# Bernoulli Numbers

Lukas Bulwahn and Manuel Eberl

March 17, 2025

**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 2**
- 1.1 Preliminaries . . . 2
- 1.2 Bernoulli Numbers and Bernoulli Polynomials . . . 2
- 1.3 Basic Observations on Bernoulli Polynomials . . . 4
- 1.4 Sum of Powers with Bernoulli Polynomials . . . 6
- 1.5 Instances for Square And Cubic Numbers . . . 8

**2 Periodic Bernoulli polynomials 8**

**3 Connection of Bernoulli numbers to formal power series 14**
- 3.1 Preliminaries . . . 14
- 3.2 Generating function of Stirling numbers . . . 20
- 3.3 Generating function of Bernoulli numbers . . . 23
- 3.4 Von Staudt–Clausen Theorem . . . 27
- 3.5 Denominators of Bernoulli numbers . . . 31
- 3.6 Akiyama–Tanigawa algorithm . . . 36
- 3.7 Efficient code . . . 40

**4 Bernoulli numbers and the zeta function at positive integers 44**

# 1 Bernoulli numbers

**theory** *Bernoulli*
**imports** *Complex-Main*
**begin**

## 1.1 Preliminaries

**lemma** *power-numeral-reduce*: $a$ ^ *numeral* $n = a * a$ ^ *pred-numeral* $n$
  **by** (*simp only*: *numeral-eq-Suc power-Suc*)

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

**lemma** *of-nat-binomial-Suc*:
  **assumes** $k \leq n$
  **shows** (*of-nat* (*Suc* $n$ *choose* $k$) :: $'a$ :: *field-char-0*) =
    *of-nat* (*Suc* $n$) / *of-nat* (*Suc* $n - k$) $*$ *of-nat* ($n$ *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** $\bigwedge x.\ ((\lambda x.\ f\ x - g\ x)$ *has-real-derivative* $0)$ (*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*: $((\sum i \leq n::nat.\ f\ (i + 1) - f\ i)::'a::field) = f\ (n + 1) - f\ 0$
  **by** (*induct* $n$) (*auto simp add*: *field-simps*)

**lemma** *Rats-sum*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}) \Longrightarrow$ *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* $\Rightarrow$ *real*
**where**
  *bernoulli* $0 = (1::real)$
| *bernoulli* (*Suc* $n$) = $(-1\ /\ (n + 2)) * (\sum k \leq n.\ ((n + 2$ *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* **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* = (λ*x*. ∑ *k* ≤ *n*. *of-nat* (*n choose k*) ∗ *of-real* (*bernoulli k*) ∗ *x* ^ (*n* − *k*))

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

**fix** *x* :: *'a*
**have** *bernpoly n x* = ($\sum$ *k*≤*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*≤*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*≤*n*. *real* (*n* + *2 choose* (*k* + *2*)) ∗ *bernoulli* (*n* − *k*))
**proof** −
**have** *bernoulli* (*Suc n*) = − *1* / (*real n* + *2*) ∗ ($\sum$ *k*≤*n*. *real* (*n* + *2 choose k*) ∗ *bernoulli k*)
**unfolding** *bernoulli.simps* **..**
**also have** ($\sum$ *k*≤*n*. *real* (*n* + *2 choose k*) ∗ *bernoulli k*) =
($\sum$ *k*≤*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*≤*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*≤*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* ∈ {..*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*
($\sum$ *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 sum-distrib-left*
**by** (*force simp*: *of-nat-binomial-Suc nat-le-iff-add intro*: *sum.cong*)
**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*:
($\sum$ *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'*:
($\sum$ *k*<*n*. *real* (*n choose k*) ∗ *bernoulli k*) = (*if n* = *1 then 1 else 0*)
**proof** (*cases n*)
**case** (*Suc m*)
**have** ($\sum$ *k*<*n*. *real* (*n choose k*) ∗ *bernoulli k*) =
($\sum$ *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*:
  $(\sum k \le n{::}nat.\ f\ k) = (if\ n = 0\ then\ f\ 0\ else\ f\ n + (\sum k \le n - 1.\ f\ k))$
  **by** (*cases n*) (*simp-all add*: *add-ac*)

**lemma** *bernoulli-unroll*:
  $n > 0 \Longrightarrow bernoulli\ n = -\ 1\ /\ (real\ n + 1) * (\sum k \le n - 1.\ real\ (n + 1\ choose\ k) * bernoulli\ k)$
**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*) =
        $(\sum k \le n.\ of\text{-}real\ (real\ (Suc\ n\ choose\ k) * bernoulli\ k))$
    **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*

$\hat{}$ *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** $n \neq 1$
**shows** *bernpoly n 1* = *of-real* (*bernoulli n*)
**proof** −
**have** *bernpoly n 1* = *bernoulli n*
**proof** (*cases* $n \geq 2$)
**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*:
($\sum k \leq n$::*nat*. (*real k*) $\hat{}$ *m*) = (*bernpoly* (*Suc m*) (*n* + *1*) − *bernpoly* (*Suc m*) *0*) / (*m* + *1*)
**proof** −
**from** *diff-bernpoly*[*of Suc m*, *simplified*] **have** (*m* + (*1*::*real*)) ∗ ($\sum k \leq n$. (*real k*) $\hat{}$ *m*) = ($\sum k \leq n$. *bernpoly* (*Suc m*) (*real k* + *1*) − *bernpoly* (*Suc m*) (*real k*))
**by** (*auto simp add*: *sum-distrib-left intro*!: *sum.cong*)
**also have** ... = ($\sum k \leq n$. *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*=$\lambda k$. *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* ($\sum k{\le}n{::}nat.\ k$ ^ *2*) = *real* (*2* ∗ *n* ^ *3* + *3* ∗ *n* ^ *2* + *n*) / *6*
**unfolding** *of-nat-sum of-nat-power sum-of-powers*
**by** (*simp add*: *bernoulli-unroll-all field-simps power2-eq-square power-numeral-reduce*)

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

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

**corollary** *sum-of-cubes-nat*: ($\sum k{\le}n{::}nat.\ k$ ^ *3*) = (*n* ^ *2* + *n*) ^ *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* ⇒ *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** $(a - x) / p \le n$ $n < (a - x) / p + 1$ **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** $*$: $f$ $-x+n*p\rightarrow$ $f$ $(x+n*p)$ **by** (*simp add*: *isCont-def f.periodic-simps*)
  **have** $(\lambda x.\ f\ (x + n*p))$ $-x\rightarrow$ $f$ $(x+n*p)$
    **by** (*intro tendsto-compose*[*OF* $*$] *tendsto-intros*)
  **thus** $f$ $-x\rightarrow$ $f\ x$ **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* ($s \cup \{x\}$)) =
(*g has-field-derivative v*) (*at y within* ($s \cup \{x\}$)) **using** *assms*
**by** (*intro has-field-derivative-cong-ev*) (*auto elim*!: *eventually-mono*)
**also from** *assms* **have** *at x within* ($s \cup \{x\}$) = *at x within s* **by** (*simp add*: *at-within-def*)
**also from** *assms* **have** *at y within* ($s \cup \{x\}$) = *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** (*blast intro*: *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** (*blast intro*: *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*) $\notin \mathbb{Z}$
  **shows** (*frac has-field-derivative 1*) (*at x*)
**proof** −
  **have** (($\lambda t.$ *t* − *of-int* $\lfloor x \rfloor$) *has-field-derivative 1*) (*at x*)
    **by** (*auto intro*!: *derivative-eq-intros*)
  **also have** *?this* $\longleftrightarrow$ *?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* ($\lambda x.$ *f* (*frac x*))
**proof** (*rule periodic-continuous-onI*)
  **have** *cont*: *continuous-on* {*0..1*} ($\lambda x.$ *f* (*frac x*))
    **unfolding** *continuous-on-def*
  **proof** *safe*
    **fix** *x* :: *real* **assume** *x*: *x* ∈ {*0..1*}
    **show** (($\lambda x.$ *f* (*frac x*)) $\longrightarrow$ *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* ($\lambda x.$ *x* ∈ {..<*1*}) (*nhds x*)
        **by** (*intro eventually-nhds-in-open*) *auto*
      **hence** *eventually* ($\lambda x.$ *frac x* = *x*) (*at x within* {*0..1*})
        **by** (*auto simp*: *eventually-at-filter frac-eq elim*!: *eventually-mono*)
      **hence** *eventually* ($\lambda x.$ *f x* = *f* (*frac x*)) (*at x within* {*0..1*})
        **by** *eventually-elim simp*
      **moreover from** *cont1 x* **have** (*f* $\longrightarrow$ *f* (*frac x*)) (*at x within* {*0..1*})
        **by** (*simp add*: *continuous-on-def*)
      **ultimately show** (($\lambda x.$ *f* (*frac x*)) $\longrightarrow$ *f* (*frac x*)) (*at x within* {*0..1*})
        **by** (*blast intro*: *Lim-transform-eventually*)
    **next**
      **case** *True*
        **from** *cont1* **have** ∗∗: (*f* $\longrightarrow$ *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* ($\lambda x.$ *frac* $x = x$) (*at 1 within* $\{0..1\}$)
**by** (*auto simp*: *eventually-at-filter frac-eq*)
**qed** (*simp add*: *filterlim-ident*)
**ultimately have** (($\lambda x.$ *f* (*frac* $x$)) $\longrightarrow$ *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* $\{0..0+1\}$ ($\lambda x.$ *f* (*frac* $x$)) **by** *simp*
**qed** (*simp-all add*: *frac.periodic-simps*)

