Dedekind Sums

Manuel Eberl, Anthony Bordg, Lawrence C. Paulson, Wenda Li

December 19, 2025

Abstract

For integers h, k, the Dedekind sum is defined as

$$s(h,k) = \sum_{r=1}^{k-1} \frac{r}{k}\left(\left\{\frac{hr}{k}\right\} - \frac{1}{2}\right)$$

where $\{x\} = x - \lfloor x \rfloor$ denotes the fractional part of x.

These sums occur in various contexts in analytic number theory, e.g. in the functional equation of the Dedekind η function or in the study of modular forms.

We give the definition of $s(h,k)$ and prove its basic properties, including the reciprocity law

$$s(h,k) + s(k,h) = \frac{1}{12hk} + \frac{h}{12k} + \frac{k}{12h} - \frac{1}{4}$$

and various congruence results.

Our formalisation follows Chapter 3 of Apostol's *Modular Functions and Dirichlet Series in Number Theory* [1] and contains all facts related to Dedekind sums from it (without the exercises).

Contents

1 Dedekind sums

```
theory Dedekind_Sums
imports
  Complex_Main
  "HOL-Library.Periodic_Fun"
  "HOL-Library.Real_Mod"
  "HOL-Number_Theory.Number_Theory"
  "Bernoulli.Bernoulli_FPS"
begin
```

1.1 Preliminaries

```
lemma rcong_of_intI: "[a = b] (mod m) ⟹ [of_int a = of_int b] (rmod
(of_int m))"
  by (metis cong_iff_lin mult.commute of_int_add of_int_mult rcong_altdef)

lemma rcong_of_int_iff: "[of_int a = of_int b] (rmod (of_int m)) ⟷
[a = b] (mod m)"
proof
  assume "[of_int a = of_int b] (rmod (of_int m))"
  then obtain k where "real_of_int b = of_int a + of_int k * of_int m"
    unfolding rcong_altdef by blast
  also have "... = of_int (a + k * m)"
    by simp
  finally have "b = a + k * m"
    by linarith
  thus "[a = b] (mod m)"
    by (simp add: cong_iff_lin)
qed (intro rcong_of_intI)
```

1.2 Definition and basic properties

```
definition dedekind_sum :: "int ⇒ int ⇒ real" where
  "dedekind_sum h k ≡ ∑r∈{1..<k}. (of_int r / of_int k * (frac (of_int
(h*r) / of_int k) - 1/2))"

definition dedekind_frac :: "real ⇒ real" where
  "dedekind_frac x = (if x ∈ ℤ then 0 else frac x - 1 / 2)"

lemma dedekind_frac_int [simp]: "x ∈ ℤ ⟹ dedekind_frac x = 0"
  by (auto simp: dedekind_frac_def)

notation dedekind_frac ("⟨_⟩")

interpretation dedekind_frac: periodic_fun_simple' dedekind_frac
proof
  show "⟨x + 1⟩ = ⟨x⟩" for x
    by (auto simp: dedekind_frac_def frac_1_eq)
qed
```

```
lemma dedekind_frac_uminus [simp]: "⟨-x⟩ = -⟨x⟩"
  by (auto simp add: dedekind_frac_def frac_neg)

lemma dedekind_frac_one_minus [simp]: "⟨1 - x⟩ = -⟨x⟩"
  by (metis dedekind_frac.minus_1 dedekind_frac_uminus minus_diff_eq)

lemma dedekind_frac_rcong:
  assumes "[x = x'] (rmod 1)"
  shows   "⟨x⟩ = ⟨x'⟩"
proof -
  from assms obtain k where k: "x' = x + of_int k"
    by (auto simp: rcong_altdef)
  thus ?thesis
    by (auto simp: dedekind_frac.plus_of_int)
qed

lemma dedekind_frac_mod:
  "⟨of_int (a mod k) / of_int k⟩ = ⟨of_int a / of_int k⟩"
proof (cases "k = 0")
  case False
  have "[real_of_int (a mod k) = real_of_int a] (rmod real_of_int k)"
    by (intro rcong_of_intI cong_mod_leftI cong_refl)
  thus ?thesis using False
    by (intro dedekind_frac_rcong rcong_divide_modulus) auto
qed auto

lemma sum_dedekind_frac_eq_0:
  "(∑r∈{1..<k}. ⟨of_int r / of_int k⟩) = 0"
proof -
  have "(∑r∈{1..<k}. ⟨of_int (r) / of_int k⟩) =
        (∑r∈{1..<k}. ⟨of_int (k - r) / of_int k⟩)"
    by (intro sum.reindex_bij_witness[of _ "λr. k - r" "λr. k - r"])
auto
  also have "(∑r∈{1..<k}. ⟨of_int (k - r) / of_int k⟩) =
             (∑r∈{1..<k}. ⟨1 + -of_int r / of_int k⟩)"
    by (intro sum.cong refl arg_cong[of _ _ dedekind_frac]) (auto simp:
field_simps)
  also have "... = -(∑r∈{1..<k}. ⟨of_int r / of_int k⟩)"
    by (simp add: sum_negf)
  finally show ?thesis
    by simp
qed

lemma sum_dedekind_aux:
  assumes "f (0::int) = 0"
  shows   "(∑r∈{0..<k}. f r) = (∑r∈{1..<k}. f r)"
proof (rule sum.mono_neutral_right)
  have "{0..<k} - {1..<k} ⊆ {0}"
```

```
    by auto
  thus "∀i∈{0..<k}-{1..<k}. f i = 0"
    using assms by blast
qed auto

lemma sum_dedekind_frac_eq_0':
  "(∑r∈{0..<k}. ⟨of_int r / of_int k⟩) = 0"
proof -
  have "(∑r∈{0..<k}. ⟨of_int r / of_int k⟩) = (∑r∈{1..<k}. ⟨of_int
r / of_int k⟩)"
    by (rule sum_dedekind_aux) auto
  with sum_dedekind_frac_eq_0[of k] show ?thesis
    by simp
qed

lemma sum_dedekind_frac_mult_eq_0:
  assumes "coprime h k"
  shows   "(∑r∈{1..<k}. ⟨of_int (h * r) / of_int k⟩) = 0"
proof -
  have "(∑r∈{1..<k}. ⟨of_int (h * r) / of_int k⟩) =
        (∑r∈{1..<k}. ⟨of_int ((h * r) mod k) / of_int k⟩)"
  proof (intro sum.cong)
    fix r assume r: "r ∈ {1..<k}"
    have k: "k > 0"
      using r by auto
    have "(h * r) mod k = h * r - k * ((h * r) div k)"
      by (metis minus_mult_div_eq_mod)
    also have "real_of_int ... / of_int k = of_int (h * r) / of_int k
- of_int ((h * r) div k)"
      using k by (auto simp: field_simps)
    also have "⟨...⟩ = ⟨of_int (h * r) / of_int k⟩"
      by (rule dedekind_frac.minus_of_int)
    finally show "⟨of_int (h * r) / of_int k⟩ = ⟨of_int ((h * r) mod k)
/ of_int k⟩" ..
  qed auto
  also have "... = (∑r∈{1..<k}. ⟨of_int r / of_int k⟩)"
    by (rule sum.reindex_bij_betw[OF bij_betw_int_remainders_mult[OF assms]])
  also have "... = 0"
    by (rule sum_dedekind_frac_eq_0)
  finally show ?thesis .
qed

lemma sum_dedekind_frac_mult_eq_0':
  assumes "coprime h k"
  shows   "(∑r∈{0..<k}. ⟨of_int (h * r) / of_int k⟩) = 0"
proof -
  have "(∑r∈{0..<k}. ⟨of_int (h * r) / of_int k⟩) = (∑r∈{1..<k}. ⟨of_int
(h * r) / of_int k⟩)"
    by (intro sum_dedekind_aux) auto
  also have "... = 0"
    by (intro sum_dedekind_frac_mult_eq_0 assms)
  finally show ?thesis .
qed

lemma dedekind_sum_altdef:
  assumes "coprime h k"
  shows   "dedekind_sum h k = (∑r∈{1..<k}. ⟨of_int r / of_int k⟩ * ⟨of_int
(h*r) / of_int k⟩)"
proof -
  have "(∑r∈{1..<k}. ⟨of_int r / of_int k⟩ * ⟨of_int (h*r) / of_int
k⟩) =
        (∑r∈{1..<k}. (of_int r / of_int k - 1 / 2) * ⟨of_int (h*r) /
of_int k⟩)"
  proof (intro sum.cong ballI)
    fix r assume r: "r ∈ {1..<k}"
    hence "1 ≤ r" "r < k"
      by auto
    hence "¬k dvd r"
      using zdvd_not_zless by auto
    hence "real_of_int r / real_of_int k ∉ ℤ"
