Congruences of Bernoulli Numbers

Manuel Eberl

June 20, 2024

Abstract

This entry provides proofs for two important congruences involving Bernoulli numbers. The proofs follow Cohen's textbook *Number Theory Volume II: Analytic and Modern Tools* [1]. In the following we write $\mathcal{B}_k = N_k/D_k$ for the k-th Bernoulli number (with $\gcd(N_k, D_k) = 1$).

The first result that I showed is *Voronoi's congruence*, which states that for any even integer $k \geq 2$ and all positive coprime integers a, n we have:

$$(a^k - 1)N_k \equiv k a^{k-1} D_k \sum_{m=1}^{n-1} m^{k-1} \left\lfloor \frac{ma}{n} \right\rfloor \pmod{n}$$

Building upon this, I then derive *Kummer's congruence*. In its common form, it states that for a prime p and even integers k, k' with $\min(k, k') \geq e + 1$ and $(p-1) \nmid k$ and $k \equiv k' \pmod{\varphi(p^e)}$, we have:

$$\frac{\mathcal{B}_k}{k} \equiv \frac{\mathcal{B}_{k'}}{k'} \pmod{p^e}$$

The version proved in my entry is slightly more general than this.

One application of these congruences is to prove that there are infinitely many irregular primes, which I formalised as well.

Contents

1 Preliminary facts

theory *Kummer_Library*
imports
"HOL-Number_Theory.Number_Theory"
"Bernoulli.Bernoulli_Zeta"
begin

1.1 Miscellaneous facts

lemma *fact_ge_monomial:*
fixes *k :: "'a :: {linordered_semidom, semiring_char_0}"*
assumes *"n ≥ n0" "fact n0 ≥ c * k ^ n0" "of_nat n0 ≥ k" "k ≥ 0"*
shows *"fact n ≥ c * k ^ n"*
⟨*proof*⟩

lemma *fact_ge_2pi_power:*
assumes *"n ≥ 23"*
shows *"fact n ≥ (2 * pi) ^ n * n"*
⟨*proof*⟩

lemma *Rats_power_int: "x ∈ ℚ ⟹ x powi n ∈ ℚ"*
⟨*proof*⟩

lemma *coprimeI_via_bezout:*
fixes *x y :: "'a :: algebraic_semidom"*
assumes *"a * x + b * y = 1"*
shows *"coprime x y"*
⟨*proof*⟩

lemma *quotient_of_eqI:*
assumes *"coprime a b" "b > 0" "x = of_int a / of_int b"*
shows *"quotient_of x = (a, b)"*
⟨*proof*⟩

lemma *quotient_of_of_nat [simp]: "quotient_of (of_nat n) = (int n, 1)"*
⟨*proof*⟩

lemma *quotient_of_of_int [simp]: "quotient_of (of_int n) = (n, 1)"*
⟨*proof*⟩

lemma *quotient_of_fraction_conv_normalize:*
"quotient_of (of_int a / of_int b) = Rat.normalize (a, b)"
⟨*proof*⟩

lemma *dvd_imp_div_dvd: "(b :: 'a :: algebraic_semidom) dvd a ⟹ a div b dvd a"*
⟨*proof*⟩

lemma *dvd_rat_normalize:*

assumes *"b ≠ 0"*
shows *"fst (Rat.normalize (a, b)) dvd a" "snd (Rat.normalize (a, b)) dvd b"*
⟨*proof*⟩

lemma *of_int_div: "b dvd a ⟹ of_int (a div b) = of_int a / (of_int b :: 'a :: field_char_0)"*
⟨*proof*⟩

lemma *coprime_lcm_left:*
fixes *a b c :: "'a :: semiring_gcd"*
shows *"coprime a c ⟹ coprime b c ⟹ coprime (lcm a b) c"*
⟨*proof*⟩

lemma *coprime_Lcm_left:*
fixes *x y :: "'a :: semiring_Gcd"*
assumes *"finite A" "⋀x. x ∈ A ⟹ coprime x y"*
shows *"coprime (Lcm A) y"*
⟨*proof*⟩

lemma *coprimeI_by_prime_factors:*
fixes *x y :: "'a :: factorial_semiring"*
assumes *"⋀p. p ∈ prime_factors x ⟹ ¬p dvd y"*
assumes *"x ≠ 0"*
shows *"coprime x y"*
⟨*proof*⟩

