^ m * (fps-deriv ^^ m) H) =*
*(∑ i≤n. of-nat (Suc i) * S n (Suc i) * G ^ Suc i * (fps-deriv ^^ Suc i) H)*
+
*(∑ i≤n. S n i * G ^ Suc i * (fps-deriv ^^ Suc i) H)*
(**is** - = *sum (λi. ?f (Suc i)) ... + ?S2*)
by (*subst sum.atMost-Suc-shift*) (*simp-all add*: *sum.distrib algebra-simps fps-of-nat S-def*
fps-const-add [*symmetric*] *fps-const-mult* [*symmetric*] *del*: *fps-const-add fps-const-mult*)
also have *sum (λi. ?f (Suc i)) {..n} = sum (λi. ?f (Suc i)) {..<n}*
by (*intro sum.mono-neutral-right*) (*auto simp*: *S-def*)
also have ... = *?f 0 + ...* **by** *simp*
also have ... = *sum ?f {..n}* **by** (*subst sum.atMost-shift* [*symmetric*]) *simp-all*
also have ... + *?S2 = (∑ x≤n. fps-XD' G (S n x * G ^ x * (fps-deriv ^^ x) H))*
unfolding *sum.distrib* [*symmetric*]
proof (*rule sum.cong*, *goal-cases*)
case (*2 i*)
thus *?case* **unfolding** *fps-XD'-prod fps-XD'-power*
by (*cases i*) (*auto simp*: *fps-XD'-prod fps-XD'-power-Suc algebra-simps of-nat-diff S-def fps-XD'-def*)
qed *simp-all*
also have ... = *(fps-XD' G ^^ Suc n) H* **by** (*simp add*: *Suc.IH fps-XD'-sum*)
finally show *?case* **..**
qed

3.1 Generating function of Stirling numbers

lemma *Stirling-n-0*: *Stirling n 0 = (if n = 0 then 1 else 0)*
by (*cases n*) *simp-all*

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 −
have *real* (*Stirling* (*Suc n*) (*Suc m*)) = *real* (∑ *i* = *0..n*. *Stirling i m* ∗ (*n choose i*))
proof (*induction n arbitrary*: *m*)
case (*Suc n m*)
have *real* (∑ *i* = *0..Suc n*. *Stirling i m* ∗ (*Suc n choose i*)) =
real (∑ *i* = *0..n*. *Stirling* (*Suc i*) *m* ∗ (*Suc n choose Suc i*)) + *real* (*Stirling 0 m*)
by (*subst sum.atLeast0-atMost-Suc-shift*) *simp-all*
also have *real* (∑ *i* = *0..n*. *Stirling* (*Suc i*) *m* ∗ (*Suc n choose Suc i*)) =
real (∑ *i* = *0..n*. (*n choose i*) ∗ *Stirling* (*Suc i*) *m*) +
real (∑ *i* = *0..n*. (*n choose Suc i*) ∗ *Stirling* (*Suc i*) *m*)
by (*simp add*: *algebra-simps sum.distrib*)
also have (∑ *i* = *0..n*. (*n choose Suc i*) ∗ *Stirling* (*Suc i*) *m*) =
(∑ *i* = *Suc 0..Suc n*. (*n choose i*) ∗ *Stirling i m*)
by (*subst sum.shift-bounds-cl-Suc-ivl*) *simp-all*
also have ... = (∑ *i* = *Suc 0..n*. (*n choose i*) ∗ *Stirling i m*)
by (*intro sum.mono-neutral-right*) *auto*
also have ... = *real* (∑ *i* = *0..n*. *Stirling i m* ∗ (*n choose i*)) − *real* (*Stirling 0 m*)
by (*simp add*: *sum.atLeast-Suc-atMost mult-ac*)
also have *real* (∑ *i* = *0..n*. *Stirling i m* ∗ (*n choose i*)) = *real* (*Stirling* (*Suc n*) (*Suc m*))
by (*rule Suc.IH* [*symmetric*])
also have *real* (∑ *i* = *0..n*. (*n choose i*) ∗ *Stirling* (*Suc i*) *m*) =
real m ∗ *real* (*Stirling* (*Suc n*) (*Suc m*)) + *real* (*Stirling* (*Suc n*) *m*)
by (*cases m*; (*simp only*: *Suc.IH*, *simp add*: *algebra-simps sum.distrib*
sum-distrib-left sum-distrib-right))
also have ... + (*real* (*Stirling* (*Suc n*) (*Suc m*)) − *real* (*Stirling 0 m*)) + *real* (*Stirling 0 m*) =
real (*Suc m* ∗ *Stirling* (*Suc n*) (*Suc m*) + *Stirling* (*Suc n*) *m*)
by (*simp add*: *algebra-simps del*: *Stirling.simps*)
also have *Suc m* ∗ *Stirling* (*Suc n*) (*Suc m*) + *Stirling* (*Suc n*) *m* =
Stirling (*Suc* (*Suc n*)) (*Suc m*)
by (*rule Stirling.simps*(*4*) [*symmetric*])
finally show *?case* **..**
qed *simp-all*
thus *?thesis* **by** (*subst* (*asm*) *of-nat-eq-iff*)
qed

lemma *Stirling-fps-aux*: (*fps-exp 1 − 1*) ^ *m* $ *n* ∗ *fact n* = *fact m* ∗ *real* (*Stirling n m*)
proof (*induction m arbitrary*: *n*)
case *0*
thus *?case* **by** (*simp add*: *Stirling-n-0*)
next
case (*Suc m n*)
show *?case*
proof (*cases n*)
case *0*
thus *?thesis* **by** *simp*
next
case (*Suc n′*)
hence (*fps-exp 1 − 1* :: *real fps*) ^ *Suc m* $ *n* ∗ *fact n* =
fps-deriv ((*fps-exp 1 − 1*) ^ *Suc m*) $ *n′* ∗ *fact n′*
by (*simp-all add*: *algebra-simps del*: *power-Suc*)
also have *fps-deriv* ((*fps-exp 1 − 1* :: *real fps*) ^ *Suc m*) =
fps-const (*real* (*Suc m*)) ∗ ((*fps-exp 1 − 1*) ^ *m* ∗ *fps-exp 1*)
by (*subst fps-deriv-power*) *simp-all*
also have ... $ *n′* ∗ *fact n′* =
real (*Suc m*) ∗ ((∑ *i* = *0..n′*. (*fps-exp 1 − 1*) ^ *m* $ *i* / *fact* (*n′ − i*)) ∗ *fact n′*)
unfolding *fps-mult-left-const-nth*
by (*simp add*: *fps-mult-nth Suc.IH sum-distrib-right del*: *of-nat-Suc*)
also have (∑ *i* = *0..n′*. (*fps-exp 1 − 1* :: *real fps*) ^ *m* $ *i* / *fact* (*n′ − i*)) ∗ *fact n′* =
(∑ *i* = *0..n′*. (*fps-exp 1 − 1*) ^ *m* $ *i* ∗ *fact n′* / *fact* (*n′ − i*))
by (*subst sum-distrib-right*, *rule sum.cong*) (*simp-all add*: *divide-simps*)
also have ... = (∑ *i* = *0..n′*. (*fps-exp 1 − 1*) ^ *m* $ *i* ∗ *fact i* ∗ (*n′ choose i*))
by (*intro sum.cong refl*) (*simp-all add*: *binomial-fact*)
also have ... = (∑ *i* = *0..n′*. *fact m* ∗ *real* (*Stirling i m*) ∗ *real* (*n′ choose i*))
by (*simp only*: *Suc.IH*)
also have *real* (*Suc m*) ∗ ... = *fact* (*Suc m*) ∗
(∑ *i* = *0..n′*. *real* (*Stirling i m*) ∗ *real* (*n′ choose i*)) (**is** - = - ∗ *?S*)
by (*simp add*: *sum-distrib-left sum-distrib-right mult-ac del*: *of-nat-Suc*)
also have *?S* = *Stirling* (*Suc n′*) (*Suc m*)
by (*subst sum-Stirling-binomial*) *simp*
also have *Suc n′* = *n* **by** (*simp add*: *Suc*)
finally show *?thesis* **.**
qed
qed

lemma *Stirling-fps-nth*: *Stirling-fps m* $ *n* = *Stirling n m* / *fact n*
unfolding *Stirling-fps-def* **using** *Stirling-fps-aux*[*of m n*] **by** (*simp add*: *field-simps*)

