Modular Forms

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

November 9, 2026

Abstract

This entry provides the definition and proofs of basic properties of modular forms and modular functions. These are holomorphic or meromorphic functions in the complex upper half plane that satisfy certain symmetries with respect to the modular group $\mathrm{SL}_2(\mathbb{Z})$ or a subgroup thereof. They are a cornerstone of modern number theory and an essential ingredient in the proof of Fermat's Last Theorem.

Most of the work focuses on level 1 modular forms/functions (i.e. the full modular group rather than a proper subgroup). Some notable results include:

- Infrastructure to reason about q-expansions of periodic functions.
- The valence formula for level 1 meromorphic forms, giving the number of zeros and poles (including both the special cases of modular forms and modular functions).
- Basic level 1 modular forms such as the Eisenstein series E_k and the modular discriminant Δ.
- The Ramanujan τ function and the identity $\Delta = (2\pi)^{12}\eta^2 4$ relating the modular discriminant to the Dedekind η function.
- Properties of Klein's J invariant, including bijectivity, non-conformality, covering properties, and that the level 1 modular functions are exactly the rational functions of J.
- The graded ring structure of the space of modular forms $\mathcal{M}_k$, including its dimension and explicit bases in terms of E_4, E_6, and Δ.
- Zagier's theory of level 1 quasimodular forms, including the structure theorem in terms of E_2, E_4, and E_6.
- The Serre derivative operator.

A number of applications are also provided:

- Ramanujan-type identities for the divisor σ functions (using the Serre derivative and the structure theorem for $\mathcal{M}_k$), e.g.

$$20\sigma_3(n) = (24n - 4)\sigma_1(n) + 48\sum_{i=1}^{n-1}\sigma_1(i)\sigma_1(n-i)$$

- The Eisenstein inversion theorem, relating elliptic curves to lattices (using the properties of Klein's J invariant).
- A short proof of Picard's Little Theorem using the fact that J is a covering map.

The proofs and definitions mostly follow Apostol's *Modular Functions and Dirichlet Series in Number Theory* [1].

Contents

1 The type of meromorphic functions in the upper half plane 5

2 q-expansions of periodic functions 30
2.1 The zero order at the cusp . . . 30
2.2 Expansion in terms of q . . . 31
2.3 Meromorphicity at infinity . . . 36
2.4 Holomorphicity at infinity . . . 41
2.5 Closure properties . . . 44

3 Definition of modular forms and related concepts 56
3.1 Weakly meromorphic forms . . . 57
3.2 Meromorphic forms . . . 62
3.3 Modular forms . . . 69
3.4 Cusp forms . . . 75

4 The valence formula for level 1 meromorphic forms 78
4.1 Constructing the deformed path . . . 81
4.1.1 The vertical segments . . . 85
4.1.2 The circular arcs . . . 86
4.2 The corners . . . 88
4.3 Putting everything together . . . 89

5 Some concrete level 1 modular forms 92
5.1 Eisenstein series . . . 92
5.2 Modular discriminant . . . 94
5.3 Klein's J invariant . . . 95

6 Level 1 modular functions 96
6.1 The weighted multiplicity of a zero/pole . . . 96
6.2 Degree . . . 99
6.3 Range . . . 100
6.4 Surjectivity and injectivity . . . 101

7 Properties of Klein's J invariant 102
7.1 Bijectivity . . . 103
7.2 Modular functions as rational functions of J . . . 105
7.3 Non-conformality of modular functions at i and ϱ . . . 105
7.4 Real values . . . 108
7.5 Inverse function . . . 112
7.6 Covering properties . . . 113
7.7 Applications . . . 114
7.8 The Eisenstein inversion problem . . . 114
7.9 A short proof of Picard's Little Theorem . . . 114

8 The Serre derivative **115**

9 The graded ring structure of level 1 modular forms **118**
9.1 Auxiliary material 119
9.1.1 Linear diophantine inequalities in two variables 119
9.1.2 Independent families of vectors and indexed bases . . 119
9.2 Basic structure lemmas 121
9.3 The standard basis $E_{k-12i}\Delta^i$ 123
9.4 The alternative basis $E_4^i E_6^j$ 124
9.5 A basis for cusp forms 124
9.6 Connection to Dedekind's η function 125

10 Zagier's theory of quasimodular forms **125**
10.1 Quasimodular functions 126
10.2 Quasimodular forms 130

11 Application: Ramanujan-style identities for σ **132**

⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩
⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨*proof*⟩⟨p

1 The type of meromorphic functions in the upper half plane

theory *Meromorphic_Upper_Half_Plane*
imports
"HOL-Complex_Analysis.Complex_Analysis"
"HOL-Library.Going_To_Filter"
"Polynomial_Interpolation.Ring_Hom"
"Elliptic_Functions.Modular_Fundamental_Region"
Modular_Forms_Library
begin

definition *is_mero_uhp :: "(complex ⇒ complex) ⇒ bool"* **where**
"is_mero_uhp f ⟷
f nicely_meromorphic_on {z. Im z > 0} ∧
(∀z. Im z ≤ 0 ∨ is_pole f z ⟶ f z = 0)"

typedef *mero_uhp =*
"{f :: complex ⇒ complex. is_mero_uhp f}"
morphisms *eval_mero_uhp Abs_mero_uhp*
⟨*proof*⟩

setup_lifting *type_definition_mero_uhp*

lemma *nicely_meromorphic_on_cong:*
assumes *"⋀z. z ∈ A ⟹ f z = g z" "open A"*
assumes *"A = B"*
shows *"f nicely_meromorphic_on A ⟷ g nicely_meromorphic_on B"*
⟨*proof*⟩

lift_definition *mero_uhp :: "(complex ⇒ complex) ⇒ mero_uhp"* **is**
"λf. if f meromorphic_on {z. Im z > 0}
then (λz. if Im z > 0 ∧ ¬is_pole f z then remove_sings f z else
0)
else (λ_. 0)"
⟨*proof*⟩

lemma *mero_uhp_cong_weak: "(⋀z. Im z > 0 ⟹ f z = g z) ⟹ mero_uhp*
f = mero_uhp g"
⟨*proof*⟩

lemma *eval_mero_uhp_outside: "Im z ≤ 0 ⟹ eval_mero_uhp f z = 0"*
⟨*proof*⟩

lemma *eval_mero_uhp_pole: "is_pole (eval_mero_uhp f) z ⟹ eval_mero_uhp*

```
f z = 0"
```
⟨*proof*⟩

lemma *eval_mero_uhp_mero_uhp_eq:*
assumes *"f meromorphic_on {z. Im z > 0}" "Im z > 0"*
shows *"eval_mero_uhp (mero_uhp f) z = remove_sings f z"*
⟨*proof*⟩

lemma *eval_mero_uhp_mero_uhp:*
assumes *"f meromorphic_on {z. Im z > 0}" "f analytic_on {z}" "Im z > 0"*
shows *"eval_mero_uhp (mero_uhp f) z = f z"*
⟨*proof*⟩

lemma *eval_mero_uhp_meromorphic:*
"eval_mero_uhp f meromorphic_on {z. Im z > 0}"
⟨*proof*⟩

lemma *eval_mero_uhp_meromorphic':*
"A ⊆ {z. Im z > 0} ⟹ eval_mero_uhp f meromorphic_on A"
⟨*proof*⟩

lemma *eventually_eval_mero_uhp_mero_uhp_eq:*
assumes *"f meromorphic_on {z. Im z > 0}" "Im z > 0"*
shows *"eventually (λw. eval_mero_uhp (mero_uhp f) w = f w) (at z)"*
⟨*proof*⟩

lemma *is_pole_eval_mero_uhp_mero_uhp_iff:*
assumes *"f meromorphic_on {z. Im z > 0}" "Im z > 0"*
shows *"is_pole (eval_mero_uhp (mero_uhp f)) z ⟷ is_pole f z"*
⟨*proof*⟩

lemma *eval_mero_uhp_nicely_meromorphic:*
"A ⊆ {z. Im z > 0} ⟹ eval_mero_uhp f nicely_meromorphic_on A"
⟨*proof*⟩

lemma *eval_mero_uhp_analytic:*
assumes *"¬is_pole (eval_mero_uhp f) z" "Im z > 0"*
shows *"eval_mero_uhp f analytic_on {z}"*
⟨*proof*⟩

lemma *not_is_pole_eval_mero_uhp_outside:*
assumes *"Im z ≤ 0"*
shows *"¬is_pole (eval_mero_uhp f) z"*
⟨*proof*⟩

lemma *meromorphic_on_eval_mero_uhp' [meromorphic_intros]:*
assumes *"g analytic_on A" "⋀z. z ∈ A ⟹ Im (g z) > 0"*
shows *"(λw. eval_mero_uhp f (g w)) meromorphic_on A"*

⟨*proof*⟩

lemma *analytic_on_eval_mero_uhp [analytic_intros]:*
"(⋀z. z ∈ A ⟹ Im z > 0) ⟹ (⋀z. z ∈ A ⟹ ¬is_pole (eval_mero_uhp f) z) ⟹
eval_mero_uhp f analytic_on A"
⟨*proof*⟩

lemma *holomorphic_on_eval_mero_uhp [holomorphic_intros]:*
"(⋀z. z ∈ A ⟹ Im z > 0) ⟹ (⋀z. z ∈ A ⟹ ¬is_pole (eval_mero_uhp f) z) ⟹
eval_mero_uhp f holomorphic_on A"
⟨*proof*⟩

lemma *analytic_on_eval_mero_uhp' [analytic_intros]:*
assumes *"g analytic_on A" "⋀z. z ∈ A ⟹ Im (g z) > 0 ∧ ¬is_pole (eval_mero_uhp f) (g z)"*
shows *"(λw. eval_mero_uhp f (g w)) analytic_on A"*
⟨*proof*⟩

lemma *holomorphic_on_eval_mero_uhp' [holomorphic_intros]:*
assumes *"g holomorphic_on A" "⋀z. z ∈ A ⟹ Im (g z) > 0 ∧ ¬is_pole (eval_mero_uhp f) (g z)"*
shows *"(λw. eval_mero_uhp f (g w)) holomorphic_on A"*
⟨*proof*⟩

definition *const_mero_uhp :: "complex ⇒ mero_uhp"* **where**
"const_mero_uhp c = mero_uhp (λ_. c)"

lemma *eval_const_mero_uhp [simp]: "Im z > 0 ⟹ eval_mero_uhp (const_mero_uhp c) z = c"*
⟨*proof*⟩

lemma *not_pole_const_mero_uhp [simp]: "Im z > 0 ⟹ ¬is_pole (eval_mero_uhp (const_mero_uhp c)) z"*
⟨*proof*⟩

declare *[[coercion eval_mero_uhp]]*

lemma *not_essential_frequently_0_imp_eventually_0':*
fixes *f :: "complex ⇒ complex"*
assumes *sing: "isolated_singularity_at f z" "not_essential f z"*
assumes *freq: "frequently (λz. f z = 0) (at z within A)"*
shows *"eventually (λz. f z = 0) (at z within B)"*
⟨*proof*⟩

lemma *frequently_eq_at_imp_eq_at:*
assumes *"frequently (λz. f z = g z) (at z within A)"*

assumes *"f analytic_on {z}" "g analytic_on {z}"*
shows *"f z = g z"*
⟨*proof*⟩

lemma *mero_uhp_eqI_strong:*
fixes *f g :: mero_uhp*
assumes *"frequently (λz. eval_mero_uhp f z = eval_mero_uhp g z) (cosparse {z. Im z > 0})"*
shows *"f = g"*
⟨*proof*⟩

lemma *mero_uhp_eqI_strong':*
fixes *f g :: mero_uhp*
assumes *"frequently (λz. eval_mero_uhp f z = eval_mero_uhp g z) (at z0)" "Im z0 > 0"*
shows *"f = g"*
⟨*proof*⟩

lemma *mero_uhp_eqI:*
fixes *f g :: mero_uhp*
assumes *"eventually (λz. eval_mero_uhp f z = eval_mero_uhp g z) (cosparse {z. Im z > 0})"*
shows *"f = g"*
⟨*proof*⟩

lemma *mero_uhp_eqI_weak:*
fixes *f g :: mero_uhp*
assumes *"⋀z. Im z > 0 ⟹ eval_mero_uhp f z = eval_mero_uhp g z"*
shows *"f = g"*
⟨*proof*⟩

lemma *frequently_eval_mero_uhp_eq_imp_const:*
assumes *"frequently (λw. eval_mero_uhp f w = c) (at z)" "Im z > 0"*
shows *"f = const_mero_uhp c"*
⟨*proof*⟩

lemma *eventually_neq_eval_mero_uhp:*
assumes *"f ≠ const_mero_uhp c" "Im z > 0"*
shows *"eventually (λz. eval_mero_uhp f z ≠ c) (at z)"*
⟨*proof*⟩

definition *mero_uhp_rel* **where**
"mero_uhp_rel f g ⟷ eventually (λz. f z = g z) (cosparse {z. Im z > 0})"

named_theorems *mero_uhp_rel_intros*

lemma *mero_uhp_rel_refl [simp, intro]: "mero_uhp_rel f f"*
⟨*proof*⟩

lemma *mero_uhp_rel_sym: "mero_uhp_rel f g ⟹ mero_uhp_rel g f"*
⟨*proof*⟩

lemma *mero_uhp_rel_sym_eq: "mero_uhp_rel f g ⟷ mero_uhp_rel g f"*
⟨*proof*⟩

lemma *mero_uhp_rel_trans [trans]: "mero_uhp_rel f g ⟹ mero_uhp_rel g h ⟹ mero_uhp_rel f h"*
⟨*proof*⟩

lemma *mero_uhp_relI_weak:*
assumes *"⋀z. Im z > 0 ⟹ f z = g z"*
shows *"mero_uhp_rel f g"*
⟨*proof*⟩

lemma *mero_uhp_rel_mero_uhp [mero_uhp_rel_intros]:*
assumes *"f meromorphic_on {z. Im z > 0}"*
shows *"mero_uhp_rel (eval_mero_uhp (mero_uhp f)) f"*
⟨*proof*⟩

lemma *mero_uhp_rel_imp_eq_mero_uhp:*
"mero_uhp_rel (eval_mero_uhp f) (eval_mero_uhp g) ⟹ f = g"
⟨*proof*⟩

definition *deriv_mero_uhp :: "mero_uhp ⇒ mero_uhp"* **where**
"deriv_mero_uhp f = mero_uhp (deriv f)"

lemma *mero_uhp_rel_unop:*
assumes *"mero_uhp_rel f f'"*
shows *"mero_uhp_rel (λz. h (f z)) (λz. h (f' z))"*
⟨*proof*⟩

lemma *mero_uhp_rel_binop:*
assumes *"mero_uhp_rel f f'" "mero_uhp_rel g g'"*
shows *"mero_uhp_rel (λz. h (f z) (g z)) (λz. h (f' z) (g' z))"*
⟨*proof*⟩

named_theorems *mero_uhp_rel_cong*

lemmas *mero_uhp_rel_cong_uminus [mero_uhp_rel_cong] = mero_uhp_rel_unop[***where** *h = "λx. -x"]*
lemmas *mero_uhp_rel_cong_inverse [mero_uhp_rel_cong] = mero_uhp_rel_unop[***where** *h = "inverse"]*

lemmas *mero_uhp_rel_cong_add [mero_uhp_rel_cong] = mero_uhp_rel_binop[***where** *h = "(+)"]*
lemmas *mero_uhp_rel_cong_diff [mero_uhp_rel_cong] = mero_uhp_rel_binop[***where** *h = "(-)"]*
lemmas *mero_uhp_rel_cong_mult [mero_uhp_rel_cong] = mero_uhp_rel_binop[***where** *h = "(*)"]*
lemmas *mero_uhp_rel_cong_divide [mero_uhp_rel_cong] = mero_uhp_rel_binop[***where** *h = "(/)"]*
lemmas *mero_uhp_rel_cong_power [mero_uhp_rel_cong] = mero_uhp_rel_unop[***where** *h = "λz. z ^ n"* **for** *n]*
lemmas *mero_uhp_rel_cong_powi [mero_uhp_rel_cong] = mero_uhp_rel_unop[***where** *h = "λz. z powi n"* **for** *n]*

lemma *mero_uhp_rel_cong_sum [mero_uhp_rel_cong]:*
"(⋀x. x ∈ A ⟹ mero_uhp_rel (f x) (f' x)) ⟹
mero_uhp_rel (λz. sum (λx. f x z) A) (λz. sum (λx. f' x z) A)"
⟨*proof*⟩

lemma *mero_uhp_rel_cong_prod [mero_uhp_rel_cong]:*
"(⋀x. x ∈ A ⟹ mero_uhp_rel (f x) (f' x)) ⟹
mero_uhp_rel (λz. prod (λx. f x z) A) (λz. prod (λx. f' x z) A)"
⟨*proof*⟩

lemma *sparse_in_union':*
assumes *"pts1 sparse_in D" "pts2 sparse_in D"*
shows *"(pts1 ∪ pts2) sparse_in D"*
⟨*proof*⟩

lemma *mero_uhp_rel_cong_deriv [mero_uhp_rel_cong]:*
assumes *"mero_uhp_rel f g"*
shows *"mero_uhp_rel (deriv f) (deriv g)"*
⟨*proof*⟩

lemma *mero_uhp_rel_cong_higher_deriv [mero_uhp_rel_cong]:*
"mero_uhp_rel f g ⟹ mero_uhp_rel ((deriv ^^ n) f) ((deriv ^^ n) g)"
⟨*proof*⟩

lemma *mero_uhp_rel_simpI: "mero_uhp_rel f h ⟹ mero_uhp_rel g h' ⟹ h = h' ⟹ mero_uhp_rel f g"*
⟨*proof*⟩

lemma *mero_uhp_rel_eval_mero_uhp_const [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (const_mero_uhp c)) (λ_. c)"
⟨*proof*⟩

lemma *eval_deriv_mero_uhp:*
assumes *"Im z > 0" "¬is_pole (eval_mero_uhp f) z"*

shows *"eval_mero_uhp (deriv_mero_uhp f) z = deriv (eval_mero_uhp f) z"*
⟨*proof*⟩

lemma *is_pole_deriv_mero_uhp_iff:*
assumes *"Im z > 0"*
shows *"is_pole (eval_mero_uhp (deriv_mero_uhp f)) z ⟷ is_pole (eval_mero_uhp f) z"*
⟨*proof*⟩

lemma *mero_uhp_rel_imp_eval_mero_uhp_eq:*
assumes *"mero_uhp_rel (eval_mero_uhp f) g"*
assumes *"g analytic_on {z}" "Im z > 0"*
shows *"eval_mero_uhp f z = g z"*
⟨*proof*⟩

lemma *mero_uhp_rel_const_refl: "mero_uhp_rel (λ_. c) (λ_. c)"*
⟨*proof*⟩

⟨*ML*⟩

lemma *mero_uhp_eval_mero_uhp [simp]: "mero_uhp (eval_mero_uhp f) = f"*
⟨*proof*⟩

lemma *mero_uhp_rel_deriv [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (deriv_mero_uhp f)) (deriv (eval_mero_uhp f))"
⟨*proof*⟩

lemma *mero_uhp_rel_higher_deriv [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp ((deriv_mero_uhp ^^ n) f)) ((deriv ^^ n) (eval_mero_uhp f))"
⟨*proof*⟩

lemma *has_field_derivative_deriv_mero_uhp [derivative_intros]:*
assumes *"Im z > 0" "¬is_pole (eval_mero_uhp f) z"*
shows *"(eval_mero_uhp f has_field_derivative eval_mero_uhp (deriv_mero_uhp f) z) (at z)"*
⟨*proof*⟩

lemma *has_field_derivative_deriv_mero_uhp' [derivative_intros]:*
assumes *"(g has_field_derivative g') (at z within A)"*
assumes *"Im (g z) > 0" "¬is_pole (eval_mero_uhp f) (g z)"*
shows *"((λz. eval_mero_uhp f (g z)) has_field_derivative eval_mero_uhp (deriv_mero_uhp f) (g z) * g') (at z within A)"*
⟨*proof*⟩

instantiation *mero_uhp :: comm_ring_1*
begin

definition *zero_mero_uhp* **where** *"zero_mero_uhp = const_mero_uhp 0"*

definition *one_mero_uhp* **where** *"one_mero_uhp = const_mero_uhp 1"*

definition *plus_mero_uhp :: "mero_uhp ⇒ mero_uhp ⇒ mero_uhp"* **where**
"f + g = mero_uhp (λz. eval_mero_uhp f z + eval_mero_uhp g z)"

definition *uminus_mero_uhp :: "mero_uhp ⇒ mero_uhp"* **where**
"-f = mero_uhp (λz. -eval_mero_uhp f z)"

definition *minus_mero_uhp :: "mero_uhp ⇒ mero_uhp ⇒ mero_uhp"* **where**
"f - g = mero_uhp (λz. eval_mero_uhp f z - eval_mero_uhp g z)"

definition *times_mero_uhp :: "mero_uhp ⇒ mero_uhp ⇒ mero_uhp"* **where**
*"f * g = mero_uhp (λz. eval_mero_uhp f z * eval_mero_uhp g z)"*

lemma *mero_uhp_rel_eval_mero_uhp_0 [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp 0) (λ_. 0)"
and *mero_uhp_rel_eval_mero_uhp_1 [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp 1) (λ_. 1)"
and *mero_uhp_rel_eval_mero_uhp_minus [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (-f)) (λz. -eval_mero_uhp f z)"
and *mero_uhp_rel_eval_mero_uhp_add [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (f + g)) (λz. eval_mero_uhp f z + eval_mero_uhp g z)"
and *mero_uhp_rel_eval_mero_uhp_diff [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (f - g)) (λz. eval_mero_uhp f z - eval_mero_uhp g z)"
and *mero_uhp_rel_eval_mero_uhp_times [mero_uhp_rel_intros]:*
*"mero_uhp_rel (eval_mero_uhp (f * g)) (λz. eval_mero_uhp f z * eval_mero_uhp g z)"*
⟨*proof*⟩

lemma *eval_mero_uhp_0 [simp]: "eval_mero_uhp 0 = (λ_. 0)"*
⟨*proof*⟩

lemma *eval_mero_uhp_1 [simp]: "Im z > 0 ⟹ eval_mero_uhp 1 z = 1"*
⟨*proof*⟩

lemma *not_pole_0_mero_uhp [simp]: "Im z > 0 ⟹ ¬is_pole (eval_mero_uhp 0) z"*
and *not_pole_1_mero_uhp [simp]: "Im z > 0 ⟹ ¬is_pole (eval_mero_uhp 1) z"*
⟨*proof*⟩

lemma
assumes *"Im z > 0" "¬is_pole (eval_mero_uhp f) z" "¬is_pole (eval_mero_uhp g) z"*
shows *eval_mero_uhp_add [simp]: "eval_mero_uhp (f + g) z = eval_mero_uhp f z + eval_mero_uhp g z"*
and *eval_mero_uhp_diff [simp]: "eval_mero_uhp (f - g) z = eval_mero_uhp f z - eval_mero_uhp g z"*
and *eval_mero_uhp_mult [simp]: "eval_mero_uhp (f * g) z = eval_mero_uhp f z * eval_mero_uhp g z"*
⟨*proof*⟩

instance ⟨*proof*⟩

end

lemma *eval_mero_uhp_cmult [simp]:*
*"eval_mero_uhp (const_mero_uhp c * f) z = c * eval_mero_uhp f z"*
⟨*proof*⟩

lemma *eval_mero_uhp_cmult' [simp]:*
*"eval_mero_uhp (f * const_mero_uhp c) z = eval_mero_uhp f z * c"*
⟨*proof*⟩

lemma *const_mero_uhp_eq_iff [simp]: "const_mero_uhp c = const_mero_uhp c' ⟷ c = c'"*
⟨*proof*⟩

lemma *const_mero_uhp_eq_0_iff [simp]: "const_mero_uhp c = 0 ⟷ c = 0"*
⟨*proof*⟩

lemma *const_mero_uhp_eq_1_iff [simp]: "const_mero_uhp c = 1 ⟷ c = 1"*
⟨*proof*⟩

interpretation *const_mero_uhp: comm_ring_hom const_mero_uhp*
⟨*proof*⟩

lemma *eval_mero_uhp_minus [simp]: "eval_mero_uhp (-f) z = -eval_mero_uhp f z"*
⟨*proof*⟩

lemma *of_nat_mero_uhp: "of_nat n = const_mero_uhp (of_nat n)"*
⟨*proof*⟩

lemma *of_int_mero_uhp: "of_int n = const_mero_uhp (of_int n)"*
⟨*proof*⟩

lemma *numeral_mero_uhp: "numeral n = const_mero_uhp (numeral n)"*

⟨*proof*⟩

lemma *mero_uhp_rel_of_nat [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (of_nat n)) (λ_. of_nat n)"
⟨*proof*⟩

lemma *mero_uhp_rel_of_int [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (of_int n)) (λ_. of_int n)"
⟨*proof*⟩

lemma *mero_uhp_rel_numeral [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (numeral n)) (λ_. numeral n)"
⟨*proof*⟩

lemma *mero_uhp_rel_sum [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (∑x∈A. f x)) (λz. ∑x∈A. eval_mero_uhp (f x) z)"
⟨*proof*⟩

lemma *mero_uhp_rel_prod [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (∏x∈A. f x)) (λz. ∏x∈A. eval_mero_uhp (f x) z)"
⟨*proof*⟩

lemma *mero_uhp_rel_power [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (f ^ n)) (λz. eval_mero_uhp f z ^ n)"
⟨*proof*⟩

instantiation *mero_uhp :: field_char_0*
begin

definition *inverse_mero_uhp :: "mero_uhp ⇒ mero_uhp"* **where**
"inverse f = mero_uhp (λz. inverse (eval_mero_uhp f z))"

definition *divide_mero_uhp :: "mero_uhp ⇒ mero_uhp ⇒ mero_uhp"* **where**
*"divide_mero_uhp f g = f * inverse g"*

lemma *mero_uhp_rel_eval_mero_uhp_inverse [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (inverse f)) (λz. inverse (eval_mero_uhp f z))"
⟨*proof*⟩

lemma *mero_uhp_rel_eval_mero_uhp_divide [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (f div g)) (λz. eval_mero_uhp f z / eval_mero_uhp g z)"
⟨*proof*⟩

lemma *eval_mero_uhp_inverse [simp]:*

assumes *"Im z > 0"*
shows *"eval_mero_uhp (inverse f) z = inverse (eval_mero_uhp f z)"*
⟨*proof*⟩

instance ⟨*proof*⟩

end

lemma *eval_mero_uhp_divide [simp]:*
assumes *"Im z > 0" "¬is_pole (eval_mero_uhp f) z" "¬isolated_zero (eval_mero_uhp g) z"*
shows *"eval_mero_uhp (f / g) z = eval_mero_uhp f z / eval_mero_uhp g z"*
⟨*proof*⟩

interpretation *const_mero_uhp: comm_ring_hom const_mero_uhp*
⟨*proof*⟩

interpretation *const_mero_uhp: field_char_0_hom const_mero_uhp*
⟨*proof*⟩

lemma *mero_uhp_rel_power_int [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (f powi n)) (λz. eval_mero_uhp f z powi n)"
⟨*proof*⟩

instantiation *mero_uhp :: real_field*
begin

definition *scaleR_mero_uhp :: "real ⇒ mero_uhp ⇒ mero_uhp"*
where *"scaleR_mero_uhp x f = const_mero_uhp x * f"*

instance
⟨*proof*⟩

end

lemma *of_real_mero_uhp: "of_real x = const_mero_uhp (of_real x)"*
⟨*proof*⟩

lemma *mero_uhp_rel_of_real [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (of_real x)) (λ_. of_real x)"
⟨*proof*⟩

lemma *of_rat_mero_uhp: "of_rat x = const_mero_uhp (of_rat x)"*
⟨*proof*⟩

Pre-modular forms are a complex vector space:

interpretation *mero_uhp: vector_space "λ(x::complex) y. const_mero_uhp x * y"*
⟨*proof*⟩

definition *inv_image_mero_uhp :: "mero_uhp ⇒ complex ⇒ complex set"* **where**
"inv_image_mero_uhp f c = {z∈$\mathcal{R}_\Gamma$'. ¬is_pole f z ∧ eval_mero_uhp f z = c}"

abbreviation *zeros_mero_uhp :: "mero_uhp ⇒ complex set"* **where**
"zeros_mero_uhp f ≡ inv_image_mero_uhp f 0"

definition *poles_mero_uhp :: "mero_uhp ⇒ complex set"* **where**
"poles_mero_uhp f = {z∈$\mathcal{R}_\Gamma$'. is_pole f z}"
lemma *inv_image_mero_uhp_conv_zeros:*
"inv_image_mero_uhp f c = zeros_mero_uhp (f - const_mero_uhp c)"
⟨*proof*⟩

definition *is_const_mero_uhp :: "mero_uhp ⇒ bool"*
where *"is_const_mero_uhp f ⟷ f ∈ range const_mero_uhp"*

lemma *is_const_mero_uhp_const_mero_uhp [simp, intro]: "is_const_mero_uhp (const_mero_uhp c)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_0 [simp, intro]: "is_const_mero_uhp 0"*
⟨*proof*⟩

lemma *is_const_mero_uhp_1 [simp, intro]: "is_const_mero_uhp 1"*
⟨*proof*⟩

lemma *is_const_mero_uhp_of_nat [simp, intro]: "is_const_mero_uhp (of_nat n)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_of_int [simp, intro]: "is_const_mero_uhp (of_int n)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_of_real [simp, intro]: "is_const_mero_uhp (of_real x)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_numeral [simp, intro]: "is_const_mero_uhp (numeral n)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_add [intro]:*

assumes *"is_const_mero_uhp f" "is_const_mero_uhp g"*
shows *"is_const_mero_uhp (f + g)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_mult [intro]:*
assumes *"is_const_mero_uhp f" "is_const_mero_uhp g"*
shows *"is_const_mero_uhp (f * g)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_uminus [intro]:*
assumes *"is_const_mero_uhp f"*
shows *"is_const_mero_uhp (-f)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_power_int [intro]:*
assumes *"is_const_mero_uhp f"*
shows *"is_const_mero_uhp (f powi n)"*
⟨*proof*⟩

lemma *is_const_mero_uhp_power [intro]:*
"is_const_mero_uhp f ⟹ is_const_mero_uhp (f ^ n)"
⟨*proof*⟩

lemma *is_const_mero_uhp_inverse [intro]:*
"is_const_mero_uhp f ⟹ is_const_mero_uhp (inverse f)"
⟨*proof*⟩

lemma *is_const_mero_uhp_diff [intro]:*
"is_const_mero_uhp f ⟹ is_const_mero_uhp g ⟹ is_const_mero_uhp (f - g)"
⟨*proof*⟩

lemma *is_const_mero_uhp_divide [intro]:*
"is_const_mero_uhp f ⟹ is_const_mero_uhp g ⟹ is_const_mero_uhp (f / g)"
⟨*proof*⟩

lemma *is_const_mero_uhp_minus_iff [simp]: "is_const_mero_uhp (-f) = is_const_mero_uhp f"*
⟨*proof*⟩

lemma *is_const_mero_uhp_add_absorb1 [simp]:*
"is_const_mero_uhp f ⟹ is_const_mero_uhp (f + g) ⟷ is_const_mero_uhp g"
⟨*proof*⟩

lemma *is_const_mero_uhp_add_absorb2 [simp]:*
"is_const_mero_uhp g ⟹ is_const_mero_uhp (f + g) ⟷ is_const_mero_uhp f"