**definition** *pbernpoly* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *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*: $x \in \{0..<1\} \Longrightarrow$ *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** $n \neq 1$
**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*: $n \geq 2$ **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** $n \neq 1$ *continuous-on A f*
**shows** *continuous-on A* ($\lambda x.$ *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*]: $n \neq 1 \Longrightarrow$ *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** $(pbernpoly\ (Suc\ n)\ has\text{-}field\text{-}derivative\ real\ (Suc\ n) * pbernpoly\ n\ x)\ (at\ x)$
**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*: $x = of\text{-}int\ k$ **by** (*auto elim*: *Ints-cases*)
**have** $(pbernpoly\ (Suc\ n)\ has\text{-}field\text{-}derivative\ real\ (Suc\ n) * pbernpoly\ n\ x)$
$(at\ x\ within\ (\{..<x\} \cup \{x<..\}))$
**proof** (*rule has-field-derivative-at-within-union*)
**have** $((\lambda x.\ bernpoly\ (Suc\ n)\ (x - of\text{-}int\ (k-1)))\ has\text{-}field\text{-}derivative$
$real\ (Suc\ n) * bernpoly\ n\ (x - of\text{-}int\ (k-1)))\ (at\text{-}left\ x)$
**by** (*auto intro*!: *derivative-eq-intros*)
**also have** $?this \longleftrightarrow (pbernpoly\ (Suc\ n)\ has\text{-}field\text{-}derivative$
$real\ (Suc\ n) * pbernpoly\ n\ x)\ (at\text{-}left\ x)$ **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** $\forall_F\ t\ in\ nhds\ x.\ t \in \{..<x\} \longrightarrow bernpoly\ (Suc\ n)\ (t - real\text{-}of\text{-}int\ (k - 1)) =$
$pbernpoly\ (Suc\ n)\ t$
**proof** (*elim eventually-mono*, *safe*)
**fix** *t* **assume** $t < x\ t \in \{x-1<..<x+1\}$
**hence** $frac\ t = t - real\text{-}of\text{-}int\ (k - 1)$ **using** *k*
**by** (*subst frac-unique-iff*) *auto*
**thus** $bernpoly\ (Suc\ n)\ (t - real\text{-}of\text{-}int\ (k - 1)) = pbernpoly\ (Suc\ n)\ t$
**by** (*simp add*: *pbernpoly-def*)
**qed**
**qed** (*insert k*, *auto simp*: *pbernpoly-def bernpoly-1*)
**finally show** $(pbernpoly\ (Suc\ n)\ has\text{-}real\text{-}derivative$
$real\ (Suc\ n) * pbernpoly\ n\ x)\ (at\text{-}left\ x)$ **.**
**next**
**have** $((\lambda x.\ bernpoly\ (Suc\ n)\ (x - of\text{-}int\ k))\ has\text{-}field\text{-}derivative$
$real\ (Suc\ n) * bernpoly\ n\ (x - of\text{-}int\ k))\ (at\text{-}right\ x)$
**by** (*auto intro*!: *derivative-eq-intros*)
**also have** $?this \longleftrightarrow (pbernpoly\ (Suc\ n)\ has\text{-}field\text{-}derivative$
$real\ (Suc\ n) * pbernpoly\ n\ x)\ (at\text{-}right\ x)$ **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** $\forall_F\ t\ in\ nhds\ x.\ t \in \{x<..\} \longrightarrow bernpoly\ (Suc\ n)\ (t - real\text{-}of\text{-}int\ k) =$
$pbernpoly\ (Suc\ n)\ t$
**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** $\{..<x\} \cup \{x<..\} = UNIV - \{x\}$ **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** $\bigwedge x.$ *norm* (*pbernpoly n x*) $\leq c$
**proof** −
**have** $\exists x \in \{0..1\}. \forall y \in \{0..1\}.$ *norm* (*bernpoly n y* :: *real*) $\leq$ *norm* (*bernpoly n x*
:: *real*)
**by** (*intro continuous-attains-sup*) (*auto intro*!: *continuous-intros*)
**then obtain** *x* **where** *x*:
$\bigwedge y.\ y \in \{0..1\} \Longrightarrow$ *norm* (*bernpoly n y* :: *real*) $\leq$ *norm* (*bernpoly n x* :: *real*)
**by** *blast*
**have** *norm* (*pbernpoly n y*) $\leq$ *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.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*)
  **by** (*simp add*: *prime-multiplicity-other*)

**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** (*cases y* = *0*)
  **case** *False*
  **have** *nz*: *f x* $\neq$ *0* **if** *x* $\in$ *A* **for** *x*
    **using** *assms*(*2*)[*of x*] *that* **by** *auto*
  **have** *prod f A* $\neq$ *0*
    **using** *assms nz* **by** (*subst prod-zero-iff*) *auto*
  **thus** *?thesis*
  **proof** (*rule multiplicity-le-imp-dvd*)
    **fix** *p* :: *'a* **assume** *prime p*
    **show** *multiplicity p* (*prod f A*) $\leq$ *multiplicity p y*
    **proof** (*cases p dvd prod f A*)
      **case** *True*
      **then obtain** *x* **where** *x*: *x* $\in$ *A* **and** *p dvd f x*
        **using** ‹*prime p*› *assms* **by** (*subst* (*asm*) *prime-dvd-prod-iff*) *auto*
      **have** *multiplicity p* (*prod f A*) = ($\sum x \in A$. *multiplicity p* (*f x*))
        **using** *assms* ‹*prime p*› *nz* **by** (*intro prime-elem-multiplicity-prod-distrib*)
*auto*
      **also have** ... = ($\sum x \in \{x\}$. *1* :: *nat*)
        **using** *assms* ‹*prime p*› ‹*p dvd f x*› *primes-dvd-imp-eq x*
        **by** (*intro Groups-Big.sum.mono-neutral-cong-right*)
          (*auto simp*: *multiplicity-prime-prime inj-on-def*)
      **finally have** *multiplicity p* (*prod f A*) = *1* **by** *simp*
      **also have** *1* $\leq$ *multiplicity p y*
        **using** *assms nz* ‹*prime p*› ‹*y* $\neq$ *0*› *x* ‹*p dvd f x*›
        **by** (*intro multiplicity-geI*) *force+*
      **finally show** *?thesis* **.**
    **qed** (*auto simp*: *not-dvd-imp-multiplicity-0*)
  **qed**
**qed** *auto*

**end**

**context** *semiring-gcd*
**begin**

**lemma** *gcd-add-dvd-right1*: $a$ *dvd* $b \Longrightarrow$ *gcd* $a$ $(b + c) =$ *gcd* $a$ $c$
**by** (*elim dvdE*) (*simp add*: *gcd-add-mult mult.commute*[*of* $a$])

**lemma** *gcd-add-dvd-right2*: $a$ *dvd* $c \Longrightarrow$ *gcd* $a$ $(b + c) =$ *gcd* $a$ $b$
**using** *gcd-add-dvd-right1*[*of* $a$ $c$ $b$] **by** (*simp add*: *add-ac*)

**lemma** *gcd-add-dvd-left1*: $a$ *dvd* $b \Longrightarrow$ *gcd* $(b + c)$ $a =$ *gcd* $c$ $a$
**using** *gcd-add-dvd-right1*[*of* $a$ $b$ $c$] **by** (*simp add*: *gcd.commute*)

**lemma** *gcd-add-dvd-left2*: $a$ *dvd* $c \Longrightarrow$ *gcd* $(b + c)$ $a =$ *gcd* $b$ $a$
**using** *gcd-add-dvd-right2*[*of* $a$ $c$ $b$] **by** (*simp add*: *gcd.commute*)

**end**

**context** *ring-gcd*
**begin**

**lemma** *gcd-diff-dvd-right1*: $a$ *dvd* $b \Longrightarrow$ *gcd* $a$ $(b - c) =$ *gcd* $a$ $c$
**using** *gcd-add-dvd-right1*[*of* $a$ $b$ $-c$] **by** *simp*

**lemma** *gcd-diff-dvd-right2*: $a$ *dvd* $c \Longrightarrow$ *gcd* $a$ $(b - c) =$ *gcd* $a$ $b$
**using** *gcd-add-dvd-right2*[*of* $a$ $-c$ $b$] **by** *simp*

**lemma** *gcd-diff-dvd-left1*: $a$ *dvd* $b \Longrightarrow$ *gcd* $(b - c)$ $a =$ *gcd* $c$ $a$
**using** *gcd-add-dvd-left1*[*of* $a$ $b$ $-c$] **by** *simp*

**lemma** *gcd-diff-dvd-left2*: $a$ *dvd* $c \Longrightarrow$ *gcd* $(b - c)$ $a =$ *gcd* $b$ $a$
**using** *gcd-add-dvd-left2*[*of* $a$ $-c$ $b$] **by** *simp*