theorem *Stirling-fps-altdef*: *Stirling-fps m* = *Abs-fps* (λ*n*. *Stirling n m* / *fact n*)

by (*simp add*: *fps-eq-iff Stirling-fps-nth*)

theorem *Stirling-closed-form*:
real (*Stirling n k*) = ($\sum j{\leq}k$. (-1) ^ $(k - j)$ * *real* $(k$ *choose* $j)$ * *real* j ^ n) / *fact k*
proof −
have (*fps-exp 1 − 1* :: *real fps*) = (*fps-exp 1* + (-1)) **by** *simp*
also have ... ^ *k* = ($\sum j{\leq}k$. *of-nat* $(k$ *choose* $j)$ * *fps-exp 1* ^ *j* * (-1) ^ $(k - j)$)
unfolding *binomial-ring* **..**
also have ... = ($\sum j{\leq}k$. *fps-const* $((-1)$ ^ $(k - j)$ * *real* $(k$ *choose* $j))$ * *fps-exp* (*real j*))
by (*simp add*: *fps-const-mult* [*symmetric*] *fps-const-power* [*symmetric*] *fps-const-neg* [*symmetric*] *mult-ac fps-of-nat fps-exp-power-mult del*: *fps-const-mult fps-const-power fps-const-neg*)
also have ... $ *n* = ($\sum j{\leq}k$. (-1) ^ $(k - j)$ * *real* $(k$ *choose* $j)$ * *real* j ^ n) / *fact n*
by (*simp add*: *fps-sum-nth sum-divide-distrib*)
also have ... * *fact n* = ($\sum j{\leq}k$. (-1) ^ $(k - j)$ * *real* $(k$ *choose* $j)$ * *real* j ^ n)
by *simp*
also note *Stirling-fps-aux*[*of k n*]
finally show *?thesis* **by** (*simp add*: *atLeast0AtMost field-simps*)
qed

3.2 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* $(\lambda n.$ *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 −
have *: *Abs-fps* $(\lambda n.$ *of-real* (*bernoulli n*) / *fact n* :: *'a*) * (*fps-exp 1 − 1*) = *fps-X*

```
  proof (rule fps-ext)
    fix n
    have (Abs-fps (λn. of-real (bernoulli n) / fact n :: 'a) * (fps-exp 1 - 1)) $ n
=
          (∑ i = 0..n. of-real (bernoulli i) * (1 / fact (n - i) - (if n = i then 1
else 0)) / fact i)
      by (auto simp: fps-mult-nth divide-simps split: if-splits intro!: sum.cong)
    also have ... = (∑ i = 0..n. of-real (bernoulli i) / (fact i * fact (n - i)) -
                                  (if n = i then of-real (bernoulli i) / fact i else 0))
      by (intro sum.cong) (simp-all add: field-simps)
    also have ... = (∑ i = 0..n. of-real (bernoulli i) / (fact i * fact (n - i))) -
                    of-real (bernoulli n) / fact n
      unfolding sum-subtractf by (subst sum.delta') simp-all
    also have ... = (∑ i<n. of-real (bernoulli i) / (fact i * fact (n - i)))
     by (cases n) (simp-all add: atLeast0AtMost lessThan-Suc-atMost [symmetric])
    also have ... = (∑ i<n. fact n * (of-real (bernoulli i) / (fact i * fact (n -
i)))) / fact n
      by (subst sum-distrib-left [symmetric]) simp-all
    also have (∑ i<n. fact n * (of-real (bernoulli i) / (fact i * fact (n - i)))) =
                (∑ i<n. of-nat (n choose i) * of-real (bernoulli i) :: 'a)
      by (intro sum.cong) (simp-all add: binomial-fact)
    also have ... = of-real (∑ i<n. (n choose i) * bernoulli i)
      by simp
    also have ... / fact n = fps-X $ n by (subst sum-binomial-times-bernoulli')
simp-all
    finally show (Abs-fps (λn. of-real (bernoulli n) / fact n :: 'a) * (fps-exp 1 -
1)) $ n =
                    fps-X $ n .
  qed
  moreover show bernoulli-fps = Abs-fps (λn. of-real (bernoulli n) / fact n :: 'a)
    unfolding bernoulli-fps-def by (subst * [symmetric]) simp-all
  ultimately show bernoulli-fps * (fps-exp 1 - 1 :: 'a fps) = fps-X by simp
qed

theorem fps-nth-bernoulli-fps [simp]:
  fps-nth bernoulli-fps n = of-real (bernoulli n) / fact n
  by (simp add: bernoulli-fps-altdef)

lemma bernoulli'-fps-aux:
    (fps-exp 1 - 1) * Abs-fps (λn. of-real (bernoulli' n) / fact n :: 'a) = fps-exp 1
* fps-X
  and bernoulli'-fps-aux':
    (1 - fps-exp (-1)) * Abs-fps (λn. of-real (bernoulli' n) / fact n :: 'a) = fps-X
  and bernoulli'-fps-altdef:
   bernoulli'-fps = Abs-fps (λn. of-real (bernoulli' n) / fact n :: 'a :: real-normed-field)
proof -
  have Abs-fps (λn. of-real (bernoulli' n) / fact n :: 'a) = bernoulli-fps + fps-X
    by (simp add: fps-eq-iff bernoulli'-def)
  also have (fps-exp 1 - 1) * ... = fps-exp 1 * fps-X
```

```
    using bernoulli-fps-aux by (simp add: algebra-simps)
  finally show (fps-exp 1 − 1) ∗ Abs-fps (λn. of-real (bernoulli′ n) / fact n :: ′a)
=
                fps-exp 1 ∗ fps-X .
  also have (fps-exp 1 − 1) = fps-exp 1 ∗ (1 − fps-exp (−1 :: ′a))
    by (simp add: algebra-simps fps-exp-add-mult [symmetric])
  also note mult.assoc
  finally show ∗: (1 − fps-exp (−1)) ∗ Abs-fps (λn. of-real (bernoulli′ n) / fact
n :: ′a) = fps-X
    by (subst (asm) mult-left-cancel) simp-all
  show bernoulli′-fps = Abs-fps (λn. of-real (bernoulli′ n) / fact n :: ′a)
    unfolding bernoulli′-fps-def by (subst ∗ [symmetric]) simp-all
qed

theorem fps-nth-bernoulli′-fps [simp]:
  fps-nth bernoulli′-fps n = of-real (bernoulli′ n) / fact n
  by (simp add: bernoulli′-fps-altdef)

lemma bernoulli-fps-conv-bernoulli′-fps: bernoulli-fps = bernoulli′-fps − fps-X
  by (simp add: fps-eq-iff bernoulli′-def)

lemma bernoulli′-fps-conv-bernoulli-fps: bernoulli′-fps = bernoulli-fps + fps-X
  by (simp add: fps-eq-iff bernoulli′-def)


theorem bernoulli-odd-eq-0:
  assumes n ≠ 1 and odd n
  shows   bernoulli n = 0
proof −
  from bernoulli-fps-aux have 2 ∗ bernoulli-fps ∗ (fps-exp 1 − 1) = 2 ∗ fps-X by
simp
  hence (2 ∗ bernoulli-fps + fps-X) ∗ (fps-exp 1 − 1) = fps-X ∗ (fps-exp 1 + 1)
    by (simp add: algebra-simps)
  also have fps-exp 1 − 1 = fps-exp (1/2) ∗ (fps-exp (1/2) − fps-exp (−1/2 ::
real))
    by (simp add: algebra-simps fps-exp-add-mult [symmetric])
  also have fps-exp 1 + 1 = fps-exp (1/2) ∗ (fps-exp (1/2) + fps-exp (−1/2 ::
real))
    by (simp add: algebra-simps fps-exp-add-mult [symmetric])
  finally have fps-exp (1/2) ∗ ((2 ∗ bernoulli-fps + fps-X) ∗ (fps-exp (1/2) −
fps-exp (− 1/2))) =
                fps-exp (1/2) ∗ (fps-X ∗ (fps-exp (1/2) + fps-exp (−1/2 :: real)))
    by (simp add: algebra-simps)
  hence ∗: (2 ∗ bernoulli-fps + fps-X) ∗ (fps-exp (1/2) − fps-exp (− 1/2)) =
            fps-X ∗ (fps-exp (1/2) + fps-exp (−1/2 :: real))
    (is ?lhs = ?rhs) by (subst (asm) mult-cancel-left) simp-all
  have fps-compose ?lhs (−fps-X) = fps-compose ?rhs (−fps-X) by (simp only: ∗)
  also have fps-compose ?lhs (−fps-X) =
          (−2 ∗ (bernoulli-fps oo − fps-X) + fps-X) ∗ (fps-exp ((1/2)) − fps-exp
```

```
(−1/2))
    by (simp add: fps-compose-mult-distrib fps-compose-add-distrib
                  fps-compose-sub-distrib algebra-simps)
  also have fps-compose ?rhs (−fps-X) = −?rhs
   by (simp add: fps-compose-mult-distrib fps-compose-add-distrib fps-compose-sub-distrib)
  also note ∗ [symmetric]
  also have − ((2 ∗ bernoulli-fps + fps-X) ∗ (fps-exp (1/2) − fps-exp (−1/2)))
=
                ((−2 ∗ bernoulli-fps − fps-X) ∗ (fps-exp (1/2) − fps-exp (−1/2)))
by (simp add: algebra-simps)
  finally have 2 ∗ (bernoulli-fps oo − fps-X) = 2 ∗ (bernoulli-fps + fps-X :: real
fps)
    by (subst (asm) mult-cancel-right) (simp add: algebra-simps)
  hence ∗∗: bernoulli-fps oo −fps-X = (bernoulli-fps + fps-X :: real fps)
    by (subst (asm) mult-cancel-left) simp

  from assms have (bernoulli-fps oo −fps-X) $ n = bernoulli n / fact n
    by (subst ∗∗) simp
  also have −fps-X = fps-const (−1 :: real) ∗ fps-X
    by (simp only: fps-const-neg [symmetric] fps-const-1-eq-1) simp
  also from assms have (bernoulli-fps oo ...) $ n = − bernoulli n / fact n
    by (subst fps-compose-linear) simp
  finally show ?thesis by simp
qed

lemma bernoulli'-odd-eq-0: n ≠ 1 ⟹ odd n ⟹ bernoulli' n = 0
  by (simp add: bernoulli'-def bernoulli-odd-eq-0)
```