      using r by (subst of_int_div_of_int_in_Ints_iff) auto
    moreover have "frac (real_of_int r / real_of_int k) = real_of_int
r / real_of_int k"
      using r by (subst frac_eq) auto
    ultimately show "⟨real_of_int r / real_of_int k⟩ * ⟨real_of_int (h
* r) / real_of_int k⟩ =
                     (real_of_int r / real_of_int k - 1 / 2) * ⟨real_of_int
(h * r) / real_of_int k⟩"
      by (subst dedekind_frac_def) auto
  qed auto
  also have "... = (∑r∈{1..<k}. of_int r / of_int k * ⟨of_int (h*r) /
of_int k⟩) -
                  1 / 2 * (∑r∈{1..<k}. ⟨of_int (h*r) / of_int k⟩)"
    unfolding sum_distrib_left sum_subtractf [symmetric] by (simp add:
ring_distribs)
  also have "(∑r∈{1..<k}. ⟨of_int (h*r) / of_int k⟩) = 0"
    by (intro sum_dedekind_frac_mult_eq_0 assms)
  also have "(∑r∈{1..<k}. of_int r / of_int k * ⟨of_int (h*r) / of_int
k⟩) = dedekind_sum h k"
    unfolding dedekind_sum_def
  proof (intro sum.cong)
    fix r assume r: "r ∈ {1..<k}"
    have "¬k dvd h * r"
    proof
      assume "k dvd h * r"
      with assms have "k dvd r"
        using coprime_commute coprime_dvd_mult_right_iff by blast
      hence "k ≤ r"
```

```
        using r by (intro zdvd_imp_le) auto
      thus False
        using r by simp
    qed
    hence "real_of_int (h * r) / real_of_int k ∉ ℤ"
      using r by (subst of_int_div_of_int_in_Ints_iff) auto
    thus "real_of_int r / real_of_int k * ⟨real_of_int (h * r) / real_of_int
k⟩ =
          real_of_int r / real_of_int k * (frac (real_of_int (h * r) /
real_of_int k) - 1 / 2)"
      by (auto simp: dedekind_frac_def)
  qed auto
  finally show ?thesis
    by simp
qed


theorem dedekind_sum_cong:
  assumes "[h' = h] (mod k)"
  assumes "coprime h' k ∨ coprime h k"
  shows "dedekind_sum h' k = dedekind_sum h k"
proof -
  have coprime: "coprime h' k" "coprime h k"
    using coprime_cong_cong_left[OF assms(1)] assms(2) by auto
  show ?thesis
    unfolding coprime[THEN dedekind_sum_altdef]
  proof (intro sum.cong)
    fix r assume r: "r ∈ {1..<k}"
    have "[real_of_int (h' * r) = real_of_int (h * r)] (rmod real_of_int
k)"
      by (intro rcong_of_intI cong_mult cong_refl assms)
    hence "⟨real_of_int (h' * r) / real_of_int k⟩ = ⟨real_of_int (h *
r) / real_of_int k⟩"
      using r by (intro dedekind_frac_rcong rcong_divide_modulus) auto
    thus "⟨real_of_int r / real_of_int k⟩ * ⟨real_of_int (h' * r) / real_of_int
k⟩ =
          ⟨real_of_int r / real_of_int k⟩ * ⟨real_of_int (h * r) / real_of_int
k⟩"
      by simp
  qed auto
qed

theorem dedekind_sum_negate:
  assumes "coprime h k"
  shows   "dedekind_sum (-h) k = -dedekind_sum h k"
proof -
  have *: "coprime (-h) k"
    using assms by auto
  show ?thesis
```

```
    unfolding dedekind_sum_altdef[OF assms] dedekind_sum_altdef[OF *]
    by (simp_all add: sum_negf)
qed

theorem dedekind_sum_negate_cong:
  assumes "[h' = -h] (mod k)" "coprime h' k ∨ coprime h k"
  shows "dedekind_sum h' k = -dedekind_sum h k"
proof -
  have coprime: "coprime h' k" "coprime h k"
    using coprime_cong_cong_left[OF assms(1)] assms(2) by auto
  from coprime show ?thesis
    using assms(1) dedekind_sum_cong dedekind_sum_negate by metis
qed

theorem dedekind_sum_inverse:
  assumes "[h * h' = 1] (mod k)"
  shows   "dedekind_sum h k = dedekind_sum h' k"
proof -
  have 1: "coprime h k"
    using assms coprime_iff_invertible_int by blast
  have 2: "coprime h' k"
    using assms coprime_iff_invertible_int by (subst (asm) mult.commute)
auto
  have "dedekind_sum h' k =
          (∑r = 1..<k. ⟨real_of_int (1 * r) / real_of_int k⟩ * ⟨real_of_int
(h' * r) / real_of_int k⟩)"
    unfolding dedekind_sum_altdef[OF 2] by simp
  also have "... = (∑r = 1..<k. ⟨real_of_int (h * ((h' * r) mod k)) /
real_of_int k⟩ *
                                ⟨real_of_int ((h' * r) mod k) / real_of_int
k⟩)"
  proof (rule sum.cong, goal_cases)
    case (2 r)
    have "[real_of_int (1 * r) = real_of_int (h * h' * r)] (rmod real_of_int
k)"
      using assms by (intro rcong_of_intI cong_mult cong_refl assms) (auto
simp: cong_sym)
    also have "[real_of_int (h * h' * r) = real_of_int (h * ((h' * r)
mod k))] (rmod of_int k)"
      by (subst mult.assoc, intro rcong_of_intI cong_mult cong_refl cong_mod_rightI)
    finally have *: "⟨real_of_int (1 * r) / real_of_int k⟩ =
                     ⟨real_of_int (h * ((h' * r) mod k)) / real_of_int
k⟩"
      using 2 by (intro ext dedekind_frac_rcong rcong_divide_modulus)
auto
    have "[real_of_int (h' * r) = real_of_int (h' * r mod k)] (rmod real_of_int
k)"
      by (intro rcong_of_intI cong_refl cong_mod_rightI)
    hence "⟨real_of_int (h' * r) / real_of_int k⟩ = ⟨real_of_int ((h'
* r) mod k) / real_of_int k⟩"
      using 2 by (intro dedekind_frac_rcong rcong_divide_modulus) auto
    thus ?case using *
      by (simp add: mult_ac)
  qed auto
  also have "... = (∑r = 1..<k. ⟨real_of_int (h * r) / real_of_int k⟩
* ⟨real_of_int r / real_of_int k⟩)"
    by (rule sum.reindex_bij_betw [OF bij_betw_int_remainders_mult]) fact
  also have "... = dedekind_sum h k"
    using 1 by (simp add: dedekind_sum_altdef mult_ac)
  finally show ?thesis ..
qed

theorem dedekind_sum_inverse':
  assumes "[h * h' = -1] (mod k)"
  shows   "dedekind_sum h k = -dedekind_sum h' k"
proof -
  have "[-(h * h') = - (-1)] (mod k)"
    using assms unfolding cong_minus_minus_iff .
  hence 1: "[h * -h' = 1] (mod k)"
    by simp
  hence "[h' * (-h) = 1] (mod k)"
    by (simp add: mult_ac)
  hence 2: "coprime h' k"
    using assms coprime_iff_invertible_int by blast
  from 1 have "dedekind_sum h k = dedekind_sum (-h') k"
    by (intro dedekind_sum_inverse)
  thus ?thesis
    using 2 by (simp add: dedekind_sum_negate)
qed

theorem dedekind_sum_eq_zero:
  assumes "[h² + 1 = 0] (mod k)"
  shows   "dedekind_sum h k = 0"
proof -
  have "[h * h = h ^ 2 + 1 - 1] (mod k)"
    by (simp add: algebra_simps power2_eq_square)
  also have "[h ^ 2 + 1 - 1 = 0 - 1] (mod k)"
    by (intro cong_diff assms cong_refl)
  finally have "[h * h = -1] (mod k)"
    by simp
  hence "dedekind_sum h k = -dedekind_sum h k"
    by (rule dedekind_sum_inverse')
  thus ?thesis
    by simp
qed
```