**end**

**lemma** *cong-int*: $[a = b]$ (*mod* $m$) $\Longrightarrow$ [*int* $a =$ *int* $b$] (*mod* $m$)
**by** (*simp add*: *cong-int-iff*)

**lemma** *Rats-int-div-natE*:
**assumes** $(x :: 'a ::$ *field-char-0*$) \in \mathbb{Q}$
**obtains** $m :: int$ **and** $n :: nat$ **where** $n > 0$ **and** $x =$ *of-int* $m$ / *of-nat* $n$ **and** *coprime* $m$ $n$
**proof** −
**from** *assms* **obtain** $r$ **where** [*simp*]: $x =$ *of-rat* $r$
**by** (*auto simp*: *Rats-def*)
**obtain** $a$ $b$ **where** [*simp*]: $r =$ *Rat.Fract* $a$ $b$ **and** *ab*: $b > 0$ *coprime* $a$ $b$
**by** (*cases* $r$)
**from** *ab* **show** *?thesis*
**by** (*intro that*[*of nat* $b$ $a$]) (*auto simp*: *of-rat-rat*)
**qed**

**lemma** *sum-in-Ints*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Z}) \Longrightarrow$ *sum* $f$ $A \in \mathbb{Z}$
**by** (*induction* $A$ *rule*: *infinite-finite-induct*) *auto*

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

**lemma** *product-dvd-fact*:
  **assumes** $a > 1$ $b > 1$ $a = b \longrightarrow a > 2$
  **shows** $(a * b)$ *dvd fact* $(a * b - 1)$
**proof** (*cases* $a = b$)
  **case** *False*
  **have** $a * 1 < a * b$ **and** $1 * b < a * b$
    **using** *assms* **by** (*intro mult-strict-left-mono mult-strict-right-mono*; *simp*)+
  **hence** *ineqs*: $a \le a * b - 1$ $b \le a * b - 1$
    **by** *linarith*+
  **from** *False* **have** $a * b = \prod\{a,b\}$ **by** *simp*
  **also have** $\ldots$ *dvd* $\prod\{1..a * b - 1\}$
    **using** *assms ineqs* **by** (*intro prod-dvd-prod-subset*) *auto*
  **finally show** *?thesis* **by** (*simp add*: *fact-prod*)
**next**
  **case** [*simp*]: *True*
  **from** *assms* **have** $a > 2$ **by** *auto*
  **hence** $a * 2 < a * b$ **using** *assms* **by** (*intro mult-strict-left-mono*; *simp*)
  **hence** $*$: $2 * a \le a * b - 1$ **by** *linarith*
  **have** $a * a$ *dvd* $(2 * a) * a$ **by** *simp*
  **also have** $\ldots = \prod\{2{*}a,\ a\}$ **using** *assms* **by** *auto*
  **also have** $\ldots$ *dvd* $\prod\{1..a * b - 1\}$
    **using** *assms* $*$ **by** (*intro prod-dvd-prod-subset*) *auto*
  **finally show** *?thesis* **by** (*simp add*: *fact-prod*)
**qed**

**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** $-$
  **from** *assms* **have** $\neg$*irreducible* $m$
    **by** (*simp flip*: *prime-elem-iff-irreducible* )
  **then obtain** $a$ **where** $a$: $a$ *dvd* $m$ $\neg m$ *dvd* $a$ $a \neq 1$
    **using** *assms* **by** (*auto simp*: *irreducible-altdef*)
  **then obtain** $b$ **where** [*simp*]: $m = a * b$
    **by** *auto*
  **from** $a$ *assms* **have** $a \neq 0$ $b \neq 0$ $b \neq 1$
    **by** (*auto intro*!: *Nat.gr0I*)
  **with** $a$ **have** $a > 1$ $b > 1$ **by** *linarith*+
  **moreover from** *this* **and** $a$ **have** $a < m$ $b < m$
    **by** *auto*
  **ultimately show** *?thesis* **using** ‹$m = a * b$›
    **by** *blast*
**qed**

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** −
  **have** $(\sum x{\in}A.\ (\prod y{\in}A.\ f\ y)\ /\ f\ x) = (\sum x{\in}A.\ \prod y{\in}A{-}\{x\}.\ f\ y)$
    **using** *prod.remove*[*of A - f*] *assms* **by** (*intro sum.cong refl*) (*auto simp*: *field-simps*)
  **thus** *?thesis*
    **using** *assms* **by** (*simp add*: *field-simps sum-distrib-right sum-distrib-left*)
**qed**

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** (*intro prod-coprime-right*)
  **fix** *x* **assume** *x*: $x \in A$
  **have** $a = (\prod y{\in}A{-}\{x\}.\ f\ y) + (\sum y{\in}A{-}\{x\}.\ \prod z{\in}A{-}\{y\}.\ f\ z)$
    **unfolding** *a-def* **using** ‹*finite A*› **and** *x* **by** (*rule sum.remove*)
  **also have** *gcd* … $(f\ x) = gcd\ (\prod y{\in}A{-}\{x\}.\ f\ y)\ (f\ x)$
    **using** ‹*finite A*› **and** *x* **by** (*intro gcd-add-dvd-left2 dvd-sum dvd-prodI*) *auto*
  **also from** *x primes inj* **have** *coprime* $(\prod y{\in}A{-}\{x\}.\ f\ y)\ (f\ x)$
    **by** (*intro prod-coprime-left*) (*auto intro*!: *primes-coprime simp*: *inj-on-def*)
  **hence** $gcd\ (\prod y{\in}A{-}\{x\}.\ f\ y)\ (f\ x) = 1$
    **by** *simp*
  **finally show** *coprime a* $(f\ x)$
    **by** (*simp only*: *coprime-iff-gcd-eq-1*)
**qed**

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-syntax*
**begin**

**definition** *fps-XD′* **where** *fps-XD′ a* = ($\lambda b$. *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* ($\lambda x$. *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** $S \equiv \lambda n\ i.$ *fps-const* (*real* (*Stirling n i*))
**shows** (*fps-XD′ G* ^^ *n*) *H* =
($\sum m \le n.$ *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** ($\sum m \le$*Suc n*. *S* (*Suc n*) *m* ∗ *G* ^ *m* ∗ (*fps-deriv* ^^ *m*) *H*) =
($\sum i \le n.$ *of-nat* (*Suc i*) ∗ *S n* (*Suc i*) ∗ *G* ^ *Suc i* ∗ (*fps-deriv* ^^ *Suc i*) *H*)
+
($\sum i \le n.$ *S n i* ∗ *G* ^ *Suc i* ∗ (*fps-deriv* ^^ *Suc i*) *H*)
(**is** - = *sum* ($\lambda i.$ *?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* ($\lambda i.$ *?f* (*Suc i*)) {..*n*} = *sum* ($\lambda i.$ *?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* = ($\sum x \le n.$ *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**

## 3.2 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:

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

**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*) = ($\sum$ *i* = *0*..*n*. *Stirling i m* ∗ (*n choose i*))
**proof** −
**have** *real* (*Stirling* (*Suc n*) (*Suc m*)) = *real* ($\sum$ *i* = *0*..*n*. *Stirling i m* ∗ (*n choose i*))
**proof** (*induction n arbitrary*: *m*)
**case** (*Suc n m*)
**have** *real* ($\sum$ *i* = *0*..*Suc n*. *Stirling i m* ∗ (*Suc n choose i*)) =
*real* ($\sum$ *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* ($\sum$ *i* = *0*..*n*. *Stirling* (*Suc i*) *m* ∗ (*Suc n choose Suc i*)) =
*real* ($\sum$ *i* = *0*..*n*. (*n choose i*) ∗ *Stirling* (*Suc i*) *m*) +
*real* ($\sum$ *i* = *0*..*n*. (*n choose Suc i*) ∗ *Stirling* (*Suc i*) *m*)
**by** (*simp add*: *algebra-simps sum.distrib*)
**also have** ($\sum$ *i* = *0*..*n*. (*n choose Suc i*) ∗ *Stirling* (*Suc i*) *m*) =
($\sum$ *i* = *Suc 0*..*Suc n*. (*n choose i*) ∗ *Stirling i m*)
**by** (*subst sum.shift-bounds-cl-Suc-ivl*) *simp-all*
**also have** . . . = ($\sum$ *i* = *Suc 0*..*n*. (*n choose i*) ∗ *Stirling i m*)
**by** (*intro sum.mono-neutral-right*) *auto*
**also have** . . . = *real* ($\sum$ *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* ($\sum$ *i* = *0*..*n*. *Stirling i m* ∗ (*n choose i*)) = *real* (*Stirling* (*Suc n*) (*Suc m*))
**by** (*rule Suc.IH* [*symmetric*])
**also have** *real* ($\sum$ *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*) = (∑ *j*≤*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* = (∑ *j*≤*k*. *of-nat* (*k choose j*) ∗ *fps-exp 1* ^ *j* ∗ (− *1*) ^ (*k* − *j*))
**unfolding** *binomial-ring* **..**

**also have** . . . = ($\sum j \le k$. *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* = ($\sum j \le k$. (− *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* = ($\sum j \le k$. (− *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**