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
  by (rule bernoulli-odd-eq-0[OF - odd-numeral]) auto

lemma bernoulli'-odd-numeral-eq-0 [simp]: bernoulli' (numeral (Num.Bit1 n)) =
0
  by (simp add: bernoulli'-def)
```

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 −
  have (∑ m≤n. ∑ k≤m. (−1) ^ k ∗ real (m choose k) ∗ real k ^ n / real (Suc m))
=
        (∑ m≤n. (∑ k≤m. (−1) ^ k ∗ real (m choose k) ∗ real k ^ n) / real (Suc
m))
```

```
    by (subst sum-divide-distrib) simp-all
  also have ... = fact n * (∑ m≤n. (- 1) ^ m / real (Suc m) * (fps-exp 1 -
1) ^ m $ n)
  proof (subst sum-distrib-left, intro sum.cong refl)
    fix m assume m: m ∈ {..n}
    have (∑ k≤m. (-1) ^k * real (m choose k) * real k^n) =
          (-1) ^m * (∑ k≤m. (-1) ^(m - k) * real (m choose k) * real k^n)
     by (subst sum-distrib-left, intro sum.cong refl) (auto simp: minus-one-power-iff)
    also have ... = (-1) ^ m * (real (Stirling n m) * fact m)
      by (subst Stirling-closed-form) simp-all
    also have real (Stirling n m) = Stirling-fps m $ n * fact n
      by (subst Stirling-fps-nth) simp-all
     also have ... * fact m = (fps-exp 1 - 1) ^ m $ n * fact n by (simp add:
Stirling-fps-def)
    finally show (∑ k≤m. (-1) ^k * real (m choose k) * real k^n) / real (Suc m)
=
                  fact n * ((- 1) ^ m / real (Suc m) * (fps-exp 1 - 1) ^ m $ n)
by simp
  qed
  also have (∑ m≤n. (- 1) ^ m / real (Suc m) * (fps-exp 1 - 1) ^ m $ n) =
              fps-compose (Abs-fps (λm. (-1) ^ m / real (Suc m))) (fps-exp 1 -
1) $ n
    by (simp add: fps-compose-def atLeast0AtMost fps-sum-nth)
  also have fps-ln 1 = fps-X * Abs-fps (λm. (-1) ^ m / real (Suc m))
    unfolding fps-ln-def by (auto simp: fps-eq-iff)
  hence Abs-fps (λm. (-1) ^ m / real (Suc m)) = fps-ln 1 / fps-X
    by (metis fps-X-neq-zero nonzero-mult-div-cancel-left)
  also have fps-compose ... (fps-exp 1 - 1) =
            fps-compose (fps-ln 1) (fps-exp 1 - 1) / (fps-exp 1 - 1)
    by (subst fps-compose-divide-distrib) auto
  also have fps-compose (fps-ln 1) (fps-exp 1 - 1 :: real fps) = fps-X
    by (simp add: fps-ln-fps-exp-inv fps-inv-fps-exp-compose)
 also have (fps-X / (fps-exp 1 - 1)) = bernoulli-fps by (simp add: bernoulli-fps-def)
  also have fact n * ... $ n = bernoulli n by simp
  finally show ?thesis ..
qed
```

3.3 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
```

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

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 (*rule fps-ext*)
fix *m* :: *nat*
show *AT-fps f* (*Suc n*) $ *m* = ((*fps-X* − *1*) ∗ *fps-deriv* (*AT-fps f n*)) $ *m*
by (*cases m*) (*simp-all add*: *AT-fps-def fps-deriv-def algebra-simps*)
qed

private 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 −
have *AT-fps f n* = (*fps-XD'* (*fps-X* − *1*) ^^ *n*) (*AT-fps f 0*)
by (*induction n*) (*simp-all add*: *AT-fps-Suc fps-XD'-def*)
also have ... = (∑ *m*≤*n*. *fps-const* (*real* (*Stirling n m*)) ∗ (*fps-X* − *1*) ^ *m* ∗
(*fps-deriv* ^^ *m*) (*AT-fps f 0*))
by (*rule fps-XD'-funpow-affine*) *simp-all*
finally show *?thesis* **.**
qed

private lemma *AT-fps-0-nth*: *AT-fps f 0* $ *n* = (*if n* = *0 then* −*f 0 else f* (*n* − *1*) − *f n*)
by (*simp add*: *AT-fps-def algebra-simps*)

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 (*cases n* = *0*)
case *False*
note [*simp del*] = *gen-akiyama-tanigawa.simps*
have *gen-akiyama-tanigawa f n 0* = −(*AT-fps f n* $ *0*) **by** (*simp add*: *AT-fps-def*)
also have *AT-fps f n* $ *0* = (∑ *k*≤*n*. *real* (*Stirling n k*) ∗ (− *1*) ^ *k* ∗ (*fact k* ∗ *AT-fps f 0* $ *k*))
by (*subst AT-fps-altdef*) (*simp add*: *fps-sum-nth fps-nth-power-0 fps-0th-higher-deriv*)

