# The Polylogarithm Function

Manuel Eberl

April 18, 2024

**Abstract**

This entry provides a definition of the *Polylogarithm function*, commonly denoted as $\mathrm{Li}_s(z)$. Here, $z$ is a complex number and $s$ an integer parameter. This function can be defined by the power series expression $\mathrm{Li}_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s}$ for $|z| < 1$ and analytically extended to the entire complex plane, except for a branch cut on $\mathbb{R}_{\geq 1}$.

Several basic properties are also proven, such as the relationship to the Eulerian polynomials via $\mathrm{Li}_{-k}(z) = z(1-z)^{k-1} A_k(z)$ for $k \geq 0$, the derivative formula $\frac{d}{dz}\mathrm{Li}_s(z) = \frac{1}{z}\mathrm{Li}_{s-1}(z)$, the relation to the "normal" logarithm via $\mathrm{Li}_1(z) = -\ln(1-z)$, and the duplication formula $\mathrm{Li}_s(z) + \mathrm{Li}_s(-z) = 2^{1-s}\mathrm{Li}_s(z^2)$.

# Contents

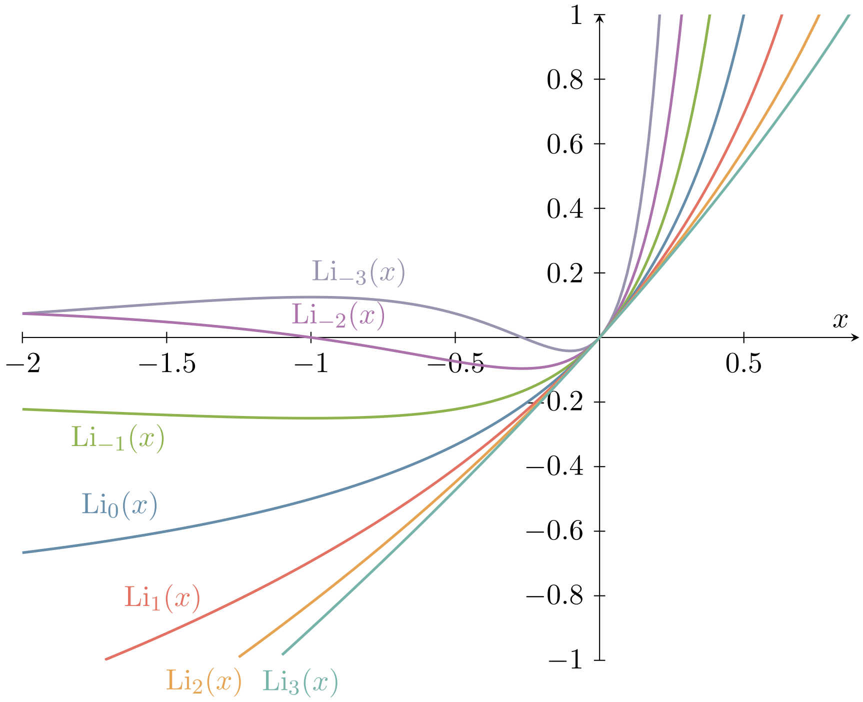


Figure 1: Plots of $\mathrm{Li}_s(x)$ for $s = -3, -2, \ldots, 3$ and real inputs $x \in [-2, 1]$

# 1 Auxiliary material

```
theory Polylog_Library
imports
  "HOL-Complex_Analysis.Complex_Analysis"
  "Linear_Recurrences.Eulerian_Polynomials"
begin
```

## 1.1 Miscellaneous

```
lemma fps_conv_radius_fps_of_poly [simp]:
  fixes p :: "'a :: {banach, real_normed_div_algebra} poly"
  shows "fps_conv_radius (fps_of_poly p) = ∞"
proof -
  have "conv_radius (poly.coeff p) = conv_radius (λ_. 0 :: 'a)"
    using MOST_coeff_eq_0 unfolding cofinite_eq_sequentially by (rule
conv_radius_cong')
  also have "... = ∞"
    by simp
  finally show ?thesis
    by (simp add: fps_conv_radius_def)
qed

lemma eval_fps_power:
  fixes F :: "'a :: {banach, real_normed_div_algebra, comm_ring_1} fps"
  assumes z: "norm z < fps_conv_radius F"
  shows     "eval_fps (F ^ n) z = eval_fps F z ^ n"
proof (induction n)
  case 0
  thus ?case
    by (auto simp: eval_fps_mult)
next
  case (Suc n)
  have "eval_fps (F ^ Suc n) z = eval_fps (F * F ^ n) z"
    by simp
  also from z have "... = eval_fps F z * eval_fps (F ^ n) z"
    by (subst eval_fps_mult) (auto intro!: less_le_trans[OF _ fps_conv_radius_power])
  finally show ?case
    using Suc.IH by simp
qed

lemma eval_fps_of_poly [simp]: "eval_fps (fps_of_poly p) z = poly p z"
proof -
  have "(λn. poly.coeff p n * z ^ n) sums poly p z"
    unfolding poly_altdef by (rule sums_finite) (auto simp: coeff_eq_0)
  moreover have "(λn. poly.coeff p n * z ^ n) sums eval_fps (fps_of_poly
p) z"
    using sums_eval_fps[of z "fps_of_poly p"] by simp
  ultimately show ?thesis
    using sums_unique2 by blast
```

```
qed

lemma poly_holomorphic_on [holomorphic_intros]:
  assumes [holomorphic_intros]: "f holomorphic_on A"
  shows   "(λz. poly p (f z)) holomorphic_on A"
  unfolding poly_altdef by (intro holomorphic_intros)

lemma simply_connected_eq_global_primitive:
  assumes "simply_connected S" "open S" "f holomorphic_on S"
  obtains h where "⋀z. z ∈ S ⟹ (h has_field_derivative f z) (at z)"
  using simply_connected_eq_global_primitive[of S] assms that by blast

lemma
  assumes "x ∈ closed_segment y z"
  shows in_closed_segment_imp_Re_in_closed_segment: "Re x ∈ closed_segment
(Re y) (Re z)" (is ?th1)
    and in_closed_segment_imp_Im_in_closed_segment: "Im x ∈ closed_segment
(Im y) (Im z)" (is ?th2)
proof -
  from assms obtain t where t: "t ∈ {0..1}" "x = linepath y z t"
    by (metis imageE linepath_image_01)
  have "Re x = linepath (Re y) (Re z) t" "Im x = linepath (Im y) (Im z)
t"
    by (simp_all add: t Re_linepath' Im_linepath')
  with t(1) show ?th1 ?th2
    using linepath_in_path[of t "Re y" "Re z"] linepath_in_path[of t "Im
y" "Im z"] by simp_all
qed

lemma linepath_in_open_segment: "t ∈ {0<..<1} ⟹ x ≠ y ⟹ linepath
x y t ∈ open_segment x y"
  unfolding greaterThanLessThan_iff by (metis in_segment(2) linepath_def)

lemma in_open_segment_imp_Re_in_open_segment:
  assumes "x ∈ open_segment y z" "Re y ≠ Re z"
  shows   "Re x ∈ open_segment (Re y) (Re z)"
proof -
  from assms obtain t where t: "t ∈ {0<..<1}" "x = linepath y z t"
    by (metis greaterThanLessThan_iff in_segment(2) linepath_def)
  have "Re x = linepath (Re y) (Re z) t"
    by (simp_all add: t Re_linepath')
  with t(1) show ?thesis
    using linepath_in_open_segment[of t "Re y" "Re z"] assms by auto
qed

lemma in_open_segment_imp_Im_in_open_segment:
  assumes "x ∈ open_segment y z" "Im y ≠ Im z"
  shows   "Im x ∈ open_segment (Im y) (Im z)"
proof -
```

```
  from assms obtain t where t: "t ∈ {0<..<1}" "x = linepath y z t"
    by (metis greaterThanLessThan_iff in_segment(2) linepath_def)
  have "Im x = linepath (Im y) (Im z) t"
    by (simp_all add: t Im_linepath')
  with t(1) show ?thesis
    using linepath_in_open_segment[of t "Im y" "Im z"] assms by auto
qed


lemma poly_eulerian_poly_0 [simp]: "poly (eulerian_poly n) 0 = 1"
  by (induction n) (auto simp: eulerian_poly.simps(2) Let_def)


lemma eulerian_poly_at_1 [simp]: "poly (eulerian_poly n) 1 = fact n"
  by (induction n) (auto simp: eulerian_poly.simps(2) Let_def algebra_simps)
```

