

Complex Lattices, Elliptic Functions, and the Modular Group

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

June 14, 2026

Abstract

This entry defines complex lattices, i.e. $\Lambda(\omega_1, \omega_2) = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$ where $\omega_1/\omega_2 \notin \mathbb{R}$. Based on this, various other related topics are covered:

- the modular group Γ and its fundamental region
- elliptic functions and their basic properties
- the Weierstraß elliptic function \wp and the fact that every elliptic function can be written in terms of \wp
- the Eisenstein series G_n (including the forbidden series G_2)
- the ordinary differential equation satisfied by \wp , the recurrence relation for G_n , and the addition and duplication theorems for \wp
- the lattice invariants g_2, g_3 , and Klein's J invariant
- the non-vanishing of the lattice discriminant Δ
- G_n, Δ, J as holomorphic functions in the upper half plane
- the Fourier expansion of $G_n(z)$ for $z \rightarrow i\infty$
- the functional equations of G_n, Δ, J , and η w.r.t. the modular group
- Dedekind's η function
- the inversion formulas for the Jacobi θ functions

In particular, this entry contains most of Chapters 1 and 3 from Apostol's *Modular Functions and Dirichlet Series in Number Theory* [1] and parts of Chapter 2.

The purpose of this entry is to provide a foundation for further formalisation of modular functions and modular forms.

Contents

1	Auxiliary material	5
1.1	The z -plane vs the q -disc	5
1.1.1	The neighbourhood of $i\infty$	5
1.1.2	The parameter q	6
1.2	Parallelogram-shaped paths	17
2	Möbius transforms and the modular group	19
2.1	Basic properties of Möbius transforms	19
2.2	Unimodular Möbius transforms	21
2.3	The modular group	24
2.3.1	Definition	24
2.3.2	Basic operations	27
2.3.3	Basic properties	29
2.4	Code generation	40
2.5	The factor of automorphy and the slash operator	42
2.6	Sending rational numbers to infinity	45
2.7	Representation as product of powers of generators	45
2.8	Induction rules in terms of generators	53
2.9	Reduction map	55
3	Complex lattices	62
3.1	Basic definitions and useful lemmas	63
3.2	Period parallelograms	75
3.3	Canonical representatives and the fundamental parallelogram	83
3.4	Equivalence of fundamental pairs	91
3.5	Additional useful facts	97
3.6	Doubly-periodic functions	107
4	Subgroups of the modular group	108
4.1	Definition and group action on the upper half plane	108
4.2	Conjugation	113
4.3	Elliptic points	115
4.4	Subgroups containing shifts	116
4.4.1	Congruence subgroups	117
4.4.2	Hecke subgroups $\Gamma_0(N)$	125
4.5	The subgroups $\Gamma_1(N)$	129
5	Fundamental regions of the modular group	131
5.1	Definition	132
5.2	The standard fundamental region	133
5.3	Proving that the standard region is fundamental	144
5.4	The corner point of the standard fundamental region	156

5.5	The elliptic points of the modular group	158
5.6	Fundamental regions for congruence subgroups	171
6	Elliptic Functions	175
6.1	Definition	175
6.2	Basic results about zeros and poles	180
6.3	Even elliptic functions	203
6.4	Closure properties of the class of elliptic functions	205
6.5	Affine transformations and surjectivity	209
7	The Weierstraß \wp Function	211
7.1	Preliminary convergence results	212
7.2	Definition and basic properties	233
7.3	Ellipticity and poles	242
7.4	The numbers e_1, e_2, e_3	247
7.5	Injectivity of \wp	251
7.6	Invariance under lattice transformations	255
7.7	Construction of arbitrary elliptic functions from \wp	260
8	Related facts about Jacobi theta functions	271
8.1	Uniqueness of quasi-periodic entire functions	272
8.2	Theta inversion	289
8.3	Theta nullwert inversions in the reals	296
9	Eisenstein series and the differential equations of \wp	301
9.1	Definition	302
9.2	The Laurent series expansion of \wp at the origin	305
9.3	Differential equations for \wp	308
9.4	Lattice invariants and a recurrence for the Eisenstein series	312
9.5	Fourier expansion	320
9.6	Behaviour under lattice transformations	327
9.7	Recurrence relation	330
10	Addition and duplication theorems for \wp	338
11	Connection between complex lattices and theta functions	356
12	Eisenstein series and related invariants as modular forms	367
12.1	Eisenstein series	368
12.2	The monomials of the Eisenstein polynomial	380
12.3	The normalised Eisenstein series	382
12.4	Identities for normalised Eisenstein series	388
12.5	The modular discriminant	396
12.6	Ramanujan's tau function	398
12.7	The modular λ function	401

12.8 Klein's J invariant	404
12.9 Klein's c	406
12.10 Values at specific points	409
12.11 Consequences for the fundamental region	411
13 The Dedekind η function	412
13.1 Definition and basic properties	413
13.2 Relation to the Jacobi ϑ functions	416
13.3 The inversion identity	421
13.4 General transformation law	424
13.5 The transformation law for G_2	436

1 Auxiliary material

1.1 The z -plane vs the q -disc

```
theory Z_Plane_Q_Disc
  imports "HOL-Complex_Analysis.Complex_Analysis" "Theta_Functions.Nome"
begin
```

In the study of modular forms and related subjects, we often need convert between the upper half of the complex plane (typically with a parameter written as z or τ) and the unit disc (with a parameter written as q).

This is particularly interesting for 1-periodic functions $f(z)$ (or more generally n -periodic functions where $n > 0$ is an integer) since such functions have a Fourier expansion in terms of q , i.e. we can view them both as functions $f(z)$ for $\text{Im}(z) > 0$ or $f(q)$ for $|q| < 1$, where the latter is only well-defined due to the periodicity.

1.1.1 The neighbourhood of $i\infty$

The following filter describes the neighbourhood of $i\infty$, i.e. the neighbourhood of all points with sufficiently big imaginary value. In terms of q , this corresponds to the point $q = 0$.

```
definition at_ii_inf :: "complex filter" ("at'_i $\infty$ ") where
  "at_ii_inf = filtercomap Im at_top"
```

```
lemma eventually_at_ii_inf:
  "eventually ( $\lambda z. \text{Im } z > c$ ) at_ii_inf"
  unfolding at_ii_inf_def using filterlim_at_top_dense by blast
```

```
lemma eventually_at_ii_inf_iff:
  " $(\forall_F z \text{ in } at\_ii\_inf. P z) \longleftrightarrow (\exists c. \forall z. \text{Im } z > c \longrightarrow P z)$ "
  by (simp add: at_ii_inf_def eventually_filtercomap_at_top_dense)
```

```
lemma eventually_at_ii_inf_iff':
  " $(\forall_F z \text{ in } at\_ii\_inf. P z) \longleftrightarrow (\exists c. \forall z. \text{Im } z \geq c \longrightarrow P z)$ "
  by (simp add: at_ii_inf_def eventually_filtercomap_at_top_linorder)
```

```
lemma filterlim_Im_at_ii_inf: "filterlim Im at_top at_i $\infty$ "
  unfolding at_ii_inf_def by (rule filterlim_filtercomap)
```

```
lemma filterlim_at_ii_infI:
  assumes "filterlim f F at_top"
  shows "filterlim ( $\lambda x. f (\text{Im } x)$ ) F at_i $\infty$ "
  using filterlim_filtercomapI[of f F at_top Im] assms by (simp add: at_ii_inf_def)
```

```
lemma filtermap_scaleR_at_ii_inf:
  assumes " $c > 0$ "
  shows "filtermap ( $\lambda z. c *_R z$ ) at_ii_inf = at_ii_inf"
```

```

proof (rule antisym)
  show "filtermap ( $\lambda z. c *_{\mathbb{R}} z$ ) at_ii_inf  $\leq$  at_ii_inf"
  proof (rule filter_leI)
    fix P
    assume "eventually P at_ii_inf"
    then obtain b where b: " $\bigwedge z. \text{Im } z > b \implies P z$ "
      by (auto simp: eventually_at_ii_inf_iff)
    have "eventually ( $\lambda z. \text{Im } z > b / c$ ) at_ii_inf"
      by (intro eventually_at_ii_inf)
    thus "eventually P (filtermap ( $\lambda z. c *_{\mathbb{R}} z$ ) at_ii_inf)"
      unfolding eventually_filtermap
      by eventually_elim (use assms in <auto intro!: b simp: field_simps>)
  qed
next
  show "filtermap ( $\lambda z. c *_{\mathbb{R}} z$ ) at_ii_inf  $\geq$  at_ii_inf"
  proof (rule filter_leI)
    fix P
    assume "eventually P (filtermap ( $\lambda z. c *_{\mathbb{R}} z$ ) at_ii_inf)"
    then obtain b where b: " $\bigwedge z. \text{Im } z > b \implies P (c *_{\mathbb{R}} z)$ "
      by (auto simp: eventually_at_ii_inf_iff eventually_filtermap)
    have b': "P z" if "Im z > b * c" for z
      using b[of "z /R c"] that assms by (auto simp: field_simps)
    have "eventually ( $\lambda z. \text{Im } z > b * c$ ) at_ii_inf"
      by (intro eventually_at_ii_inf)
    thus "eventually P at_ii_inf"
      unfolding eventually_filtermap
      by eventually_elim (use assms in <auto intro!: b' simp: field_simps>)
  qed
qed

```

lemma at_ii_inf_neq_bot [simp]: "at_ii_inf \neq bot"
 unfolding at_ii_inf_def by (intro filtercomap_neq_bot_surj surj_Im)
 auto

1.1.2 The parameter q

The standard mapping from z to q is $z \mapsto \exp(2i\pi z)$, which is also sometimes referred to as the square of the *nome*. However, if the period of the function is $n > 0$, we have to opt for $z \mapsto \exp(2i\pi z/n)$ instead, so we allow this additional flexibility here.

Note that the inverse mapping from q to z is multivalued. We arbitrarily choose the strip with $\text{Re}(z) \in (-\frac{1}{2}, \frac{1}{2}]$ as the codomain of the inverse mapping.

definition to_q :: "nat \Rightarrow complex \Rightarrow complex" where
 "to_q n $\tau = \exp (2 * \text{pi} * i * \tau / n)$ "

lemma to_nome_conv_to_q: "to_nome = to_q 2"
 by (auto simp: fun_eq_iff to_q_def to_nome_def mult_ac)

```

lemma to_q_conv_to_nome: "to_q n z = to_nome (2 * z / of_nat n)"
  by (simp add: to_q_def to_nome_def field_simps)

lemma to_q_add: "to_q n (w + z) = to_q n w * to_q n z"
  and to_q_diff: "to_q n (w - z) = to_q n w / to_q n z"
  and to_q_minus: "to_q n (-w) = inverse (to_q n w)"
  and to_q_power: "to_q n w ^ k = to_q n (of_nat k * w)"
  and to_q_power_int: "to_q n w powi m = to_q n (of_int m * w)"
  by (simp_all add: to_q_def add_divide_distrib diff_divide_distrib
      exp_add exp_diff exp_minus ring_distrib mult_ac exp_power_int
      flip: exp_of_nat_mult)

interpretation to_q: periodic_fun_simple "to_q n" "of_nat n"
proof
  show "to_q n (z + of_nat n) = to_q n z" for z
    by (cases "n = 0") (simp_all add: to_q_def ring_distrib exp_add add_divide_distrib)
qed

lemma to_q_of_nat_period [simp]: "to_q n (of_nat n) = 1"
  by (simp add: to_q_def exp_eq_polar)

lemma to_q_of_int [simp]:
  assumes "int n dvd m"
  shows "to_q n (of_int m) = 1"
proof -
  obtain k where m_eq: "m = int n * k"
    using assms by (elim dvdE)
  have "to_q n (of_int m) = to_q n (of_nat n * of_int k)"
    by (simp add: m_eq)
  also have "... = to_q n (of_nat n) powi k"
    by (simp only: to_q_power_int mult_ac)
  also have "... = 1"
    by simp
  finally show ?thesis .
qed

lemma to_q_of_nat [simp]:
  assumes "n dvd m"
  shows "to_q n (of_nat m) = 1"
  using to_q_of_int[of n "int m"] assms by (simp del: to_q_of_int)

lemma to_q_numeral [simp]:
  assumes "n dvd numeral m"
  shows "to_q n (numeral m) = 1"
  using to_q_of_nat[of n "numeral m"] assms by (simp del: to_q_of_nat)

lemma to_q_of_nat_period_1 [simp]: "w ∈ ℤ ⇒ to_q (Suc 0) w = 1"
  by (auto elim!: Ints_cases)

```

```

lemma Ln_to_q:
  assumes "x ∈ Re -' {n/2<..n/2}" "n > 0"
  shows "Ln (to_q n x) = 2 * pi * i * x / n"
unfolding to_q_def
proof (rule Ln_exp)
  have "-1/2 * pi < Re x / n * pi" "Re x / n * pi ≤ 1/2 * pi"
    using assms by (auto simp: field_simps)
  thus "-pi < Im (complex_of_real (2 * pi) * i * x / n)"
    using assms(2) by (auto simp: field_simps)
  show "Im (complex_of_real (2 * pi) * i * x / n) ≤ pi"
    using <Re x / n * pi ≤ 1/2 * pi> using assms(2) by (auto simp: field_simps)
qed

lemma to_q_nonzero [simp]: "to_q n τ ≠ 0"
  by (auto simp: to_q_def)

lemma norm_to_q [simp]: "norm (to_q n z) = exp (-2 * pi * Im z / n)"
  by (simp add: to_q_def)

lemma to_q_has_field_derivative [derivative_intros]:
  assumes [derivative_intros]: "(f has_field_derivative f') (at z)" and
  n: "n > 0"
  shows "((λz. to_q n (f z)) has_field_derivative (2 * pi * i * f' /
  n * to_q n (f z))) (at z)"
  unfolding to_q_def by (rule derivative_eq_intros refl | (use n in simp;
  fail))+

lemma deriv_to_q [simp]: "n > 0 ⇒ deriv (to_q n) z = 2 * pi * i / n
  * to_q n z"
  by (auto intro!: DERIV_imp_deriv derivative_eq_intros)

lemma to_q_holomorphic_on [holomorphic_intros]:
  "f holomorphic_on A ⇒ n > 0 ⇒ (λz. to_q n (f z)) holomorphic_on
  A"
  unfolding to_q_def by (intro holomorphic_intros) auto

lemma to_q_analytic_on [analytic_intros]:
  "f analytic_on A ⇒ n > 0 ⇒ (λz. to_q n (f z)) analytic_on A"
  unfolding to_q_def by (intro analytic_intros) auto

lemma to_q_continuous_on [continuous_intros]:
  "continuous_on A f ⇒ n > 0 ⇒ continuous_on A (λz. to_q n (f z))"
  unfolding to_q_def by (intro continuous_intros) auto

lemma to_q_continuous [continuous_intros]:
  "continuous F f ⇒ n > 0 ⇒ continuous F (λz. to_q n (f z))"
  unfolding to_q_def by (auto intro!: continuous_intros simp: divide_inverse)

```

```

lemma to_q_tendsto [tendsto_intros]:
  "(f ⟶ x) F ⟹ n > 0 ⟹ ((λz. to_q n (f z)) ⟶ to_q n x) F"
  unfolding to_q_def by (intro tendsto_intros) auto

lemma to_q_eq_to_qE:
  assumes "to_q m τ = to_q m τ'" "m > 0"
  obtains n where "τ' = τ + of_int n * of_nat m"
proof -
  from assms obtain n where "2 * pi * i * τ / m = 2 * pi * i * τ' / m
+ real_of_int (2 * n) * pi * i"
  using assms unfolding to_q_def exp_eq by blast
  also have "... = 2 * pi * i * (τ' / m + of_int n)"
  by (simp add: algebra_simps)
  also have "2 * pi * i * τ / m = 2 * pi * i * (τ / m)"
  by simp
  finally have "τ / m = τ' / m + of_int n"
  by (subst (asm) mult_cancel_left) auto
  hence "τ = τ' + of_int n * of_nat m"
  using <m > 0> by (metis divide_add_eq_iff divide_cancel_right of_nat_eq_0_iff
rel_simps(70))
  thus ?thesis
  by (intro that[of "-n"]) auto
qed

lemma to_q_inj_on_standard:
  assumes n: "n > 0"
  shows "inj_on (to_q n) (Re -' {-n/2..