⟨*proof*⟩

lemma *is_const_mero_uhp_diff_absorb1 [simp]:*
"is_const_mero_uhp f ⟹ is_const_mero_uhp (f - g) ⟷ is_const_mero_uhp g"
⟨*proof*⟩

lemma *is_const_mero_uhp_diff_absorb2 [simp]:*
"is_const_mero_uhp g ⟹ is_const_mero_uhp (f - g) ⟷ is_const_mero_uhp f"
⟨*proof*⟩

lemma *is_pole_mero_uhp_power_iff [simp]:*
"is_pole (eval_mero_uhp (f ^ n)) z ⟷ Im z > 0 ∧ is_pole f z ∧ n > 0"
⟨*proof*⟩

lemma *eval_mero_uhp_power [simp]:*
assumes *z: "Im z > 0"*
shows *"eval_mero_uhp (f ^ n) z = eval_mero_uhp f z ^ n"*
⟨*proof*⟩

lemma *constant_on_cong [cong]:*
"(⋀x. x ∈ A ⟹ f x = g x) ⟹ A = B ⟹ f constant_on A ⟷ g constant_on B"
⟨*proof*⟩

lemma *constant_on_const [intro]: "(λ_. c) constant_on A"*
⟨*proof*⟩

lemma *is_const_eval_const_mero_uhp:*
assumes *"is_const_mero_uhp f" "A ⊆ {z. Im z > 0}"*
shows *"eval_mero_uhp f constant_on A"*
⟨*proof*⟩

lemma *not_constant_on_eval_mero_uhp:*
assumes *"¬is_const_mero_uhp f" "¬A sparse_in {z. Im z > 0}"*
shows *"¬eval_mero_uhp f constant_on A"*
⟨*proof*⟩

lemma *constant_on_eval_mero_uhp_iff:*
assumes *"¬A sparse_in {z. Im z > 0}" "A ⊆ {z. Im z > 0}"*
shows *"eval_mero_uhp f constant_on A ⟷ is_const_mero_uhp f"*
⟨*proof*⟩

lemma *meromorphic_on_imp_sparse_poles:*
assumes *"f meromorphic_on A"*

shows *"{z. is_pole f z} sparse_in A"*
⟨*proof*⟩

lemma *is_pole_inverse_iff:*
fixes *f :: "'a::{real_normed_div_algebra, division_ring} ⇒ 'a"*
shows *"is_pole (λz. inverse (f z)) z ⟷ filterlim f (at 0) (at z)"*
⟨*proof*⟩

lemma *filterlim_at_conv_at_0:*
fixes *c :: "'a::real_normed_vector"*
shows *"filterlim f (at c) F ⟷ filterlim (λx. f x - c) (at 0) F"*
⟨*proof*⟩

lemma *filterlim_at_conv_at_0':*
fixes *c :: "'a::real_normed_vector"*
assumes *"NO_MATCH 0 c"*
shows *"filterlim f (at c) F ⟷ filterlim (λx. f x - c) (at 0) F"*
⟨*proof*⟩

abbreviation *upper_half_plane ("ℋ")* **where** *"ℋ ≡ {z. Im z > 0}"*

lemma *eval_mero_uhp_avoid:*
assumes *"f ≠ const_mero_uhp c"*
shows *"∀≈z∈ℋ. f z ≠ c"*
⟨*proof*⟩

lemma *eval_mero_uhp_avoid_0:*
assumes *"f ≠ 0"*
shows *"eventually (λz. eval_mero_uhp f z ≠ 0) (cosparse {z. Im z > 0})"*
⟨*proof*⟩

lemma *poles_mero_uhp_const_mero_uhp [simp]: "is_const_mero_uhp f ⟹ poles_mero_uhp f = {}"*
⟨*proof*⟩

lemma *eventually_eval_mero_uhp_neq_iff:*
assumes *"Im z > 0"*
shows *"eventually (λz. eval_mero_uhp f z ≠ c) (at z) ⟷ f ≠ const_mero_uhp c"*
⟨*proof*⟩

lemma *analytic_at_mero_uhp_iff:*
assumes *"Im z > 0"*
shows *"eval_mero_uhp f analytic_on {z} ⟷ ¬is_pole (eval_mero_uhp f) z"*
⟨*proof*⟩

lemma *has_laurent_expansion_imp_subdegree_nonneg:*

assumes *"f has_laurent_expansion F" "f −0→ c"*
shows *"fls_subdegree F ≥ 0"*
⟨*proof*⟩

lemma *nicely_meromorphic_at_tendsto_imp_analytic_at:*
assumes *"f nicely_meromorphic_on {z}" "f −z→ c"*
shows *"f analytic_on {z}"*
⟨*proof*⟩

lemma *tendsto_eval_mero_uhp_iff:*
assumes *z: "Im z > 0"*
shows *"eval_mero_uhp f −z→ c ⟷ ¬is_pole (eval_mero_uhp f) z ∧ f z = c"*
⟨*proof*⟩

lemma *is_pole_inverse_mero_uhp_iff:*
assumes *z: "Im z > 0"*
shows *"is_pole (eval_mero_uhp (inverse f)) z ⟷ f ≠ 0 ∧ ¬is_pole f z ∧ eval_mero_uhp f z = 0"*
⟨*proof*⟩

lemma *poles_mero_uhp_inverse:*
assumes *"f ≠ 0"*
shows *"poles_mero_uhp (inverse f) = zeros_mero_uhp f"*
⟨*proof*⟩

lemma *zeros_mero_uhp_inverse:*
assumes *"f ≠ 0"*
shows *"zeros_mero_uhp (inverse f) = poles_mero_uhp f"*
⟨*proof*⟩

lemma *is_pole_cong':*
assumes *"eventually (λx. f x = g x) (cosparse A)" "x ∈ A" "open A"*
shows *"is_pole f x ⟷ is_pole g x"*
⟨*proof*⟩

definition *compose_modgrp_mero_uhp :: "mero_uhp ⇒ modgrp ⇒ mero_uhp"*
where
"compose_modgrp_mero_uhp f h = mero_uhp (λz. f (apply_modgrp h z))"

lemma *mero_uhp_rel_compose_modgrp [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (compose_modgrp_mero_uhp f h)) (λz. eval_mero_uhp f (apply_modgrp h z))"
⟨*proof*⟩

lemma *compose_modgrp_mero_uhp_uminus_right [simp]:*
"compose_modgrp_mero_uhp f (-h) = compose_modgrp_mero_uhp f h"
⟨*proof*⟩

lemma *eval_compose_modgrp_mero_uhp [simp]:*
"eval_mero_uhp (compose_modgrp_mero_uhp f h) z = eval_mero_uhp f (apply_modgrp h z)"
⟨*proof*⟩

lemma *compose_modgrp_mero_uhp_const [simp]:*
"compose_modgrp_mero_uhp (const_mero_uhp c) h = const_mero_uhp c"
⟨*proof*⟩

lemma *compose_modgrp_mero_uhp_1 [simp]:*
"compose_modgrp_mero_uhp f 1 = f"
⟨*proof*⟩

lemma *mero_uhp_rel_cong_compose_modgrp [mero_uhp_rel_cong]:*
assumes *"mero_uhp_rel f g"*
shows *"mero_uhp_rel (λz. f (apply_modgrp h z)) (λz. g (apply_modgrp h z))"*
⟨*proof*⟩

interpretation *compose_modgrp_mero_uhp: field_char_0_hom "λf. compose_modgrp_mero_uhp f h"*
⟨*proof*⟩

lemma *compose_modgrp_mero_uhp_mult_right:*
*"compose_modgrp_mero_uhp f (g * h) = compose_modgrp_mero_uhp (compose_modgrp_mero_uhp f g) h"*
⟨*proof*⟩

definition *automorphy_factor_mero_uhp :: "modgrp ⇒ mero_uhp"* **where**
"automorphy_factor_mero_uhp h = mero_uhp (automorphy_factor h)"

lemma *mero_uhp_rel_automorphy_factor [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (automorphy_factor_mero_uhp h)) (λz. automorphy_factor h z)"
⟨*proof*⟩

lemma *automorphy_factor_mero_uhp_uminus [simp]:*
"automorphy_factor_mero_uhp (-h) = -automorphy_factor_mero_uhp h"
⟨*proof*⟩

lemma *eval_automorphy_factor_mero_uhp [simp]:*
"Im z > 0 ⟹ eval_mero_uhp (automorphy_factor_mero_uhp h) z = automorphy_factor h z"
⟨*proof*⟩

lemma *automorphy_factor_mero_uhp_1 [simp]: "automorphy_factor_mero_uhp 1 = 1"*

⟨*proof*⟩

lemma *automorphy_factor_mero_uhp_shift [simp]: "automorphy_factor_mero_uhp (shift_modgrp n) = 1"*
⟨*proof*⟩

lemma *automorphy_factor_mero_uhp_T [simp]: "automorphy_factor_mero_uhp T_modgrp = 1"*
⟨*proof*⟩

lemma *automorphy_factor_mero_uhp_neq_0 [simp]: "automorphy_factor_mero_uhp h ≠ 0"*
⟨*proof*⟩

lemma *deriv_mero_uhp_automorphy_factor [simp]:*
"deriv_mero_uhp (automorphy_factor_mero_uhp h) = of_int (modgrp_c h)"
⟨*proof*⟩

lemma *automorphy_factor_mero_uhp_mult:*
*"automorphy_factor_mero_uhp (g * h) =*
*compose_modgrp_mero_uhp (automorphy_factor_mero_uhp g) h * automorphy_factor_mero_uhp h"*
⟨*proof*⟩

definition *slash_mero_uhp :: "int ⇒ modgrp ⇒ mero_uhp ⇒ mero_uhp"* **where**
*"slash_mero_uhp k g f = automorphy_factor_mero_uhp g powi (-k) * compose_modgrp_mero_uhp f g"*

lemma *mero_uhp_rel_slash [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (slash_mero_uhp k g f))
*(λz. automorphy_factor g z powi (-k) * eval_mero_uhp f (apply_modgrp g z))"*
⟨*proof*⟩

lemma *eval_slash_mero_uhp:*
assumes *"Im z > 0" "¬is_pole (eval_mero_uhp f) (apply_modgrp h z)"*
shows *"eval_mero_uhp (slash_mero_uhp k h f) z =*
*automorphy_factor h z powi (-k) * eval_mero_uhp f (apply_modgrp h z)"*
⟨*proof*⟩

lemma *slash_mero_uhp_eq_0_iff [simp]: "slash_mero_uhp k g f = 0 ⟷ f = 0"*
⟨*proof*⟩

lemma *slash_mero_uhp_const [simp]: "slash_mero_uhp 0 h (const_mero_uhp c) = const_mero_uhp c"*
and *slash_mero_uhp_1_right [simp]: "slash_mero_uhp 0 h 1 = 1"*

and *slash_mero_uhp_0 [simp]: "slash_mero_uhp k h 0 = 0"*
⟨*proof*⟩

lemma *slash_mero_uhp_uminus [simp]: "even k ⟹ slash_mero_uhp k (-h) f = slash_mero_uhp k h f"*
⟨*proof*⟩

lemma *slash_mero_uhp_1 [simp]: "slash_mero_uhp k 1 f = f"*
⟨*proof*⟩

lemma *slash_mero_uhp_mult:*
*"slash_mero_uhp k (g * h) f = slash_mero_uhp k h (slash_mero_uhp k g f)"* (**is** *"?lhs = ?rhs"*)
⟨*proof*⟩

interpretation *slash_mero_uhp: ab_group_add_hom "λf. slash_mero_uhp k h f"*
⟨*proof*⟩

lemma *slash_mero_uhp_mult_right:*
*"slash_mero_uhp k h f * slash_mero_uhp l h g = slash_mero_uhp (k + l) h (f * g)"*
⟨*proof*⟩

lemma *slash_mero_uhp_divide:*
"slash_mero_uhp k h f / slash_mero_uhp l h g = slash_mero_uhp (k - l) h (f / g)"
⟨*proof*⟩

lemma *slash_mero_uhp_power_int:*
*"slash_mero_uhp k h f powi n = slash_mero_uhp (n * k) h (f powi n)"*
⟨*proof*⟩

lemma *slash_mero_uhp_power:*
*"slash_mero_uhp k h f ^ n = slash_mero_uhp (n * k) h (f ^ n)"*
⟨*proof*⟩

lemma *slash_mero_uhp_inverse:*
"inverse (slash_mero_uhp k h f) = slash_mero_uhp (-k) h (inverse f)"
⟨*proof*⟩

lemma *slash_mero_uhp_cmult_left [simp]:*
*"slash_mero_uhp k h (const_mero_uhp c * f) = const_mero_uhp c * slash_mero_uhp k h f"*
⟨*proof*⟩

lemma *slash_mero_uhp_cmult_right [simp]:*
*"slash_mero_uhp k h (f * const_mero_uhp c) = slash_mero_uhp k h f * const_mero_uhp c"*

⟨*proof*⟩

lemma *slash_mero_uhp_T:*
"slash_mero_uhp k T_modgrp f = compose_modgrp_mero_uhp f T_modgrp"
⟨*proof*⟩

lemma *zorder_mero_uhp_neg_iff [simp]:*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (eval_mero_uhp f) z < 0 ⟷ is_pole f z"*
⟨*proof*⟩

lemma *zorder_mero_uhp_nonneg_iff [simp]:*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (eval_mero_uhp f) z ≥ 0 ⟷ ¬is_pole f z"*
⟨*proof*⟩

lemma *zorder_mero_uhp_pos_iff [simp]:*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (eval_mero_uhp f) z > 0 ⟷ ¬is_pole f z ∧ eval_mero_uhp f z = 0"*
⟨*proof*⟩

lemma *zorder_mero_uhp_nonpos_iff [simp]:*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (eval_mero_uhp f) z ≤ 0 ⟷ is_pole f z ∨ eval_mero_uhp f z ≠ 0"*
⟨*proof*⟩

lemma *zorder_mero_uhp_eq_0_iff [simp]:*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (eval_mero_uhp f) z = 0 ⟷ ¬is_pole f z ∧ eval_mero_uhp f z ≠ 0"*
⟨*proof*⟩

lemma *zorder_const_mero_uhp [simp]:*
assumes *"is_const_mero_uhp f" "f ≠ 0" "Im z > 0"*
shows *"zorder f z = 0"*
⟨*proof*⟩

lemma *zorder_mult_mero_uhp:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "Im z > 0"*
shows *"zorder (f * g) z = zorder f z + zorder g z"*
⟨*proof*⟩

lemma *zorder_cmult_mero_uhp [simp]:*
fixes *f :: mero_uhp*

assumes *[simp]: "c ≠ 0"*
shows *"zorder (const_mero_uhp c * f) z = zorder f z"*
⟨*proof*⟩

lemma *zorder_cmult_mero_uhp' [simp]:*
fixes *f :: mero_uhp*
assumes *[simp]: "c ≠ 0"*
shows *"zorder (f * const_mero_uhp c) z = zorder f z"*
⟨*proof*⟩

lemma *zorder_uminus_mero_uhp [simp]:*
fixes *f :: mero_uhp*
shows *"zorder (-f) z = zorder f z"*
⟨*proof*⟩

lemma *zorder_power_mero_uhp [simp]:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (f ^ n) z = int n * zorder f z"*
⟨*proof*⟩

lemma *zorder_inverse_mero_uhp [simp]:*
fixes *f :: mero_uhp*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (inverse f) z = -zorder f z"*
⟨*proof*⟩

lemma *zorder_power_int_mero_uhp [simp]:*
fixes *f :: mero_uhp*
assumes *"f ≠ 0" "Im z > 0"*
shows *"zorder (f powi n) z = n * zorder f z"*
⟨*proof*⟩

lemma *zorder_divide_mero_uhp:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "Im z > 0"*
shows *"zorder (f / g) z = zorder f z - zorder g z"*
⟨*proof*⟩

lemma *zorder_prod_mero_uhp:*
fixes *f :: "'a ⇒ mero_uhp"*
assumes *"⋀x. x ∈ A ⟹ f x ≠ 0" "Im z > 0"*
shows *"zorder (∏x∈A. f x) z = (∑x∈A. zorder (f x) z)"*
⟨*proof*⟩

lemma *zorder_mero_uhp_add_ge:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "f + g ≠ 0" "zorder f z ≥ c" "zorder g z ≥ c" "Im z > 0"*

shows *"zorder (f + g) z ≥ c"*
⟨*proof*⟩

lemma *zorder_mero_uhp_diff_ge:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "f ≠ g" "zorder f z ≥ c" "zorder g z ≥ c"*
"Im z > 0"
shows *"zorder (f - g) z ≥ c"*
⟨*proof*⟩

lemma *zorder_mero_uhp_add1:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "zorder f z < zorder g z" "Im z > 0"*
shows *"zorder (f + g) z = zorder f z"*
⟨*proof*⟩

lemma *zorder_mero_uhp_add2:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "zorder f z > zorder g z" "Im z > 0"*
shows *"zorder (f + g) z = zorder g z"*
⟨*proof*⟩

lemma *zorder_mero_uhp_diff1:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "zorder f z < zorder g z" "Im z > 0"*
shows *"zorder (f - g) z = zorder f z"*
⟨*proof*⟩

lemma *zorder_mero_uhp_diff2:*
fixes *f g :: mero_uhp*
assumes *"f ≠ 0" "g ≠ 0" "zorder f z > zorder g z" "Im z > 0"*
shows *"zorder (f - g) z = zorder g z"*
⟨*proof*⟩

lemma *filtermap_apply_modgrp_cosparse_uhp:*
"filtermap (apply_modgrp h) (cosparse {z. Im z > 0}) = cosparse {z.
Im z > 0}"
⟨*proof*⟩

lemma *eventually_cosparse_uhp_apply_modgrp_iff:*
"eventually (λz. P (apply_modgrp h z)) (cosparse {z. Im z > 0}) ⟷
eventually P (cosparse {z. Im z > 0})"
⟨*proof*⟩

lemma *deriv_mero_uhp_compose_modgrp:*
"deriv_mero_uhp (compose_modgrp_mero_uhp f h) =
compose_modgrp_mero_uhp (deriv_mero_uhp f) h / automorphy_factor_mero_uhp
h ^ 2"
⟨*proof*⟩

lemma *mero_uhp_rel_deriv_mero_uhp_compose_modgrp [mero_uhp_rel_intros]:*
"mero_uhp_rel (deriv_mero_uhp (compose_modgrp_mero_uhp f h))
(compose_modgrp_mero_uhp (deriv_mero_uhp f) h / automorphy_factor_mero_uhp h ^ 2)"
⟨*proof*⟩

lemma *eventually_not_is_pole_eval_mero_uhp:*
"eventually (λz. ¬is_pole (eval_mero_uhp f) z) (cosparse {z. Im z > 0})"
⟨*proof*⟩

lemma *is_pole_plus_analytic_mero_uhp_iff1:*
fixes *f g :: mero_uhp*
assumes *"¬is_pole g z"*
shows *"is_pole (eval_mero_uhp (f + g)) z ⟷ is_pole f z"*
⟨*proof*⟩

lemma *is_pole_plus_analytic_mero_uhp_iff2:*
fixes *f g :: mero_uhp*
assumes *"¬is_pole f z"*
shows *"is_pole (eval_mero_uhp (f + g)) z ⟷ is_pole g z"*
⟨*proof*⟩

lemma *not_is_pole_const_mero_uhp [simp]:*
assumes *"is_const_mero_uhp f"*
shows *"¬is_pole f z"*
⟨*proof*⟩

lemma *deriv_mero_uhp_const [simp]: "deriv_mero_uhp (const_mero_uhp c) = 0"*
⟨*proof*⟩

lemma *deriv_mero_uhp_0 [simp]: "deriv_mero_uhp 0 = 0"*
⟨*proof*⟩

lemma *deriv_mero_uhp_1 [simp]: "deriv_mero_uhp 1 = 0"*
⟨*proof*⟩

lemma *deriv_mero_uhp_power_int:*
*"deriv_mero_uhp (f powi n) = of_int n * deriv_mero_uhp f * f powi (n-1)"*
⟨*proof*⟩

lemma *deriv_mero_uhp_add [simp]: "deriv_mero_uhp (f + g) = deriv_mero_uhp f + deriv_mero_uhp g"*
⟨*proof*⟩

lemma *deriv_mero_uhp_mult:*
*"deriv_mero_uhp (f * g) = deriv_mero_uhp f * g + f * deriv_mero_uhp*

```
g"
```
⟨*proof*⟩

lemma *deriv_mero_uhp_minus [simp]:*
```
  "deriv_mero_uhp (-f) = -deriv_mero_uhp f"
```
⟨*proof*⟩

lemma *deriv_mero_uhp_diff [simp]:*
```
  "deriv_mero_uhp (f - g) = deriv_mero_uhp f - deriv_mero_uhp g"
```
⟨*proof*⟩

lemma *deriv_mero_uhp_power [simp]:*
```
  "deriv_mero_uhp (f ^ n) = of_nat n * deriv_mero_uhp f * f ^ (n - 1)"
```
⟨*proof*⟩

lemma *deriv_mero_uhp_inverse:*
```
  "deriv_mero_uhp (inverse f) = -deriv_mero_uhp f / f ^ 2"
```
⟨*proof*⟩

lemma *deriv_mero_uhp_divide:*
```
  "deriv_mero_uhp (f / g) = (g * deriv_mero_uhp f - f * deriv_mero_uhp
g) / g ^ 2"
```
⟨*proof*⟩

lemma *deriv_mero_uhp_sum:* `"deriv_mero_uhp (∑x∈A. f x) = (∑x∈A. deriv_mero_uhp (f x))"`
⟨*proof*⟩

lemma *deriv_mero_uhp_poly:*
```
  "deriv_mero_uhp (poly p f) =
     poly (pderiv p) f * deriv_mero_uhp f + poly (map_poly deriv_mero_uhp
p) f"
```
⟨*proof*⟩

lemma *constant_on_extend_mero_uhp_rel:*
```
  assumes "mero_uhp_rel (eval_mero_uhp f) g" "g constant_on A"
  assumes "open A" "A ≠ {}" "A ⊆ {z. Im z > 0}"
  shows   "is_const_mero_uhp f"
```
⟨*proof*⟩

definition *holo_uhp :: "mero_uhp ⇒ bool"* **where**
```
  "holo_uhp f ⟷ (∀z. ¬is_pole f z)"
```

lemma *holo_uhp_mero_uhp_rel_transfer:*
```
  assumes "mero_uhp_rel (eval_mero_uhp f) g"
  assumes "g analytic_on {z. Im z > 0}"
  shows   "holo_uhp f"
```
⟨*proof*⟩

lemma *holo_uhp_0 [intro, simp]: "holo_uhp 0"*
⟨*proof*⟩

lemma *holo_uhp_1 [intro, simp]: "holo_uhp 1"*
⟨*proof*⟩

lemma *holo_uhp_const [intro, simp]: "holo_uhp (const_mero_uhp c)"*
⟨*proof*⟩

lemma *holo_uhp_of_nat [intro, simp]: "holo_uhp (of_nat n)"*
⟨*proof*⟩

lemma *holo_uhp_of_int [intro, simp]: "holo_uhp (of_int n)"*
⟨*proof*⟩

lemma *holo_uhp_add:*
assumes *"holo_uhp f" "holo_uhp g"*
shows *"holo_uhp (f + g)"*
⟨*proof*⟩

lemma *holo_uhp_diff:*
assumes *"holo_uhp f" "holo_uhp g"*
shows *"holo_uhp (f - g)"*
⟨*proof*⟩

lemma *holo_uhp_uminus:*
assumes *"holo_uhp f"*
shows *"holo_uhp (-f)"*
⟨*proof*⟩

lemma *holo_uhp_mult:*
assumes *"holo_uhp f" "holo_uhp g"*
shows *"holo_uhp (f * g)"*
⟨*proof*⟩

lemma *holo_uhp_power:*
assumes *"holo_uhp f"*
shows *"holo_uhp (f ^ n)"*
⟨*proof*⟩

lemma *holo_uhp_inverse:*
assumes *"holo_uhp f" "⋀z. Im z > 0 ⟹ f z ≠ 0"*
shows *"holo_uhp (inverse f)"*
⟨*proof*⟩

lemma *holo_uhp_divide:*
assumes *"holo_uhp f" "holo_uhp g" "⋀z. Im z > 0 ⟹ g z ≠ 0"*
shows *"holo_uhp (f / g)"*

⟨*proof*⟩

lemma *holo_uhp_power_int:*
assumes *"holo_uhp f" "n ≥ 0 ∨ (∀z. Im z > 0 ⟶ f z ≠ 0)"*
shows *"holo_uhp (f powi n)"*
⟨*proof*⟩

lemma *holo_uhp_sum:*
"(⋀x. x ∈ A ⟹ holo_uhp (f x)) ⟹ holo_uhp (∑x∈A. f x)"
⟨*proof*⟩

lemma *holo_uhp_deriv:*
assumes *"holo_uhp f"*
shows *"holo_uhp (deriv_mero_uhp f)"*
⟨*proof*⟩

lemma *holo_uhp_automorphy_factor: "holo_uhp (automorphy_factor_mero_uhp h)"*
⟨*proof*⟩

lemma *holo_uhp_compose_mero_uhp_modgrp:*
assumes *"holo_uhp f"*
shows *"holo_uhp (compose_modgrp_mero_uhp f h)"*
⟨*proof*⟩

lemma *holo_uhp_slash:*
assumes *"holo_uhp f"*
shows *"holo_uhp (slash_mero_uhp weight h f)"*
⟨*proof*⟩

end

2 q-expansions of periodic functions

theory *Fourier_Expansion_Mero_UHP*
imports
"HOL-Complex_Analysis.Complex_Analysis"
"HOL-Real_Asymp.Real_Asymp"
Meromorphic_Upper_Half_Plane
Elliptic_Functions.Z_Plane_Q_Disc
Elliptic_Functions.FPS_Homomorphism
Modular_Forms_Library
begin

2.1 The zero order at the cusp

definition *eval_mero_uhp_at_ii_inf :: "mero_uhp ⇒ complex"* **where**
"eval_mero_uhp_at_ii_inf f = (if ∃L. (f ⟶⟶ L) at_i∞ then Lim at_i∞ f else 0)"

lemma *eval_mero_uhp_at_ii_inf_eqI:*
"(eval_mero_uhp f ⟶ c) at_i∞ ⟹ eval_mero_uhp_at_ii_inf f = c"
⟨*proof*⟩

definition *zorder_at_ii_inf :: "nat ⇒ mero_uhp ⇒ int"* **where**
"zorder_at_ii_inf period f =
(if period = 0 ∨ f = 0 then 0 else (THE n. eval_mero_uhp f ∈ Θ[at_i∞](λz. to_q period z powi n)))"

definition *is_pole_ii_inf* **where** *"is_pole_ii_inf f ⟷ filterlim (eval_mero_uhp f) at_infinity at_i∞"*

lemma *zorder_at_ii_inf_0 [simp]: "zorder_at_ii_inf period 0 = 0"*
⟨*proof*⟩

lemma *zorder_at_ii_inf_unique:*
assumes *"k > 0"*
shows *"∃≤1 n. f ∈ Θ[at_i∞](λx. to_q k x powi n)"* (**is** *"Uniq ?P"*)
⟨*proof*⟩

lemma *zorder_at_ii_inf_eqI:*
assumes *"eval_mero_uhp f ∈ Θ[at_i∞](λx. to_q k x powi n)" "k > 0"*
shows *"zorder_at_ii_inf k f = n"*
⟨*proof*⟩

abbreviation *"zorder_mero_uhp f ≡ zorder (eval_mero_uhp f)"*

2.2 Expansion in terms of q

Let *f(τ)* be a meromorphic function on the halfplane $\{\tau \mid \mathrm{Im}(\tau) > c\}$ with period n. The variable change $q = 2i\pi\tau/n$ gives us a function $f(q)$ that is meromorphic on a punctured disc of radius $e^{-2\pi c}$ around 0. The point $q = 0$ corresponds to $\tau = i\infty$. Thus, this function $f(q)$ can be viewed as the expansion of $f(\tau)$ at the point $\tau = i\infty$.

definition *fourier_expansion :: "nat ⇒ mero_uhp ⇒ complex ⇒ complex"*
where
"fourier_expansion period f =
(λz. if z = 0 then remove_sings (eval_mero_uhp f ∘ of_q period) 0

else eval_mero_uhp f (of_q period z))"

lemma *fourier_expansion_0 [simp]: "fourier_expansion period 0 z = 0"*
⟨*proof*⟩

definition *laurent_expansion_at_ii_inf :: "nat ⇒ mero_uhp ⇒ complex fls"*
("laurent'_expansion'_at'_i∞") **where**
"laurent_expansion_at_ii_inf period f = laurent_expansion (fourier_expansion

```
period f) 0"

definition fps_expansion_at_ii_inf :: "nat ⇒ mero_uhp ⇒ complex fps"
("fps'_expansion'_at'_i∞") where
  "fps_expansion_at_ii_inf period f = fps_expansion (fourier_expansion
period f) 0"

definition meromorphic_at_infinity :: "modgrp set ⇒ mero_uhp ⇒ bool"
where
  "meromorphic_at_infinity G f ⟷
     fourier_expansion (cusp_width∞ G) f meromorphic_on {0}"

definition holomorphic_at_infinity :: "mero_uhp ⇒ bool" where
  "holomorphic_at_infinity f ⟷ (∃c. (f ⟶ c) at_i∞)"

lemma holomorphic_at_infinity_const [simp, intro]: "holomorphic_at_infinity
(const_mero_uhp c)"
⟨proof⟩

lemma holomorphic_at_infinity_0 [simp, intro]: "holomorphic_at_infinity
0"
  ⟨proof⟩

lemma holomorphic_at_infinity_1 [simp, intro]: "holomorphic_at_infinity
1"
  ⟨proof⟩



locale fourier_expansion_context =
  fixes period :: nat
  assumes period_pos: "period > 0"
begin

abbreviation (input) fourier_expansion' (<(<notation=<postfix q>>_q)>
[1000] 1000) where
  "fourier_expansion' ≡ fourier_expansion period"

end


locale fourier_expansion_locale =
  fixes period :: nat and f :: mero_uhp
  assumes period_pos: "period > 0"
  assumes invariant_compose_shift_period: "compose_modgrp_mero_uhp f
(shift_modgrp period) = f"
begin

sublocale periodic_fun_simple "eval_mero_uhp f" "of_nat period"
```

⟨*proof*⟩

lemma *fourier_expansion_locale_mono:*
  assumes *"period dvd period'" "period' > 0"*
  shows *"fourier_expansion_locale period' f"*
⟨*proof*⟩

lemma *invariant_compose_shift [simp]:*
  assumes *"period dvd n"*
  shows *"compose_modgrp_mero_uhp f (shift_modgrp n) = f"*
⟨*proof*⟩

interpretation *ctxt: fourier_expansion_context period*
  ⟨*proof*⟩

lemma *fourier_nz_eq:*
  assumes *q: "q ≠ 0"*
  shows *"f^q q = f (of_q period q)"*
  ⟨*proof*⟩

lemma *fourier_0_aux:*
  assumes *"(f ⟶ y) at_i∞"*
  shows *"f^q 0 = y"*
⟨*proof*⟩

lemma *isCont_0_aux:*
  assumes *"(f ⟶ y) at_i∞"*
  shows *"isCont f^q 0"*
⟨*proof*⟩

lemma *fourier_to_q [simp]: "f^q (to_q period τ) = f τ"*
⟨*proof*⟩

lemma *fourier_expansion_mult_period:*
  assumes *k: "k > 0"*
  shows *"fourier_expansion (period * k) f q = fourier_expansion period f (q ^ k)"*
⟨*proof*⟩

lemma *has_laurent_expansion_fourier_mult_period:*
  assumes *"f^q has_laurent_expansion F" "k > 0"*
  shows *"fourier_expansion (period * k) f has_laurent_expansion fls_compose_fps F (fps_X ^ k)"*
⟨*proof*⟩

lemma *has_fps_expansion_fourier_mult_period:*
  assumes *"f^q has_fps_expansion F" "k > 0"*
  shows *"fourier_expansion (period * k) f has_fps_expansion fps_compose F (fps_X ^ k)"*