## 1.2 The slotted complex plane

```
lemma closed_slot_left: "closed (complex_of_real ` {..c})"
  by (intro closed_injective_linear_image) (auto simp: inj_def)

lemma closed_slot_right: "closed (complex_of_real ` {c..})"
  by (intro closed_injective_linear_image) (auto simp: inj_def)

lemma complex_slot_left_eq: "complex_of_real ` {..c} = {z. Re z ≤ c
∧ Im z = 0}"
  by (auto simp: image_iff complex_eq_iff)

lemma complex_slot_right_eq: "complex_of_real ` {c..} = {z. Re z ≥ c
∧ Im z = 0}"
  by (auto simp: image_iff complex_eq_iff)

lemma complex_double_slot_eq:
  "complex_of_real ` ({..c1} ∪ {c2..}) = {z. Im z = 0 ∧ (Re z ≤ c1 ∨
Re z ≥ c2)}"
  by (auto simp: image_iff complex_eq_iff)

lemma starlike_slotted_complex_plane_left_aux:
  assumes z: "z ∈ -(complex_of_real ` {..c})" and c: "c < c'"
  shows   "closed_segment (complex_of_real c') z ⊆ -(complex_of_real
` {..c})"
proof -
  show "closed_segment c' z ⊆ -of_real ` {..c}"
  proof (cases "Im z = 0")
    case True
    thus ?thesis using z c
      by (auto simp: closed_segment_same_Im closed_segment_eq_real_ivl
complex_slot_left_eq)
  next
```

```
    case False
    show ?thesis
    proof
      fix x assume x: "x ∈ closed_segment (of_real c') z"
      consider "x = of_real c'" | "x = z" | "x ∈ open_segment (of_real
c') z"
        unfolding open_segment_def using x by blast
      thus "x ∈ -complex_of_real ` {..c}"
      proof cases
        assume "x ∈ open_segment (of_real c') z"
        hence "Im x ∈ open_segment (Im (complex_of_real c')) (Im z)"
          by (intro in_open_segment_imp_Im_in_open_segment) (use False
in auto)
        hence "Im x ≠ 0"
          by (auto simp: open_segment_eq_real_ivl split: if_splits)
        thus ?thesis
          by (auto simp: complex_slot_right_eq)
      qed (use z c in ‹auto simp: complex_slot_left_eq›)
    qed
  qed
qed

lemma starlike_slotted_complex_plane_left: "starlike (-(complex_of_real
` {..c}))"
  unfolding starlike_def
proof (rule bexI[of _ "of_real c + 1"]; (intro ballI)?)
  show "complex_of_real c + 1 ∈ -complex_of_real ` {..c}"
    by (auto simp: complex_eq_iff)
  show "closed_segment (complex_of_real c + 1) z ⊆ - complex_of_real
` {..c}"
    if "z ∈ - complex_of_real ` {..c}" for z
    using starlike_slotted_complex_plane_left_aux[OF that, of "c + 1"]
by simp
qed


lemma starlike_slotted_complex_plane_right_aux:
  assumes z: "z ∈ -(complex_of_real ` {c..})" and c: "c > c'"
  shows   "closed_segment (complex_of_real c') z ⊆ -(complex_of_real
` {c..})"
proof -
  show "closed_segment c' z ⊆ -of_real ` {c..}"
  proof (cases "Im z = 0")
    case True
    thus ?thesis using z c
      by (auto simp: closed_segment_same_Im closed_segment_eq_real_ivl
complex_slot_right_eq)
  next
    case False
```

```
    show ?thesis
    proof
      fix x assume x: "x ∈ closed_segment (of_real c') z"
      consider "x = of_real c'" | "x = z" | "x ∈ open_segment (of_real
c') z"
        unfolding open_segment_def using x by blast
      thus "x ∈ -complex_of_real ` {c..}"
      proof cases
        assume "x ∈ open_segment (of_real c') z"
        hence "Im x ∈ open_segment (Im (complex_of_real c')) (Im z)"
          by (intro in_open_segment_imp_Im_in_open_segment) (use False
in auto)
        hence "Im x ≠ 0"
          by (auto simp: open_segment_eq_real_ivl split: if_splits)
        thus ?thesis
          by (auto simp: complex_slot_right_eq)
      qed (use z c in ‹auto simp: complex_slot_right_eq›)
    qed
  qed
qed

lemma starlike_slotted_complex_plane_right: "starlike (-(complex_of_real
` {c..}))"
  unfolding starlike_def
proof (rule bexI[of _ "of_real c - 1"]; (intro ballI)?)
  show "complex_of_real c - 1 ∈ -complex_of_real ` {c..}"
    by (auto simp: complex_eq_iff)
  show "closed_segment (complex_of_real c - 1) z ⊆ - complex_of_real
` {c..}"
    if "z ∈ - complex_of_real ` {c..}" for z
    using starlike_slotted_complex_plane_right_aux[OF that, of "c - 1"]
by simp
qed


lemma starlike_doubly_slotted_complex_plane_aux:
  assumes z: "z ∈ -(complex_of_real ` ({..c1} ∪ {c2..}))" and c: "c1
< c" "c < c2"
  shows   "closed_segment (complex_of_real c) z ⊆ -(complex_of_real `
({..c1} ∪ {c2..}))"
proof -
  show "closed_segment c z ⊆ -of_real ` ({..c1} ∪ {c2..})"
  proof (cases "Im z = 0")
    case True
    thus ?thesis using z c
      by (auto simp: closed_segment_same_Im closed_segment_eq_real_ivl
complex_double_slot_eq)
  next
    case False
```

```
    show ?thesis
    proof
      fix x assume x: "x ∈ closed_segment (of_real c) z"
      consider "x = of_real c" | "x = z" | "x ∈ open_segment (of_real
c) z"
        unfolding open_segment_def using x by blast
      thus "x ∈ -complex_of_real ` ({..c1} ∪ {c2..})"
      proof cases
        assume "x ∈ open_segment (of_real c) z"
        hence "Im x ∈ open_segment (Im (complex_of_real c)) (Im z)"
          by (intro in_open_segment_imp_Im_in_open_segment) (use False
in auto)
        hence "Im x ≠ 0"
          by (auto simp: open_segment_eq_real_ivl split: if_splits)
        thus ?thesis
          by (auto simp: complex_slot_right_eq)
      qed (use z c in ‹auto simp: complex_slot_right_eq›)
    qed
  qed
qed

lemma starlike_doubly_slotted_complex_plane:
  assumes "c1 < c2"
  shows   "starlike (-(complex_of_real ` ({..c1} ∪ {c2..})))"
proof -
  from assms obtain c where c: "c1 < c" "c < c2"
    using dense by blast
  show ?thesis
    unfolding starlike_def
  proof (rule bexI[of _ "of_real c"]; (intro ballI)?)
    show "complex_of_real c ∈ -complex_of_real ` ({..c1} ∪ {c2..})"
      using c by (auto simp: complex_eq_iff)
    show "closed_segment (complex_of_real c) z ⊆ - complex_of_real `
({..c1} ∪ {c2..})"
      if "z ∈ - complex_of_real ` ({..c1} ∪ {c2..})" for z
      using starlike_doubly_slotted_complex_plane_aux[OF that, of c] c
by simp
  qed
qed

lemma simply_connected_slotted_complex_plane_left:
  "simply_connected (-(complex_of_real ` {..c}))"
  by (intro starlike_imp_simply_connected starlike_slotted_complex_plane_left)

lemma simply_connected_slotted_complex_plane_right:
  "simply_connected (-(complex_of_real ` {c..}))"
  by (intro starlike_imp_simply_connected starlike_slotted_complex_plane_right)

lemma simply_connected_doubly_slotted_complex_plane:
```

```
  "c1 < c2 ⟹ simply_connected (-(complex_of_real ` ({..c1} ∪ {c2..})))"
  by (intro starlike_imp_simply_connected starlike_doubly_slotted_complex_plane)

end
```