## 3.3 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* (λ*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* (λ*n*. *of-real* (*bernoulli n*) / *fact n* :: ′*a*) ∗ (*fps-exp 1* − *1*) = *fps-X*
**proof** (*rule fps-ext*)
**fix** *n*
**have** (*Abs-fps* (λ*n*. *of-real* (*bernoulli n*) / *fact n* :: ′*a*) ∗ (*fps-exp 1* − *1*)) $ *n* =
($\sum$ *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** . . . = ($\sum$ *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** ... = ($\sum i = 0..n.$ *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** ... = ($\sum i<n.$ *of-real* (*bernoulli i*) / (*fact i* ∗ *fact* (*n* − *i*)))
**by** (*cases n*) (*simp-all add*: *atLeast0AtMost lessThan-Suc-atMost* [*symmetric*])
**also have** ... = ($\sum i<n.$ *fact n* ∗ (*of-real* (*bernoulli i*) / (*fact i* ∗ *fact* (*n* − *i*)))) / *fact n*
**by** (*subst sum-distrib-left* [*symmetric*]) *simp-all*
**also have** ($\sum i<n.$ *fact n* ∗ (*of-real* (*bernoulli i*) / (*fact i* ∗ *fact* (*n* − *i*)))) =
($\sum i<n.$ *of-nat* (*n choose i*) ∗ *of-real* (*bernoulli i*) :: ′*a*)
**by** (*intro sum.cong*) (*simp-all add*: *binomial-fact*)
**also have** ... = *of-real* ($\sum i<n.$ (*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* (λ*n*. *of-real* (*bernoulli n*) / *fact n* :: ′*a*) ∗ (*fps-exp 1* − *1*)) $ *n* =
*fps-X* $ *n* **.**
**qed**
**moreover show** *bernoulli-fps* = *Abs-fps* (λ*n*. *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* (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** −
**have** *Abs-fps* (λ*n*. *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* (λ*n*. *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* (λ*n*. *of-real* (*bernoulli′ n*) / *fact n* :: ′*a*) = *fps-X*
**by** (*subst* (*asm*) *mult-left-cancel*) *simp-all*

**show** *bernoulli′-fps* = *Abs-fps* (λ*n*. *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* (λ*m*. (−*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* (λ*m*. (−*1*) ^ *m* / *real* (*Suc m*))
**unfolding** *fps-ln-def* **by** (*auto simp*: *fps-eq-iff*)
**hence** *Abs-fps* (λ*m*. (−*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**

**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*) =
(∑ *k*≤*n*. ∑ *i*≤*k*. (−*1*) ^ *i* ∗ (*k choose i*) ∗ *i* ^ *n* / *real* (*k* + *1*))
**proof** (*intro sum.cong*, *goal-cases*)
**case** (*2 k*)
**have** (−*1*) ^ *k* ∗ *fact k* / (*k* + *1*) ∗ *Stirling n k* =
(∑ *j*≤*k*. (−*1*) ^ *k* ∗ (−*1*) ^ (*k* − *j*) ∗ (*k choose j*) ∗ *j* ^ *n* / (*k* + *1*))
**by** (*simp add*: *Stirling-closed-form sum-distrib-left sum-divide-distrib mult-ac*)
**also have** ... = (∑ *j*≤*k*. (−*1*) ^ *j* ∗ (*k choose j*) ∗ *j* ^ *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** $[(\sum m<p. (-1) \,\hat{}\, m * ((p - 1)\ choose\ m) * m \,\hat{}\, (2*n)) =$
$(if\ (p - 1)\ dvd\ (2 * n)\ then\ -1\ else\ 0)]\ (mod\ p)$
**proof** $(cases\ (p - 1)\ dvd\ (2 * n))$
**case** *True*
**have** *cong-power-2n*: $[m \,\hat{}\, (2 * n) = 1]\ (mod\ p)$ **if** $m > 0\ m < p$ **for** $m$
**proof** $-$
**from** *True* **obtain** $q$ **where** $2 * n = (p - 1) * q$
**by** *blast*
**hence** $[m \,\hat{}\, (2 * n) = (m \,\hat{}\, (p - 1)) \,\hat{}\, q]\ (mod\ p)$
**by** (*simp add*: *power-mult*)
**also have** $[(m \,\hat{}\, (p - 1)) \,\hat{}\, q = 1 \,\hat{}\, q]\ (mod\ p)$
**using** *assms* ‹$m > 0$› ‹$m < p$› **by** (*intro cong-pow fermat-theorem*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**have** $(\sum m<p. (-1)\hat{}m * ((p - 1)\ choose\ m) * m \,\hat{}\, (2*n)) =$
$(\sum m\in\{0<..<p\}. (-1)\hat{}m * ((p - 1)\ choose\ m) * m \,\hat{}\, (2*n))$
**using** ‹$n > 0$› **by** (*intro sum.mono-neutral-right*) *auto*
**also have** $[\ldots = (\sum m\in\{0<..<p\}. (-1)\hat{}m * ((p - 1)\ choose\ m) * int\ 1)]\ (mod$
$p)$
**by** (*intro cong-sum cong-mult cong-power-2n cong-int*) *auto*
**also have** $(\sum m\in\{0<..<p\}. (-1)\hat{}m * ((p - 1)\ choose\ m) * int\ 1) =$
$(\sum m\in insert\ 0\ \{0<..<p\}. (-1)\hat{}m * ((p - 1)\ choose\ m)) - 1$
**by** (*subst sum.insert*) *auto*
**also have** $insert\ 0\ \{0<..<p\} = \{..p-1\}$
**using** *assms prime-gt-0-nat*[*of p*] **by** *auto*
**also have** $(\sum m\le p-1. (-1)\hat{}m * ((p - 1)\ choose\ m)) = 0$
**using** *prime-gt-1-nat*[*of p*] *assms* **by** (*subst choose-alternating-sum*) *auto*
**finally show** *?thesis* **using** *True* **by** *simp*
**next**
**case** *False*
**define** $n'$ **where** $n' = (2 * n)\ mod\ (p - 1)$
**from** *assms False* **have** $n' > 0$
**by** (*auto simp*: *n′-def dvd-eq-mod-eq-0*)
**from** *False* **have** $p \neq 2$ **by** *auto*
**with** *assms* **have** *odd p*
**using** *prime-prime-factor two-is-prime-nat* **by** *blast*

**have** *cong-pow-2n*: $[m \,\hat{}\, (2*n) = m \,\hat{}\, n']\ (mod\ p)$ **if** $m > 0\ m < p$ **for** $m$
**proof** $-$
**from** *assms* **and** *that* **have** *coprime p m*
**by** (*intro prime-imp-coprime*) *auto*
**have** $[2 * n = n']\ (mod\ (p - 1))$
**by** (*simp add*: *n′-def*)
**moreover have** *ord p m dvd* $(p - 1)$
**using** *order-divides-totient*[*of p m*] ‹*coprime p m*› *assms* **by** (*auto simp*:
*totient-prime*)
**ultimately have** $[2 * n = n']\ (mod\ ord\ p\ m)$
**by** (*rule cong-dvd-modulus-nat*)

**thus** *?thesis*
**using** ‹*coprime p m*› **by** (*subst order-divides-expdiff*) *auto*
**qed**

**have** ($\sum m<p.\ (-1)\widehat{\ }m * ((p - 1)\ choose\ m) * m\ \widehat{\ }(2*n)) =$
($\sum m\in\{0<..<p\}.\ (-1)\widehat{\ }m * ((p - 1)\ choose\ m) * m\ \widehat{\ }(2*n))$
**using** ‹$n > 0$› **by** (*intro sum.mono-neutral-right*) *auto*
**also have** [$\ldots = (\sum m\in\{0<..<p\}.\ (-1)\widehat{\ }m * ((p - 1)\ choose\ m) * m\ \widehat{\ }n')$] (*mod p*)
**by** (*intro cong-sum cong-mult cong-pow-2n cong-int*) *auto*
**also have** ($\sum m\in\{0<..<p\}.\ (-1)\widehat{\ }m * ((p - 1)\ choose\ m) * m\ \widehat{\ }n') =$
($\sum m\le p-1.\ (-1)\widehat{\ }m * ((p - 1)\ choose\ m) * m\ \widehat{\ }n'$)
**using** ‹$n' > 0$› **by** (*intro sum.mono-neutral-left*) *auto*
**also have** $\ldots = (\sum m\le p-1.\ (-1)\widehat{\ }(p - Suc\ m) * ((p - 1)\ choose\ m) * m\ \widehat{\ }n')$
**using** ‹$n' > 0$› *assms* ‹*odd p*› **by** (*intro sum.cong*) (*auto simp*: *uminus-power-if*)
**also have** $\ldots = 0$
**proof** −
**have** *of-int* ($\sum m\le p-1.\ (-1)\widehat{\ }(p - Suc\ m) * ((p - 1)\ choose\ m) * m\ \widehat{\ }n') =$
*real* (*Stirling* $n'$ $(p - 1)) *$ *fact* $(p - 1)$
**by** (*simp add*: *Stirling-closed-form*)
**also have** $n' < p - 1$
**using** *assms prime-gt-1-nat*[*of p*] **by** (*auto simp*: $n'$*-def*)
**hence** *Stirling* $n'$ $(p - 1) = 0$
**by** *simp*
**finally show** *?thesis* **by** *linarith*
**qed**
**finally show** *?thesis* **using** *False* **by** *simp*
**qed**