⟨*proof*⟩

lemma *filtermap_fourier_expansion_at_0: "filtermap* f^q *(at 0) = filtermap f at_i∞"*
⟨*proof*⟩

lemma *fourier_tendsto_0_iff: "*f^q *−0→ y ⟷ (f ⟶ y) at_i∞"*
⟨*proof*⟩

lemma *fourier_is_pole_0_iff:*
"is_pole f^q *0 ⟷ filterlim f at_infinity at_i∞"*
⟨*proof*⟩

lemma *fourier_is_pole_to_q_iff: "is_pole* f^q *(to_q period z) ⟷ is_pole f z"*
⟨*proof*⟩

lemma *has_field_derivative_fourier:*
assumes *"¬is_pole (eval_mero_uhp f) z" "Im z > 0"*
assumes *q: "q = to_q period z"*
defines *"f' ≡ eval_mero_uhp (deriv_mero_uhp f) z"*
shows *"(*f^q *has_field_derivative*
*f' * period / (of_real (2 * pi) * i * q)) (at q within A)"*
⟨*proof*⟩

definition *fourier_poles :: "complex set"*
where *"fourier_poles = to_q period ‘ {z. 0 < Im z ∧ is_pole (eval_mero_uhp f) z}"*

lemma *zero_not_in_fourier_poles [simp]: "0 ∉ fourier_poles"*
⟨*proof*⟩

lemma *is_pole_of_q_iff:*
assumes *"q ∈ ball 0 1 - {0}"*
shows *"is_pole f (of_q period q) ⟷ q ∈ fourier_poles"*
⟨*proof*⟩

lemma *to_q_in_fourier_poles_iff [simp]:*
"to_q period z ∈ fourier_poles ⟷ is_pole f z"
⟨*proof*⟩

lemma *not_islimpt_fourier_poles:*
assumes *"z ∈ ball 0 1 - {0}"*
shows *"¬z islimpt fourier_poles"*
⟨*proof*⟩

lemma *open_Diff_fourier_poles':*
assumes *"fourier_poles' ⊆ fourier_poles"*

shows *"open (ball 0 1 - {0} - fourier_poles')"*
⟨*proof*⟩

lemma *open_Diff_fourier_poles: "open (ball 0 1 - {0} - fourier_poles)"*
⟨*proof*⟩

lemma *analytic_fourier:*
assumes *"A ⊆ ball 0 1 - fourier_poles - {0}"*
shows *"f^q analytic_on A"*
⟨*proof*⟩

lemma *fourier_poles_altdef:*
"fourier_poles = {q∈ball 0 1-{0}. is_pole f^q q}"
⟨*proof*⟩

lemma *fourier_meromorphic_weak:*
assumes *"A ⊆ ball 0 1 - {0}"*
shows *"f^q meromorphic_on A"*
⟨*proof*⟩

lemma *tendsto_fourier_to_q:*
assumes *"f $-z\to$ c" "q = to_q period z"*
shows *"f^q $-q\to$ c"*
⟨*proof*⟩

lemma *deriv_fourier:*
assumes *"Im z > 0" "¬is_pole f z" "q = to_q period z"*
shows *"deriv f^q q = eval_mero_uhp (deriv_mero_uhp f) z * of_nat period / (of_real (2 * pi) * i * q)"*
⟨*proof*⟩

lemma *eval_fourier_outside:*
assumes *"norm q > 1"*
shows *"f^q q = 0"*
⟨*proof*⟩

lemma *not_pole_eval_fourier_outside: "norm q ≥ 1 ⟹ ¬is_pole f^q q"*
⟨*proof*⟩

lemma *fourier_expansion_locale_deriv:*
"fourier_expansion_locale period (deriv_mero_uhp f)"
⟨*proof*⟩

lemma *deriv_conv_deriv_fourier_expansion:*
assumes *z: "Im z > 0" "¬is_pole f z"*
defines *"q ≡ to_q period z"*
shows *"deriv f z = 2 * i * pi / period * q * deriv f^q q"*
⟨*proof*⟩

end

2.3 Meromorphicity at infinity

definition *has_laurent_expansion_at_ii_inf :: "mero_uhp ⇒ nat ⇒ complex fls ⇒ bool"*
(‹(‹notation=‹infix has_laurent_expansion_at_ii_inf››(_) has'_laurent'_expansion'_at'_i∞[(_)]›› [60, 0, 60] 60) **where**
"f has_laurent_expansion_at_i∞[period] F ⟷
fourier_expansion_locale period f ∧ fourier_expansion period f has_laurent_expansion F"

definition *has_fps_expansion_at_ii_inf :: "mero_uhp ⇒ nat ⇒ complex fps ⇒ bool"*
(‹(‹notation=‹infix has_fps_expansion_at_ii_inf››(_) has'_fps'_expansion'_at'_i∞[(_)]›› [60, 0, 60] 60) **where**
"f has_fps_expansion_at_i∞[period] F ⟷
fourier_expansion_locale period f ∧ fourier_expansion period f has_fps_expansion F"

abbreviation *has_laurent_expansion_at_ii_inf_1 :: "mero_uhp ⇒ complex fls ⇒ bool"*
(‹(‹notation=‹infix has_laurent_expansion_at_ii_inf_1››(_) has'_laurent'_expansion'_at'_i∞[(_)]›› [60, 60] 60) **where**
"f has_laurent_expansion_at_i∞ F ≡ f has_laurent_expansion_at_i∞[Suc 0] F"

abbreviation *has_fps_expansion_at_ii_inf_1 :: "mero_uhp ⇒ complex fps ⇒ bool"*
(‹(‹notation=‹infix has_fps_expansion_at_ii_inf_1››(_) has'_fps'_expansion'_at'_i∞[(_)]›› [60, 60] 60) **where**
"f has_fps_expansion_at_i∞ F ≡ f has_fps_expansion_at_i∞[Suc 0] F"

locale *fourier_expansion_meromorphic = fourier_expansion_locale +*
assumes *fourier_meromorphic_at_0: "fourier_expansion period f meromorphic_on {0}"*
begin

interpretation *ctxt: fourier_expansion_context period*
⟨*proof*⟩

lemma *fourier_meromorphic [meromorphic_intros]:*
assumes *"A ⊆ ball 0 1"*
shows *"f^q meromorphic_on A"*
⟨*proof*⟩

lemma *fourier_meromorphic' [meromorphic_intros]:*
assumes *"f analytic_on A" "⋀z. z ∈ A ⟹ norm (f z) < 1"*

shows *"(λz. f^q (f z)) meromorphic_on A"*
⟨*proof*⟩

lemma *fourier_nicely_meromorphic: "f^q nicely_meromorphic_on ball 0 1"*
⟨*proof*⟩

lemma *frequently_fourier_eq0_imp_const:*
assumes *"frequently (λq. f^q q = c) (at q)" "norm q < 1"*
shows *"f = const_mero_uhp c"*
⟨*proof*⟩

lemma *laurent_expansion_fourier_eq_0_iff:*
assumes *"(λw. f^q (z + w)) has_laurent_expansion F" "norm z < 1"*
shows *"F = 0 ⟷ f = 0"*
⟨*proof*⟩

lemma *laurent_expansion_fourier_eq_0_iff0:*
assumes *"f^q has_laurent_expansion F"*
shows *"F = 0 ⟷ f = 0"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_conv_fourier:*
"f has_laurent_expansion_at_i∞[period] F ⟷ f^q has_laurent_expansion F"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_conv_fourier:*
"f has_fps_expansion_at_i∞[period] F ⟷ f^q has_fps_expansion F"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf:*
"f has_laurent_expansion_at_i∞[period] laurent_expansion_at_i∞ period f"
⟨*proof*⟩

lemma *zorder_at_ii_inf_conv_fourier:*
assumes *"f ≠ 0"*
shows *"zorder_at_ii_inf period f = zorder f^q 0"*
⟨*proof*⟩

lemma *eventually_neq_at_ii_inf:*
assumes *"f ≠ const_mero_uhp c"*
shows *"eventually (λz. f z ≠ c) at_i∞"*
⟨*proof*⟩

lemma *eventually_neq_fourier:*
assumes *"f ≠ const_mero_uhp c" "norm q < 1"*
shows *"eventually (λq. f^q q ≠ c) (at q)"*
⟨*proof*⟩

lemma *eventually_no_isolated_zero: "eventually (λz. ¬isolated_zero f z) at_i∞"*
⟨*proof*⟩

lemma *eventually_no_poles: "eventually (λz. ¬is_pole f z) at_i∞"*
⟨*proof*⟩

lemma *eval_at_ii_inf_conv_fourier: "eval_mero_uhp_at_ii_inf f = f^q 0"*
⟨*proof*⟩

lemma *is_pole_ii_inf_conv_fourier: "is_pole_ii_inf f ⟷ is_pole f^q 0"*
⟨*proof*⟩

lemma *analytic_fourier' [analytic_intros]:*
assumes *"g analytic_on A"*
assumes *"⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ ¬is_pole f^q (g z)"*
shows *"(λz. f^q (g z)) analytic_on A"*
⟨*proof*⟩

lemma *holomorphic_fourier' [holomorphic_intros]:*
assumes *"g holomorphic_on A"*
assumes *"⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ ¬is_pole f^q (g z)"*
shows *"(λz. f^q (g z)) holomorphic_on A"*
⟨*proof*⟩

lemma *continuous_on_fourier [continuous_intros]:*
assumes *"continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ ¬is_pole f^q (g z)"*
shows *"continuous_on A (λz. f^q (g z))"*
⟨*proof*⟩

lemma *continuous_fourier [continuous_intros]:*
assumes *"continuous (at z within A) g"* **assumes** *"norm (g z) < 1" "¬is_pole f^q (g z)"*
shows *"continuous (at z within A) (λz. f^q (g z))"*
⟨*proof*⟩

lemma *tendsto_fourier [tendsto_intros]:*
assumes *"(g ⟶ q) F"* **assumes** *"norm q < 1" "¬is_pole f^q q"*
shows *"((λz. f^q (g z)) ⟶ f^q q) F"*
⟨*proof*⟩

lemma *zorder_fourier_neg_iff [simp]:*
assumes *"f ≠ 0" "norm q < 1"*
shows *"zorder f^q q < 0 ⟷ is_pole f^q q"*
⟨*proof*⟩

lemma *zorder_fourier_nonneg_iff [simp]:*
assumes *"f ≠ 0" "norm q < 1"*
shows *"zorder f q ≥ 0 ⟷ ¬is_pole f q"*
⟨*proof*⟩

lemma *zorder_fourier_pos_iff [simp]:*
assumes *"f ≠ 0" "norm q < 1"*
shows *"zorder f q > 0 ⟷ ¬is_pole f q ∧ f q = 0"*
⟨*proof*⟩

lemma *zorder_fourier_nonpos_iff [simp]:*
assumes *"f ≠ 0" "norm q < 1"*
shows *"zorder f q ≤ 0 ⟷ is_pole f q ∨ f q ≠ 0"*
⟨*proof*⟩

lemma *zorder_fourier_eq_0_iff [simp]:*
assumes *"f ≠ 0" "norm q < 1"*
shows *"zorder f q = 0 ⟷ ¬is_pole f q ∧ f q ≠ 0"*
⟨*proof*⟩

lemma *laurent_expansion_eq_0_iff:*
"laurent_expansion_at_i∞ period f = 0 ⟷ f = 0"
⟨*proof*⟩

lemma *zorder_at_ii_inf_conv_subdegree:*
assumes *"f ≠ 0"*
shows *"zorder_at_ii_inf period f = fls_subdegree (laurent_expansion_at_i∞ period f)"*
⟨*proof*⟩

The following are some alternative, equivalent ways of showing that a function is holomorphic at infinity.

lemma *holomorphic_at_infinity_via_fourier_isCont:*
assumes *"isCont (fourier_expansion period f) 0"*
shows *"holomorphic_at_infinity f"*
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_not_is_pole_ii_inf:*
assumes *"¬is_pole_ii_inf f"*
shows *"holomorphic_at_infinity f"*
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_zorder:*
assumes *"zorder_at_ii_inf period f ≥ 0"*
shows *"holomorphic_at_infinity f"*
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_laurent_expansion:*
assumes *"f has_laurent_expansion_at_i∞[period] F" "fls_subdegree F*

```
≥ 0"
  shows   "holomorphic_at_infinity f"
```
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_fps_expansion:*
```
  assumes "f has_fps_expansion_at_i∞[period] F"
  shows   "holomorphic_at_infinity f"
```
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_bounded_sequence:*
```
  assumes "frequently (λz. norm (f z) ≤ c) at_i∞"
  shows   "holomorphic_at_infinity f"
```
⟨*proof*⟩

lemma *has_laurent_expansion_deriv:*
```
  assumes "f^q has_laurent_expansion F"
  defines "f' ≡ fourier_expansion period (deriv_mero_uhp f)"
  defines "c ≡ 2 * i * pi / period"
  shows   "f' has_laurent_expansion fls_const c * fls_X * fls_deriv F"
```
⟨*proof*⟩

lemma *fourier_expansion_meromorphic_deriv:*
```
  "fourier_expansion_meromorphic period (deriv_mero_uhp f)"
```
⟨*proof*⟩

end

locale *fourier_expansion_meromorphic_explicit = fourier_expansion_locale*
```
+
  fixes fls_fourier :: "complex fls"
  assumes has_laurent_expansion_at_ii_inf_explicit:
    "f has_laurent_expansion_at_i∞[period] fls_fourier"
```
begin

sublocale *fourier_expansion_meromorphic*
⟨*proof*⟩

lemma *fourier_expansion_meromorphic_explicit_mono:*
```
  assumes "period dvd period'" "period' > 0"
  defines "k ≡ period' div period"
  shows   "fourier_expansion_meromorphic_explicit period' f (fls_compose_fps
fls_fourier (fps_X ^ k))"
```
⟨*proof*⟩

lemma *laurent_expansion_eq [simp]: "laurent_expansion_at_i∞ period f*
```
= fls_fourier"
```
⟨*proof*⟩

lemma *fourier_expansion_meromorphic_explicit_deriv:*
defines *"F ≡ fls_const (2*i*pi/period) * fls_X * fls_deriv fls_fourier"*
shows *"fourier_expansion_meromorphic_explicit period (deriv_mero_uhp f) F"*
⟨*proof*⟩

end

lemma (**in** *fourier_expansion_meromorphic*) *fourier_expansion_meromorphic_mono:*
assumes *"period' > 0" "period dvd period'"*
shows *"fourier_expansion_meromorphic period' f"*
⟨*proof*⟩

2.4 Holomorphicity at infinity

locale *fourier_expansion_holomorphic = fourier_expansion_locale +*
assumes *holo_uhp: "holo_uhp f"*
assumes *holomorphic_at_infinity_explicit: "holomorphic_at_infinity f"*
begin

interpretation *ctxt: fourier_expansion_context period*
⟨*proof*⟩

lemma *fourier_analytic_at_0: "f^q analytic_on {0}"*
⟨*proof*⟩

sublocale *fourier_expansion_meromorphic*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf:*
"f has_fps_expansion_at_i∞[period] fps_expansion_at_i∞ period f"
⟨*proof*⟩

lemma *no_poles [simp]: "¬is_pole (eval_mero_uhp f) z"*
⟨*proof*⟩

lemma *tendsto_at_ii_inf: "(f ⟶ eval_mero_uhp_at_ii_inf f) at_i∞"*
⟨*proof*⟩

lemma *no_poles_fourier' [simp]: "fourier_poles = {}"*
⟨*proof*⟩

lemma *no_poles_fourier [simp]: "¬is_pole (fourier_expansion period f) q"*
⟨*proof*⟩

lemma *not_is_pole_ii_inf [simp]: "¬is_pole_ii_inf f"*
⟨*proof*⟩

lemma *analytic [analytic_intros]:*
"g analytic_on A ⟹ (⋀z. z ∈ A ⟹ Im z > 0) ⟹ eval_mero_uhp f analytic_on A"
⟨*proof*⟩

lemmas *[analytic_intros del] = analytic_fourier'*
lemmas *[holomorphic_intros del] = holomorphic_fourier'*

lemma *fourier_analytic_full [analytic_intros]:*
assumes *"g analytic_on A" "⋀z. z ∈ A ⟹ norm (g z) < 1"*
shows *"(λz. fourier_expansion period f (g z)) analytic_on A"*
⟨*proof*⟩

lemma *fourier_holomorphic_full [holomorphic_intros]:*
assumes *"g holomorphic_on A" "⋀z. z ∈ A ⟹ norm (g z) < 1"*
shows *"(λz. fourier_expansion period f (g z)) holomorphic_on A"*
⟨*proof*⟩

lemma *zorder_at_ii_inf_ge_0 [simp, intro]: "zorder_at_ii_inf period f ≥ 0"*
⟨*proof*⟩

end

locale *fourier_unop_meromorphic = fourier_expansion_meromorphic +*
fixes *g :: "mero_uhp ⇒ mero_uhp"* **and** *g' :: "complex ⇒ complex"* **and** *g'' :: "complex fls ⇒ complex fls"*
assumes *mero_uhp_rel_map [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (g f)) (λz. g' (eval_mero_uhp f z))"
assumes *compose_modgrp_mero_uhp_map_distrib [simp]:*
"compose_modgrp_mero_uhp (g f) h = g (compose_modgrp_mero_uhp f h)"
assumes *map_laurent_expansion:*
"⋀f F. f has_laurent_expansion F ⟹ (λz. g' (f z)) has_laurent_expansion g'' F"
begin

interpretation *ctxt: fourier_expansion_context period*
⟨*proof*⟩

sublocale *map: fourier_expansion_locale period "g f"*
⟨*proof*⟩

lemma *map_fourier_eq_aux:*
assumes *q: "q ∈ ball 0 1 - {0}" "¬is_pole f^q q" "(λq. g' (f^q q)) analytic_on {q}"*
shows *"fourier_expansion period (g f) q = g' (fourier_expansion period f q)"*

⟨*proof*⟩

lemma *meromorphic_map [meromorphic_intros]:*
  assumes *"A ⊆ ball 0 1"*
  shows *"(λw. g' (f^q w)) meromorphic_on A"*
  ⟨*proof*⟩

lemma *eventually_map_fourier_eq:*
  "eventually (λq. fourier_expansion period (g f) q = g' (f^q q)) (cosparse (ball 0 1))"
  ⟨*proof*⟩

sublocale *fourier_expansion_meromorphic period "g f"*
⟨*proof*⟩

lemma *map_fourier_eq:*
  assumes *q: "q ∈ ball 0 1" "¬is_pole f^q q" "(λq. g' (f^q q)) analytic_on {q}"*
  shows *"fourier_expansion period (g f) q = g' (f^q q)"*
⟨*proof*⟩

lemma *map_has_laurent_expansion_at_ii_inf:*
  assumes *"f^q has_laurent_expansion F"*
  shows *"(fourier_expansion period (g f)) has_laurent_expansion g'' F"*
⟨*proof*⟩

end

locale *fourier_expansion_holomorphic_explicit = fourier_expansion_locale +*
  fixes *fps_fourier :: "complex fps"*
  assumes *holo_uhp': "holo_uhp f"*
  assumes *has_laurent_expansion_at_ii_inf_fps_explicit:*
    "f has_laurent_expansion_at_ii∞[period] fps_to_fls fps_fourier"
begin

sublocale *fourier_expansion_meromorphic_explicit period f "fps_to_fls fps_fourier"*
  ⟨*proof*⟩

sublocale *fourier_expansion_holomorphic*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_explicit: "f has_fps_expansion_at_ii∞[period] fps_fourier"*
⟨*proof*⟩

lemma *fps_expansion_eq [simp]: "fps_expansion_at_i∞ period f = fps_fourier"*
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_eq: "eval_mero_uhp_at_ii_inf f = fps_nth fps_fourier 0"*
⟨*proof*⟩

end

2.5 Closure properties

locale *fourier_binop_meromorphic = fourier_expansion_context period +*
f: fourier_expansion_meromorphic period f + g: fourier_expansion_meromorphic period g
for *period f g +*
fixes *h :: "mero_uhp ⇒ mero_uhp ⇒ mero_uhp"* **and** *h' :: "complex ⇒ complex ⇒ complex"*
and *h'' :: "complex fls ⇒ complex fls ⇒ complex fls"*
assumes *mero_uhp_rel_map [mero_uhp_rel_intros]:*
"mero_uhp_rel (eval_mero_uhp (h f g)) (λz. h' (eval_mero_uhp f z) (eval_mero_uhp g z))"
assumes *compose_modgrp_mero_uhp_map_distrib [simp]:*
"compose_modgrp_mero_uhp (h f g) j = h (compose_modgrp_mero_uhp f j) (compose_modgrp_mero_uhp g j)"
assumes *map_laurent_expansion:*
"⋀f g F G. f has_laurent_expansion F ⟹ g has_laurent_expansion G ⟹
(λz. h' (f z) (g z)) has_laurent_expansion h'' F G"
begin

sublocale *map: fourier_expansion_locale period "h f g"*
⟨*proof*⟩

lemma *map_fourier_eq_aux:*
assumes *q: "q ∈ ball 0 1 - {0}"* *"¬is_pole f^q q"* *"¬is_pole g^q q"*
assumes *"(λq. h' (f^q q) (g^q q)) analytic_on {q}"*
shows *"(fourier_expansion period (h f g)) q = h' (f^q q) (g^q q)"*
⟨*proof*⟩

lemma *meromorphic_map [meromorphic_intros]:*
assumes *"A ⊆ ball 0 1"*
shows *"(λw. h' (f^q w) (g^q w)) meromorphic_on A"*
⟨*proof*⟩

lemma *eventually_map_fourier_eq:*
"eventually (λq. fourier_expansion period (h f g) q = h' (f^q q) (g^q q)) (cosparse (ball 0 1))"
⟨*proof*⟩

sublocale *fourier_expansion_meromorphic period "h f g"*
⟨*proof*⟩

lemma *map_fourier_eq:*
assumes *q: "q ∈ ball 0 1" "¬is_pole f^q q" "¬is_pole g^q q"*
assumes *"(λq. h' (f^q q) (g^q q)) analytic_on {q}"*
shows *"fourier_expansion period (h f g) q = h' (f^q q) (g^q q)"*
⟨*proof*⟩

lemma *map_has_laurent_expansion_at_ii_inf:*
assumes *"f^q has_laurent_expansion F" "g^q has_laurent_expansion G"*
shows *"fourier_expansion period (h f g) has_laurent_expansion h'' F G"*
⟨*proof*⟩

end

context *fourier_expansion_context*
begin

sublocale *const: fourier_expansion_locale period "const_mero_uhp c"*
⟨*proof*⟩

lemma *fourier_const [simp]:*
assumes *"norm q < 1"*
shows *"fourier_expansion period (const_mero_uhp c) q = c"*
⟨*proof*⟩

lemma *not_is_pole_const_fourier [simp]: "¬is_pole (fourier_expansion period (const_mero_uhp c)) q"*
⟨*proof*⟩

sublocale *const: fourier_expansion_meromorphic period "const_mero_uhp c"*
⟨*proof*⟩

lemma *zorder_fourier_0_const [simp]:*
assumes *"c ≠ 0"*
shows *"zorder (fourier_expansion period (const_mero_uhp c)) 0 = 0"*
⟨*proof*⟩

lemma *const_fourier_has_fps_expansion [fps_expansion_intros]:*
"fourier_expansion period (const_mero_uhp c) has_fps_expansion fps_const c"
⟨*proof*⟩

lemma *const_fourier_has_laurent_expansion [fps_expansion_intros]:*
"fourier_expansion period (const_mero_uhp c) has_laurent_expansion fls_const c"
⟨*proof*⟩

end

lemma *zorder_at_ii_inf_const [simp]: "zorder_at_ii_inf n (const_mero_uhp c) = 0"*
⟨*proof*⟩

context *fourier_expansion_locale*
begin

lemma *fourier_expansion_inverse: "fourier_expansion_locale period (inverse f)"*
and *fourier_expansion_uminus: "fourier_expansion_locale period (-f)"*
and *fourier_expansion_power: "fourier_expansion_locale period (f ^ n)"*
and *fourier_expansion_power_int: "fourier_expansion_locale period (f ^ n)"*
⟨*proof*⟩

end

locale *fourier_expansion_pair = fourier_expansion_context period +*
f: fourier_expansion_locale period f + g: fourier_expansion_locale period g
for *period f g*
begin

lemma *fourier_expansion_add: "fourier_expansion_locale period (f + g)"*
and *fourier_expansion_diff: "fourier_expansion_locale period (f - g)"*
and *fourier_expansion_mult: "fourier_expansion_locale period (f * g)"*
and *fourier_expansion_divide: "fourier_expansion_locale period (f / g)"*
⟨*proof*⟩

end

context *fourier_expansion_meromorphic*
begin

interpretation *minus: fourier_unop_meromorphic period f "λx. -x" "λx. -x" "λx. -x"*

⟨*proof*⟩
lemma *fourier_expansion_meromorphic_minus: "fourier_expansion_meromorphic period (-f)"* ⟨*proof*⟩
lemmas *fourier_minus_eventually_eq = minus.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_minus = minus.map_has_laurent_expansion_at_ii_inf*

interpretation *ctxt: fourier_expansion_context period*
⟨*proof*⟩

lemma *fourier_minus_eq:*
assumes *"q ∈ ball 0 1" "¬is_pole f^q q"*
shows *"(-f)^q q = -f^q q"*
⟨*proof*⟩

lemma *zorder_fourier_minus_eq:*
assumes *"q ∈ ball 0 1 - {0}"*
shows *"zorder (-f)^q q = zorder f^q q"*
⟨*proof*⟩

interpretation *cmult_left: fourier_unop_meromorphic period f "λx. const_mero_uhp c * x" "λx. c * x" "λx. fls_const c * x"*
⟨*proof*⟩
lemma *fourier_expansion_meromorphic_cmult_left: "fourier_expansion_meromorphic period (const_mero_uhp c * f)"* ⟨*proof*⟩
lemmas *fourier_cmult_left_eventually_eq = cmult_left.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_cmult_left = cmult_left.map_has_laurent_expansion_a*

lemma *fourier_cmult_left_eq:*
assumes *"q ∈ ball 0 1" "¬is_pole f^q q"*
shows *"(const_mero_uhp c * f)^q q = c * f^q q"*
⟨*proof*⟩

lemma *zorder_fourier_cmult_left_eq:*
assumes *"q ∈ ball 0 1 - {0}" "c ≠ 0"*
shows *"zorder (const_mero_uhp c * f)^q q = zorder f^q q"*
⟨*proof*⟩

interpretation *cmult_right: fourier_unop_meromorphic period f "λx. x * const_mero_uhp c" "λx. x * c" "λx. x * fls_const c"*
⟨*proof*⟩
lemma *fourier_expansion_meromorphic_cmult_right: "fourier_expansion_meromorphic period (const_mero_uhp c * f)"* ⟨*proof*⟩
lemmas *fourier_cmult_right_eventually_eq = cmult_right.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_cmult_right = cmult_right.map_has_laurent_expansion*

lemma *fourier_cmult_right_eq:*
assumes *"q ∈ ball 0 1" "¬is_pole f^q q"*
shows *"(f * const_mero_uhp c)^q q = f^q q * c"*
⟨*proof*⟩

lemma *zorder_fourier_cmult_right_eq:*
assumes *"q ∈ ball 0 1 - {0}" "c ≠ 0"*
shows *"zorder (f * const_mero_uhp c)^q q = zorder f^q q"*
⟨*proof*⟩

interpretation *inverse: fourier_unop_meromorphic period f inverse inverse inverse*
⟨*proof*⟩
lemma *fourier_expansion_meromorphic_inverse: "fourier_expansion_meromorphic period (inverse f)"* ⟨*proof*⟩
lemmas *fourier_inverse = inverse.map_fourier_eq*
lemmas *fourier_inverse_eventually_eq = inverse.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_inverse = inverse.map_has_laurent_expansion_at_ii_i*

lemma *fourier_inverse_eq:*
assumes *"q ∈ ball 0 1" "¬is_pole f^q q" "f^q q ≠ 0"*
shows *"(inverse f)^q q = inverse (f^q q)"*
⟨*proof*⟩

lemma *zorder_fourier_inverse_eq:*
assumes *"q ∈ ball 0 1" "f ≠ 0"*
shows *"zorder (inverse f)^q q = -zorder f^q q"*
⟨*proof*⟩

interpretation *power: fourier_unop_meromorphic period f "λx. x ^ n" "λx. x ^ n" "λx. x ^ n"*
⟨*proof*⟩
lemma *fourier_expansion_meromorphic_power: "fourier_expansion_meromorphic period (f ^ n)"* ⟨*proof*⟩
lemmas *fourier_power_eventually_eq = power.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_power = power.map_has_laurent_expansion_at_ii_inf*

lemma *fourier_power_eq:*
assumes *"q ∈ ball 0 1" "¬is_pole f^q q"*
shows *"(f ^ n)^q q = f^q q ^ n"*
⟨*proof*⟩

interpretation *power_int: fourier_unop_meromorphic period f "λx. x powi n" "λx. x powi n" "λx. x powi n"*
⟨*proof*⟩
lemma *fourier_expansion_meromorphic_power_int: "fourier_expansion_meromorphic*

period (f powi n)" ⟨*proof*⟩
lemmas *fourier_power_int_eventually_eq = power_int.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_power_int = power_int.map_has_laurent_expansion_at_*

lemma *zorder_fourier_power_int_eq:*
  assumes *"q ∈ ball 0 1" "f ≠ 0"*
  shows *"zorder (f powi n)^q q = n * zorder f^q q"*
⟨*proof*⟩

lemma *zorder_fourier_power_eq:*
  assumes *"q ∈ ball 0 1" "f ≠ 0"*
  shows *"zorder (f ^ n)^q q = n * zorder f^q q"*
  ⟨*proof*⟩

end

locale *fourier_expansion_meromorphic_pair = fourier_expansion_context period*
+
  f: fourier_expansion_meromorphic period f + g: fourier_expansion_meromorphic
period g
  for *period f g*
begin

sublocale *add: fourier_binop_meromorphic period f g "(+)" "(+)" "(+)"*
  ⟨*proof*⟩
lemma *fourier_expansion_meromorphic_add: "fourier_expansion_meromorphic*
period (f + g)" ⟨*proof*⟩
lemmas *fourier_add_eventually_eq = add.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_add = add.map_has_laurent_expansion_at_ii_inf*

lemma *fourier_add_eq:*
  assumes *"q ∈ ball 0 1" "¬is_pole f^q q" "¬is_pole g^q q"*
  shows *"(f + g)^q q = f^q q + g^q q"*
  ⟨*proof*⟩

lemma *zorder_fourier_add_eq1:*
  assumes *"q ∈ ball 0 1" "f ≠ 0" "g ≠ 0" "zorder f^q q < zorder g^q q"*
  shows *"zorder (f + g)^q q = zorder f^q q"*
⟨*proof*⟩

lemma *zorder_fourier_add_eq2:*
  assumes *"q ∈ ball 0 1" "f ≠ 0" "g ≠ 0" "zorder f^q q > zorder g^q q"*
  shows *"zorder (f + g)^q q = zorder g^q q"*
⟨*proof*⟩

sublocale *diff: fourier_binop_meromorphic period f g "(-)" "(-)" "(-)"*

⟨*proof*⟩
lemma *fourier_expansion_meromorphic_diff: "fourier_expansion_meromorphic period (f - g)"* ⟨*proof*⟩
lemmas *fourier_diff_eventually_eq = diff.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_diff = diff.map_has_laurent_expansion_at_ii_inf*

lemma *fourier_diff_eq:*
  assumes *"q ∈ ball 0 1" "¬is_pole f^q q" "¬is_pole g^q q"*
  shows *"(f - g)^q q = f^q q - g^q q"*
  ⟨*proof*⟩

lemma *zorder_fourier_diff_eq1:*
  assumes *"q ∈ ball 0 1" "f ≠ 0" "g ≠ 0" "zorder f^q q < zorder g^q q"*
  shows *"zorder (f - g)^q q = zorder f^q q"*
⟨*proof*⟩

lemma *zorder_fourier_diff_eq2:*
  assumes *"q ∈ ball 0 1" "f ≠ 0" "g ≠ 0" "zorder f^q q > zorder g^q q"*
  shows *"zorder (f - g)^q q = zorder g^q q"*
⟨*proof*⟩

sublocale *mult: fourier_binop_meromorphic period f g "(*)" "(*)" "(*)"*
  ⟨*proof*⟩
lemma *fourier_expansion_meromorphic_mult: "fourier_expansion_meromorphic period (f * g)"* ⟨*proof*⟩
lemmas *fourier_mult_eventually_eq = mult.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_mult = mult.map_has_laurent_expansion_at_ii_inf*

lemma *fourier_mult_eq:*
  assumes *"q ∈ ball 0 1" "¬is_pole f^q q" "¬is_pole g^q q"*
  shows *"(f * g)^q q = f^q q * g^q q"*
  ⟨*proof*⟩

lemma *zorder_fourier_mult_eq:*
  assumes *"q ∈ ball 0 1" "f ≠ 0" "g ≠ 0"*
  shows *"zorder (f * g)^q q = zorder f^q q + zorder g^q q"*
⟨*proof*⟩

sublocale *divide: fourier_binop_meromorphic period f g "(/)" "(/)" "(/)"*
  ⟨*proof*⟩
lemma *fourier_expansion_meromorphic_divide: "fourier_expansion_meromorphic period (f / g)"* ⟨*proof*⟩
lemmas *fourier_divide_eventually_eq = divide.eventually_map_fourier_eq*
lemmas *has_laurent_expansion_at_ii_inf_divide = divide.map_has_laurent_expansion_at_ii_inf*

lemma *fourier_divide_eq:*
  assumes *"q ∈ ball 0 1" "¬is_pole f^q q" "¬is_pole g^q q" "g^q q ≠ 0"*

shows *"(f / g)^q q = f^q q / g^q q"*
⟨*proof*⟩

lemma *zorder_fourier_divide_eq:*
assumes *"q ∈ ball 0 1" "f ≠ 0" "g ≠ 0"*
shows *"zorder (f / g)^q q = zorder f^q q - zorder g^q q"*
⟨*proof*⟩

end

lemma *compose_modgrp_mero_uhp_compose:*
"compose_modgrp_mero_uhp (compose_modgrp_mero_uhp f g) h =
*compose_modgrp_mero_uhp f (g * h)"*
⟨*proof*⟩

lemma *slash_mero_uhp_shift:*
"slash_mero_uhp k (shift_modgrp n) f = compose_modgrp_mero_uhp f (shift_modgrp n)"
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_laurent:*
assumes *"f has_laurent_expansion_at_i∞[period] F" "fls_subdegree F ≥ 0"*
shows *"holomorphic_at_infinity f"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_altdef:*
"f has_laurent_expansion_at_i∞[period] F ⟷
fourier_expansion_meromorphic_explicit period f F"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_0_imp_0:*
assumes *"f has_laurent_expansion_at_i∞[period] 0"*
shows *"f = 0"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_unique:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
"f has_laurent_expansion_at_i∞[period] G"
shows *"F = G"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_mult_period:*
assumes *"f has_laurent_expansion_at_i∞[period] F" "period' = period * k" "k > 0"*
shows *"f has_laurent_expansion_at_i∞[period'] (fls_compose_fps F*

```
(fps_X ^ k))"
```
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_at_0:*
assumes *"f has_laurent_expansion_at_i∞[period] F" "fls_subdegree F ≥ 0"*
shows *"eval_mero_uhp_at_ii_inf f = fls_nth F 0"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_const [laurent_expansion_intros]:*
"period > 0 ⟹ const_mero_uhp c has_laurent_expansion_at_i∞[period] fls_const c"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_0 [laurent_expansion_intros]:*
"period > 0 ⟹ 0 has_laurent_expansion_at_i∞[period] 0"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_1 [laurent_expansion_intros]:*
"period > 0 ⟹ 1 has_laurent_expansion_at_i∞[period] 1"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_of_nat [laurent_expansion_intros]:*
"period > 0 ⟹ of_nat n has_laurent_expansion_at_i∞[period] of_nat n"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_of_int [laurent_expansion_intros]:*
"period > 0 ⟹ of_int n has_laurent_expansion_at_i∞[period] of_int n"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_add [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
assumes *"g has_laurent_expansion_at_i∞[period] G"*
shows *"f + g has_laurent_expansion_at_i∞[period] F + G"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_mult [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
assumes *"g has_laurent_expansion_at_i∞[period] G"*
shows *"f * g has_laurent_expansion_at_i∞[period] F * G"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_power_int [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
shows *"f powi n has_laurent_expansion_at_i∞[period] F powi n"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_power [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
shows *"f ^ n has_laurent_expansion_at_i∞[period] F ^ n"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_inverse [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
shows *"inverse f has_laurent_expansion_at_i∞[period] inverse F"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_divide [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
assumes *"g has_laurent_expansion_at_i∞[period] G"*
shows *"f / g has_laurent_expansion_at_i∞[period] F / G"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_minus [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
shows *"-f has_laurent_expansion_at_i∞[period] (-F)"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_diff [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
assumes *"g has_laurent_expansion_at_i∞[period] G"*
shows *"f - g has_laurent_expansion_at_i∞[period] F - G"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_deriv [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
defines *"F' ≡ fls_const (2*i*pi/period) * fls_X * fls_deriv F"*
shows *"deriv_mero_uhp f has_laurent_expansion_at_i∞[period] F'"*
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_poly [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F"*
assumes *"period > 0"*
shows *"poly (map_poly const_mero_uhp P) f has_laurent_expansion_at_i∞[period]*
poly (map_poly fls_const P) F"
⟨*proof*⟩

lemma *has_laurent_expansion_at_ii_inf_poly2 [laurent_expansion_intros]:*
assumes *"f has_laurent_expansion_at_i∞[period] F" "g has_laurent_expansion_at_i∞[period] G"*
assumes *"period > 0"*
shows *"poly2 (map_poly2 const_mero_uhp P) f g has_laurent_expansion_at_i∞[period]*
poly2 (map_poly2 fls_const P) F G"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_conv_laurent:*
"f has_fps_expansion_at_i∞[period] F ⟷ f has_laurent_expansion_at_i∞[period] fps_to_fls F"
⟨*proof*⟩

lemma *holomorphic_at_infinity_via_fps:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
shows *"holomorphic_at_infinity f"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_imp_laurent:*
"f has_fps_expansion_at_i∞[period] F ⟹ f has_laurent_expansion_at_i∞[period] fps_to_fls F"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_mult_period:*
assumes *"f has_fps_expansion_at_i∞[period] F" "period' = period * k" "k > 0"*
shows *"f has_fps_expansion_at_i∞[period'] (fps_compose F (fps_X ^ k))"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_0_imp_0:*
assumes *"f has_fps_expansion_at_i∞[period] 0"*
shows *"f = 0"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_unique:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
"f has_fps_expansion_at_i∞[period] G"
shows *"F = G"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_at_0:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
shows *"eval_mero_uhp_at_ii_inf f = fps_nth F 0"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_const [fps_expansion_intros]:*
"period > 0 ⟹ const_mero_uhp c has_fps_expansion_at_i∞[period] fps_const c"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_0 [fps_expansion_intros]:*
"period > 0 ⟹ 0 has_fps_expansion_at_i∞[period] 0"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_1 [fps_expansion_intros]:*
"period > 0 ⟹ 1 has_fps_expansion_at_i∞[period] 1"

⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_of_nat [fps_expansion_intros]:*
"period > 0 ⟹ of_nat n has_fps_expansion_at_i∞[period] of_nat n"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_of_int [fps_expansion_intros]:*
"period > 0 ⟹ of_int n has_fps_expansion_at_i∞[period] of_int n"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_add [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
assumes *"g has_fps_expansion_at_i∞[period] G"*
shows *"f + g has_fps_expansion_at_i∞[period] F + G"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_mult [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
assumes *"g has_fps_expansion_at_i∞[period] G"*
shows *"f * g has_fps_expansion_at_i∞[period] F * G"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_power [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
shows *"f ^ n has_fps_expansion_at_i∞[period] F ^ n"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_minus [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
shows *"-f has_fps_expansion_at_i∞[period] (-F)"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_diff [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
assumes *"g has_fps_expansion_at_i∞[period] G"*
shows *"f - g has_fps_expansion_at_i∞[period] F - G"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_deriv [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
defines *"F' ≡ fps_const (2*i*pi/period) * fps_X * fps_deriv F"*
shows *"deriv_mero_uhp f has_fps_expansion_at_i∞[period] F'"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_deriv_1 [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞ F"*
defines *"F' ≡ fps_const (2*i*pi) * fps_X * fps_deriv F"*
shows *"deriv_mero_uhp f has_fps_expansion_at_i∞ F'"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_sum [fps_expansion_intros]:*
assumes *"⋀x. x ∈ A ⟹ f x has_fps_expansion_at_i∞[period] F x"*
assumes *"period > 0"*
shows *"(∑x∈A. f x) has_fps_expansion_at_i∞[period] (∑x∈A. F x)"*
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_poly [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F"*
assumes *"period > 0"*
shows *"poly (map_poly const_mero_uhp P) f has_fps_expansion_at_i∞[period]*
poly (map_poly fps_const P) F"
⟨*proof*⟩

lemma *has_fps_expansion_at_ii_inf_poly2 [fps_expansion_intros]:*
assumes *"f has_fps_expansion_at_i∞[period] F" "g has_fps_expansion_at_i∞[period] G"*
assumes *"period > 0"*
shows *"poly2 (map_poly2 const_mero_uhp P) f g has_fps_expansion_at_i∞[period]*
poly2 (map_poly2 fps_const P) F G"
⟨*proof*⟩

lemma *zero_has_fps_expansion_at_ii_inf_iff:*
"0 has_fps_expansion_at_i∞[period] F ⟷ F = 0 ∧ period > 0"
⟨*proof*⟩

end

3 Definition of modular forms and related concepts

theory *Modular_Forms*
imports *Meromorphic_Upper_Half_Plane Fourier_Expansion_Mero_UHP*
begin

named_theorems *mform_intros*

notation *const_mero_uhp ("⟨_⟩")*

lemma *apply_modgrp_meromorphic [meromorphic_intros]:*
assumes *"f meromorphic_on A"*
shows *"(λz. apply_modgrp h (f z)) meromorphic_on A"*

⟨*proof*⟩

definition *zorder_at_cusp_modgrp :: "int ⇒ modgrp set ⇒ mero_uhp ⇒ modgrp ⇒ int"* **where**
"zorder_at_cusp_modgrp weight G f h =
zorder_at_ii_inf (cusp_width_modgrp h G) (slash_mero_uhp weight h f)"

3.1 Weakly meromorphic forms

Weakly modular forms of weight k and level G (where G is congruence subgroup) are meromorphic functions on the upper half plane that are invariant under the weight-k slash operator for every $h \in G$.

locale *weakly_meromorphic_form = cong_subgroup G*
for *f :: mero_uhp* **and** *weight :: int* **and** *G :: "modgrp set" +*
assumes *invariant_slash_modgrp: "h ∈ G ⟹ slash_mero_uhp weight h f = f"*
begin

lemma *periodic: "compose_modgrp_mero_uhp f (shift_modgrp (cusp_width$_\infty$ G)) = f"*
⟨*proof*⟩

sublocale *fourier_expansion_locale "cusp_width$_\infty$ G" f*
⟨*proof*⟩

lemma *mero_uhp_rel_apply_modgrp:*
assumes *"h ∈ G"*
shows *"mero_uhp_rel (λz. eval_mero_uhp f (apply_modgrp h z))*
*(λz. automorphy_factor h z powi weight * eval_mero_uhp f z)"*
⟨*proof*⟩

lemma *invariant_compose_modgrp:*
assumes *"h ∈ G"*
shows *"compose_modgrp_mero_uhp f h = automorphy_factor_mero_uhp h powi weight * f"*
⟨*proof*⟩

lemmas *[analytic_intros del] = constant_on_imp_analytic_on*

lemma *is_pole_apply_modgrp_iff [simp]:*
assumes *h: "h ∈ G"*
shows *"is_pole (eval_mero_uhp f) (apply_modgrp h z) ⟷ is_pole f z"*
⟨*proof*⟩

lemma *invariant_apply_modgrp [simp]:*
assumes *h: "h ∈ G"*
shows *"eval_mero_uhp f (apply_modgrp h z) = automorphy_factor h z powi weight * eval_mero_uhp f z"*

⟨*proof*⟩

lemma *rel_imp_zorder_eq:*
assumes *"rel z z'"*
shows *"zorder f z = zorder f z'"*
⟨*proof*⟩

lemma *zorder_apply_modgrp:*
assumes *"h ∈ G" "Im z > 0"*
shows *"zorder f (apply_modgrp h z) = zorder f z"*
⟨*proof*⟩

lemma *rel_imp_is_pole_iff:*
assumes *"rel z z'"*
shows *"is_pole f z ⟷ is_pole f z'"*
⟨*proof*⟩

lemma *deriv_apply_modgrp:*
fixes *h :: modgrp*
assumes *h: "h ∈ G"*
assumes *z: "Im z > 0" "¬is_pole f z"*
defines *"c ≡ complex_of_int (modgrp_c h)"*
defines *"T ≡ automorphy_factor h"*
shows *"deriv f (apply_modgrp h z) =*
*T z powi (weight + 2) * (deriv f z + of_int weight * (c / T z) * f z)"*
⟨*proof*⟩

lemma *weakly_meromorphic_form_conj:*
"weakly_meromorphic_form (slash_mero_uhp weight h f) weight (conj_modgrp h G)"
⟨*proof*⟩

lemma *weakly_meromorphic_form_minus [intro]: "weakly_meromorphic_form (-f) weight G"*
⟨*proof*⟩

lemma *weakly_meromorphic_form_power [intro]: "weakly_meromorphic_form (f ^ n) (n * weight) G"*
⟨*proof*⟩

lemma *weakly_meromorphic_form_power_int [intro]: "weakly_meromorphic_form (f powi n) (n * weight) G"*
⟨*proof*⟩

lemma *weakly_meromorphic_form_inverse [intro]: "weakly_meromorphic_form (inverse f) (-weight) G"*
⟨*proof*⟩

end

We show the usual closure properties for weakly modular forms w.r.t. algebraic operations.

lemma (**in** *cong_subgroup*) *weakly_meromorphic_form_0 [intro]:*
"weakly_meromorphic_form 0 weight G"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *weakly_meromorphic_form_const [intro]:*
"weakly_meromorphic_form (const_mero_uhp c) 0 G"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *weakly_meromorphic_form_1 [intro]:*
"weakly_meromorphic_form 1 0 G"
⟨*proof*⟩

locale *weakly_meromorphic_form_weight_0 = weakly_meromorphic_form f 0 G*
for *f G*
begin

lemma *rel_imp_eval_eq:*
assumes *"rel z z'"*
shows *"eval_mero_uhp f z = eval_mero_uhp f z'"*
⟨*proof*⟩

end

locale *weakly_meromorphic_form_pair =*
f: weakly_meromorphic_form f weight G + g: weakly_meromorphic_form g weight' G
for *f g weight weight' G*
begin

lemma *weakly_meromorphic_form_add [intro]:*
"weight = weight' ⟹ weakly_meromorphic_form (f + g) weight G"
⟨*proof*⟩

lemma *weakly_meromorphic_form_diff [intro]:*
"weight = weight' ⟹ weakly_meromorphic_form (f - g) weight G"
⟨*proof*⟩

lemma *weakly_meromorphic_form_mult [intro]: "weakly_meromorphic_form (f * g) (weight + weight') G"*
⟨*proof*⟩

lemma *weakly_meromorphic_form_divide [intro]: "weakly_meromorphic_form (f / g) (weight - weight') G"*

⟨*proof*⟩

end

If the subgroup in question is in fact the full modular group, the following characterisation allows us to prove weak modularity more easily by only looking at the two generators S and T of the modular group.

lemma *weakly_meromorphic_formI_generators:*
  fixes *f :: mero_uhp* **and** *g :: "complex ⇒ complex"*
  assumes *k: "even k"*
  assumes *[mero_uhp_rel_intros]: "mero_uhp_rel f g"*
  assumes *"⋀z. Im z > 0 ⟹ g (z + 1) = g z"*
  assumes *"⋀z. Im z > 0 ⟹ g (-(1/z)) = z powi k * g z"*
  shows *"weakly_meromorphic_form f k UNIV"*
⟨*proof*⟩

definition *WMForms :: "modgrp set ⇒ int ⇒ mero_uhp set" ("WMForms[_,_]")*
where
  "WMForms G k = {f. weakly_meromorphic_form f k G}"

abbreviation *WMForms' :: "int ⇒ mero_uhp set" ("WMForms[_]")* **where**
  "WMForms' ≡ WMForms UNIV"

lemma *weakly_meromorphic_form_WMForms [dest]:*
  "f ∈ WMForms[G, k] ⟹ weakly_meromorphic_form f k G"
  ⟨*proof*⟩

context
  fixes *G*
  assumes *G: "cong_subgroup G"*
begin

interpretation *cong_subgroup G*
  ⟨*proof*⟩

lemma *WMForms_0 [mform_intros, simp, intro]: "0 ∈ WMForms[G, k]"*
  ⟨*proof*⟩

lemma *WMForms_const [mform_intros, simp, intro]: "const_mero_uhp c ∈ WMForms[G, 0]"*
  ⟨*proof*⟩

lemma *WMForms_is_const: "is_const_mero_uhp f ⟹ f ∈ WMForms[G, 0]"*
  ⟨*proof*⟩

lemma *WMForms_1 [mform_intros, simp, intro]: "1 ∈ WMForms[G, 0]"*
  and *WMForms_of_nat [mform_intros, simp, intro]: "of_nat n ∈ WMForms[G, 0]"*

and *WMForms_of_int [mform_intros, simp, intro]: "of_int m ∈ WMForms[G, 0]"*
and *WMForms_of_real [mform_intros, simp, intro]: "of_real x ∈ WMForms[G, 0]"*
and *WMForms_numeral [mform_intros, simp, intro]: "numeral num ∈ WMForms[G, 0]"*
⟨*proof*⟩

lemma *WMForms_uminus [mform_intros]: "f ∈ WMForms[G, k] ⟹ -f ∈ WMForms[G, k]"*
⟨*proof*⟩

lemma *WMForms_add [mform_intros]:*
"f ∈ WMForms[G, k] ⟹ g ∈ WMForms[G, k] ⟹ f + g ∈ WMForms[G, k]"
⟨*proof*⟩

lemma *WMForms_diff [mform_intros]:*
"f ∈ WMForms[G, k] ⟹ g ∈ WMForms[G, k] ⟹ f - g ∈ WMForms[G, k]"
⟨*proof*⟩

lemma *WMForms_mult [mform_intros]:*
*"f ∈ WMForms[G, k] ⟹ g ∈ WMForms[G, m - k] ⟹ f * g ∈ WMForms[G, m]"*
⟨*proof*⟩

lemma *WMForms_inverse [mform_intros]:*
assumes *"f ∈ WMForms[G, m]" "f ≠ 0" "m = -l"*
shows *"inverse f ∈ WMForms[G, l]"*
⟨*proof*⟩

lemma *WMForms_divide [mform_intros]:*
assumes *"f ∈ WMForms[G, k]" "g ∈ WMForms[G, k-m]" "g ≠ 0"*
shows *"f / g ∈ WMForms[G, m]"*
⟨*proof*⟩

lemma *WMForms_power [mform_intros]: "f ∈ WMForms[G, k] ⟹ m = n * k ⟹ f ^ n ∈ WMForms[G, m]"*
⟨*proof*⟩

lemma *WMForms_power_int [mform_intros]: "f ∈ WMForms[G, k] ⟹ m = n * k ⟹ f powi n ∈ WMForms[G, m]"*
⟨*proof*⟩

lemma *WMForms_sum [mform_intros]:*
"(⋀x. x ∈ A ⟹ f x ∈ WMForms[G, k]) ⟹ (∑x∈A. f x) ∈ WMForms[G, k]"
⟨*proof*⟩

lemma *WMForms_prod [mform_intros]:*

"(⋀x. x ∈ A ⟹ f x ∈ WMForms[G, k x]) ⟹ m = (∑x∈A. k x) ⟹ (∏x∈A. f x) ∈ WMForms[G, m]"
⟨*proof*⟩

end

3.2 Meromorphic forms

A *meromorphic form* of weight k and level G is a weakly meromorphic form that is additionally meromorphic at all cusps. The "main" cusp is located at infinity, and meromorphicity at infinity is defined as meromorphicity of the corresponding Fourier expansion at $q = 0$.

The other cusps are located at the rational numbers, and the easiest way to define meromorphicity at them is to map them to infinity via some unimodular transformation h and then demand that the transformed form be meromorphic at infinity. We simply demand that this work for *all* unimodular transformations h, so that we are sure to cover all the cusps. In practice it is enough to check one representative per coset of G (of which there are typically finitely many).

When looking at the Fourier expansion, note that our subgroup only contains *some* shift operator T^n but not necessarily the 'shift by 1' operator T, the 'normal' q-expansion w.r.t. $q = \exp(2i\pi)$ may not work. We therefore expand w.r.t. the smallest positive n such that $T^n \in G$. This is called the *cusp width* of G (at infinity).

Note that the cusp width may be different at every cusp, since the conjugated group does not necessarily contain the same shift operators as the original one.

unbundle modfun_region_notation

locale meromorphic_form = weakly_meromorphic_form +
  assumes meromorphic_at_cusps:
    "⋀h::modgrp. meromorphic_at_infinity (conj_modgrp h G) (slash_mero_uhp weight h f)"
begin

sublocale fourier_expansion_meromorphic "cusp_width$_\infty$ G" f
⟨*proof*⟩

lemma rel_imp_eval_eq_0_iff: "rel z z' ⟹ f z = 0 ⟷ f z' = 0"
  ⟨*proof*⟩

lemma eventually_no_poles_at_ii_inf: "eventually (λz. ¬is_pole f z) at_i∞"
  ⟨*proof*⟩

lemma meromorphic_form_conj:

"meromorphic_form (slash_mero_uhp weight h f) weight (conj_modgrp h G)"
⟨*proof*⟩

lemma *finite_inv_image_mero_uhp:*
assumes *"f ≠ const_mero_uhp c"*
shows *"finite (inv_image_mero_uhp f c)"*
⟨*proof*⟩

lemma *finite_zeros_mero_uhp [intro]:*
assumes *"f ≠ 0"*
shows *"finite (zeros_mero_uhp f)"*
⟨*proof*⟩

lemma *meromorphic_at_cusps':*
fixes *h :: modgrp*
defines *"k ≡ cusp_width$_\infty$ (conj_modgrp h G)"*
shows *"fourier_expansion_meromorphic k (slash_mero_uhp weight h f)"*
⟨*proof*⟩

sublocale *fourier_expansion_meromorphic "cusp_width$_\infty$ G" f*
⟨*proof*⟩

lemma *meromorphic_form_minus: "meromorphic_form (-f) weight G"*
⟨*proof*⟩

lemma *meromorphic_form_power_int: "meromorphic_form (f powi n) (n * weight) G"*
⟨*proof*⟩

lemma *meromorphic_form_power: "meromorphic_form (f ^ n) (n * weight) G"*
⟨*proof*⟩

lemma *meromorphic_form_inverse: "meromorphic_form (inverse f) (-weight) G"*
⟨*proof*⟩

end

lemma (**in** *cong_subgroup*) *meromorphic_form_0:*
"meromorphic_form 0 weight G"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_const: "meromorphic_form (const_mero_uhp c) 0 G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_1: "meromorphic_form 1 0 G"*
⟨*proof*⟩

lemmas *meromorphic_form_minus = meromorphic_form.meromorphic_form_minus*
lemmas *meromorphic_form_power = meromorphic_form.meromorphic_form_power*
lemmas *meromorphic_form_power_int = meromorphic_form.meromorphic_form_power_int*
lemmas *meromorphic_form_inverse = meromorphic_form.meromorphic_form_inverse*

lemma *meromorphic_form_add:*
assumes *"meromorphic_form f weight G"*
assumes *"meromorphic_form g weight G"*
shows *"meromorphic_form (f + g) weight G"*
⟨*proof*⟩

lemma *meromorphic_form_diff:*
assumes *"meromorphic_form f weight G"*
assumes *"meromorphic_form g weight G"*
shows *"meromorphic_form (f - g) weight G"*
⟨*proof*⟩

lemma *meromorphic_form_mult:*
assumes *"meromorphic_form f weight1 G"*
assumes *"meromorphic_form g weight2 G"*
shows *"meromorphic_form (f * g) (weight1 + weight2) G"*
⟨*proof*⟩

lemma *meromorphic_form_divide:*
assumes *"meromorphic_form f weight1 G"*
assumes *"meromorphic_form g weight2 G"*
shows *"meromorphic_form (f / g) (weight1 - weight2) G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_sum:*
assumes *"⋀x. x ∈ A ⟹ meromorphic_form (f x) weight G"*
shows *"meromorphic_form (∑x∈A. f x) weight G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_prod:*
assumes *"⋀x. x ∈ A ⟹ meromorphic_form (f x) (weight x) G"*
shows *"meromorphic_form (∏x∈A. f x) (∑x∈A. weight x) G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_sum_list:*
assumes *"⋀f. f ∈ set fs ⟹ meromorphic_form f weight G"*
shows *"meromorphic_form (sum_list fs) weight G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_prod_list:*
assumes *"list_all2 (λf weight. meromorphic_form f weight G) fs weights"*

shows *"meromorphic_form (prod_list fs) (sum_list weights) G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *meromorphic_form_sum_mset:*
assumes *"⋀f. f ∈# fs ⟹ meromorphic_form f weight G"*
shows *"meromorphic_form (sum_mset fs) weight G"*
⟨*proof*⟩

lemma (**in** *meromorphic_form*) *finite_poles_mero_uhp [intro]: "finite (poles_mero_uhp f)"*
⟨*proof*⟩

locale *meromorphic_form_full =*
fixes *f :: mero_uhp* **and** *weight :: int*
assumes *meromorphic_form_UNIV: "meromorphic_form f weight UNIV"*
begin

sublocale *meromorphic_form f weight UNIV*
rewrites *"cusp_width$_\infty$ UNIV = Suc 0"*
⟨*proof*⟩

end

locale *modular_function = meromorphic_form f 0 G*
for *f :: mero_uhp* **and** *G :: "modgrp set"*
begin

sublocale *weakly_meromorphic_form_weight_0* ⟨*proof*⟩

lemmas *[simp del] = invariant_apply_modgrp*

lemma *invariant_apply_modgrp' [simp]:*
assumes *"h ∈ G"*
shows *"eval_mero_uhp f (apply_modgrp h z) = eval_mero_uhp f z"*
⟨*proof*⟩

lemma *modular_function_minus [intro]: "modular_function (-f) G"*
⟨*proof*⟩

lemma *modular_function_inverse [intro]: "modular_function (inverse f) G"*
⟨*proof*⟩

lemma *modular_function_power [intro]: "modular_function (f ^ n) G"*
⟨*proof*⟩

lemma *modular_function_power_int [intro]: "modular_function (f powi n) G"*
⟨*proof*⟩

end

context *cong_subgroup*
begin

lemma *modular_function_0 [intro]: "modular_function 0 G"*
⟨*proof*⟩

lemma *modular_function_1 [intro]: "modular_function 1 G"*
⟨*proof*⟩

lemma *modular_function_const [intro]: "modular_function (const_mero_uhp c) G"*
⟨*proof*⟩

end

context
fixes *f g G*
assumes *f: "modular_function f G"*
assumes *g: "modular_function g G"*
begin

lemma *modular_function_add [intro]: "modular_function (f + g) G"*
⟨*proof*⟩

lemma *modular_function_diff [intro]: "modular_function (f - g) G"*
⟨*proof*⟩

lemma *modular_function_mult [intro]: "modular_function (f * g) G"*
⟨*proof*⟩

lemma *modular_function_divide [intro]: "modular_function (f / g) G"*
⟨*proof*⟩

end

definition *MeForms :: "modgrp set ⇒ int ⇒ mero_uhp set" ("MeForms[_,_]")*
where
"MeForms G k = {f. meromorphic_form f k G}"

abbreviation *MeForms' :: "int ⇒ mero_uhp set" ("MeForms[_]")* **where**
"MeForms' ≡ MeForms UNIV"

lemma *meromorphic_form_MeForms [dest]: "f ∈ MeForms[G, k] ⟹ meromorphic_form f k G"*
⟨*proof*⟩

lemma *MeForms_has_laurent_expansion_at_ii_inf:*
assumes *"f ∈ MeForms[G, k]"*
shows *"f has_laurent_expansion_at_i∞[cusp_width∞ G] laurent_expansion_at_i∞ (cusp_width∞ G) f"*
⟨*proof*⟩

lemma *MeForms_UNIV_has_laurent_expansion_at_ii_inf:*
assumes *"f ∈ MeForms[k]"*
shows *"f has_laurent_expansion_at_i∞ laurent_expansion_at_i∞ (Suc 0) f"*
⟨*proof*⟩

context
fixes *G*
assumes *G: "cong_subgroup G"*
begin

interpretation *cong_subgroup G*
⟨*proof*⟩

lemma *MeForms_0 [mform_intros, simp, intro]: "0 ∈ MeForms[G, k]"*
⟨*proof*⟩

lemma *MeForms_const [mform_intros, simp, intro]: "const_mero_uhp c ∈ MeForms[G, 0]"*
⟨*proof*⟩

lemma *MeForms_is_const: "is_const_mero_uhp f ⟹ f ∈ MeForms[G, 0]"*
⟨*proof*⟩

lemma *MeForms_1 [mform_intros, simp, intro]: "1 ∈ MeForms[G, 0]"*
and *MeForms_of_nat [mform_intros, simp, intro]: "of_nat n ∈ MeForms[G, 0]"*
and *MeForms_of_int [mform_intros, simp, intro]: "of_int m ∈ MeForms[G, 0]"*
and *MeForms_of_real [mform_intros, simp, intro]: "of_real x ∈ MeForms[G, 0]"*

and *MeForms_numeral [mform_intros, simp, intro]: "numeral num ∈ MeForms[G, 0]"*
⟨*proof*⟩

end

lemma *MeForms_uminus [mform_intros]: "f ∈ MeForms[G, k] ⟹ -f ∈ MeForms[G, k]"*
⟨*proof*⟩

lemma *MeForms_add [mform_intros]:*
"f ∈ MeForms[G, k] ⟹ g ∈ MeForms[G, k] ⟹ f + g ∈ MeForms[G, k]"
⟨*proof*⟩

lemma *MeForms_diff [mform_intros]:*
"f ∈ MeForms[G, k] ⟹ g ∈ MeForms[G, k] ⟹ f - g ∈ MeForms[G, k]"
⟨*proof*⟩

lemma *MeForms_mult [mform_intros]:*
*"f ∈ MeForms[G, k1] ⟹ g ∈ MeForms[G, k2] ⟹ k = k1 + k2 ⟹ f * g ∈ MeForms[G, k]"*
⟨*proof*⟩

lemma *MeForms_inverse [mform_intros]:*
assumes *"f ∈ MeForms[G, m]" "f ≠ 0" "m = -l"*
shows *"inverse f ∈ MeForms[G, l]"*
⟨*proof*⟩

lemma *MeForms_divide [mform_intros]:*
assumes *"f ∈ MeForms[G, k1]" "g ∈ MeForms[G, k2]" "m = k1 - k2"*
shows *"f / g ∈ MeForms[G, m]"*
⟨*proof*⟩

lemma *MeForms_power [mform_intros]: "f ∈ MeForms[G, k] ⟹ m = n * k ⟹ f ^ n ∈ MeForms[G, m]"*
⟨*proof*⟩

lemma *MeForms_power_int [mform_intros]: "f ∈ MeForms[G, k] ⟹ m = n * k ⟹ f powi n ∈ MeForms[G, m]"*
⟨*proof*⟩

context
fixes *G*
assumes *G: "cong_subgroup G"*
begin

lemma *MeForms_sum [mform_intros]:*
"(⋀x. x ∈ A ⟹ f x ∈ MeForms[G, k]) ⟹ (∑x∈A. f x) ∈ MeForms[G,

k]"
⟨proof⟩

lemma *MeForms_prod [mform_intros]:*
"(⋀x. x ∈ A ⟹ f x ∈ MeForms[G, k x]) ⟹ m = (∑x∈A. k x) ⟹ (∏x∈A. f x) ∈ MeForms[G, m]"
⟨proof⟩

end

abbreviation *MFuns :: "modgrp set ⇒ mero_uhp set" ("MFuns[_]")* **where**
"MFuns G ≡ MeForms G 0"

abbreviation *MFuns' :: "mero_uhp set" ("MFuns")* **where**
"MFuns' ≡ MFuns[UNIV]"

lemma *MFuns_altdef: "MFuns[G] = {f. modular_function f G}"*
⟨proof⟩

lemma *modular_function_MFuns [dest]: "f ∈ MFuns[G] ⟹ modular_function f G"*
⟨proof⟩

3.3 Modular forms

A modular form is a meromorphic form that is holomorphic on its entire domain, including at the cusps.