lemma *multiplicity_int: "multiplicity (int p) (int n) = multiplicity p n"*
⟨*proof*⟩

lemma *squarefree_int_iff [simp]: "squarefree (int n) ⟷ squarefree n"*
⟨*proof*⟩

lemma *squarefree_imp_multiplicity_prime_le_1:*
"squarefree n ⟹ n ≠ 0 ⟹ prime p ⟹ multiplicity p n ≤ 1"
⟨*proof*⟩

lemma *residue_primroot_is_generator':*
assumes *"m > 1"* **and** *"residue_primroot m g"*
shows *"bij_betw (λi. g ^ i mod m) {1..totient m} (totatives m)"*
⟨*proof*⟩

1.2 Facts about congruence

lemma *cong_modulus_mono:*
assumes *"[a = b] (mod m)" "m' dvd m"*
shows *"[a = b] (mod m')"*

⟨*proof*⟩

lemma *cong_pow_totient:*
fixes *x x' n k k' :: nat*
assumes *"[x = x'] (mod n)" "[k = k'] (mod totient n)" "coprime x n"*
shows *"[x ^ k = x' ^ k'] (mod n)"*
⟨*proof*⟩

lemma *cong_modulus_power:*
assumes *"[a = b] (mod (n ^ k))" "k > 0"*
shows *"[a = b] (mod n)"*
⟨*proof*⟩

lemma *cong_mult_cancel:*
assumes *"[n * a = n * b] (mod (n * m))" "n ≠ 0"*
shows *"[a = b] (mod m)"*
⟨*proof*⟩

lemma *cong_mult_square:*
assumes *"[a = 0] (mod n)" "[b = b'] (mod n)"*
shows *"[a * b = a * b'] (mod (n^2))"*
⟨*proof*⟩

lemma *sum_reindex_bij_betw_cong:*
assumes *"⋀a. a ∈ S ⟹ i (j a) = a"*
assumes *"⋀a. a ∈ S ⟹ j a ∈ T"*
assumes *"⋀b. b ∈ T ⟹ j (i b) = b"*
assumes *"⋀b. b ∈ T ⟹ i b ∈ S"*
assumes *"⋀a. a ∈ S ⟹ [h (j a) = g a] (mod m)"*
shows *"[sum g S = sum h T] (mod m)"*
⟨*proof*⟩

lemma *power_mult_cong:*
fixes *a b :: "'a :: unique_euclidean_ring"*
assumes *"[a = b] (mod n^k)"* **and** *"k' ≤ k + l"*
shows *"[n^l * a = n^l * b] (mod n^k')"*
⟨*proof*⟩

lemma *residue_primroot_power_cong_neg1:*
fixes *x :: nat* **and** *p :: nat*
assumes *"prime p" "p ≠ 2" "residue_primroot p x"*
shows *"[int x ^ ((p - 1) div 2) = -1] (mod p)"*
⟨*proof*⟩

lemma *cong_mod_left: "[a = b] (mod p) ⟹ [a mod p = b] (mod p)"*
⟨*proof*⟩

lemma *cong_mod_right: "[a = b] (mod p) ⟹ [a = b mod p] (mod p)"*
⟨*proof*⟩

lemma *cong_mod: "[a = b] (mod p) ⟹ [a mod p = b mod p] (mod p)"*
⟨*proof*⟩

1.3 Modular inverses

definition *modular_inverse* **where**
"modular_inverse p n = fst (bezout_coefficients n p) mod p"

lemma *cong_modular_inverse1:*
assumes *"coprime n p"*
shows *"[n * modular_inverse p n = 1] (mod p)"*
⟨*proof*⟩

lemma *cong_modular_inverse2:*
assumes *"coprime n p"*
shows *"[modular_inverse p n * n = 1] (mod p)"*
⟨*proof*⟩

lemma *coprime_modular_inverse [simp, intro]:*
fixes *n :: "'a :: {euclidean_ring_gcd,unique_euclidean_semiring}"*
assumes *"coprime n p"*
shows *"coprime (modular_inverse p n) p"*
⟨*proof*⟩

lemma *modular_inverse_int_nonneg: "p > 0 ⟹ modular_inverse p (n :: int) ≥ 0"*
⟨*proof*⟩