```

```

ing assms
  by simp
qed
moreover have sineq:"exp (- (pp * ix / n)) * sin prx
  = exp (- (pp * iy / n)) * sin pry"
proof -
  have "Im (to_q n x) = Im (to_q n y)"
    using eq by simp
  then show ?thesis
    unfolding x y to_q_def Re_exp Im_exp pp_def prx_def pry_def
    by simp
qed
ultimately have "(exp (- (pp * ix / n)) * sin (prx))
  / (exp (- (pp * ix / n)) * cos (prx))
  = (exp (- (pp * iy / n)) * sin (pry))
  / (exp (- (pp * iy / n)) * cos (pry))"
  by auto
then have teq:"tan prx = tan pry"
  by (subst (asm) (1 2) mult_divide_mult_cancel_left) (auto simp: tan_def)

have sgn_eq_cos:"sgn (cos (prx)) = sgn (cos (pry))"
proof -
  have "sgn (exp (- (pp * ix / n)) * cos prx)
    = sgn (exp (- (pp * iy / n)) * cos pry)"
    using coseq by simp
  then show ?thesis by (simp add:sgn_mult)
qed
have sgn_eq_sin:"sgn (sin (prx)) = sgn (sin (pry))"
proof -
  have "sgn (exp (- (pp * ix / n)) * sin prx)
    = sgn (exp (- (pp * iy / n)) * sin pry)"
    using sineq by simp
  then show ?thesis by (simp add:sgn_mult)
qed

have "prx=pry" if "tan prx = 0"
proof -
  define pi2 where "pi2 = pi / 2"
  have [simp]: "cos pi2 = 0" "cos (-pi2) = 0"
    "sin pi2 = 1" "sin (-pi2) = -1"
  unfolding pi2_def by auto
  have "prx∈{-pi,-pi2,0,pi2}"
  proof -
    from tan_eq_0_Ex[OF that]
    obtain k0 where k0:"prx = real_of_int k0 / 2 * pi"
      by auto
    then have "rx / n = real_of_int k0 / 4"
      unfolding pp_def prx_def using n by (auto simp: field_simps)
    with rxry have "k0∈{-2,-1,0,1}"

```

```

    by auto
    then show ?thesis using k0 pi2_def by auto
qed
moreover have "pry ∈ {-pi, -pi2, 0, pi2}"
proof -
  from tan_eq_0_Ex that teq
  obtain k0 where k0: "pry = real_of_int k0 / 2 * pi"
    by auto
  then have "ry / n = real_of_int k0/4"
    unfolding pp_def pry_def using n by (auto simp: field_simps)
  with rxry have "k0 ∈ {-2, -1, 0, 1}"
    by auto
  then show ?thesis using k0 pi2_def by auto
qed
moreover note sgn_eq_cos sgn_eq_sin
ultimately show "prx=pry" by auto
qed
moreover have "prx=pry" if "tan prx ≠ 0"
proof -
  from tan_eq[OF teq that]
  obtain k0 where k0: "prx = pry + real_of_int k0 * pi"
    by auto
  then have "pi * (2 * rx / n) = pi * (2 * ry / n + real_of_int k0)"
    unfolding pp_def prx_def pry_def by (auto simp: algebra_simps)
  then have "real_of_int k0 = 2 * ((rx - ry) / n)"
    by (subst diff_divide_distrib, subst (asm) mult_left_cancel) (use
n in auto)
  also have "... ∈ {-2<..<2}"
    using rxry using n by (auto simp: field_simps)
  finally have "real_of_int k0 ∈ {- 2<..<2}" by simp
  then have "k0 ∈ {-1, 0, 1}" by auto
  then have "prx=pry-pi ∨ prx = pry ∨ prx = pry+pi"
    using k0 by auto
  moreover have False if "prx=pry-pi ∨ prx = pry+pi"
  proof -
    have "cos prx = - cos pry"
      using that by auto
    moreover note sgn_eq_cos
    ultimately have "cos prx = 0"
      by (simp add: sgn_zero_iff)
    then have "tan prx = 0" unfolding tan_def by auto
    then show False
      using <tan prx ≠ 0> unfolding prx_def by auto
  qed
  ultimately show "prx = pry" by auto
qed
ultimately have "prx=pry" by auto
then have "rx = ry" unfolding prx_def pry_def pp_def using n by auto
moreover have "ix = iy"

```

```

proof -
  have "cos prx ≠ 0 ∨ sin prx ≠ 0"
    using sin_zero_norm_cos_one by force
  then have "exp (- (pp * ix)) = exp (- (pp * iy))"
    using coseq sineq <prx = pry> n by auto
  then show "ix = iy" unfolding pp_def by auto
qed
ultimately show "x=y" unfolding x y by auto
qed

lemma filterlim_to_q_at_ii_inf' [tendsto_intros]:
  assumes n: "n > 0"
  shows "filterlim (to_q n) (nhds 0) at_ii_inf"
proof -
  have "((λz. exp (- (2 * pi * Im z) / n)) → 0) at_ii_inf"
    unfolding at_ii_inf_def
    by (rule filterlim_compose[OF _ filterlim_filtercomap]) (use n in
real_asymp)
  hence "filterlim (λz. norm (to_q n z)) (nhds 0) at_ii_inf"
    by (simp add: to_q_def)
  thus ?thesis
    using tendsto_norm_zero_iff by blast
qed

lemma filterlim_to_q_at_ii_inf [tendsto_intros]: "n > 0 ⇒ filterlim
(to_q n) (at 0) at_ii_inf"
  using filterlim_to_q_at_ii_inf' by (auto simp: filterlim_at)

lemma eventually_to_q_neq:
  assumes n: "n > 0"
  shows "eventually (λw. to_q n w ≠ to_q n z) (at z)"
proof -
  have "eventually (λw. w ∈ ball z 1 - {z}) (at z)"
    by (intro eventually_at_in_open) auto
  thus ?thesis
proof eventually_elim
  case (elim w)
  show ?case
proof
  assume "to_q n w = to_q n z"
  then obtain m where eq: "z = w + of_int m * of_nat n"
    using n by (elim to_q_eq_to_qE)
  with elim have "real_of_int (|m| * int n) < 1"
    by (simp add: dist_norm norm_mult)
  hence "|m| * int n < 1"
    by linarith
  with n have "m = 0"
    by (metis add_0 mult_pos_pos not_less of_nat_0_less_iff zero_less_abs_iff
zless_imp_add1_zle)

```

```

      thus False
      using elim eq by simp
    qed
  qed
qed

```

```

lemma inj_on_to_q:
  assumes n: "n > 0"
  shows "inj_on (to_q n) (ball z (1/2))"
proof
  fix x y assume xy: "x ∈ ball z (1/2)" "y ∈ ball z (1/2)" "to_q n x
= to_q n y"
  from xy have "dist x z < 1 / 2" "dist y z < 1 / 2"
    by (auto simp: dist_commute)
  hence "dist x y < 1 / 2 + 1 / 2"
    using dist_triangle_less_add by blast
  moreover obtain m where m: "y = x + of_int m * of_nat n"
    by (rule to_q_eq_to_qE[OF xy(3) n]) auto
  ultimately have "real_of_int (|m| * int n) < 1"
    by (auto simp: dist_norm norm_mult)
  hence "|m| * int n < 1"
    by linarith
  with n have "m = 0"
    by (metis add_0 mult_pos_pos not_less of_nat_0_less_iff zero_less_abs_iff
zless_imp_add1_zle)
  thus "x = y"
    using m by simp
qed

```

```

lemma filtermap_to_q_nhds:
  assumes n: "n > 0"
  shows "filtermap (to_q n) (nhds z) = nhds (to_q n z)"
proof (rule filtermap_nhds_open_map')
  show "open (ball z (1 / 2))" "z ∈ ball z (1 / 2)" "isCont (to_q n)
z"
    using n by (auto intro!: continuous_intros)
  show "open (to_q n ' S)" if "open S" "S ⊆ ball z (1 / 2)" for S
  proof (rule open_mapping_thm3)
    show "inj_on (to_q n) S"
      using that by (intro inj_on_subset[OF inj_on_to_q] n)
    qed (use that n in <auto intro!: holomorphic_intros>)
  qed

```

```

lemma filtermap_to_q_at:
  assumes n: "n > 0"
  shows "filtermap (to_q n) (at z) = at (to_q n z)"
  using filtermap_to_q_nhds
proof (rule filtermap_nhds_eq_imp_filtermap_at_eq)

```

```

    show "eventually ( $\lambda x. \text{to\_q } n \ x = \text{to\_q } n \ z \longrightarrow x = z$ ) (at z)"
      using eventually_to_q_neq[OF n, of z] by eventually_elim (use n in
auto)
qed fact+

lemma is_pole_to_q_iff:
  assumes n: "n > 0"
  shows "is_pole f (to_q n x)  $\longleftrightarrow$  is_pole (f o to_q n) x"
proof -
  have "filtermap f (at (to_q n x))
    = filtermap f (filtermap (to_q n) (at x))"
    unfolding filtermap_to_q_at[OF n] by simp
  also have "... = filtermap (f o to_q n) (at x)"
    unfolding filtermap_filtermap unfolding comp_def by simp
  finally show ?thesis unfolding is_pole_def filterlim_def by simp
qed

definition of_q :: "nat  $\Rightarrow$  complex  $\Rightarrow$  complex" where
  "of_q n q = ln q / (2 * pi * i / n)"

lemma Im_of_q: "q  $\neq$  0  $\implies$  n > 0  $\implies$  Im (of_q n q) = -n * ln (norm q)
/ (2 * pi)"
  by (simp add: of_q_def Im_divide power2_eq_square)

lemma Im_of_q_gt:
  assumes "norm q < exp (-2 * pi * c / n)" "q  $\neq$  0" "n > 0"
  shows "Im (of_q n q) > c"
proof -
  have "-Im (of_q n q) = n * ln (norm q) / (2 * pi)"
    using assms by (subst Im_of_q) auto
  also have "ln (norm q) < ln (exp (-2 * pi * c / n))"
    by (subst ln_less_cancel_iff) (use assms in auto)
  hence "n * ln (norm q) / (2 * pi) < n * ln (exp (-2 * pi * c / n)) /
(2 * pi)"
    using <n > 0> by (intro mult_strict_left_mono divide_strict_right_mono)
  auto
  also have "... = -c"
    using <n > 0> by simp
  finally show ?thesis
    by simp
qed

lemma to_q_of_q [simp]: "q  $\neq$  0  $\implies$  n > 0  $\implies$  to_q n (of_q n q) = q"
  by (simp add: to_q_def of_q_def)

lemma of_q_to_q:
  assumes "m > 0"
  shows " $\exists n. \text{of\_q } m \ (\text{to\_q } m \ \tau) = \tau + \text{of\_int } n * \text{of\_nat } m$ "
proof

```

```

show "of_q m (to_q m τ) =
      τ + of_int (-unwinding (complex_of_real (2 * pi) * i * τ / m))
* of_nat m"
  using unwinding[of "2 * pi * i * τ / m"] assms
  by (simp add: of_q_def to_q_def field_simps)
qed

lemma filterlim_norm_at_0: "filterlim norm (at_right 0) (at 0)"
  unfolding filterlim_at
  by (auto simp: eventually_at tendsto_norm_zero_iff intro: exI[of _ 1])

lemma filterlim_of_q_at_0:
  assumes n: "n > 0"
  shows "filterlim (of_q n) at_ii_inf (at 0)"
proof -
  have "filterlim (λq. -n * ln (norm q) / (2 * pi)) at_top (at 0)"
    by (rule filterlim_compose[OF _ filterlim_norm_at_0]) (use n in real_asymp)
  also have "eventually (λq::complex. q ≠ 0) (at 0)"
    by (auto simp: eventually_at intro: exI[of _ 1])
  hence "eventually (λq. -n * ln (norm q) / (2 * pi) = Im (of_q n q))
(at 0)"
    by eventually_elim (use n in <simp add: Im_of_q>)
  hence "filterlim (λq::complex. -n * ln (norm q) / (2 * pi)) at_top (at
0) ↔
      filterlim (λq. Im (of_q n q)) at_top (at 0)"
    by (intro filterlim_cong) auto
  finally show ?thesis
    by (simp add: at_ii_inf_def filterlim_filtercomap_iff o_def)
qed

lemma at_ii_inf_filtermap:
  assumes "n > 0"
  shows "filtermap (to_q n) at_ii_inf = at 0"
proof (rule filtermap_fun_inverse[OF filterlim_of_q_at_0 filterlim_to_q_at_ii_inf])
  have "eventually (λx. x ≠ 0) (at (0 :: complex))"
    by (rule eventually_neq_at_within)
  thus "∀F x in at 0. to_q n (of_q n x) = x"
    by eventually_elim (use assms in auto)
qed fact+

lemma eventually_at_ii_inf_to_q:
  assumes n: "n > 0"
  shows "eventually P (at 0) = (∀F x in at_ii_inf. P (to_q n x))"
proof -
  have "(∀F x in at_ii_inf. P (to_q n x)) ↔ (∀F x in filtermap (to_q
n) at_ii_inf. P x)"
    by (simp add: eventually_filtermap)
  also have "... ↔ eventually P (at 0)"
    by (simp add: at_ii_inf_filtermap n)

```

```

    finally show ?thesis ..
qed

lemma of_q_tendsto:
  assumes "x ∈ Re -' {real n / 2<.. $real n / 2$ } " "n > 0"
  shows "of_q n -to_q n x → x"
proof -
  obtain rx ix where x:"x = Complex rx ix"
  using complex.exhaust_sel by blast
  have "Re (to_q n x) > 0" if "Im (to_q n x) = 0"
  proof -
    have "sin (2 * pi * rx / n) = 0"
      using that unfolding to_q_def x Im_exp Re_exp by simp
    then obtain k where "pi * (2 * rx / n) = pi * real_of_int k"
      unfolding sin_zero_iff_int2 by (auto simp: algebra_simps)
    hence k: "2 * rx / n = real_of_int k"
      using mult_cancel_left pi_neq_zero by blast
    moreover have "2*rx/n ∈ {-1<.. $<1$ } "
      using assms unfolding x by simp
    ultimately have "k=0" by auto
    then have "rx=0" using k <n > 0> by auto
    then show ?thesis unfolding to_q_def x
      using Re_exp by simp
  qed
  then have "to_q n x ∉ ℝ≤0"
    unfolding complex_nonpos_Reals_iff
    unfolding Im_exp Re_exp to_q_def
    by (auto simp add: complex_nonpos_Reals_iff)
  moreover have "Ln (to_q n x) / (2 * complex_of_real pi * i / n) = x"

  by (subst Ln_to_q) (use assms in <auto simp: field_simps>)
  ultimately show "of_q n -to_q n x → x"
    unfolding of_q_def by (auto intro!: tendsto_eq_intros)
qed

lemma of_q_to_q_Re:
  assumes "x ∈ Re -' {real n / 2<.. $real n / 2$ } " "n > 0"
  shows "of_q n (to_q n x) = x"
proof -
  have "- pi < Im (complex_of_real (2 * pi) * i * x / n)"
  proof -
    have "pi * (-1/2) < pi * (Re x / n)"
      by (rule mult_strict_left_mono) (use assms in auto)
    then show ?thesis by auto
  qed
  moreover have "Im (complex_of_real (2 * pi) * i * x / n) ≤ pi"
    using assms by auto
  ultimately show ?thesis using assms(2)
    unfolding to_q_def of_q_def

```

```

    by (subst Ln_exp) auto
qed

end

```

1.2 Parallelogram-shaped paths

```

theory Parallelogram_Paths
  imports "HOL-Complex_Analysis.Complex_Analysis"
begin

definition parallelogram_path :: "'a :: real_normed_vector  $\Rightarrow$  'a  $\Rightarrow$  'a
 $\Rightarrow$  real  $\Rightarrow$  'a" where
  "parallelogram_path z a b =
    linepath z (z + a) +++ linepath (z + a) (z + a + b) +++
    linepath (z + a + b) (z + b) +++ linepath (z + b) z"

lemma path_parallelogram_path [intro]: "path (parallelogram_path z a
b)"
  and valid_path_parallelogram_path [intro]: "valid_path (parallelogram_path
z a b)"
  and pathstart_parallelogram_path [simp]: "pathstart (parallelogram_path
z a b) = z"
  and pathfinish_parallelogram_path [simp]: "pathfinish (parallelogram_path
z a b) = z"
  by (auto simp: parallelogram_path_def intro!: valid_path_join)

lemma parallelogram_path_altdef:
  fixes z a b :: complex
  defines "g  $\equiv$  ( $\lambda w. z + \text{Re } w *_R a + \text{Im } w *_R b$ )"
  shows "parallelogram_path z a b = g  $\circ$  rectpath 0 (1 + i)"
  by (simp add: parallelogram_path_def rectpath_def g_def Let_def o_def
joinpaths_def
      fun_eq_iff linepath_def scaleR_conv_of_real algebra_simps)

lemma
  fixes f :: "complex  $\Rightarrow$  complex" and z  $\omega_1 \omega_2$  :: complex
  defines "I  $\equiv$  ( $\lambda a b. \text{contour\_integral (linepath (z + a) (z + b)) f}$ )"
  defines "P  $\equiv$  parallelogram_path z  $\omega_1 \omega_2$ "
  assumes "continuous_on (path_image P) f"
  shows contour_integral_parallelogram_path:
    "contour_integral P f =
      (I 0  $\omega_1$  - I  $\omega_2$  ( $\omega_1 + \omega_2$ )) - (I 0  $\omega_2$  - I  $\omega_1$  ( $\omega_1 + \omega_2$ ))"
  and contour_integral_parallelogram_path':
    "contour_integral P f =
      contour_integral (linepath z (z +  $\omega_1$ )) ( $\lambda x. f x - f (x +$ 
 $\omega_2)$ ) -
      contour_integral (linepath z (z +  $\omega_2$ )) ( $\lambda x. f x - f (x +$ 
 $\omega_1)$ )"

```

```

proof -
  define L where "L = (λa b. linepath (z + a) (z + b))"
  have I: "(f has_contour_integral (I a b)) (L a b)"
    if "closed_segment (z + a) (z + b) ⊆ path_image P" for a b
    unfolding I_def L_def
    by (intro has_contour_integral_integral contour_integrable_continuous_linepath
        continuous_on_subset[OF assms(3)] that)
  have "(f has_contour_integral
    (I 0 ω1 + (I ω1 (ω1 + ω2) + ((-I ω2 (ω1 + ω2)) + (-I 0 ω2))))
    (L 0 ω1 +++ L ω1 (ω1 + ω2) +++ reversepath (L ω2 (ω1 + ω2))
    +++ reversepath (L 0 ω2))"
    unfolding P_def parallelogram_path_def
    by (intro has_contour_integral_join valid_path_join valid_path_linepath
        has_contour_integral_reversepath I)
    (auto simp: parallelogram_path_def path_image_join closed_segment_commute
        P_def L_def add_ac)
  thus "contour_integral P f =
    (I 0 ω1 - I ω2 (ω1 + ω2)) - (I 0 ω2 - I ω1 (ω1 + ω2))"
    by (intro contour_integral_unique)
    (simp_all add: parallelogram_path_def P_def L_def algebra_simps)

  also have "I 0 ω2 - I ω1 (ω1 + ω2) = contour_integral (L 0 ω2) (λx.
  f x - f (x + ω1))"
  proof -
    have "(f has_contour_integral I ω1 (ω1 + ω2)) (L ω1 (ω1 + ω2))"
      by (rule I) (auto simp: parallelogram_path_def path_image_join P_def
      add_ac)
    also have "L ω1 (ω1 + ω2) = (+) ω1 ∘ L 0 ω2"
      by (simp add: linepath_def L_def fun_eq_iff algebra_simps)
    finally have "((λx. f (x + ω1)) has_contour_integral I ω1 (ω1 + ω2))
    (L 0 ω2)"
      unfolding has_contour_integral_translate .
    hence "((λx. f x - f (x + ω1)) has_contour_integral (I 0 ω2 - I ω1
    (ω1 + ω2))) (L 0 ω2)"
      by (intro has_contour_integral_diff I)
      (auto simp: parallelogram_path_def path_image_join closed_segment_commute
      L_def P_def)
    thus ?thesis
      by (rule contour_integral_unique [symmetric])
  qed

  also have "I 0 ω1 - I ω2 (ω1 + ω2) = contour_integral (L 0 ω1) (λx.
  f x - f (x + ω2))"
  proof -
    have "(f has_contour_integral I ω2 (ω1 + ω2)) (L ω2 (ω1 + ω2))"
      by (rule I) (auto simp: parallelogram_path_def path_image_join closed_segment_commute
      P_def L_def add_ac)
    also have "L ω2 (ω1 + ω2) = (+) ω2 ∘ L 0 ω1"
      by (simp add: linepath_def L_def o_def algebra_simps)
  end

```

```

    finally have "(( $\lambda x. f (x + \omega 2)$ ) has_contour_integral I  $\omega 2$  ( $\omega 1 + \omega 2$ ))
(L 0  $\omega 1$ )"
      unfolding has_contour_integral_translate .
    hence "(( $\lambda x. f x - f (x + \omega 2)$ ) has_contour_integral (I 0  $\omega 1 - I \omega 2$ 
( $\omega 1 + \omega 2$ ))) (L 0  $\omega 1$ )"
      by (intro has_contour_integral_diff I)
      (auto simp: parallelogram_path_def path_image_join closed_segment_commute
P_def)
    thus ?thesis
      by (rule contour_integral_unique [symmetric])
    qed

    finally show "contour_integral P f =
contour_integral (linepath z (z +  $\omega 1$ )) ( $\lambda x. f x - f$ 
(x +  $\omega 2$ )) -
contour_integral (linepath z (z +  $\omega 2$ )) ( $\lambda x. f x - f$ 
(x +  $\omega 1$ ))"
      by (simp add: L_def add_ac)
    qed

end