# 2 The Polylogarithm Function

```
theory Polylog
imports
  "HOL-Complex_Analysis.Complex_Analysis"
  "Linear_Recurrences.Eulerian_Polynomials"
  "HOL-Real_Asymp.Real_Asymp"
  Polylog_Library
begin
```

## 2.1 Definition and basic properties

The principal branch of the Polylogarithm function $\mathrm{Li}_s(z)$ is defined as

$$\mathrm{Li}_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s}$$

for $|z| < 1$ and elsewhere by analytic continuation. For integer $s \leq 0$ it is holomorphic except for a pole at $z = 1$. For other values of $s$ it is holomorphic except for a branch cut along the line $[1, \infty)$.

Special values include $\mathrm{Li}_0(z) = \frac{z}{1-z}$ and $\mathrm{Li}_1(z) = -\log(1 - z)$.

One could potentially generalise this to arbitrary $s \in \mathbb{C}$, but this makes the analytic continuation somewhat more complicated, so we chosed not to do this at this point.

In the following, we define the principal branch of $\mathrm{Li}_s(z)$ for integer $s$.

```
definition polylog :: "int ⇒ complex ⇒ complex" where
  "polylog k z =
     (if k ≤ 0 then z * poly (eulerian_poly (nat (-k))) z * (1 - z) powi
(k - 1)
      else if z ∈ of_real ` {1..} then 0
           else (SOME f. f holomorphic_on -of_real`{1..} ∧
                     (∀z∈ball 0 1. f z = (∑n. of_nat (Suc n) powi (-k)
* z ^ Suc n))) z)"

lemma conv_radius_polylog: "conv_radius (λr. of_nat r powi k :: complex)
= 1"
proof (rule conv_radius_ratio_limit_ereal_nonzero)
  have "(λn. ereal (real n powi k / real (Suc n) powi k)) ⟶ ereal
1"
  proof (cases "k ≥ 0")
    case True
```

```
    have "(λn. ereal (real n ^ nat k / real (Suc n) ^ nat k)) ⟶ ereal
1"
      by (intro tendsto_ereal) real_asymp
    thus ?thesis
      using True by (simp add: power_int_def)
  next
    case False
    have "(λn. ereal (inverse (real n) ^ nat (-k) / inverse (real (Suc
n)) ^ nat (-k))) ⟶ ereal 1"
      by (intro tendsto_ereal) real_asymp
    thus ?thesis
      using False by (simp add: power_int_def)
  qed
  thus "(λn. ereal (norm (of_nat n powi k :: complex) / norm (of_nat (Suc
n) powi k :: complex))) ⟶ 1"
    unfolding one_ereal_def [symmetric] by (simp add: norm_power_int del:
of_nat_Suc)
qed auto

lemma abs_summable_polylog:
  "norm z < 1 ⟹ summable (λr. norm (of_nat r powi k * z ^ r :: complex))"
  by (rule abs_summable_in_conv_radius) (use conv_radius_polylog[of k]
in auto)
```

Two very central results that characterise the polylogarithm:

$$\mathrm{Li}'_s(z) = \frac{1}{z}\mathrm{Li}_{s-1}(z) \qquad \text{and} \qquad \mathrm{Li}_s(z) = \sum_{n=1}^{\infty} \frac{z^n}{n^s} \quad \text{for } |z| < 1$$

```
theorem has_field_derivative_polylog [derivative_intros]:
        "⋀z. z ∈ (if k ≤ 0 then -{1} else -(of_real ` {1..})) ⟹
              (polylog k has_field_derivative (if z = 0 then 1 else polylog
(k - 1) z / z)) (at z within A)"
  and sums_polylog: "norm z < 1 ⟹ (λn. of_nat (Suc n) powi (-k) * z
^ Suc n) sums polylog k z"
proof -
  let ?S = "-(complex_of_real ` {1..})"
  have "open ?S"
    by (intro open_Compl closed_slot_right)
  define S where "S = (λk::int. if k ≤ 0 then -{1} else ?S)"
  have [simp]: "open (S k)" for k
    using ‹open ?S› by (auto simp: S_def)

  have *: "(∀z∈S k. (polylog k has_field_derivative (if z = 0 then 1
else polylog (k - 1) z / z)) (at z)) ∧
          (∀z∈ball 0 1. (λn. of_nat (Suc n) powi (-k) * z ^ Suc n) sums
polylog k z)"
  proof (induction "nat k" arbitrary: k)
    case 0
```

```
    define k' where "k' = nat (-k)"
    have k_eq: "k = -int k'"
      using 0 by (simp add: k'_def)

    have "(polylog k has_field_derivative (if z = 0 then 1 else polylog
(k - 1) z / z)) (at z)"
      if z: "z ∈ S k" for z
    proof -
      have [simp]: "z ≠ 1"
        using z 0 by (auto simp: S_def)
      write eulerian_poly ("E")
      have "polylog (k - 1) z = z * (poly (E (Suc k')) z * (1 - z) powi
(k - 2))"
        using 0 by (simp add: polylog_def k_eq nat_add_distrib algebra_simps)
      also have "... = z * poly (E (Suc k')) z / (1 - z) ^ (k' + 2)"
        by (simp add: k_eq power_int_def nat_add_distrib field_simps)
      finally have eq1: "polylog (k - 1) z = ..." .

      have "polylog k = (λz. z * poly (E k') z * (1 - z) powi (k - 1))"
        using 0 by (simp add: polylog_def [abs_def] k_eq)
      also have "... = (λz. z * poly (E k') z / (1 - z) ^ Suc k')"
        by (simp add: k_eq power_int_def field_simps nat_add_distrib)
      finally have eq2: "polylog k = (λz. z * poly (E k') z / (1 - z) ^
Suc k')" .

      have "((λz. z * poly (E k') z / (1 - z) ^ Suc k') has_field_derivative
               (poly (E (Suc k')) z / (1 - z) ^ (k' + 2))) (at z)"
        apply (rule derivative_eq_intros refl poly_DERIV)+
         apply (simp)
        apply (simp add: eulerian_poly.simps(2) Let_def divide_simps)
        apply (simp add: algebra_simps)
        done
      also note eq2 [symmetric]
      also have "poly (E (Suc k')) z / (1 - z) ^ (k' + 2) =
               (if z = 0 then 1 else polylog (k - 1) z / z)"
        by (subst eq1) (auto)
      finally show ?thesis .
    qed

    moreover have "(λn. of_nat (Suc n) powi (-k) * z ^ Suc n) sums polylog
k z"
      if z: "norm z < 1" for z
    proof (cases "k = 0")
      case True
      thus ?thesis using z geometric_sums[of z]
        by (auto simp: polylog_def divide_inverse intro!: sums_mult)
    next
      case False
      with 0 have k: "k < 0"
```

```
        by simp
      define F where "F = Abs_fps (λn. of_nat n ^ nat (-k) :: complex)"
      have "fps_conv_radius (1 - fps_X :: complex fps) ≥ ∞"
        by (intro order.trans[OF _ fps_conv_radius_diff]) auto
      hence [simp]: "fps_conv_radius (1 - fps_X :: complex fps) = ∞"
        by simp
      have *: "fps_conv_radius ((1 - fps_X) ^ (nat (-k) + 1) :: complex
fps) ≥ ∞"
        by (intro order.trans[OF _ fps_conv_radius_power]) auto

      have "ereal (norm z) < 1"
        using that by simp
      also have "1 ≤ fps_conv_radius F"
        unfolding F_def fps_conv_radius_def using conv_radius_polylog[of
"-k"] 0
        by (simp add: power_int_def)
      finally have "(λn. fps_nth F n * z ^ n) sums eval_fps F z"
        by (rule sums_eval_fps)
      also have "(λn. fps_nth F n * z ^ n) = (λn. of_nat n powi (-k) *
z ^ n)"
        using 0 by (simp add: F_def power_int_def)
      also have "eval_fps F z = poly (fps_monom_poly 1 (nat (- k))) z
/
                              eval_fps ((1 - fps_X) ^ (nat (- k) + 1))
z"
        unfolding F_def fps_monom_aux
      proof (subst eval_fps_divide')
        show "fps_conv_radius (fps_of_poly (fps_monom_poly 1 (nat (-
k)))) > 0"
          by simp
        show "fps_conv_radius ((1 - fps_X :: complex fps) ^ (nat (- k)
+ 1)) > 0"
          by (intro less_le_trans[OF _ fps_conv_radius_power]) auto
        show "1 > (0 :: ereal)"
          by simp
        show "eval_fps ((1 - fps_X) ^ (nat (-k) + 1)) z ≠ 0"
          if "z ∈ eball 0 1" for z :: complex
          using that by (subst eval_fps_power) (auto simp: eval_fps_diff)
        show "ereal (norm z) < Min {1, fps_conv_radius (fps_of_poly (fps_monom_poly
1 (nat (- k)))),
                fps_conv_radius ((1 - fps_X :: complex fps) ^ (nat (-
k) + 1))}" using * z
          by auto
      qed auto
      also have "eval_fps ((1 - fps_X) ^ (nat (- k) + 1)) z = (1 - z)
^ (nat (-k) + 1)"
        by (subst eval_fps_power) (auto simp: eval_fps_diff)
      also have "... = (1 - z) powi int (nat (-k) + 1)"
        by (rule power_int_of_nat [symmetric])
```

```
      also have "int (nat (-k) + 1) = -(k-1)"
        using 0 by simp
      also have "(poly (fps_monom_poly 1 (nat (- k))) z / (1 - z) powi
- (k - 1)) = polylog k z"
        using k
        by (auto simp add: fps_monom_poly_def polylog_def power_int_diff)
      finally show "(λn. of_nat (Suc n) powi - k * z ^ (Suc n)) sums polylog
k z"
        by (subst sums_Suc_iff) (use k in auto)
    qed
    ultimately show ?case
      using 0 by (auto simp: polylog_def [abs_def])
  next
    case (Suc k' k)
    have [simp]: "nat k = Suc k'" "nat (k - 1) = k'"
      using Suc(2) by auto
    from Suc(2) have k: "k > 0"
      by linarith
    have deriv: "(polylog (k - 1) has_field_derivative
          (if z = 0 then 1 else polylog (k - 2) z / z)) (at z)" if "z
∈ S (k - 1)" for z
      using Suc(1)[of "k-1"] that by auto
    hence holo: "polylog (k - 1) holomorphic_on S (k - 1)"
      by (subst holomorphic_on_open) auto

    have sums: "(λn. of_nat (Suc n) powi -(k-1) * z ^ Suc n) sums polylog
(k-1) z"
      if "norm z < 1" for z
      using that Suc(1)[of "k - 1"] by auto

    define g where "g = (λz. if z = 0 then 1 else polylog (k - 1) z /
z)"
    have "g holomorphic_on S (k - 1)"
      unfolding g_def
    proof (rule removable_singularity)
      show "(λz. polylog (k - 1) z / z) holomorphic_on S (k - 1) - {0}"
        using Suc by (intro holomorphic_intros holomorphic_on_subset[OF
holo]) auto

      define F where "F = Abs_fps (λn. of_nat (Suc n) powi (1-k) :: complex)"
      have radius: "fls_conv_radius (fps_to_fls F) = 1"
      proof -
        have "F = fps_shift 1 (Abs_fps (λn. of_int n powi (1 - k)))"
          using k by (simp add: F_def fps_eq_iff power_int_def)
        also have "fps_conv_radius ... = 1"
          using conv_radius_polylog[of "1 - k"] unfolding fps_conv_radius_shift
          by (simp add: fps_conv_radius_def)
        finally show ?thesis by simp
      qed
```

```
      have "eventually (λz::complex. z ∈ ball 0 1) (nhds 0)"
        by (intro eventually_nhds_in_open) auto
      hence "eventually (λz::complex. z ∈ ball 0 1 - {0}) (at 0)"
        unfolding eventually_at_filter by eventually_elim auto
      hence "eventually (λz. eval_fls (fps_to_fls F) z = polylog (k -
1) z / z) (at 0)"
      proof eventually_elim
        case (elim z)
        have "(λn. of_nat (Suc n) powi - (k - 1) * z ^ Suc n / z) sums
(polylog (k - 1) z / z)"
          by (intro sums_divide sums) (use elim in auto)
        also have "(λn. of_nat (Suc n) powi - (k - 1) * z ^ Suc n / z)
=
                   (λn. of_nat (Suc n) powi - (k - 1) * z ^ n)"