lemma *modular_inverse_int_less: "p > 0 ⟹ modular_inverse p (n :: int) < p"*
⟨*proof*⟩

lemma *modular_inverse_int_eqI:*
fixes *x y :: int*
assumes *"y ∈ {0..<m}" "[x * y = 1] (mod m)"*
shows *"modular_inverse m x = y"*
⟨*proof*⟩

lemma *modular_inverse_1 [simp]:*
assumes *"m > (1 :: int)"*
shows *"modular_inverse m 1 = 1"*
⟨*proof*⟩

lemma *modular_inverse_int_mult:*
fixes *x y :: int*
assumes *"coprime x m" "coprime y m" "m > 0"*
shows *"modular_inverse m (x * y) = (modular_inverse m y * modular_inverse m x) mod m"*

⟨*proof*⟩

lemma *bij_betw_int_remainders_mult:*
  fixes *a n :: int*
  assumes *a: "coprime a n"*
  shows *"bij_betw (λm. a * m mod n) {1..<n} {1..<n}"*
⟨*proof*⟩

1.4 Facts about Bernoulli numbers

definition *bernoulli_rat :: "nat ⇒ rat"*
  where *"bernoulli_rat n = of_int (bernoulli_num n) / of_int (bernoulli_denom n)"*

bundle *bernoulli_notation*
begin
notation *bernoulli_rat ("ℬ")*
end

bundle *no_bernoulli_notation*
begin
no_notation *bernoulli_rat ("ℬ")*
end

lemma *bernoulli_num_eq_0_iff: "bernoulli_num n = 0 ⟷ odd n ∧ n ≠ 1"*
⟨*proof*⟩

lemma *bernoulli_num_odd_eq_0: "odd k ⟹ k ≠ 1 ⟹ bernoulli_num k = 0"*
  ⟨*proof*⟩

lemma *prime_dvd_bernoulli_denom_iff:*
  assumes *"prime p" "even k" "k > 0"*
  shows *"p dvd bernoulli_denom k ⟷ (p - 1) dvd k"*
⟨*proof*⟩

lemma *bernoulli_num_denom_eqI:*
  assumes *"bernoulli k = of_int a / of_nat b" "coprime a b" "b > 0"*
  shows *"bernoulli_num k = a" "bernoulli_denom k = b"*
⟨*proof*⟩

lemma *bernoulli_rat_eq_0_iff: "bernoulli_rat n = 0 ⟷ odd n ∧ n ≠ 1"*
  ⟨*proof*⟩

lemma *bernoulli_rat_odd_eq_0: "odd n ⟹ n ≠ 1 ⟹ bernoulli_rat n = 0"*
  ⟨*proof*⟩

lemma *bernoulli_rat_conv_bernoulli: "of_rat (bernoulli_rat n) = bernoulli n"*
⟨*proof*⟩

lemma *quotient_of_bernoulli_rat [simp]:*
"quotient_of (bernoulli_rat n) = (bernoulli_num n, int (bernoulli_denom n))"
⟨*proof*⟩

end

2 Congruence of rational numbers modulo an integer

theory *Rat_Congruence*
imports *Kummer_Library*
begin

2.1 p-adic valuation of a rational

The notion of the multiplicity $\nu_p(n)$ of a prime p in an integer n can be generalised to rational numbers via $\nu_p(a/b) = \nu_p(a) - \nu_p(b)$. This is also called the p-adic valuation of a/b.

definition *qmultiplicity :: "int ⇒ rat ⇒ int"* **where**
"qmultiplicity p x = (case quotient_of x of (a, b) ⇒ int (multiplicity p a) - int (multiplicity p b))"

lemma *qmultiplicity_of_int [simp]:*
"qmultiplicity p (of_int n) = int (multiplicity p n)"
⟨*proof*⟩

lemma *qmultiplicity_of_nat [simp]:*
"qmultiplicity p (of_nat n) = int (multiplicity p n)"
⟨*proof*⟩

lemma *qmultiplicity_numeral [simp]:*
"qmultiplicity p (numeral n) = int (multiplicity p (numeral n))"
⟨*proof*⟩