```

2 Möbius transforms and the modular group

```

theory Modular_Group
imports
  "HOL-Complex_Analysis.Complex_Analysis"
  "HOL-Number_Theory.Number_Theory"
begin

```

2.1 Basic properties of Möbius transforms

```

lemma moebius_uminus [simp]: "moebius (-a) (-b) (-c) (-d) = moebius a
b c d"
  by (simp add: fun_eq_iff moebius_def divide_simps) (simp add: algebra_simps
add_eq_0_iff2)

```

```

lemma moebius_uminus': "moebius (-a) b c d = moebius a (-b) (-c) (-d)"
  by (simp add: fun_eq_iff moebius_def divide_simps) (simp add: algebra_simps
add_eq_0_iff2)

```

```

lemma moebius_diff_eq:
  fixes a b c d :: "'a :: field"
  defines "f  $\equiv$  moebius a b c d"
  assumes *: "c = 0  $\vee$  z  $\neq$  -d / c  $\wedge$  w  $\neq$  -d / c"
  shows "f w - f z = (a * d - b * c) / ((c * w + d) * (c * z + d)) *
(w - z)"

```

```

using * by (auto simp: moebius_def divide_simps f_def add_eq_0_iff)
          (auto simp: algebra_simps)

lemma moebius_shift:
  "moebius a b c d (z + of_int n) = moebius a (a * of_int n + b) c (c
* of_int n + d) z"
  by (simp add: moebius_def algebra_simps)

lemma moebius_eq_shift: "moebius 1 (of_int n) 0 1 z = z + of_int n"
  by (simp add: moebius_def)

lemma moebius_S:
  assumes "a * d - b * c ≠ 0" "z ≠ 0"
  shows "moebius a b c d (-1 / z) = moebius b (- a) d (- c) (z ::
'a :: field)"
  using assms by (auto simp: divide_simps moebius_def)

lemma moebius_eq_S: "moebius 0 1 (-1) 0 z = -1 / z"
  by (simp add: moebius_def)

lemma continuous_on_moebius [continuous_intros]:
  fixes a b c d :: "'a :: real_normed_field"
  assumes "c ≠ 0 ∨ d ≠ 0" "c = 0 ∨ -d / c ∉ A"
  shows "continuous_on A (moebius a b c d)"
  unfolding moebius_def
  by (intro continuous_intros) (use assms in <auto simp: add_eq_0_iff>)

lemma continuous_on_moebius' [continuous_intros]:
  fixes a b c d :: "'a :: real_normed_field"
  assumes "continuous_on A f" "c ≠ 0 ∨ d ≠ 0" "∧z. z ∈ A ⇒ c = 0
∨ f z ≠ -d / c"
  shows "continuous_on A (λx. moebius a b c d (f x))"
proof -
  have "continuous_on A (moebius a b c d ∘ f)" using assms
    by (intro continuous_on_compose assms continuous_on_moebius) force+
  thus ?thesis
    by (simp add: o_def)
qed

lemma holomorphic_on_moebius [holomorphic_intros]:
  assumes "c ≠ 0 ∨ d ≠ 0" "c = 0 ∨ -d / c ∉ A"
  shows "(moebius a b c d) holomorphic_on A"
  unfolding moebius_def
  by (intro holomorphic_intros) (use assms in <auto simp: add_eq_0_iff>)

lemma holomorphic_on_moebius' [holomorphic_intros]:
  assumes "f holomorphic_on A" "c ≠ 0 ∨ d ≠ 0" "∧z. z ∈ A ⇒ c =
0 ∨ f z ≠ -d / c"

```

```

shows "(λx. moebius a b c d (f x)) holomorphic_on A"
proof -
  have "(moebius a b c d ∘ f) holomorphic_on A" using assms
    by (intro holomorphic_on_compose assms holomorphic_on_moebius) force+
  thus ?thesis
    by (simp add: o_def)
qed

```

```

lemma analytic_on_moebius [analytic_intros]:
  assumes "c ≠ 0 ∨ d ≠ 0" "c = 0 ∨ -d / c ∉ A"
  shows "(moebius a b c d) analytic_on A"
  unfolding moebius_def
  by (intro analytic_intros) (use assms in <auto simp: add_eq_0_iff>)

```

```

lemma analytic_on_moebius' [analytic_intros]:
  assumes "f analytic_on A" "c ≠ 0 ∨ d ≠ 0" "∧z. z ∈ A ⇒ c = 0 ∨
f z ≠ -d / c"
  shows "(λx. moebius a b c d (f x)) analytic_on A"
proof -
  have "(moebius a b c d ∘ f) analytic_on A" using assms
    by (intro analytic_on_compose assms analytic_on_moebius) force+
  thus ?thesis
    by (simp add: o_def)
qed

```

```

lemma moebius_has_field_derivative:
  assumes "c = 0 ∨ x ≠ -d / c" "c ≠ 0 ∨ d ≠ 0"
  shows "(moebius a b c d has_field_derivative (a * d - b * c) / (c
* x + d) ^ 2) (at x within A)"
  unfolding moebius_def
  apply (rule derivative_eq_intros refl)+
  using assms
  apply (auto simp: divide_simps add_eq_0_iff power2_eq_square split:
if_splits)
  apply (auto simp: algebra_simps)?
  done

```

2.2 Unimodular Möbius transforms

A unimodular Möbius transform has integer coefficients and determinant ± 1 .

```

locale unimodular_moebius_transform =
  fixes a b c d :: int
  assumes unimodular: "a * d - b * c = 1"
begin

```

```

definition φ :: "complex ⇒ complex" where
  "φ = moebius (of_int a) (of_int b) (of_int c) (of_int d)"

```

```

lemma cnj_φ: "φ (cnj z) = cnj (φ z)"
  by (simp add: moebius_def φ_def)

lemma Im_transform:
  "Im (φ z) = Im z / norm (of_int c * z + of_int d) ^ 2"
proof -
  have "Im (φ z) = Im z * of_int (a * d - b * c) /
        ((real_of_int c * Re z + real_of_int d)^2 + (real_of_int
c * Im z)^2)"
    by (simp add: φ_def moebius_def Im_divide algebra_simps)
  also have "a * d - b * c = 1"
    using unimodular by simp
  also have "((real_of_int c * Re z + real_of_int d)^2 + (real_of_int c
* Im z)^2) =
        norm (of_int c * z + of_int d) ^ 2"
    unfolding cmod_power2 by (simp add: power2_eq_square algebra_simps)
  finally show ?thesis
    by simp
qed

lemma Im_transform_pos_aux:
  assumes "Im z ≠ 0"
  shows "of_int c * z + of_int d ≠ 0"
proof (cases "c = 0")
  case False
  hence "Im (of_int c * z + of_int d) ≠ 0"
    using assms by auto
  moreover have "Im 0 = 0"
    by simp
  ultimately show ?thesis
    by metis
next
  case True
  thus ?thesis using unimodular
    by auto
qed

lemma Im_transform_pos: "Im z > 0 ⇒ Im (φ z) > 0"
  using Im_transform_pos_aux[of z] by (auto simp: Im_transform)

lemma Im_transform_neg: "Im z < 0 ⇒ Im (φ z) < 0"
  using Im_transform_pos[of "cnj z"] by (simp add: cnj_φ)

lemma Im_transform_zero_iff [simp]: "Im (φ z) = 0 ⇔ Im z = 0"
  using Im_transform_pos_aux[of z] by (auto simp: Im_transform)

lemma Im_transform_pos_iff [simp]: "Im (φ z) > 0 ⇔ Im z > 0"
  using Im_transform_pos[of z] Im_transform_neg[of z] Im_transform_zero_iff[of
z]

```

```

    by (cases "Im z" "0 :: real" rule: linorder_cases) (auto simp del: Im_transform_zero_iff)

lemma Im_transform_neg_iff [simp]: "Im ( $\varphi$  z) < 0  $\longleftrightarrow$  Im z < 0"
  using Im_transform_pos_iff[of "cnj z"] by (simp add: cnj_ $\varphi$ )

lemma Im_transform_nonneg_iff [simp]: "Im ( $\varphi$  z)  $\geq$  0  $\longleftrightarrow$  Im z  $\geq$  0"
  using Im_transform_neg_iff[of z] by linarith

lemma Im_transform_nonpos_iff [simp]: "Im ( $\varphi$  z)  $\leq$  0  $\longleftrightarrow$  Im z  $\leq$  0"
  using Im_transform_pos_iff[of z] by linarith

lemma transform_in_reals_iff [simp]: " $\varphi$  z  $\in \mathbb{R}$   $\longleftrightarrow$  z  $\in \mathbb{R}$ "
  using Im_transform_zero_iff[of z] by (auto simp: complex_is_Real_iff)

end

lemma Im_one_over_neg_iff [simp]: "Im (1 / z) < 0  $\longleftrightarrow$  Im z > 0"
proof -
  interpret unimodular_moebius_transform 0 1 "-1" 0
  by standard auto
  show ?thesis
  using Im_transform_pos_iff[of z] by (simp add:  $\varphi$ _def moebius_def)
qed

locale inverse_unimodular_moebius_transform = unimodular_moebius_transform
begin

sublocale inv: unimodular_moebius_transform d "-b" "-c" a
  by unfold_locales (use unimodular in Groebner_Basis.algebra)

lemma inv_ $\varphi$ :
  assumes "of_int c * z + of_int d  $\neq$  0"
  shows "inv. $\varphi$  ( $\varphi$  z) = z"
  using unimodular_assms
  unfolding inv. $\varphi$ _def  $\varphi$ _def of_int_minus
  by (subst moebius_inverse) (auto simp flip: of_int_mult)

lemma inv_ $\varphi$ ':
  assumes "of_int c * z - of_int a  $\neq$  0"
  shows " $\varphi$  (inv. $\varphi$  z) = z"
  using unimodular_assms
  unfolding inv. $\varphi$ _def  $\varphi$ _def of_int_minus
  by (subst moebius_inverse') (auto simp flip: of_int_mult)

end

```

2.3 The modular group

2.3.1 Definition

We define the modular group as all integer tuples (a, b, c, d) with $ad - bc = 1$.

```
typedef modgrp = "{(a,b,c,d::int). a * d - b * c = 1}"
  by (rule exI[of _ "(1,0,0,1)"]) auto

setup_lifting type_definition_modgrp

instantiation modgrp :: one
begin

lift_definition one_modgrp :: modgrp is "(1, 0, 0, 1)"
  by auto

instance ..
end

instantiation modgrp :: uminus
begin

lift_definition uminus_modgrp :: "modgrp  $\Rightarrow$  modgrp" is " $\lambda(a,b,c,d). (-a,$ 
 $-b, -c, -d)$ "
  by auto

instance ..
end

lemma minus_minus_modgrp [simp]: " $\text{--}(\text{--} f :: \text{modgrp}) = f$ "
  by transfer auto

instantiation modgrp :: sgn
begin

lift_definition sgn_modgrp :: "modgrp  $\Rightarrow$  modgrp"
  is " $\lambda(a,b,c,d). \text{if } c < 0 \vee c = 0 \wedge d < 0 \text{ then } (-1,0,0,-1) \text{ else } (1,0,0,1)$ "
  by (auto split: if_splits)

instance ..

end

instantiation modgrp :: abs
begin
```

```

lift_definition abs_modgrp :: "modgrp  $\Rightarrow$  modgrp"
  is "\lambda(a,b,c,d). if c < 0  $\vee$  c = 0  $\wedge$  d < 0 then (-a,-b,-c,-d) else (a,b,c,d)"
  by (auto split: if_splits)

instance ..

end

instantiation modgrp :: times
begin

lift_definition times_modgrp :: "modgrp  $\Rightarrow$  modgrp  $\Rightarrow$  modgrp"
  is "\lambda(a,b,c,d) (a',b',c',d'). (a * a' + b * c', a * b' + b * d', c *
a' + d * c', c * b' + d * d')"
  by (auto simp: algebra_simps)

instance ..
end

instantiation modgrp :: inverse
begin

lift_definition inverse_modgrp :: "modgrp  $\Rightarrow$  modgrp"
  is "\lambda(a, b, c, d). (d, -b, -c, a)"
  by (auto simp: algebra_simps)

definition divide_modgrp :: "modgrp  $\Rightarrow$  modgrp  $\Rightarrow$  modgrp" where
  "divide_modgrp x y = x * inverse y"

instance ..

end

interpretation modgrp: Groups.group "(*)" :: modgrp  $\Rightarrow$  _" 1 inverse
proof
  show "a * b * c = a * (b * c :: modgrp)" for a b c
    by transfer (auto simp: algebra_simps)
next
  show "1 * a = a" for a :: modgrp
    by transfer (auto simp: algebra_simps)
next
  show "inverse a * a = 1" for a :: modgrp
    by transfer (auto simp: algebra_simps)
qed

instance modgrp :: monoid_mult
  by standard (simp_all add: modgrp.assoc)

```

```

lemma inverse_power_modgrp: "inverse (x ^ n :: modgrp) = inverse x ^
n"
  by (induction n) (simp_all add: algebra_simps modgrp.inverse_distrib_swap
power_commuting_commutates)

lemma inverse_mult_assoc1_modgrp [simp]: "inverse f * (f * g) = (g ::
modgrp)"
  by (subst mult.assoc [symmetric]) auto

lemma inverse_mult_assoc2_modgrp [simp]: "f * (inverse f * g) = (g ::
modgrp)"
  by (subst mult.assoc [symmetric]) auto

lemma abs_modgrp_conv_sgn: "abs f = sgn f * (f :: modgrp)"
  by transfer auto

lemma modgrp_conv_sgn_abs: "f = sgn f * (abs f :: modgrp)"
  by transfer auto

lemma sgn_1_modgrp [simp]: "sgn (1 :: modgrp) = 1"
  by transfer auto

lemma sgn_uminus_modgrp [simp]: "sgn (-f :: modgrp) = -sgn f"
  by transfer (auto simp: zmult_eq_1_iff)

lemma sgn_sgn_modgrp [simp]: "sgn (sgn f) = sgn (f :: modgrp)"
  by transfer auto

lemma sgn_abs_modgrp [simp]: "sgn (abs f) = (1 :: modgrp)"
  by transfer auto

lemma abs_1_modgrp [simp]: "abs (1 :: modgrp) = 1"
  by transfer auto

lemma abs_uminus_modgrp [simp]: "abs (-f :: modgrp) = abs f"
  by transfer (auto simp: zmult_eq_1_iff)

lemma abs_sgn_modgrp [simp]: "abs (sgn f) = (1 :: modgrp)"
  by transfer auto

lemma abs_abs_modgrp [simp]: "abs (abs f) = abs (f :: modgrp)"
  by transfer auto

lemma minus_modgrp_neq_self [simp]: "-f ≠ (f :: modgrp)" "f ≠ -f"
  by (transfer; force)+

```

2.3.2 Basic operations

Application to a field

lift_definition *apply_modgrp* :: "modgrp \Rightarrow 'a :: field \Rightarrow 'a" is
 " $\lambda(a,b,c,d). \text{moebius } (\text{of_int } a) (\text{of_int } b) (\text{of_int } c) (\text{of_int } d)$ " .

The shift operation $z \mapsto z + n$

lift_definition *shift_modgrp* :: "int \Rightarrow modgrp" is " $\lambda n. (1, n, 0, 1)$ "
 by auto

The shift operation $z \mapsto z + 1$

lift_definition *T_modgrp* :: modgrp is "(1, 1, 0, 1)"
 by auto

The operation $z \mapsto -\frac{1}{z}$

lift_definition *S_modgrp* :: modgrp is "(0, -1, 1, 0)"
 by auto

Whether or not the transformation has a pole in the complex plane

lift_definition *is_singular_modgrp* :: "modgrp \Rightarrow bool" is " $\lambda(a,b,c,d). c \neq 0$ " .

The position of the transformation's pole in the complex plane (if it has one)

lift_definition *pole_modgrp* :: "modgrp \Rightarrow 'a :: field" is " $\lambda(a,b,c,d). -\text{of_int } d / \text{of_int } c$ " .

lemma *pole_modgrp_in_Rats*: "pole_modgrp $f \in (\mathbb{Q} :: 'a :: \text{real_field set})$ "
 by transfer auto

lemma *pole_modgrp_in_Reals*: "pole_modgrp $f \in (\mathbb{R} :: 'a :: \text{real_field set})$ "
 by transfer auto

lemma *Im_pole_modgrp [simp]*: "Im (pole_modgrp f) = 0"
 by transfer auto

The complex number to which complex infinity is mapped by the transformation. This is undefined if the transformation maps complex infinity to itself.

lift_definition *pole_image_modgrp* :: "modgrp \Rightarrow 'a :: field" is " $\lambda(a,b,c,d). \text{of_int } a / \text{of_int } c$ " .

lemma *Im_pole_image_modgrp [simp]*: "Im (pole_image_modgrp f) = 0"
 by transfer auto

The normalised coefficients of the transformation. The convention that is chosen is that c is always non-negative, and if c is zero then d is positive.

```

lift_definition modgrp_a :: "modgrp  $\Rightarrow$  int" is " $\lambda(a,b,c,d). a$ " .
lift_definition modgrp_b :: "modgrp  $\Rightarrow$  int" is " $\lambda(a,b,c,d). b$ " .
lift_definition modgrp_c :: "modgrp  $\Rightarrow$  int" is " $\lambda(a,b,c,d). c$ " .
lift_definition modgrp_d :: "modgrp  $\Rightarrow$  int" is " $\lambda(a,b,c,d). d$ " .

lemma modgrp_abcd_S [simp]:
  "modgrp_a S_modgrp = 0" "modgrp_b S_modgrp = -1" "modgrp_c S_modgrp
= 1" "modgrp_d S_modgrp = 0"
  by (transfer; simp)+

lemma modgrp_abcd_T [simp]:
  "modgrp_a T_modgrp = 1" "modgrp_b T_modgrp = 1" "modgrp_c T_modgrp =
0" "modgrp_d T_modgrp = 1"
  by (transfer; simp)+

lemma modgrp_abcd_shift [simp]:
  "modgrp_a (shift_modgrp n) = 1" "modgrp_b (shift_modgrp n) = n"
  "modgrp_c (shift_modgrp n) = 0" "modgrp_d (shift_modgrp n) = 1"
  by (transfer; simp)+

lemma modgrp_c_shift_left [simp]:
  "modgrp_c (shift_modgrp n * f) = modgrp_c f"
  by transfer auto

lemma modgrp_d_shift_left [simp]:
  "modgrp_d (shift_modgrp n * f) = modgrp_d f"
  by transfer auto

lemma modgrp_abcd_det: "modgrp_a x * modgrp_d x - modgrp_b x * modgrp_c
x = 1"
  by transfer auto

lemma modgrp_a_nz_or_b_nz: "modgrp_a x  $\neq$  0  $\vee$  modgrp_b x  $\neq$  0"
  by transfer auto

lemma modgrp_c_nz_or_d_nz: "modgrp_c x  $\neq$  0  $\vee$  modgrp_d x  $\neq$  0"
  by transfer auto

lemma apply_modgrp_altdef:
  "(apply_modgrp x :: 'a :: field  $\Rightarrow$  _) =
  moebius (of_int (modgrp_a x)) (of_int (modgrp_b x)) (of_int (modgrp_c
x)) (of_int (modgrp_d x))"
proof (transfer, goal_cases)
  case (1 x)
  thus ?case
    by (auto simp: case_prod_unfold moebius_uminus')
qed

lemma sgn_modgrp_altdef:

```

```

"sgn f = (if modgrp_c f < 0 ∨ modgrp_c f = 0 ∧ modgrp_d f < 0 then
-1 else 1)"
by transfer auto

```

```

lemma abs_modgrp_altdef:
"abs f = (if modgrp_c f < 0 ∨ modgrp_c f = 0 ∧ modgrp_d f < 0 then
-f else f)"
by transfer auto

```

```

lemma abs_eq_modgrpE:
assumes "abs f = (g :: modgrp)"
obtains "f = g" | "f = -g"
using assms that by (auto simp: abs_modgrp_altdef split: if_splits)

```

Converting a quadruple of numbers into an element of the modular group.

```

lift_definition modgrp :: "int ⇒ int ⇒ int ⇒ int ⇒ modgrp" is
"λa b c d. if a * d - b * c = 1 then (a, b, c, d) else (1, 0, 0, 1)"
by transfer auto

```

```

lemma modgrp_eq_iff:
"f = g ⟷ modgrp_a f = modgrp_a g ∧ modgrp_b f = modgrp_b g ∧
modgrp_c f = modgrp_c g ∧ modgrp_d f = modgrp_d g"
by transfer auto

```

```

lemma modgrp_wrong: "a * d - b * c ≠ 1 ⟹ modgrp a b c d = 1"
by transfer auto

```

```

lemma modgrp_abcd [simp]: "modgrp (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x) = x"
by transfer auto

```

```

lemma
assumes "a * d - b * c = 1"
shows modgrp_a_modgrp: "modgrp_a (modgrp a b c d) = a"
and modgrp_b_modgrp: "modgrp_b (modgrp a b c d) = b"
and modgrp_c_modgrp: "modgrp_c (modgrp a b c d) = c"
and modgrp_d_modgrp: "modgrp_d (modgrp a b c d) = d"
using assms by (transfer; simp; fail)+

```

```

lemmas modgrp_abcd_modgrp = modgrp_a_modgrp modgrp_b_modgrp modgrp_c_modgrp
modgrp_d_modgrp

```

2.3.3 Basic properties

```

lemma continuous_on_apply_modgrp [continuous_intros]:
fixes g :: "'a :: topological_space ⇒ 'b :: real_normed_field"
assumes "continuous_on A g" "∧z. z ∈ A ⟹ ¬is_singular_modgrp f ∨
g z ≠ pole_modgrp f"
shows "continuous_on A (λz. apply_modgrp f (g z))"

```

```

using assms
by transfer (auto intro!: continuous_intros)

lemma holomorphic_on_apply_modgrp [holomorphic_intros]:
  assumes "g holomorphic_on A" "\z. z \in A \implies \neg is_singular_modgrp f
\vee g z \neq pole_modgrp f"
  shows "(lambda z. apply_modgrp f (g z)) holomorphic_on A"
  using assms
  by transfer (auto intro!: holomorphic_intros)

lemma analytic_on_apply_modgrp [analytic_intros]:
  assumes "g analytic_on A" "\z. z \in A \implies \neg is_singular_modgrp f \vee
g z \neq pole_modgrp f"
  shows "(lambda z. apply_modgrp f (g z)) analytic_on A"
  using assms
  by transfer (auto intro!: analytic_intros)

lemma isCont_apply_modgrp [continuous_intros]:
  fixes z :: "'a :: real_normed_field"
  assumes "\neg is_singular_modgrp f \vee z \neq pole_modgrp f"
  shows "isCont (apply_modgrp f) z"
proof -
  define S where "S = (if is_singular_modgrp f then -{pole_modgrp f}
else (UNIV :: 'a set))"
  have "continuous_on S (apply_modgrp f)"
    unfolding S_def by (intro continuous_intros) auto
  moreover have "open S" "z \in S"
    using assms by (auto simp: S_def)
  ultimately show ?thesis
    using continuous_on_eq_continuous_at by blast
qed

lemmas tendsto_apply_modgrp [tendsto_intros] = isCont_tendsto_compose[OF
isCont_apply_modgrp]

lift_definition diff_scale_factor_modgrp :: "modgrp \Rightarrow 'a :: field \Rightarrow 'a
\Rightarrow 'a" is
  "\lambda(a,b,c,d) w z. (of_int c * w + of_int d) * (of_int c * z + of_int
d)" .

lemma diff_scale_factor_modgrp_commutates:
  "diff_scale_factor_modgrp f w z = diff_scale_factor_modgrp f z w"
  by transfer (simp add: case_prod_unfold)

lemma diff_scale_factor_modgrp_zero_iff:
  fixes w z :: "'a :: field_char_0"
  shows "diff_scale_factor_modgrp f w z = 0 \iff is_singular_modgrp f
\wedge pole_modgrp f \in \{w, z\}"
  by transfer

```

```

(auto simp: case_prod_unfold divide_simps add_eq_0_iff mult.commute

      minus_equation_iff[of "of_int x" for x])

lemma apply_modgrp_diff_eq:
  fixes g :: modgrp
  defines "f ≡ apply_modgrp g"
  assumes *: "¬is_singular_modgrp g ∨ pole_modgrp g ∉ {w, z}"
  shows "f w - f z = (w - z) / diff_scale_factor_modgrp g w z"
  unfolding f_def using *
  by transfer
      (auto simp: moebius_diff_eq zmult_eq_1_iff simp flip: of_int_diff
of_int_mult split: if_splits)

lemma norm_modgrp_dividend_ge:
  fixes z :: complex
  shows "norm (of_int c * z + of_int d) ≥ |c * Im z|"
proof -
  define x y where "x = Re z" and "y = Im z"
  have z_eq: "z = Complex x y"
    by (simp add: x_def y_def)
  have "(real_of_int c * y) ^ 2 ≤ (real_of_int c * x + real_of_int d)
^ 2 + (real_of_int c * y) ^ 2"
    by simp
  also have "... = norm (of_int c * z + of_int d) ^ 2"
    by (simp add: cmod_power2 x_def y_def)
  finally show "norm (of_int c * z + of_int d) ≥ |c * y|"
    by (metis abs_le_square_iff abs_norm_cancel)
qed

lemma diff_scale_factor_modgrp_altdef:
  fixes g :: modgrp
  defines "c ≡ modgrp_c g" and "d ≡ modgrp_d g"
  shows "diff_scale_factor_modgrp g w z = (of_int c * w + of_int d) *
(of_int c * z + of_int d)"
  unfolding c_def d_def by transfer (auto simp: algebra_simps)

lemma norm_diff_scale_factor_modgrp_ge_complex:
  fixes w z :: complex
  assumes "w ≠ z"
  shows "norm (diff_scale_factor_modgrp g w z) ≥ of_int (modgrp_c g)
^ 2 * |Im w * Im z|"
proof -
  have "norm (diff_scale_factor_modgrp g w z) ≥
|of_int (modgrp_c g) * Im w| * |of_int (modgrp_c g) * Im z|"
    unfolding diff_scale_factor_modgrp_altdef norm_mult
    by (intro mult_mono norm_modgrp_dividend_ge) auto
  thus ?thesis
    by (simp add: algebra_simps abs_mult power2_eq_square)

```

qed

```
lemma apply_uminus_modgrp [simp]: "apply_modgrp (-f) z = apply_modgrp
f z"
  by transfer auto
```

```
lemma apply_shift_modgrp [simp]: "apply_modgrp (shift_modgrp n) z = z
+ of_int n"
  by transfer (auto simp: moebius_def)
```

```
lemma apply_modgrp_T [simp]: "apply_modgrp T_modgrp z = z + 1"
  by transfer (auto simp: moebius_def)
```

```
lemma apply_modgrp_S [simp]: "apply_modgrp S_modgrp z = -1 / z"
  by transfer (auto simp: moebius_def)
```

```
lemma apply_modgrp_1 [simp]: "apply_modgrp 1 z = z"
  by transfer (auto simp: moebius_def)
```

```
lemma apply_modgrp_mult_aux:
  fixes z :: "'a :: field_char_0"
  assumes ns: "c' = 0  $\vee$  z  $\neq$  -d' / c'"
  assumes det: "a * d - b * c = 1" "a' * d' - b' * c' = 1"
  shows "moebius a b c d (moebius a' b' c' d' z) =
        moebius (a * a' + b * c') (a * b' + b * d')
              (c * a' + d * c') (c * b' + d * d') z"
```

proof -

```
  have det': "c  $\neq$  0  $\vee$  d  $\neq$  0" "c'  $\neq$  0  $\vee$  d'  $\neq$  0"
    using det by auto
  from assms have nz: "c' * z + d'  $\neq$  0"
    by (auto simp add: divide_simps add_eq_0_iff split: if_splits)
  show ?thesis using det nz
    by (simp add: moebius_def divide_simps) (auto simp: algebra_simps)
```

qed

```
lemma apply_modgrp_mult:
  fixes z :: "'a :: field_char_0"
  assumes "¬is_singular_modgrp y  $\vee$  z  $\neq$  pole_modgrp y"
  shows "apply_modgrp (x * y) z = apply_modgrp x (apply_modgrp y z)"
  using assms
proof (transfer, goal_cases)
  case (1 y z x)
  obtain a b c d where [simp]: "x = (a, b, c, d)"
    using prod_cases4 by blast
  obtain a' b' c' d' where [simp]: "y = (a', b', c', d')"
    using prod_cases4 by blast
  show ?case
    using apply_modgrp_mult_aux[of "of_int c'" z "of_int d'" "of_int a"
"of_int d"]
```

```

"of_int b" "of_int c" "of_int a'" "of_int
b'] 1
  by (simp flip: of_int_mult of_int_add of_int_diff of_int_minus)
qed

lemma apply_modgrp_eq_apply_modgrp_iff:
  assumes "Im w > 0" "Im z > 0"
  shows "apply_modgrp f w = apply_modgrp g z  $\longleftrightarrow$  apply_modgrp (inverse
g * f) w = z"
proof
  assume "apply_modgrp f w = apply_modgrp g z"
  hence "apply_modgrp (inverse g) (apply_modgrp f w) = apply_modgrp (inverse
g) (apply_modgrp g z)"
    by (rule arg_cong)
  thus "apply_modgrp (inverse g * f) w = z"
    using assms by (subst (asm) (1 2) apply_modgrp_mult [symmetric]) auto
next
  assume "apply_modgrp (inverse g * f) w = z"
  hence "apply_modgrp g (apply_modgrp (inverse g * f) w) = apply_modgrp
g z"
    by (rule arg_cong)
  thus "apply_modgrp f w = apply_modgrp g z"
    using assms by (subst (asm) apply_modgrp_mult [symmetric]) auto
qed

lemma apply_modgrp_sgn [simp]: "apply_modgrp (sgn f) = ( $\lambda$ x. x)"
  by (auto simp: sgn_modgrp_altdef)

lemma apply_modgrp_abs [simp]: "apply_modgrp (abs f) = apply_modgrp f"
  by (auto simp: abs_modgrp_altdef)

lemma is_singular_modgrp_altdef: "is_singular_modgrp x  $\longleftrightarrow$  modgrp_c
x  $\neq$  0"
  by transfer (auto split: if_splits)

lemma not_is_singular_modgrpD:
  assumes " $\neg$ is_singular_modgrp x"
  shows "abs x = shift_modgrp (sgn (modgrp_a x) * modgrp_b x)"
  using assms
proof (transfer, goal_cases)
  case (1 x)
  obtain a b c d where [simp]: "x = (a, b, c, d)"
    using prod_cases4 by blast
  from 1 have [simp]: "c = 0"
    by auto
  from 1 have "a = 1  $\wedge$  d = 1  $\vee$  a = -1  $\wedge$  d = -1"
    by (auto simp: zmult_eq_1_iff)
  thus ?case
    by auto

```

qed

```
lemma is_singular_modgrp_inverse [simp]: "is_singular_modgrp (inverse
x)  $\longleftrightarrow$  is_singular_modgrp x"
  by transfer auto
```

```
lemma is_singular_modgrp_S_left_iff [simp]: "is_singular_modgrp (S_modgrp
* f)  $\longleftrightarrow$  modgrp_a f  $\neq$  0"
  by transfer (auto split: if_splits)
```

```
lemma is_singular_modgrp_S_right_iff [simp]: "is_singular_modgrp (f *
S_modgrp)  $\longleftrightarrow$  modgrp_d f  $\neq$  0"
  by transfer (auto split: if_splits)
```

```
lemma is_singular_modgrp_T_left_iff [simp]:
  "is_singular_modgrp (T_modgrp * f)  $\longleftrightarrow$  is_singular_modgrp f"
  by transfer (auto split: if_splits)
```

```
lemma is_singular_modgrp_T_right_iff [simp]:
  "is_singular_modgrp (f * T_modgrp)  $\longleftrightarrow$  is_singular_modgrp f"
  by transfer (auto split: if_splits)
```

```
lemma is_singular_modgrp_shift_left_iff [simp]:
  "is_singular_modgrp (shift_modgrp n * f)  $\longleftrightarrow$  is_singular_modgrp f"
  by transfer (auto split: if_splits)
```

```
lemma is_singular_modgrp_shift_right_iff [simp]:
  "is_singular_modgrp (f * shift_modgrp n)  $\longleftrightarrow$  is_singular_modgrp f"
  by transfer (auto split: if_splits)
```

```
lemma pole_modgrp_inverse [simp]: "pole_modgrp (inverse x) = pole_image_modgrp
x"
  by transfer auto
```

```
lemma pole_image_modgrp_inverse [simp]: "pole_image_modgrp (inverse x)
= pole_modgrp x"
  by transfer auto
```

```
lemma pole_image_modgrp_in_Reals: "pole_image_modgrp x  $\in$  ( $\mathbb{R}$  :: 'a ::
{real_field, field} set)"
  by transfer (auto intro!: Reals_divide)
```

```
lemma apply_modgrp_inverse_eqI:
  fixes x y :: "'a :: field_char_0"
  assumes " $\neg$ is_singular_modgrp f  $\vee$  y  $\neq$  pole_modgrp f" "apply_modgrp
f y = x"
  shows "apply_modgrp (inverse f) x = y"
proof -
  have "apply_modgrp (inverse f) x = apply_modgrp (inverse f * f) y"
```

```

    using assms by (subst apply_modgrp_mult) auto
  also have "... = y"
    by simp
  finally show ?thesis .
qed

lemma apply_modgrp_eq_iff [simp]:
  fixes x y :: "'a :: field_char_0"
  assumes "¬is_singular_modgrp f ∨ x ≠ pole_modgrp f ∧ y ≠ pole_modgrp
  f"
  shows "apply_modgrp f x = apply_modgrp f y ↔ x = y"
  using assms by (metis apply_modgrp_inverse_eqI)

lemma is_singular_modgrp_times_aux:
  assumes det: "a * d - b * c = 1" "a' * d' - b' * (c' :: int) = 1"
  shows "(c * a' + d * c' ≠ 0) ↔ ((c = 0 → c' ≠ 0) ∧ (c = 0 ∨
  c' = 0 ∨ -d * c' ≠ a' * c))"
  using assms by Groebner_Basis.algebra

lemma is_singular_modgrp_times_iff:
  "is_singular_modgrp (x * y) ↔
  (is_singular_modgrp x ∨ is_singular_modgrp y) ∧
  (¬is_singular_modgrp x ∨ ¬is_singular_modgrp y ∨ pole_modgrp x
  ≠ (pole_image_modgrp y :: real))"
proof (transfer, goal_cases)
  case (1 x y)
  obtain a b c d where [simp]: "x = (a, b, c, d)"
    using prod_cases4 by blast
  obtain a' b' c' d' where [simp]: "y = (a', b', c', d')"
    using prod_cases4 by blast
  show ?case
    using is_singular_modgrp_times_aux[of a d b c a' d' b' c'] 1
    by (auto simp: field_simps simp flip: of_int_mult of_int_add of_int_minus
  of_int_diff)
qed

lemma is_singular_modgrp_uminus_iff [simp]: "is_singular_modgrp (-f)
  ↔ is_singular_modgrp f"
  by transfer auto

lemma times_modgrp_uminus_left [simp]: "(-f :: modgrp) * g = -(f * g)"
  by transfer auto

lemma times_modgrp_uminus_right [simp]: "(f :: modgrp) * (-g) = -(f *
  g)"
  by transfer auto

lemma inverse_modgrp_uminus [simp]: "inverse (-f :: modgrp) = -inverse
  f"

```

```

    by transfer auto

lemma shift_modgrp_1: "shift_modgrp 1 = T_modgrp"
  by transfer auto

lemma shift_modgrp_eq_iff: "shift_modgrp n = shift_modgrp m  $\longleftrightarrow$  n = m"
  by transfer auto

lemma shift_modgrp_neq_S [simp]: "shift_modgrp n  $\neq$  S_modgrp"
  by transfer auto

lemma S_neq_shift_modgrp [simp]: "S_modgrp  $\neq$  shift_modgrp n"
  by transfer auto

lemma shift_modgrp_eq_T_iff [simp]: "shift_modgrp n = T_modgrp  $\longleftrightarrow$  n = 1"
  by transfer auto

lemma T_eq_shift_modgrp_iff [simp]: "T_modgrp = shift_modgrp n  $\longleftrightarrow$  n = 1"
  by transfer auto

lemma shift_modgrp_0 [simp]: "shift_modgrp 0 = 1"
  by transfer auto

lemma shift_modgrp_add: "shift_modgrp (m + n) = shift_modgrp m * shift_modgrp n"
  by transfer auto

lemma shift_modgrp_minus: "shift_modgrp (-m) = inverse (shift_modgrp m)"
proof -
  have "1 = shift_modgrp (m + (-m))"
    by simp
  also have "shift_modgrp (m + (-m)) = shift_modgrp m * shift_modgrp (-m)"
    by (subst shift_modgrp_add) auto
  finally show ?thesis
    by (simp add: modgrp.inverse_unique)
qed

lemma shift_modgrp_power: "shift_modgrp n ^ m = shift_modgrp (n * m)"
  by (induction m) (auto simp: algebra_simps shift_modgrp_add)

lemma shift_modgrp_power_int: "shift_modgrp n powi m = shift_modgrp (n * m)"
  by (cases "m  $\geq$  0")
    (auto simp: power_int_def shift_modgrp_power simp flip: shift_modgrp_minus)