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** −
**define** $P$ :: *nat* ⇒ *real*
**where** $P = (\lambda m.\ if\ prime\ (m + 1) \wedge m\ dvd\ (2 * n)\ then\ 1\ /\ (m + 1)\ else\ 0)$

**define** $P'$ :: *nat* ⇒ *int*
**where** $P' = (\lambda m.\ if\ prime\ (m + 1) \wedge m\ dvd\ (2 * n)\ then\ 1\ else\ 0)$

**have** *?P* $= (\sum p \mid prime\ (p + 1) \wedge p\ dvd\ (2 * n).\ 1\ /\ real\ (p + 1))$

**by** (*rule sum.reindex-bij-witness*[*of* - $\lambda p.\ p + 1\ \lambda p.\ p - 1$])
(*use prime-gt-0-nat* **in** *auto*)
**also have** ... = ($\sum m \leq 2*n.\ P\ m$)
**using** ‹$n > 0$› **by** (*intro sum.mono-neutral-cong-left*) (*auto simp*: *P-def dest*!: *dvd-imp-le*)
**finally have** *bernoulli* ($2 * n$) + *?P* =
($\sum m \leq 2*n.\ (-1)$ ^ $m$ * (*of-int* (*fact m* * *Stirling* ($2*n$) *m*) / ($m + 1$)) + *P m*)
**by** (*simp add*: *sum.distrib bernoulli-conv-Stirling sum-divide-distrib algebra-simps*)
**also have** ... = ($\sum m \leq 2*n.$ *of-int* (($-1$) ^ $m$ * *fact m* * *Stirling* ($2*n$) *m* + $P'$ *m*) / ($m + 1$))
**by** (*intro sum.cong*) (*auto simp*: $P'$-*def P-def field-simps*)
**also have** ... $\in \mathbb{Z}$
**proof** (*rule sum-in-Ints*, *goal-cases*)
**case** (*1 m*)
**have** $m = 0 \vee m = 3 \vee$ *prime* ($m + 1$) $\vee$ ($\neg$*prime* ($m + 1$) $\wedge$ $m > 3$)
**by** (*cases* $m = 1$; *cases* $m = 2$) (*auto simp flip*: *numeral-2-eq-2*)
**then consider** $m = 0$ | $m = 3$ | *prime* ($m + 1$) | $\neg$*prime* ($m + 1$) $m > 3$
**by** *blast*
**thus** *?case*
**proof** *cases*
**assume** $m = 0$
**thus** *?case* **by** *auto*
**next**
**assume** [*simp*]: $m = 3$
**have** *real-of-int* (*fact m* * *Stirling* ($2 * n$) *m*) =
*real-of-int* ($9$ ^ $n + 3 - 3 * 4$ ^ $n$)
**using** ‹$n > 0$› **by** (*auto simp*: $P'$-*def fact-numeral Stirling-closed-form power-mult*
*atMost-nat-numeral binomial-fact zero-power*)
**hence** *int* (*fact m* * *Stirling* ($2 * n$) *m*) = $9$ ^ $n + 3 - 3 * 4$ ^ $n$
**by** *linarith*
**also have** [... = $1$ ^ $n$ + ($-1$) $- 3 * 0$ ^ $n$] (*mod 4*)
**by** (*intro cong-add cong-diff cong-mult cong-pow*) (*auto simp*: *cong-def*)
**finally have** *dvd*: *4 dvd int* (*fact m* * *Stirling* ($2 * n$) *m*)
**using** ‹$n > 0$› **by** (*simp add*: *cong-0-iff zero-power*)

**have** *real-of-int* (($-$ 1) ^ $m$ * *fact m* * *Stirling* ($2 * n$) *m* + $P'$ *m*) / ($m + 1$) =
$-$(*real-of-int* (*int* (*fact m* * *Stirling* ($2 * n$) *m*)) / *real-of-int 4*)
**using** ‹$n > 0$› **by** (*auto simp*: $P'$-*def*)
**also have** ... $\in \mathbb{Z}$
**by** (*intro Ints-minus of-int-divide-in-Ints dvd*)
**finally show** *?case* **.**
**next**
**assume** *composite*: $\neg$*prime* ($m + 1$) **and** $m > 3$
**obtain** *a b* **where** *ab*: $a * b = m + 1$ $a > 1$ $b > 1$
**using** ‹$m > 3$› *composite composite-imp-factors-nat*[*of* $m + 1$] **by** *auto*
**have** $a = b \longrightarrow a > 2$

**proof**
**assume** *a* = *b*
**hence** *a* ^ *2* > *2* ^ *2*
**using** ‹*m* > *3*› **and** *ab* **by** (*auto simp*: *power2-eq-square*)
**thus** *a* > *2*
**using** *power-less-imp-less-base* **by** *blast*
**qed**
**hence** *dvd*: (*m* + *1*) *dvd fact m*
**using** *product-dvd-fact*[*of a b*] *ab* **by** *auto*

**have** *real-of-int* ((− *1*) ^ *m* ∗ *fact m* ∗ *Stirling* (*2* ∗ *n*) *m* + *P′ m*) / *real* (*m* + *1*) =
*real-of-int* ((− *1*) ^ *m* ∗ *Stirling* (*2* ∗ *n*) *m*) ∗ (*real* (*fact m*) / (*m* + *1*))
**using** *composite* **by** (*auto simp*: *P′-def*)
**also have** ... ∈ ℤ
**by** (*intro Ints-mult Ints-real-of-nat-divide dvd*) *auto*
**finally show** *?case* **.**
**next**
**assume** *prime*: *prime* (*m* + *1*)
**have** *real-of-int* ((−*1*) ^ *m* ∗ *fact m* ∗ *int* (*Stirling* (*2* ∗ *n*) *m*)) =
(∑*j*≤*m*. (−*1*) ^ *m* ∗ (−*1*) ^ (*m* − *j*) ∗ (*m choose j*) ∗ *real-of-int j* ^ (*2* ∗ *n*))
**by** (*simp add*: *Stirling-closed-form sum-divide-distrib sum-distrib-left mult-ac*)
**also have** ... = *real-of-int* (∑*j*≤*m*. (−*1*) ^ *j* ∗ (*m choose j*) ∗ *j* ^ (*2* ∗ *n*))
**unfolding** *of-int-sum* **by** (*intro sum.cong*) (*auto simp*: *uminus-power-if*)
**finally have** (−*1*) ^ *m* ∗ *fact m* ∗ *int* (*Stirling* (*2* ∗ *n*) *m*) =
(∑*j*≤*m*. (−*1*) ^ *j* ∗ (*m choose j*) ∗ *j* ^ (*2* ∗ *n*)) **by** *linarith*
**also have** ... = (∑*j*<*m*+*1*. (−*1*) ^ *j* ∗ (*m choose j*) ∗ *j* ^ (*2* ∗ *n*))
**by** (*intro sum.cong*) *auto*
**also have** [... = (*if m dvd 2* ∗ *n then* − *1 else 0*)] (*mod* (*m* + *1*))
**using** *vonStaudt-Clausen-lemma*[*of n m* + *1*] *prime* ‹*n* > *0*› **by** *simp*
**also have** (*if m dvd 2* ∗ *n then* − *1 else 0*) = − *P′ m*
**using** *prime* **by** (*simp add*: *P′-def*)
**finally have** *int* (*m* + *1*) *dvd* ((− *1*) ^ *m* ∗ *fact m* ∗ *int* (*Stirling* (*2* ∗ *n*) *m*) + *P′ m*)
**by** (*simp add*: *cong-iff-dvd-diff*)
**hence** *real-of-int* ((−*1*)^*m* ∗ *fact m* ∗ *int* (*Stirling* (*2*∗*n*) *m*) + *P′ m*) / *of-int* (*int* (*m*+*1*)) ∈ ℤ
**by** (*intro of-int-divide-in-Ints*)
**thus** *?case* **by** *simp*
**qed**
**qed**
**finally show** *?thesis* **.**
**qed**

## 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* $\Rightarrow$ *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* $\Rightarrow$ *int* **where**
*bernoulli-num n* = $\lfloor$*bernoulli n* $*$ *bernoulli-denom n*$\rfloor$

**lemma** *finite-bernoulli-denom-set*: $n > (0 :: nat) \Longrightarrow$ *finite* $\{p.$ *prime p* $\wedge$ $(p - 1)$ *dvd n*$\}$
**by** (*rule finite-subset*[*of* - $\{..2*n+1\}$]) (*auto dest*!: *dvd-imp-le*)

**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*]: $n \neq 1 \Longrightarrow$ *odd n* $\Longrightarrow$ *bernoulli-denom n = 1*
**and** *bernoulli-denom-even*:
$n > 0 \Longrightarrow$ *even n* $\Longrightarrow$ *bernoulli-denom n* = $\prod\{p.$ *prime p* $\wedge$ $(p - 1)$ *dvd n*$\}$
**by** (*auto simp*: *bernoulli-denom-def*)