lemma *qmultiplicity_0 [simp]: "qmultiplicity p 0 = 0"*
⟨*proof*⟩

lemma *qmultiplicity_1 [simp]: "qmultiplicity p 1 = 0"*
⟨*proof*⟩

lemma *qmultiplicity_minus [simp]: "qmultiplicity p (-x) = qmultiplicity*

```
p x"
```
⟨*proof*⟩

lemma *qmultiplicity_divide_of_int:*
assumes *"x ≠ 0" "y ≠ 0" "prime_elem p"*
shows *"qmultiplicity p (of_int x / of_int y) = int (multiplicity p x) - int (multiplicity p y)"*
⟨*proof*⟩

lemma *qmultiplicity_mult [simp]:*
assumes *"prime_elem p" "x ≠ 0" "y ≠ 0"*
shows *"qmultiplicity p (x * y) = qmultiplicity p x + qmultiplicity p y"*
⟨*proof*⟩

lemma *qmultiplicity_inverse [simp]:*
"qmultiplicity p (inverse x) = -qmultiplicity p x"
⟨*proof*⟩

lemma *qmultiplicity_divide [simp]:*
assumes *"prime_elem p" "x ≠ 0" "y ≠ 0"*
shows *"qmultiplicity p (x / y) = qmultiplicity p x - qmultiplicity p y"*
⟨*proof*⟩

lemma *qmultiplicity_nonneg_iff:*
assumes *"a ≠ 0" "b ≠ 0" "coprime a b" "prime p"*
shows *"qmultiplicity p (of_int a / of_int b) ≥ 0 ⟷ ¬p dvd b"*
⟨*proof*⟩

lemma *qmultiplicity_nonneg_imp_not_dvd_denom:*
assumes *"qmultiplicity p x ≥ 0" "|p| ≠ 1"*
shows *"¬p dvd snd (quotient_of x)"*
⟨*proof*⟩

lemma *qmultiplicity_prime_nonneg_imp_coprime_denom:*
assumes *"qmultiplicity p x ≥ 0" "prime p"*
shows *"coprime (snd (quotient_of x)) p"*
⟨*proof*⟩

2.2 Rational modulo operation

Similarly, we can define (a/b) mod m whenever b and m are coprime by choosing to interpret $(1/b)$ mod m as the modular inverse of b modulo m:

definition *qmod :: "rat ⇒ int ⇒ int"* (**infixl** *"qmod" 70*) **where**
*"x qmod m = (let (a, b) = quotient_of x in if coprime b m then (a * modular_inverse m b) mod m else 0)"*

lemma *qmod_mod_absorb [simp]: "x qmod m mod m = x qmod m"*

⟨*proof*⟩

lemma *qmod_of_nat [simp]: "m > 1 ⟹ of_nat x qmod m = int x mod m"*
⟨*proof*⟩

lemma *qmod_of_int [simp]: "m > 1 ⟹ of_int x qmod m = x mod m"*
⟨*proof*⟩

lemma *qmod_numeral [simp]: "m > 1 ⟹ numeral n qmod m = numeral n mod m"*
⟨*proof*⟩

lemma *qmod_0 [simp]: "0 qmod m = 0"*
⟨*proof*⟩

lemma *qmod_1 [simp]: "m > 1 ⟹ 1 qmod m = 1"*
⟨*proof*⟩

lemma *qmod_fraction_eq:*
assumes *"coprime b m" "b ≠ 0" "m > 0"*
shows *"(of_int a / of_int b) qmod m = a * modular_inverse m b mod m"*
⟨*proof*⟩

2.3 Congruence relation

With this, it is now straightforward to define the congruence relation $x = y \pmod{m}$ for rational x, y:

definition *qcong :: "rat ⇒ rat ⇒ int ⇒ bool" (‹(1[_ = _] '(' qmod _'))›)*
where
"[a = b] (qmod m) ⟷
coprime (snd (quotient_of a)) m ∧ coprime (snd (quotient_of b)) m ∧ a qmod m = b qmod m"

lemma *qcong_of_int_iff [simp]:*
assumes *"m > 1"*
shows *"[of_int a = of_int b] (qmod m) ⟷ [a = b] (mod m)"*
⟨*proof*⟩

lemma *cong_imp_qcong:*
assumes *"[a = b] (mod m)" "m > 1"*
shows *"[of_int a = of_int b] (qmod m)"*
⟨*proof*⟩