```

```

lemma shift_shift_modgrp: "shift_modgrp n * (shift_modgrp m * x) = shift_modgrp
(n + m) * x"
  by (simp add: mult.assoc shift_modgrp_add)

lemma shift_modgrp_conv_T_power: "shift_modgrp n = T_modgrp powi n"
  by (simp flip: shift_modgrp_1 add: shift_modgrp_power_int)

lemma modgrp_S_S [simp]: "S_modgrp * S_modgrp = -1"
  by transfer auto

lemma inverse_S_modgrp [simp]: "inverse S_modgrp = -S_modgrp"
  by (rule modgrp.inverse_unique) simp

lemma modgrp_S_S' [simp]: "S_modgrp * (S_modgrp * x) = -x"
  by (subst mult.assoc [symmetric], subst modgrp_S_S) simp

lemma not_is_singular_1_modgrp [simp]: "¬is_singular_modgrp 1"
  by transfer auto

lemma not_is_singular_T_modgrp [simp]: "¬is_singular_modgrp T_modgrp"
  by transfer auto

lemma not_is_singular_shift_modgrp [simp]: "¬is_singular_modgrp (shift_modgrp
n)"
  by transfer auto

lemma is_singular_S_modgrp [simp]: "is_singular_modgrp S_modgrp"
  by transfer auto

lemma pole_modgrp_S [simp]: "pole_modgrp S_modgrp = 0"
  by transfer auto

lemma pole_modgrp_1 [simp]: "pole_modgrp 1 = 0"
  by transfer auto

lemma pole_modgrp_T [simp]: "pole_modgrp T_modgrp = 0"
  by transfer auto

lemma pole_modgrp_shift [simp]: "pole_modgrp (shift_modgrp n) = 0"
  by transfer auto

lemma pole_image_modgrp_1 [simp]: "pole_image_modgrp 1 = 0"
  by transfer auto

lemma pole_image_modgrp_T [simp]: "pole_image_modgrp T_modgrp = 0"
  by transfer auto

```

```

lemma pole_image_modgrp_shift [simp]: "pole_image_modgrp (shift_modgrp
n) = 0"
  by transfer auto

lemma pole_image_modgrp_S [simp]: "pole_image_modgrp S_modgrp = 0"
  by transfer auto

lemma minus_minus_power2_eq: "(-x - y :: 'a :: ring_1) ^ 2 = (x + y)
^ 2"
  by (simp add: algebra_simps power2_eq_square)

lemma modgrp_a_1 [simp]: "modgrp_a 1 = 1"
  and modgrp_b_1 [simp]: "modgrp_b 1 = 0"
  and modgrp_c_1 [simp]: "modgrp_c 1 = 0"
  and modgrp_d_1 [simp]: "modgrp_d 1 = 1"
  by (transfer; simp; fail)+

lemma not_singular_modgrpD:
  assumes "~is_singular_modgrp f"
  shows "abs f = shift_modgrp (modgrp_b (abs f))"
  using assms by transfer (auto simp: zmult_eq_1_iff)

lemma S_conv_modgrp: "S_modgrp = modgrp 0 (-1) 1 0"
  and T_conv_modgrp: "T_modgrp = modgrp 1 1 0 1"
  and shift_conv_modgrp: "shift_modgrp n = modgrp 1 n 0 1"
  and one_conv_modgrp: "1 = modgrp 1 0 0 1"
  by (transfer; simp; fail)+

lemma modgrp_times:
  assumes "a * d - b * c = 1"
  assumes "a' * d' - b' * c' = 1"
  shows "modgrp a b c d * modgrp a' b' c' d' =
  modgrp (a * a' + b * c') (a * b' + b * d') (c * a' + d * c')
(c * b' + d * d')"
  using assms by transfer (auto simp: algebra_simps)

lemma modgrp_uminus: "a * d - b * c = 1  $\implies$  -modgrp a b c d = modgrp
(-a) (-b) (-c) (-d)"
  by transfer auto

lemma modgrp_inverse:
  assumes "a * d - b * c = 1"
  shows "inverse (modgrp a b c d) = modgrp d (-b) (-c) a"
  by (intro modgrp.inverse_unique, subst modgrp_times)
  (use assms in <auto simp: algebra_simps one_conv_modgrp>)

lemma modgrp_a_mult_shift [simp]: "modgrp_a (f * shift_modgrp m) = modgrp_a
f"

```

```

    by transfer auto

lemma modgrp_b_mult_shift [simp]: "modgrp_b (f * shift_modgrp m) = modgrp_a
f * m + modgrp_b f"
  by transfer auto

lemma modgrp_c_mult_shift [simp]: "modgrp_c (f * shift_modgrp m) = modgrp_c
f"
  by transfer auto

lemma modgrp_d_mult_shift [simp]: "modgrp_d (f * shift_modgrp m) = modgrp_c
f * m + modgrp_d f"
  by transfer auto

lemma coprime_modgrp_c_d: "coprime (modgrp_c f) (modgrp_d f)"
proof -
  define a b c d where "a = modgrp_a f" "b = modgrp_b f" "c = modgrp_c
f" "d = modgrp_d f"
  have "[a * 0 - b * c = a * d - b * c] (mod d)"
    by (intro cong_diff cong_refl cong_mult) (auto simp: Cong.cong_def)
  also have "a * d - b * c = 1"
    unfolding a_b_c_d_def modgrp_abcd_det[of f] by simp
  finally have "[c * (-b) = 1] (mod d)"
    by (simp add: mult_ac)
  thus "coprime c d"
    by (subst coprime_iff_invertible_int) (auto intro!: exI[of _ "-b"])
qed

context unimodular_moebius_transform
begin

lift_definition as_modgrp :: modgrp is "(a, b, c, d)"
  using unimodular by auto

lemma as_modgrp_altdef: "as_modgrp = modgrp a b c d"
  using unimodular by transfer auto

lemma  $\varphi$ _as_modgrp: " $\varphi$  = apply_modgrp as_modgrp"
  unfolding  $\varphi$ _def by transfer simp

end

interpretation modgrp: unimodular_moebius_transform "modgrp_a x" "modgrp_b
x" "modgrp_c x" "modgrp_d x"
  rewrites "modgrp.as_modgrp = x" and "modgrp. $\varphi$  = apply_modgrp x"
proof -
  show *: "unimodular_moebius_transform (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x)"
    by unfold_locales (rule modgrp_abcd_det)

```

```

show "unimodular_moebius_transform.as_modgrp (modgrp_a x) (modgrp_b
x) (modgrp_c x) (modgrp_d x) = x"
  by (subst unimodular_moebius_transform.as_modgrp_altdef[OF *]) auto
show "unimodular_moebius_transform.φ (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x) = apply_modgrp x"
  by (subst unimodular_moebius_transform.φ_def[OF *], subst apply_modgrp_altdef)
auto
qed

```

2.4 Code generation

```
code_datatype modgrp
```

```

lemma one_modgrp_code [code]: "1 = modgrp 1 0 0 1"
and S_modgrp_code [code]: "S_modgrp = modgrp 0 (-1) 1 0"
and T_modgrp_code [code]: "T_modgrp = modgrp 1 1 0 1"
and shift_modgrp_code [code]: "shift_modgrp n = modgrp 1 n 0 1"
by (simp_all add: one_conv_modgrp S_conv_modgrp T_conv_modgrp shift_conv_modgrp)

```

```

lemma inverse_modgrp_code [code]: "inverse (modgrp a b c d) = modgrp
d (-b) (-c) a"
  by (cases "a * d - b * c = 1")
  (auto simp: modgrp_inverse modgrp_wrong algebra_simps)

```

```

lemma modgrp_a_uminus [simp]: "modgrp_a (-f) = -modgrp_a f"
and modgrp_b_uminus [simp]: "modgrp_b (-f) = -modgrp_b f"
and modgrp_c_uminus [simp]: "modgrp_c (-f) = -modgrp_c f"
and modgrp_d_uminus [simp]: "modgrp_d (-f) = -modgrp_d f"
by (transfer; force)+

```

```

lemma modgrp_a_inverse [simp]: "modgrp_a (inverse f) = modgrp_d f"
and modgrp_b_inverse [simp]: "modgrp_b (inverse f) = -modgrp_b f"
and modgrp_c_inverse [simp]: "modgrp_c (inverse f) = -modgrp_c f"
and modgrp_d_inverse [simp]: "modgrp_d (inverse f) = modgrp_a f"
by (transfer; force)+

```

```

lemma times_modgrp_code [code]:
"modgrp a b c d * modgrp a' b' c' d' = (
  if a * d - b * c ≠ 1 then modgrp a' b' c' d'
  else if a' * d' - b' * c' ≠ 1 then modgrp a b c d
  else modgrp (a * a' + b * c') (a * b' + b * d') (c * a' + d * c')
(c * b' + d * d'))"
  by (simp add: modgrp_times modgrp_wrong)

```

```

lemma modgrp_a_times [simp]: "modgrp_a (f * g) = modgrp_a f * modgrp_a
g + modgrp_b f * modgrp_c g"
and modgrp_b_times [simp]: "modgrp_b (f * g) = modgrp_a f * modgrp_b
g + modgrp_b f * modgrp_d g"
and modgrp_c_times [simp]: "modgrp_c (f * g) = modgrp_c f * modgrp_a

```

```

g + modgrp_d f * modgrp_c g"
  and modgrp_d_times [simp]: "modgrp_d (f * g) = modgrp_c f * modgrp_b
g + modgrp_d f * modgrp_d g"
  by (transfer; force)+

lemma modgrp_a_code [code]:
  "modgrp_a (modgrp a b c d) = (if a * d - b * c = 1 then a else 1)"
  by transfer auto

lemma modgrp_b_code [code]:
  "modgrp_b (modgrp a b c d) = (if a * d - b * c = 1 then b else 0)"
  by transfer auto

lemma modgrp_c_code [code]:
  "modgrp_c (modgrp a b c d) = (if a * d - b * c = 1 then c else 0)"
  by transfer auto

lemma modgrp_d_code [code]:
  "modgrp_d (modgrp a b c d) = (if a * d - b * c = 1 then d else 1)"
  by transfer auto

lemma sgn_modgrp_code [code]:
  "sgn (modgrp a b c d) = (if a * d - b * c = 1 ∧ (c < 0 ∨ c = 0 ∧ d
< 0) then -1 else 1)"
  by (auto simp: sgn_modgrp_altdef modgrp_a_code modgrp_b_code modgrp_c_code
modgrp_d_code)

lemma abs_modgrp_code [code]:
  "abs (modgrp a b c d) =
  (if a * d - b * c = 1 ∧ (c < 0 ∨ c = 0 ∧ d < 0) then -modgrp a b
c d else modgrp a b c d)"
  by (auto simp: abs_modgrp_altdef modgrp_a_code modgrp_b_code modgrp_c_code
modgrp_d_code)

lemma apply_modgrp_code [code]:
  "apply_modgrp (modgrp a b c d) z =
  (if a * d - b * c ≠ 1 then z else (of_int a * z + of_int b) / (of_int
c * z + of_int d))"
  by transfer (auto simp: moebius_def)

lemma is_singular_modgrp_code [code]:
  "is_singular_modgrp (modgrp a b c d) ↔ a * d - b * c = 1 ∧ c ≠ 0"
  by transfer auto

lemma pole_modgrp_code [code]:
  "pole_modgrp (modgrp a b c d) = (if a * d - b * c = 1 then -of_int d
/ of_int c else 0)"
  by transfer auto

```

```

lemma pole_image_modgrp_code [code]:
  "pole_image_modgrp (modgrp a b c d) =
    (if a * d - b * c = 1 ∧ c ≠ 0 then of_int a / of_int c else 0)"
  by transfer auto

```

2.5 The factor of automorphy and the slash operator

The following will be needed to define the slash operator and it appears in a few other places as well. Diamond and Shurman call it the *factor of automorphy* of a modular transformation.

```

lift_definition automorphy_factor :: "modgrp ⇒ complex ⇒ complex" is
  "λ(a,b,c,d) z. of_int c * z + of_int d" .

```

```

lemma automorphy_factor_altdef:
  "automorphy_factor g z = of_int (modgrp_c g) * z + of_int (modgrp_d
g)"
  by transfer auto

```

```

lemma automorphy_factor_uminus [simp]: "automorphy_factor (-h) z = -automorphy_factor
h z"
  by (auto simp: automorphy_factor_altdef)

```

```

lemma automorphy_factor_1 [simp]: "automorphy_factor 1 z = 1"
  by (auto simp: automorphy_factor_altdef)

```

```

lemma automorphy_factor_shift [simp]: "automorphy_factor (shift_modgrp
n) z = 1"
  by (simp add: automorphy_factor_altdef)

```

```

lemma automorphy_factor_T [simp]: "automorphy_factor T_modgrp z = 1"
  by (simp add: automorphy_factor_altdef)

```

```

lemma automorphy_factor_S [simp]: "automorphy_factor S_modgrp z = z"
  by (simp add: automorphy_factor_altdef)

```

```

lemma automorphy_factor_shift_right [simp]:
  "automorphy_factor (f * shift_modgrp n) z = automorphy_factor f (z +
of_int n)"
  unfolding automorphy_factor_def
  by transfer (auto simp: algebra_simps)

```

```

lemma automorphy_factor_shift_left [simp]:
  "automorphy_factor (shift_modgrp n * f) z = automorphy_factor f z"
  by (simp add: automorphy_factor_altdef)

```

```

lemma automorphy_factor_T_right [simp]:
  "automorphy_factor (f * T_modgrp) z = automorphy_factor f (z + 1)"
  unfolding shift_modgrp_1 [symmetric] by (subst automorphy_factor_shift_right)
auto

```

```

lemma automorphy_factor_T_left [simp]:
  "automorphy_factor (T_modgrp * f) z = automorphy_factor f z"
  unfolding shift_modgrp_1 [symmetric] by (subst automorphy_factor_shift_left)
auto

```

```

lemma automorphy_factor_nonzero [simp]:
  assumes "Im z  $\neq$  0"
  shows "automorphy_factor g z  $\neq$  0"
proof -
  define c d where "c = modgrp_c g" and "d = modgrp_d g"
  have "c  $\neq$  0  $\vee$  d  $\neq$  0"
    unfolding c_def d_def using modgrp_c_nz_or_d_nz by blast
  have "of_int c * z + of_int d  $\neq$  0"
    using assms <c  $\neq$  0  $\vee$  d  $\neq$  0> by (auto simp: complex_eq_iff)
  thus ?thesis
    by (simp add: automorphy_factor_altdef c_def d_def)
qed

```

```

lemma automorphy_factor_mult:
  assumes "Im z  $\neq$  0"
  shows "automorphy_factor (f * g) z =
    automorphy_factor f (apply_modgrp g z) * automorphy_factor
g z"
proof -
  define x where "x = (of_int (modgrp_a g) * z + of_int (modgrp_b g))"
  define y where "y = automorphy_factor g z"
  have "y  $\neq$  0"
    using assms by (auto simp: y_def)
  have "automorphy_factor f (apply_modgrp g z) * automorphy_factor g z
=
  (of_int (modgrp_c f) * x / y + of_int (modgrp_d f)) * y"
    by (auto simp: x_def y_def automorphy_factor_altdef apply_modgrp_altdef
moebius_def)
  also have "... = of_int (modgrp_c f) * x + of_int (modgrp_d f) * y"
    using <y  $\neq$  0> by (auto simp: field_simps)
  also have "... = automorphy_factor (f * g) z"
    by (auto simp: automorphy_factor_altdef x_def y_def algebra_simps)
  finally show ?thesis ..
qed

```

```

lemma has_field_derivative_automorphy_factor [derivative_intros]:
  assumes "(f has_field_derivative f') (at x within A)"
  shows "(( $\lambda$ x. automorphy_factor g (f x)) has_field_derivative (of_int
(modgrp_c g) * f')) (at x within A)"
  unfolding automorphy_factor_altdef by (auto intro!: derivative_eq_intros
assms)

```

```

lemma automorphy_factor_analytic [analytic_intros]: "automorphy_factor

```

```

g analytic_on A"
  unfolding automorphy_factor_altdef by (auto simp: automorphy_factor_def
intro!: analytic_intros)

lemma automorphy_factor_meromorphic [meromorphic_intros]: "automorphy_factor
h meromorphic_on A"
  unfolding automorphy_factor_altdef by (auto intro!: meromorphic_intros)

lemma tendsto_automorphy_factor [tendsto_intros]:
"(f  $\longrightarrow$  c) F  $\implies$  (( $\lambda$ x. automorphy_factor g (f x))  $\longrightarrow$  automorphy_factor
g c) F"
  unfolding automorphy_factor_altdef by (auto intro!: tendsto_intros)

lemma apply_modgrp_has_field_derivative [derivative_intros]:
  assumes " $\neg$ is_singular_modgrp f  $\vee$  x  $\neq$  pole_modgrp f"
  shows "(apply_modgrp f has_field_derivative (1 / automorphy_factor
f x  $\wedge$  2)) (at x within A)"
  using assms
proof (transfer, goal_cases)
  case (1 f x A)
  obtain a b c d where [simp]: "f = (a, b, c, d)"
    using prod_cases4 .
  have "(moebius (of_int a) (of_int b) (of_int c) (of_int d) has_field_derivative
(of_int (a * d - b * c) / ((of_int c * x + of_int d) $^2$ )))
(at x within A)"
    unfolding of_int_mult of_int_diff
    by (rule moebius_has_field_derivative) (use 1 in auto)
  also have "a * d - b * c = 1"
    using 1 by simp
  finally show ?case
    by (simp add: field_simps)
qed

lemma apply_modgrp_has_field_derivative' [derivative_intros]:
  assumes "(g has_field_derivative g') (at x within A)"
  assumes " $\neg$ is_singular_modgrp f  $\vee$  g x  $\neq$  pole_modgrp f"
  shows "(( $\lambda$ x. apply_modgrp f (g x)) has_field_derivative g' / automorphy_factor
f (g x)  $\wedge$  2)
(at x within A)"
proof -
  have "((apply_modgrp f  $\circ$  g) has_field_derivative
(1 / automorphy_factor f (g x)  $\wedge$  2) * g') (at x within A)"
    by (intro DERIV_chain assms derivative_intros)
  thus ?thesis
    by (simp add: o_def)
qed

lemma Im_apply_modgrp: "Im (apply_modgrp f t) = Im t / norm (automorphy_factor
f t)  $\wedge$  2"

```

by (auto simp: modgrp.Im_transform automorphy_factor_altdef)