          using elim by auto
        finally have "polylog (k - 1) z / z = (∑n. of_nat (Suc n) powi
- (k - 1) * z ^ n)"
          by (simp add: sums_iff)
        also have "... = eval_fps F z"
          unfolding eval_fps_def F_def by simp
        finally show ?case
          using radius elim by (simp add: eval_fps_to_fls)
      qed
      hence "(λz. polylog (k - 1) z / z) has_laurent_expansion fps_to_fls
F"
        unfolding has_laurent_expansion_def using radius by auto
      hence "(λz. polylog (k - 1) z / z) —0→ fls_nth (fps_to_fls F)
0"
        by (intro has_laurent_expansion_imp_tendsto_0 fls_subdegree_fls_to_fps_gt0)
auto
      thus "(λy. polylog (k - 1) y / y) —0→ 1"
        by (simp add: F_def)
    qed auto
    hence holo: "g holomorphic_on ?S"
      by (rule holomorphic_on_subset) (auto simp: S_def)
    have "simply_connected ?S"
      by (rule simply_connected_slotted_complex_plane_right)
    then obtain f where f: "⋀z. z ∈ ?S ⟹ (f has_field_derivative g
z) (at z)"
      using simply_connected_eq_global_primitive holo ‹open ?S› by blast

    define h where "h = (λz. f z - f 0)"
    have deriv_h [derivative_intros]: "(h has_field_derivative g z) (at
z)" if "z ∈ ?S" for z
      unfolding h_def using that by (auto intro!: derivative_eq_intros
f)
    hence holo_h: "h holomorphic_on S k" (is "?P1 h")
      by (subst holomorphic_on_open) (use k ‹open ?S› in ‹auto simp: S_def›)
```

```
    have summable: "summable (λn. of_nat n powi (-k) * z ^ n)"
      if "norm z < 1" for z :: complex
      by (rule summable_in_conv_radius)
         (use that conv_radius_polylog[of "-k"] in auto)

    define F where "F = Abs_fps (λn. of_nat n powi (-k) :: complex)"
    have radius: "fps_conv_radius F = 1"
      using conv_radius_polylog[of "-k"] by (simp add: fps_conv_radius_def
F_def)

    have F_deriv [derivative_intros]:
      "(eval_fps F has_field_derivative g z) (at z)" if "z ∈ ball 0 1"
for z
    proof -
      have "(eval_fps F has_field_derivative eval_fps (fps_deriv F) z)
(at z)"
        using that radius by (auto intro!: derivative_eq_intros)
      also have "eval_fps (fps_deriv F) z = g z"
      proof (cases "z = 0")
        case False
        have "(λn. of_nat (Suc n) powi - (k - 1) * z ^ Suc n / z) sums
(polylog (k - 1) z / z)"
          by (intro sums_divide sums) (use that in auto)
        also have "... = g z"
          using False by (simp add: g_def)
        also have "(λn. of_nat (Suc n) powi - (k - 1) * z ^ Suc n / z)
=
                   (λn. of_nat (Suc n) powi - (k - 1) * z ^ n)"
          using False by simp
        finally show ?thesis
          by (auto simp add: eval_fps_def F_def sums_iff power_int_diff
power_int_minus field_simps
                   simp del: of_nat_Suc)
      qed (auto simp: F_def g_def eval_fps_at_0)
      finally show ?thesis .
    qed

    hence h_eq_sum: "h z = eval_fps F z" if "z ∈ ball 0 1" for z
    proof -
      have "∃c. ∀z∈ball 0 1. h z - eval_fps F z = c"
      proof (rule has_field_derivative_zero_constant)
        fix z :: complex assume z: "z ∈ ball 0 1"
        have "((λx. h x - eval_fps F x) has_field_derivative 0) (at z)"
          using z by (auto intro!: derivative_eq_intros)
        thus "((λx. h x - eval_fps F x) has_field_derivative 0) (at z
within ball 0 1)"
          using z by (subst at_within_open) auto
      qed auto
```

```
      then obtain c where c: "⋀z. norm z < 1 ⟹ h z - eval_fps F z
= c"
        by force
      from c[of 0] and k have "c = 0"
        by (simp add: h_def F_def eval_fps_at_0)
      thus ?thesis
        using c[of z] that by auto
    qed

    have h_eq_sum': "(∀z∈ball 0 1. h z = (∑n. of_nat (Suc n) powi -
k * z ^ Suc n))" (is "?P2 h")
    proof safe
      fix z :: complex assume z: "z ∈ ball 0 1"
      have "summable (λn. of_nat (Suc n) powi - k * z ^ Suc n)"
        using z summable[of z] by (subst summable_Suc_iff) auto
      also have "?this ⟷ summable (λn. of_nat n powi - k * z ^ n)"
        by (rule summable_Suc_iff)
      finally have "(λn. of_nat (Suc n) powi -k * z ^ Suc n) sums h z"
        using h_eq_sum[of z] k unfolding summable_Suc_iff
        by (subst sums_Suc_iff) (use z in ‹auto simp: eval_fps_def F_def›)
      thus "h z = (∑n. of_nat (Suc n) powi - k * z ^ Suc n)"
        by (simp add: sums_iff)
    qed

    define h' where "h' = (SOME h. ?P1 h ∧ ?P2 h)"
    have "∃h. ?P1 h ∧ ?P2 h"
      using h_eq_sum' holo_h by blast
    from someI_ex[OF this] have h'_props: "?P1 h'" "?P2 h'"
      unfolding h'_def by blast+
    have h'_eq: "h' z = polylog k z" if "z ∈ S k" for z
      using that k by (auto simp: polylog_def h'_def S_def)

    have polylog_sums: "(λn. of_nat (Suc n) powi (-k) * z ^ Suc n) sums
polylog k z"
      if "norm z < 1" for z
    proof -
      have "summable (λn. of_nat (Suc n) powi (-k) * z ^ Suc n)"
        using summable[of z] that by (subst summable_Suc_iff)
      moreover from that have "z ∈ S k"
        by (auto simp: S_def)
      ultimately show ?thesis
        using h'_props using that by (force simp: sums_iff h'_eq)
    qed

    have eq': "polylog k z = h z" if "z ∈ S k" for z
    proof -
      have "h' z = h z"
      proof (rule analytic_continuation_open[where g = h])
        show "h' holomorphic_on S k" "h holomorphic_on S k"
```

```
        by fact+
      show "ball 0 1 ≠ ({} :: complex set)" "open (ball 0 1 :: complex
set)"
        by auto
      show "open (S k)" "connected (S k)" "ball 0 1 ⊆ S k"
        using k ‹open ?S› simply_connected_slotted_complex_plane_right[of
1]
        by (auto simp: S_def simply_connected_imp_connected)
      show "z ∈ S k"
        by fact
      show "h' z = h z" if "z ∈ ball 0 1" for z
        using h'_props(2) h_eq_sum' that by simp
    qed
    with that show ?thesis
      by (simp add: h'_eq)
  qed

  have deriv_polylog: "(polylog k has_field_derivative g z) (at z)"
if "z ∈ S k" for z
  proof -
    have "(h has_field_derivative g z) (at z)"
      by (intro deriv_h) (use that k in ‹auto simp: S_def›)
    also have "?this ⟷ ?thesis"
    proof (rule DERIV_cong_ev)
      have "eventually (λw. w ∈ S k) (nhds z)"
        by (intro eventually_nhds_in_open) (use that in auto)
      thus "eventually (λw. h w = polylog k w) (nhds z)"
        by eventually_elim (auto simp: eq')
    qed auto
    finally show ?thesis .
  qed

  show ?case
    using deriv_polylog polylog_sums unfolding g_def by simp
 qed

 show "(polylog k has_field_derivative (if z = 0 then 1 else polylog
(k - 1) z / z)) (at z within A)"
  if "z ∈ (if k ≤ 0 then -{1} else -(of_real ` {1..}))" for z
  using * that unfolding S_def by (blast intro: has_field_derivative_at_within)
 show "(λn. of_nat (Suc n) powi (-k) * z ^ Suc n) sums polylog k z"
if "norm z < 1" for z
  using * that by force
qed

lemma has_field_derivative_polylog' [derivative_intros]:
 assumes "(f has_field_derivative f') (at z within A)"
 assumes "if k ≤ 0 then f z ≠ 1 else Im (f z) ≠ 0 ∨ Re (f z) < 1"
 shows   "((λz. polylog k (f z)) has_field_derivative
```

```
            (if f z = 0 then 1 else polylog (k-1) (f z) / f z) * f')
(at z within A)"
proof -
  have "(polylog k ∘ f has_field_derivative
          (if f z = 0 then 1 else polylog (k-1) (f z) / f z) * f') (at
z within A)"
    using assms(2) by (intro DERIV_chain assms has_field_derivative_polylog)
auto
  thus ?thesis
    by (simp add: o_def)
qed

lemma polylog_0 [simp]: "polylog k 0 = 0"
proof -
  have "(λ_. 0) sums polylog k 0"
    using sums_polylog[of 0 k] by simp
  moreover have "(λ_. 0 :: complex) sums 0"
    by simp
  ultimately show ?thesis
    using sums_unique2 by blast
qed
```