lemma *cong_imp_qcong_of_nat:*
assumes *"[a = b] (mod m)" "m > 1"*
shows *"[of_nat a = of_nat b] (qmod m)"*
⟨*proof*⟩

lemma *qcong_refl [intro]: "coprime (snd (quotient_of q)) m ⟹ [q = q] (qmod m)"*
⟨*proof*⟩

lemma *qcong_sym_eq: "[q1 = q2] (qmod m) ⟷ [q2 = q1] (qmod m)"*
⟨*proof*⟩

lemma *qcong_sym: "[q1 = q2] (qmod m) ⟹ [q2 = q1] (qmod m)"*
⟨*proof*⟩

lemma *qcong_trans [trans]:*
assumes *"[q1 = q2] (qmod m)" "[q2 = q3] (qmod m)"*
shows *"[q1 = q3] (qmod m)"*
⟨*proof*⟩

lemma *qcong_0D:*
assumes *"[x = 0] (qmod m)"*
shows *"m dvd fst (quotient_of x)"*
⟨*proof*⟩

lemma *qcong_0_iff:*
"[x = 0] (qmod m) ⟷ m dvd fst (quotient_of x) ∧ coprime (snd (quotient_of x)) m"
⟨*proof*⟩

lemma *qcong_1 [simp]: "[a = b] (qmod 1)"*
⟨*proof*⟩

lemma *mod_minus_cong':*
fixes *a b :: "'a :: euclidean_ring_cancel"*
assumes *"(- a) mod b = (- a') mod b"*
shows *"a mod b = a' mod b"*
⟨*proof*⟩

lemma *qcong_minus_minus_iff:*
"[-b = -c] (qmod a) ⟷ [b = c] (qmod a)"
⟨*proof*⟩

lemma *qcong_minus: "[b = c] (qmod a) ⟹ [-b = -c] (qmod a)"*
⟨*proof*⟩

lemma *qcong_fraction_iff:*
assumes *"b ≠ 0" "d ≠ 0" "coprime b m" "coprime d m" "m > 0"*
shows *"[of_int a / of_int b = of_int c / of_int d] (qmod m) ⟷ [a * d = b * c] (mod m)"*
⟨*proof*⟩

lemma *qcong_fractionI:*
assumes *"x = of_int a / of_int b" "b ≠ 0" "coprime b m"*

shows *"[x = of_int a / of_int b] (qmod m)"*
⟨*proof*⟩

lemma *qcong_add:*
assumes *"[x = x'] (qmod m)" "[y = y'] (qmod m)" "m > 0"*
shows *"[x + y = x' + y'] (qmod m)"*
⟨*proof*⟩

lemma *qcong_diff:*
assumes *"[x = x'] (qmod m)" "[y = y'] (qmod m)" "m > 0"*
shows *"[x - y = x' - y'] (qmod m)"*
⟨*proof*⟩

lemma *qcong_mult:*
assumes *"[x = x'] (qmod m)" "[y = y'] (qmod m)" "m > 0"*
shows *"[x * y = x' * y'] (qmod m)"*
⟨*proof*⟩

lemma *qcong_divide_of_int:*
assumes *"[x = x'] (qmod m)" "[c = c'] (mod m)" "coprime c m" "c ≠ 0" "c' ≠ 0" "m > 0"*
shows *"[x / of_int c = x' / of_int c'] (qmod m)"*
⟨*proof*⟩

lemma *qcong_mult_of_int_cancel_left:*
assumes *"[of_int a * b = of_int a * c] (qmod m)" "coprime a m" "a ≠ 0" "m > 0"*
shows *"[b = c] (qmod m)"*
⟨*proof*⟩

lemma *qcong_pow:*
assumes *"[a = b] (qmod m)" "m > 0"*
shows *"[a ^ n = b ^ n] (qmod m)"*
⟨*proof*⟩

lemma *qcong_sum:*
"[sum f A = sum g A] (qmod m)" **if** *"⋀x. x ∈ A ⟹ [f x = g x] (qmod m)" "m > 0"*
⟨*proof*⟩

lemma *qcong_prod:*
"[prod f A = prod g A] (qmod m)" **if** *"(⋀x. x ∈ A ⟹ [f x = g x] (qmod m))" "m > 0"*
⟨*proof*⟩