```
  also have ... = (∑k≤n. real (Stirling n k) * (- 1) ^ k * (fact k * (f (k - 1)
- f k)))
    using False by (intro sum.cong refl) (auto simp: Stirling-n-0 AT-fps-0-nth)
  also have ... = (∑k≤n. fact k * (real (Stirling n k) * (- 1) ^ k) * f (k - 1))
-
                  (∑k≤n. fact k * (real (Stirling n k) * (- 1) ^ k) * f k)
    (is - = sum ?f - - ?S2) by (simp add: sum-subtractf algebra-simps)
  also from False have sum ?f {..n} = sum ?f {0<..n}
    by (intro sum.mono-neutral-right) (auto simp: Stirling-n-0)
  also have ... = sum ?f {0<..Suc n}
    by (intro sum.mono-neutral-left) auto
  also have {0<..Suc n} = {Suc 0..Suc n} by auto
  also have sum ?f ... = sum (λn. ?f (Suc n)) {0..n}
    by (subst sum.atLeast-Suc-atMost-Suc-shift) simp-all
  also have {0..n} = {..n} by auto
  also have sum (λn. ?f (Suc n)) ... - ?S2 =
              (∑k≤n. -((-1) ^k * fact k * real (Stirling (Suc n) (Suc k)) * f k))
   by (subst sum-subtractf [symmetric], intro sum.cong) (simp-all add: algebra-simps)
  also have -... = (∑k≤n. ((-1) ^k * fact k * real (Stirling (Suc n) (Suc k)) *
f k))
    by (simp add: sum-negf)
  finally show ?thesis .
qed simp-all
```

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 (λn. gen-akiyama-tanigawa f n 0 / fact n) = fps-exp 1 * fps-compose
(Abs-fps f) (1 - fps-exp 1)
proof (rule fps-ext)
  fix n :: nat
  have (fps-const (fact n) *
        (fps-compose (Abs-fps (λn. gen-akiyama-tanigawa f 0 n)) (1 - fps-exp 1)
* fps-exp 1)) $ n =
       (∑m≤n. ∑k≤m. (1 - fps-exp 1) ^ k $ m * fact n / fact (n - m) * f k)
    unfolding fps-mult-left-const-nth
   by (simp add: fps-times-def fps-compose-def gen-akiyama-tanigawa-n-0 sum-Stirling-binomial
                 field-simps sum-distrib-left sum-distrib-right atLeast0AtMost
           del: Stirling.simps of-nat-Suc)
  also have ... = (∑m≤n. ∑k≤m. (-1) ^k * fact k * real (Stirling m k) * real
(n choose m) * f k)
  proof (intro sum.cong refl, goal-cases)
```

```
    case (1 m k)
    have (1 − fps-exp 1 :: real fps) ^ k = (−fps-exp 1 + 1 :: real fps) ^ k by simp
    also have ... = (∑ i≤k. of-nat (k choose i) * (−1) ^ i * fps-exp (real i))
     by (subst binomial-ring) (simp add: atLeast0AtMost power-minus' fps-exp-power-mult
mult.assoc)
     also have ... = (∑ i≤k. fps-const (real (k choose i) * (−1) ^ i) * fps-exp
(real i))
    by (simp add: fps-const-mult [symmetric] fps-of-nat fps-const-power [symmetric]

                                fps-const-neg [symmetric] del: fps-const-mult fps-const-power
fps-const-neg)
    also have ... $ m = (∑ i≤k. real (k choose i) * (− 1) ^ i * real i ^ m) / fact
m
      (is - = ?S / -) by (simp add: fps-sum-nth sum-divide-distrib [symmetric])
    also have ?S = (−1) ^ k * (∑ i≤k. (−1) ^ (k − i) * real (k choose i) * real
i ^ m)
     by (subst sum-distrib-left, intro sum.cong refl) (auto simp: minus-one-power-iff)
    also have (∑ i≤k. (−1) ^ (k − i) * real (k choose i) * real i ^ m) =
                real (Stirling m k) * fact k
      by (subst Stirling-closed-form) (simp-all add: field-simps)
    finally have ∗: (1 − fps-exp 1 :: real fps) ^ k $ m * fact n / fact (n − m) =
                      (− 1) ^ k * fact k * real (Stirling m k) * real (n choose m)
      using 1 by (simp add: binomial-fact del: of-nat-Suc)
    show ?case using 1 by (subst ∗) simp
  qed
  also have ... = (∑ m≤n. ∑ k≤n. (− 1) ^ k * fact k *
                    real (Stirling m k) * real (n choose m) * f k)
    by (rule sum.cong[OF refl], rule sum.mono-neutral-left) auto
  also have ... = (∑ k≤n. ∑ m≤n. (− 1) ^ k * fact k *
                    real (Stirling m k) * real (n choose m) * f k)
    by (rule sum.swap)
  also have ... = gen-akiyama-tanigawa f n 0
    by (simp add: gen-akiyama-tanigawa-n-0 sum-Stirling-binomial sum-distrib-left
sum-distrib-right
          mult.assoc atLeast0AtMost del: Stirling.simps)
  finally show Abs-fps (λn. gen-akiyama-tanigawa f n 0 / fact n) $ n =
                 (fps-exp 1 * (Abs-fps f oo 1 − fps-exp 1)) $ n
    by (subst (asm) fps-mult-left-const-nth) (simp add: field-simps del: of-nat-Suc)
qed
```

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 −
define *f* **where** *f* = (λ*n*. *1* / *real* (*Suc n*))
note *gen-akiyama-tanigawa-fps*[*of f*]
also {
have *fps-ln 1* = *fps-X* ∗ *Abs-fps* (λ*n*. (−*1*) ^ *n* / *real* (*Suc n*))
by (*intro fps-ext*) (*simp del*: *of-nat-Suc add*: *fps-ln-def*)
hence *fps-ln 1* / *fps-X* = *Abs-fps* (λ*n*. (−*1*) ^ *n* / *real* (*Suc n*))
by (*metis fps-X-neq-zero nonzero-mult-div-cancel-left*)
also have *fps-compose* ... (−*fps-X*) = *Abs-fps f*
by (*simp add*: *fps-compose-uminus' fps-eq-iff f-def*)
finally have *Abs-fps f* = *fps-compose* (*fps-ln 1* / *fps-X*) (−*fps-X*) **..**
also have *fps-ln 1* / *fps-X oo* − *fps-X oo 1* − *fps-exp* (*1*::*real*) = *fps-ln 1* / *fps-X oo fps-exp 1* − *1*
by (*subst fps-compose-assoc* [*symmetric*])
(*simp-all add*: *fps-compose-uminus*)
also have ... = (*fps-ln 1 oo fps-exp 1* − *1*) / (*fps-exp 1* − *1*)
by (*subst fps-compose-divide-distrib*) *auto*
also have ... = *fps-X* / (*fps-exp 1* − *1*) **by** (*simp add*: *fps-ln-fps-exp-inv fps-inv-fps-exp-compose*)
finally have *Abs-fps f oo 1* − *fps-exp 1* = *fps-X* / (*fps-exp 1* − *1*) **.**
}
also have *fps-exp* (*1*::*real*) − *1* = (*1* − *fps-exp* (−*1*)) ∗ *fps-exp 1*
by (*simp add*: *algebra-simps fps-exp-add-mult* [*symmetric*])
also have *fps-exp 1* ∗ (*fps-X* / ...) = *bernoulli'-fps* **unfolding** *bernoulli'-fps-def*
by (*subst dvd-div-mult2-eq*) (*auto simp*: *fps-dvd-iff intro*!: *subdegree-leI*)
finally have *Abs-fps* (λ*n*. *gen-akiyama-tanigawa f n 0* / *fact n*) = *bernoulli'-fps*
.
thus *?thesis* **by** (*simp add*: *fps-eq-iff akiyama-tanigawa-def f-def*)
qed

theorem *bernoulli-conv-akiyama-tanigawa*:
bernoulli n = *akiyama-tanigawa n 0* − (*if n* = *1 then 1 else 0*)
using *bernoulli'-conv-akiyama-tanigawa*[*of n*] **by** (*auto simp*: *bernoulli-conv-bernoulli'*)

end

end

3.4 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
  by (induction m xs rule: akiyama-tanigawa-step-aux.induct) simp-all

lemma akiyama-tanigawa-step-aux-eq-Nil-iff [simp]:
  akiyama-tanigawa-step-aux m xs = [] ⟷ length xs < 2
  by (subst length-0-conv [symmetric]) auto

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 (induction m xs arbitrary: n rule: akiyama-tanigawa-step-aux.induct)
  case (1 m x y xs n)
  thus ?case by (cases n) auto
qed auto

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
  by (simp add: gen-akiyama-tanigawa-row-def)

lemma gen-akiyama-tanigawa-row-eq-Nil-iff [simp]:
  gen-akiyama-tanigawa-row f n l u = [] ⟷ l ≥ u
  by (auto simp add: gen-akiyama-tanigawa-row-def)

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)
  by (simp add: gen-akiyama-tanigawa-row-def add-ac)

lemma gen-akiyama-tanigawa-row-0 [code]:
  gen-akiyama-tanigawa-row f 0 l u = map f [l..<u]
  by (simp add: gen-akiyama-tanigawa-row-def)

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))
 by (rule nth-equalityI) (auto simp: nth-gen-akiyama-tanigawa-row nth-akiyama-tanigawa-step-aux)