A simple consequence of the derivative formula is the following recurrence for $\mathrm{Li}_s$ via a contour integral:

$$\mathrm{Li}_s(z) = \int_0^z \frac{1}{w}\mathrm{Li}_{s-1}(w)\,\mathrm{d}w$$

```
theorem polylog_has_contour_integral:
  assumes "z ∉ complex_of_real ` ({..-1} ∪ {1..})"
  shows   "((λw. polylog s w / w) has_contour_integral polylog (s + 1)
z) (linepath 0 z)"
proof -
  let ?l = "linepath 0 z"
  define A where "A = -complex_of_real ` ({..-1} ∪ {1..})"
  have "((λw. if w = 0 then 1 else polylog s w / w) has_contour_integral
           (polylog (s + 1) (pathfinish ?l) - polylog (s + 1) (pathstart
?l))) (linepath 0 z)"
  proof (rule contour_integral_primitive)
    have [simp]: "complex_of_real x = -1 ⟷ x = -1" for x
      by (simp add: Complex_eq_neg_1 complex_of_real_def)
    show "(polylog (s + 1) has_field_derivative (if z = 0 then 1 else
polylog s z / z))
            (at z within A)" if "z ∈ A" for z
      using that by (intro derivative_eq_intros) (auto simp: A_def split:
if_splits)
  next
    show "valid_path (linepath 0 z)"
      by (rule valid_path_linepath)
  next
```

```
    show "path_image (linepath 0 z) ⊆ A"
      using assms starlike_doubly_slotted_complex_plane_aux[of z "-1"
1 0]
      by (auto simp: A_def)
  qed
  hence "((λw. if w = 0 then 1 else polylog s w / w) has_contour_integral
          (polylog (s + 1) z)) (linepath 0 z)"
    by simp
  thus ?thesis
    unfolding has_contour_integral_def
  proof (rule has_integral_spike[rotated 2])
    show "negligible {0 :: real}"
      by simp
  qed (auto simp: vector_derivative_linepath_within)
qed

lemma sums_polylog':
  "norm z < 1 ⟹ k ≠ 0 ⟹ (λn. of_nat n powi - k * z ^ n) sums polylog
k z"
  using sums_polylog[of z k] by (subst (asm) sums_Suc_iff) auto

lemma polylog_altdef1:
  "norm z < 1 ⟹ polylog k z = (∑n. of_nat (Suc n) powi -k * z ^ Suc
n)"
  using sums_polylog[of z k] by (simp add: sums_iff)

lemma polylog_altdef2:
  "norm z < 1 ⟹ k ≠ 0 ⟹ polylog k z = (∑n. of_nat n powi -k * z
^ n)"
  using sums_polylog'[of z k] by (simp add: sums_iff)

lemma polylog_at_pole: "polylog k 1 = 0"
  by (auto simp: polylog_def)

lemma polylog_at_branch_cut: "x ≥ 1 ⟹ k > 0 ⟹ polylog k (of_real
x) = 0"
  by (auto simp: polylog_def)

lemma holomorphic_on_polylog [holomorphic_intros]:
  assumes "A ⊆ (if k ≤ 0 then -{1} else -of_real ` {1..})"
  shows   "polylog k holomorphic_on A"
proof -
  let ?S = "-(complex_of_real ` {1..})"
  have *: "open ?S"
    by (intro open_Compl closed_slot_right)
  have "polylog k holomorphic_on (if k ≤ 0 then -{1} else ?S)"
    by (subst holomorphic_on_open) (use * in ‹auto intro!: derivative_eq_intros
exI›)
  thus ?thesis
```

```
    by (rule holomorphic_on_subset) (use assms in ‹auto split: if_splits›)
qed

lemmas holomorphic_on_polylog' [holomorphic_intros] =
  holomorphic_on_compose_gen [OF _ holomorphic_on_polylog[OF order.refl],
unfolded o_def]

lemma analytic_on_polylog [analytic_intros]:
  assumes "A ⊆ (if k ≤ 0 then -{1} else -of_real ` {1..})"
  shows   "polylog k analytic_on A"
proof -
  let ?S = "-(complex_of_real ` {1..})"
  have *: "open ?S"
    by (intro open_Compl closed_slot_right)
  have "polylog k analytic_on (if k ≤ 0 then -{1} else ?S)"
    by (subst analytic_on_open) (use * in ‹auto intro!: holomorphic_intros›)
  thus ?thesis
    by (rule analytic_on_subset) (use assms in ‹auto split: if_splits›)
qed

lemmas analytic_on_polylog' [analytic_intros] =
  analytic_on_compose_gen [OF _ analytic_on_polylog[OF order.refl], unfolded
o_def]

lemma continuous_on_polylog [analytic_intros]:
  assumes "A ⊆ (if k ≤ 0 then -{1} else -of_real ` {1..})"
  shows   "continuous_on A (polylog k)"
proof -
  let ?S = "-(complex_of_real ` {1..})"
  have *: "open ?S"
    by (intro open_Compl closed_slot_right)
  have "continuous_on (if k ≤ 0 then -{1} else ?S) (polylog k)"
    by (intro holomorphic_on_imp_continuous_on holomorphic_intros) auto
  thus ?thesis
    by (rule continuous_on_subset) (use assms in auto)
qed

lemmas continuous_on_polylog' [continuous_intros] =
  continuous_on_compose2 [OF continuous_on_polylog [OF order.refl]]
```