1.3 The Reciprocity Law

```
theorem sum_of_powers':
  "(∑k<n::nat. (real k) ^ m) = (bernpoly (Suc m) n - bernpoly (Suc m)
0) / (m + 1)"
proof (cases "n = 0")
  case True
  thus ?thesis
    by auto
next
  case False
  hence "{..<n} = {..n-1}"
    by auto
  thus ?thesis
    using sum_of_powers[of m "n - 1"] False by (simp add: of_nat_diff)
qed

theorem sum_of_powers'_int:
  assumes "n ≥ 0"
  shows   "(∑k=0..<n::int. real_of_int k ^ m) = (bernpoly (Suc m) n
- bernpoly (Suc m) 0) / (m + 1)"
proof -
  have "(∑k=0..<n::int. real_of_int k ^ m) = (∑k<nat n::nat. real k
^ m)"
    by (intro sum.reindex_bij_witness[of _ int nat]) auto
  also have "(∑k<nat n::nat. real k ^ m) = (bernpoly (Suc m) n - bernpoly
(Suc m) 0) / (m + 1)"
    using assms by (subst sum_of_powers') (auto)
  finally show ?thesis by simp
qed

theorem sum_of_powers'_int_from_1:
  assumes "n ≥ 0" "m > 0"
  shows   "(∑k=1..<n::int. real_of_int k ^ m) = (bernpoly (Suc m) n
- bernpoly (Suc m) 0) / (m + 1)"
proof -
  have "(∑k=1..<n::int. real_of_int k ^ m) = (∑k=0..<n::int. real_of_int
k ^ m)"
    using assms by (intro sum.mono_neutral_left) auto
  also have "... = (bernpoly (Suc m) n - bernpoly (Suc m) 0) / (m + 1)"
    using assms by (subst sum_of_powers'_int) auto
  finally show ?thesis by simp
qed


theorem dedekind_sum_reciprocity:
  assumes "h > 0" and "k > 0" and "coprime h k"
  shows "12 * h * k * dedekind_sum h k + 12 * k * h * dedekind_sum k h
=
         h^2 + k^2 - 3 * h * k + 1"
```

```
proof -
  have upto_3_eq [simp]: "{..(3::nat)} = {0, 1, 2, 3}"
    by auto
  have [simp]: "(3 choose 2) = 3"
    by (simp add: eval_nat_numeral)

  define S where "S = (∑r∈{1..<k}. ⟨of_int (h*r) / of_int k⟩ ^ 2)"
  define T where "T = (∑r = 1..<k. ⌊real_of_int (h * r) / k⌋ * (⌊real_of_int
(h * r) / k⌋ + 1))"

  have "S = (∑r∈{1..<k}. ⟨of_int ((h * r) mod k) / of_int k⟩ ^ 2)"
    unfolding S_def
    by (intro sum.cong arg_cong[of _ _ "λx. x ^ 2"] dedekind_frac_mod
[symmetric] refl)
  also have "... = (∑r∈{1..<k}. ⟨of_int r / of_int k⟩ ^ 2)"
    by (rule sum.reindex_bij_betw[OF bij_betw_int_remainders_mult]) fact
  also have "... = (∑r∈{1..<k}. (of_int r / of_int k - 1 / 2) ^ 2)"
  proof (rule sum.cong, goal_cases)
    case (2 r)
    have "¬k dvd r"
      using 2 by (auto dest!: zdvd_imp_le)
    hence "of_int r / real_of_int k ∉ ℤ"
      using 2 by (subst of_int_div_of_int_in_Ints_iff) auto
    moreover have "frac (of_int r / real_of_int k) = of_int r / of_int
k"
      using 2 by (subst frac_eq) auto
    ultimately show ?case
      by (simp add: dedekind_frac_def)
  qed auto
  also have "... = 1 / k^2 * (∑r=1..<k. of_int r ^ 2) -
                   1 / k * (∑r=1..<k. of_int r) +
                   (∑r=1..<k. 1 / 4)"
    unfolding sum_distrib_left sum_subtractf [symmetric] sum.distrib [symmetric]
dedekind_sum_def
    by (intro sum.cong) (auto simp: field_simps power2_eq_square)
  finally have "S = ..." .

  note this [symmetric]
  also have "S = (∑r = 1..<k. (frac (real_of_int (h * r) / real_of_int
k) - 1 / 2)^2)"
    unfolding S_def
  proof (rule sum.cong, goal_cases)
    case (2 r)
    have "¬k dvd r"
      using 2 by (auto dest!: zdvd_imp_le)
    hence "¬k dvd h * r"
      using assms(3) coprime_commute coprime_dvd_mult_right_iff by blast
    hence "of_int (h * r) / real_of_int k ∉ ℤ"
      using 2 by (subst of_int_div_of_int_in_Ints_iff) auto
    thus ?case
      by (simp add: dedekind_frac_def)
  qed auto
  also have "... =
    2 * h * dedekind_sum h k +
    (∑r=1..<k. real_of_int (⌊of_int(h*r)/k⌋ * (⌊h*r/k⌋ + 1))) -
    h^2/k^2 * (∑r=1..<k. real_of_int r ^ 2) +
    (∑r=1..<k. 1/4)"
    unfolding sum_distrib_left sum_subtractf [symmetric] sum.distrib [symmetric]
dedekind_sum_def
    by (intro sum.cong) (auto simp: field_simps power2_eq_square frac_def)
  finally have "2 * h * dedekind_sum h k =
                 -(∑r=1..<k. real_of_int (⌊of_int(h*r)/k⌋ * (⌊h*r/k⌋
+ 1))) +
                 (h^2 + 1)/k^2 * (∑r=1..<k. real_of_int r ^ 2) - 1/k *
(∑r=1..<k. of_int r)"
    by (simp add: add_divide_distrib ring_distribs)
  hence "6 * k * (2 * h * dedekind_sum h k) = 6 * k * (
          -(∑r=1..<k. real_of_int (⌊of_int(h*r)/k⌋ * (⌊h*r/k⌋ + 1)))
+
           (h^2 + 1)/k^2 * (∑r=1..<k. real_of_int r ^ 2) - 1/k * (∑r=1..<k.
of_int r))"
    by (simp only: )
  also have "... = -6 * k * (∑r=1..<k. real_of_int (⌊of_int(h*r)/k⌋ *
(⌊h*r/k⌋ + 1))) +
                  6 * (h^2 + 1)/k * (∑r=1..<k. real_of_int r ^ 2) -
                  6 * (∑r=1..<k. real_of_int r)"
    using ‹k > 0› by (simp add: field_simps power2_eq_square)
  also have "(∑v=1..<k. real_of_int v) = k ^ 2 / 2 - k / 2"
    using sum_of_powers'_int_from_1[of k 1] assms
    by (simp add: bernpoly_def algebra_simps power2_eq_square)
  also have "6 * ... = 3 * k^2 - 3 *k"
    using assms by (simp add: field_simps)
  also have "(∑v=1..<k. real_of_int v ^ 2) = k ^ 3 / 3 - k ^ 2 / 2 +
k / 6"
    using assms by (subst sum_of_powers'_int_from_1) (auto simp: bernpoly_def)
  also have "real_of_int (6 * (h^2 + 1)) / real_of_int k * ... =
            2 * h^2 * k^2 + 2 * k^2 - 3 * h^2 * k - 3 * k + h^2 + 1"
    using assms by (simp add: field_simps power3_eq_cube power2_eq_square)
  finally have sum_eq1: "12 * h * k * dedekind_sum h k =
    -6 * k * T + 2 * h^2 * k^2 + 2 * k^2 - 3 * h^2 * k - 3 * k + h^2 + 1 -
3 * k^2 + 3 * k"
    by (simp add: mult_ac T_def)

  define N where "N = (λv. card {r∈{1..<k}. ⌊real_of_int (h*r)/k⌋ = v
- 1})"
  have N_eq_aux: "{r∈{1..<k}. ⌊real_of_int (h*r)/k⌋ = v - 1} =
                  {1..<k} ∩ {⌊k * (v - 1) / h⌋<..⌊k * v / h⌋}" for v
  proof -
```

```
    have "⌊real_of_int (h*r)/k⌋ = v - 1 ⟷ r ∈ {⌊k * (v - 1) / h⌋<..⌊k
* v / h⌋}"
      if r: "r ∈ {1..<k}" for r
    proof -
      have neq: "real_of_int r ≠ k * v / h" for v
      proof
        assume "real_of_int r = k * v / h"
        hence "real_of_int (r * h) = real_of_int (k * v)"
          using assms unfolding of_int_mult by (simp add: field_simps)
        hence *: "r * h = k * v"
          by linarith
        have "k dvd r * h"
          by (subst *) auto
        with ‹coprime h k› have "k dvd r"
          by (simp add: coprime_commute coprime_dvd_mult_left_iff)
        with r show False
          by (auto dest!: zdvd_imp_le)
      qed

      have "⌊real_of_int (h*r)/k⌋ = v - 1 ⟷ h * r / k ∈ {v-1..<v}"
        unfolding atLeastLessThan_iff by linarith
      also have "... ⟷ r ∈ {k * (v - 1) / h..<k * v / h}"
        using assms by (auto simp: field_simps)
      also have "... ⟷ r ∈ {k * (v - 1) / h<..k * v / h}"
        using neq[of "v - 1"] neq[of v] by auto
      also have "... ⟷ r ∈ {⌊k * (v - 1) / h⌋<..⌊k * v / h⌋}"