2.6 Sending rational numbers to infinity

The following gives a unimodular transformation that sends a specific rational number to infinity. This is not unique.

```
lift_definition modgrp_of_rat :: "rat  $\Rightarrow$  modgrp" is
  "\x. let (u, v) = quotient_of x; (a, b) = bezout_coefficients u v in
  (a, b, -v, u)"
proof goal_cases
  case (1 x)
  obtain u v where x_eq: "quotient_of x = (u, v)"
    by (cases "quotient_of x")
  obtain a b where ab: "bezout_coefficients u v = (a, b)"
    by (cases "bezout_coefficients u v")
  have "coprime u v"
    using x_eq by (metis quotient_of_coprime)
  hence "a * u + b * v = 1"
    using bezout_coefficients_fst_snd[of u v] unfolding ab by auto
  thus ?case
    by (auto simp: x_eq ab)
qed
```

```
lemma modgrp_of_rat_of_int: "modgrp_of_rat (of_int n) = -S_modgrp * shift_modgrp
(-n)"
  by transfer (auto simp: bezout_coefficients_def euclid_ext_aux.simps)
```

```
lemma modgrp_of_rat_of_nat: "modgrp_of_rat (of_nat n) = -S_modgrp * shift_modgrp
(-int n)"
  using modgrp_of_rat_of_int[of "int n"] by simp
```

```
lemma modgrp_of_rat_0 [simp]: "modgrp_of_rat 0 = -S_modgrp"
  using modgrp_of_rat_of_int[of 0] by simp
```

```
lemma pole_modgrp_of_rat [simp]: "pole_modgrp (modgrp_of_rat x) = x"
proof (transfer, goal_cases)
  case (1 x)
  have "rat_of_int (fst (quotient_of x)) / rat_of_int (snd (quotient_of
x)) = x"
    by (metis snd_conv quotient_of_div fst_conv prod_eqI)
  thus ?case
    by (auto simp: case_prod_unfold Let_def)
qed
```

2.7 Representation as product of powers of generators

```
definition modgrp_from_gens :: "int option list  $\Rightarrow$  modgrp" where
  "modgrp_from_gens xs = prod_list (map ( $\lambda$ x. case x of None  $\Rightarrow$  S_modgrp
| Some n  $\Rightarrow$  shift_modgrp n) xs)"
```

```

lemma modgrp_from_gens_Nil [simp]:
  "modgrp_from_gens [] = 1"
  and modgrp_from_gens_append [simp]:
    "modgrp_from_gens (xs @ ys) = modgrp_from_gens xs * modgrp_from_gens
ys"
  and modgrp_from_gens_Cons1 [simp]:
    "modgrp_from_gens (None # xs) = S_modgrp * modgrp_from_gens xs"
  and modgrp_from_gens_Cons2 [simp]:
    "modgrp_from_gens (Some n # xs) = shift_modgrp n * modgrp_from_gens
xs"
  and modgrp_from_gens_Cons:
    "modgrp_from_gens (x # xs) =
      (case x of None  $\Rightarrow$  S_modgrp | Some n  $\Rightarrow$  shift_modgrp n) *
modgrp_from_gens xs"
  by (simp_all add: modgrp_from_gens_def)

definition invert_modgrp_gens :: "int option list  $\Rightarrow$  int option list"
  where "invert_modgrp_gens = rev  $\circ$  map (map_option uminus)"

lemma invert_modgrp_gens_Nil [simp]:
  "invert_modgrp_gens [] = []"
  and invert_modgrp_gens_append [simp]:
    "invert_modgrp_gens (xs @ ys) = invert_modgrp_gens ys @ invert_modgrp_gens
xs"
  and invert_modgrp_gens_Cons1 [simp]:
    "invert_modgrp_gens (None # xs) = invert_modgrp_gens xs @ [None]"
  and invert_modgrp_gens_Cons2 [simp]:
    "invert_modgrp_gens (Some n # xs) = invert_modgrp_gens xs @ [Some
(-n)]"
  and invert_modgrp_gens_Cons:
    "invert_modgrp_gens (x # xs) = invert_modgrp_gens xs @ [map_option
uminus x]"
  by (simp_all add: invert_modgrp_gens_def)

lemma sgn_S_modgrp [simp]: "sgn S_modgrp = 1"
  by transfer auto

lemma sgn_T_modgrp [simp]: "sgn T_modgrp = 1"
  by transfer auto

lemma sgn_shift_modgrp [simp]: "sgn (shift_modgrp n) = 1"
  by transfer auto

lemma abs_S_modgrp [simp]: "abs S_modgrp = S_modgrp"
  by transfer auto

lemma abs_T_modgrp [simp]: "abs T_modgrp = T_modgrp"
  by transfer auto

```

```

lemma abs_shift_modgrp [simp]: "abs (shift_modgrp n) = shift_modgrp n"
  by transfer auto

lemma abs_modgrp_eqE:
  assumes "abs f = abs g"
  obtains "f = g" | "f = (-g :: modgrp)"
  using assms by transfer (auto split: if_splits)

lemma abs_modgrp_eq_1_iff: "abs (f :: modgrp) = 1  $\longleftrightarrow$  f = 1  $\vee$  f = -1"
  using abs_modgrp_eqE[of f 1] by auto

lemma abs_modgrp_uminus_cong:
  "abs f = abs f'  $\implies$  abs (-f) = abs (-f' :: modgrp)"
  by auto

lemma abs_modgrp_mult_cong:
  "abs f = abs f'  $\implies$  abs g = abs g'  $\implies$  abs (f * g) = abs (f' * g' ::
modgrp)"
  by (auto elim!: abs_modgrp_eqE)

lemma abs_modgrp_inverse_cong:
  "abs f = abs f'  $\implies$  abs (inverse f) = abs (inverse f' :: modgrp)"
  by (auto elim!: abs_modgrp_eqE)

lemmas abs_modgrp_congs = abs_modgrp_uminus_cong abs_modgrp_mult_cong
abs_modgrp_inverse_cong

lemma modgrp_from_gens_invert [simp]:
  "abs (modgrp_from_gens (invert_modgrp_gens xs)) = abs (inverse (modgrp_from_gens
xs))"
  by (induction xs)
    (auto simp: invert_modgrp_gens_Cons map_option_case algebra_simps

      modgrp.inverse_distrib_swap shift_modgrp_minus
      intro: abs_modgrp_congs split: option.splits)

function modgrp_genseq :: "int  $\Rightarrow$  int  $\Rightarrow$  int  $\Rightarrow$  int  $\Rightarrow$  int option list"
where
  "modgrp_genseq a b c d =
    (if c = 0 then let b' = (if a > 0 then b else -b) in [Some b']
    else modgrp_genseq (-a * (d div c) + b) (-a) (d mod c) (-c) @ [None,
Some (d div c)])"
  by auto
termination
  by (relation "Wellfounded.measure (nat  $\circ$  abs  $\circ$  ( $\lambda$ (_,_,c,_)  $\Rightarrow$  c))")
(auto simp: abs_mod_less)

lemmas [simp del] = modgrp_genseq.simps

```

```

lemma modgrp_genseq_c_0: "modgrp_genseq a b 0 d = (let b' = (if a > 0
then b else -b) in [Some b'])"
  and modgrp_genseq_c_nz:
    "c ≠ 0 ⇒ modgrp_genseq a b c d =
      (let q = d div c in modgrp_genseq (-a * q + b) (-a) (d mod
c) (-c) @ [None, Some q])"
  by (subst modgrp_genseq.simps; simp add: Let_def)+

lemma modgrp_genseq_code [code]:
  "modgrp_genseq a b c d =
    (if c = 0 then [Some (if a > 0 then b else -b)]
    else (let q = d div c in modgrp_genseq (-a * q + b) (-a) (d mod
c) (-c) @ [None, Some q]))"
  by (subst modgrp_genseq.simps) (auto simp: Let_def)

lemma modgrp_genseq_correct:
  assumes "a * d - b * c = 1"
  shows "abs (modgrp_from_gens (modgrp_genseq a b c d)) = abs (modgrp
a b c d)"
  using assms
proof (induction a b c d rule: modgrp_genseq.induct)
  case (1 a b c d)
  write S_modgrp ("S")
  write shift_modgrp ("T")

  show ?case
  proof (cases "c = 0")
    case [simp]: True
    thus ?thesis using 1
      by (auto simp: modgrp_genseq_c_0 zmult_eq_1_iff shift_conv_modgrp
abs_modgrp_code modgrp_uminus)
  next
    case False
    define q r where "q = d div c" and "r = d mod c"
    have "q * c + r = d"
      by (simp add: q_def r_def)
    with <a * d - b * c = 1> have det': "a * r - (b - a * q) * c = 1"
      by Groebner_Basis.algebra

    have "|modgrp_from_gens (modgrp_genseq a b c d)| = |modgrp (-a * q
+ b) (-a) r (-c) * (S * T q)|"
      using False "1.IH" det'
      by (auto simp: modgrp_genseq_c_nz Let_def q_def r_def intro: abs_modgrp_congs)
    also have "S * T q = modgrp 0 (- 1) 1 q"
      unfolding S_conv_modgrp shift_conv_modgrp by (subst modgrp_times)
  simp_all
  also have "modgrp (-a * q + b) (-a) r (-c) * ... = modgrp (- a) (-
b) (- c) (-r - c * q)"

```

```

    using det' by (subst modgrp_times) simp_all
    also have "|...| = |modgrp a b c (q * c + r)|"
    using det' by (auto simp: abs_modgrp_code algebra_simps modgrp_uminus)
    also have "q * c + r = d"
    by (simp add: q_def r_def)
    finally show ?thesis .
qed
qed

lemma filterlim_apply_modgrp_at:
  assumes "~is_singular_modgrp g  $\vee$  z  $\neq$  pole_modgrp g"
  shows "filterlim (apply_modgrp g) (at (apply_modgrp g z)) (at (z :: 'a :: real_normed_field))"
proof -
  have " $\forall_F$  x in at z. x  $\in$  (if is_singular_modgrp g then ~{pole_modgrp g} else UNIV) - {z}"
  by (intro eventually_at_in_open) (use assms in auto)
  hence " $\forall_F$  x in at z. apply_modgrp g x  $\neq$  apply_modgrp g z"
  by eventually_elim (use assms in <auto split: if_splits>)
  thus ?thesis
  using assms by (auto simp: filterlim_at intro!: tendsto_eq_intros)
qed

lemma apply_modgrp_neq_pole_image [simp]:
  "is_singular_modgrp g  $\implies$  z  $\neq$  pole_modgrp g  $\implies$ 
  apply_modgrp g (z :: 'a :: field_char_0)  $\neq$  pole_image_modgrp g"
  by transfer (auto simp: field_simps add_eq_0_iff moebius_def
    simp flip: of_int_add of_int_mult of_int_diff)

lemma image_apply_modgrp_conv_vimage:
  fixes A :: "'a :: field_char_0 set"
  assumes "~is_singular_modgrp f  $\vee$  pole_modgrp f  $\notin$  A"
  defines "S  $\equiv$  (if is_singular_modgrp f then ~{pole_image_modgrp f :: 'a} else UNIV)"
  shows "apply_modgrp f ' A = apply_modgrp (inverse f) -' A  $\cap$  S"
proof (intro equalityI subsetI)
  fix z assume z: "z  $\in$  (apply_modgrp (inverse f) -' A)  $\cap$  S"
  thus "z  $\in$  apply_modgrp f ' A" using assms
  by (auto elim!: rev_image_eqI simp flip: apply_modgrp_mult simp: S_def
  split: if_splits)
next
  fix z assume z: "z  $\in$  apply_modgrp f ' A"
  then obtain x where x: "x  $\in$  A" "z = apply_modgrp f x"
  by auto
  have "apply_modgrp (inverse f) (apply_modgrp f x)  $\in$  A"
  using x assms by (subst apply_modgrp_mult [symmetric]) auto
  moreover have "apply_modgrp f x  $\neq$  pole_image_modgrp f" if "is_singular_modgrp f"
  using x assms that by (intro apply_modgrp_neq_pole_image) auto

```

```

ultimately show "z ∈ (apply_modgrp (inverse f) -' A) ∩ S"
  using x by (auto simp: S_def)
qed

lemma apply_modgrp_open_map:
  fixes A :: "'a :: real_normed_field set"
  assumes "open A" "¬is_singular_modgrp f ∨ pole_modgrp f ∉ A"
  shows "open (apply_modgrp f ' A)"
proof -
  define S :: "'a set" where "S = (if is_singular_modgrp f then -{pole_image_modgrp
f} else UNIV)"
  have "open S"
    by (auto simp: S_def)
  have "apply_modgrp f ' A = apply_modgrp (inverse f) -' A ∩ S"
    using image_apply_modgrp_conv_vimage[of f A] assms by (auto simp:
S_def)
  also have "apply_modgrp (inverse f) -' A ∩ S = S ∩ apply_modgrp (inverse
f) -' A"
    by (simp only: Int_commute)
  also have "open ..."
    using assms by (intro continuous_open_preimage continuous_intros assms
<open S>)
    (auto simp: S_def)
  finally show ?thesis .
qed

lemma filtermap_at_apply_modgrp:
  fixes z :: "'a :: real_normed_field"
  assumes "¬is_singular_modgrp g ∨ z ≠ pole_modgrp g"
  shows "filtermap (apply_modgrp g) (at z) = at (apply_modgrp g z)"
proof (rule filtermap_fun_inverse)
  show "filterlim (apply_modgrp g) (at (apply_modgrp g z)) (at z)"
    using assms by (intro filterlim_apply_modgrp_at) auto
next
  have "filterlim (apply_modgrp (inverse g)) (at (apply_modgrp (inverse
g) (apply_modgrp g z)))
    (at (apply_modgrp g z))"
    using assms by (intro filterlim_apply_modgrp_at) auto
  also have "apply_modgrp (inverse g) (apply_modgrp g z) = z"
    using assms by (simp flip: apply_modgrp_mult)
  finally show "filterlim (apply_modgrp (inverse g)) (at z) (at (apply_modgrp
g z))" .
next
  have "eventually (λx. x ∈ (if is_singular_modgrp g then -{pole_image_modgrp
g} else UNIV))
    (at (apply_modgrp g z))"
    by (intro eventually_at_in_open') (use assms in auto)
  thus "∀F x in at (apply_modgrp g z). apply_modgrp g (apply_modgrp (inverse
g) x) = x"

```

by eventually_elim (auto simp flip: apply_modgrp_mult split: if_splits)
qed

lemma zorder_moebius_zero:

assumes "a \neq 0" "a * d - b * c \neq 0"

shows "zorder (moebius a b c d) (-b / a) = 1"

proof (rule zorder_eqI)

note [simp del] = div_mult_self3 div_mult_self4 div_mult_self2 div_mult_self1

define A where "A = (if c = 0 then UNIV else $\{-d/c\}$)"

show "open A"

by (auto simp: A_def)

show " $-b/a \in A$ "

using assms by (auto simp: A_def field_simps)

show " $(\lambda x. a / (c * x + d))$ holomorphic_on A"

proof (intro holomorphic_intros)

fix x assume "x $\in A$ "

thus " $c * x + d \neq 0$ "

unfolding A_def using assms

by (auto simp: A_def field_simps add_eq_0_iff split: if_splits)

qed

show " $a / (c * (-b / a) + d) \neq 0$ "

using assms by (auto simp: field_simps)

show "moebius a b c d w = a / (c * w + d) * (w - - b / a) powi 1"

if " $w \in (if c = 0 then UNIV else $\{-d/c\})$ " " $w \neq -b/a$ " for w$

using that assms by (auto simp: divide_simps moebius_def split: if_splits)

qed

lemma zorder_moebius_pole:

assumes "c \neq 0" "a * d - b * c \neq 0"

shows "zorder (moebius a b c d) (-d / c) = -1"

proof -

note [simp del] = div_mult_self3 div_mult_self4 div_mult_self2 div_mult_self1

have "zorder (moebius a b c d) (-d / c) =

zorder $(\lambda x. (a * x + b) / c / (x - (-d / c))) ^ 1 (-d / c)$ "

proof (rule zorder_cong)

have "eventually $(\lambda z. z \neq -d/c)$ (at $(-d/c)$)"

by (simp add: eventually_neq_at_within)

thus " $\forall_F z$ in at $(-d / c). \text{moebius } a \ b \ c \ d \ z = (a * z + b) / c /$
 $(z - -d / c) ^ 1$ "

by eventually_elim (use assms in <auto simp: moebius_def divide_simps
mult_ac>)

qed auto

also have "zorder $(\lambda x. (a * x + b) / c / (x - (-d / c))) ^ 1 (-d / c)$
= -int 1"

proof (rule zorder_nonzero_div_power)

show " $(\lambda w. (a * w + b) / c)$ holomorphic_on UNIV"

using assms by (intro holomorphic_intros)

show " $(a * (-d / c) + b) / c \neq 0$ "

using assms by (auto simp: field_simps)

```

qed auto
finally show ?thesis by simp
qed

lemma zorder_moebius:
  assumes "c = 0  $\vee$  z  $\neq$  -d / c" "a * d - b * c  $\neq$  0"
  shows "zorder ( $\lambda$ x. moebius a b c d x - moebius a b c d z) z = 1"
proof (rule zorder_eqI)
  define S where "S = (if c = 0 then UNIV else -{-d/c})"
  define g where "g = ( $\lambda$ w. (a * d - b * c) / ((c * w + d) * (c * z + d)))"
  show "open S" "z  $\in$  S"
  using assms by (auto simp: S_def)
  show "g holomorphic_on S"
  unfolding g_def using assms
  by (intro holomorphic_intros no_zero_divisors)
  (auto simp: S_def field_simps add_eq_0_iff split: if_splits)
  show "(a * d - b * c) / ((c * z + d) * (c * z + d))  $\neq$  0"
  using assms by (auto simp: add_eq_0_iff split: if_splits)
  show "moebius a b c d w - moebius a b c d z =
    (a * d - b * c) / ((c * w + d) * (c * z + d)) * (w - z) powi
  1" if "w  $\in$  S" for w
  by (subst moebius_diff_eq) (use that assms in <auto simp: S_def split:
if_splits>)
qed

lemma zorder_apply_modgrp:
  assumes " $\neg$ is_singular_modgrp g  $\vee$  z  $\neq$  pole_modgrp g"
  shows "zorder ( $\lambda$ x. apply_modgrp g x - apply_modgrp g z) z = 1"
  using assms
proof (transfer, goal_cases)
  case (1 f z)
  obtain a b c d where [simp]: "f = (a, b, c, d)"
  using prod_cases4 .
  show ?case using 1 zorder_moebius[of c z d a b]
  by (auto simp: simp flip: of_int_mult)
qed

lemma zorder_fls_modgrp_pole:
  assumes "is_singular_modgrp f"
  shows "zorder (apply_modgrp f) (pole_modgrp f) = -1"
  using assms
proof (transfer, goal_cases)
  case (1 f)
  obtain a b c d where [simp]: "f = (a, b, c, d)"
  using prod_cases4 .
  show ?case using 1 zorder_moebius_pole[of c a d b]
  by (auto simp: simp flip: of_int_mult)
qed

```

2.8 Induction rules in terms of generators

Theorem 2.1

```

lemma modgrp_induct_S_shift [case_names id uminus S shift]:
  assumes "P 1"
    " $\bigwedge x. P x \implies P (-x)$ "
    " $\bigwedge x. P x \implies P (S\_modgrp * x)$ "
    " $\bigwedge x n. P x \implies P (shift\_modgrp n * x)$ "
  shows "P x"
proof -
  define xs where "xs = modgrp_genseq (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x)"
  have "P (modgrp_from_gens xs)"
    by (induction xs) (auto intro: assms simp: modgrp_from_gens_Cons split:
option.splits)
  moreover have "|x| = |modgrp_from_gens xs|" unfolding xs_def
    using modgrp_genseq_correct[of "modgrp_a x" "modgrp_d x" "modgrp_b
x" "modgrp_c x"]
      modgrp_abcd_det[of x] by auto
  hence "x = modgrp_from_gens xs  $\vee$  x = -modgrp_from_gens xs"
    by (elim abs_modgrp_eqE) auto
  ultimately show ?thesis
    using assms(2)[of "modgrp_from_gens xs"] by auto
qed