**lemma** *bernoulli-denom-pos*: *bernoulli-denom n* $> 0$
**by** (*auto simp*: *bernoulli-denom-def intro*!: *prod-pos*)

**lemma** *bernoulli-denom-nonzero* [*simp*]: *bernoulli-denom n* $\neq 0$
**using** *bernoulli-denom-pos*[*of n*] **by** *simp*

**lemma** *bernoulli-denom-code* [*code*]:
*bernoulli-denom n* =
(*if n = 1 then 2 else if n = 0* $\vee$ *odd n then 1*
*else prod-list* (*filter* ($\lambda p.$ $(p - 1)$ *dvd n*) (*primes-upto* $(n + 1)$))) (**is** - = *?rhs*)
**proof** (*cases even n* $\wedge$ $n > 0$)
**case** *True*
**hence** *?rhs* = *prod-list* (*filter* ($\lambda p.$ $(p - 1)$ *dvd n*) (*primes-upto* $(n + 1)$))
**by** *auto*
**also have** ... = $\prod$(*set* (*filter* ($\lambda p.$ $(p - 1)$ *dvd n*) (*primes-upto* $(n + 1)$)))
**by** (*subst prod.distinct-set-conv-list*) *auto*
**also have** (*set* (*filter* ($\lambda p.$ $(p - 1)$ *dvd n*) (*primes-upto* $(n + 1)$))) =
$\{p\in\{..n+1\}.$ *prime p* $\wedge$ $(p - 1)$ *dvd n*$\}$
**by** (*auto simp*: *set-primes-upto*)
**also have** ... = $\{p.$ *prime p* $\wedge$ $(p - 1)$ *dvd n*$\}$
**using** *True* **by** (*auto dest*: *dvd-imp-le*)
**also have** $\prod$... = *bernoulli-denom n*
**using** *True* **by** (*simp add*: *bernoulli-denom-even*)
**finally show** *?thesis* **..**
**qed** *auto*

```
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* $(k * \prod P - int\ c)$ $(\prod P)$
  **by** (*simp only*: *coprime-iff-gcd-eq-1*)

**have** *eq'*: $\prod P$ = *bernoulli-denom* (*2* ∗ *n*)
  **using** *n* **by** (*simp add*: *bernoulli-denom-def P-def*)
**show** *?thesis*
  **by** (*rule that*[*of* $k * \prod P - int\ c$]) (*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** −
**obtain** *a* :: *int* **where** *a*: *coprime a* (*bernoulli-denom n*) *bernoulli n* = *a* / *bernoulli-denom n*
  **using** *bernoulli-denom-correct*[*of n*] **by** *blast*
**thus** *?th1* **by** (*simp add*: *bernoulli-num-def*)
**with** *a* **show** *?th2* **by** *auto*
**qed**

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** −
**from** *assms* **have** $\prod$ {*2*, *3*} *dvd* $\prod$ {*p*. *prime p* ∧ (*p* − *1*) *dvd n*}
  **by** (*intro prod-dvd-prod-subset finite-bernoulli-denom-set*) *auto*
**with** *assms* **show** *?thesis* **by** (*simp add*: *bernoulli-denom-even*)
**qed**

**lemma** *squarefree-bernoulli-denom*: *squarefree* (*bernoulli-denom n*)
**by** (*auto intro*!: *squarefree-prod-coprime primes-coprime*
  *simp*: *bernoulli-denom-def squarefree-prime*)

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** (*cases even n* ∧ *n* > *0*)
**case** *True*
**hence** *bernoulli-denom n* = $\prod$ {*p*. *prime p* ∧ (*p* − *1*) *dvd n*}
  **by** (*auto simp*: *bernoulli-denom-def*)
**also have** ... *dvd* (*2* ∗ (*2* ^ *n* − *1*))
**proof** (*rule prime-prod-dvdI*; *clarify?*)
  **from** *True* **show** *finite* {*p*. *prime p* ∧ (*p* − *1*) *dvd n*}
    **by** (*intro finite-bernoulli-denom-set*) *auto*
**next**

```
    fix p assume p: prime p (p − 1) dvd n
    show p dvd (2 * (2 ^ n − 1))
    proof (cases p = 2)
      case False
      with p have p > 2
        using prime-gt-1-nat[of p] by force
      have [2 ^ n − 1 = 1 − 1] (mod p)
        using p ‹p > 2› prime-odd-nat
        by (intro cong-diff-nat Carmichael-divides) (auto simp: Carmichael-prime)
      hence p dvd (2 ^ n − 1)
        by (simp add: cong-0-iff)
      thus ?thesis by simp
    qed auto
  qed auto
  finally show ?thesis .
qed (auto simp: bernoulli-denom-def)

corollary bernoulli-bound:
  assumes n > 0
  shows   bernoulli-denom n ≤ 2 * (2 ^ n − 1)
proof −
  from assms have 2 ^ n > (1 :: nat)
    by (intro one-less-power) auto
  thus ?thesis
    by (intro dvd-imp-le[OF bernoulli-denom-dvd]) auto
qed
```

It can also be shown fairly easily from the von Staudt–Clausen theorem that if $p$ is prime and $2p + 1$ is not, then $B_{2p} \equiv \frac{1}{6}$ (mod 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 −
  from assms have p > 0
    using prime-gt-0-nat by auto
  define P where P = {q. prime q ∧ (q − 1) dvd (2 * p)}
  have P-eq: P = {2, 3}
  proof (intro equalityI subsetI)
    fix q assume q ∈ P
    hence q: prime q (q − 1) dvd (2 * p)
      by (simp-all add: P-def)
    have q − 1 ∈ {1, 2, p, 2 * p}
    proof −
      obtain b c where bc: b dvd 2 c dvd p q − 1 = b * c
        using division-decomp[OF q(2)] by auto
```

```
      from bc have b ∈ {1, 2} and c ∈ {1, p}
        using prime-nat-iff two-is-prime-nat ‹prime p› by blast+
      with bc show ?thesis by auto
    qed
    hence q ∈ {2, 3, p + 1, 2 ∗ p + 1}
      using prime-gt-0-nat[OF ‹prime q›] by force
    moreover have q ≠ p + 1
    proof
      assume [simp]: q = p + 1
      have even q ∨ even p by auto
      with ‹prime q› and ‹prime p› have p = 2
      using prime-odd-nat[of p] prime-odd-nat[of q] prime-gt-1-nat[of p] prime-gt-1-nat[of
q]
        by force
      with assms show False by (simp add: cong-def)
    qed
    ultimately show q ∈ {2, 3}
      using assms ‹prime q› by auto
  qed (auto simp: P-def)

  show [simp]: bernoulli-denom (2 ∗ p) = 6
    using ‹p > 0› P-eq by (subst bernoulli-denom-even) (auto simp: P-def)
  have bernoulli (2 ∗ p) + 5 / 6 ∈ ℤ
    using ‹p > 0› P-eq vonStaudt-Clausen[of p] by (auto simp: P-def)
  hence bernoulli (2 ∗ p) + 5 / 6 − 1 ∈ ℤ
    by (intro Ints-diff) auto
  thus bernoulli (2 ∗ p) − 1 / 6 ∈ ℤ by simp
  then obtain a where of-int a = bernoulli (2 ∗ p) − 1 / 6
    by (elim Ints-cases) auto
  hence real-of-int a = real-of-int (bernoulli-num (2 ∗ p) − 1) / 6
    by (auto simp: bernoulli-conv-num-denom)
  hence bernoulli-num (2 ∗ p) − 1 = 6 ∗ a
    by simp
  thus [bernoulli-num (2 ∗ p) = 1] (mod 6)
    by (auto simp: cong-iff-dvd-diff)
qed
```