        unfolding greaterThanAtMost_iff by safe linarith+
      finally show "⌊real_of_int (h*r)/k⌋ = v - 1 ⟷ r ∈ {⌊k * (v -
1) / h⌋<..⌊k * v / h⌋}" .
    qed
    thus "{r∈{1..<k}. ⌊real_of_int (h*r)/k⌋ = v - 1} =
           {1..<k} ∩ {⌊k * (v - 1) / h⌋<..⌊k * v / h⌋}"
      by blast
  qed

  define N' where "N' = (λv. if v = h then k - 1 else ⌊k * v / h⌋)"

  have N_eq: "int (N v) = N' v - N' (v - 1)" if v: "v ∈ {1..h}" for v
  proof (cases "v = h")
    case False
    have le: "⌊real_of_int k * (real_of_int v - 1) / real_of_int h⌋
              ≤ ⌊real_of_int k * real_of_int v / real_of_int h⌋"
      using assms by (intro floor_mono divide_right_mono mult_left_mono)
auto
    have "N v = card ({1..<k} ∩ {⌊k * (v - 1) / h⌋<..⌊k * v / h⌋})"
      unfolding N_def N_eq_aux ..
    also have "{⌊k * (v - 1) / h⌋<..⌊k * v / h⌋} ⊆ {1..<k}"
    proof -
      have "⌊k * (v - 1) / h⌋ ≥ 0"
```

```
        using v ‹k > 0› ‹h > 0› by auto
      moreover have "⌊k * v / h⌋ < k"
        using v False ‹k > 0› ‹h > 0› by (subst floor_less_iff) (auto
simp: field_simps)
      ultimately have "{⌊k * (v - 1) / h⌋<..⌊k * v / h⌋} ⊆ {0<..k-1}"
        unfolding Ioc_subset_iff by auto
      also have "{0<..k-1} = {1..<k}"
        by force
      finally show ?thesis .
    qed
    hence "{1..<k} ∩ {⌊k * (v - 1) / h⌋<..⌊k * v / h⌋} = {⌊k * (v - 1)
/ h⌋<..⌊k * v / h⌋}"
      by blast
    finally show ?thesis
      using le v False by (simp add: N'_def)
  next
    case [simp]: True
    have le: "⌊real_of_int k * (real_of_int h - 1) / real_of_int h⌋ ≤
k - 1"
      using assms by (subst floor_le_iff) (auto simp: field_simps)
    have "N h = card ({1..<k} ∩ {⌊k * (h - 1) / h⌋<..⌊k * h / h⌋})"
      unfolding N_def N_eq_aux ..
    also have "⌊k * h / h⌋ = k"
      using assms by simp
    also have "{1..<k} ∩ {⌊real_of_int (k * (h - 1)) / real_of_int h⌋<..k}
=
               {⌊real_of_int (k * (h - 1)) / real_of_int h⌋+1..<k}"
    proof -
      have nonneg: "⌊real_of_int k * (real_of_int h - 1) / real_of_int
h⌋ ≥ 0"
        unfolding of_int_mult using assms by auto
      have "{1..<k} ∩ {⌊real_of_int (k * (h - 1)) / real_of_int h⌋+1..<k+1}
=
             {max 1 (⌊real_of_int k * (real_of_int h - 1) / real_of_int
h⌋+1)..<k}"
        by simp
      also have "max 1 (⌊real_of_int k * (real_of_int h - 1) / real_of_int
h⌋+1) =
                ⌊real_of_int k * (real_of_int h - 1) / real_of_int h⌋
+ 1"
        using nonneg by auto
      also have "{⌊real_of_int (k * (h - 1)) / real_of_int h⌋+1..<k+1}
=
                {⌊real_of_int (k * (h - 1)) / real_of_int h⌋<..k}"
        by force
      finally show ?thesis by simp
    qed
    finally show ?thesis
      using le by (simp add: N'_def)
  qed

  have "T = (∑(v,r) ∈ (SIGMA v:{1..h}. {r∈{1..<k}. ⌊of_int (h*r)/k⌋
= v - 1}). (v - 1) * v)"
    unfolding T_def
  proof (intro sum.reindex_bij_witness[of _ snd "λr. (⌊of_int (h*r)/k⌋
+ 1, r)"], goal_cases)
    case (2 r)
    have "⌊real_of_int h * real_of_int r / real_of_int k⌋ < h"
      using 2 assms by (subst floor_less_iff) (auto simp: field_simps)
    thus ?case using 2 assms by auto
  qed (use assms in auto)
  also have "... = (∑v=1..h. ∑r|r∈{1..<k} ∧ ⌊real_of_int(h*r)/k⌋ =
v - 1. (v - 1) * v)"
  proof (intro sum.Sigma [symmetric] ballI)
    show "finite {r ∈ {1..<k}. ⌊real_of_int (h * r) / real_of_int k⌋
= v - 1}" for v
      by (rule finite_subset[of _ "{1..<k}"]) auto
  qed auto
  also have "... = (∑v=1..h. (v - 1) * v * int (N v))"
    by (simp add: N_def mult_ac)
  also have "... = (∑v=1..h. (v - 1) * v * (N’ v - N’ (v - 1)))"
    by (intro sum.cong) (auto simp: N_eq)
  also have "... = (∑v=1..h. (v - 1) * v * N’ v) - (∑v=1..h. (v - 1)
* v * N’ (v - 1))"
    by (simp add: ring_distribs sum_subtractf)
  also have "(∑v=1..h. (v - 1) * v * N’ (v - 1)) = (∑v=0..<h. (v +
1) * v * N’ v)"
    by (rule sum.reindex_bij_witness[of _ "λv. v+1" "λv. v-1"]) auto
  also have "... = (∑v=1..<h. (v + 1) * v * N’ v)"
    by (intro sum.mono_neutral_right) auto
  also have "{1..h} = insert h {1..<h}"
    using assms by auto
  also have "(∑v∈.... (v - 1) * v * N’ v) =
               (∑v = 1..<h. (v - 1) * v * N’ v) + (h - 1) * h * N’ h"
    by (subst sum.insert) auto
  also have "(∑v=1..<h. (v-1) * v * N’ v) + (h-1) * h * N’ h - (∑v=1..<h.
(v+1) * v * N’ v) =
             (h-1) * h * N’ h - 2 * (∑v=1..<h. v * N’ v)"
    by (simp add: ring_distribs sum_subtractf sum.distrib sum_distrib_left
sum_negf mult_ac)
  also have "(∑v = 1..<h. v * N’ v) = (∑v = 1..<h. v * ⌊of_int (k *
v)/h⌋)"
    by (intro sum.cong) (auto simp: N’_def)
  also have "N’ h = k - 1"
    by (simp add: N’_def)
  finally have *: "- 2 * (∑v = 1..<h. v * ⌊of_int (k * v)/h⌋) =
                   -(h - 1) * h * (k - 1) + T"
    by linarith
  have "12 * k * h * dedekind_sum k h =
         6 * k * (-2 * (∑v=1..<h. v * ⌊real_of_int (k * v)/h⌋)) +
          12*k^2/h * (∑v=1..<h. real_of_int v ^ 2) -
          6 * k * (∑v=1..<h. real_of_int v)"
    by (simp add: dedekind_sum_def algebra_simps power2_eq_square sum.distrib
sum_subtractf
               sum_distrib_left sum_distrib_right sum_negf frac_def
sum_divide_distrib)
  also note *
  also have "real_of_int (6 * k * (- (h - 1) * h * (k - 1) + T)) =
           6 * k * T - 6 * h^2 * k^2 + 6 * h * k^2 + 6 * h^2 * k - 6 * h
* k"
    by (simp add: algebra_simps power2_eq_square)
  also have "(∑v=1..<h. real_of_int v) = h ^ 2 / 2 - h / 2"
    using sum_of_powers'_int_from_1[of h 1] assms
    by (simp add: bernpoly_def algebra_simps power2_eq_square)
  also have "real_of_int (6 * k) * ... = 3 * h^2 * k - 3 * h * k"
    using assms by (simp add: field_simps)
  also have "(∑v=1..<h. real_of_int v ^ 2) = h ^ 3 / 3 - h ^ 2 / 2 +
h / 6"
    using assms by (subst sum_of_powers'_int_from_1) (auto simp: bernpoly_def)
  also have "real_of_int (12 * k^2) / real_of_int h * ... =
           4 * h^2 * k^2 - 6 * h * k^2 + 2 * k^2"
    using assms by (simp add: field_simps power3_eq_cube power2_eq_square)
  finally have sum_eq2: "12 * k * h * dedekind_sum k h =
                      6 * k * T - 2 * h^2 * k^2 + 3 * h^2 * k - 3 * h
* k + 2 * k^2"
    by simp


  have "real_of_int (12 * h * k) * dedekind_sum h k + real_of_int (12
* k * h) * dedekind_sum k h =
       real_of_int (h^2 - 3 * h * k + k^2 + 1)"
    unfolding sum_eq1 sum_eq2 by simp
  thus ?thesis
    by simp
qed

theorem dedekind_sum_reciprocity':
  assumes "h > 0" and "k > 0" and "coprime h k"
  shows "dedekind_sum h k = -dedekind_sum k h + h / k / 12 + k / h / 12
- 1 / 4 + 1 / (12 * h * k)"
  using dedekind_sum_reciprocity[OF assms] assms
  by (auto simp: field_simps power2_eq_square)
```