```

```

lemma modgrp_induct [case_names id uminus S T inv_T]:
  assumes "P 1"
    " $\bigwedge x. P x \implies P (-x)$ "
    " $\bigwedge x. P x \implies P (S\_modgrp * x)$ "
    " $\bigwedge x. P x \implies P (T\_modgrp * x)$ "
    " $\bigwedge x. P x \implies P (inverse T\_modgrp * x)$ "
  shows "P x"
proof -
  define xs where "xs = modgrp_genseq (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x)"
  have *: "P (T_modgrp ^ n * x)" if "P x" for n x
    by (induction n) (auto simp: mult.assoc shift_modgrp_1 intro: assms
that)
  have **: "P (inverse T_modgrp ^ n * x)" if "P x" for n x
    by (induction n) (auto simp: shift_modgrp_add mult.assoc shift_modgrp_1
intro: assms that)
  have ***: "P (shift_modgrp n * x)" if "P x" for n x
    using *[of x "nat n"] **[of x "nat (-n)"] that
    by (auto simp: shift_modgrp_conv_T_power power_int_def)
  have "P (modgrp_from_gens xs)"
    by (induction xs) (auto intro: assms *** simp: modgrp_from_gens_Cons
split: option.splits)
  moreover have "|x| = |modgrp_from_gens xs|" unfolding xs_def
    using modgrp_genseq_correct[of "modgrp_a x" "modgrp_d x" "modgrp_b

```

```

x" "modgrp_c x"]
  modgrp_abcd_det[of x] by auto
hence "x = modgrp_from_gens xs ∨ x = -modgrp_from_gens xs"
  by (elim abs_modgrp_eqE) auto
ultimately show ?thesis
  using assms(2)[of "modgrp_from_gens xs"] by auto
qed

lemma modgrp_induct_S_shift' [case_names id uminus S shift]:
  assumes "P 1"
    "∧x. P x ⇒ P (-x)"
    "∧x. abs x = x ⇒ P x ⇒ P (x * S_modgrp)"
    "∧x n. abs x = x ⇒ P x ⇒ P (x * shift_modgrp n)"
  shows "P x"
proof -
  define xs where "xs = modgrp_genseq (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x)"
  have 1: "P (x * S_modgrp)" if "P x" for x using assms that
    by (metis abs_eq_modgrpE abs_modgrp_altdef abs_uminus_modgrp times_modgrp_uminus_left)
  have 2: "P (x * shift_modgrp n)" if "P x" for x n using assms that
    by (metis abs_eq_modgrpE abs_modgrp_altdef abs_uminus_modgrp times_modgrp_uminus_left)
  have "P (modgrp_from_gens xs)"
    by (induction xs rule: rev_induct)
    (auto intro: assms(1,2) 1 2 simp: modgrp_from_gens_Cons split:
option.splits)
  moreover have "|x| = |modgrp_from_gens xs|" unfolding xs_def
    using modgrp_genseq_correct[of "modgrp_a x" "modgrp_d x" "modgrp_b
x" "modgrp_c x"]
    modgrp_abcd_det[of x] by auto
  hence "x = modgrp_from_gens xs ∨ x = -modgrp_from_gens xs"
    by (elim abs_modgrp_eqE) auto
  ultimately show ?thesis
    using assms(2)[of "modgrp_from_gens xs"] by auto
qed

lemma modgrp_induct' [case_names id uminus S T inv_T]:
  assumes "P 1"
    "∧x. P x ⇒ P (-x)"
    "∧x. P x ⇒ P (x * S_modgrp)"
    "∧x. P x ⇒ P (x * T_modgrp)"
    "∧x. P x ⇒ P (x * inverse T_modgrp)"
  shows "P x"
proof -
  define xs where "xs = modgrp_genseq (modgrp_a x) (modgrp_b x) (modgrp_c
x) (modgrp_d x)"
  have *: "P (x * T_modgrp ^ n)" if "P x" for n x
    using assms(4)[of "x * T_modgrp ^ n" for n]
    by (induction n) (auto intro: that simp: algebra_simps power_commuting_commutates)
  have **: "P (x * inverse T_modgrp ^ n)" if "P x" for n x

```

```

using assms(5)[of "x * inverse T_modgrp ^ n" for n]
by (induction n) (auto intro: that simp: algebra_simps power_commuting_commutates)

have ***: "P (x * shift_modgrp n)" if "P x" for n x
  using *[of x "nat n"] **[of x "nat (-n)"] that
  by (auto simp: shift_modgrp_conv_T_power power_int_def)
have "P (modgrp_from_gens xs)"
  by (induction xs rule: rev_induct)
  (auto intro: assms *** simp: modgrp_from_gens_Cons split: option.splits)
moreover have "|x| = |modgrp_from_gens xs|" unfolding xs_def
  using modgrp_genseq_correct[of "modgrp_a x" "modgrp_d x" "modgrp_b
x" "modgrp_c x"]
  modgrp_abcd_det[of x] by auto
hence "x = modgrp_from_gens xs ∨ x = -modgrp_from_gens xs"
  by (elim abs_modgrp_eqE) auto
ultimately show ?thesis
  using assms(2)[of "modgrp_from_gens xs"] by auto
qed

```

```

definition apply_modgrp' :: "modgrp ⇒ 'a × 'a ⇒ 'a × 'a :: ring_1"
  where "apply_modgrp' f =
    (λ(x,y). (of_int (modgrp_a f) * x + of_int (modgrp_b f) * y,
            of_int (modgrp_c f) * x + of_int (modgrp_d f) * y))"

```

```

lemma apply_modgrp'_z_one:
  assumes "z ∉ ℝ"
  shows "apply_modgrp' f (z, 1) = (automorphy_factor f z * apply_modgrp
f z, automorphy_factor f z)"
proof -
  from assms have "complex_of_int (modgrp_c f) * z + complex_of_int (modgrp_d
f) ≠ 0"
  by (simp add: complex_is_Real_iff modgrp.Im_transform_pos_aux)
  thus ?thesis
  by (simp add: apply_modgrp'_def automorphy_factor_altdef apply_modgrp_altdef
moebius_def)
qed

```

2.9 Reduction map

Two elements of the modular group are considered congruent modulo n if their entries are.

```

definition cong_modgrp :: "modgrp ⇒ modgrp ⇒ int ⇒ bool"
  (<<(indent=1 notation=<mixfix cong>>[_ = _] '(mod_Γ _'))>>) where
  "[f = g] (mod_Γ n) ↔ list_all (λh. [h f = h g] (mod n)) [modgrp_a,
modgrp_b, modgrp_c, modgrp_d]"

```

```

lemma cong_modgrp_abs [simp]: "[f = g] (mod_Γ (abs n)) ↔ [f = g] (mod_Γ
n)"

```

```

by (auto simp: cong_modgrp_def)

lemma cong_modgrp_refl [intro]: "[f = f] (modΓ n)"
  by (auto simp: cong_modgrp_def)

lemma cong_modgrp_mult:
  "[f1 = g1] (modΓ n)  $\implies$  [f2 = g2] (modΓ n)  $\implies$  [f1 * f2 = g1 * g2]
(modΓ n)"
  unfolding cong_modgrp_def list_all_Cons_iff list_all_Nil_iff
    modgrp_a_times modgrp_b_times modgrp_c_times modgrp_d_times
  by (elim conjE TrueE; intro conjI TrueI cong_add cong_mult)

lemma cong_modgrp_inverse:
  "[f = g] (modΓ n)  $\implies$  [inverse f = inverse g] (modΓ n)"
  unfolding cong_modgrp_def list_all_Cons_iff list_all_Nil_iff
    modgrp_a_inverse modgrp_b_inverse modgrp_c_inverse modgrp_d_inverse
  by (elim conjE TrueE; intro conjI TrueI cong_add cong_mult cong_uminus)

The following version of the Chinese Remainder Theorem also works when
the two moduli are not coprime.

lemma chinese_remainder_theorem_gen:
  fixes m n :: "'a :: {unique_euclidean_ring, euclidean_ring_gcd}"
  assumes "[x = y] (mod gcd m n)"
  obtains z where "[z = x] (mod m)" "[z = y] (mod n)"
proof -
  obtain u v where "bezout_coefficients m n = (u, v)"
    by (cases "bezout_coefficients m n")
  hence uv: "gcd m n = u * m + v * n"
    using bezout_coefficients_fst_snd[of m n] by simp
  define k where "k = (x - y) div gcd m n"
  have "x = y + k * gcd m n"
    using assms by (auto simp: cong_iff_dvd_diff k_def)
  hence x_eq: "x = y + k * u * m + k * v * n"
    by (simp add: uv algebra_simps)
  show ?thesis
  proof (rule that)
    have "[y + k * u * 0 + k * v * n = y + k * u * m + k * v * n] (mod
m)"
      by (intro cong_add cong_mult cong_refl) (auto simp: cong_def)
    thus "[y + k * v * n = x] (mod m)"
      by (simp add: x_eq)
  next
    have "[y + k * v * n = y + k * v * 0] (mod n)"
      by (intro cong_add cong_mult cong_refl) (auto simp: cong_def)
    thus "[y + k * v * n = y] (mod n)"
      by (simp add: x_eq)
  qed
qed

```

```

lemma cong_solve:
  fixes a :: "'a :: {unique_euclidean_semiring, euclidean_ring_gcd}"
  shows "∃x. [a * x = gcd a n] (mod n)"
proof -
  obtain u v where uv: "bezout_coefficients a n = (u, v)"
    by (cases "bezout_coefficients a n")
  have "gcd a n = u * a + v * n"
    using bezout_coefficients[of a n u v] uv by auto
  also have "[u * a + v * n = u * a + v * 0] (mod n)"
    by (intro cong_add cong_mult) (auto simp: cong_0_iff)
  finally show ?thesis
    by (intro exI[of _ u]) (simp add: cong_sym_eq mult_ac)
qed

lemma cong_solve_coprime:
  fixes a :: "'a :: {unique_euclidean_ring, euclidean_ring_gcd}"
  shows "coprime a n ⟹ ∃x. [a * x = 1] (mod n)"
  using cong_solve[of a n] by simp

lemma chinese_remainder:
  fixes A :: "'a set"
  and m :: "'a ⇒ 'b :: {unique_euclidean_ring, euclidean_ring_gcd}"
  and u :: "'a ⇒ 'b"
  assumes fin: "finite A"
  and cop: "∀i ∈ A. ∀j ∈ A. i ≠ j ⟶ coprime (m i) (m j)"
  shows "∃x. ∀i ∈ A. [x = u i] (mod m i)"
proof -
  have "∃b. (∀i ∈ A. [b i = 1] (mod m i) ∧ [b i = 0] (mod (∏j ∈ A - {i}. m j)))"
  proof (rule finite_set_choice, rule fin, rule ballI)
    fix i
    assume "i ∈ A"
    with cop have "coprime (∏j ∈ A - {i}. m j) (m i)"
      by (intro prod_coprime_left) auto
    then have "∃x. [(∏j ∈ A - {i}. m j) * x = 1] (mod m i)"
      by (elim cong_solve_coprime)
    then obtain x where "[ (∏j ∈ A - {i}. m j) * x = 1] (mod m i)"
      by auto
    moreover have "[ (∏j ∈ A - {i}. m j) * x = 0] (mod (∏j ∈ A - {i}. m j))"
      by (simp add: cong_0_iff)
    ultimately show "∃a. [a = 1] (mod m i) ∧ [a = 0] (mod prod m (A - {i}))"
      by blast
  qed
  then obtain b where b: "∧i. i ∈ A ⟹ [b i = 1] (mod m i) ∧ [b i

```

```

= 0] (mod ( $\prod_{j \in A - \{i\}} m j$ ))"
  by blast
let ?x = " $\sum_{i \in A} (u i) * (b i)$ "
show ?thesis
proof (rule exI, clarify)
  fix i
  assume a: "i  $\in$  A"
  show "[?x = u i] (mod m i)"
  proof -
    from fin a have "?x = ( $\sum_{j \in \{i\}} u j * b j$ ) + ( $\sum_{j \in A - \{i\}} u j * b j$ )"
    by (subst sum.union_disjoint [symmetric]) (auto intro: sum.cong)
    then have "[?x = u i * b i + ( $\sum_{j \in A - \{i\}} u j * b j$ )] (mod m i)"
    by auto
    also have "[u i * b i + ( $\sum_{j \in A - \{i\}} u j * b j$ ) = u i * 1 + ( $\sum_{j \in A - \{i\}} u j * 0$ )] (mod m i)"
    proof (intro cong_add cong_scalar_left cong_sum)
      show "[b i = 1] (mod m i)"
      using a b by blast
      show "[b x = 0] (mod m i)" if "x  $\in$  A - {i}" for x
      proof -
        have "x  $\in$  A" "x  $\neq$  i"
        using that by auto
        then show ?thesis
        using a b [OF <x  $\in$  A>] cong_dvd_modulus fin by blast
      qed
    qed
    finally show ?thesis
    by simp
  qed
qed
qed
qed

```

```

lemma coprime_via_prime_factors:
  fixes x y :: "'a :: factorial_semiring_gcd"
  assumes "x  $\neq$  0" " $\bigwedge p$ . prime p  $\implies$  p dvd x  $\implies$   $\neg$ p dvd y"
  shows "coprime x y"
proof (rule ccontr)
  assume " $\neg$ coprime x y"
  with <x  $\neq$  0> have " $\neg$ is_unit (gcd x y)" "gcd x y  $\neq$  0"
  by auto
  then obtain p where "prime p" "p dvd gcd x y"
  by (metis prime_divisor_exists)
  with assms(2)[of p] show False
  by auto
qed

```

The following shows that the reduction map $SL(2, \mathbb{Z}) \rightarrow SL(2, \mathbb{Z}/n\mathbb{Z})$ is

surjective.

This means that if we have a matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $ad - bc = 1 \pmod{n}$,

there exists a matrix $A' = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix}$ with $a'd' - b'c' = 1$ and $A = A' \pmod{n}$.

So, basically, we can take a matrix whose determinant is only congruent to 1 modulo n and turn it into one with determinant exactly 1 while only adding and subtracting multiples of n to the entries.

```

lemma exists_coprime_shifted_int_aux:
  fixes c d :: int
  assumes c: "c ≠ 0"
  assumes coprime: "coprime (gcd c d) n"
  obtains d' where "[d' = d] (mod n)" "coprime c d'"
proof -
  define g where "g = gcd c d"
  have "g ≠ 0"
    using c by (auto simp: g_def)
  have g: "coprime g n"
    using coprime unfolding g_def by (metis coprime_iff_gcd_eq_1 gcd.assoc
gcd_1_int)

  define P1 where "P1 = prime_factors g"
  define P2 where "P2 = prime_factors c - P1"
  define a where "a = (λp. if p dvd d then modular_inverse p n * (1 -
d) else 0)"

  have "∃x. ∀p∈prime_factors c. [x = a p] (mod p)"
proof (rule chinese_remainder)
  show "finite (prime_factors c)"
    by (auto simp: P1_def P2_def)
  by (auto simp: P1_def P2_def in_prime_factors_iff intro: primes_coprime)
next
  show "∀p∈prime_factors c. ∀j∈prime_factors c. p ≠ j → coprime
p j"
  by (auto simp: P1_def P2_def in_prime_factors_iff intro: primes_coprime)
qed
then obtain x where x: "∧p. prime p ⇒ p dvd c ⇒ [x = a p] (mod
p)"
  using c by (auto simp: prime_factors_dvd)
define d' where "d' = d + x * n"

have "coprime c d'"
proof (rule coprime_via_prime_factors)
  fix p assume p: "prime p" "p dvd c"
  show "¬p dvd d'"
  proof (cases "p dvd d")
    case False
    have "[d' = d + a p * n] (mod p)"
      unfolding d'_def by (intro cong_add cong_mult x cong_refl p)
  end
end

```

```

    hence "[d' = d] (mod p)"
      using False by (simp add: a_def)
    thus "¬p dvd d'"
      using False cong_dvd_iff by blast
  next
  case True
  with coprime have "coprime p n"
    using coprime_divisors[OF _ dvd_refl, of p "gcd c d" n] p by auto
  have "[d' = d + a p * n] (mod p)"
    unfolding d'_def by (intro cong_add cong_mult x cong_refl p)
  also have "a p * n = (1 - d) * (modular_inverse p n * n)"
    using True by (simp add: algebra_simps a_def)
  also have "[d + ... = d + (1 - d) * 1] (mod p)"
    unfolding a_def using <coprime p n>
    by (intro cong_add cong_mult cong_modular_inverse2 cong_refl)
  (auto simp: coprime_commute)
  finally have "[d' = 1] (mod p)"
    by simp
  hence "coprime d' p"
    by (simp add: coprime_cong_cong_left)
  thus "¬p dvd d'"
    using coprime_absorb_right not_prime_unit p(1) by blast
  qed
qed fact+

  thus ?thesis
    by (intro that[of d']) (auto simp: d'_def cong_iff_dvd_diff)
qed

lemma exists_coprime_shifted_int:
  fixes c d :: int
  assumes coprime: "coprime (gcd c d) n"
  obtains c' d' where "[c' = c] (mod n)" "[d' = d] (mod n)" "coprime
c' d'"
proof (cases "abs n = 1")
  case True
  hence "n = 1 ∨ n = -1"
    by linarith
  thus ?thesis
    by (intro that[of 0 1]) auto
next
  case False
  hence "c ≠ 0 ∨ d ≠ 0"
    using coprime by auto
  thus ?thesis
  proof
    assume "c ≠ 0"
    then obtain d' where "[d' = d] (mod n)" "coprime c d'"
      using exists_coprime_shifted_int_aux[of c d n] coprime by auto

```

```

    thus ?thesis
      by (intro that[of c d']) auto
  next
    assume "d ≠ 0"
    then obtain c' where "[c' = c] (mod n)" "coprime c' d"
      using exists_coprime_shifted_int_aux[of d c n] coprime
      by (auto simp: gcd.commute coprime_commute)
    thus ?thesis
      by (intro that[of c' d]) auto
  qed
qed

theorem modgrp_reduction_surj:
  fixes a b c d n :: int
  assumes "[a * d - b * c = 1] (mod n)"
  obtains f :: modgrp where
    "[modgrp_a f = a] (mod n)" "[modgrp_b f = b] (mod n)"
    "[modgrp_c f = c] (mod n)" "[modgrp_d f = d] (mod n)"
proof -
  obtain c' d' where cd': "[c' = c] (mod n)" "[d' = d] (mod n)" "coprime
c' d'"
  proof (rule exists_coprime_shifted_int)
    have "coprime (a * d - b * c) n"
      using assms by (simp add: coprime_cong_cong_left)
    thus "coprime (gcd c d) n"
      using coprime_divisors[OF _ dvd_refl, of "gcd c d" "a * d - b *
c" n] by auto
  qed metis

  define z where "z = (a * d' - b * c' - 1) div n"
  have "[a * d' - b * c' = a * d - b * c] (mod n)"
    by (intro cong_diff cong_mult cd' cong_refl)
  also have "[a * d - b * c = 1] (mod n)"
    by fact
  finally have z: "a * d' - b * c' = 1 + z * n"
    by (auto simp: cong_iff_dvd_diff z_def)

  obtain x y where xy: "y * c' - x * d' = z"
  proof -
    obtain u v where "u * c' + v * d' = 1"
      using bezout_int[of c' d'] cd'(3) by auto
    hence "z * (u * c' + v * d') = z * 1"
      by (rule arg_cong)
    thus ?thesis
      by (intro that[of "z * u" "-z * v"]) (auto simp: algebra_simps)
  qed

  define a' b' where "a' = a + x * n" and "b' = b + y * n"
  have ab_cong: "[a' = a] (mod n)" "[b' = b] (mod n)"