lemma *qcong_modulus_abs_1:*
assumes *"|n| = 1"*
shows *"[a = b] (qmod n)"*
⟨*proof*⟩

lemma *qcong_divide_of_int_left_iff:*
assumes *"coprime c n" "c ≠ 0" ‹n > 0›*
shows *"[a / of_int c = b] (qmod n) ⟷ [a = b * of_int c] (qmod n)"*
⟨*proof*⟩

lemma *qcong_divide_of_nat_left_iff:*
assumes *"coprime (int c) n" "c ≠ 0" "n > 0"*
shows *"[a / of_nat c = b] (qmod n) ⟷ [a = b * of_nat c] (qmod n)"*
⟨*proof*⟩

lemma *qcong_divide_of_int_right_iff:*
assumes *"coprime c n" "c ≠ 0" "n > 0"*
shows *"[a = b / of_int c] (qmod n) ⟷ [a * of_int c = b] (qmod n)"*
⟨*proof*⟩

lemma *qcong_divide_of_nat_right_iff:*
assumes *"coprime (int c) n" "c ≠ 0" "n > 0"*
shows *"[a = b / of_nat c] (qmod n) ⟷ [a * of_nat c = b] (qmod n)"*
⟨*proof*⟩

lemma *qcong_qmultiplicity_pos_transfer:*
assumes *"[x = y] (qmod m)" "qmultiplicity m x > 0"*
shows *"y = 0 ∨ qmultiplicity m y > 0"*
⟨*proof*⟩

end

3 The Voronoi congruence

theory *Voronoi_Congruence*
imports *Kummer_Library Rat_Congruence*
begin

unbundle *bernoulli_notation*

lemma *sum_of_powers_mod_prime:*
assumes *p: "prime p"*
shows *"[(∑x=1..<p. int x ^ m) = (if (p - 1) dvd m then -1 else 0)] (mod p)"*
⟨*proof*⟩

lemma *sum_of_powers_mod_prime':*
fixes *p m :: nat*
assumes *p: "prime p" "¬(p - 1) dvd m"*
shows *"[(∑x=1..<p. x ^ m) = 0] (mod p)"*
⟨*proof*⟩

lemma *voronoi_congruence_aux1:*

assumes *"prime p" "j ≥ 4"*
shows *"multiplicity p (j + 1) ≤ (if p ∈ {2, 3} then 1 else 0) + j - 2"*
⟨*proof*⟩

context
fixes *S :: "nat ⇒ nat ⇒ nat"* **and** *D :: "nat ⇒ nat"* **and** *N :: "nat ⇒ int"*
defines *"S ≡ (λk n. ∑r<n. r ^ k)"*
defines *"N ≡ bernoulli_num"* **and** *"D ≡ bernoulli_denom"*
begin

lemma *voronoi_congruence_aux2:*
fixes *k n :: nat*
assumes *k: "even k" "k ≥ 2"* **and** *n: "n > 0"*
shows *"real (S k n) = (∑j≤k. real (k choose j) / real (j + 1) * bernoulli (k - j) * real (n ^ (j + 1)))"*
⟨*proof*⟩

lemma *voronoi_congruence_aux3:*
fixes *k n :: nat*
assumes *k: "even k" "k ≥ 2"* **and** *n: "n > 0"*
shows *"[D k * S k n = N k * n] (mod (n^2))"*
⟨*proof*⟩

Proposition 9.5.20

theorem *voronoi_congruence:*
fixes *k n :: nat* **and** *a :: int*
assumes *k: "even k" "k ≥ 2"* **and** *n: "n > 0"* **and** *a: "coprime a n"*
shows *"[(a^k-1) * N k = k * a^(k-1) * D k * (∑m=1..<n. m^(k-1) * ((m * a) div n))] (mod n)"*
⟨*proof*⟩

corollary *voronoi_congruence':*
fixes *k p :: nat* **and** *a :: int*
assumes *k: "even k" "k ≥ 2"* **and** *p: "prime p" "¬(p - 1) dvd k"* **and** *a: "¬p dvd a" "[a ^ k ≠ 1] (mod p)"*
shows *"[ℬ k = of_int (k * a^(k-1)) / of_int (a^k - 1) * of_int (∑m=1..<p. m^(k-1) * ((m * a) div p))] (qmod p)"*
⟨*proof*⟩