1.4 Congruence Properties

```
definition dedekind_sum' :: "int ⇒ int ⇒ int" where
```

```
  "dedekind_sum' h k = ⌊6 * real_of_int k * dedekind_sum h k⌋"

lemma dedekind_sum'_cong:
  "[h = h'] (mod k) ⟹ coprime h k ∨ coprime h' k ⟹ dedekind_sum'
h k = dedekind_sum' h' k"
  unfolding dedekind_sum'_def by (subst dedekind_sum_cong[of h h' k])
auto


lemma
  assumes "k > 0"
  shows   of_int_dedekind_sum':
            "real_of_int (dedekind_sum' h k) = 6 * real_of_int k * dedekind_sum
h k"
    and   dedekind_sum'_altdef:
            "dedekind_sum' h k = h * (k - 1) * (2 * k - 1) -
               6 * (∑r = 1..<k. r * ⌊of_int (h * r) / k⌋) - 3 * (k *
(k - 1) div 2)"
    and   dedekind_sum'_cong_3: "[dedekind_sum' h k = h * (k - 1) * (2
* k - 1)] (mod 3)"
proof -
  have [simp]: "{..3} = {0,1,2,3::nat}"
    by auto
  have [simp]: "(3 choose 2) = 3"
    by (auto simp: eval_nat_numeral)
  have "6 * k * dedekind_sum h k = 6 * h / k * (∑r=1..<k. of_int r ^
2) -
          6 * (∑r=1..<k. of_int r * ⌊h*r/k⌋) - 3 * (∑r=1..<k. of_int
r ^ 1)"
    by (simp add: dedekind_sum_def frac_def algebra_simps sum_distrib_left
sum_distrib_right
                  sum_subtractf sum.distrib sum_divide_distrib power2_eq_square)
  also have "6 * h / k * (∑r=1..<k. real_of_int r ^ 2) = 2 * h * k^2 +
h - 3 * h * k"
    using assms by (subst sum_of_powers'_int_from_1)
                   (auto simp: bernpoly_def field_simps power2_eq_square
power3_eq_cube)
  also have "3 * (∑r=1..<k. real_of_int r ^ 1) = 3 * (k * (k - 1) / 2)"
    using assms by (subst sum_of_powers'_int_from_1)
                   (auto simp: bernpoly_def field_simps power2_eq_square)
  also have "even (k * (k - 1))"
    by auto
  hence "(k * (k - 1) / 2) = real_of_int ((k * (k - 1)) div 2)"
    by fastforce
  also have "2 * h * k^2 + h - 3 * h * k = h * (k - 1) * (2 * k - 1)"
    by (simp add: algebra_simps power2_eq_square)
  finally have eq: "6 * real_of_int k * dedekind_sum h k =
                    real_of_int (h * (k - 1) * (2 * k - 1) -
                       6 * (∑r = 1..<k. r * ⌊of_int (h * r) / k⌋) - 3
* (k * (k - 1) div 2))"
    unfolding of_int_diff of_int_add by simp

  have "6 * real_of_int k * dedekind_sum h k ∈ ℤ"
    unfolding eq by (rule Ints_of_int)
  thus "real_of_int (dedekind_sum' h k) = 6 * of_int k * dedekind_sum
h k"
    unfolding dedekind_sum'_def by (auto elim!: Ints_cases)
  show eq': "dedekind_sum' h k = h * (k - 1) * (2 * k - 1) -
             6 * (∑r = 1..<k. r * ⌊of_int (h * r) / k⌋) - 3 * (k *
(k - 1) div 2)"
    unfolding dedekind_sum'_def eq by (simp only: floor_of_int)
  have "[dedekind_sum' h k = h * (k - 1) * (2 * k - 1) - 0 - 0] (mod 3)"
    unfolding eq' by (intro cong_diff cong_refl) (auto simp: Cong.cong_def)
  thus"[dedekind_sum' h k = h * (k - 1) * (2 * k - 1)] (mod 3)"
    by simp
qed

lemma three_dvd_dedekind_sum'_iff_aux:
  fixes h k :: int
  defines "ϑ ≡ gcd 3 k"
  assumes "k > 0" "coprime h k"
  shows   "3 dvd (2 * dedekind_sum' h k) ⟷ ¬3 dvd k"
proof -
  have "[2 * dedekind_sum' h k = 2 * (h * (k - 1) * (2 * k - 1))] (mod
3)"
    by (intro cong_mult dedekind_sum'_cong_3 cong_refl assms)
  also have "2 * (h * (k - 1) * (2 * k - 1)) = h * (k - 1) * (4 * k +
(-2))"
    by (simp add: algebra_simps)
  also have "[... = h * (k - 1) * (1 * k + 1)] (mod 3)"
    by (intro cong_mult cong_refl cong_add cong_diff) (auto simp: Cong.cong_def)
  finally have cong: "[2 * dedekind_sum' h k = h * (k - 1) * (k + 1)] (mod
3)"
    by simp

  show "3 dvd (2 * dedekind_sum' h k) ⟷ ¬3 dvd k"
  proof (cases "3 dvd k")
    case True
    have "¬3 dvd h"
      using ‹coprime h k› True by fastforce
    have "[2 * dedekind_sum' h k = h * (k - 1) * (k + 1)] (mod 3)"
      by (fact cong)
    also have "[h * (k - 1) * (k + 1) = h * (0 - 1) * (0 + 1)] (mod 3)"
      using True by (intro cong_mult cong_diff cong_add cong_refl) (auto
simp: cong_0_iff)
    also have "h * (0 - 1) * (0 + 1) = -h"
      by simp
    finally have "3 dvd (2 * dedekind_sum' h k) ⟷ 3 dvd (-h)"
```

```
      using cong_dvd_iff by blast
    with ‹¬3 dvd h› True show ?thesis by auto
  next
    case False
    hence "3 dvd (k + 1) ∨ 3 dvd (k - 1)"
      by presburger
    hence "3 dvd (h * (k - 1) * (k + 1))"
      by force
    also have "?this ⟷ 3 dvd (2 * dedekind_sum' h k)"
      using cong cong_dvd_iff by blast
    finally show ?thesis
      using False by auto
  qed
qed


lemma dedekind_sum'_reciprocity:
  fixes h k :: int
  assumes "h > 0" "k > 0" "coprime h k"
  shows "2 * h * dedekind_sum' h k = -2 * k * dedekind_sum' k h + h^2
+ k^2 - 3 * h * k + 1"
proof -
  have "real_of_int (2 * h * dedekind_sum' h k) = 12 * h * k * dedekind_sum
h k"
    unfolding of_int_mult of_int_dedekind_sum'[OF assms(2)] by simp
  also have "... = real_of_int (-12 * k * h) * dedekind_sum k h + real_of_int
(h^2 + k^2 - 3 * h * k + 1)"
    using dedekind_sum_reciprocity[OF assms] by simp
  also have "... = real_of_int (-2 * k * dedekind_sum' k h + h^2 + k^2 -
3 * h * k + 1)"
    using of_int_dedekind_sum'[OF assms(1)] by simp
  finally show ?thesis by linarith
qed

lemma cong_dedekind_sum'_1:
  fixes h k :: int
  defines "ϑ ≡ gcd 3 k"
  assumes "h > 0" "coprime h k"
  shows "[2 * k * dedekind_sum' k h = 0] (mod ϑ * k)"
proof -
  have "ϑ = 1 ∨ ϑ = 3"
    using gcd_prime_int[of 3 k] unfolding ϑ_def by auto
  thus ?thesis
  proof
    assume "ϑ = 3"
    hence "3 dvd k"
      unfolding ϑ_def by (metis gcd_dvd2)
    with ‹coprime h k› have "¬3 dvd h"
      by force
    have "3 dvd 2 * dedekind_sum' k h"
      using assms ‹¬3 dvd h›
      by (subst three_dvd_dedekind_sum'_iff_aux) (auto simp: coprime_commute)
    hence "3 * k dvd 2 * dedekind_sum' k h * k"
      by auto
    thus ?thesis
      using ‹ϑ = 3› by (simp add: cong_0_iff)
  qed (auto simp: cong_0_iff)
qed

lemma cong_dedekind_sum'_2_aux:
  fixes h k :: int
  defines "ϑ ≡ gcd 3 k"
  assumes "h > 0" "k > 0" "coprime h k"
  shows "[2 * h * dedekind_sum' h k = h^2 + 1] (mod ϑ * k)"
proof -
  have "[-(2 * k * dedekind_sum' k h) + h^2 + k^2 - 3 * h * k + 1 = -0 +
h^2 + 0 - 0 + 1] (mod ϑ * k)"
    unfolding ϑ_def
    by (intro cong_diff cong_add cong_mult cong_refl cong_uminus cong_dedekind_sum'_1
assms)
       (simp_all add: cong_0_iff power2_eq_square)
  also have "-(2 * k * dedekind_sum' k h) + h^2 + k^2 - 3 * h * k + 1 =
2 * h * dedekind_sum' h k"
    using dedekind_sum'_reciprocity[of k h] assms by (auto simp: coprime_commute)
  finally show ?thesis by simp
qed

lemma dedekind_sum'_negate:
  assumes "k > 0" "coprime h k"
  shows   "dedekind_sum' (-h) k = -dedekind_sum' h k"
proof -
  have "real_of_int (dedekind_sum' (-h) k) = real_of_int (-dedekind_sum'
h k)"
    using assms unfolding of_int_minus
    by (subst (1 2) of_int_dedekind_sum') (auto simp: dedekind_sum_negate)
  thus ?thesis
    by linarith
qed

lemma cong_dedekind_sum'_2:
  fixes h k :: int
  defines "ϑ ≡ gcd 3 k"
  assumes "k > 0" "coprime h k"
  shows "[2 * h * dedekind_sum' h k = h^2 + 1] (mod ϑ * k)"
proof (cases h "0 :: int" rule: linorder_cases)
  case greater
  thus ?thesis
    using cong_dedekind_sum'_2_aux[of h k] assms by auto
```

```
next
  case less
  thus ?thesis
    using cong_dedekind_sum'_2_aux[of "-h" k] assms by (auto simp: dedekind_sum'_negate)
next
  case equal
  thus ?thesis using assms by (auto simp: Cong.cong_def)
qed



theorem dedekind_sum'_cong_8:
  assumes "k > 0" "coprime h k"
  shows "[2 * dedekind_sum' h k =
          (k-1)*(k+2) - 4*h*(k-1) + 4*(∑r∈{1..<(k+1) div 2}. ⌊2*h*r/k⌋)]
(mod 8)"
proof -
  define S1 where "S1 = (∑r=1..<k. r * ⌊of_int (h * r) / k⌋)"
  define S2 where "S2 = (∑r | r ∈ {1..<k} ∧ odd r. ⌊of_int (h * r) /
k⌋)"
  define S3 where "S3 = (∑r=1..<k. ⌊of_int (h * r) / k⌋)"
  define S4 where "S4 = (∑r=1..<(k+1) div 2. ⌊of_int (2 * h * r) / k⌋)"
  have "4 * 2 dvd 4 * (k * (k - 1))"
    by (intro mult_dvd_mono) auto
  hence dvd: "8 dvd 4 * k * (k - 1)"
    by (simp add: mult_ac)

  have "[4 * S1 = 4 * (∑r=1..<k. if even r then 0 else ⌊of_int (h *
r) / k⌋)] (mod 8)"
    unfolding S1_def sum_distrib_left
  proof (intro cong_sum, goal_cases)
    case (1 r)
    show ?case
    proof (cases "odd r")
      case True
      have *: "4 * r mod 8 = 4"
        using ‹odd r› by presburger
      have "[4 * r * ⌊real_of_int (h * r) / k⌋ = 4 * ⌊real_of_int (h *
r) / k⌋] (mod 8)"
        by (rule cong_mult[OF _ cong_refl]) (use * in ‹auto simp: Cong.cong_def›)
      thus ?thesis
        using True by (simp add: mult_ac)
    qed (auto simp: cong_0_iff)
  qed