locale *modular_form = weakly_meromorphic_form +*
assumes *holo_uhp'': "holo_uhp f"*
assumes *holomorphic_at_cusps:*
"⋀h::modgrp. holomorphic_at_infinity (slash_mero_uhp weight h f)"
begin

sublocale *fourier_expansion_holomorphic "cusp_width$_\infty$ G" f*
⟨proof⟩

lemma *modular_form_conj:*
"modular_form (slash_mero_uhp weight h f) weight (conj_modgrp h G)"
⟨proof⟩

sublocale *meromorphic_form*
⟨proof⟩

lemma *modular_form_minus: "modular_form (-f) weight G"*
⟨proof⟩

end

lemma (**in** *meromorphic_form*) *modular_form_via_zorder:*
  assumes *"holo_uhp f"*
  assumes *"⋀h::modgrp. zorder_at_cusp_modgrp weight G f h ≥ 0"*
  shows *"modular_form f weight G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *modular_form_0: "modular_form 0 weight G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *modular_form_const: "modular_form (const_mero_uhp c) 0 G"*
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *modular_form_1: "modular_form 1 0 G"*
  ⟨*proof*⟩

lemmas *modular_form_minus = modular_form.modular_form_minus*

lemma *modular_form_add:*
  assumes *"modular_form f weight G" "modular_form g weight G"*
  shows *"modular_form (f + g) weight G"*
⟨*proof*⟩

lemma *modular_form_diff:*
  assumes *"modular_form f weight G" "modular_form g weight G"*
  shows *"modular_form (f - g) weight G"*
  ⟨*proof*⟩

lemma *modular_form_mult:*
  assumes *"modular_form f weight1 G" "modular_form g weight2 G"*
  shows *"modular_form (f * g) (weight1 + weight2) G"*
⟨*proof*⟩

lemma *modular_form_power:*
  assumes *"modular_form f weight G"*
  shows *"modular_form (f ^ n) (n * weight) G"*
⟨*proof*⟩

lemma *modular_form_power_int:*
  assumes *"modular_form f weight G" "n ≥ 0"*
  shows *"modular_form (f powi n) (n * weight) G"*
  ⟨*proof*⟩

lemma *modular_form_divide:*
  fixes *G :: "modgrp set"*

assumes *"modular_form f weight1 G" "modular_form g weight2 G"*
assumes *zorder_le:*
"⋀z. f ≠ 0 ⟹ g ≠ 0 ⟹ Im z > 0 ⟹ eval_mero_uhp g z = 0 ⟹

zorder_mero_uhp f z ≥ zorder_mero_uhp g z"
assumes *zorder_at_cusps_le:*
"⋀h::modgrp. f ≠ 0 ⟹ g ≠ 0 ⟹
zorder_at_cusp_modgrp weight2 G g h ≤ zorder_at_cusp_modgrp weight1 G f h"
shows *"modular_form (f / g) (weight1 - weight2) G"*
⟨*proof*⟩

definition *MForms :: "modgrp set ⇒ int ⇒ mero_uhp set" ("MForms[_,_]")* **where**
"MForms G k = {f. modular_form f k G}"

abbreviation *MForms' :: "int ⇒ mero_uhp set" ("MForms[_]")* **where**
"MForms' ≡ MForms UNIV"

lemma *modular_form_MForms [dest]: "f ∈ MForms[G, k] ⟹ modular_form f k G"*
⟨*proof*⟩

lemma *holo_uhp_MForms: "f ∈ MForms[G, k] ⟹ holo_uhp f"*
⟨*proof*⟩

lemma *no_poles_MForms: "f ∈ MForms[G, k] ⟹ ¬is_pole f z"*
⟨*proof*⟩

lemma *MForms_has_fps_expansion_at_ii_inf:*
assumes *"f ∈ MForms[G, k]"*
shows *"f has_fps_expansion_at_i∞[cusp_width∞ G] fps_expansion_at_i∞ (cusp_width∞ G) f"*
⟨*proof*⟩

lemma *MForms_UNIV_has_fps_expansion_at_ii_inf:*
assumes *"f ∈ MForms[k]"*
shows *"f has_fps_expansion_at_i∞ fps_expansion_at_i∞ (Suc 0) f"*
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_const [simp]: "eval_mero_uhp_at_ii_inf (const_mero_uhp c) = c"*
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_0 [simp]: "eval_mero_uhp_at_ii_inf 0 = 0"*
and *eval_mero_uhp_at_ii_inf_1 [simp]: "eval_mero_uhp_at_ii_inf 1 = 1"*

and *eval_mero_uhp_at_ii_inf_of_nat [simp]: "eval_mero_uhp_at_ii_inf (of_nat n) = of_nat n"*
and *eval_mero_uhp_at_ii_inf_of_int [simp]: "eval_mero_uhp_at_ii_inf (of_int k) = of_int k"*
and *eval_mero_uhp_at_ii_inf_of_numeral [simp]: "eval_mero_uhp_at_ii_inf (numeral num) = numeral num"*
⟨*proof*⟩

context *fourier_expansion_meromorphic*
begin

lemma *eval_mero_uhp_at_ii_inf_minus [simp]:*
assumes *"¬is_pole_ii_inf f"*
shows *"eval_mero_uhp_at_ii_inf (-f) = -eval_mero_uhp_at_ii_inf f"*
⟨*proof*⟩

end

context
fixes *G*
assumes *G: "cong_subgroup G"*
begin

interpretation *cong_subgroup G*
⟨*proof*⟩

lemma *MForms_0 [mform_intros, simp, intro]: "0 ∈ MForms[G, k]"*
⟨*proof*⟩

lemma *MForms_const [mform_intros, simp, intro]: "const_mero_uhp c ∈ MForms[G, 0]"*
⟨*proof*⟩

lemma *MForms_is_const: "is_const_mero_uhp f ⟹ f ∈ MForms[G, 0]"*
⟨*proof*⟩

lemma *MForms_1 [mform_intros, simp, intro]: "1 ∈ MForms[G, 0]"*
and *MForms_of_nat [mform_intros, simp, intro]: "of_nat n ∈ MForms[G, 0]"*
and *MForms_of_int [mform_intros, simp, intro]: "of_int m ∈ MForms[G, 0]"*
and *MForms_of_real [mform_intros, simp, intro]: "of_real x ∈ MForms[G, 0]"*
and *MForms_numeral [mform_intros, simp, intro]: "numeral num ∈ MForms[G, 0]"*
⟨*proof*⟩

end

lemma *MForms_uminus [mform_intros]: "f ∈ MForms[G, k] ⟹ -f ∈ MForms[G, k]"*
⟨*proof*⟩

lemma *MForms_add [mform_intros]:*
"f ∈ MForms[G, k] ⟹ g ∈ MForms[G, k] ⟹ f + g ∈ MForms[G, k]"
⟨*proof*⟩

lemma *MForms_diff [mform_intros]:*
"f ∈ MForms[G, k] ⟹ g ∈ MForms[G, k] ⟹ f - g ∈ MForms[G, k]"
⟨*proof*⟩

lemma *MForms_mult [mform_intros]:*
*"f ∈ MForms[G, k1] ⟹ g ∈ MForms[G, k2] ⟹ k = k1 + k2 ⟹ f * g ∈ MForms[G, k]"*
⟨*proof*⟩

lemma *MForms_power [mform_intros]: "f ∈ MForms[G, k] ⟹ m = n * k ⟹ f ^ n ∈ MForms[G, m]"*
⟨*proof*⟩

lemma *MForms_power_int [mform_intros]: "n ≥ 0 ⟹ f ∈ MForms[G, k] ⟹ m = n * k ⟹ f powi n ∈ MForms[G, m]"*
⟨*proof*⟩

context
fixes *G*
assumes *G: "cong_subgroup G"*
begin

lemma *MForms_sum [mform_intros]:*
"(⋀x. x ∈ A ⟹ f x ∈ MForms[G, k]) ⟹ (∑x∈A. f x) ∈ MForms[G, k]"
⟨*proof*⟩

lemma *MForms_prod [mform_intros]:*
"(⋀x. x ∈ A ⟹ f x ∈ MForms[G, k x]) ⟹ m = (∑x∈A. k x) ⟹ (∏x∈A. f x) ∈ MForms[G, m]"
⟨*proof*⟩

end

lemma *MForms_divide [mform_intros]:*
fixes *G*
assumes *"f ∈ MForms[G, k1]" "g ∈ MForms[G, k2]" "k = k1 - k2"*
assumes *"⋀z. f ≠ 0 ⟹ g ≠ 0 ⟹ Im z > 0 ⟹*
zorder_mero_uhp f z ≥ zorder_mero_uhp g z"
assumes *"⋀h. f ≠ 0 ⟹ g ≠ 0 ⟹*

```
                zorder_at_cusp_modgrp k1 G f h ≥ zorder_at_cusp_modgrp
k2 G g h"
  shows   "f / g ∈ MForms[G, k]"
```
⟨*proof*⟩

lemma *MForms_UNIV_divide [mform_intros]:*
```
  assumes "f ∈ MForms[k1]" "g ∈ MForms[k2]" "k = k1 - k2"
  assumes "⋀z. f ≠ 0 ⟹ g ≠ 0 ⟹ Im z > 0 ⟹
                zorder_mero_uhp f z ≥ zorder_mero_uhp g z"
  assumes "f ≠ 0 ⟹ g ≠ 0 ⟹ zorder_at_ii_inf 1 f ≥ zorder_at_ii_inf
1 g"
  shows   "f / g ∈ MForms[k]"
```
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_uminus [simp]:*
```
  assumes "f ∈ MForms[G, k]"
  shows   "eval_mero_uhp_at_ii_inf (-f) = -eval_mero_uhp_at_ii_inf f"
```
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_add [simp]:*
```
  assumes "f ∈ MForms[G, k]" "g ∈ MForms[G, k]"
  shows   "eval_mero_uhp_at_ii_inf (f + g) = eval_mero_uhp_at_ii_inf f
+ eval_mero_uhp_at_ii_inf g"
```
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_diff [simp]:*
```
  assumes "f ∈ MForms[G, k]" "g ∈ MForms[G, k]"
  shows   "eval_mero_uhp_at_ii_inf (f - g) = eval_mero_uhp_at_ii_inf f
- eval_mero_uhp_at_ii_inf g"
```
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_mult [simp]:*
```
  assumes "f ∈ MForms[G, k]" "g ∈ MForms[G, k']"
  shows   "eval_mero_uhp_at_ii_inf (f * g) = eval_mero_uhp_at_ii_inf f
* eval_mero_uhp_at_ii_inf g"
```
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_power [simp]:*
```
  assumes [mform_intros]: "f ∈ MForms[G, k]"
  shows   "eval_mero_uhp_at_ii_inf (f ^ n) = eval_mero_uhp_at_ii_inf f
^ n"
```
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_power_int [simp]:*
```
  assumes "f ∈ MForms[G, k]" "n ≥ 0"
  shows   "eval_mero_uhp_at_ii_inf (f powi n) = eval_mero_uhp_at_ii_inf
f powi n"
```

⟨*proof*⟩

lemma *zorder_MForms_nonneg [simp, intro]:*
assumes *"f ∈ MForms[G, k]" "Im z > 0" "f ≠ 0"*
shows *"zorder f z ≥ 0"*
⟨*proof*⟩

lemma *zorder_at_ii_inf_nonneg [simp, intro]:*
assumes *"f ∈ MForms[G, k]" "f ≠ 0"*
shows *"zorder_at_ii_inf (cusp_width$_\infty$ G) f ≥ 0"*
⟨*proof*⟩

3.4 Cusp forms

A cusp form is a modular form that vanishes at all cusps.

locale *cusp_form = modular_form +*
assumes *vanishes_at_cusps:*
"⋀h::modgrp. eval_mero_uhp_at_ii_inf (slash_mero_uhp weight h f) = 0"
begin

lemma *vanishes_at_infinity [simp]: "eval_mero_uhp_at_ii_inf f = 0"*
⟨*proof*⟩

lemma *cusp_form_conj: "cusp_form (slash_mero_uhp weight h f) weight (conj_modgrp h G)"*
⟨*proof*⟩

lemma *isolated_zero_fourier:*
assumes *"f ≠ 0"*
shows *"isolated_zero (fourier_expansion (cusp_width$_\infty$ G) f) 0"*
⟨*proof*⟩

lemma *zorder_at_ii_inf_pos:*
assumes *"f ≠ 0"*
shows *"zorder_at_ii_inf (cusp_width$_\infty$ G) f > 0"*
⟨*proof*⟩

lemma *cusp_form_minus: "cusp_form (-f) weight G"*
⟨*proof*⟩

end

lemma (**in** *cong_subgroup*) *cusp_form_0: "cusp_form 0 weight G"*
⟨*proof*⟩

lemmas *cusp_form_minus = cusp_form.cusp_form_minus*

lemma *cusp_form_add:*
assumes *"cusp_form f weight G" "cusp_form g weight G"*
shows *"cusp_form (f + g) weight G"*
⟨*proof*⟩

lemma *cusp_form_diff:*
assumes *"cusp_form f weight G" "cusp_form g weight G"*
shows *"cusp_form (f - g) weight G"*
⟨*proof*⟩

lemma *cusp_form_mult_left:*
assumes *"cusp_form f weight1 G" "modular_form g weight2 G"*
shows *"cusp_form (f * g) (weight1 + weight2) G"*
⟨*proof*⟩

lemma *cusp_form_mult_right:*
assumes *"modular_form f weight1 G" "cusp_form g weight2 G"*
shows *"cusp_form (f * g) (weight1 + weight2) G"*
⟨*proof*⟩

lemma *cusp_form_power:*
assumes *"cusp_form f weight G" "n > 0"*
shows *"cusp_form (f ^ n) (n * weight) G"*
⟨*proof*⟩

lemma *cusp_form_power_int:*
assumes *"cusp_form f weight G" "n > 0"*
shows *"cusp_form (f powi n) (n * weight) G"*
⟨*proof*⟩

definition *CForms :: "modgrp set ⇒ int ⇒ mero_uhp set" ("CForms[_,_]")*
where
"CForms G k = {f. cusp_form f k G}"

abbreviation *CForms' :: "int ⇒ mero_uhp set" ("CForms[_]")* **where**
"CForms' ≡ CForms UNIV"

lemma *cusp_form_CForms [dest]: "f ∈ CForms[G, k] ⟹ cusp_form f k G"*
⟨*proof*⟩

lemma *CForms_MForms [dest]: "f ∈ CForms[G, k] ⟹ f ∈ MForms[G, k]"*
⟨*proof*⟩

lemma *CForms_UNIV_altdef: "CForms[k] = {f∈MForms[k]. eval_mero_uhp_at_ii_inf f = 0}"*
⟨*proof*⟩

lemma *zorder_at_ii_inf_CForms:*

assumes *"f ∈ CForms[G, k]" "f ≠ 0"*
shows *"zorder_at_ii_inf (cusp_width$_\infty$ G) f > 0"*
⟨*proof*⟩

context
fixes *G*
assumes *G: "cong_subgroup G"*
begin

interpretation *cong_subgroup G*
⟨*proof*⟩

lemma *CForms_0 [mform_intros, simp, intro]: "0 ∈ CForms[G, k]"*
⟨*proof*⟩

end

lemma *CForms_add [mform_intros]:*
assumes *"f ∈ CForms[G, k]" "g ∈ CForms[G, k]"*
shows *"f + g ∈ CForms[G, k]"*
⟨*proof*⟩

lemma *CForms_uminus [mform_intros]:*
assumes *"f ∈ CForms[G, k]"*
shows *"-f ∈ CForms[G, k]"*
⟨*proof*⟩

lemma *CForms_diff [mform_intros]:*
assumes *"f ∈ CForms[G, k]" "g ∈ CForms[G, k]"*
shows *"f - g ∈ CForms[G, k]"*
⟨*proof*⟩

lemma *CForms_mult_left [mform_intros]:*
*"f ∈ CForms[G, k1] ⟹ g ∈ MForms[G, k2] ⟹ k = k1 + k2 ⟹ f * g ∈ CForms[G, k]"*
⟨*proof*⟩

lemma *CForms_mult_right [mform_intros]:*
*"f ∈ MForms[G, k1] ⟹ g ∈ CForms[G, k2] ⟹ k = k1 + k2 ⟹ f * g ∈ CForms[G, k]"*
⟨*proof*⟩

lemma *CForms_power [mform_intros]: "f ∈ CForms[G, k] ⟹ m = n * k ⟹ n > 0 ⟹ f ^ n ∈ CForms[G, m]"*
⟨*proof*⟩

lemma *CForms_power_int [mform_intros]: "f ∈ CForms[G, k] ⟹ m = n * k ⟹ n > 0 ⟹ f powi n ∈ CForms[G, m]"*
⟨*proof*⟩

lemma *CForms_sum [mform_intros]:*
"cong_subgroup G ⟹ (⋀x. x ∈ A ⟹ f x ∈ CForms[G, k]) ⟹ (∑x∈A. f x) ∈ CForms[G, k]"
⟨*proof*⟩

lemma
assumes *"cong_subgroup G"*
shows *subspace_WMForms: "mero_uhp.subspace (WMForms[G, k])"*
and *subspace_MeForms: "mero_uhp.subspace (MeForms[G, k])"*
and *subspace_MForms: "mero_uhp.subspace (MForms[G, k])"*
and *subspace_CForms: "mero_uhp.subspace (CForms[G, k])"*
⟨*proof*⟩

end

4 The valence formula for level 1 meromorphic forms

theory *Meromorphic_Forms_Valence_Formula*
imports
"Winding_Number_Eval.Winding_Number_Eval"
"Path_Automation.Path_Automation"
Argument_Principle_Sparse
Detour_Calculus.Detour_Calculus
Modular_Forms
begin

The following is an important ingedient in our evaluation of the integral in Theorem 2.4: If a function has an isolated non-essential singularity at `u` and is not identically zero, then consider the integral $\int_\gamma(\varepsilon)\frac{f'(w)}{f(w)}\,\mathrm{d}w$ where γ`(ε)` is a circular arc with radius ε around `u` beginning at the angle `a(ε)` and ending at the angle `b(ε)`.

Assume that `a(ε)` $\to$ `a` and `b(ε)` $\to$ `b` as $\varepsilon \to$ `0`. Then as $\varepsilon \to$ `0`, the value of that integral tends to `n (b - a)` i, where `n = zorder f u`.

This is in some sense a more precise version of the Argument Principle for circular paths: The Residue Theorem would tell us in such a situation that the value of the integral will tend to $2i\pi nm$ for a circular path that winds `m` times around `u` (where `m` is an integer). This proposition tells us that the same also holds for circular paths that do not wind around `u` an integer number of times; i.e. `m` can be an arbitrary real number.

proposition *contour_integral_logderiv_part_circlepath_not_essential:*
fixes *f :: "complex ⇒ complex"* **and** *a b :: "real ⇒ real"*
assumes *"f meromorphic_on {u}"*
assumes *"eventually (λz. f z ≠ 0) (at u)"*
assumes *"(a ⟶ A) (at_right 0)" "(b ⟶ B) (at_right 0)"*

assumes *"K = (of_int (zorder f u) * (B - A)) *$_R$ i"*
defines *"circ ≡ (λε. part_circlepath u ε (a ε) (b ε))"*
shows *"((λε. contour_integral (circ ε) (λz. deriv f z / f z)) ⟶ K) (at_right 0)"*
⟨*proof*⟩

lemma *winding_preserving_inverse_upper_halfspace:*
assumes *"A ⊆ {z. Im z > 0}"*
shows *"winding_preserving A inverse (λx. x)"*
⟨*proof*⟩

lemmas *[simp del] = pathstart_part_circlepath pathfinish_part_circlepath*

This is the contour around the fundamental region $\mathcal{R}_\Gamma$ that we want to integrate on. Unfortunately, since there may be roots and zeros on the contour, we will have to deform it slightly later. But for now, we will show the basic properties of this contour: it is a positively oriented (i.e. counter-clockwise) simple closed curve that contains all points in $\mathcal{R}_\Gamma$ with imaginary part < b, and it does not contain any points outside the closure of $\mathcal{R}_\Gamma$.

definition *mypath :: "real ⇒ real ⇒ complex"* **where**
*"mypath b = part_circlepath 0 1 (2/3*pi) (1/3*pi) +++*
*linepath (ϱ + 1) (1 / 2 + b *$_R$ i) +++*
*linepath (1 / 2 + b *$_R$ i) (-1 / 2 + b *$_R$ i) +++*
*linepath (-1 / 2 + b *$_R$ i) ϱ"*

lemma *mypath_ends: "cis (pi / 3) = ϱ + 1" "cis (2 * pi / 3) = ϱ"*
⟨*proof*⟩

lemma *path_mypath [simp, intro]: "path (mypath b)"*
and *valid_path_mypath [simp, intro]: "valid_path (mypath b)"*
and *closed_mypath: "pathstart (mypath b) = pathfinish (mypath b)"*
⟨*proof*⟩

lemma *sin_not_gt_one [simp]: "¬sin (x :: real) > 1"*
⟨*proof*⟩

lemma *cos_not_gt_one [simp]: "¬cos (x :: real) > 1"*
⟨*proof*⟩

lemma *simple_path_mypath [simp, intro]:*
assumes *"b > 1"*
shows *"simple_path (mypath b)"*
⟨*proof*⟩

lemma *simple_loop_mypath [simp, intro]: "b > 1 ⟹ simple_loop (mypath b)"*
⟨*proof*⟩

lemma *path_image_mypath:*

assumes *"b > sqrt 3 / 2"*
shows *"path_image (mypath b) =*
{z. norm z = 1 ∧ Im z > 0 ∧ |Re z| ≤ 1 / 2} ∪
{z. Im z = b ∧ |Re z| ≤ 1 / 2} ∪
{z. |Re z| = 1 / 2 ∧ Im z ∈ {sqrt 3 / 2..b}}"
⟨*proof*⟩

lemma *path_image_mypath_subset:*
assumes *"b > 1"*
shows *"path_image (mypath b) ⊆ closure $\mathcal{R}_\Gamma$ ∩ {z. Im z ≤ b}"*
⟨*proof*⟩

lemma *path_image_mypath_superset_frontier:*
assumes *"b > sqrt 3 / 2"*
shows *"frontier $\mathcal{R}_\Gamma$ ∩ {z. Im z ≤ b} ⊆ path_image (mypath b)"*
⟨*proof*⟩

lemma *frontier_subset_path_image_mypath:*
assumes *"b > 1"*
shows *"path_image (mypath b) ⊆ frontier $\mathcal{R}_\Gamma$ ∩ {z. Im z ≤ b} ∪*
*closed_segment (-1/2 + b *$_R$ i) (1/2 + b *$_R$ i)"* **(is** *"_ ⊆ ?rhs")*
⟨*proof*⟩

lemma *path_image_mypath_conv_frontier:*
assumes *"b > 1"*
shows *"path_image (mypath b) =*
frontier $\mathcal{R}_\Gamma$ ∩ {z. Im z ≤ b} ∪ closed_segment (-1/2 + b
**$_R$ i) (1/2 + b *$_R$ i)"*
⟨*proof*⟩

lemma *winding_number_mypath_inside:*
assumes *"z ∈ $\mathcal{R}_\Gamma$" "Im z < b" "b > 1"*
shows *"winding_number (mypath b) z = 1"*
⟨*proof*⟩

lemma *simple_loop_ccw_mypath:*
assumes *"b > 1"*
shows *"simple_loop_ccw (mypath b)"*
⟨*proof*⟩

lemma *simple_loop_orientation_mypath [simp]:*
assumes *"b > 1"*
shows *"simple_loop_orientation (mypath b) = 1"*
⟨*proof*⟩

lemma *winding_number_mypath_outside:*
assumes *"z ∉ closure $\mathcal{R}_\Gamma$ ∩ {z. Im z ≤ b}" "b > 1"*
shows *"winding_number (mypath b) z = 0"*
⟨*proof*⟩

lemma *inside_mypath:*
assumes *"b > 1"*
shows *"inside (path_image (mypath b)) = $\mathcal{R}_\Gamma$ ∩ {z. Im z < b}"*
⟨*proof*⟩

lemma *outside_mypath:*
assumes *"b > 1"*
shows *"outside (path_image (mypath b)) = -(closure $\mathcal{R}_\Gamma$ ∩ {z. Im z ≤ b})"*
⟨*proof*⟩

4.1 Constructing the deformed path

definition *mypath'_circ1* **where**
*"mypath'_circ1 ε = part_circlepath (ϱ + 1) ε (5 * pi / 6 + arcsin (ε / 2)) (pi / 2)"*

definition *mypath'_circ2* **where**
"mypath'_circ2 ε = part_circlepath ϱ ε (pi / 2) (pi / 6 - arcsin (ε / 2))"

definition *mypath'_circ3* **where**
"mypath'_circ3 ε = part_circlepath i ε (pi + arcsin (ε / 2)) (- arcsin (ε / 2))"

definition *mypath'_line* **where**
*"mypath'_line b = linepath (1/2 + b *$_R$ i) (-1/2 + b *$_R$ i)"*

locale *mypath_detour = meromorphic_form_full g k* **for** *g k +*
fixes *A :: "complex set"* **and** *b :: real*
defines *"A ≡ {z. Im z > 0 ∧ g z = 0} ∪ {z. Im z > 0 ∧ is_pole g z}"*
assumes *g_nonzero: "g ≠ 0"*
assumes *subset_zeros_poles: "A ⊆ {z. Im z ∈ {0<..<b}}"*
assumes *b_gt: "b > 1"*
begin

definition *f* **where** *"f = (λz. deriv g z / g z)"*

lemma *sparse_A: "A sparse_in {z. Im z > 0}"*
⟨*proof*⟩

lemma *open_A_compl [intro]: "open ({z. Im z > 0} - A)"*
⟨*proof*⟩

lemma *open_A_compl_subset: "X ⊆ A ⟹ open ({z. Im z > 0} - X)"*
⟨*proof*⟩

lemma *g_has_field_derivative [derivative_intros]:*
"Im z > 0 ⟹ z ∉ A ⟹ (g has_field_derivative deriv g z) (at z within X)"
⟨*proof*⟩

lemma *g_has_field_derivative' [derivative_intros]:*
assumes *"(h has_field_derivative h') (at z within X)" "Im (h z) > 0" "h z ∉ A"*
shows *"((λx. g (h x)) has_field_derivative deriv g (h z) * h') (at z within X)"*
⟨*proof*⟩

lemma *f_holomorphic: "f holomorphic_on ({z. Im z > 0} - A)"*
⟨*proof*⟩

lemma *f_analytic: "f analytic_on X"* **if** *"X ⊆ {z. Im z > 0} - A"* **for** *X*
⟨*proof*⟩

lemma *g_plus1: "g (z + 1) = g z"*
⟨*proof*⟩

lemma *g_invariant: "g (-(1 / z)) = z powi k * eval_mero_uhp g z"*
⟨*proof*⟩

lemma *A_invariant1: "z + 1 ∈ A ⟷ z ∈ A"*
⟨*proof*⟩

lemma *A_invariant2: "-(1 / z) ∈ A ⟷ z ∈ A"*
⟨*proof*⟩

lemma *deriv_g_invariant:*
assumes *"Im z > 0" "z ∉ A"*
shows *"deriv g (z + 1) = deriv g z"*
and *"deriv g (-(1 / z)) = deriv g z * z powi (k + 2) + of_int k * z * g (- 1 / z)"*
⟨*proof*⟩

lemma *f_invariant:*
assumes *"Im z > 0" "z ∉ A"*
shows *"f (z + 1) = f z"* **and** *"f (-1 / z) = f z * z ^ 2 + of_int k * z"*
⟨*proof*⟩

The singularities on the left vertical line and on the circular arc at the bottom, respectively:

definition *S_l* **where** *"S_l = {z∈A∩closure $\mathcal{R}_\Gamma$-{ϱ}. Re z = -1/2}"*
definition *S_arc* **where** *"S_arc = {z∈A∩closure $\mathcal{R}_\Gamma$-{ϱ,* i*, ϱ+1}. norm z = 1}"*

The vertical positions of the singularities on the vertical lines.

definition *Y* **where** *"Y = Im ‘ S_1"*
definition *ys* **where** *"ys = sorted_list_of_set Y"*

The angles of the singularities on the circular arc.

definition Φ **where** *"Φ = Arg ‘ {z∈S_arc. Arg z < pi / 2}"*
definition *phis* **where** *"phis = sorted_list_of_set Φ"*
definition *phis'* **where** *"phis' = phis @ [pi/2] @ rev (map (λx. pi - x) phis)"*

The following two functions turn a vertical position on the left/right line into the actual point.

definition *lv :: "real ⇒ complex"* **where** *"lv y = -1/2 + y *$_R$* i"*
definition *rv :: "real ⇒ complex"* **where** *"rv y = 1/2 + y *$_R$* i"*

The only points on the original contour that we want to include in the deformed contour are the singularities on the right vertical line (except for *ϱ + 1*).

definition *I* **where** *"I = S_1 ∪ cis ‘ (λx. pi - x) ‘ Φ"*

The points on the original contour that we want to exclude ar the singularities on the left vertical line and on the circular arc at the bottom (including *ϱ*, *i*, and *ϱ + 1*).

definition *X* **where** *"X = (+) 1 ‘ S_1 ∪ cis ‘ Φ ∪ {ϱ, i, ϱ+1}"*

All the singularities not lying on the original curve should be preserved, i.e. they should be (strictly) inside the new curve if and only if they were inside before and (strictly) outside it if and only if they were outside before.

definition *P* **where** *"P = {z. Im z ≤ 0} ∪ (A - frontier $\mathcal{R}_\Gamma$)"*

lemma *finite_A: "finite (A ∩ closure $\mathcal{R}_\Gamma$)"*
⟨*proof*⟩

lemma *Y_finite: "finite Y"*
⟨*proof*⟩

definition *vjunc* **where** *"vjunc = (Min (insert b Y) + sqrt 3 / 2) / 2"*
definition *arcjunc* **where**
"arcjunc = (if Φ = {} then pi/3 + pi/18 else (Min (insert (pi/2) Φ) + pi / 3) / 2)"
definition *arcjunc'* **where**
*"arcjunc' = (if Φ = {} then pi/3 + 2*pi/18 else (pi / 2 + Max (insert (pi / 3) Φ)) / 2)"*

lemma *b_gt': "b > sqrt 3 / 2"*

⟨*proof*⟩

lemma *Y_subset: "Y ⊆ {sqrt 3 / 2 <..< b}"*
⟨*proof*⟩

lemma *ys: "set ys = Y" "sorted_wrt (<) ys"*
⟨*proof*⟩

lemma *vjunc: "vjunc ∈ {sqrt 3 / 2 <..< b}" "∀y∈Y. vjunc < y"*
⟨*proof*⟩

lemma *vjunc_pos: "vjunc > 0"*
⟨*proof*⟩

lemma *Φ_finite: "finite Φ"*
⟨*proof*⟩

lemma *Arg_S_arc_gt:*
assumes *"z ∈ S_arc"*
shows *"Arg z > pi / 3"*
⟨*proof*⟩

lemma *Φ_subset: "Φ ⊆ {pi / 3 <..< pi / 2}"*
⟨*proof*⟩

lemma *phis: "set phis = Φ" "sorted_wrt (<) phis"*
⟨*proof*⟩

lemma *arcjunc: "arcjunc ∈ {pi/3 <..< pi/2}" "⋀x. x ∈ Φ ⟹ arcjunc < x"*
⟨*proof*⟩

lemma *arcjunc': "arcjunc' ∈ {pi/3 <..< pi/2}" "⋀x. x ∈ Φ ⟹ arcjunc' > x"*
⟨*proof*⟩

lemma *arcjunc_less_arcjunc': "arcjunc < arcjunc'"*
⟨*proof*⟩

lemma *sorted1: "sorted_wrt (<) (vjunc # ys @ [b])"*
⟨*proof*⟩

lemma *sorted2: "sorted_wrt (<) (arcjunc # phis @ [arcjunc'])"*
⟨*proof*⟩

lemma *minus_cnj_in_A_iff: "norm z = 1 ⟹ -cnj z ∈ A ⟷ z ∈ A"*
⟨*proof*⟩

lemma *minus_cnj_eq_rho_iff [simp]: "-cnj z = ϱ ⟷ z = ϱ + 1"*
  and *minus_cnj_eq_rho'_iff [simp]: "-cnj z = ϱ + 1 ⟷ z = ϱ"*
  and *minus_cnj_eq_i_iff [simp]: "-cnj z = i ⟷ z = i"*
  ⟨*proof*⟩

lemma *minus_cnj_in_S_arc_iff [simp]: "-cnj z ∈ S_arc ⟷ z ∈ S_arc"*
⟨*proof*⟩

lemma *minus_cnj_in_closure_std_fund_region_iff [simp]:*
  "- cnj z ∈ closure $\mathcal{R}_\Gamma$ ⟷ z ∈ closure $\mathcal{R}_\Gamma$"
  ⟨*proof*⟩

lemma *S_arc_decompose: "S_arc = cis ‘ Φ ∪ cis ‘ (λx. pi - x) ‘ Φ"*
⟨*proof*⟩

4.1.1 The vertical segments

fun *mypath'_vert :: "bool ⇒ real ⇒ real ⇒ real list ⇒ real ⇒ real ⇒ complex"* **where**
  "mypath'_vert left l u [] ε = linepath (rv l) (rv u)"
| "mypath'_vert left l u [y] ε = avoid_linepath left (rv l) (rv u) (rv y) ε"
| "mypath'_vert left l u (y1 # y2 # ys') ε =
    avoid_linepath left (rv l) (rv ((y1 + y2) / 2)) (rv y1) ε +++
    mypath'_vert left ((y1 + y2) / 2) u (y2 # ys') ε"

definition *mypath'_vert'*
  where *"mypath'_vert' left l u ys' ε = reversepath ((+) (-1) ∘ mypath'_vert left l u ys' ε)"*

lemma *detour_rel_mypath'_vert:*
  assumes *"sorted_wrt (<) (l # ys' @ [u])"*
  shows *"detour_rel {} (rv ‘ set ys') (linepath (rv l) (rv u))*
    (mypath'_vert False l u ys') (mypath'_vert True l u ys')"
  ⟨*proof*⟩

lemma *pathstart_mypath'_vert [simp]: "pathstart (mypath'_vert left l u ys' ε) = rv l"*
  and *pathfinish_mypath'_vert [simp]: "pathfinish (mypath'_vert left l u ys' ε) = rv u"*
  ⟨*proof*⟩

lemma *valid_path_mypath'_vert:*
  "sorted_wrt (<) (l # ys' @ [u]) ⟹ valid_path (mypath'_vert left l u ys' ε)"
⟨*proof*⟩

lemma *detour_rel_mypath'_vert':*
  assumes *"sorted_wrt (<) (l # ys' @ [u])"*

shows *"detour_rel (lv ‘ set ys’) {} (linepath (lv u) (lv l))*
(mypath’_vert’ False l u ys’) (mypath’_vert’ True l u ys’)"
⟨*proof*⟩

lemma *pathstart_mypath’_vert’ [simp]: "pathstart (mypath’_vert’ left l u ys’ ε) = lv u"*
and *pathfinish_mypath’_vert’ [simp]: "pathfinish (mypath’_vert’ left l u ys’ ε) = lv l"*
⟨*proof*⟩

lemma *valid_path_mypath’_vert’:*
"sorted_wrt (<) (l # ys’ @ [u]) ⟹ valid_path (mypath’_vert’ left l u ys’ ε)"
⟨*proof*⟩

4.1.2 The circular arcs

On one half of the arc we will simply place circular arcs to avoid the bad points.