corollary *voronoi_congruence_harvey:*
fixes *k p :: nat* **and** *c a :: int* **and** *h :: "nat ⇒ rat"*
assumes *k: "even k" "k ∈ {2..p-3}"* **and** *p: "prime p" "p ≥ 5"* **and** *c: "c ∈ {0<..<p}" "[c^k ≠ 1] (mod p)"*
assumes *a: "[a * c = 1] (mod p)"*
defines *"h ≡ (λm. of_int (m - c * ((a * m) mod p)) / of_int p + of_int (c - 1) / 2)"*

shows *"[ℬ k = rat_of_nat k / rat_of_int (1 - c ^ k) * (∑m=1..<p. rat_of_nat m^(k-1) * h m)] (qmod p)"*
⟨*proof*⟩

corollary *voronoi_congruence_harvey':*
fixes *k p :: nat* **and** *g :: nat* **and** *h :: "nat ⇒ rat"* **and** *a :: int*
assumes *k: "even k" "k ∈ {2..p-3}"* **and** *p: "prime p" "p ≥ 5"*
assumes *g: "residue_primroot p g" "g ∈ {0<..<p}"*
assumes *a: "[a * int g = 1] (mod int p)"*
defines *"h' ≡ (λm. rat_of_int (int m mod p - g * ((a * m) mod p)) / of_int p + of_int (int g - 1) / 2)"*
shows *"[ℬ k = 2 * of_nat k / of_int (1 - int g ^ k) * (∑i=1..(p-1) div 2. of_nat (g^(i*(k-1))) * h' (g^i))] (qmod int p)"*
⟨*proof*⟩

unbundle *no_bernoulli_notation*

end

end

4 Kummer's Congruence

theory *Kummer_Congruence*
imports *Voronoi_Congruence*
begin

unbundle *bernoulli_notation*

context
fixes *S :: "nat ⇒ nat ⇒ nat"* **and** *D :: "nat ⇒ nat"* **and** *N :: "nat ⇒ int"*
defines *"S ≡ (λk n. ∑r<n. r ^ k)"*
defines *"N ≡ bernoulli_num"* **and** *"D ≡ bernoulli_denom"*
begin

Auxiliary lemma for Proposition 9.5.23: if k is even and $(p-1) \nmid k$, then $\nu_p(N_k) \geq \nu_p(k)$.

lemma *multiplicity_prime_bernoulli_num_ge:*
fixes *p k :: nat*
assumes *p: "prime p" "¬(p - 1) dvd k"* **and** *k: "even k"*
shows *"multiplicity p (N k) ≥ multiplicity p k"*
⟨*proof*⟩

Proposition 9.5.23: if k is even and $(p-1) \nmid k$, then B_k/k is p-integral.

lemma *bernoulli_k_over_k_is_p_integral:*
fixes *p k :: nat*

assumes *p: "prime p" "¬(p - 1) dvd k"* **and** *k: "k ≠ 1"*
shows *"qmultiplicity p (ℬ k / of_nat k) ≥ 0"*
⟨*proof*⟩

lemma *kummer_congruence_aux:*
fixes *k p a :: nat*
assumes *k: "even k" "k ≥ 2"* **and** *p: "¬(p - 1) dvd k" "prime p"*
assumes *a: "¬p dvd a"*
assumes *s: "s ≥ multiplicity p k"*
shows *"[of_int ((1 - int p^(k-1)) * (int a^k - 1)) * ℬ k / of_nat k =*
*of_int (int a^(k-1) *
(∑m∈{m∈{1..<p^(s+e)}. ¬p dvd m}. m^(k-1) * (int m * a div p ^ (e + s))))] (qmod p^e)"*
⟨*proof*⟩

theorem *kummer_congruence:*
fixes *k k' p :: nat*
assumes *k: "even k" "k ≥ 2"* **and** *k': "even k'" "k' ≥ 2"* **and** *p: "¬(p - 1) dvd k" "prime p"*
assumes *cong: "[k = k'] (mod totient (p ^ e))"*
shows *"[(of_nat p^(k-1)-1) * ℬ k / of_nat k =*
*(of_nat p^(k'-1)-1) * ℬ k' / of_nat k'] (qmod (p^e))"*
⟨*proof*⟩