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))
  by (simp only: numeral-eq-Suc gen-akiyama-tanigawa-row-Suc)
```

lemma *gen-akiyama-tanigawa-code* [*code*]:
gen-akiyama-tanigawa f n k = hd (*gen-akiyama-tanigawa-row f n k* (*Suc k*))
by (*subst hd-conv-nth*) (*auto simp*: *nth-gen-akiyama-tanigawa-row length-0-conv* [*symmetric*])

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*
by (*simp add*: *akiyama-tanigawa-row-def*)

lemma *akiyama-tanigawa-row-eq-Nil-iff* [*simp*]:
akiyama-tanigawa-row n l u = [] ⟷ *l* ≥ *u*
by (*auto simp add*: *akiyama-tanigawa-row-def*)

lemma *nth-akiyama-tanigawa-row*:
i < *u* − *l* ⟹ *akiyama-tanigawa-row n l u* ! *i* = *akiyama-tanigawa n* (*i* + *l*)
by (*simp add*: *akiyama-tanigawa-row-def add-ac*)

lemma *akiyama-tanigawa-row-0* [*code*]:
akiyama-tanigawa-row 0 l u = map (λ*n*. *inverse* (*real* (*Suc n*))) [*l*..<*u*]
by (*simp add*: *akiyama-tanigawa-row-def akiyama-tanigawa-def divide-simps*)

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*))
by (*rule nth-equalityI*) (*auto simp*: *nth-akiyama-tanigawa-row*
nth-akiyama-tanigawa-step-aux akiyama-tanigawa-def)

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*))
by (*simp only*: *numeral-eq-Suc akiyama-tanigawa-row-Suc*)

lemma *akiyama-tanigawa-code* [*code*]:
akiyama-tanigawa n k = hd (*akiyama-tanigawa-row n k* (*Suc k*))
by (*subst hd-conv-nth*) (*auto simp*: *nth-akiyama-tanigawa-row length-0-conv* [*symmetric*])

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 (*cases n = 0* ∨ *n = 1* ∨ *odd n*)

case *False*
thus *?thesis* **by** (*auto simp add*: *bernoulli-conv-akiyama-tanigawa*)
qed (*auto simp*: *bernoulli-odd-eq-0*)

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*)
by (*simp add*: *bernoulli'-def bernoulli-code*)

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*)
by (*simp-all add*: *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*)
by (*simp-all add*: *bernoulli'-def*)

end

4 Bernoulli numbers and the zeta function at positive integers

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

lemma *joinpaths-cong*: $f = f' \Longrightarrow g = g' \Longrightarrow f +++ g = f' +++ g'$
by *simp*

lemma *linepath-cong*: $a = a' \Longrightarrow b = b' \Longrightarrow$ *linepath a b* = *linepath a' b'*
by *simp*

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* $(\lambda z.\ 1\ /\ (z\ \hat{}\ m * (exp\ z - 1)))\ (2 * pi * real\text{-}of\text{-}int\ n * \mathrm{i}) =$
$1\ /\ (2 * pi * real\text{-}of\text{-}int\ n * \mathrm{i})\ \hat{}\ m$
proof −
have *residue* $(\lambda z.\ (1\ /\ z\ \hat{}\ m)\ /\ (exp\ z - 1))\ (2 * pi * real\text{-}of\text{-}int\ n * \mathrm{i}) =$
$1\ /\ (2 * pi * real\text{-}of\text{-}int\ n * \mathrm{i})\ \hat{}\ m\ /\ 1$
using *exp-integer-2pi*[*of real-of-int n*] **and** *assms*
by (*rule-tac residue-simple-pole-deriv*[**where** *s*=−{*0*}])
(*auto intro*!: *holomorphic-intros derivative-eq-intros connected-open-delete-finite*
simp add: *mult-ac connected-punctured-universe*)
thus *?thesis* **by** (*simp add*: *divide-simps*)
qed

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 -
  define n where n = 2 * n'
  from n' have n: n ≥ 2 even n by (auto simp: n-def)
  define zeta :: complex where zeta = (∑ k. 1 / of-nat (Suc k) ^ n)

  have summable (λk. 1 / of-nat (Suc k) ^ n :: complex)
    using inverse-power-summable[of n] n
    by (subst summable-Suc-iff) (simp add: divide-simps)
  hence (λk. ∑ i<k. 1 / of-nat (Suc i) ^ n) ⟶ zeta
    by (subst (asm) summable-sums-iff) (simp add: sums-def zeta-def)
  also have (λk. ∑ i<k. 1 / of-nat (Suc i) ^ n) = (λk. ∑ i∈{0<..k}. 1 / of-nat
i ^ n)
    by (intro ext sum.reindex-bij-witness[of - λn. n - 1 Suc]) auto
  finally have zeta-limit: (λk. ∑ i∈{0<..k}. 1 / of-nat i ^ n) ⟶ zeta .

  — This is the exponential generating function of the Bernoulli numbers.
  define f where f = (λz::complex. if z = 0 then 1 else z / (exp z - 1))

  — We will integrate over this function, since its residue at the origin is the n-th
coefficient of f. Note that it has singularities at all points 2ikπ for k ∈ ℤ.
  define g where g = (λz::complex. 1 / (z ^ n * (exp z - 1)))

  — We integrate along a rectangle of width 2m and height 2(2m + 1)π with its
centre at the origin. The benefit of the rectangular path is that it is easier to bound
the value of the exponential appearing in the integrand. The horizontal lines of the
rectangle are always right in the middle between two adjacent singularities.
```

define γ :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *complex*
where γ = (λ*m*. *rectpath* (−*real m* − *real* (*2*∗*m*+*1*)∗*pi*∗i) (*real m* + *real* (*2*∗*m*+*1*)∗*pi*∗i))

— This set is a convex open enclosing set the contains our path.
define *A* **where** *A* = (λ*m*::*nat*. *box* (−(*real m*+*1*) − (*2*∗*m*+*2*)∗*pi*∗i) (*real m*+*1* + (*2*∗*m*+*2*)∗*pi*∗i))

— These are all the singularities in the enclosing inside the path (and also inside *A*).
define *S* **where** *S* = (λ*m*::*nat*. (λ*n*. *2* ∗ *pi* ∗ *of-int n* ∗ i) ‘ {−*m*..*m*})

— Any singularity in *A* is of the form $2ki\pi$ where $|k| \leq m$.
have *int-bound*: *k* ∈ {−*int m*..*int m*} **if** *2* ∗ *pi* ∗ *k* ∗ i ∈ *A m* **for** *k m*
proof −
from *that* **have** (−*real* (*Suc m*)) ∗ (*2* ∗ *pi*) < *real-of-int k* ∗ (*2* ∗ *pi*) ∧
real (*Suc m*) ∗ (*2* ∗ *pi*) > *real-of-int k* ∗ (*2* ∗ *pi*)
by (*auto simp*: *A-def in-box-complex-iff algebra-simps*)
hence −*real* (*Suc m*) < *real-of-int k* ∧ *real-of-int k* < *real* (*Suc m*)
by *simp*
also have −*real* (*Suc m*) = *real-of-int* (−*int* (*Suc m*)) **by** *simp*
also have *real* (*Suc m*) = *real-of-int* (*int* (*Suc m*)) **by** *simp*
also have *real-of-int* (− *int* (*Suc m*)) < *real-of-int k* ∧
real-of-int k < *real-of-int* (*int* (*Suc m*)) ⟷ *k* ∈ {−*int m*..*int m*}
by (*subst of-int-less-iff*) *auto*
finally show *k* ∈ {−*int m*..*int m*} **.**
qed

have *zeros*: ∃*k*∈{−*int m*..*int m*}. *z* = *2* ∗ *pi* ∗ *of-int k* ∗ i **if** *z* ∈ *A m exp z* = *1* **for** *z m*
proof −
from *that*(*2*) **obtain** *k* **where** *z-eq*: *z* = *2* ∗ *pi* ∗ *of-int k* ∗ i
unfolding *exp-eq-1* **by** (*auto simp*: *complex-eq-iff*)
with *int-bound*[*of k*] **and** *that*(*1*) **show** *?thesis* **by** *auto*
qed
have *zeros′*: *z* ^ *n* ∗ (*exp z* − *1*) ≠ *0* **if** *z* ∈ *A m* − *S m* **for** *z m*
using *zeros*[*of z*] *that* **by** (*auto simp*: *S-def*)