fun *arc_aux* **where**
"arc_aux left l u [] ε = [part_circlepath 0 1 u l]"
| "arc_aux left l u [y] ε = [avoid_part_circlepath left 0 1 u l y ε]"
| "arc_aux left l u (y1 # y2 # ys’) ε =
avoid_part_circlepath left 0 1 u ((y1 + y2) / 2) y1 ε #
arc_aux left l ((y1 + y2) / 2) (y2 # ys’) ε"

definition *mypath’_arcm :: "bool ⇒ real ⇒ real ⇒ complex"* **where**
"mypath’_arcm left ε =
avoid_part_circlepath left 0 1 (pi - arcjunc’) arcjunc’ (pi / 2) ε"

definition *mypath’_arcr :: "bool ⇒ real ⇒ real ⇒ complex"*
where *"mypath’_arcr left = joinpaths_list ∘ arc_aux left arcjunc arcjunc’ (rev phis)"*

definition *mypath’_arcl :: "bool ⇒ real ⇒ real ⇒ complex"*
where *"mypath’_arcl left ε = (λz. -(1/z)) ∘ reversepath (mypath’_arcr left ε)"*

definition *mypath’_arc :: "bool ⇒ real ⇒ real ⇒ complex"* **where**
"mypath’_arc left ε = mypath’_arcl left (ε/2) +++ mypath’_arcm left ε +++ mypath’_arcr left (ε/2)"

lemma *arc_aux_not_empty [simp]: "arc_aux left l u ys’ ε ≠ []"*
⟨*proof*⟩

lemma *detour_rel_arc_aux:*
assumes *"ys’ = [] ∨ sorted_wrt (>) (u # ys’ @ [l])"*
shows *"detour_rel {} (cis ‘ set ys’) (part_circlepath 0 1 u l)*

```
            (λε. joinpaths_list (arc_aux False l u ys' ε))
            (λε. joinpaths_list (arc_aux True l u ys' ε))"
```
⟨*proof*⟩

lemma *detour_rel_arcr_aux:*
```
  "detour_rel {} (cis ‘ Φ) (part_circlepath 0 1 arcjunc' arcjunc)
     (mypath'_arcr False) (mypath'_arcr True)"
```
⟨*proof*⟩

lemma *detour_rel_arcr:*
```
  "detour_rel {} (cis ‘ Φ) (part_circlepath 0 1 arcjunc' arcjunc)
     (λε. mypath'_arcr False (ε / 2)) (λε. mypath'_arcr True (ε / 2))"
```
⟨*proof*⟩

lemma *detour_rel_arcl:*
```
  "detour_rel (cis ‘ (λx. pi - x) ‘ Φ) {} (part_circlepath 0 1 (pi - arcjunc)
(pi - arcjunc'))
     (λε. mypath'_arcl False (ε / 2)) (λε. mypath'_arcl True (ε / 2))"
```
⟨*proof*⟩

lemma *detour_rel_arcm:*
```
  "detour_rel {} {i} (part_circlepath 0 1 (pi - arcjunc') arcjunc')
     (mypath'_arcm False) (mypath'_arcm True)"
```
⟨*proof*⟩

lemma *detour_rel_arc:*
```
  "detour_rel (cis ‘ (λx. pi - x) ‘ Φ) (insert i (cis ‘ Φ))
     (part_circlepath 0 1 (pi - arcjunc) arcjunc)
     (mypath'_arc False) (mypath'_arc True)"
```
⟨*proof*⟩

lemma *pathstart_mypath'_arc_aux [simp]: "pathstart (hd (arc_aux left
l u ys' ε)) = cis u"*
  and *pathfinish_mypath'_arc_aux [simp]: "pathfinish (last (arc_aux left
l u ys' ε)) = cis l"*
  ⟨*proof*⟩

lemma *pathstart_mypath'_arcr [simp]: "pathstart (mypath'_arcr left ε)
= cis arcjunc'"*
  and *pathfinish_mypath'_arcr [simp]: "pathfinish (mypath'_arcr left ε)
= cis arcjunc"*
  ⟨*proof*⟩

lemma *pathstart_mypath'_arcl [simp]: "pathstart (mypath'_arcl left ε)
= cis (pi - arcjunc)"*
  and *pathfinish_mypath'_arcl [simp]: "pathfinish (mypath'_arcl left ε)
= cis (pi - arcjunc')"*
  ⟨*proof*⟩

lemma *pathstart_mypath'_arcm [simp]: "pathstart (mypath'_arcm left ε) = cis (pi - arcjunc')"*
  and *pathfinish_mypath'_arcm [simp]: "pathfinish (mypath'_arcm left ε) = cis arcjunc'"*
  ⟨*proof*⟩

lemma *pathstart_mypath'_arc [simp]: "pathstart (mypath'_arc left ε) = cis (pi - arcjunc)"*
  and *pathfinish_mypath'_arc [simp]: "pathfinish (mypath'_arc left ε) = cis arcjunc"*
  ⟨*proof*⟩

lemma *valid_path_arc_aux:*
  "ε ∈ {0<..<2} ⟹ ys' = [] ∨ sorted_wrt (>) (u # ys' @ [1]) ⟹
    valid_path (joinpaths_list (arc_aux left 1 u ys' ε))"
⟨*proof*⟩

lemma *valid_path_arcm: "ε ∈ {0<..<2} ⟹ valid_path (mypath'_arcm left ε)"*
  ⟨*proof*⟩

lemma *valid_path_arcr:*
  assumes *"ε ∈ {0<..<2}"*
  shows *"valid_path (mypath'_arcr left ε)"*
⟨*proof*⟩

lemma *valid_path_arcl: "eventually (λε. valid_path (mypath'_arcl left (ε / 2))) (at_right 0)"*
⟨*proof*⟩

4.2 The corners

definition *mypath'_lcorner* **where**
  "mypath'_lcorner left ε = avoid_linepath_circlepath left vjunc (pi - arcjunc) ε"

definition *mypath'_rcorner* **where**
  "mypath'_rcorner left ε = (λz. -cnj z) ∘ reversepath (mypath'_lcorner left ε)"

lemma *pathstart_mypath'_lcorner [simp]: "pathstart (mypath'_lcorner left ε) = lv vjunc"*
  and *pathfinish_mypath'_lcorner [simp]: "pathfinish (mypath'_lcorner left ε) = cis (pi - arcjunc)"*
  ⟨*proof*⟩

lemma *pathstart_mypath'_rcorner [simp]: "pathstart (mypath'_rcorner left ε) = cis arcjunc"*
  and *pathfinish_mypath'_rcorner [simp]: "pathfinish (mypath'_rcorner*

```
left ε) = rv vjunc"
```
⟨*proof*⟩

lemma *valid_path_lcorner: "ε ∈ {0..2} ⟹ valid_path (mypath'_lcorner left ε)"*
⟨*proof*⟩

lemma *valid_path_rcorner:*
assumes *"ε ∈ {0..2}"*
shows *"valid_path (mypath'_rcorner left ε)"*
⟨*proof*⟩

lemma *detour_rel_mypath'_lcorner:*
```
"detour_rel {} {ϱ}
   (linepath (lv vjunc) ϱ +++ part_circlepath 0 1 (2 / 3 * pi) (pi - arcjunc))
   (mypath'_lcorner False) (mypath'_lcorner True)"
```
⟨*proof*⟩

lemma *detour_rel_mypath'_rcorner:*
```
"detour_rel {} {ϱ + 1}
   (part_circlepath 0 1 arcjunc (pi / 3)  +++ linepath (ϱ + 1) (rv vjunc))
   (mypath'_rcorner False) (mypath'_rcorner True)"
```
⟨*proof*⟩

4.3 Putting everything together

definition *mypath'* **where**
```
"mypath' left ε = (
    mypath'_lcorner left ε +++
    mypath'_arc left ε +++
    mypath'_rcorner left ε +++
    mypath'_vert left vjunc b ys ε +++
    linepath (rv b) (lv b) +++
    mypath'_vert' left vjunc b ys ε)"
```

definition *mypath_shifted :: "real ⇒ complex"* **where**
```
"mypath_shifted =
   (linepath (lv vjunc) ϱ +++
    part_circlepath 0 1 (2/3*pi) (pi - arcjunc)) +++
    part_circlepath 0 1 (pi - arcjunc) arcjunc +++
   (part_circlepath 0 1 arcjunc (pi / 3) +++
    linepath (ϱ + 1) (rv vjunc)) +++
   linepath (rv vjunc) (rv b) +++
   linepath (rv b) (lv b) +++
   linepath (lv b) (lv vjunc)"
```

lemma *valid_path_mypath_shifted: "valid_path mypath_shifted"*
⟨*proof*⟩

lemma *eq_loops_mypath_shifted: "mypath b* $\equiv_{\circlearrowleft}$ *mypath_shifted"*
⟨*proof*⟩

lemma *eventually_detour:*
"eventually (λε. detour ε I X P (mypath b) mypath_shifted (mypath' True ε)) (at_right 0)"
⟨*proof*⟩

lemmas *valid_intros =*
valid_path_join valid_path_lcorner valid_path_rcorner
valid_path_mypath'_vert' valid_path_mypath'_vert

lemmas *path_intros = valid_intros [THEN valid_path_imp_path]*

lemma *A_decompose: "A* $\subseteq$ *I* $\cup$ *X* $\cup$ *P"*
⟨*proof*⟩

lemma *Int_cong1: "(*$\bigwedge$*x. x* $\in$ *A* $\Longrightarrow$ *x* $\in$ *B* $\longleftrightarrow$ *x* $\in$ *C)* $\Longrightarrow$ *A* $\cap$ *B = A* $\cap$ *C"*
⟨*proof*⟩

lemma *S_1_in_region: "S_1* $\subseteq$ $\mathcal{R}_\Gamma$*'"*
⟨*proof*⟩

lemma *S_r_not_in_region: "(+) 1 ‘ S_1* $\cap$ $\mathcal{R}_\Gamma$*' = {}"*
⟨*proof*⟩

lemma *S_arc_left_in_region: "(λφ. cis (pi - φ)) ‘* Φ $\subseteq$ $\mathcal{R}_\Gamma$*'"*
⟨*proof*⟩

lemma *S_arc_right_not_in_region: "cis ‘* Φ $\cap$ $\mathcal{R}_\Gamma$*' = {}"*
⟨*proof*⟩

lemma *I_subset_region: "I* $\subseteq$ $\mathcal{R}_\Gamma$*'"*
⟨*proof*⟩

lemma *X_inter_region_eq: "X* $\cap$ $\mathcal{R}_\Gamma$*' = {*ϱ*,* i*}"*
⟨*proof*⟩

definition *err :: "real* $\Rightarrow$ *complex"* **where**
*"err ε = k * i * (pi / 6 - 4 * arcsin (ε / 2))"*

lemma *eventually_between_0_and_onehalf:*
"eventually (λx. x $\in$ *{0<..<1/2}) (at_right (0 :: real))"*
⟨*proof*⟩

lemma *integral_eq1:*
"eventually (λε. ($\sum$ *p*$\leftarrow$*[mypath'_circ1 ε, mypath'_circ2 ε, mypath'_circ3*

```
ε, mypath'_line b].
                contour_integral p f) + err ε =
                of_real (2 * pi) * i * (∑z∈A ∩ ℛΓ'-{i, ϱ}. zorder g
z)) (at_right 0)"
```
⟨*proof*⟩

lemma *circ1:*
```
  "((λε. contour_integral (mypath'_circ1 ε) f) ⟶ -of_int (zorder g
ϱ) * i * pi / 3) (at_right 0)"
```
⟨*proof*⟩

lemma *circ2:*
```
  "((λε. contour_integral (mypath'_circ2 ε) f) ⟶ -of_int (zorder g
ϱ) * i * pi / 3) (at_right 0)"
```
⟨*proof*⟩

lemma *circ3:*
```
  "((λε. contour_integral (mypath'_circ3 ε) f) ⟶ -of_int (zorder g
i) * i * pi) (at_right 0)"
```
⟨*proof*⟩

lemma *err:*
```
"(err ⟶ k * i * pi / 6) (at_right 0)"
```
⟨*proof*⟩

lemma *integral_eq2:*
```
  "contour_integral (mypath'_line b) f =
     of_real (2 * pi) * i * (-of_int k / 12 + of_int (zorder g ϱ) / 3
+ of_int (zorder g i) / 2 +
       (∑z∈A∩ℛΓ'-{i,ϱ}. of_int (zorder g z)))"
```
⟨*proof*⟩

end

Theorem 6.1

theorem *MeForms_valence_formula:*
```
  assumes "f ∈ MeForms[k] - {0}"
  shows "(∑z∈zeros_mero_uhp f ∪ poles_mero_uhp f - {i,ϱ}. real_of_int
(zorder f z)) +
           of_int (zorder f ϱ) / 3 + of_int (zorder f i) / 2 + of_int
(zorder_at_ii_inf 1 f) =
           of_int k / 12"
```
⟨*proof*⟩

lemma *MeForms_valence_formula':*
```
  assumes "f ∈ MeForms[k] - {0}"
  defines "C ≡ (∑z∈zeros_mero_uhp f ∪ poles_mero_uhp f - {i, ϱ}. zorder_mero_uhp
f z)"
  shows   "12 * C + 4 * zorder_mero_uhp f ϱ + 6 * zorder_mero_uhp f i
```

```
+ 12 * zorder_at_ii_inf 1 f = k"
    (is "?lhs = _")
```
⟨*proof*⟩

lemma *MForms_valence_formula':*
  assumes *"f ∈ MForms[k] - {0}"*
  defines *"C ≡ (∑z∈zeros_mero_uhp f - {i, ϱ}. zorder_mero_uhp f z)"*
  shows *"12 * C + 4 * zorder_mero_uhp f ϱ + 6 * zorder_mero_uhp f i*
*+ 12 * zorder_at_ii_inf 1 f = k"*
⟨*proof*⟩

end

5 Some concrete level 1 modular forms

theory *Basic_Modular_Forms_Mero_UHP*
  imports *"Elliptic_Functions.Basic_Modular_Forms" Modular_Forms*
begin

5.1 Eisenstein series

definition *E_mero_uhp :: "nat ⇒ mero_uhp" ("ℰ")*
  where *"E_mero_uhp n = mero_uhp (Eisenstein_E n)"*

lemma *mero_uhp_rel_E [mero_uhp_rel_intros]: "mero_uhp_rel (ℰ n) (Eisenstein_E n)"*
  ⟨*proof*⟩

lemma *E_mero_uhp_0 [simp]: "ℰ 0 = 1"*
⟨*proof*⟩

lemma *E_mero_uhp_odd:*
  assumes *"odd n"*
  shows *"ℰ n = 0"*
⟨*proof*⟩

lemma *holo_uhp_E_mero_uhp: "holo_uhp (ℰ n)"*
⟨*proof*⟩

lemma *not_is_pole_E_mero_uhp [simp]: "¬is_pole (ℰ n) z"*
  ⟨*proof*⟩

lemma *poles_mero_uhp_E_mero_uhp [simp]: "poles_mero_uhp (ℰ n) = {}"*
  ⟨*proof*⟩

lemma *eval_E_mero_uhp [simp]: "Im z > 0 ⟹ eval_mero_uhp (ℰ n) z = Eisenstein_E n z"*
  ⟨*proof*⟩

interpretation *Eisenstein_E: fourier_expansion_holomorphic_explicit "Suc 0" "$\mathcal{E}$ n" "fps_Eisenstein_E n"*
⟨*proof*⟩

locale *Eisenstein_E_not_2 =*
  fixes *n :: nat*
  assumes *not_2: "n ≠ 2"*
begin

sublocale *weakly_meromorphic_form "$\mathcal{E}$ n" "int n" UNIV*
  rewrites *"cusp_width$_\infty$ UNIV = Suc 0"*
⟨*proof*⟩

sublocale *modular_form "$\mathcal{E}$ n" "int n" UNIV*
  rewrites *"cusp_width$_\infty$ UNIV = Suc 0"*
⟨*proof*⟩

end

lemma *E_in_MForms [mform_intros]:*
  assumes *"n ≠ 2" "m = int n"*
  shows *"$\mathcal{E}$ n ∈ MForms[m]"*
⟨*proof*⟩

lemma *eval_mero_uhp_at_ii_inf_E:*
 "eval_mero_uhp_at_ii_inf ($\mathcal{E}$ n) = (if even n then 1 else 0)"
  ⟨*proof*⟩

lemma *E_mero_uhp_eq_0_iff [simp]: "$\mathcal{E}$ n = 0 ⟷ odd n"*
⟨*proof*⟩

lemma *zorder_mero_uhp_at_ii_inf_E [simp]: "zorder_at_ii_inf (Suc 0) ($\mathcal{E}$ n) = 0"*
⟨*proof*⟩

abbreviation *Eisenstein_E4_mero_uhp :: mero_uhp ("$\mathcal{E}_4$")* **where** *"$\mathcal{E}_4$ ≡ $\mathcal{E}$ 4"*

interpretation *Eisenstein_E4: Eisenstein_E_not_2 4*
  ⟨*proof*⟩

abbreviation *Eisenstein_E6_mero_uhp :: mero_uhp ("$\mathcal{E}_6$")* **where** *"$\mathcal{E}_6$ ≡ $\mathcal{E}$ 6"*

interpretation *Eisenstein_E6: Eisenstein_E_not_2 6*
  ⟨*proof*⟩

lemmas *[fps_expansion_intros] = Eisenstein_E.has_fps_expansion_at_ii_inf_explicit*

lemma *eisenstein_series_poly'_mero_uhp:*
fixes *n :: nat*
defines *"P ≡ eisenstein_series_poly' n"*
defines *"E ≡ fps_Eisenstein_E"*
shows *"poly2 (map_poly2 of_rat P) $\mathcal{E}_4$ $\mathcal{E}_6$ = $\mathcal{E}$ (2 * n + 4)"*
⟨*proof*⟩

5.2 Modular discriminant

definition *modular_discr_mero_uhp :: "mero_uhp" ("Δ")*
where *"modular_discr_mero_uhp = const_mero_uhp ((4/3)^3 * of_real pi ^ 12) * ($\mathcal{E}_4$ ^ 3 - $\mathcal{E}_6$ ^ 2)"*

lemma *mero_uhp_rel_modular_discr [mero_uhp_rel_intros]: "mero_uhp_rel Δ modular_discr"*
⟨*proof*⟩

lemma *eval_modular_discr_mero_uhp [simp]: "Im z > 0 ⟹ eval_mero_uhp Δ z = modular_discr z"*
⟨*proof*⟩

lemma *modular_discr_in_MForms [mform_intros]: "Δ ∈ MForms[12]"*
⟨*proof*⟩

interpretation *modular_discr: modular_form Δ 12 UNIV*
rewrites *"cusp_width$_\infty$ UNIV = Suc 0"*
⟨*proof*⟩

interpretation *modular_discr: fourier_expansion_holomorphic_explicit "Suc 0" Δ fps_modular_discr*
⟨*proof*⟩

lemmas *[fps_expansion_intros] = modular_discr.has_fps_expansion_at_ii_inf_explicit*

interpretation *modular_discr: cusp_form Δ 12 UNIV*
rewrites *"cusp_width$_\infty$ UNIV = Suc 0"*
⟨*proof*⟩

lemma *modular_discr_in_CForms [mform_intros]: "Δ ∈ CForms[12]"*
⟨*proof*⟩

lemma *modular_discr_mform_nonzero [simp]: "Δ ≠ 0"*
⟨*proof*⟩

lemma *zorder_modular_discr_mero_uhp [simp]:*
assumes *"Im z > 0"*
shows *"zorder Δ z = 0"*
⟨*proof*⟩

lemma *zorder_mero_uhp_at_ii_inf_modular_discr [simp]:*
"zorder_at_ii_inf (Suc 0) Δ = 1"
⟨*proof*⟩

The following tives the Fourier expansion of the modular discriminant explicitly in terms of the Ramanujan τ function:

lemma *ramanujan_tau_sums_modular_discr:*
assumes *z: "Im z > 0"*
defines *"q ≡ to_q 1 z"*
defines *"c ≡ (2 * of_real pi) ^ 12"*
shows *"(λn. c * of_int (ramanujan_tau n) * q ^ n) sums modular_discr z"*
⟨*proof*⟩

5.3 Klein's J invariant

definition *J_mero_uhp :: "mero_uhp" ("𝒥")*
where *"J_mero_uhp = ℰ₄ ^ 3 / (ℰ₄ ^ 3 - ℰ₆ ^ 2)"*

lemma *mero_uhp_rel_J [mero_uhp_rel_intros]: "mero_uhp_rel 𝒥 Klein_J"*
⟨*proof*⟩

lemma *eval_J_mero_uhp [simp]: "Im z > 0 ⟹ eval_mero_uhp 𝒥 z = Klein_J z"*
⟨*proof*⟩

lemma *in_MForms_imp_in_MeForms: "f ∈ MForms[G, k] ⟹ f ∈ MeForms[G, k]"*
⟨*proof*⟩

lemma *J_in_MForms [mform_intros]: "𝒥 ∈ MFuns"*
⟨*proof*⟩

interpretation *Klein_J: modular_function 𝒥 UNIV*
rewrites *"cusp_width∞ UNIV = Suc 0"*
⟨*proof*⟩

interpretation *Klein_J: fourier_expansion_meromorphic_explicit "Suc 0" 𝒥 fls_Klein_J*
rewrites *"cusp_width∞ UNIV = Suc 0"*
⟨*proof*⟩

lemma *J_mero_uhp_nonzero [simp]: "𝒥 ≠ 0"*
⟨*proof*⟩

lemma *not_is_pole_Klein_J [simp]: "¬is_pole Klein_J z"*
⟨*proof*⟩

lemma *not_is_pole_Klein_J' [simp]: "¬is_pole 𝒥 z"*
⟨*proof*⟩

lemma *fourier_poles_Klein_J [simp]: "Klein_J.fourier_poles = {}"*
⟨*proof*⟩

lemma *poles_mero_uhp_J [simp]: "poles_mero_uhp 𝒥 = {}"*
⟨*proof*⟩

lemma *is_pole_fourier_expansion_Klein_J_0: "is_pole (fourier_expansion (Suc 0) 𝒥) 0"*
⟨*proof*⟩

lemma *zorder_at_ii_inf_J [simp]: "zorder_at_ii_inf (Suc 0) 𝒥 = -1"*
⟨*proof*⟩

lemma *is_pole_fourier_expansion_Klein_J_iff:*
"is_pole (fourier_expansion (Suc 0) 𝒥) q ⟷ q = 0"
⟨*proof*⟩

lemma *not_is_pole_fourier_expansion_Klein_J [simp]:*
"q ≠ 0 ⟹ ¬is_pole (fourier_expansion (Suc 0) 𝒥) q"
⟨*proof*⟩

The following gives the Fourier expansion of Klein's J invariant explicitly in terms of the Klein c numbers:

lemma *sums_Klein_J:*
assumes *z: "Im z > 0"*
defines *"q ≡ to_q 1 z"*
shows *"(λi. of_int (Klein_c i) / 1728 * q ^ i) sums (Klein_J z - 1 / (1728 * q))"*
⟨*proof*⟩

end

6 Level 1 modular functions

theory *Modular_Functions*
imports *Meromorphic_Forms_Valence_Formula Basic_Modular_Forms_Mero_UHP*
begin

unbundle *modgrp_notation*

6.1 The weighted multiplicity of a zero/pole

definition *zorder_modgrp :: "modgrp set ⇒ mero_uhp ⇒ complex ⇒ int"*
where
"zorder_modgrp G f z = (if f = 0 then 0 else zorder (eval_mero_uhp f) z div ellorder_modgrp G z)"

abbreviation *zorder_modgrp_UNIV :: "mero_uhp ⇒ complex ⇒ int" ("(<notation=prefix>zorder[*
where
"zorder[Γ] ≡ zorder_modgrp UNIV"

lemma (**in** *meromorphic_form*) *rel_imp_zorder_modgrp_eq:*
assumes *"rel z z'"*
shows *"zorder[Γ] f z = zorder[Γ] f z'"*
⟨*proof*⟩

lemma
assumes *"f ∈ MFuns - {0}"*
shows *MFuns_zorder_rho_multiple_3: "3 dvd zorder f ϱ"*
and *MFuns_zorder_i_multiple_2: "2 dvd zorder f* i*"*
⟨*proof*⟩

lemma *MFuns_ellorder_dvd_zorder:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"int (ellorder_modgrp UNIV z) dvd zorder f z"*
⟨*proof*⟩

lemma *zorder_conv_zorder_modgrp:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder f z = zorder[Γ] f z * ellorder_modgrp UNIV z"*
⟨*proof*⟩

lemma *zorder_modgrp_nonneg_iff [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] f z ≥ 0 ⟷ ¬is_pole f z"*
⟨*proof*⟩

lemma *zorder_modgrp_neg_iff [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] f z < 0 ⟷ is_pole f z"*
⟨*proof*⟩

lemma *zorder_modgrp_pos_iff [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] f z > 0 ⟷ ¬is_pole f z ∧ f z = 0"*
⟨*proof*⟩

lemma *zorder_modgrp_nonpos_iff [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] f z ≤ 0 ⟷ is_pole f z ∨ f z ≠ 0"*
⟨*proof*⟩

lemma *zorder_modgrp_eq_0_iff [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*

shows *"zorder[Γ] f z = 0 ⟷ ¬is_pole f z ∧ f z ≠ 0"*
⟨*proof*⟩

lemma *zorder_modgrp_cmult [simp]:*
assumes *"Im z > 0" "c ≠ 0"*
shows *"zorder[Γ] (⟨c⟩ * f) z = zorder[Γ] f z"*
⟨*proof*⟩

lemma *zorder_modgrp_cmult' [simp]:*
assumes *"Im z > 0" "c ≠ 0"*
shows *"zorder[Γ] (f * ⟨c⟩) z = zorder[Γ] f z"*
⟨*proof*⟩

lemma *zorder_modgrp_uminus [simp]:*
assumes *"Im z > 0"*
shows *"zorder[Γ] (-f) z = zorder[Γ] f z"*
⟨*proof*⟩

lemma *zorder_modgrp_const [simp]:*
assumes *"Im z > 0"*
shows *"zorder[Γ] (const_mero_uhp c) z = 0"*
⟨*proof*⟩

lemma *zorder_modgrp_const' [simp]:*
assumes *"Im z > 0" "is_const_mero_uhp f"*
shows *"zorder[Γ] f z = 0"*
⟨*proof*⟩

lemma *zorder_modgrp_inverse [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] (inverse f) z = -zorder[Γ] f z"*
⟨*proof*⟩

lemma *zorder_modgrp_mult:*
assumes *"f ∈ MFuns - {0}" "g ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] (f * g) z = zorder[Γ] f z + zorder[Γ] g z"*
⟨*proof*⟩

lemma *zorder_modgrp_divide:*
assumes *"f ∈ MFuns - {0}" "g ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] (f / g) z = zorder[Γ] f z - zorder[Γ] g z"*
⟨*proof*⟩

lemma *zorder_modgrp_power_int [simp]:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] (f powi n) z = n * zorder[Γ] f z"*
⟨*proof*⟩

lemma *zorder_modgrp_power [simp]:*

assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] (f ^ n) z = n * zorder[Γ] f z"*
⟨*proof*⟩

lemma *zorder_modgrp_prod:*
assumes *"⋀x. x ∈ A ⟹ f x ∈ MFuns - {0}" "Im z > 0"*
shows *"zorder[Γ] (∏x∈A. f x) z = (∑x∈A. zorder[Γ] (f x) z)"*
⟨*proof*⟩

lemma *MFuns_valence_formula:*
assumes *"f ∈ MFuns - {0}"*
defines *"Z ≡ zeros_mero_uhp f ∪ poles_mero_uhp f"*
shows *"(∑z∈Z. zorder[Γ] f z) + zorder_at_ii_inf 1 f = 0"*
⟨*proof*⟩

6.2 Degree

definition *degree_modfun :: "mero_uhp ⇒ nat"* **where**
"degree_modfun f =
(if is_const_mero_uhp f then 0
else (∑z∈zeros_mero_uhp f. nat (zorder[Γ] f z)) + nat (zorder_at_ii_inf 1 f))"

lemma *degree_modfun_is_const [simp]: "is_const_mero_uhp f ⟹ degree_modfun f = 0"*
⟨*proof*⟩

The following two statements show that the number of zeros of a modular function is the same as the number of poles of a modular function, with all the usual conventions about counting zeros and poles of modular functions.

lemma *int_degree_modfun_conv_zeros:*
assumes *"f ∈ MFuns"*
shows *"int (degree_modfun f) = (∑z∈zeros_mero_uhp f. zorder[Γ] f z) + max 0 (zorder_at_ii_inf 1 f)"*
⟨*proof*⟩

lemma *int_degree_modfun_conv_poles:*
assumes *"f ∈ MFuns"*
shows *"int (degree_modfun f) = (∑z∈poles_mero_uhp f. -zorder[Γ] f z) - min 0 (zorder_at_ii_inf 1 f)"*
⟨*proof*⟩

lemma *abs_zorder_modgrp_le_degree_modfun:*
assumes *"f ∈ MFuns - {0}" "Im z > 0"*
shows *"|zorder[Γ] f z| ≤ degree_modfun f"*
⟨*proof*⟩

lemma *abs_zorder_at_ii_inf_le_degree_modfun:*
assumes *"f ∈ MFuns - {0}"*

shows *"|zorder_at_ii_inf 1 f| ≤ degree_modfun f"*
⟨*proof*⟩

Together with this, the previous two results directly imply Apostol's Theorem 2.5: If $f(z)$ is a non-constant modular function, then for any constant c, the modular function $f(z) + c$ has the same degree as the modular function $f(z)$. Therefore, the number of zeros of $f(z) - c$ is the same as the number of zeros in $f(z)$.

In other words: $f(z)$ takes every value $z \in \mathbb{C}$ equally (and finitely) often – and that number is its degree.

lemma *degree_modfun_plus_const_eq:*
assumes *f: "f ∈ MFuns"*
shows *"degree_modfun (f + const_mero_uhp c) = degree_modfun f"*
⟨*proof*⟩

lemma *WMForms_nonpole_exists:*
assumes *"f ∈ WMForms[k]"*
obtains *z* **where** *"Im z > 0" "z ∈ $\mathcal{R}_\Gamma$'" "¬is_pole f z"*
⟨*proof*⟩

The only modular functions of degree 0 are the constant functions.

theorem *degree_modfun_eq_0_iff:*
assumes *f: "f ∈ MFuns"*
shows *"degree_modfun f = 0 ⟷ is_const_mero_uhp f"*
⟨*proof*⟩

lemma *degree_modfun_pos_iff:*
assumes *f: "f ∈ MFuns"*
shows *"degree_modfun f > 0 ⟷ ¬is_const_mero_uhp f"*
⟨*proof*⟩

6.3 Range

We define the *range* of a meromorphic function on the upper half plane to be the set of all the values it takes in the upper half plane (minus the places where it has poles) or at the cusp $i\infty$ (if there is one).

definition *range_mero_uhp :: "mero_uhp ⇒ complex set"* **where**
"range_mero_uhp f = f ‘ {z∈$\mathcal{H}$. ¬is_pole f z} ∪ {L. (f ⟶ L) at_i∞}"

lemma *range_mero_uhpI1:*
assumes *"eval_mero_uhp f z = y" "Im z > 0" "¬is_pole f z"*
shows *"y ∈ range_mero_uhp f"*
⟨*proof*⟩

lemma *range_mero_uhpI2:*
assumes *"(eval_mero_uhp f ⟶ y) at_i∞"*
shows *"y ∈ range_mero_uhp f"*

⟨*proof*⟩

lemma *range_mero_uhpE:*
assumes *"y ∈ range_mero_uhp f"*
obtains *z* **where** *"z ∈ ℋ" "¬is_pole f z" "f z = y" | "(f ⟶ y) at_i∞"*
⟨*proof*⟩

lemma *range_mero_uhp_altdef1:*
assumes *"f ∈ MeForms[G, k]"*
defines *"n ≡ cusp_width∞ G"*
defines *"F ≡ fourier_expansion n f"*
shows *"range_mero_uhp f = F ‘ {z∈ball 0 1. ¬is_pole F z}"*
⟨*proof*⟩

lemma *range_mero_uhp_altdef2:*
assumes *"f ∈ MFuns"*
defines *"P ≡ (if f ≠ 0 ∧ zorder_at_ii_inf (Suc 0) f ≥ 0 then {eval_mero_uhp_at_ii_inf f} else {})"*
shows *"range_mero_uhp f = f ‘ (ℛΓ’ - poles_mero_uhp f) ∪ P"*
⟨*proof*⟩

6.4 Surjectivity and injectivity

A non-constant meromorphic form w.r.t. the full modular group is always surjective.

lemma *MFuns_surj_obtain:*
assumes *f: "f ∈ MFuns" "¬is_const_mero_uhp f"*
obtains *z* **where**
"z ∈ ℛΓ’" "¬is_pole f z" "f z = c"
| "zorder_at_ii_inf 1 f ≥ 0" "eval_mero_uhp_at_ii_inf f = c"
⟨*proof*⟩

theorem *MFuns_surj:*
assumes *f: "f ∈ MFuns" "¬is_const_mero_uhp f"*
shows *"range_mero_uhp f = UNIV"*
⟨*proof*⟩

A modular function of degree 1 is injective on the fundamental region. Note that one could prove something stronger, namely that it is injective on its fundamental region plus its cusp.

lemma *MFuns_degree_1_imp_inj_on:*
assumes *f: "f ∈ MFuns" "degree_modfun f = 1"*
shows *"inj_on f (ℛΓ’ - poles_mero_uhp f)"*
⟨*proof*⟩

Every non-constant modular function either has a pole in the fundamental region or a pole at the cusp.

theorem *MFuns_pole_exists:*

assumes *"f ∈ MFuns" "¬is_const_mero_uhp f"*
shows *"poles_mero_uhp f ≠ {} ∨ zorder_at_ii_inf 1 f < 0"*
⟨*proof*⟩

Every non-constant modular function either has a pole or a zero in the fundamental region.

lemma *MFuns_zero_or_pole_exists:*
assumes *"f ∈ MFuns" "¬is_const_mero_uhp f"*
shows *"poles_mero_uhp f ∪ zeros_mero_uhp f ≠ {}"*
⟨*proof*⟩

Apostol's Theorem 2.6: a bounded modular function is constant.

theorem *modular_and_bounded_imply_constant:*
assumes *"f ∈ MFuns"* **and** *"bounded (f ` (ℛ_Γ' - poles_mero_uhp f))"*
shows *"is_const_mero_uhp f"*
⟨*proof*⟩

end

7 Properties of Klein's J invariant

theory *Klein_J*
imports *Modular_Functions*
begin

The Klein J function has degree 1.

lemma *degree_modfun_J: "degree_modfun 𝒥 = 1"*
⟨*proof*⟩

Theorem 2.7

lemma *zorder_J_mero_uhp_conv_Klein_J:*
assumes *"Im z > 0"*
shows *"zorder 𝒥 z = zorder Klein_J z"*
⟨*proof*⟩

lemma *zorder_modgrp_J_minus_const:*
assumes *"Im w > 0"*
shows *"zorder[Γ] (𝒥 - ⟨c⟩) w = (if Klein_J w = c then 1 else 0)"*
⟨*proof*⟩

lemma *zorder_modgrp_J_rho [simp]: "zorder[Γ] 𝒥 ϱ = 1"*
⟨*proof*⟩

lemma *zorder_modgrp_J_rho' [simp]: "z ∼_Γ ϱ ⟹ zorder[Γ] 𝒥 z = 1"*
⟨*proof*⟩

lemma *zorder_modgrp_J_ii [simp]: "zorder[Γ] (𝒥 - 1)* i *= 1"*
⟨*proof*⟩

lemma *zorder_modgrp_J_ii' [simp]: "z* $\sim_\Gamma$ i $\Longrightarrow$ *zorder[*Γ*] (*$\mathcal{J}$ *- 1) z = 1"*
⟨*proof*⟩

lemma *zorder_Klein_J:*
assumes *"Im z > 0"*
shows *"zorder (*λ*u. Klein_J u - Klein_J z) z = ellorder_modgrp UNIV z"*
⟨*proof*⟩