## 2.2 Special values

```
lemma polylog_neg_int_left:
  "k < 0 ⟹ polylog k z = z * poly (eulerian_poly (nat (-k))) z * (1
- z) powi (k - 1)"
  by (auto simp: polylog_def)

lemma polylog_0_left: "polylog 0 z = z / (1 - z)"
  by (simp add: polylog_def field_simps)
```

```
lemma polylog_neg1_left: "polylog (-1) x = x / (1 - x) ^ 2"
  by (simp add: polylog_neg_int_left eval_nat_numeral eulerian_poly.simps
                power_int_minus field_simps)

lemma polylog_neg2_left: "polylog (-2) x = x * (1 + x) / (1 - x) ^ 3"
  by (simp add: polylog_neg_int_left eval_nat_numeral eulerian_poly.simps
                power_int_minus field_simps)

lemma polylog_neg3_left: "polylog (-3) x = x * (1 + 4 * x + x²) / (1
- x) ^ 4"
  by (simp add: polylog_neg_int_left eval_nat_numeral eulerian_poly.simps
Let_def pderiv_add
                pderiv_pCons power_int_minus field_simps numeral_poly)

lemma polylog_1:
  assumes "z ∉ of_real ` {1..}"
  shows   "polylog 1 z = -ln (1 - z)"
proof -
  have "(λz. polylog 1 z + ln (1 - z)) constant_on -of_real ` {1..}"
  proof (rule has_field_derivative_0_imp_constant_on)
    show "connected (-complex_of_real ` {1..})"
      using starlike_slotted_complex_plane_right[of 1] starlike_imp_connected
by blast
    show "open (- complex_of_real ` {1..})"
      using closed_slot_right by blast
    show "((λz. polylog 1 z + ln (1 - z)) has_field_derivative 0) (at
z)"
      if "z ∈ -of_real ` {1..}" for z
      using that
      by (auto intro!: derivative_eq_intros simp: complex_nonpos_Reals_iff
                       complex_slot_right_eq polylog_0_left divide_simps)
  qed
  then obtain c where c: "⋀z. z ∈ -of_real`{1..} ⟹ polylog 1 z + ln
(1 - z) = c"
    unfolding constant_on_def by blast
  from c[of 0] have "c = 0"
    by (auto simp: complex_slot_right_eq)
  with c[of z] show ?thesis
    using assms by (auto simp: add_eq_0_iff)
qed

lemma is_pole_polylog_1:
  assumes "k ≤ 0"
  shows   "is_pole (polylog k) 1"
proof (cases "k = 0")
  case True
  have "filtermap (λz. -z) (filtermap (λz. z - 1) (at 1)) = filtermap
(λz. -z) (at (0 :: complex))"
```

```
    by (simp add: at_to_0' filtermap_filtermap)
  also have "... = at 0"
    by (subst filtermap_at_minus) auto
  finally have "filtermap ((λz. -z) ∘ (λz. z - 1)) (at 1) = at (0 :: complex)"
    unfolding filtermap_compose .
  hence *: "filtermap (λz. 1 - z) (at 1) = at (0 :: complex)"
    by (simp add: o_def)

  have "is_pole (λz::complex. z / (1 - z)) 1"
    unfolding is_pole_def
    by (rule filterlim_divide_at_infinity tendsto_intros)+
       (use * in ‹auto simp: filterlim_def›)
  also have "(λz. z / (1 - z)) = polylog k"
    using True by (auto simp: fun_eq_iff polylog_0_left)
  finally show ?thesis .
next
  case False
  have "∀F x in at 1. x ≠ (1 :: complex)"
    using eventually_at zero_less_one by blast
  hence ev: "∀F x in at 1. 1 - x ≠ (0 :: complex)"
    by eventually_elim auto
  have "is_pole (λz::complex. z * poly (eulerian_poly (nat (- k))) z *
(1 - z) powi (k - 1)) 1"
    unfolding is_pole_def
    by (rule tendsto_mult_filterlim_at_infinity tendsto_eq_intros refl
ev
             filterlim_power_int_neg_at_infinity | (use assms in simp;
fail))+
  also have "(λz::complex. z * poly (eulerian_poly (nat (- k))) z * (1
- z) powi (k - 1)) =
             polylog k"
    using assms False by (intro ext) (simp add: polylog_neg_int_left)
  finally show ?thesis .
qed

lemma zorder_polylog_1:
  assumes "k ≤ 0"
  shows   "zorder (polylog k) 1 = k - 1"
proof (cases "k = 0")
  case True
  have "filtermap (λz. -z) (filtermap (λz. z - 1) (at 1)) = filtermap
(λz. -z) (at (0 :: complex))"
    by (simp add: at_to_0' filtermap_filtermap)
  also have "... = at 0"
    by (subst filtermap_at_minus) auto
  finally have "filtermap ((λz. -z) ∘ (λz. z - 1)) (at 1) = at (0 :: complex)"
    unfolding filtermap_compose .
  hence *: "filtermap (λz. 1 - z) (at 1) = at (0 :: complex)"
    by (simp add: o_def)
```

```
  have "zorder (λz::complex. (-z) / (z - 1) ^ 1) 1 = -int 1"
    by (rule zorder_nonzero_div_power [of UNIV]) (auto intro!: holomorphic_intros)
  also have "(λz. (-z) / (z - 1) ^ 1) = polylog k"
    using True by (auto simp: fun_eq_iff polylog_0_left divide_simps)
(auto simp: algebra_simps)?
  finally show ?thesis
    using True by simp
next
  case False
  have "zorder (λz::complex. (-1) ^ nat (1 - k) * z * poly (eulerian_poly
(nat (- k))) z /
                 (z - 1) ^ nat (1 - k)) 1 = -int (nat (1 - k))" (is "zorder
?f _ = _")
    using False assms
    by (intro zorder_nonzero_div_power [of UNIV]) (auto intro!: holomorphic_intros)
  also have "?f = polylog k"
  proof
    fix z :: complex
    have "(z - 1) ^ nat (1 - k) = (-1) ^ nat (1 - k) * (1 - z) ^ nat (1
- k)"
      by (subst power_mult_distrib [symmetric]) auto
    thus "?f z = polylog k z"
      using False assms by (auto simp: polylog_neg_int_left power_int_def
field_simps)
  qed
  finally show ?thesis
    using False assms by simp
qed

lemma isolated_singularity_polylog_1:
  assumes "k ≤ 0"
  shows   "isolated_singularity_at (polylog k) 1"
  unfolding isolated_singularity_at_def using assms
  by (intro exI[of _ 1]) (auto intro!: analytic_intros)

lemma not_essential_polylog_1:
  assumes "k ≤ 0"
  shows   "not_essential (polylog k) 1"
  unfolding not_essential_def using is_pole_polylog_1[of k] assms by auto

lemma polylog_meromorphic_on [meromorphic_intros]:
  assumes "k ≤ 0"
  shows   "polylog k meromorphic_on {1}"
  using assms
  by (simp add: isolated_singularity_polylog_1 meromorphic_at_iff not_essential_polylog_1)
```