  also have "(∑r=1..<k. if even r then 0 else ⌊of_int (h * r) / k⌋) =
S2"
    unfolding S2_def by (intro sum.mono_neutral_cong_right) auto
  finally have S12: "[4 * S1 = 4 * S2] (mod 8)" .
  have "2 * dedekind_sum' h k =
          2 * (h * (k - 1) * (2 * k - 1)) - 12 * S1 - 6 * (k * (k - 1)
div 2)"
    using assms by (subst dedekind_sum'_altdef) (auto simp: S1_def)
  also have "6 * (k * (k - 1) div 2) = 3 * k * (k - 1)"
    by (subst div_mult_swap) auto
  also have "2 * (h * (k - 1) * (2 * k - 1)) - 12 * S1 - 3 * k * (k -
1) =
             -2 * h * (k - 1) + h * (4 * k * (k - 1)) - 12 * S1 + k *
(k - 1) - 4 * k * (k - 1)"
    by (simp add: algebra_simps)
  also have "[... = -2 * h * (k - 1) + h * 0 - 4 * S1 + k * (k - 1) - 0]
(mod 8)"
    using dvd by (intro cong_diff cong_add S12 cong_mult cong_refl)
                 (auto simp: Cong.cong_def mod_eq_0_iff_dvd)
  also have "-2 * h * (k - 1) + h * 0 - 4 * S1 + k * (k - 1) - 0 = (k
- 1) * (k - 2 * h) - 4 * S1"
    by (simp add: algebra_simps)
  also have "[... = (k - 1) * (k - 2 * h) - 4 * S2] (mod 8)"
    by (intro cong_diff cong_refl S12)
  also have "S2 = S3 - S4"
  proof -
    have *: "{r. r ∈ {1..<k} ∧ odd r} = {1..<k} - {r. r ∈ {1..<k} ∧
even r}"
      by auto
    have "S2 = S3 - (∑r | r ∈ {1..<k} ∧ even r. ⌊of_int (h * r) / k⌋)"
      unfolding S2_def S3_def * by (subst Groups_Big.sum_diff) auto
    also have "(∑r | r ∈ {1..<k} ∧ even r. ⌊of_int (h * r) / k⌋) = S4"
      unfolding S4_def using ‹k > 0›
      by (intro sum.reindex_bij_witness[of _ "λr. 2 * r" "λr. r div 2"])
         (auto simp: real_of_int_div)
    finally show ?thesis .
  qed
  also have "4 * (S3 - S4) = 4 * S3 - 4 * S4"
    by (simp add: algebra_simps)
  also have "4 * S3 = 2 * (h - 1) * (k - 1)"
  proof -
    have "real_of_int S3 = (∑r=1..<k. -⟨of_int (h * r) / k⟩ + of_int
(h * r) / k - 1 / 2)"
      unfolding S3_def of_int_sum
    proof (intro sum.cong)
      fix r assume r: "r ∈ {1..<k}"
      hence "¬k dvd r"
        by (auto dest!: zdvd_imp_le)
      hence "¬k dvd (h * r)"
        using assms coprime_commute coprime_dvd_mult_right_iff by blast
      hence "real_of_int (h * r) / real_of_int k ∉ ℤ"
        using assms by (subst of_int_div_of_int_in_Ints_iff) auto
      thus "real_of_int ⌊real_of_int (h * r) / real_of_int k⌋ =
               -⟨real_of_int (h * r) / real_of_int k⟩ + real_of_int (h
* r) / real_of_int k - 1 / 2"
        by (simp add: dedekind_frac_def frac_def)
    qed auto
    also have "4 * ... = -4 * (∑r=1..<k. ⟨of_int (h * r) / k⟩) +
                          4 * h / k * (∑r=1..<k. of_int r ^ 1) - ((real_of_int
k * 4 - 4) / 2)"
      using assms
      by (simp add: sum.distrib sum_subtractf sum_negf of_nat_diff mult_ac

                    sum_distrib_left sum_distrib_right sum_divide_distrib)
    also have "(∑r=1..<k. ⟨of_int (h * r) / k⟩) = 0"
      using assms by (intro sum_dedekind_frac_mult_eq_0)
    also have "4 * h / k * (∑r=1..<k. real_of_int r ^ 1) = 2 * h * (k
- 1)"
      using assms by (subst sum_of_powers'_int_from_1) (auto simp: field_simps
bernpoly_def)
    also have "-4 * 0 + real_of_int (2 * h * (k - 1)) - (real_of_int k
* 4 - 4) / 2 =
               real_of_int (2 * (h - 1) * (k - 1))"
      by (simp add: field_simps)
    also have "4 * real_of_int S3 = real_of_int (4 * S3)"
      by simp
    finally show ?thesis by linarith
  qed
  also have "(k - 1) * (k - 2 * h) - (2 * (h - 1) * (k - 1) - 4 * S4)
=
             (k - 1) * (k + 2) - 4 * h * (k - 1) + 4 * S4"
    by (simp add: algebra_simps)
  finally show "[2 * dedekind_sum' h k = (k-1)*(k+2) - 4*h*(k-1) + 4*S4]
(mod 8)"
    unfolding S4_def .
qed

theorem dedekind_sum'_cong_8_odd:
  assumes "k > 0" "coprime h k" "odd k"
  shows "[2 * dedekind_sum' h k =
          k - 1 + 4*(∑r∈{1..<(k+1) div 2}. ⌊2*h*r/k⌋)] (mod 8)"
proof -
  define S where "S = (∑r=1..<(k+1) div 2. ⌊of_int (2 * h * r) / k⌋)"

  have 1: "[(k-1)*(k+2) = k - 1] (mod 8)"
  proof -
    from assms obtain k' where k': "k = 2 * k' + 1"
      by (elim oddE)
    define k'' where "k'' = k' mod 4"
    have "k'' ∈ {0..<4}"
      unfolding k''_def by simp
    also have "{0..<4} = {0, 1, 2, 3::int}"
```

```
      by auto
    finally have "(2 * k'' + 1) ^ 2 mod 8 = 1"
      by auto

    have "[k ^ 2 = ((2 * k') mod (2 * 4) + 1) ^ 2 mod 8] (mod 8)"
      unfolding k' by (intro cong_add cong_refl cong_pow cong_mod_rightI)
(auto simp: Cong.cong_def)
    also have "(2 * k') mod (2 * 4) = 2 * k''"
      by (subst mod_mult_mult1) (auto simp: k''_def)
    also have "(2 * k'' + 1) ^ 2 mod 8 = 1"
      by fact
    finally have *: "[k ^ 2 = 1] (mod 8)" .

    have "(k-1)*(k+2) = k^2 + k - 2"
      by (simp add: algebra_simps power2_eq_square)
    also have "[... = 1 + k - 2] (mod 8)"
      by (intro cong_add cong_refl cong_diff *)
    finally show ?thesis
      by simp
  qed

  have 2: "[4 * h * (k - 1) = 0] (mod 8)"
  proof -
    have "4 * 1 * 2 dvd 4 * h * (k - 1)"
      using assms by (intro mult_dvd_mono) auto
    thus ?thesis by (simp add: cong_0_iff)
  qed

  have "[2 * dedekind_sum' h k = (k-1)*(k+2) - 4*h*(k-1) + 4*S] (mod 8)"
    unfolding S_def using assms by (intro dedekind_sum'_cong_8)
  also have "[(k-1)*(k+2) - 4*h*(k-1) + 4*S = (k - 1) - 0 + 4 * S] (mod
8)"
    by (intro cong_add cong_diff cong_refl 1 2)
  finally show ?thesis
    by (simp add: S_def)
qed



lemma dedekind_sum'_cong_power_of_two:
  fixes h k k1 :: int and n :: nat
  assumes "h > 0" "k1 > 0" "odd k1" "n > 0" "k = 2 ^ n * k1" "coprime
h k"
  shows   "[2 * h * dedekind_sum' h k =
             h² + k² + 1 + 5 * k - 4 * k * (∑v=1..<(h+1) div 2. ⌊of_int
(2 * k * v) / h⌋)]
           (mod 2 ^ (n + 3))"
proof -
  from assms have "even k"
```

```
    by auto
  with ‹coprime h k› have "odd h"
    using coprime_common_divisor odd_one by blast
  from assms have "k > 0"
    by auto
  define S where "S = (∑v=1..<(h+1) div 2. ⌊of_int (2 * k * v) / h⌋)"
  have "[2 * dedekind_sum' k h * k = (h - 1 + 4 * S) * k] (mod (8 * k))"
    unfolding S_def using ‹h > 0› ‹k > 0› ‹coprime h k› ‹odd h›
    by (intro dedekind_sum'_cong_8_odd cong_cmult_rightI) (auto simp:
coprime_commute)
  also have "8 * k = 2 ^ (n + 3) * k1"
    using assms by (simp add: power_add)
  finally have *: "[2 * dedekind_sum' k h * k = (h - 1 + 4 * S) * k] (mod
2 ^ (n + 3))"
    using cong_modulus_mult by blast

  have **: "[k - 4 * h * k = 5 * k] (mod 2 ^ (n + 3))"
  proof -
    have "2 ^ 2 * 2 ^ n * 2 dvd 4 * k * (h + 1)"
      using assms by (intro mult_dvd_mono) auto
    hence "2 ^ (n + 3) dvd (5 * k - (k - 4 * h * k))"
      by (simp add: algebra_simps power_add)
    thus ?thesis
      by (subst cong_sym) (auto simp: cong_iff_dvd_diff)
  qed

  have "2 * h * dedekind_sum' h k = h^2 + k^2 - 3 * h * k + 1 - 2 * dedekind_sum'
k h * k"
    using dedekind_sum'_reciprocity[of h k] ‹h > 0› ‹k > 0› ‹coprime h
k› by simp
  also have "[... = h^2 + k^2 - 3 * h * k + 1 - (h - 1 + 4 * S) * k] (mod
2 ^ (n + 3))"
    using * by (intro cong_add cong_diff cong_refl)
  also have "h^2 + k^2 - 3*h*k + 1 - (h - 1 + 4*S) * k = h^2 + k^2 + 1 + (k
- 4*h*k) - 4*k*S"
    by (simp add: algebra_simps)
  also have "[... = h^2 + k^2 + 1 + 5 * k - 4*k*S] (mod 2 ^ (n + 3))"
    by (intro cong_add cong_diff cong_refl **)
  finally show ?thesis
    by (simp add: S_def)
qed

lemma dedekind_sum'_cong_power_of_two':
  fixes h k k1 :: int
  assumes "h > 0" "k > 0" "even k" "coprime h k"
  shows   "[2 * h * dedekind_sum' h k =
              h^2 + k^2 + 1 + 5 * k - 4 * k * (∑v=1..<(h+1) div 2. ⌊of_int
(2 * k * v) / h⌋)]
           (mod 2 ^ (multiplicity 2 k + 3))"
```

```
proof (rule dedekind_sum'_cong_power_of_two)
  define k1 where "k1 = k div 2 ^ multiplicity 2 k"
  have "2 ^ multiplicity 2 k dvd k"
    using multiplicity_dvd by blast
  thus k_eq: "k = 2 ^ multiplicity 2 k * k1"
    by (auto simp: k1_def)
  show "odd k1"
    using ‹k > 0› multiplicity_decompose[of k 2] by (auto simp: k1_def)
  show "multiplicity 2 k > 0"
    using ‹even k› ‹k > 0› by (simp add: multiplicity_gt_zero_iff)
  show "k1 > 0"
    using ‹k > 0› by (subst (asm) k_eq) (auto simp: zero_less_mult_iff)
qed (use assms in auto)
```

```
lemma dedekind_sum_diff_even_int_aux:
  fixes a b c d :: int assumes det: "a * d - b * c = 1"
  fixes q c1 r δ' :: int and δ :: real
  assumes a: "a > 0"
  assumes q: "q ∈ {3, 5, 7, 13}" and "c1 > 0"
  assumes c: "c = q * c1"
  defines "r ≡ 24 div (q - 1)"
  defines "δ' ≡ (2 * dedekind_sum' a c - (a + d)) - (2 * q * dedekind_sum'
a c1 - (a + d) * q)"