corollary *kummer_congruence':*
assumes *kk': "even k" "even k'" "k ≥ e+1" "k' ≥ e+1"*
assumes *cong: "[k = k'] (mod totient (p ^ e))"*
assumes *p: "prime p" "¬(p-1) dvd k"*
shows *"[ℬ k / of_nat k = ℬ k' / of_nat k'] (qmod (p^e))"*
⟨*proof*⟩

corollary *kummer_congruence'_prime:*
assumes *kk': "even k" "even k'" "k > 0" "k' > 0"*
assumes *cong: "[k = k'] (mod (p - 1))"*
assumes *p: "prime p" "¬(p-1) dvd k"*
shows *"[ℬ k / of_nat k = ℬ k' / of_nat k'] (qmod p)"*
⟨*proof*⟩

end

unbundle *no_bernoulli_notation*

end

5 Regular primes

theory *Regular_Primes*

imports *Kummer_Congruence Zeta_Function.Zeta_Function*
begin

definition *regular_prime :: "nat ⇒ bool"* **where**
"regular_prime p ⟷ prime p ∧ (p = 2 ∨ (∀k∈{2..p-3}. even k ⟶ ¬p dvd bernoulli_num k))"

definition *irregular_prime :: "nat ⇒ bool"* **where**
"irregular_prime p ⟷ prime p ∧ (p ≠ 2 ∧ (∃k∈{2..p-3}. even k ∧ p dvd bernoulli_num k))"

lemma *irregular_primeI:*
assumes *"prime p" "p ≠ 2" "p dvd bernoulli_num k" "even k" "k ∈ {2..p-3}"*
shows *"irregular_prime p"*
⟨*proof*⟩

lemma *bernoulli_32: "bernoulli 32 = -7709321041217 / 510"*
⟨*proof*⟩

The smallest irregular prime is 37.

lemma *irregular_prime_37: "irregular_prime 37"*
⟨*proof*⟩

Irregularity of primes can be certified relatively easily with the code generator:

experiment
begin

lemma *irregular_59: "irregular_prime 59"*
⟨*proof*⟩

lemma *irregular_67: "irregular_prime 67"*
⟨*proof*⟩

end

end

6 Infinitude of irregular primes

theory *Irregular_Primes_Infinite*
imports *Regular_Primes*
begin

One consequence of Kummer's congruence is that there are infinitely many irregular primes. We shall derive this here.

lemma *zeta_real_gt_1:*

assumes *"x > 1"*
shows *"Re (zeta (of_real x)) > 1"*
⟨*proof*⟩

lemma *zeta_real_gt_1':*
assumes *"Re s > 1" "s ∈ ℝ"*
shows *"Re (zeta s) > 1"*
⟨*proof*⟩

lemma *bernoulli_even_conv_zeta:*
*"complex_of_real (bernoulli (2*n)) = (-1)^Suc n * 2 * fact (2*n) / (2*pi)^(2*n) * zeta (2 * of_nat n)"*
⟨*proof*⟩

lemma *bernoulli_even_conv_zeta':*
*"bernoulli (2*n) = (-1)^Suc n * 2 * fact (2*n) / (2*pi)^(2*n) * Re (zeta (2 * of_nat n))"*
⟨*proof*⟩

lemma *abs_bernoulli_even_conv_zeta:*
assumes *"even n" "n > 0"*
shows *"|bernoulli n| = 2 * fact n / (2*pi)^n * Re (zeta (of_nat n))"*
⟨*proof*⟩

lemma *abs_bernoulli_over_n_ge_2:*
assumes *"n ≥ 23" "even n"*
shows *"|bernoulli n / n| ≥ 2"*
⟨*proof*⟩

lemma *infinite_irregular_primes_aux:*
assumes *"finite P" "∀p∈P. irregular_prime p" "37 ∈ P"*
shows *"∃p. irregular_prime p ∧ p ∉ P"*
⟨*proof*⟩

theorem *infinite_irregular_primes: "infinite {p. irregular_prime p}"*
⟨*proof*⟩

end

References

[1] H. Cohen. *Number Theory: Volume II: Analytic and Modern Tools.* Graduate Texts in Mathematics. Springer New York, 2007.