### 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
```

**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* ($\lambda n$. *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* =
($\sum$ *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** . . . = ($\sum$ *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* =
($\sum$ *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* = ($\sum$ *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** ... = ($\sum$ *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** ... = ($\sum$ *k≤n. fact k* ∗ (*real* (*Stirling n k*) ∗ (− *1*) ^ *k*) ∗ *f* (*k* − *1*)) −
($\sum$ *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* (λ*n*. *?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* (λ*n*. *?f* (*Suc n*)) ... − *?S2* =
($\sum$ *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** −... = ($\sum$ *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* (λ*n*. *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* (λ*n*. *gen-akiyama-tanigawa f 0 n*)) (*1* − *fps-exp 1*) ∗ *fps-exp 1*)) \$ *n* =
($\sum$ *m≤n*. $\sum$ *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** ... = ($\sum$ *m≤n*. $\sum$ *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** ... = ($\sum$ *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** ... = ($\sum$ *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* = ($\sum$ *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* ∗ ($\sum$ *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** ($\sum$ *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** ... = ($\sum$ *m*≤*n*. $\sum$ *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** ... = ($\sum$ *k*≤*n*. $\sum$ *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* (λ*n*. *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 = (λn. 1 / real (Suc n))*
  **note** *gen-akiyama-tanigawa-fps[of f]*
  **also** {
    **have** *fps-ln 1 = fps-X ∗ Abs-fps (λn. (−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 (λn. (−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 (λn. 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**

### 3.7 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*
**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−Complex-Analysis.Complex-Analysis*
  *Bernoulli-FPS*
**begin**

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

**lemma** *linepath-cong*: $a = a' \Longrightarrow b = b' \Longrightarrow \mathit{linepath}\ a\ b = \mathit{linepath}\ a'\ b'$
  **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** $\mathit{residue}\ (\lambda z.\ 1\ /\ (z\ \hat{}\ m * (\mathit{exp}\ z - 1)))\ (2 * \mathit{pi} * \mathit{real\text{-}of\text{-}int}\ n * \mathrm{i}) =$
    $1\ /\ (2 * \mathit{pi} * \mathit{real\text{-}of\text{-}int}\ n * \mathrm{i})\ \hat{}\ m$
**proof** −
  **have** $\mathit{residue}\ (\lambda z.\ (1\ /\ z\ \hat{}\ m)\ /\ (\mathit{exp}\ z - 1))\ (2 * \mathit{pi} * \mathit{real\text{-}of\text{-}int}\ n * \mathrm{i}) =$
      $1\ /\ (2 * \mathit{pi} * \mathit{real\text{-}of\text{-}int}\ n * \mathrm{i})\ \hat{}\ m\ /\ 1$
    **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** $(\lambda k.\ 1\ /\ \textit{of-nat}\ (\textit{Suc}\ k)\ \widehat{}\ (2{*}n') :: \textit{complex})\ \textit{sums}$
  $\textit{of-real}\ ((-1)\ \widehat{}\ \textit{Suc}\ n' * \textit{bernoulli}\ (2{*}n') * (2 * \textit{pi})\ \widehat{}\ (2 * n')\ /\ (2 * \textit{fact}\ (2{*}n')))$
**proof** −
  **define** *n* **where** $n = 2 * n'$
  **from** *n′* **have** *n*: $n \geq 2$ *even n* **by** (*auto simp*: *n-def*)
  **define** *zeta* :: *complex* **where** $\textit{zeta} = (\sum k.\ 1\ /\ \textit{of-nat}\ (\textit{Suc}\ k)\ \widehat{}\ n)$

  **have** *summable* $(\lambda k.\ 1\ /\ \textit{of-nat}\ (\textit{Suc}\ k)\ \widehat{}\ n :: \textit{complex})$
    **using** *inverse-power-summable*[*of n*] *n*
    **by** (*subst summable-Suc-iff*) (*simp add*: *divide-simps*)
  **hence** $(\lambda k.\ \sum i{<}k.\ 1\ /\ \textit{of-nat}\ (\textit{Suc}\ i)\ \widehat{}\ n) \longrightarrow \textit{zeta}$
    **by** (*subst* (*asm*) *summable-sums-iff*) (*simp add*: *sums-def zeta-def*)
  **also have** $(\lambda k.\ \sum i{<}k.\ 1\ /\ \textit{of-nat}\ (\textit{Suc}\ i)\ \widehat{}\ n) = (\lambda k.\ \sum i{\in}\{0{<}..k\}.\ 1\ /\ \textit{of-nat}\ i\ \widehat{}\ n)$
    **by** (*intro ext sum.reindex-bij-witness*[*of* - $\lambda n.\ n - 1$ *Suc*]) *auto*
  **finally have** *zeta-limit*: $(\lambda k.\ \sum i{\in}\{0{<}..k\}.\ 1\ /\ \textit{of-nat}\ i\ \widehat{}\ n) \longrightarrow \textit{zeta}$ **.**

  — This is the exponential generating function of the Bernoulli numbers.
  **define** *f* **where** $f = (\lambda z{::}\textit{complex}.\ \textit{if}\ z = 0\ \textit{then}\ 1\ \textit{else}\ z\ /\ (\textit{exp}\ z - 1))$

  — 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 $2ik\pi$ for $k \in \mathbb{Z}$.
  **define** *g* **where** $g = (\lambda z{::}\textit{complex}.\ 1\ /\ (z\ \widehat{}\ n * (\textit{exp}\ z - 1)))$

  — We integrate along a rectangle of width $2m$ and height $2(2m + 1)\pi$ 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** $\gamma$ :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *complex*
  **where** $\gamma = (\lambda m.\ rectpath\ (-real\ m - real\ (2*m+1)*pi*i)\ (real\ m + real\ (2*m+1)*pi*i))$

— This set is a convex open enclosing set the contains our path.
**define** $A$ **where** $A = (\lambda m{::}nat.\ box\ (-(real\ m+1) - (2*m+2)*pi*i)\ (real\ m+1 + (2*m+2)*pi*i))$

— These are all the singularities in the enclosing inside the path (and also inside $A$).
**define** $S$ **where** $S = (\lambda m{::}nat.\ (\lambda n.\ 2 * pi * of\text{-}int\ n * i)\ `\ \{-m..m\})$

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

**have** *zeros*: $\exists k{\in}\{-int\ m..int\ m\}.\ z = 2 * pi * of\text{-}int\ k * i$ **if** $z \in A\ m\ exp\ z = 1$ **for** $z\ m$
**proof** −
  **from** *that*(*2*) **obtain** $k$ **where** *z-eq*: $z = 2 * pi * of\text{-}int\ k * i$
    **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'*: $z\ \hat{}\ n * (exp\ z - 1) \neq 0$ **if** $z \in A\ m - S\ m$ **for** $z\ m$
  **using** *zeros*[*of z*] *that* **by** (*auto simp*: *S-def*)

— The singularities all lie strictly inside the integration path.
**have** *subset*: $S\ m \subseteq box\ (-real\ m - real(2*m+1)*pi*i)\ (real\ m + real(2*m+1)*pi*i)$ **if** $m > 0$ **for** $m$
**proof** (*rule*, *goal-cases*)
  **case** (*1 z*)
  **then obtain** $k$ :: *int* **where** $k$: $k \in \{-int\ m..int\ m\}\ z = 2 * pi * k * i$
    **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*) ⊆ *A m − S m* **if** $m > 0$ **for** *m*
**proof** −
**have** *path-image* ($\gamma$ *m*) ⊆ *cbox* (−*real m* − (2 ∗ *m* + 1) ∗ *pi* ∗ i) (*real m* + (2 ∗ *m* + 1) ∗ *pi* ∗ i)
**unfolding** $\gamma$*-def* **by** (*rule path-image-rectpath-subset-cbox*) *auto*
**also have** . . . ⊆ *A m* **unfolding** *A-def*
**by** (*subst subset-box-complex*) *auto*
**finally have** *path-image* ($\gamma$ *m*) ⊆ *A m* **.**
**moreover have** *path-image* ($\gamma$ *m*) ∩ *S m* = {}
**proof** *safe*
**fix** *z* **assume** *z*: *z* ∈ *path-image* ($\gamma$ *m*) *z* ∈ *S m*
**from** *this*(2) **obtain** *k* :: *int* **where** *k*: *z* = 2 ∗ *pi* ∗ *k* ∗ i
**by** (*auto simp*: *S-def*)
**hence** [*simp*]: *Re z* = 0 **by** *simp*
**from** *z*(1) **have** |*Im z*| = *of-int* (2∗*m*+1) ∗ *pi*
**using** ‹$m > 0$› **by** (*auto simp*: $\gamma$*-def path-image-rectpath*)
**also have** |*Im z*| = *of-int* (2 ∗ |*k*|) ∗ *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* ∈ {} **..**
**qed**
**ultimately show** *path-image* ($\gamma$ *m*) ⊆ *A m − S m* **by** *blast*
**qed**

— We now obtain a closed form for the Bernoulli numbers using the integral.
**have** *eq*: ($\sum$ *x*∈{0<..*m*}. 1 / *of-nat x* ^ *n*) =
*contour-integral* ($\gamma$ *m*) *g* ∗ (2 ∗ *pi* ∗ i) ^ *n* / (4 ∗ *pi* ∗ i) −
*complex-of-real* (*bernoulli n* / *fact n*) ∗ (2 ∗ *pi* ∗ i) ^ *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* (*fps-exp* 1 − 1 :: *complex 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* =