  defines "δ ≡ (dedekind_sum a c - (a+d)/(12*c)) - (dedekind_sum a c1
- (a+d)/(12*c1))"
  shows   "of_int δ' = 12 * c * δ" and "24 * c dvd r * δ'"
proof -
  define ϑ where "ϑ = gcd 3 c"
  define ϑ1 where "ϑ1 = gcd 3 c1"
  have "even r" "odd q"
    using q by (auto simp: r_def)

  have "q > 0"
    using q by auto
  have "c > 0"
    using q and ‹c1 > 0› and c by auto
  have "[a * d - b * 0 = a * d - b * c] (mod c)"
    by (intro cong_diff cong_mult cong_refl) (auto simp: Cong.cong_def)
  also have "a * d - b * c = 1"
    by fact
  finally have "[a * d = 1] (mod c)"
    by simp
  hence "coprime a c"
    using coprime_iff_invertible_int by blast
  have "coprime a c1"
    using ‹coprime a c› c by auto
  show of_int_δ': "of_int δ' = 12 * c * δ"
    using ‹c > 0› ‹c1 > 0› ‹q > 0› unfolding δ'_def δ_def of_int_mult
of_int_diff
    by (subst (1 2) of_int_dedekind_sum') (auto simp: field_simps c)

  have "[2 * a * dedekind_sum' a c = a^2 + 1] (mod ϑ * c)"
    using ‹coprime a c› ‹c > 0› unfolding ϑ_def by (intro cong_dedekind_sum'_2)
auto
  hence "[2 * a * dedekind_sum' a c - a * (a + d) = (a^2 + 1) - a * (a
+ d)] (mod ϑ * c)"
    by (intro cong_diff cong_refl)
  also have "(a^2 + 1) - a * (a + d) = -b * c"
    using det unfolding power2_eq_square by (simp add: algebra_simps)
  finally have "[2 * a * dedekind_sum' a c - a * (a + d) = -b * c] (mod
ϑ * c)" .
  hence 1: "[2 * a * dedekind_sum' a c - a * (a + d) = -b * c] (mod ϑ1
* c)"
    by (rule cong_dvd_mono_modulus) (auto simp: ϑ_def ϑ1_def c)

  have "[2 * a * dedekind_sum' a c1 * q = (a^2 + 1) * q] (mod ϑ1 * c1
* q)"
    using ‹coprime a c1› ‹c1 > 0› unfolding ϑ1_def
    by (intro cong_cmult_rightI cong_dedekind_sum'_2) auto
  hence "[2 * a * dedekind_sum' a c1 * q - a * (a + d) * q =
          (a^2 + 1) * q - a * (a + d) * q] (mod ϑ1 * c)"
    by (intro cong_diff cong_refl) (auto simp: c mult_ac)
  also have "(a^2 + 1) * q - a * (a + d) * q = -q * b * c"
    using det unfolding power2_eq_square by (simp add: algebra_simps)
  finally have 2: "[2 * a * q * dedekind_sum' a c1 - a * (a + d) * q =
-q * b * c] (mod ϑ1 * c)"
    by (simp add: mult_ac)

  have r_δ': "r * δ' = r * (2 * dedekind_sum' a c - (a + d) -
                    (2 * q * dedekind_sum' a c1 - (a + d) * q))"
    by (simp add: δ'_def algebra_simps)

  have r_a_δ': "r * a * δ' = r * (2 * a * dedekind_sum' a c - a * (a +
d) -
                    (2 * a * q * dedekind_sum' a c1 - a * (a + d) * q))"
    by (simp add: δ'_def algebra_simps)
  also have "[... = r * (-b * c - (-q * b * c))] (mod ϑ1 * c)"
    by (intro cong_mult cong_diff 1 2 cong_refl)
  also have "r * (-b * c - (-q * b * c)) = r * (q - 1) * b * c"
    by (simp add: algebra_simps)
  also have "r * (q - 1) = 24"
    using q by (auto simp: r_def)
  also have "ϑ1 * 1 * c dvd 24 * b * c"
    by (intro mult_dvd_mono) (auto simp: ϑ1_def gcd_dvdI1)
  hence "[24 * b * c = 0] (mod ϑ1 * c)"
```

```
    by (simp add: cong_0_iff)
  finally have "ϑ1 * c dvd r * δ' * a"
    by (simp add: cong_0_iff mult_ac)
  moreover have "coprime a (gcd 3 c1)"
    using ‹coprime a c1› coprime_imp_coprime dvd_trans by blast
  ultimately have "ϑ1 * c dvd r * δ'"
    using ‹coprime a c›
    by (subst (asm) coprime_dvd_mult_left_iff) (auto simp: ϑ1_def coprime_commute)

  have "3 * c dvd r * δ'"
  proof (cases "q = 3")
    case False
    hence "¬3 dvd q"
      using q by auto
    hence "coprime 3 q"
      by (intro prime_imp_coprime) auto
    hence [simp]: "ϑ1 = ϑ"
      by (auto simp: ϑ_def ϑ1_def c gcd_mult_right_left_cancel)
    show ?thesis
    proof (cases "ϑ = 3")
      case True
      with ‹ϑ1 * c dvd r * δ'› show ?thesis by auto
    next
      case False
      hence [simp]: "ϑ = 1"
        unfolding ϑ_def by (subst gcd_prime_int) auto
      hence "¬3 dvd c"
        unfolding ϑ_def by (subst (asm) gcd_prime_int) auto
      hence "¬3 dvd c1"
        unfolding c using ‹coprime 3 q› coprime_dvd_mult_right_iff by
blast
      have "[2 * dedekind_sum' a c = 0] (mod 3)"
        using three_dvd_dedekind_sum'_iff_aux[of c a] ‹c > 0› ‹coprime
a c› ‹¬3 dvd c›
        by (auto simp: cong_0_iff)
      moreover have "[2 * dedekind_sum' a c1 * q = 0] (mod 3)"
        using three_dvd_dedekind_sum'_iff_aux[of c1 a] ‹c1 > 0› ‹coprime
a c1› ‹¬3 dvd c1›
        by (auto simp: cong_0_iff)
      ultimately have "[r * δ' =
                        r * ((0 - (a + d)) - (0 - (a + d) * q))] (mod
3)"
        unfolding r_δ' by (intro cong_mult cong_diff cong_refl) (simp_all
add: mult_ac)
      also have "r * ((0 - (a + d)) - (0 - (a + d) * q)) = r * (q - 1)
* (a + d)"
        by (simp add: algebra_simps)
      also have "r * (q - 1) = 24"
        using q by (auto simp: r_def)
      also have "[24 * (a + d) = 0] (mod 3)"
        by (simp add: cong_0_iff)
      finally have "3 dvd r * δ'"
        by (simp add: cong_0_iff)
      moreover from ‹ϑ1 * c dvd r * δ'› have "c dvd r * δ'"
        by (simp add: mult_ac)
      moreover have "coprime 3 c"
        using ‹¬3 dvd c› by (intro prime_imp_coprime) auto
      ultimately show "3 * c dvd r * δ'"
        using divides_mult by blast
    qed
  next
    case [simp]: True
    have [simp]: "r = 12"
      by (simp add: r_def)
    have [simp]: "ϑ = 3"
      by (auto simp: ϑ_def c)
    show ?thesis
    proof (cases "ϑ1 = 3")
      case True
      with ‹ϑ1 * c dvd r * δ'› show ?thesis by simp
    next
      case False
      hence [simp]: "ϑ1 = 1"
        unfolding ϑ1_def by (subst gcd_prime_int) auto
      hence "¬3 dvd c1"
        unfolding ϑ1_def by (subst (asm) gcd_prime_int) auto
      have "[2 * dedekind_sum' a c1 * a * q = 0 * q] (mod 3 * q)"
        using three_dvd_dedekind_sum'_iff_aux[of c1 a] ‹c1 > 0› ‹coprime
a c1› ‹¬3 dvd c1›
        by (intro cong_cmult_rightI) (auto simp: cong_0_iff)
      hence "[r * a * δ' = r * (2 * a * dedekind_sum' a c - a * (a + d)
- (0 - a * (a + d) * q))] (mod 9)"
        unfolding r_a_δ' by (intro cong_mult cong_diff cong_refl) (auto
simp: mult_ac)
      also have "r * (2 * a * dedekind_sum' a c - a * (a + d) - (0 - a
* (a + d) * q)) =
                 r * (2 * a * dedekind_sum' a c) + 2 * r * (a^2 + a * d)"
        by (simp add: algebra_simps power2_eq_square)
      also have "[2 * a * dedekind_sum' a c = a^2 + 1] (mod ϑ * c)"
        using cong_dedekind_sum'_2[of c a] ‹c > 0› ‹coprime a c› unfolding ϑ_def by auto
      hence "[2 * a * dedekind_sum' a c = a^2 + 1] (mod 9)"
        by (rule cong_dvd_mono_modulus) (auto simp: c)
      hence "[r * (2 * a * dedekind_sum' a c) + 2 * r * (a^2 + a * d) =
              r * (a^2 + 1) + 2 * r * (a^2 + a * d)] (mod 9)"
        by (intro cong_add cong_mult cong_refl)
      also have "r * (a^2 + 1) + 2 * r * (a^2 + a * d) = 3 * r * a^2 + r
+ 2 * r * (a * d)"
```

```
        by (simp add: algebra_simps)
      also have "a * d = b * c + 1"
        using det by (simp add: algebra_simps)
      also have "3 * r * a^2 + r + 2 * r * (b * c + 1) = 9 * (4 + 8 * b
* c1 + a^2 * 4)"
        by (simp add: algebra_simps c)
      also have "[... = 0] (mod 9)"
        by (simp add: cong_0_iff)
      finally have "9 dvd r * a * δ'"
        by (simp add: cong_0_iff)
      moreover have "coprime a (3 ^ 2)"
        using ‹coprime a c› c by (subst coprime_power_right_iff) auto
      hence "coprime a 9"
        by (simp del: coprime_power_right_iff)
      ultimately have "9 dvd r * δ'"
        by (metis coprime_commute coprime_dvd_mult_right_iff mult.assoc
mult.commute)

      have "3 * c1 dvd 4 * δ'"
      proof (rule divides_mult)
        show "coprime 3 c1"
          using ‹ϑ1 = 1› unfolding ϑ1_def by auto
      next
        have "3 * c1 dvd 3 * (4 * δ')"
          using ‹ϑ1 * c dvd r * δ'› by (simp add: c mult_ac)
        thus "c1 dvd 4 * δ'"
          by (subst (asm) dvd_mult_cancel_left) auto
      next
        have "3 * 3 dvd 3 * (4 * δ')"
          using ‹9 dvd r * δ'› by simp
        thus "3 dvd 4 * δ'"
          by (subst (asm) dvd_mult_cancel_left) auto
      qed
      hence "3 * c dvd 3 * (4 * δ')"
        by (intro mult_dvd_mono dvd_refl) (auto simp: c)
      thus ?thesis
        by (simp add: c mult_ac)
    qed
  qed

  show "24 * c dvd r * δ'"
  proof (cases "even c")
    assume "odd c"
    hence "odd c1"
      by (auto simp: c)

    define T :: "int ⇒ int" where
      "T = (λc. (∑r = 1..<(c + 1) div 2. ⌊real_of_int (2 * a * r) / real_of_int
c⌋))"
```

```
    have "[2 * dedekind_sum’ a c = c - 1 + 4 * T c] (mod 8)"
      unfolding T_def by (rule dedekind_sum’_cong_8_odd)
                         (use ‹coprime a c› ‹odd c› ‹c > 0› in auto)