— The singularities all lie strictly inside the integration path.
have *subset*: *S m* ⊆ *box* (−*real m* − *real*(*2*∗*m*+*1*)∗*pi*∗i) (*real m* + *real*(*2*∗*m*+*1*)∗*pi*∗i) **if** *m* > *0* **for** *m*
proof (*rule*, *goal-cases*)
case (*1 z*)
then obtain *k* :: *int* **where** *k*: *k* ∈ {−*int m*..*int m*} *z* = *2* ∗ *pi* ∗ *k* ∗ i
unfolding *S-def* **by** *blast*
have *2* ∗ *pi* ∗ −*m* + −*pi* < *2* ∗ *pi* ∗ *k* + *0*
using *k* **by** (*intro add-le-less-mono mult-left-mono*) *auto*
moreover have *2* ∗ *pi* ∗ *k* + *0* < *2* ∗ *pi* ∗ *m* + *pi*
using *k* **by** (*intro add-le-less-mono mult-left-mono*) *auto*

ultimately show *?case* **using** k ‹$m > 0$›
by (*auto simp*: *A-def in-box-complex-iff algebra-simps*)
qed
from n **and** *zeros'* **have** *holo*: g *holomorphic-on* $A\ m - S\ m$ **for** m
unfolding *g-def* **by** (*intro holomorphic-intros*) *auto*

— The integration path lies completely inside A and does not cross any singularities.
have *path-subset*: *path-image* $(\gamma\ m) \subseteq A\ m - S\ m$ **if** $m > 0$ **for** m
proof −
have *path-image* $(\gamma\ m) \subseteq$ *cbox* $(-\mathit{real}\ m - (2 * m + 1) * \mathit{pi} * \mathrm{i})\ (\mathit{real}\ m + (2 * m + 1) * \mathit{pi} * \mathrm{i})$
unfolding γ*-def* **by** (*rule path-image-rectpath-subset-cbox*) *auto*
also have $\ldots \subseteq A\ m$ **unfolding** *A-def*
by (*subst subset-box-complex*) *auto*
finally have *path-image* $(\gamma\ m) \subseteq A\ m$ **.**
moreover have *path-image* $(\gamma\ m) \cap S\ m = \{\}$
proof *safe*
fix z **assume** z: $z \in$ *path-image* $(\gamma\ m)$ $z \in S\ m$
from *this*(*2*) **obtain** $k :: \mathit{int}$ **where** k: $z = 2 * \mathit{pi} * k * \mathrm{i}$
by (*auto simp*: *S-def*)
hence [*simp*]: *Re* $z = 0$ **by** *simp*
from z(*1*) **have** $|\mathit{Im}\ z| =$ *of-int* $(2{*}m{+}1) * \mathit{pi}$
using ‹$m > 0$› **by** (*auto simp*: γ*-def path-image-rectpath*)
also have $|\mathit{Im}\ z| =$ *of-int* $(2 * |k|) * \mathit{pi}$
by (*simp add*: *k abs-mult*)
finally have $2 * |k| = 2 * m + 1$
by (*subst* (*asm*) *mult-cancel-right*, *subst* (*asm*) *of-int-eq-iff*) *simp*
hence *False* **by** *presburger*
thus $z \in \{\}$ **..**
qed
ultimately show *path-image* $(\gamma\ m) \subseteq A\ m - S\ m$ **by** *blast*
qed

— We now obtain a closed form for the Bernoulli numbers using the integral.
have *eq*: $(\sum x{\in}\{0{<}..m\}.\ 1\ /\ \mathit{of\text{-}nat}\ x\ \hat{}\ n) =$
contour-integral $(\gamma\ m)\ g * (2 * \mathit{pi} * \mathrm{i})\ \hat{}\ n\ /\ (4 * \mathit{pi} * \mathrm{i}) -$
complex-of-real $(\mathit{bernoulli}\ n\ /\ \mathit{fact}\ n) * (2 * \mathit{pi} * \mathrm{i})\ \hat{}\ n\ /\ 2$
if m: $m > 0$ **for** m
proof −
— We relate the formal power series of the Bernoulli numbers to the corresponding complex function.
have *subdegree* $(\mathit{fps\text{-}exp}\ 1 - 1 :: \mathit{complex}\ \mathit{fps}) = 1$
by (*intro subdegreeI*) *auto*
hence *expansion*: f *has-fps-expansion bernoulli-fps*
unfolding *f-def bernoulli-fps-def* **by** (*auto intro*!: *fps-expansion-intros*)

— We use the Residue Theorem to explicitly compute the integral.
have *contour-integral* $(\gamma\ m)\ g =$
$2 * \mathit{pi} * \mathrm{i} * (\sum z{\in}S\ m.\ \mathit{winding\text{-}number}\ (\gamma\ m)\ z * \mathit{residue}\ g\ z)$