$2 * pi * \mathrm{i} * (\sum z{\in}S\ m.\ winding\text{-}number\ (\gamma\ m)\ z * residue\ g\ z)$
**proof** (*rule Residue-theorem*)
**have** *cbox* ($-real\ m - (2 * m + 1) * pi * \mathrm{i}$) ($real\ m + (2 * m + 1) * pi * \mathrm{i}$) $\subseteq$ *A m*
**unfolding** *A-def* **by** (*subst subset-box-complex*) *simp-all*
**thus** $\forall z.\ z \notin A\ m \longrightarrow winding\text{-}number\ (\gamma\ m)\ z = 0$ **unfolding** *$\gamma$-def*
**by** (*intro winding-number-rectpath-outside allI impI*) *auto*
**qed** (*insert holo path-subset m*, *auto simp*: *$\gamma$-def A-def S-def intro*: *convex-connected*)
— Clearly, all the winding numbers are 1
**also have** *winding-number* ($\gamma$ *m*) *z* = *1* **if** $z \in S\ m$ **for** *z*
**unfolding** *$\gamma$-def* **using** *subset*[*of m*] *that m* **by** (*subst winding-number-rectpath*) *blast+*
**hence** $(\sum z{\in}S\ m.\ winding\text{-}number\ (\gamma\ m)\ z * residue\ g\ z) = (\sum z{\in}S\ m.\ residue\ g\ z)$
**by** (*intro sum.cong*) *simp-all*
**also have** ... = $(\sum k{=}{-int}\ m..int\ m.\ residue\ g\ (2 * pi * of\text{-}int\ k * \mathrm{i}))$
**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** $(\sum k{\in}\ldots.\ residue\ g\ (2 * pi * of\text{-}int\ k * \mathrm{i})) =$
$residue\ g\ 0 + (\sum k{\in}\{-int\ m..m\}-\{0\}.\ residue\ g\ (2 * pi * of\text{-}int\ k * \mathrm{i}))$
**by** (*subst sum.insert*) *auto*
— The residue at the origin is just the *n*-th coefficient of *f*.
**also have** $residue\ g\ 0 = residue\ (\lambda z.\ f\ z\ /\ z\ \widehat{}\ Suc\ n)\ 0$ **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** $(\sum k{\in}\{-int\ m..m\}-\{0\}.\ residue\ g\ (2 * pi * of\text{-}int\ k * \mathrm{i})) =$
$(\sum k{\in}\{-int\ m..m\}-\{0\}.\ 1\ /\ of\text{-}int\ k\ \widehat{}\ n)\ /\ (2 * pi * \mathrm{i})\ \widehat{}\ n$
**proof** (*subst sum-divide-distrib*, *intro refl sum.cong*, *goal-cases*)
**case** (*1 k*)
**hence** $*$: $residue\ g\ (2 * pi * of\text{-}int\ k * \mathrm{i}) = 1\ /\ (2 * complex\text{-}of\text{-}real\ pi * of\text{-}int\ k * \mathrm{i})\ \widehat{}\ n$
**unfolding** *g-def* **by** (*subst residue-bernoulli*) *auto*
**thus** *?case* **using** *1* **by** (*subst* $*$) (*simp add*: *divide-simps power-mult-distrib*)
**qed**
**also have** $(\sum k{\in}\{-int\ m..m\}-\{0\}.\ 1\ /\ of\text{-}int\ k\ \widehat{}\ n) =$
$(\sum (a,b){\in}\{0{<}..m\}\times\{-1,1{::}int\}.\ 1\ /\ of\text{-}int\ (int\ a)\ \widehat{}\ n :: complex)$ **using** *n*
**by** (*intro sum.reindex-bij-witness*[*of - $\lambda k.\ snd\ k * int\ (fst\ k)$ $\lambda k.\ (nat\ |k|, sgn\ k)$*])
(*auto split*: *if-splits simp*: *abs-if*)
**also have** ... = $(\sum x{\in}\{0{<}..m\}.\ 2\ /\ of\text{-}nat\ x\ \widehat{}\ n)$

**using** *n* **by** (*subst sum.Sigma* [*symmetric*]) *auto*
**also have** ... $= (\sum x \in \{0 <.. m\}.\ 1 \ / \ of\text{-}nat\ x \ \hat{}\ n) * 2$
**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* ($\lambda m.$ *norm* (*contour-integral* ($\gamma$ *m*) *g*) $\leq$
$((4 + 12 * pi) + 6 * pi / m) / real\ m \ \hat{}\ (n - 1))$ *sequentially*
**using** *eventually-gt-at-top*[*of 1*::*nat*]
**proof** *eventually-elim*
**case** (*elim m*)
**let** $?c = (2{*}m{+}1) * pi * \mathrm{i}$
**define** *I* **where** $I = (\lambda p1\ p2.$ *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*: $\gamma\ m =$ *linepath p1 p2* +++ *linepath p2 p3* +++ *linepath p3 p4* +++ *linepath p4 p1*
(**is** $\gamma\ m = ?\gamma'$) **unfolding** $\gamma$*-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* $\gamma$ *m* **using** *elim*
**by** (*intro contour-integrable-holomorphic-simple*[*OF holo - - path-subset*])
(*auto simp*: $\gamma$*-def A-def S-def intro*!: *finite-imp-closed*)
**have** *norm* (*contour-integral* ($\gamma$ *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** ... $\leq$ *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*) $\leq 1 / real\ m \ \hat{}\ n *$ *norm* $(p2 - p1)$ (**is** - $\leq$ *?B1* $*$ -)
**unfolding** *I-def*
**proof** (*intro contour-integral-bound-linepath*)
**fix** *z* **assume** *z*: $z \in$ *closed-segment p1 p2*
**define** *a* **where** $a = Re\ z$
**from** *z* **have** *z*: $z = a - (2{*}m{+}1) * pi * \mathrm{i}$
**by** (*subst* (*asm*) *closed-segment-same-Im*)
(*auto simp*: *p1-def p2-def complex-eq-iff a-def*)
**have** $real\ m * 1 \leq (2{*}m{+}1) * pi$
**using** *pi-ge-two* **by** (*intro mult-mono*) *auto*
**also have** $(2{*}m{+}1) * pi = |Im\ z|$ **by** (*simp add*: *z*)
**also have** $|Im\ z| \leq$ *norm z* **by** (*rule abs-Im-le-cmod*)
**finally have** *norm* $z \geq m$ **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*: $z = m + b * \mathrm{i}$
**by** (*subst* (*asm*) *closed-segment-same-Re*)
(*auto simp*: *p2-def p3-def algebra-simps complex-eq-iff b-def*)
**from** *elim* **have** $2 \le 1 + real\ m$ **by** *simp*
**also have** $\ldots \le exp\ (real\ m)$ **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** $\ldots \le norm\ (exp\ z - 1)$
**by** (*rule norm-triangle-ineq2*)
**finally have** $norm\ (exp\ z - 1) \ge 1$ **by** *simp*
**moreover have** $norm\ z \ge m$
**using** *z* **and** *abs-Re-le-cmod*[*of z*] **by** *simp*
**ultimately have** $norm\ z \ \hat{}\ n * norm\ (exp\ z - 1) \ge real\ m \ \hat{}\ n * 1$ **using** *elim*
**by** (*intro mult-mono power-mono*) (*auto simp*: *z*)
**thus** $norm\ (g\ z) \le 1\ /\ real\ m \ \hat{}\ n$ **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** $p3 - p2 = of\text{-}real\ (2*(2*real\ m+1)*pi) * \mathrm{i}$ **by** (*simp add*: *p2-def p3-def*)
**also have** $norm\ \ldots = 2 * (2 * real\ m + 1) * pi$
**unfolding** *norm-mult norm-of-real* **by** *simp*

**also have** $norm\ (I\ p4\ p1) \le (2\ /\ real\ m \ \hat{}\ n) * norm\ (p1 - p4)$ (**is** $- \le ?B4 * -$)
**unfolding** *I-def*
**proof** (*rule contour-integral-bound-linepath*)
**fix** *z* **assume** *z*: $z \in closed\text{-}segment\ p4\ p1$
**define** *b* **where** $b = Im\ z$
**from** *z* **have** *z*: $z = -real\ m + b * \mathrm{i}$
**by** (*subst* (*asm*) *closed-segment-same-Re*)
(*auto simp*: *p1-def p4-def algebra-simps b-def complex-eq-iff*)
**from** *elim* **have** $2 \le 1 + real\ m$ **by** *simp*
**also have** $\ldots \le exp\ (real\ m)$ **by** (*rule exp-ge-add-one-self*)
**finally have** $1\ /\ 2 \le 1 - exp\ (-real\ m)$
**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** $\ldots \le norm\ (exp\ z - 1)$
**by** (*subst norm-minus-commute*, *rule norm-triangle-ineq2*)
**finally have** $norm\ (exp\ z - 1) \ge 1\ /\ 2$ **by** *simp*
**moreover have** $norm\ z \ge m$
**using** *z* **and** *abs-Re-le-cmod*[*of z*] **by** *simp*
**ultimately have** $norm\ z \ \hat{}\ n * norm\ (exp\ z - 1) \ge real\ m \ \hat{}\ n * (1\ /\ 2)$ **using** *elim*
**by** (*intro mult-mono power-mono*) (*auto simp*: *z*)
**thus** $norm\ (g\ z) \le 2\ /\ real\ m \ \hat{}\ n$ **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 $2 \leq n$.
**moreover have** (λ*m*. (*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** ∗: (λ*m*. *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** (λ*m*. *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* ⟷ (λ*m*. ∑*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** (λ*m*. ∑*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** (λ*m*. ∑*k*∈{*0*<..*m*}. *1* / *of-nat k* ^ *n*) = (λ*m*. ∑*k*∈{..<*m*}. *1* / *of-nat* (*Suc k*) ^ *n*)
**by** (*intro ext sum.reindex-bij-witness*[*of* - *Suc* λ*n*. *n* − *1*]) *auto*
**also have** ... ⟶ *?L* ⟷ (λ*k*. *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** (λ*k*. *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** (λ*k*. *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** $*$: (λ*k*. *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*: (λ*n*. *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*: (λ*n*. *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*: (λ*n*. *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*: (λ*n*. *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.