## 2.3 Duplication formula

Lastly, we prove the following duplication formula that the polylogarithm satisfies:

$$\mathrm{Li}_s(z) + \mathrm{Li}_s(-z) = 2^{1-s}\mathrm{Li}_s(z^2)$$

The proof is a relatively simple manipulation of infinite sum that defines $\mathrm{Li}_s(z)$ for $|z| < 1$, followed by analytic continuation to its full domain.

```
theorem polylog_duplication:
  assumes "if s ≤ 0 then z ∉ {-1, 1} else z ∉ complex_of_real ` ({..-1}
∪ {1..})"
  shows   "polylog s z + polylog s (-z) = 2 powi (1 - s) * polylog s (z²)"
proof -
  define A where "A = -(if s ≤ 0 then {-1, 1} else complex_of_real `
({..-1} ∪ {1..}))"
  show ?thesis
  proof (rule analytic_continuation_open[where f = "λz. polylog s z +
polylog s (-z)"])
    show "ball 0 1 ⊆ A"
      by (auto simp: A_def)
  next
    have "closed (complex_of_real ` ({..-1} ∪ {1..}))"
      unfolding image_Un by (intro open_Compl closed_Un closed_slot_right
closed_slot_left)
    thus "open A"
      unfolding A_def by auto
  next
    have "connected (-complex_of_real ` ({..-1} ∪ {1..}))"
      by (intro simply_connected_imp_connected simply_connected_doubly_slotted_complex_plan
auto
    moreover have "connected (-{-1, 1 :: complex})"
      by (intro path_connected_imp_connected path_connected_complement_countable)
auto
    ultimately show "connected A"
      unfolding A_def by auto
  next
    show "(λz. polylog s z + polylog s (- z)) holomorphic_on A"
      by (intro holomorphic_intros) (auto simp: complex_eq_iff A_def)
  next
    show "(λz. 2 powi (1 - s) * polylog s (z²)) holomorphic_on A"
    proof (intro holomorphic_intros; safe)
      fix z assume z: "z ∈ A"
      show "z^2 ∈ (if s ≤ 0 then - {1} else - complex_of_real ` {1..})"
      proof (cases "s ≤ 0")
        case True
        thus ?thesis using z by (auto simp: A_def power2_eq_1_iff)
      next
        case False
        {
```

```
        fix x :: real
        assume x: "x ≥ 1" "z ^ 2 = of_real x"
        have "Im (z ^ 2) = 0"
          by (simp add: x)
        hence "Im z = 0 ∨ Re z = 0"
          by (simp add: power2_eq_square)
        moreover have "Im z ^ 2 ≥ 0"
          by auto
        hence "Im z ^ 2 > -1"
          by linarith
        ultimately have "x = Re z ^ 2" "Im z = 0"
          using x unfolding power2_eq_square by (auto simp: complex_eq_iff)
        with x have "|Re z| ≥ 1"
          by (auto simp: power2_ge_1_iff)
        with ‹Im z = 0› have "z ∉ A"
          using False by (auto simp: A_def complex_double_slot_eq)
      }
      with False show ?thesis using z
        by (auto simp: A_def)
    qed
  qed
next
  show "polylog s z + polylog s (-z) = 2 powi (1 - s) * polylog s (z²)"
    if z: "z ∈ ball 0 1" for z
  proof -
    have ran: "range (λn::nat. Suc (2 * n)) = {n. odd n}"
      by (auto simp: image_def elim!: oddE)
    have "(λn. of_nat (Suc n) powi -s * (z ^ Suc n + (-z) ^ Suc n))
sums
            (polylog s z + polylog s (-z))" (is "?f sums _")
      unfolding ring_distribs using z
      by (intro sums_add sums_mult sums_polylog) (simp_all add: norm_power)
    also have "?this ⟷ (λn. ?f (2 * n + 1)) sums (polylog s z + polylog
s (-z))"
      by (rule sym, intro sums_mono_reindex) (auto simp: ran strict_mono_def)
    also have "(λn. ?f (2 * n + 1)) = (λn. 2 * (2 * of_nat (Suc n))
powi -s * (z²) ^ Suc n)"
      by (intro ext) (simp_all add: algebra_simps power_mult power2_eq_square
power_minus')
    also have "... = (λn. 2 powi (1 - s) * (of_nat (Suc n) powi -s *
(z²) ^ Suc n))" (is "_ = ?g")
      by (simp add: power_int_diff power_int_minus fun_eq_iff field_simps
              flip: power_int_mult_distrib)
    finally have "?g sums (polylog s z + polylog s (-z))" .
    moreover have "?g sums (2 powi (1 - s) * polylog s (z²))"
      using z by (intro sums_mult sums_polylog) (simp_all add: norm_power
abs_square_less_1)
    ultimately show ?thesis
      using sums_unique2 by blast
```

```
    qed
  qed (use assms in ‹auto simp: A_def›)
qed

end
```

## References

[1] J. Mason and D. Handscomb. *Chebyshev Polynomials.* CRC Press, 2002.