proof (*rule Residue-theorem*)
have *cbox* (−*real m* − (*2* ∗ *m* + *1*) ∗ *pi* ∗ i) (*real m* + (*2* ∗ *m* + *1*) ∗ *pi* ∗ i) ⊆ *A m*
unfolding *A-def* **by** (*subst subset-box-complex*) *simp-all*
thus ∀ *z*. *z* ∉ *A m* ⟶ *winding-number* (γ *m*) *z* = *0* **unfolding** γ*-def*
by (*intro winding-number-rectpath-outside allI impI*) *auto*
qed (*insert holo path-subset m, auto simp*: γ*-def A-def S-def intro*: *convex-connected*)
— Clearly, all the winding numbers are 1
also have *winding-number* (γ *m*) *z* = *1* **if** *z* ∈ *S m* **for** *z*
unfolding γ*-def* **using** *subset*[*of m*] *that m* **by** (*subst winding-number-rectpath*) *blast*+
hence (∑ *z*∈*S m*. *winding-number* (γ *m*) *z* ∗ *residue g z*) = (∑ *z*∈*S m*. *residue g z*)
by (*intro sum.cong*) *simp-all*
also have ... = (∑ *k*=−*int m*..*int m*. *residue g* (*2* ∗ *pi* ∗ *of-int k* ∗ i))
unfolding *S-def* **by** (*subst sum.reindex*) (*auto simp*: *inj-on-def o-def*)
also have {−*int m*..*int m*} = *insert 0* ({−*int m*..*int m*}−{*0*})
by *auto*
also have (∑ *k*∈.... *residue g* (*2* ∗ *pi* ∗ *of-int k* ∗ i)) =
residue g 0 + (∑ *k*∈{−*int m*..*m*}−{*0*}. *residue g* (*2* ∗ *pi* ∗ *of-int k* ∗ i))
by (*subst sum.insert*) *auto*
— The residue at the origin is just the n-th coefficient of f.
also have *residue g 0* = *residue* (λ*z*. *f z* / *z* ^ *Suc n*) *0* **unfolding** *f-def g-def*
by (*intro residue-cong eventually-mono*[*OF eventually-at-ball*[*of 1*]]) *auto*
also have ... = *fps-nth bernoulli-fps n*
by (*rule residue-fps-expansion-over-power-at-0* [*OF expansion*])
also have ... = *of-real* (*bernoulli n* / *fact n*)
by *simp*
also have (∑ *k*∈{−*int m*..*m*}−{*0*}. *residue g* (*2* ∗ *pi* ∗ *of-int k* ∗ i)) =
(∑ *k*∈{−*int m*..*m*}−{*0*}. *1* / *of-int k* ^ *n*) / (*2* ∗ *pi* ∗ i) ^ *n*
proof (*subst sum-divide-distrib*, *intro refl sum.cong*, *goal-cases*)
case (*1 k*)
hence ∗: *residue g* (*2* ∗ *pi* ∗ *of-int k* ∗ i) = *1* / (*2* ∗ *complex-of-real pi* ∗ *of-int k* ∗ i) ^ *n*
unfolding *g-def* **by** (*subst residue-bernoulli*) *auto*
thus *?case* **using** *1* **by** (*subst* ∗) (*simp add*: *divide-simps power-mult-distrib*)
qed
also have (∑ *k*∈{−*int m*..*m*}−{*0*}. *1* / *of-int k* ^ *n*) =
(∑ (*a*,*b*)∈{*0*<..*m*}×{−*1*,*1*::*int*}. *1* / *of-int* (*int a*) ^ *n* :: *complex*) **using** *n*
by (*intro sum.reindex-bij-witness*[*of* - λ*k*. *snd k* ∗ *int* (*fst k*) λ*k*. (*nat* |*k*|,*sgn k*)])
(*auto split*: *if-splits simp*: *abs-if*)
also have ... = (∑ *x*∈{*0*<..*m*}. *2* / *of-nat x* ^ *n*)
using *n* **by** (*subst sum.Sigma* [*symmetric*]) *auto*
also have ... = (∑ *x*∈{*0*<..*m*}. *1* / *of-nat x* ^ *n*) ∗ *2*
by (*simp add*: *sum-distrib-right*)
finally show *?thesis*

```
    by (simp add: field-simps)
  qed

  — The ugly part: We have to prove a bound on the integral by splitting it into four integrals over lines and bounding each part separately.
  have eventually (λm. norm (contour-integral (γ m) g) ≤
        ((4 + 12 * pi) + 6 * pi / m) / real m ^ (n - 1)) sequentially
    using eventually-gt-at-top[of 1::nat]
  proof eventually-elim
    case (elim m)
    let ?c = (2*m+1) * pi * i
    define I where I = (λp1 p2. contour-integral (linepath p1 p2) g)
    define p1 p2 p3 p4 where p1 = -real m - ?c and p2 = real m - ?c
                         and p3 = real m + ?c and p4 = -real m + ?c
    have eq: γ m = linepath p1 p2 +++ linepath p2 p3 +++ linepath p3 p4 +++ linepath p4 p1
      (is γ m = ?γ') unfolding γ-def rectpath-def Let-def
      by (intro joinpaths-cong linepath-cong)
        (simp-all add: p1-def p2-def p3-def p4-def complex-eq-iff)
    have integrable: g contour-integrable-on γ m using elim
      by (intro contour-integrable-holomorphic-simple[OF holo - - path-subset])
        (auto simp: γ-def A-def S-def intro!: finite-imp-closed)
    have norm (contour-integral (γ m) g) = norm (I p1 p2 + I p2 p3 + I p3 p4 + I p4 p1)
      unfolding I-def by (insert integrable, unfold eq)
                         (subst contour-integral-join; (force simp: add-ac)?)+
    also have ... ≤ norm (I p1 p2) + norm (I p2 p3) + norm (I p3 p4) + norm (I p4 p1)
      by (intro norm-triangle-mono order.refl)

    also have norm (I p1 p2) ≤ 1 / real m ^ n * norm (p2 - p1) (is - ≤ ?B1 * -)
      unfolding I-def
    proof (intro contour-integral-bound-linepath)
      fix z assume z: z ∈ closed-segment p1 p2
      define a where a = Re z
      from z have z: z = a - (2*m+1) * pi * i
        by (subst (asm) closed-segment-same-Im)
          (auto simp: p1-def p2-def complex-eq-iff a-def)
      have real m * 1 ≤ (2*m+1) * pi
        using pi-ge-two by (intro mult-mono) auto
      also have (2*m+1) * pi = |Im z| by (simp add: z)
      also have |Im z| ≤ norm z by (rule abs-Im-le-cmod)
      finally have norm z ≥ m by simp
      moreover {
        have exp z - 1 = -of-real (exp a + 1) using exp-integer-2pi-plus1[of m]
          by (simp add: z exp-diff algebra-simps exp-of-real)
        also have norm ... ≥ 1
          unfolding norm-minus-cancel norm-of-real by simp
```

```
    finally have norm (exp z − 1) ≥ 1 .
  }
  ultimately have norm z ^ n * norm (exp z − 1) ≥ real m ^ n * 1
    by (intro mult-mono power-mono) auto
  thus norm (g z) ≤ 1 / real m ^ n using elim
      by (simp add: g-def divide-simps norm-divide norm-mult norm-power
mult-less-0-iff)
qed (insert integrable, auto simp: eq)
also have norm (p2 − p1) = 2 * m by (simp add: p2-def p1-def)

also have norm (I p3 p4) ≤ 1 / real m ^ n * norm (p4 − p3) (is - ≤ ?B3 *
-)
  unfolding I-def
proof (intro contour-integral-bound-linepath)
  fix z assume z: z ∈ closed-segment p3 p4
  define a where a = Re z
  from z have z: z = a + (2*m+1) * pi * i
    by (subst (asm) closed-segment-same-Im)
      (auto simp: p3-def p4-def complex-eq-iff a-def)
  have real m * 1 ≤ (2*m+1) * pi
    using pi-ge-two by (intro mult-mono) auto
  also have (2*m+1) * pi = |Im z| by (simp add: z)
  also have |Im z| ≤ norm z by (rule abs-Im-le-cmod)
  finally have norm z ≥ m by simp
  moreover {
    have exp z − 1 = −of-real (exp a + 1) using exp-integer-2pi-plus1[of m]
      by (simp add: z exp-add algebra-simps exp-of-real)
    also have norm ... ≥ 1
      unfolding norm-minus-cancel norm-of-real by simp
    finally have norm (exp z − 1) ≥ 1 .
  }
  ultimately have norm z ^ n * norm (exp z − 1) ≥ real m ^ n * 1
    by (intro mult-mono power-mono) auto
  thus norm (g z) ≤ 1 / real m ^ n using elim
      by (simp add: g-def divide-simps norm-divide norm-mult norm-power
mult-less-0-iff)
qed (insert integrable, auto simp: eq)
also have norm (p4 − p3) = 2 * m by (simp add: p4-def p3-def)

also have norm (I p2 p3) ≤ (1 / real m ^ n) * norm (p3 − p2) (is - ≤ ?B2
* -)
  unfolding I-def
proof (rule contour-integral-bound-linepath)
  fix z assume z: z ∈ closed-segment p2 p3
  define b where b = Im z
  from z have z: z = m + b * i
    by (subst (asm) closed-segment-same-Re)
      (auto simp: p2-def p3-def algebra-simps complex-eq-iff b-def)
  from elim have 2 ≤ 1 + real m by simp
```

also have ... $\leq$ *exp* (*real m*) **by** (*rule exp-ge-add-one-self*)
also have *exp* (*real m*) $-$ *1* = *norm* (*exp z*) $-$ *norm* (*1*::*complex*)
by (*simp add*: *z*)
also have ... $\leq$ *norm* (*exp z* $-$ *1*)
by (*rule norm-triangle-ineq2*)
finally have *norm* (*exp z* $-$ *1*) $\geq$ *1* **by** *simp*
moreover have *norm z* $\geq$ *m*
using *z* **and** *abs-Re-le-cmod*[*of z*] **by** *simp*
ultimately have *norm z* ^ *n* $*$ *norm* (*exp z* $-$ *1*) $\geq$ *real m* ^ *n* $*$ *1* **using** *elim*
by (*intro mult-mono power-mono*) (*auto simp*: *z*)
thus *norm* (*g z*) $\leq$ *1* / *real m* ^ *n* **using** *n* **and** *elim*
by (*simp add*: *g-def norm-mult norm-divide norm-power divide-simps mult-less-0-iff*)
qed (*insert integrable*, *auto simp*: *eq*)
also have *p3* $-$ *p2* = *of-real* (*2*$*$(*2*$*$*real m*+*1*)$*$*pi*) $*$ i **by** (*simp add*: *p2-def p3-def*)
also have *norm* ... = *2* $*$ (*2* $*$ *real m* + *1*) $*$ *pi*
unfolding *norm-mult norm-of-real* **by** *simp*

also have *norm* (*I p4 p1*) $\leq$ (*2* / *real m* ^ *n*) $*$ *norm* (*p1* $-$ *p4*) (**is** - $\leq$ *?B4* $*$ -)
unfolding *I-def*
proof (*rule contour-integral-bound-linepath*)
fix *z* **assume** *z*: *z* $\in$ *closed-segment p4 p1*
define *b* **where** *b* = *Im z*
from *z* **have** *z*: *z* = $-$*real m* + *b* $*$ i
by (*subst* (*asm*) *closed-segment-same-Re*)
(*auto simp*: *p1-def p4-def algebra-simps b-def complex-eq-iff*)
from *elim* **have** *2* $\leq$ *1* + *real m* **by** *simp*
also have ... $\leq$ *exp* (*real m*) **by** (*rule exp-ge-add-one-self*)
finally have *1* / *2* $\leq$ *1* $-$ *exp* ($-$*real m*)
by (*subst exp-minus*) (*simp add*: *field-simps*)
also have *1* $-$ *exp* ($-$*real m*) = *norm* (*1*::*complex*) $-$ *norm* (*exp z*)
by (*simp add*: *z*)
also have ... $\leq$ *norm* (*exp z* $-$ *1*)
by (*subst norm-minus-commute*, *rule norm-triangle-ineq2*)
finally have *norm* (*exp z* $-$ *1*) $\geq$ *1* / *2* **by** *simp*
moreover have *norm z* $\geq$ *m*
using *z* **and** *abs-Re-le-cmod*[*of z*] **by** *simp*
ultimately have *norm z* ^ *n* $*$ *norm* (*exp z* $-$ *1*) $\geq$ *real m* ^ *n* $*$ (*1* / *2*) **using** *elim*
by (*intro mult-mono power-mono*) (*auto simp*: *z*)
thus *norm* (*g z*) $\leq$ *2* / *real m* ^ *n* **using** *n* **and** *elim*
by (*simp add*: *g-def norm-mult norm-divide norm-power divide-simps mult-less-0-iff*)
qed (*insert integrable*, *auto simp*: *eq*)
also have *p1* $-$ *p4* = $-$*of-real* (*2*$*$(*2*$*$*real m*+*1*)$*$*pi*) $*$ i
by (*simp add*: *p1-def p4-def algebra-simps*)