Apostol's Theorem 2.7

lemma *zorder_Klein_J_rho: "zorder Klein_J* ϱ *= 3"*
⟨*proof*⟩

lemma *zorder_Klein_J_rho':*
assumes *"z* $\sim_\Gamma$ ϱ*"*
shows *"zorder Klein_J z = 3"*
⟨*proof*⟩

lemma *zorder_Klein_J_ii: "zorder (*λ*z. Klein_J z - 1)* i *= 2"*
⟨*proof*⟩

lemma *zorder_Klein_J_ii':*
assumes *"z* $\sim_\Gamma$ i*"*
shows *"zorder (*λ*w. Klein_J w - 1) z = 2"*
⟨*proof*⟩

7.1 Bijectivity

theorem *bij_betw_Klein_J: "bij_betw Klein_J* $\mathcal{R}_\Gamma$*' UNIV"*
⟨*proof*⟩

lemma *inj_on_Klein_J: "inj_on Klein_J* $\mathcal{R}_\Gamma$*'"*
⟨*proof*⟩

lemma *Klein_J_eqD:*
assumes *"Klein_J z1 = Klein_J z2" "Im z1 > 0" "Im z2 > 0"*
shows *"z1* $\sim_\Gamma$ *z2"*
⟨*proof*⟩

lemma *Klein_J_eq_iff:*
assumes *"Im z1 > 0" "Im z2 > 0"*
shows *"Klein_J z1 = Klein_J z2* $\longleftrightarrow$ *z1* $\sim_\Gamma$ *z2"*
⟨*proof*⟩

lemma *Klein_J_eq_0_iff: "Im z > 0* $\Longrightarrow$ *Klein_J z = 0* $\longleftrightarrow$ *z* $\sim_\Gamma$ ϱ*"*
⟨*proof*⟩

lemma *Klein_J_eq_1_iff: "Im z > 0 ⟹ Klein_J z = 1 ⟷ z ∼Γ* i"
⟨*proof*⟩

lemma *surj_Klein_J: "∃z. z ∈ ℛΓ' ∧ Klein_J z = u"*
⟨*proof*⟩

lemma *std_fund_region'_unique:*
assumes *"z ∈ ℛΓ'" "z' ∈ ℛΓ'" "z ∼Γ z'"*
shows *"z = z'"*
⟨*proof*⟩

definition *modular_group_repr :: "complex ⇒ complex"* **where**
"modular_group_repr z = (if Im z ≤ 0 then 0 else (THE z'. z ∼Γ z' ∧ z' ∈ ℛΓ'))"

lemma *modular_group_repr:*
assumes *"Im z > 0"*
shows *"z ∼Γ modular_group_repr z"* **and** *[intro]: "modular_group_repr z ∈ ℛΓ'"*
⟨*proof*⟩

lemma *modular_group_repr_eqI: "z ∼Γ z' ⟹ z' ∈ ℛΓ' ⟹ modular_group_repr z = z'"*
⟨*proof*⟩

lemma *Im_modular_group_repr_pos_iff [simp]:*
"Im (modular_group_repr z) > 0 ⟷ Im z > 0"
⟨*proof*⟩

lemma *Im_modular_group_repr_nonneg_iff [simp]:*
"Im (modular_group_repr z) ≤ 0 ⟷ Im z ≤ 0"
⟨*proof*⟩

lemma *modular_group_repr_rel_iff_left [simp]: "modular_group_repr z ∼Γ z' ⟷ z ∼Γ z'"*
⟨*proof*⟩

lemma *modular_group_repr_rel_iff_right [simp]: "z' ∼Γ modular_group_repr z ⟷ z' ∼Γ z"*
⟨*proof*⟩

lemma *modular_group_repr_in_std_fund_region'_iff: "modular_group_repr z ∈ ℛΓ' ⟷ Im z > 0"*
⟨*proof*⟩

lemma *modular_group_repr_eq_iff:*
assumes *"Im z > 0" "Im z' > 0"*
shows *"modular_group_repr z = modular_group_repr z' ⟷ z ∼Γ z'"*
⟨*proof*⟩

lemma *modular_group_repr_eq_iff':*
assumes *"Im z > 0"*
shows *"modular_group_repr z = z' ⟷ z' ∈ $\mathcal{R}_\Gamma$' ∧ z ∼$_\Gamma$ z'"*
⟨*proof*⟩

lemma *ellorder_modgrp_UNIV_modular_group_repr [simp]:*
"ellorder_modgrp UNIV (modular_group_repr z) = ellorder_modgrp UNIV z"
⟨*proof*⟩

7.2 Modular functions as rational functions of J

lemma *poly_in_MFuns:*
assumes *"⋀i. poly.coeff p i ∈ MFuns[G]" "f ∈ MFuns[G]" "k = 0"*
shows *"poly p f ∈ MeForms[G, k]"*
⟨*proof*⟩

The theorems in this subsection consitute Apostol's Theorem 2.8. First of all: any function that can be expressed as a rational function of J is a modular function.

theorem *ratfun_J_modform_in_MFuns:*
fixes *p q :: "complex poly"*
defines *"f ≡ eval_poly const_mero_uhp p $\mathcal{J}$ / eval_poly const_mero_uhp q $\mathcal{J}$"*
shows *"f ∈ MFuns"*
⟨*proof*⟩

In the other direction: every modular function can be expressed as a rational function in J.

lemma *in_terms_of_J_aux:*
assumes *f: "f ∈ MFuns - {0}"*
defines *"Z ≡ zeros_mero_uhp f ∪ poles_mero_uhp f"*
obtains *c :: complex*
where *"f = ⟨c⟩ * (∏w∈Z. ($\mathcal{J}$ - ⟨Klein_J w⟩) powi zorder[Γ] f w)"*
⟨*proof*⟩

theorem *in_terms_of_J:*
assumes *f: "f ∈ MFuns"*
obtains *p q :: "complex poly"*
where *"q ≠ 0" "f = eval_poly const_mero_uhp p $\mathcal{J}$ / eval_poly const_mero_uhp q $\mathcal{J}$"*
⟨*proof*⟩

7.3 Non-conformality of modular functions at i and ϱ

context *meromorphic_form*
begin

context
  fixes *f' h c d fac z*
  defines *"f' ≡ deriv_mero_uhp f"*
  defines *"c ≡ modgrp_c h"* **and** *"d ≡ modgrp_d h"*
  defines *"fac ≡ automorphy_factor h z"*
  assumes *z: "Im z > 0" "apply_modgrp h z = z" "¬is_pole f z"* **and** *h [simp]: "h ∈ G"*
begin

lemma *apply_modgrp_fixpoint_imp_deriv_eq:*
  *"(1 - fac powi (weight+2)) * f' z = of_int (weight * c) * fac powi (weight+1) * f z"*
⟨*proof*⟩

If we have a meromorphic function $f(z)$ of weight Γ and some singular transformation $h \in \Gamma$, then the derivative of f must vanish at any fixed points of h (i.e. f is not conformal there).

theorem *apply_modgrp_fixpoint_imp_deriv_eq_0:*
  assumes *"weight = 0" "is_singular_modgrp h"*
  shows *"deriv f z = 0"*
⟨*proof*⟩

end

end

context *modular_function*
begin

lemma *deriv_apply_modgrp:*
  assumes *"Im z > 0" "¬is_pole f z" "h ∈ G"*
  shows *"deriv f (apply_modgrp h z) = deriv f z * automorphy_factor h z ^ 2"*
⟨*proof*⟩

Since ρ and i are fixed points of the transformations $T^{-1}S$ and S, respectively, any modular function over a subgroup containing the corresponding transformation is non-conformal at ρ and i, respectively:

corollary *deriv_rho_eq_0:*
  assumes *"¬ is_pole f ϱ" "shift_modgrp (-1) * S_modgrp ∈ G"*
  shows *"deriv f ϱ = 0"*
⟨*proof*⟩

corollary *deriv_ii_eq_0:*
  assumes *"¬ is_pole f* i*" "S_modgrp ∈ G"*
  shows *"deriv f* i *= 0"*
  ⟨*proof*⟩

corollary *deriv_rho_eq_0':*
assumes *"¬is_pole f z" "rel z ϱ" "shift_modgrp (-1) * S_modgrp ∈ G"*
shows *"deriv f z = 0"*
⟨*proof*⟩

corollary *deriv_ii_eq_0':*
assumes *"¬is_pole f z" "rel z* i*" "S_modgrp ∈ G"*
shows *"deriv f z = 0"*
⟨*proof*⟩

end

We now show that every point in $\mathcal{R}_\Gamma$' except for the special points ϱ and i has an open neighbourhood in which no two points are equivalent. This means that in principle, a modular function *can* be conformal (i.e. locally injective) there, and indeed that Klein's `J` function is conformal there.

Any point inside $\mathcal{R}_\Gamma$ clearly has such a neighbourhood (namely $\mathcal{R}_\Gamma$ itself) since $\mathcal{R}_\Gamma$ is open and does not contain equivalent points. The problematic cases are the ones where `z` lies on the border of $\mathcal{R}_\Gamma$, i.e. either on the left vertical line `L` or on the left half `C` of the circular arc.

The geometric intuition in both cases is fairly obvious: we can find a suitably small neighbourhood such that each point in it either lies in $\mathcal{R}_\Gamma$, the adjacent region, or the border between them. Thus any two equivalent points `x`, `y` in that neighbourhood that are not equal must fulfil w.l.o.g. `x = y + 1` (in the case of `L`) or `x = -1/y` (in the case of `C`). But in both cases, this transformation sends `y` relatively far away from `x`, so by making our neighbourhood sufficiently small, we preclude that case.

proposition *std_fund_region'_locally_no_equiv_points:*
assumes *"z ∈ $\mathcal{R}_\Gamma$' - {*i*, ϱ}"*
obtains *A* **where** *"open A" "z ∈ A" "A ⊆ {z. Im z > 0}" "⋀x y. x ∈ A ⟹ y ∈ A ⟹ x ∼$_\Gamma$ y ⟹ x = y"*
⟨*proof*⟩

The Klein `J` function is conformal everywhere except at i and ϱ (where no modular function can be conformal).

lemma *deriv_Klein_J_nonzero':*
assumes *"z ∈ $\mathcal{R}_\Gamma$' - {*i*, ϱ}"*
shows *"deriv Klein_J z ≠ 0"*
⟨*proof*⟩

lemma *deriv_Klein_J_nonzero:*
assumes *"Im z > 0" "¬z ∼$_\Gamma$ ϱ" "¬z ∼$_\Gamma$* i*"*
shows *"deriv Klein_J z ≠ 0"*
⟨*proof*⟩

lemma *deriv_J_modform [simp]:*
assumes *"Im z > 0"*

shows *"deriv 𝒥 z = deriv Klein_J z"*
⟨*proof*⟩

theorem *deriv_Klein_J_eq_0_iff:*
assumes *"Im z > 0"*
shows *"deriv Klein_J z = 0 ⟷ z ∼Γ i ∨ z ∼Γ ϱ"*
⟨*proof*⟩

7.4 Real values

Since the power series for J only has real coefficients, it has a certain symmetry with respect to the imaginary axis.

lemma *Klein_J_mirror:*
assumes *z: "Im z > 0"*
shows *"Klein_J (-cnj z) = cnj (Klein_J z)"*
⟨*proof*⟩

In particular, this means that J takes real values on the half-circle of radius 1 around the origin, on the vertical axes at real value $\pm\frac{1}{2}$, and on the imaginary axis (i.e. in particular on the border of the fundamental domain).

lemma *Klein_J_arc_is_real:*
assumes *"norm z = 1" "Im z > 0"*
shows *"Klein_J z ∈ ℝ"*
⟨*proof*⟩

lemma *Klein_J_vertical_onehalf_is_real:*
assumes *"|Re z| = 1 / 2" "Im z > 0"*
shows *"Klein_J z ∈ ℝ"*
⟨*proof*⟩

lemma *Klein_J_imag_axis_is_real:*
assumes *"Re z = 0" "Im z > 0"*
shows *"Klein_J z ∈ ℝ"*
⟨*proof*⟩

We now show a few more mapping properties of Klein's `J` function, hinted at (mostly without proof) in Section 2.7 in Apostol's book. We will look at three regions in and around the standard fundamental region $\mathcal{R}_\Gamma$: the left vertical line, the left half of the circular arc, and the segment of the imaginary axis with *Im(z)* $\geq$ *1*.
We will show that Klein's `J` function takes on precisely all real numbers on these regions. To be more precise:

- On the left vertical line, it starts with the value 0 at ϱ and then decreases strictly monotonically towards $-\infty$ as one moves up.
- On the circular arc, it starts with the value 0 at ϱ and increases strictly monotonically until reaching the value 1 at i

- On the $\mathit{Im}(z) \geq 1$-segment of the imaginary axis, it starts with the value 1 at i and increases strictly monotonically to ∞ as one moves up.

We define a path with domain $\mathbb{R}$ whose image is precisely all the values in $\mathcal{R}_\Gamma{}'$ at which Klein's J function takes real values. We will furthermore show that that real value strictly increases as we traverse the path, and that it tends to ∞ towards the end of the path and towards $-\infty$ at the beginning of the path.

Together with the injectivity of J on $\mathcal{R}_\Gamma{}'$, this implies that these are the *only* points at which J takes real values.

The path consists of precisely the three edges described earlier: a vertical line with real value $-\frac{1}{2}$ that ends at ϱ, a circular arc of radius 1 connecting ϱ and i, and another vertical line extending upwards from i (lying on the imaginary axis):

definition *Klein_J_aux_path :: "real ⇒ complex"* **where**
"Klein_J_aux_path x =
*(if x ≤ 0 then -1/2 + (sqrt 3 / 2 - x) * i*
*else if x ≤ 1 then cis (2 / 3 * pi - pi / 6 * x)*
*else x * i)"*

lemma *Klein_J_aux_path_0 [simp]: "Klein_J_aux_path 0 = ϱ"*
⟨*proof*⟩

lemma *Klein_J_aux_path_1 [simp]: "Klein_J_aux_path 1 = i"*
⟨*proof*⟩

lemma *continuous_Klein_J_aux_path: "continuous_on A Klein_J_aux_path"*
⟨*proof*⟩

lemma *Im_Klein_J_aux_path_pos: "Im (Klein_J_aux_path x) > 0"*
⟨*proof*⟩

lemma *Im_Klein_J_aux_path_nonzero: "Im (Klein_J_aux_path x) ≠ 0"*
⟨*proof*⟩

lemma *Klein_J_aux_path_left:*
*"x ≤ 0 ⟹ Klein_J_aux_path x = -1/2 + (sqrt 3 / 2 - x) *$_R$ i"*
⟨*proof*⟩

lemma *Klein_J_aux_path_middle:*
*"x ∈ {0..1} ⟹ Klein_J_aux_path x = cis (2 * pi / 3 - x * pi / 6)"*
⟨*proof*⟩

lemma *Klein_J_aux_path_right:*
*"x ≥ 1 ⟹ Klein_J_aux_path x = x *$_R$ i"*

⟨*proof*⟩

lemma *Klein_J_aux_path_inj: "inj Klein_J_aux_path"*
⟨*proof*⟩

lemma *Klein_J_aux_path_range: "range Klein_J_aux_path ⊆ $\mathcal{R}_\Gamma$'"*
⟨*proof*⟩

definition *Klein_J_aux_path' :: "real ⇒ real"* **where**
"Klein_J_aux_path' = Re ∘ Klein_J ∘ Klein_J_aux_path"

lemma *continuous_Klein_J_aux_path': "continuous_on A Klein_J_aux_path'"*
⟨*proof*⟩

lemma *Klein_J_Klein_J_aux_path_in_Reals: "Klein_J (Klein_J_aux_path x) ∈ ℝ"*
⟨*proof*⟩

lemma *inj_on_Re [intro]: "A ⊆ ℝ ⟹ inj_on Re A"*
⟨*proof*⟩

lemma *Klein_J_aux_path'_inj: "inj Klein_J_aux_path'"*
⟨*proof*⟩

lemma *Klein_J_aux_path'_strict_mono: "strict_mono Klein_J_aux_path'"*
⟨*proof*⟩

lemma *Re_Klein_J_left_vertical_less:*
assumes *"Re x = -1 / 2" "Re y = - 1 / 2" "sqrt 3 / 2 ≤ Im x" "Im x < Im y"*
shows *"Re (Klein_J x) > Re (Klein_J y)"*
⟨*proof*⟩

lemma *Re_Klein_J_left_vertical_le:*
assumes *"Re x = -1 / 2" "Re y = - 1 / 2" "sqrt 3 / 2 ≤ Im x" "Im x ≤ Im y"*
shows *"Re (Klein_J x) ≥ Re (Klein_J y)"*
⟨*proof*⟩

lemma *Re_Klein_J_arc_less:*
assumes *"pi / 2 ≤ x" "x < y" "y ≤ 2 * pi / 3"*
shows *"Re (Klein_J (cis x)) > Re (Klein_J (cis y))"*
⟨*proof*⟩

lemma *Re_Klein_J_arc_le:*
assumes *"pi / 2 ≤ x" "x ≤ y" "y ≤ 2 * pi / 3"*
shows *"Re (Klein_J (cis x)) ≥ Re (Klein_J (cis y))"*
⟨*proof*⟩

lemma *Re_Klein_J_arc_less':*
assumes *"norm x = 1" "norm y = 1" "pi / 2 ≤ Arg x" "Arg x < Arg y" "Arg y ≤ 2 * pi / 3"*
shows *"Re (Klein_J x) > Re (Klein_J y)"*
⟨*proof*⟩

lemma *Re_Klein_J_arc_le':*
assumes *"norm x = 1" "norm y = 1" "pi / 2 ≤ Arg x" "Arg x ≤ Arg y" "Arg y ≤ 2 * pi / 3"*
shows *"Re (Klein_J x) ≥ Re (Klein_J y)"*
⟨*proof*⟩

lemma *Re_Klein_J_imag_axis_less:*
assumes *"Re x = 0" "Re y = 0" "1 ≤ Im x" "Im x < Im y"*
shows *"Re (Klein_J x) < Re (Klein_J y)"*
⟨*proof*⟩

lemma *Re_Klein_J_imag_axis_le:*
assumes *"Re x = 0" "Re y = 0" "1 ≤ Im x" "Im x ≤ Im y"*
shows *"Re (Klein_J x) ≤ Re (Klein_J y)"*
⟨*proof*⟩

lemma *Re_Klein_J_neg:*
"Re x = -1 / 2 ⟹ Im x > sqrt 3 / 2 ⟹ Re (Klein_J x) < 0"
⟨*proof*⟩

lemma *Re_Klein_J_nonpos:*
"Re x = -1 / 2 ⟹ Im x ≥ sqrt 3 / 2 ⟹ Re (Klein_J x) ≤ 0"
⟨*proof*⟩

lemma *Re_Klein_J_gt_1:*
"Re x = 0 ⟹ Im x > 1 ⟹ Re (Klein_J x) > 1"
⟨*proof*⟩

lemma *Re_Klein_J_ge_1:*
"Re x = 0 ⟹ Im x ≥ 1 ⟹ Re (Klein_J x) ≥ 1"
⟨*proof*⟩

lemma *Re_Klein_J_pos:*
*"x ∈ {pi/2..<2/3*pi} ⟹ Re (Klein_J (cis x)) > 0"*
⟨*proof*⟩

lemma *Re_Klein_J_pos':*
*"norm x = 1 ⟹ Arg x ∈ {pi/2..<2/3*pi} ⟹ Re (Klein_J x) > 0"*
⟨*proof*⟩

lemma *Re_Klein_J_nonneg:*
assumes *"x ∈ {pi/2..2/3*pi}"*

shows *"Re (Klein_J (cis x)) ∈ {0..1}"*
⟨*proof*⟩

lemma *Re_Klein_J_nonneg':*
*"norm x = 1 ⟹ Arg x ∈ {pi/2..2/3*pi} ⟹ Re (Klein_J x) ≥ 0"*
⟨*proof*⟩

lemma *Re_Klein_J_less_1:*
*"x ∈ {pi/2<..2/3*pi} ⟹ Re (Klein_J (cis x)) < 1"*
⟨*proof*⟩

lemma *Re_Klein_J_less_1':*
*"norm x = 1 ⟹ Arg x ∈ {pi/2<..2/3*pi} ⟹ Re (Klein_J x) < 1"*
⟨*proof*⟩

lemma *filterlim_Klein_J_at_ii_inf: "filterlim Klein_J at_infinity at_i∞"*
⟨*proof*⟩

lemma *filterlim_Re_Klein_J_at_bot:*
*"filterlim (λx. Re (Klein_J (-1/2 + x *$_R$* i))) at_bot at_top"*
⟨*proof*⟩

lemma *filterlim_Re_Klein_J_at_top:*
*"filterlim (λx. Re (Klein_J (x *$_R$* i))) at_top at_top"*
⟨*proof*⟩

lemma *Klein_J_on_arc: "(Klein_J ∘ cis) ‘ {pi/2..2/3*pi} = of_real ‘ {0..1}"*
⟨*proof*⟩

lemma *Klein_J_on_left_vline:*
*"(λx. Klein_J (-1/2 + x *$_R$* i)) ‘ {sqrt 3/2..} = of_real ‘ {..0}"*
⟨*proof*⟩

lemma *Klein_J_on_imag_axis:*
*"(λx. Klein_J (x *$_R$* i)) ‘ {1..} = of_real ‘ {1..}"*
⟨*proof*⟩

lemma *Klein_J_on_border: "Klein_J ‘ (*$\mathcal{R}_\Gamma$*' - *$\mathcal{R}_\Gamma$*) = of_real ‘ {..1}"*
⟨*proof*⟩

7.5 Inverse function

This is the branch of the inverse function to Klein's *J* function whose codomain is $\mathcal{R}_\Gamma$'. It has a branch cut on the slit *(-∞, 1]*.

definition *Klein_J_inv :: "complex ⇒ complex"* **where**
"Klein_J_inv = the_inv_into $\mathcal{R}_\Gamma$*' Klein_J"*

lemma *Klein_J_inv_in_std_fund_region' [intro]: "Klein_J_inv z ∈* $\mathcal{R}_\Gamma$*'"*
⟨*proof*⟩

lemma *Im_Klein_J_inv_pos [intro]: "Im (Klein_J_inv z) > 0"*
⟨*proof*⟩

lemma *Klein_J_Klein_J_inv [simp]: "Klein_J (Klein_J_inv z) = z"*
⟨*proof*⟩

lemma *Klein_J_inv_Klein_J [simp]: "z ∈ $\mathcal{R}_\Gamma$' ⟹ Klein_J_inv (Klein_J z) = z"*
⟨*proof*⟩

lemma *Klein_J_inv_eq_iff: "Klein_J_inv z = u ⟷ Klein_J u = z ∧ u ∈ $\mathcal{R}_\Gamma$'"*
⟨*proof*⟩

lemma *Klein_J_inv_not_real [simp]: "Klein_J_inv z ∉ ℝ"*
⟨*proof*⟩

lemma *Klein_J_inv_eqI: "Klein_J u = z ⟹ u ∈ $\mathcal{R}_\Gamma$' ⟹ Klein_J_inv z = u"*
⟨*proof*⟩

lemma *Klein_J_inv_eq_i_iff [simp]: "Klein_J_inv z = i ⟷ z = 1"*
⟨*proof*⟩

lemma *Klein_J_inv_eq_rho_iff [simp]: "Klein_J_inv z = ϱ ⟷ z = 0"*
⟨*proof*⟩

lemma *Klein_J_inv_equiv_iff [simp]:*
assumes *"Im x > 0"*
shows *"Klein_J_inv z ∼$_\Gamma$ x ⟷ Klein_J x = z"*
⟨*proof*⟩

lemma *Klein_J_inv_equiv_iff' [simp]:*
assumes *"Im x > 0"*
shows *"x ∼$_\Gamma$ Klein_J_inv z ⟷ Klein_J x = z"*
⟨*proof*⟩

lemma *Klein_J_inv_has_field_derivative:*
assumes *z: "z ∉ of_real ` {..1}"*
shows *"(Klein_J_inv has_field_derivative inverse (deriv Klein_J (Klein_J_inv z))) (at z)"*
⟨*proof*⟩

7.6 Covering properties

Next, we show a topological fact that Apostol implicitly uses to prove Picard's little theorem without mentioning it: We view J as a map from the upper half plane minus the elliptic points (i.e. the orbits of i and ρ) to

$\mathbb{C} \setminus \{0, 1\}$. This map is then a *covering*, i.e. it maps infinitely many copies of the fundamental region to $\mathbb{C} \setminus \{0, 1\}$.

Some additional facts about coverings would probably make this proof less bulky. Most textbook also derive Picard's little theorem using the modular lamdba function instead, whose derivative does not vanish, which means that no points have to be removed and the topology becomes easier to manage.

theorem *covering_map_Klein_J:*
  defines *"A ≡ {z. Im z > 0} - modular_group.orbit ϱ - modular_group.orbit i"*
  shows *"open A" "covering_space A Klein_J (-{0,1})"*
⟨*proof*⟩

7.7 Applications

7.8 The Eisenstein inversion problem

We now prove Apostol's Theorem 2.9: The Eisenstein inversion problem.

It states that for any numbers $a_2, a_3 \in \mathbb{C}$ with $a_2^3 - 27a_3^2 \neq 0$ there is a lattice whose g_2 and g_3 invariant have exactly the values a_2 and a_3.

The broader significance of this is that it relates elliptic curves to complex lattices: in our earlier AFP entry on complex lattices and elliptic functions, we formalised the fact that for any complex lattice Λ with invariants g_2 and g_3, the map $z \mapsto (\wp(z), \wp'(z))$ is a group isomorphism between the additive group of $\mathbb{C}/\Lambda$ (a complex torus) and the elliptic curve $y^2 = 4x^3 - g_2 x - g_3$.

The present theorem now shows that the reverse also holds, since for any elliptic curve in the form $y^2 = 4x^3 - ax - b$ with non-zero discriminant $a^3 - 27b^2$, a lattice can be found such that $a = g_2$ and $b = g_3$.

theorem *eisenstein_series_inversion:*
  fixes *a2 a3 :: complex*
  assumes *discr: "a2 ^ 3 - 27 * a3 ^ 2 ≠ 0"*
  obtains *ω1 ω2* **where**
    "Im (ω1 / ω2) ≠ 0"
    "complex_lattice.invariant_g2 ω2 ω1 = a2"
    "complex_lattice.invariant_g3 ω2 ω1 = a3"
⟨*proof*⟩

7.9 A short proof of Picard's Little Theorem

Proving Picard's Little Theorem using our results on Klein's J function is Apostol's Theorem 2.10. We already have this result in the library, but we re-prove it here again anyway to illustrate how simple the proof is.

lemma *little_Picard_01_via_Klein_J:*
  fixes *g :: "complex ⇒ complex"*
  assumes *g: "g holomorphic_on UNIV" "g ∈ UNIV → -{0, 1}"*
  shows *"g constant_on UNIV"*

⟨*proof*⟩

lemma *little_Picard:*
  fixes *g :: "complex ⇒ complex"*
  assumes *g: "g holomorphic_on UNIV" "g ∈ UNIV → -{a,b}" "a ≠ b"*
  shows *"g constant_on UNIV"*
⟨*proof*⟩

end

8 The Serre derivative

theory *Serre_Derivative*
  imports *Basic_Modular_Forms_Mero_UHP "Elliptic_Functions.Eisenstein_G2"*
begin

The derivative of a non-constant level 1 modular form f is *not* a modular form due to the way the derivative operator interacts with the slash operator. However, the Serre derivative $\partial_k f = \frac{1}{2i\pi} f' - \frac{k}{12} E_2 f$ *is* a modular form again, since the extra term produced by the "defect" of f' under the slash operator perfectly cancels with the defect of E_2 under the slash operator. The same also holds for quasimodular forms.

First, we define an auxiliary notion. It is not clear to us whether there is a standard name for this, other than the logarithmic derivative of the automorphy factor $j(z) = cz + d$.
The significance is that the "defect" of quasimodular functions under the slash operator is a polynomial of this function, so we call it "defect".

definition *defect_modgrp :: "modgrp ⇒ mero_uhp"*
  where *"defect_modgrp h = of_int (modgrp_c h) / automorphy_factor_mero_uhp h"*

lemma *holo_uhp_defect_modgrp: "holo_uhp (defect_modgrp h)"*
  ⟨*proof*⟩

lemma *defect_modgrp_1 [simp]: "defect_modgrp 1 = 0"*
  ⟨*proof*⟩

lemma *defect_modgrp_T [simp]: "defect_modgrp T_modgrp = 0"*
  ⟨*proof*⟩

lemma *defect_modgrp_shift [simp]: "defect_modgrp (shift_modgrp n) = 0"*
  ⟨*proof*⟩

lemma *defect_modgrp_as_logderiv:*
  fixes *h :: modgrp*
  defines *"j ≡ automorphy_factor_mero_uhp h"*

shows *"defect_modgrp h = deriv_mero_uhp j / j"*
⟨*proof*⟩

lemma *defect_modgrp_mult:*
*"defect_modgrp (g * h) = slash_mero_uhp 2 h (defect_modgrp g) + defect_modgrp h"*
⟨*proof*⟩

lemma *eval_mero_uhp_defect:*
assumes *"Im z > 0"*
shows *"eval_mero_uhp (defect_modgrp h) z = of_int (modgrp_c h) / automorphy_factor h z"*
⟨*proof*⟩

lemma *deriv_mero_uhp_defect [simp]: "deriv_mero_uhp (defect_modgrp h) = -(defect_modgrp h ^ 2)"*
⟨*proof*⟩

lemma *slash_mero_uhp_Eisenstein_E2:*
*"slash_mero_uhp 2 h (ℰ 2) = ℰ 2 - ⟨6 * i / of_real pi⟩ * defect_modgrp h"*
⟨*proof*⟩

definition *serre_deriv :: "int ⇒ mero_uhp ⇒ mero_uhp"* **where**
*"serre_deriv k f = ⟨1/(2*i*pi)⟩ * deriv_mero_uhp f - ⟨of_int k / 12⟩ * ℰ 2 * f"*

lemma *mero_uhp_rel_serre_deriv [mero_uhp_rel_intros]:*
*"mero_uhp_rel (serre_deriv k f) (λz. deriv f z / (2*i*pi) - of_int k / 12 * Eisenstein_E 2 z * f z)"*
⟨*proof*⟩

The Serre derivative satisfies all the usual laws of a derivative operator:

lemma *serre_deriv_0 [simp]: "serre_deriv k 0 = 0"*
⟨*proof*⟩

lemma *serre_deriv_1 [simp]: "serre_deriv 0 1 = 0"*
⟨*proof*⟩

lemma *serre_deriv_const [simp]: "serre_deriv 0 (const_mero_uhp c) = 0"*
⟨*proof*⟩

lemma *serre_deriv_minus [simp]: "serre_deriv k (-f) = -serre_deriv k f"*
⟨*proof*⟩

lemma *serre_deriv_add [simp]: "serre_deriv k (f + g) = serre_deriv k f + serre_deriv k g"*

⟨*proof*⟩

lemma *serre_deriv_diff [simp]: "serre_deriv k (f - g) = serre_deriv k f - serre_deriv k g"*
⟨*proof*⟩

lemma *serre_deriv_mult:*
*"serre_deriv (k1 + k2) (f * g) = serre_deriv k1 f * g + f * serre_deriv k2 g"*
⟨*proof*⟩

lemma *serre_deriv_power: "serre_deriv (k * n) (f ^ n) = of_nat n * serre_deriv k f * f ^ (n - 1)"*
⟨*proof*⟩

lemma *serre_deriv_inverse: "serre_deriv k (inverse f) = -serre_deriv (-k) f / f ^ 2"*
⟨*proof*⟩

lemma *serre_deriv_divide:*
*"serre_deriv (k1 - k2) (f / g) = (g * serre_deriv k1 f - f * serre_deriv k2 g) / g ^ 2"*
⟨*proof*⟩

lemma *serre_deriv_power_int:*
*"serre_deriv (k * n) (f powi n) = of_int n * serre_deriv k f * f powi (n-1)"*
⟨*proof*⟩

lemma *serre_deriv_defect:*
"serre_deriv k (defect_modgrp h) =
*-⟨1 / (2*i*pi)⟩ * defect_modgrp h ^ 2 - ⟨of_int k / 12⟩ * ℰ 2 * defect_modgrp h"*
⟨*proof*⟩

The Serre derivative commutes with the slash operator. This is the crucial fact that implies that the Serre derivative maps modular forms to modular forms.

lemma *serre_deriv_slash [simp]:*
"serre_deriv k (slash_mero_uhp k h f) = slash_mero_uhp (k+2) h (serre_deriv k f)"
⟨*proof*⟩

lemma *deriv_mero_uhp_slash:*
"deriv_mero_uhp (slash_mero_uhp weight h f) =
slash_mero_uhp (weight + 2) h (deriv_mero_uhp f) -
*of_int weight * defect_modgrp h * slash_mero_uhp weight h f"*
⟨*proof*⟩

The Serre derivative preserves weakly meromorphic forms, meromorphic forms, and modular forms.

lemma (**in** *weakly_meromorphic_form*) *weakly_meromorphic_form_serre_deriv:*
  "weakly_meromorphic_form (serre_deriv weight f) (weight + 2) G"
⟨*proof*⟩

lemma (**in** *meromorphic_form*) *meromorphic_form_serre_deriv:*
  "meromorphic_form (serre_deriv weight f) (weight + 2) G"
⟨*proof*⟩

lemma (**in** *modular_form*) *modular_form_serre_deriv:*
  "modular_form (serre_deriv weight f) (weight + 2) G"
⟨*proof*⟩

lemma *serre_deriv_in_WMForms [mform_intros]:*
  assumes *"f ∈ WMForms[G, k]" "k' = k + 2"*
  shows *"serre_deriv k f ∈ WMForms[G, k']"*
  ⟨*proof*⟩

lemma *serre_deriv_in_MeForms [mform_intros]:*
  assumes *"f ∈ MeForms[G, k]" "k' = k + 2"*
  shows *"serre_deriv k f ∈ MeForms[G, k']"*
  ⟨*proof*⟩

lemma *serre_deriv_in_MForms [mform_intros]:*
  assumes *"f ∈ MForms[G, k]" "k' = k + 2"*
  shows *"serre_deriv k f ∈ MForms[G, k']"*
  ⟨*proof*⟩

end

9 The graded ring structure of level 1 modular forms

theory *Modular_Forms_Structure*
imports
  Meromorphic_Forms_Valence_Formula
  Basic_Modular_Forms_Mero_UHP
  "Elliptic_Functions.Dedekind_Eta"
begin

In this section, we will use the valence formula for the full modular group $\mathrm{SL}(2,\mathbb{Z})$ to analyse the structure of the vector space of modular forms on it.