    moreover have "[2 * dedekind_sum’ a c1 * q = (c1 - 1 + 4 * T c1)
* q] (mod 8)"
      unfolding T_def by (intro cong_mult cong_refl dedekind_sum’_cong_8_odd)

                         (use ‹coprime a c1› ‹odd c1› ‹c1 > 0› in auto)
    ultimately have "[r * δ’ = r * ((c - 1 + 4 * T c - (a + d)) -
                                   ((c1 - 1 + 4 * T c1) * q - (a + d)
* q))] (mod 8)"
      unfolding δ’_def by (intro cong_diff cong_mult[of r] cong_refl)
(auto simp: mult_ac)
    also have "r * ((c - 1 + 4*T c - (a+d)) - ((c1 - 1 + 4*T c1)*q - (a+d)*q))
=
                r * (q - 1) * (a + d + 1) + (r * 4) * (T c - q * T c1)"
      by (simp add: c algebra_simps)
    also have "r * (q - 1) = 24"
      using q by (auto simp: r_def)
    also have "[24 * (a + d + 1) + r * 4 * (T c - q * T c1) =
               0 * (a + d + 1) + 0 * (T c - q * T c1)] (mod 8)"
      using ‹even r› by (intro cong_add cong_mult cong_refl) (auto simp:
cong_0_iff)
    finally have "8 dvd r * δ’"
      by (simp add: cong_0_iff)

    have "8 * (3 * c) dvd r * δ’"
    proof (rule divides_mult)
      have "coprime (2 ^ 3) (3 * c)"
        using ‹odd c› unfolding coprime_power_left_iff by auto
      thus "coprime 8 (3 * c)"
        by (simp del: coprime_power_left_iff)
    qed fact+
    thus ?thesis
      by simp
  next
    assume "even c"
    with ‹coprime a c› have "odd a"
      using coprime_common_divisor odd_one by blast
    from ‹even c› and ‹odd q› have "even c1"
      by (auto simp: c)

    define n where "n = multiplicity 2 c"
    define c’ where "c’ = c div 2 ^ n"
    have "c = 2 ^ n * c’"
      unfolding c’_def n_def by (simp add: multiplicity_dvd)
    have n_altdef: "n = multiplicity 2 c1"
      using ‹odd q› by (auto simp: n_def c multiplicity_prime_elem_times_other)
    have "odd c'"
      unfolding c'_def n_def using ‹c > 0› multiplicity_decompose[of c
2] by auto

    define T where "T = (λc. (∑v = 1..<(a + 1) div 2. ⌊real_of_int (2
* c * v) / real_of_int a⌋))"

    have "[2 * a * dedekind_sum' a c = a^2 + c^2 + 1 + 5 * c - 4 * c * T
c] (mod 2 ^ (n + 3))"
      unfolding n_def T_def using ‹a > 0› ‹c > 0› ‹coprime a c› ‹even
c› ‹odd a›
      by (intro dedekind_sum'_cong_power_of_two') auto
    moreover have "[2 * a * dedekind_sum' a c1 * q = (a^2 + c1^2 + 1 +
5 * c1 - 4 * c1 * T c1) * q]
                     (mod 2 ^ (n + 3))"
      unfolding n_altdef T_def using ‹a > 0› ‹c1 > 0› ‹coprime a c1›
‹even c1› ‹odd a›
      by (intro cong_mult[of _ _ _ q] dedekind_sum'_cong_power_of_two')
auto
    ultimately have "[r * a * δ' =
                        r * ((a^2 + c^2 + 1 + 5 * c - 4 * c * T c) - a *
(a + d) -
                        ((a^2 + c1^2 + 1 + 5 * c1 - 4 * c1 * T c1) * q -
a * (a + d) * q))]
                      (mod 2 ^ (n + 3))"
      unfolding r_a_δ' by (intro cong_mult[of r] cong_diff cong_refl)
(auto simp: mult_ac)
    also have "r * ((a^2 + c^2 + 1 + 5 * c - 4 * c * T c) - a * (a + d)
-
                        ((a^2 + c1^2 + 1 + 5 * c1 - 4 * c1 * T c1) * q -
a * (a + d) * q)) =
               r*(q-1) * (a * d - 1 + c * c1) - 4*c*r * (T c - T c1)"
      by (simp add: algebra_simps c power2_eq_square)
    also have "r * (q - 1) = 24"
      using q by (auto simp: r_def)
    also have "a * d - 1 = b * c"
      using det by (simp add: algebra_simps)
    also have "24 * (b * c + c * c1) = 24 * c * (b + c1)"
      by (simp add: algebra_simps)
    also have "[24 * c * (b + c1) - 4 * c * r * (T c - T c1) = 0 - 0]
(mod 2 ^ (n + 3))"
    proof (intro cong_diff)
      have "2 ^ (n + 3) dvd 2 ^ (n + 3) * (3 * c' * (b + c1))"
        using dvd_triv_left by blast
      also have "... = 24 * c * (b + c1)"
        by (simp add: ‹c = 2 ^ n * c'› mult_ac power_add)
      finally show "[24 * c * (b + c1) = 0] (mod 2 ^ (n + 3))"
        by (simp add: cong_0_iff)
    next
```

```
      have "4 * 2 ^ n * 2 * 1 dvd 4 * c * r * (T c - T c1)"
        using ‹even r› by (intro mult_dvd_mono) (auto simp: ‹c = 2 ^ n
* c'›)
      thus "[4 * c * r * (T c - T c1) = 0] (mod 2 ^ (n + 3))"
        by (simp add: power_add cong_0_iff mult_ac)
    qed
    finally have "2 ^ (n + 3) dvd r * δ' * a"
      by (simp add: cong_0_iff mult_ac)
    hence "2 ^ (n + 3) dvd r * δ'"
      using ‹odd a› by (subst (asm) coprime_dvd_mult_left_iff) auto

    have "2 ^ (n + 3) * (3 * c') dvd r * δ'"
    proof (rule divides_mult)
      show "coprime (2 ^ (n + 3)) (3 * c')"
        using ‹odd c'› by simp
      show "2 ^ (n + 3) dvd r * δ'"
        by fact
      have "3 * c' dvd 3 * c"
        by (auto simp: ‹c = 2 ^ n * c'›)
      also have "3 * c dvd r * δ'"
        by fact
      finally show "3 * c' dvd r * δ'" .
    qed
    thus ?thesis
      by (simp add: ‹c = 2 ^ n * c'› power_add mult_ac)
  qed
qed


theorem dedekind_sum_diff_even_int:
  fixes a b c d :: int assumes det: "a * d - b * c = 1"
  fixes q c1 r :: int and δ' :: "int ⇒ int" and δ :: "int ⇒ real"
  assumes q: "q ∈ {3, 5, 7, 13}" and "c1 > 0"
  assumes c: "c = q * c1"
  defines "r ≡ 24 div (q - 1)"
  defines "δ' ≡ (λa. 2 * dedekind_sum' a c - (a + d) - (2 * q * dedekind_sum'
a c1 - (a + d) * q))"
  defines "δ ≡ (λa. dedekind_sum a c - (a+d)/(12*c) - (dedekind_sum a
c1 - (a+d)/(12*c1)))"
  shows   "of_int (δ' a) = 12 * c * δ a"
    and   "24 * c dvd r * δ' a"
    and   "real_of_int r * δ a / 2 ∈ ℤ"
proof -
  define a' t where "a' = a mod c" and "t = a div c"
  have a'_eq: "a' = a - t * c"
    by (simp add: a'_def t_def algebra_simps)
  have cong1: "[a' = a] (mod c)"
    by (simp add: a'_def)
  hence cong2: "[a' = a] (mod c1)"
```

```
    using c by (metis cong_modulus_mult mult.commute)

  have "q > 0"
    using q by auto
  have "c > 0"
    using q and ‹c1 > 0› and c by auto
  have "[a * d - b * 0 = a * d - b * c] (mod c)"
    by (intro cong_diff cong_mult cong_refl) (auto simp: Cong.cong_def)
  also have "a * d - b * c = 1"
    by fact
  finally have "[a * d = 1] (mod c)"
    by simp
  hence "coprime a c"
    using coprime_iff_invertible_int by blast
  have "coprime a c1"
    using ‹coprime a c› c by auto

  have "a' ≥ 0"
    using ‹c > 0› by (auto simp: a'_def)
  moreover have "a' ≠ 0"
    using ‹coprime a c› q by (auto simp: a'_def c)
  ultimately have "a' > 0"
    by linarith

  have 1: "dedekind_sum a' c = dedekind_sum a c" using cong1
    by (rule dedekind_sum_cong) (use ‹coprime a c› in ‹auto simp: a'_def
coprime_commute›)
  have 2: "dedekind_sum' a' c = dedekind_sum' a c" using cong1
    by (rule dedekind_sum'_cong) (use ‹coprime a c› in ‹auto simp: a'_def
coprime_commute›)
  have 3: "dedekind_sum a' c1 = dedekind_sum a c1" using cong2
    by (rule dedekind_sum_cong) (use ‹coprime a c1› in ‹auto simp: a'_def
coprime_commute›)
  have 4: "dedekind_sum' a' c1 = dedekind_sum' a c1" using cong2
    by (rule dedekind_sum'_cong) (use ‹coprime a c1› in ‹auto simp: a'_def
coprime_commute›)

  have δ_eq: "δ a' = δ a - t * (q - 1) / 12"
    unfolding δ_def 1 3 using ‹c > 0› ‹c1 > 0› ‹q > 0›
    by (simp add: field_simps c a'_eq)
  have δ'_eq: "δ' a' = δ' a - c * t * (q - 1)"
    unfolding δ'_def 2 4 using ‹c > 0› ‹c1 > 0› ‹q > 0›
    by (simp add: field_simps c a'_eq)

  have det': "a' * d - (b - t * d) * c = 1"
    using det by (simp add: a'_eq algebra_simps)

  have "of_int (δ' a') = 12 * c * δ a'"
    unfolding δ'_def δ_def using det' ‹c1 > 0› q c ‹a' > 0›
    by (intro dedekind_sum_diff_even_int_aux[of a' d "b - t * d" c]) auto
  thus of_int_δ': "of_int (δ' a) = 12 * c * δ a"
    by (simp add: δ_eq δ'_eq field_simps)

  have "24 * c dvd r * δ' a'"
    unfolding r_def δ'_def using det' ‹c1 > 0› q c ‹a' > 0›
    by (intro dedekind_sum_diff_even_int_aux[of a' d "b - t * d" c]) auto
  also have "r * δ' a' = r * δ' a - (q - 1) * r * c * t"
    by (simp add: δ'_eq algebra_simps)
  also have "(q - 1) * r = 24"
    unfolding r_def using q by auto
  also have "24 * c dvd r * δ' a - 24 * c * t ⟷ 24 * c dvd r * δ' a"
    by (rule dvd_diff_left_iff) auto
  finally show "24 * c dvd r * δ' a" .

  then obtain m where m: "r * δ' a = 24 * c * m"
    by auto
  hence "real_of_int (r * δ' a) = real_of_int (24 * c * m)"
    by (simp only: )
  hence "real_of_int r * δ a / 2 = of_int m"
    using ‹c > 0› by (simp add: of_int_δ' field_simps)
  also have "... ∈ ℤ"
    by simp
  finally show "real_of_int r * δ a / 2 ∈ ℤ" .
qed

no_notation dedekind_frac ("⟨_⟩")

end
```

References

[1] T. M. Apostol. *Modular Functions and Dirichlet Series in Number Theory.* Graduate Texts in Mathematics. Springer, 1990.