also have *norm ... = 2 * (2 * real m + 1) * pi*
unfolding *norm-mult norm-of-real norm-minus-cancel* **by** *simp*

also have *?B1 * (2*m) + ?B2 * (2*(2*real m+1)*pi) + ?B3 * (2*m) + ?B4 * (2*(2*real m+1)*pi) =*
*(4 * m + 6 * (2 * m + 1) * pi) / real m ^ n*
by (*simp add*: *divide-simps*)
also have *(4 * m + 6 * (2 * m + 1) * pi) = (4 + 12 * pi) * m + 6 * pi*
by (*simp add*: *algebra-simps*)
also have *... / real m ^ n = ((4 + 12 * pi) + 6 * pi / m) / real m ^ (n − 1)*
using *n* **by** (*cases n*) (*simp-all add*: *divide-simps*)
finally show *cmod (contour-integral (γ m) g) ≤ ...* **by** *simp*
qed

— It is clear that this bound goes to 0 since *2 ≤ n*.
moreover have *(λm. (4 + 12 * pi + 6 * pi / real m) / real m ^ (n − 1)) ⟶ 0*
by (*rule real-tendsto-divide-at-top tendsto-add tendsto-const*
filterlim-real-sequentially filterlim-pow-at-top | use n **in** *simp*)+
ultimately have *∗: (λm. contour-integral (γ m) g) ⟶ 0*
by (*rule Lim-null-comparison*)

— Since the infinite sum over the residues can expressed using the zeta function, we have now related the Bernoulli numbers at even positive integers to the zeta function.

have *(λm. contour-integral (γ m) g * (2 * pi * i) ^ n / (4 * pi * i) − of-real (bernoulli n / fact n) * (2 * pi * i) ^ n / 2) ⟶ 0 * (2 * pi * i) ^ n / (4 * pi * i) − of-real (bernoulli n / fact n) * (2 * pi * i) ^ n / 2*
using *n* **by** (*intro tendsto-intros ∗ zeta-limit*) *auto*
also have *?this ⟷ (λm. ∑k∈{0<..m}. 1 / of-nat k ^ n) ⟶ − of-real (bernoulli n / fact n) * (2 * pi * i) ^ n / 2*
by (*intro filterlim-cong eventually-mono* [*OF eventually-gt-at-top*[*of 0*::*nat*]])
(*use eq* **in** *simp-all*)
finally have *(λm. ∑k∈{0<..m}. 1 / of-nat k ^ n) ⟶ − of-real (bernoulli n / fact n) * (of-real (2 * pi) * i) ^ n / 2*
(**is** - *⟶ ?L*) **.**
also have *(λm. ∑k∈{0<..m}. 1 / of-nat k ^ n) = (λm. ∑k∈{..<m}. 1 / of-nat (Suc k) ^ n)*
by (*intro ext sum.reindex-bij-witness*[*of - Suc λn. n − 1*]) *auto*
also have *... ⟶ ?L ⟷ (λk. 1 / of-nat (Suc k) ^ n) sums ?L*
by (*simp add*: *sums-def*)
also have *(2 * pi * i) ^ n = (2 * pi) ^ n * (−1) ^ n′*
by (*simp add*: *n-def divide-simps power-mult-distrib power-mult power-minus′*)
also have *− of-real (bernoulli n / fact n) * ... / 2 =*
*of-real ((−1) ^ Suc n′ * bernoulli (2*n′) * (2*pi) ^ (2*n′) / (2 * fact*

```
(2*n'')))
    by (simp add: n-def divide-simps)
  finally show ?thesis unfolding n-def .
qed

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 -
  have (λk. complex-of-real (?f k)) sums complex-of-real ?L
    using nat-even-power-sums-complex[OF assms] by simp
  thus ?thesis by (simp only: sums-of-real-iff)
qed
```

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 -
  have *: (λk. 1 / real (Suc k) ^ (2 * n)) sums
            ((- 1) ^ Suc n * bernoulli (2 * n) * (2 * pi) ^ (2 * n) / (2 * fact (2
* n)))
    using assms by (rule nat-even-power-sums-real)
  from * have 0 < (∑k. 1 / real (Suc k) ^ (2*n))
    by (intro suminf-pos) (auto simp: sums-iff)
  hence sgn (∑k. 1 / real (Suc k) ^ (2*n)) = 1
    by simp
  also have (∑k. 1 / real (Suc k) ^ (2*n)) =
              (- 1) ^ Suc n * bernoulli (2 * n) * (2 * pi) ^ (2 * n) / (2 * fact (2
* n))
    using * by (simp add: sums-iff)
  also have sgn ... = (-1) ^ Suc n * sgn (bernoulli (2 * n))
    by (simp add: sgn-mult)
  finally show ?thesis
    by (simp add: minus-one-power-iff split: if-splits)
qed

corollary bernoulli-even-nonzero: even n ⟹ bernoulli n ≠ 0
  using sgn-bernoulli-even[of n div 2] by (cases n = 0) (auto elim!: evenE)

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))
  using sgn-bernoulli-even [of n div 2] by (auto simp: bernoulli-odd-eq-0)

corollary bernoulli-zero-iff: bernoulli n = 0 ⟷ odd n ∧ n ≠ 1
  by (auto simp: bernoulli-even-nonzero bernoulli-odd-eq-0)

corollary bernoulli'-zero-iff: (bernoulli' n = 0) ⟷ (n ≠ 1 ∧ odd n)
  by (auto simp: bernoulli'-def bernoulli-zero-iff)

corollary bernoulli-pos-iff: bernoulli n > 0 ⟷ n = 0 ∨ n mod 4 = 2
proof −
  have bernoulli n > 0 ⟷ sgn (bernoulli n) = 1
    by (simp add: sgn-if)
  also have ... ⟷ n = 0 ∨ even n ∧ odd (n div 2)
    by (subst sgn-bernoulli) auto
  also have even n ∧ odd (n div 2) ⟷ n mod 4 = 2
    by presburger
  finally show ?thesis .
qed

corollary bernoulli-neg-iff: bernoulli n < 0 ⟷ n = 1 ∨ n > 0 ∧ 4 dvd n
proof −
  have bernoulli n < 0 ⟷ sgn (bernoulli n) = −1
    by (simp add: sgn-if)
  also have ... ⟷ n = 1 ∨ n > 0 ∧ even n ∧ even (n div 2)
    by (subst sgn-bernoulli) (auto simp: minus-one-power-iff)
  also have even n ∧ even (n div 2) ⟷ 4 dvd n
    by presburger
  finally show ?thesis .
qed
```

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: (λn. 1 / (n+1) ^ 2) sums (pi ^ 2 / 6)
  using nat-even-power-sums-real[of 1] by (simp add: fact-numeral)

corollary nat-power4-sums: (λn. 1 / (n+1) ^ 4) sums (pi ^ 4 / 90)
  using nat-even-power-sums-real[of 2] by (simp add: fact-numeral)

corollary nat-power6-sums: (λn. 1 / (n+1) ^ 6) sums (pi ^ 6 / 945)
  using nat-even-power-sums-real[of 3] by (simp add: fact-numeral)

corollary nat-power8-sums: (λn. 1 / (n+1) ^ 8) sums (pi ^ 8 / 9450)
  using nat-even-power-sums-real[of 4] by (simp add: fact-numeral)

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.