9.1 Auxiliary material

9.1.1 Linear diophantine inequalities in two variables

We first need some simple number-theoretic facts related to linear diophantine equations with two variables, i.e. $ax + by = l$, where a, b, l are integer parameters with a and b not both 0 and x and y are integer variables.

lemma *bezout_imp_coprime:*
  assumes *"a * u + b * v = 1"*
  shows *"Rings.coprime a b"*
  ⟨*proof*⟩

The solutions to the equation $ax + by = l$ are freely generated as $(x, y) = (u + b/dk, v - a/dk)$ with k ranging over the integers, where $d = \gcd(a, b)$ and (u, v) is an arbitrary particular solution of the equation.
Due to Bézout's theorems, such a particular solution exists iff $d \mid l$, but we do not show that here.

lemma *gen_bezout_solutions:*
  fixes *u v a b :: "'a :: ring_gcd"*
  assumes *"a * u + b * v = l"* **and** *nz: "a ≠ 0 ∨ b ≠ 0"*
  shows *"bij_betw (λk. (u + b div gcd a b * k, v - a div gcd a b * k)) UNIV*
*{(x,y). a * x + b * y = l}"*
⟨*proof*⟩

9.1.2 Independent families of vectors and indexed bases

We define independent families of vectors and indexed bases, which is more convenient for our purposes than looking at a basis only as a set of vectors.

lemma (**in** *vector_space*) *dim_ge_card_imp_independent:*
  assumes *"finite X" "dim X ≥ card X"*
  shows *"independent X"*
⟨*proof*⟩

lemma (**in** *vector_space*) *dim_trivial_eq_0 [simp]: "dim {0} = 0"*
⟨*proof*⟩

locale *vector_space_independent_family = vector_space +*
  fixes *f :: "'c ⇒ 'b"* **and** *I :: "'c set"*
  assumes *inj: "inj_on f I"*
  assumes *independent: "independent (f ‘ I)"*

locale *vector_space_indexed_basis = vector_space_independent_family +*
  fixes *X :: "'b set"*
  assumes *span: "span (f ‘ I) = X"*

abbreviation (**in** *vector_space*) *independent_family* :: "('c ⇒ 'b) ⇒ 'c set ⇒ bool" **where**
  "independent_family ≡ vector_space_independent_family scale"

abbreviation (**in** *vector_space*) *indexed_basis* :: "('c ⇒ 'b) ⇒ 'c set ⇒ 'b set ⇒ bool" **where**
  "indexed_basis ≡ vector_space_indexed_basis scale"

lemma (**in** *vector_space*) *independent_familyI*:
  assumes "⋀J c. J ⊆ I ⟹ finite J ⟹ (∑x∈J. scale (c x) (f x)) = 0 ⟹ (∀x∈J. c x = 0)"
  shows "independent_family f I"
⟨*proof*⟩

lemma (**in** *vector_space*) *independent_familyI_finite*:
  assumes "⋀c. (∑x∈I. scale (c x) (f x)) = 0 ⟹ (∀x∈I. c x = 0)"
  assumes "finite I"
  shows "local.independent_family f I"
⟨*proof*⟩

lemma (**in** *vector_space_independent_family*) *representation_0_iff*:
  assumes "J ⊆ I" "finite J"
  shows "(∑x∈J. scale (c x) (f x)) = 0 ⟷ (∀x∈J. c x = 0)"
⟨*proof*⟩

lemma (**in** *vector_space_independent_family*) *independent_family_subset*:
  assumes "J ⊆ I"
  shows "independent_family f J"
⟨*proof*⟩

lemma (**in** *vector_space_independent_family*) *indexed_basisI*:
  assumes "subspace X"
  assumes "⋀i. i ∈ I ⟹ f i ∈ X"
  assumes "⋀x. x ∈ X ⟹ ∃J c. finite J ∧ J ⊆ I ∧ x = (∑y∈J. scale (c y) (f y))"
  shows "indexed_basis f I X"
⟨*proof*⟩

lemma (**in** *vector_space_independent_family*) *indexed_basisI_finite*:
  assumes "subspace X" "finite I"
  assumes "⋀i. i ∈ I ⟹ f i ∈ X"
  assumes "⋀x. x ∈ X ⟹ ∃c. x = (∑y∈I. scale (c y) (f y))"
  shows "indexed_basis f I X"
⟨*proof*⟩

lemma (**in** *vector_space_indexed_basis*) *indexed_basis_imp_representation*:
  assumes "x ∈ X"
  obtains J c **where** "J ⊆ I" "finite J" "x = (∑y∈J. scale (c y) (f y))"
⟨*proof*⟩

lemma (**in** *vector_space_indexed_basis*) *indexed_basis_imp_representation_finite:*
  assumes *"x ∈ X" "finite I"*
  obtains *c* **where** *"x = (∑y∈I. scale (c y) (f y))"*
⟨*proof*⟩

lemma (**in** *vector_space_indexed_basis*) *indexed_basis_imp_dim:*
  assumes *"finite I"*
  shows *"dim X = card I"*
⟨*proof*⟩

9.2 Basic structure lemmas

We first derive a few facts about the structure of low-weight modular forms on $\mathrm{SL}(2,\mathbb{Z})$, all of which follow directly from the valence formula.

unbundle *modgrp_notation*

lemmas (**in** *cong_subgroup*) *[intro] = cong_subgroup_axioms*

interpretation *modgrp: cong_subgroup UNIV*
  ⟨*proof*⟩

The following four theorems constitute Apostol's Theorem 6.2.

First, the modular forms of weight 0 are exactly the constant functions.

theorem *MForms_0_eq_constant: "MForms[0] = range const_mero_uhp"*
⟨*proof*⟩

The weight of a non-zero modular form must be even and at least 4.

theorem *MForms_eq_0:*
  assumes *"k < 0 ∨ k = 2 ∨ odd k"*
  shows *"MForms[k] = {0}"*
⟨*proof*⟩

lemma *MForms_weight_even: "f ∈ MForms[k] - {0} ⟹ even k"*
  ⟨*proof*⟩

theorem *MForms_weight_ge_4:*
  assumes *"f ∈ MForms[k]" "¬is_const_mero_uhp f"*
  shows *"k ≥ 4"*
⟨*proof*⟩

The following lemma lists the zeros of all non-zero modular forms up to weight 15 (except weight 12), including multiplicities.

lemma *zeros_MForms_upto_15:*
  assumes *f [mform_intros]: "f ∈ MForms[k]"* **and** *k: "k ∈ {1..15} - {12}"*
  assumes *[simp]: "f ≠ 0"*
  shows *"Im z > 0 ⟹ ¬z ∼Γ ϱ ⟹ ¬z ∼Γ* i *⟹ eval_mero_uhp f z ≠ 0"*

and *"z ∼Γ ϱ ⟹ zorder_mero_uhp f z = (if k = 6 then 0 else if k ∈ {8, 14} then 2 else 1)"*
and *"z ∼Γ i ⟹ zorder_mero_uhp f z = (if k ∈ {4, 8} then 0 else 1)"*
⟨*proof*⟩

The zeros of G_4 and G_6 are exactly the points equivalent to i and ρ, respectively, and they have multiplicity 1 (in the normal complex analysis sense, not factoring in the elliptic order).

lemma *Eisenstein_E_4_zero_iff [simp]:*
assumes *"Im z > 0"*
shows *"Eisenstein_E 4 z = 0 ⟷ z ∼Γ ϱ"*
⟨*proof*⟩

lemma *Eisenstein_E_6_zero_iff [simp]:*
assumes *"Im z > 0"*
shows *"Eisenstein_E 6 z = 0 ⟷ z ∼Γ i"*
⟨*proof*⟩

lemma *zorder_Eisenstein_E4_rho: "zorder ℰ4 ϱ = 1"*
⟨*proof*⟩

lemma *zorder_Eisenstein_E6_ii: "zorder ℰ6 i = 1"*
⟨*proof*⟩

lemma *MForms_zorder_ge_imp_same:*
assumes *"f ∈ MForms[k]" "g ∈ MForms[k]" "g ≠ 0"*
assumes *"⋀z. f ≠ 0 ⟹ Im z > 0 ⟹ zorder_mero_uhp f z ≥ zorder_mero_uhp g z"*
assumes *"⋀h. f ≠ 0 ⟹ zorder_at_ii_inf 1 f ≥ zorder_at_ii_inf 1 g"*
shows *"∃c. f = ⟨c⟩ * g"*
⟨*proof*⟩

Any modular form of weight $k \leq 15$ and $k \neq 12$ is a constant multiple of G_k.

lemma *MForms_upto_15:*
assumes *f: "f ∈ MForms[k]"* **and** *"k ∈ {0..15} - {12}"*
shows *"∃c. f = ⟨c⟩ * ℰ (nat k)"*
⟨*proof*⟩

There are no non-zero cusp forms of weight below 12.

theorem *CForms_eq_0_weak:*
assumes *"k < 12"*
shows *"CForms[k] = {0}"*
⟨*proof*⟩

Every cusp form of level k can be written as a product of Δ and a modular form of level $k - 12$.

lemma *CForms_split:*
  assumes *"f ∈ CForms[k]"*
  obtains *g* **where** *"g ∈ MForms[k-12]" "f = g * Δ"*
⟨*proof*⟩

lemma *CForms_eq_0:*
  assumes *"k < 12 ∨ odd k ∨ k = 14"*
  shows *"CForms[k] = {0}"*
  ⟨*proof*⟩

hide_fact *CForms_eq_0_weak*

All cusp forms of weight 12 are multiples of the modular discriminant.

lemma *CForms_12:*
  assumes *"f ∈ CForms[12]"*
  obtains *c* **where** *"f = ⟨c⟩ * Δ"*
  ⟨*proof*⟩

9.3 The standard basis $E_{k-12i}\Delta^i$

The vector space of modular forms of weight k (for even $k \geq 0$) is spanned by basis elements of the form $E_{k-12i}\Delta^i$, where $12i \leq k$ and $k - 12i \neq 2$. This also directly leads to the dimension formula: the dimension of the space is $\lfloor k/12 \rfloor$ if $k = 2 \pmod{12}$ and $\lfloor k/12 \rfloor + 1$ otherwise.

The basis element for $i = 0$ is the "Eisenstein part" and the parts for $i > 0$ span the space of cusp forms.

locale *modular_forms_UNIV_std_basis =*
  fixes *k :: int*
  assumes *even_weight: "even k"*
  assumes *nonneg_weight: "k ≥ 0"*
begin

definition *basis :: "nat ⇒ mero_uhp"* **where** *"basis r = ℰ (nat k - 12 * r) * Δ ^ r"*
definition *idxs :: "nat set"* **where** *"idxs = {r. 12 * int r ≤ k ∧ k - 12 * int r ≠ 2}"*

lemma *finite_idxs [intro]: "finite idxs"*
⟨*proof*⟩

lemma *idxs_altdef: "idxs = (if [k = 2] (mod 12) then {..<nat k div 12} else {..nat k div 12})"*
⟨*proof*⟩

lemma *card_idxs: "card idxs = nat k div 12 + (if [k = 2] (mod 12) then 0 else 1)"*
  ⟨*proof*⟩

sublocale *vector_space_independent_family "λc f. ⟨c⟩ * f" basis idxs*
⟨*proof*⟩

sublocale *vector_space_indexed_basis "λc f. ⟨c⟩ * f" basis idxs "MForms[k]"*
⟨*proof*⟩

lemma *dim_eq: "mero_uhp.dim MForms[k] = nat k div 12 + (if [k = 2] (mod 12) then 0 else 1)"*
⟨*proof*⟩

end

theorem *dim_MForms_UNIV:*
"mero_uhp.dim MForms[k] =
(if even k ∧ k ≥ 0 then nat k div 12 + (if [k = 2] (mod 12) then 0 else 1) else 0)"
⟨*proof*⟩

9.4 The alternative basis $E_4^i E_6^j$

An alternative basis for the modular forms of weight k (for even $k \geq 0$) is given by the elements of the form $E_4^i E_6^j$ with $4i + 6j = k$.

locale *modular_forms_UNIV_E46_basis =*
fixes *k :: int*
assumes *even_weight: "even k"*
assumes *nonneg_weight: "k ≥ 0"*
begin

definition *basis :: "nat × nat ⇒ mero_uhp"* **where** *"basis = (λ(i,j). ℰ 4 ^ i * ℰ 6 ^ j)"*
definition *idxs :: "(nat × nat) set"* **where** *"idxs = {(i,j). 4 * i + 6 * j = k}"*

lemma *finite_idxs [intro]: "finite idxs"*
⟨*proof*⟩

lemma *card_idxs: "card idxs = nat k div 12 + (if [k = 2] (mod 12) then 0 else 1)"*
⟨*proof*⟩

sublocale *vector_space_indexed_basis "λc f. ⟨c⟩ * f" basis idxs "MForms[k]"*
⟨*proof*⟩

end

9.5 A basis for cusp forms

context *modular_forms_UNIV_std_basis*

begin

sublocale *cusp: vector_space_independent_family "λc f. ⟨c⟩ * f" basis "idxs - {0}"*
⟨*proof*⟩

sublocale *cusp: vector_space_indexed_basis "λc f. ⟨c⟩ * f" basis "idxs - {0}" "CForms[k]"*
⟨*proof*⟩

lemma *dim_CForms_eq:*
"mero_uhp.dim CForms[k] = mero_uhp.dim MForms[k] - 1"
⟨*proof*⟩

end

The dimension of cusp forms is one less than that of modular forms. Note due to behaviour of Isabelle's subtraction on natural numbers, there is a subtlety hidden here: when the dimension of modular forms is already 0 (i.e. for k odd, $k < 0$, or $k = 2$), the dimension of cusp forms is of course also 0.

theorem *dim_CForms_UNIV:*
"mero_uhp.dim CForms[k] = mero_uhp.dim MForms[k] - 1"
⟨*proof*⟩

9.6 Connection to Dedekind's η function

We already have the transformation laws for Dedekind's eta function, namely $\eta(z+1) = e^{i\pi/12}\eta(z)$ and $\eta(-1/z) = \sqrt{-iz}\eta(z)$. This corresponds to a kind of "weight $\frac{1}{2}$" transformation, including a 24-th root of unity.

Raising η to the 24-th power clears these roots of unity, giving us a modular form of weight 12. The Fourier expansion of η^{24} is, by definition, $q\varphi(q)^{24}$, where φ denotes the Euler function. This means that η^{24} is a cusp form.

Since all the space of cusp forms of weight 12 is one-dimensional, there must be some constant c such that $\eta^{24} = c\Delta$. This constant can easily be determined to be $c = (2\pi)^{-12}$ by comparing the coefficients of the q^1 term in the Fourier expansions of η^{24} and Δ.

theorem *modular_discr_conv_dedekind_eta:*
assumes *"Im z > 0"*
shows *"modular_discr z = (2 * pi) ^ 12 * dedekind_eta z ^ 24"*
⟨*proof*⟩

end

10 Zagier's theory of quasimodular forms

theory *Quasimodular_Forms*

imports *Serre_Derivative Modular_Forms_Structure*
begin

In this section, we define quasimodular forms in the style of Zagier, following the presentation by Royer [2]. Like Royer, we only define level 1 quasimodular forms since higher levels are somewhat tedious and we have no immediate concrete applications for them that would justify the effort.

The concrete motivation for level 1 quasimodular forms is that they include the 'forbidden' Eisenstein series E_2 and are closed under the Serre derivative, which in particular gives rise to a Ramanujan-style identity involving E_2, as we will see later.

lemma *at_ii_inf_le_at_infinity: "at_i∞ ≤ at_infinity"*
⟨*proof*⟩

lemma *(in cong_subgroup) cusp_width_conj_pos [intro]: "cusp_width∞ (conj_modgrp h G) > 0"*
⟨*proof*⟩

10.1 Quasimodular functions

lemma *(in cong_subgroup) infinite_image_defect_modgrp: "infinite (defect_modgrp ‘ G)"*
⟨*proof*⟩

locale *quasimodular_function_explicit = cong_subgroup G*
  for *f :: mero_uhp* **and** *weight :: int* **and** *G :: "modgrp set"* **and** *p :: "mero_uhp poly" +*
  assumes *holo_uhp_coeffs: "⋀k. holo_uhp (poly.coeff p k)"*
  assumes *slash: "⋀h. h ∈ G ⟹ slash_mero_uhp weight h f = poly p (defect_modgrp h)"*
begin

lemma *coeff_0 [simp]: "poly.coeff p 0 = f"*
  ⟨*proof*⟩

lemma *poly_p_0 [simp]: "poly p 0 = f"*
  ⟨*proof*⟩

lemma *holo_uhp: "holo_uhp f"*
  ⟨*proof*⟩

sublocale *fourier_expansion_locale "cusp_width∞ G" f*
⟨*proof*⟩

end

locale *quasimodular_function_explicit_UNIV =*
  fixes *f :: mero_uhp* **and** *weight :: int* **and** *p :: "mero_uhp poly"*
  assumes *holo_uhp_coeffs_UNIV: "⋀k. holo_uhp (poly.coeff p k)"*
  assumes *slash_UNIV: "⋀h. slash_mero_uhp weight h f = poly p (defect_modgrp h)"*
begin

sublocale *quasimodular_function_explicit f weight UNIV p*
  rewrites *"cusp_width$_\infty$ UNIV ≡ Suc 0"*
  ⟨*proof*⟩

end

lemma (**in** *quasimodular_function_explicit*) *quasimodular_function_explicit_imp_UNIV:*
  assumes *"G = UNIV"*
  shows *"quasimodular_function_explicit_UNIV f weight p"*
  ⟨*proof*⟩

lemma *quasimodular_function_explicit_unique:*
  assumes *"quasimodular_function_explicit f weight G p1"*
  assumes *"quasimodular_function_explicit f weight G p2"*
  shows *"p1 = p2"*
⟨*proof*⟩

definition *qmod_poly :: "int ⇒ modgrp set ⇒ mero_uhp ⇒ mero_uhp poly"*
where
  "qmod_poly weight G f = (THE p. quasimodular_function_explicit f weight G p)"

definition *qmod_depth :: "int ⇒ modgrp set ⇒ mero_uhp ⇒ nat"* **where**
  "qmod_depth weight G f = degree (qmod_poly weight G f)"

definition *qmod_coeff_poly_aux :: "nat ⇒ mero_uhp poly ⇒ mero_uhp poly"*
where
  *"qmod_coeff_poly_aux k p = Abs_poly (λi. of_nat ((k+i) choose k) * poly.coeff p (k + i))"*

lemma *coeff_qmod_coeff_poly_aux:*
  *"poly.coeff (qmod_coeff_poly_aux k p) i = of_nat ((k+i) choose k) * poly.coeff p (k + i)"*
  ⟨*proof*⟩

lemma *qmod_coeff_poly_aux_0 [simp]: "qmod_coeff_poly_aux k 0 = 0"*
  ⟨*proof*⟩

lemma *qmod_coeff_poly_aux_eq_0 [simp]: "k > degree p ⟹ qmod_coeff_poly_aux k p = 0"*

⟨*proof*⟩

lemma *qmod_coeff_poly_aux_eq_0_iff: "qmod_coeff_poly_aux k p = 0 ⟷ p = 0 ∨ k > degree p"*
⟨*proof*⟩

lemma *qmod_coeff_poly_aux_degree: "qmod_coeff_poly_aux (degree p) p = [:lead_coeff p:]"*
⟨*proof*⟩

lemma *degree_qmod_coeff_poly_aux [simp]: "degree (qmod_coeff_poly_aux k p) = degree p - k"*
⟨*proof*⟩

E_2 is a quasimodular function of weight 2 and depth 1:

interpretation *Eisenstein_E2: quasimodular_function_explicit_UNIV "ℰ 2" 2 "[:ℰ 2, - ⟨6 * i / of_real pi⟩:]"*
⟨*proof*⟩

Any modular form is quadimodular of depth 0.

lemma (**in** *modular_form*) *quasimodular_function:*
"quasimodular_function_explicit f weight G [:f:]"
⟨*proof*⟩

Constant functions are obviously quasimodular, and the usual closure properties under basic arithmetic also follow.

lemma (**in** *cong_subgroup*) *quasimodular_function_explicit_0:*
"quasimodular_function_explicit 0 k G 0"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *quasimodular_function_explicit_1:*
"quasimodular_function_explicit 1 0 G 1"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *quasimodular_function_explicit_const:*
"quasimodular_function_explicit (const_mero_uhp c) 0 G [:const_mero_uhp c:]"
⟨*proof*⟩

lemma *quasimodular_function_uminus:*
assumes *"quasimodular_function_explicit f weight G pf"*
shows *"quasimodular_function_explicit (-f) weight G (-pf)"*
⟨*proof*⟩

lemma *quasimodular_function_explicit_add:*
assumes *"quasimodular_function_explicit f weight G pf"*
assumes *"quasimodular_function_explicit g weight G pg"*
shows *"quasimodular_function_explicit (f + g) weight G (pf + pg)"*
⟨*proof*⟩

lemma *quasimodular_function_explicit_diff:*
assumes *"quasimodular_function_explicit f weight G pf"*
assumes *"quasimodular_function_explicit g weight G pg"*
shows *"quasimodular_function_explicit (f - g) weight G (pf - pg)"*
⟨*proof*⟩

lemma *quasimodular_function_explicit_mult:*
assumes *"quasimodular_function_explicit f weight1 G pf"*
assumes *"quasimodular_function_explicit g weight2 G pg"*
assumes *"weight = weight1 + weight2"*
shows *"quasimodular_function_explicit (f * g) weight G (pf * pg)"*
⟨*proof*⟩

lemma *quasimodular_function_power:*
assumes *"quasimodular_function_explicit f weight' G pf"*
assumes *"weight = weight' * k"*
shows *"quasimodular_function_explicit (f ^ k) weight G (pf ^ k)"*
⟨*proof*⟩

Crucially. the derivative of a quasimodular form is also a quasimodular form. The depth increases by at most 1.

definition *deriv_qmod_poly :: "int ⇒ mero_uhp poly ⇒ mero_uhp poly"* **where**
"deriv_qmod_poly weight p =
*map_poly deriv_mero_uhp p + Polynomial.monom (of_int weight) 1 * p -*
*Polynomial.monom 1 2 * pderiv p"*

lemma *degree_deriv_qmod_poly_le:*
"degree (deriv_qmod_poly weight p) ≤ degree p + 1"
⟨*proof*⟩

To be more precise, if the original quasimodular form is non-zero and has depth n and weight k, then taking the derivative will increase the depth by 1 unless $k = n$. As we will see later, this is actually impossible.

lemma *quasi_lead_coeff_deriv_qmod_poly:*
"poly.coeff (deriv_qmod_poly weight p) (degree p + 1) = 0 ⟷ p = 0 ∨ weight = int (degree p)"
⟨*proof*⟩

lemma (**in** *quasimodular_function_explicit*) *quasimodular_function_explicit_deriv:*
"quasimodular_function_explicit (deriv_mero_uhp f) (weight + 2) G (deriv_qmod_poly weight p)"
⟨*proof*⟩

The k-th coefficient of the polynomial associated to a quasimodular function of weight w and depth s is a quasimodular function with weight $w - 2k$ and depth $s - k$.

lemma (**in** *quasimodular_function_explicit*) *quasimodular_function_explicit_coeff:*
  *"quasimodular_function_explicit (poly.coeff p k) (weight - 2 * int k)*
G (qmod_coeff_poly_aux k p)"
⟨*proof*⟩

10.2 Quasimodular forms

A quasimodular form is a quasimodular function where all coefficients of the polynomial are additionally holomorphic at all cusps.

locale *quasimodular_form_explicit_UNIV =*
  quasimodular_function_explicit_UNIV f weight p **for** *f weight p +*
  assumes *coeffs_holomorphic_at_cusp: "⋀i. holomorphic_at_infinity (poly.coeff*
p i)"
begin

lemma *fourier_expansion_holomorphic_coeff:*
  "fourier_expansion_holomorphic (Suc 0) (poly.coeff p i)"
⟨*proof*⟩

sublocale *fourier_expansion_holomorphic "Suc 0" f*
  ⟨*proof*⟩

end

locale *modular_form_UNIV = modular_form f weight UNIV* **for** *f weight*

Any modular form is a quasimodular form of depth 0.

lemma (**in** *modular_form_UNIV*) *quasimodular_form_explicit:*
  "quasimodular_form_explicit_UNIV f weight [:f:]"
⟨*proof*⟩

The converse also holds.

lemma (**in** *quasimodular_form_explicit_UNIV*) *depth_0_imp_modular_form:*
  assumes *"degree p = 0"*
  shows *"modular_form f weight UNIV"*
⟨*proof*⟩

The coefficients of a quasimodular form are again quasimodular form. More precisely, if the original form has weight w and depth s, the k-th coefficient has weight $w - 2k$ and depth $s - k$.

lemma (**in** *quasimodular_form_explicit_UNIV*) *quasimodular_form_explicit_UNIV_coeff:*
  *"quasimodular_form_explicit_UNIV (poly.coeff p k) (weight - 2 * int*
k) (qmod_coeff_poly_aux k p)"
⟨*proof*⟩

In particular, the leading coefficient of the polynomial of a quasimodular form of weight w and depth s is an actual modular form of weight $w - 2s$.

lemma (**in** *quasimodular_form_explicit_UNIV*) *modular_form_lead_coeff:*
  *"modular_form (lead_coeff p) (weight - 2 * int (degree p)) UNIV"*
⟨*proof*⟩

lemma (**in** *quasimodular_form_explicit_UNIV*) *depth_le:*
  assumes *"f ≠ 0"*
  shows *"2 * degree p ≤ weight"*
⟨*proof*⟩

lemma (**in** *quasimodular_form_explicit_UNIV*) *degree_deriv_qmod_poly:*
  assumes *"f ≠ 0" "weight ≠ 0"*
  shows *"degree (deriv_qmod_poly weight p) = degree p + 1"*
⟨*proof*⟩

interpretation *Eisenstein_E2: quasimodular_form_explicit_UNIV "ℰ 2" 2 "[:ℰ 2, - ⟨6 * ⅈ / of_real pi⟩:]"*
⟨*proof*⟩

Again, we show that constants are quasimodular forms and that the usual closure properties hold.

lemma (**in** *cong_subgroup*) *quasimodular_form_explicit_UNIV_0:*
  "quasimodular_form_explicit_UNIV 0 weight 0"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *quasimodular_form_explicit_UNIV_const:*
  "quasimodular_form_explicit_UNIV (const_mero_uhp c) 0 [:const_mero_uhp c:]"
⟨*proof*⟩

lemma (**in** *cong_subgroup*) *quasimodular_form_explicit_UNIV_1:*
  "quasimodular_form_explicit_UNIV 1 0 1"
  ⟨*proof*⟩

lemma *quasimodular_form_explicit_UNIV_add:*
  assumes *"quasimodular_form_explicit_UNIV f weight pf"*
  assumes *"quasimodular_form_explicit_UNIV g weight pg"*
  shows *"quasimodular_form_explicit_UNIV (f + g) weight (pf + pg)"*
⟨*proof*⟩

lemma *quasimodular_form_explicit_UNIV_mult:*
  assumes *"quasimodular_form_explicit_UNIV f weight1 pf"*
  assumes *"quasimodular_form_explicit_UNIV g weight2 pg" "weight = weight1 + weight2"*
  shows *"quasimodular_form_explicit_UNIV (f * g) weight (pf * pg)"*
⟨*proof*⟩

lemma *quasimodular_form_explicit_UNIV_uminus:*
  assumes *"quasimodular_form_explicit_UNIV f weight pf"*

shows *"quasimodular_form_explicit_UNIV (-f) weight (-pf)"*
⟨*proof*⟩

lemma *quasimodular_form_explicit_UNIV_diff:*
assumes *"quasimodular_form_explicit_UNIV f weight pf"*
assumes *"quasimodular_form_explicit_UNIV g weight pg"*
shows *"quasimodular_form_explicit_UNIV (f - g) weight (pf - pg)"*
⟨*proof*⟩

lemma *quasimodular_form_explicit_UNIV_power:*
assumes *"quasimodular_form_explicit_UNIV f weight pf"*
shows *"quasimodular_form_explicit_UNIV (f ^ n) (weight * int n) (pf ^ n)"*
⟨*proof*⟩

lemma (**in** *quasimodular_form_explicit_UNIV*) *quasimodular_form_explicit_UNIV_deriv:*
"quasimodular_form_explicit_UNIV (deriv_mero_uhp f) (weight + 2) (deriv_qmod_poly weight p)"
⟨*proof*⟩

A quasimodular form (on the full modular group) of weight w and depth s can be written as a polynomial in E_2 whose coefficients are modular forms (the i-th coefficient having weight $w - 2i$) and whose degree is exactly s.

theorem *quasimodular_form_as_Eisenstein_E2:*
assumes *"quasimodular_form_explicit_UNIV f weight p"*
shows *"∃q. (∀i. poly.coeff q i ∈ MForms[weight - 2 * int i]) ∧ degree q = degree p ∧ poly q (ℰ 2) = f"*
⟨*proof*⟩

end

11 Application: Ramanujan-style identities for σ

theory *Eisenstein_Derivative_Identities*
imports *Serre_Derivative Quasimodular_Forms*
begin

theorem *serre_deriv_Eisenstein_E4: "serre_deriv 4 (ℰ 4) = -1/3 * ℰ 6"*
and *deriv_Eisenstein_E4: "deriv_mero_uhp (ℰ 4) = ⟨2 / 3 * i * pi⟩ * (ℰ 2 * ℰ 4 - ℰ 6)"*
and *divisor_sigma_5_3_1:*
*"80 * divisor_sigma 3 n + 168 * divisor_sigma 5 n = 8 * divisor_sigma 1 n + 240 * n * divisor_sigma 3 n + 1920 * (∑i∈{0<..<n}. divisor_sigma 1 i * divisor_sigma 3 (n - i))"*
⟨*proof*⟩

theorem *serre_deriv_Eisenstein_E6: "serre_deriv 6 (ℰ 6) = -1/2 * ℰ 8"*
and *deriv_Eisenstein_E6: "deriv_mero_uhp (ℰ 6) = ⟨i * pi⟩ * (ℰ 2 * ℰ 6 - ℰ 8)"*
and *divisor_sigma_7_5_1:*
*"12 * divisor_sigma 1 n + 252 * divisor_sigma 5 n + 240 * divisor_sigma 7 n =*
*504 * n * divisor_sigma 5 n + 6048 * (∑ i∈{0<..<n}. divisor_sigma 1 i * divisor_sigma 5 (n - i))"*
⟨*proof*⟩

lemma *degree_eq_SucE:*
assumes *"degree p = Suc n"*
obtains *c q* **where** *"degree q = n" "p = pCons c q"*
⟨*proof*⟩

theorem
defines *"E ≡ fps_Eisenstein_E"*
defines *"σ ≡ (λs n. complex_of_nat (divisor_sigma s n))"*
shows *deriv_Eisenstein_2: "deriv_mero_uhp (ℰ 2) = ⟨i * pi / 6⟩ * (ℰ 2 ^ 2 - ℰ 4)"*
and *divisor_sigma_3_1:*
*"20 * σ 3 n = (24 * of_nat n - 4) * σ 1 n + 48 * (∑ i∈{0<..<n}. σ 1 i * σ 1 (n-i))"*
⟨*proof*⟩

end

References

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

[2] E. Royer. Quasimodular forms: an introduction. *Annales mathématiques Blaise Pascal*, 19(2):297–306, 2012.