```

```

    by (auto simp: a'_def b'_def cong_iff_dvd_diff)

  have "a' * d' - b' * c' = a * d' - b * c' - (y * c' - x * d') * n"
    by (simp add: a'_def b'_def algebra_simps)
  also have "a * d' - b * c' = 1 + z * n"
    by (rule z)
  also have "y * c' - x * d' = z"
    by (rule xy)
  finally have det: "a' * d' - b' * c' = 1"
    by simp

  show ?thesis using det ab_cong cd'
    by (intro that[of "modgrp a' b' c' d'"]) (auto simp: modgrp_abcd_modgrp)
qed

end

```

3 Complex lattices

```

theory Complex_Lattices
  imports "HOL-Complex_Analysis.Complex_Analysis" Parallelogram_Paths
begin

lemmas [simp del] = div_mult_self1 div_mult_self2 div_mult_self3 div_mult_self4

lemma (in -) image_plus_conv_vimage_plus:
  fixes c :: "'a :: group_add"
  shows "(+) c ` A = (+) (-c) -` A"
proof safe
  fix z assume "-c + z ∈ A"
  thus "z ∈ (+) c ` A"
    by (intro image_eqI[of _ _ "-c + z"]) (auto simp: algebra_simps)
qed auto

lemma filtermap_cosparse_translate:
  "filtermap ((+) (c :: 'a :: real_normed_vector)) (cosparse A) = cosparse
  ((+) c ` A)"
proof (rule antisym)
  have *: "filtermap ((+) c) (cosparse A) ≥ cosparse ((+) c ` A)" for
  c :: 'a and A
  proof (rule filter_leI)
    fix P assume P: "eventually P (filtermap ((+) c) (cosparse A))"
    hence "(+) c ` {x. ¬ P (c + x)} sparse_in (+) c ` A"
      using sparse_in_translate[of "{x. ¬ P (c + x)}" A c]
      by (simp add: eventually_filtermap eventually_cosparse)
    also have "(+) c ` {x. ¬ P (c + x)} = {x. ¬ P x}"

```

```

    unfolding image_plus_conv_vimage_plus by auto
    finally show "eventually P (cosparse ((+) c ' A))"
      by (simp add: eventually_cosparse)
qed

show "filtermap ((+) c) (cosparse A) ≥ cosparse ((+) c ' A)"
  by (rule *)
have "filtermap ((+) (-c)) (cosparse ((+) c ' A)) ≥ cosparse ((+) (-c)
' (+) c ' A)"
  by (rule *)
also have "((+) (-c) ' (+) c ' A) = A"
  by (simp add: image_image)
finally have "filtermap ((+) c) (cosparse A) ≤ filtermap ((+) c) (filtermap
((+) (-c)) (cosparse ((+) c ' A)))"
  by (intro filtermap_mono)
also have "... = cosparse ((+) c ' A)"
  by (simp add: filtermap_filtermap)
finally show "filtermap ((+) c) (cosparse A) ≤ cosparse ((+) c ' A)"
.
qed

```

3.1 Basic definitions and useful lemmas

We define a complex lattice with two generators $\omega_1, \omega_2 \in \mathbb{C}$ as the set $\Lambda(\omega_1, \omega_2) = \omega_1\mathbb{Z} + \omega_2\mathbb{Z}$. For now, we make no restrictions on the generators, but for most of our results we will require that they be independent (i.e. neither is a multiple of the other, or, in terms of complex numbers, their quotient is not real).

```

locale pre_complex_lattice =
  fixes  $\omega_1 \omega_2 :: \text{complex}$ 
begin

```

The following function convergs from lattice coordinates into cartesian coordinates.

```

definition of_ $\omega_1\omega_2$ _coords :: "real × real ⇒ complex" where
  "of_ $\omega_1\omega_2$ _coords = ( $\lambda(x,y)$ . of_real x *  $\omega_1$  + of_real y *  $\omega_2$ )"

```

```

sublocale of_ $\omega_1\omega_2$ _coords: linear of_ $\omega_1\omega_2$ _coords
  unfolding of_ $\omega_1\omega_2$ _coords_def by (auto simp: linear_iff algebra_simps
scaleR_conv_of_real)

```

```

sublocale of_ $\omega_1\omega_2$ _coords: bounded_linear of_ $\omega_1\omega_2$ _coords
  using of_ $\omega_1\omega_2$ _coords.linear_axioms linear_conv_bounded_linear by auto

```

```

lemmas [continuous_intros] = of_ $\omega_1\omega_2$ _coords.continuous_on of_ $\omega_1\omega_2$ _coords.continuous
lemmas [tendsto_intros] = of_ $\omega_1\omega_2$ _coords.tendsto

```

lemmas [simp] = of_ω12_coords.add of_ω12_coords.diff of_ω12_coords.neg
of_ω12_coords.scaleR

lemma of_ω12_coords_fst [simp]: "of_ω12_coords (a, 0) = of_real a *
ω1"
and of_ω12_coords_snd [simp]: "of_ω12_coords (0, a) = of_real a * ω2"
and of_ω12_coords_scaleR': "of_ω12_coords (c *_R z) = of_real c * of_ω12_coords
z"
by (simp_all add: of_ω12_coords_def algebra_simps case_prod_unfold
scaleR_prod_def scaleR_conv_of_real)

The following is our lattice as a set of lattice points.

definition lattice :: "complex set" ("Λ") where
"lattice = of_ω12_coords ' (ℤ × ℤ)"

definition lattice0 :: "complex set" ("Λ*") where
"lattice0 = lattice - {0}"

lemma countable_lattice [intro]: "countable lattice"
unfolding lattice_def by (intro countable_image countable_SIGMA countable_int)

lemma latticeI: "of_ω12_coords (x, y) = z ⇒ x ∈ ℤ ⇒ y ∈ ℤ ⇒
z ∈ Λ"
by (auto simp: lattice_def)

lemma latticeE:
assumes "z ∈ Λ"
obtains x y where "z = of_ω12_coords (of_int x, of_int y)"
using assms unfolding lattice_def Ints_def by auto

lemma lattice0I [intro]: "z ∈ Λ ⇒ z ≠ 0 ⇒ z ∈ Λ*"
by (auto simp: lattice0_def)

lemma lattice0E [elim]: "∧P. z ∈ Λ* ⇒ (z ∈ Λ ⇒ z ≠ 0 ⇒ P) ⇒
P"
by (auto simp: lattice0_def)

lemma in_lattice0_iff: "z ∈ Λ* ↔ z ∈ Λ ∧ z ≠ 0"
by (auto simp: lattice0_def)

named_theorems lattice_intros

lemma zero_in_lattice [lattice_intros, simp]: "0 ∈ lattice"
by (rule latticeI[of 0 0]) (auto simp: of_ω12_coords_def)

lemma generator_in_lattice [lattice_intros, simp]: "ω1 ∈ lattice" "ω2
∈ lattice"
by (auto intro: latticeI[of 0 1] latticeI[of 1 0] simp: of_ω12_coords_def)

```

lemma uminus_in_lattice [lattice_intros]: "z ∈ Λ ⇒ -z ∈ Λ"
proof -
  assume "z ∈ Λ"
  then obtain x y where "z = of_ω12_coords (of_int x, of_int y)"
    by (elim latticeE)
  thus ?thesis
    by (intro latticeI[of "-x" "-y"]) (auto simp: of_ω12_coords_def)
qed

lemma uminus_in_lattice_iff: "-z ∈ Λ ↔ z ∈ Λ"
  using uminus_in_lattice minus_minus by metis

lemma uminus_in_lattice0_iff: "-z ∈ Λ* ↔ z ∈ Λ*"
  by (auto simp: lattice0_def uminus_in_lattice_iff)

lemma add_in_lattice [lattice_intros]: "z ∈ Λ ⇒ w ∈ Λ ⇒ z + w ∈ Λ"
proof -
  assume "z ∈ Λ" "w ∈ Λ"
  then obtain a b c d
    where "z = of_ω12_coords (of_int a, of_int b)" "w = of_ω12_coords
(of_int c, of_int d)"
    by (elim latticeE)
  thus ?thesis
    by (intro latticeI[of "a + c" "b + d"]) (auto simp: of_ω12_coords_def
algebra_simps)
qed

lemma lattice_lattice0: "Λ = insert 0 Λ*"
  by (auto simp: lattice0_def)

lemma mult_of_nat_left_in_lattice [lattice_intros]: "z ∈ Λ ⇒ of_nat
n * z ∈ Λ"
  by (induction n) (auto intro!: lattice_intros simp: ring_distrib)

lemma mult_of_nat_right_in_lattice [lattice_intros]: "z ∈ Λ ⇒ z *
of_nat n ∈ Λ"
  using mult_of_nat_left_in_lattice[of z n] by (simp add: mult.commute)

lemma mult_of_int_left_in_lattice [lattice_intros]: "z ∈ Λ ⇒ of_int
n * z ∈ Λ"
  using mult_of_nat_left_in_lattice[of z "nat n"]
    uminus_in_lattice[OF mult_of_nat_left_in_lattice[of z "nat (-n)"]]
  by (cases "n ≥ 0") auto

lemma mult_of_int_right_in_lattice [lattice_intros]: "z ∈ Λ ⇒ z *
of_int n ∈ Λ"
  using mult_of_int_left_in_lattice[of z n] by (simp add: mult.commute)

```

```

lemma diff_in_lattice [lattice_intros]: "z ∈ Λ ⇒ w ∈ Λ ⇒ z - w
∈ Λ"
  using add_in_lattice[OF _ uminus_in_lattice, of z w] by simp

lemma diff_in_lattice_commute: "z - w ∈ Λ ↔ w - z ∈ Λ"
  using uminus_in_lattice_iff[of "z - w"] by simp

lemma of_ω12_coords_in_lattice [lattice_intros]: "ab ∈ ℤ × ℤ ⇒ of_ω12_coords
ab ∈ Λ"
  unfolding lattice_def by auto

lemma lattice_plus_right_cancel [simp]: "y ∈ Λ ⇒ x + y ∈ Λ ↔ x
∈ Λ"
  by (metis add_diff_cancel_right' add_in_lattice diff_in_lattice)

lemma lattice_plus_left_cancel [simp]: "x ∈ Λ ⇒ x + y ∈ Λ ↔ y ∈
Λ"
  by (metis add.commute lattice_plus_right_cancel)

lemma lattice_induct [consumes 1, case_names zero gen1 gen2 add uminus]:
  assumes "z ∈ Λ"
  assumes zero: "P 0"
  assumes gens: "P ω1" "P ω2"
  assumes plus: "∧w z. P w ⇒ P z ⇒ P (w + z)"
  assumes uminus: "∧w. P w ⇒ P (-w)"
  shows "P z"
proof -
  from assms(1) obtain a b where z_eq: "z = of_ω12_coords (of_int a,
of_int b)"
  by (elim latticeE)
  have nat1: "P (of_ω12_coords (of_nat n, 0))" for n
  by (induction n) (auto simp: of_ω12_coords_def ring_distrib intro:
zero plus gens)
  have int1: "P (of_ω12_coords (of_int n, 0))" for n
  using nat1[of "nat n"] uminus[OF nat1[of "nat (-n)"]]
  by (cases "n ≥ 0") (auto simp: of_ω12_coords_def)
  have nat2: "P (of_ω12_coords (of_int a, of_nat n))" for a n
  proof (induction n)
    case 0
    thus ?case
    using int1[of a] by simp
  next
    case (Suc n)
    from plus[OF Suc gens(2)] show ?case
    by (simp add: of_ω12_coords_def algebra_simps)
  qed
  have int2: "P (of_ω12_coords (of_int a, of_int b))" for a b
  using nat2[of a "nat b"] uminus[OF nat2[of "-a" "nat (-b)"]]

```

```

    by (cases "b ≥ 0") (auto simp: of_ω12_coords_def)
  from this[of a b] and z_eq show ?thesis
    by simp
qed

```

The following equivalence relation equates two points if they differ by a lattice point.

```

definition rel :: "complex ⇒ complex ⇒ bool" where
  "rel x y ↔ (x - y) ∈ Λ"

```

```

lemma rel_refl [simp, intro]: "rel x x"
  by (auto simp: rel_def)

```

```

lemma relE:
  assumes "rel x y"
  obtains z where "z ∈ Λ" "y = x + z"
  using assms unfolding rel_def using pre_complex_lattice.uminus_in_lattice
  by force

```

```

lemma rel_symI: "rel x y ⇒ rel y x"
  by (auto simp: rel_def diff_in_lattice_commute)

```

```

lemma rel_sym: "rel x y ↔ rel y x"
  by (auto simp: rel_def diff_in_lattice_commute)

```

```

lemma rel_0_right_iff: "rel x 0 ↔ x ∈ Λ"
  by (simp add: rel_def)

```

```

lemma rel_0_left_iff: "rel 0 x ↔ x ∈ Λ"
  by (simp add: rel_def uminus_in_lattice_iff)

```

```

lemma rel_trans [trans]: "rel x y ⇒ rel y z ⇒ rel x z"
  using add_in_lattice rel_def by fastforce

```

```

lemma rel_minus [lattice_intros]: "rel a b ⇒ rel (-a) (-b)"
  unfolding rel_def by (simp add: diff_in_lattice_commute)

```

```

lemma rel_minus_iff: "rel (-a) (-b) ↔ rel a b"
  by (auto simp: rel_def diff_in_lattice_commute)

```

```

lemma rel_add [lattice_intros]: "rel a b ⇒ rel c d ⇒ rel (a + c)
(b + d)"
  unfolding rel_def by (simp add: add_diff_add add_in_lattice)

```

```

lemma rel_diff [lattice_intros]: "rel a b ⇒ rel c d ⇒ rel (a - c)
(b - d)"
  by (metis rel_add rel_minus uminus_add_conv_diff)

```

```

lemma rel_mult_of_nat_left [lattice_intros]: "rel a b ⇒ rel (of_nat

```

```

n * a) (of_nat n * b)"
  by (induction n) (auto intro!: lattice_intros simp: ring_distrib)

lemma rel_mult_of_nat_right [lattice_intros]: "rel a b  $\implies$  rel (a *
of_nat n) (b * of_nat n)"
  by (induction n) (auto intro!: lattice_intros simp: ring_distrib)

lemma rel_mult_of_int_left [lattice_intros]: "rel a b  $\implies$  rel (of_int
n * a) (of_int n * b)"
  by (induction n) (auto intro!: lattice_intros simp: ring_distrib)

lemma rel_mult_of_int_right [lattice_intros]: "rel a b  $\implies$  rel (a *
of_int n) (b * of_int n)"
  by (induction n) (auto intro!: lattice_intros simp: ring_distrib)

lemma rel_sum [lattice_intros]:
  " $(\bigwedge i. i \in A \implies \text{rel } (f \ i) \ (g \ i)) \implies \text{rel } (\sum_{i \in A} f \ i) \ (\sum_{i \in A} g \ i)$ "
  by (induction A rule: infinite_finite_induct) (auto intro!: lattice_intros)

lemma rel_sum_list [lattice_intros]:
  "list_all2 rel xs ys  $\implies$  rel (sum_list xs) (sum_list ys)"
  by (induction rule: list_all2_induct) (auto intro!: lattice_intros)

lemma rel_lattice_trans_left [trans]: "x  $\in$   $\Lambda \implies$  rel x y  $\implies$  y  $\in$   $\Lambda$ "
  using rel_0_left_iff rel_trans by blast

lemma rel_lattice_trans_right [trans]: "rel x y  $\implies$  y  $\in$   $\Lambda \implies$  x  $\in$   $\Lambda$ "
  using rel_lattice_trans_left rel_sym by blast

end

Exchanging the two generators clearly does not change the underlying lattice.

locale pre_complex_lattice_swap = pre_complex_lattice
begin

sublocale swap: pre_complex_lattice  $\omega_2 \ \omega_1$  .

lemma swap_of_omega12_coords [simp]: "swap.of_omega12_coords = of_omega12_coords
 $\circ$  prod.swap"
  by (auto simp: fun_eq_iff swap.of_omega12_coords_def of_omega12_coords_def
add_ac)

lemma swap_lattice [simp]: "swap.lattice = lattice"
  unfolding swap.lattice_def lattice_def swap_of_omega12_coords image_comp
[symmetric] product_swap ..

lemma swap_lattice0 [simp]: "swap.lattice0 = lattice0"
  unfolding swap.lattice0_def lattice0_def swap_lattice ..

```

```

lemma swap_rel [simp]: "swap.rel = rel"
  unfolding swap.rel_def rel_def swap_lattice ..

end

A pair  $(\omega_1, \omega_2)$  of complex numbers with  $\omega_2 / \omega_1 \notin \mathbb{R}$  is called a fundamental pair. Two such pairs are called equivalent if

definition fundpair :: "complex  $\times$  complex  $\Rightarrow$  bool" where
  "fundpair = ( $\lambda(a, b). b / a \notin \mathbb{R}$ )"

lemma fundpair_swap: "fundpair ab  $\longleftrightarrow$  fundpair (prod.swap ab)"
  unfolding fundpair_def prod.swap_def case_prod_unfold fst_conv snd_conv
  by (metis Un_insert_right collinear_iff_Reals insert_is_Un)

lemma fundpair_cnj_iff [simp]: "fundpair (cnj a, cnj b) = fundpair (a, b)"
  by (auto simp: fundpair_def complex_is_Real_iff simp flip: complex_cnj_divide)

lemma fundpair_altdef: "fundpair = ( $\lambda(a,b). a / b \notin \mathbb{R}$ )"
proof
  fix ab :: "complex  $\times$  complex"
  show "fundpair ab = (case ab of (a, b)  $\Rightarrow$  a / b  $\notin \mathbb{R}$ )"
    using fundpair_swap[of ab] by (auto simp: fundpair_def)
qed

lemma
  assumes "fundpair (a, b)"
  shows fundpair_imp_nonzero [dest]: "a  $\neq$  0" "b  $\neq$  0"
  and fundpair_imp_neq: "a  $\neq$  b" "b  $\neq$  a"
  using assms unfolding fundpair_def by (auto split: if_splits)

lemma fundpair_imp_independent:
  assumes "fundpair ( $\omega_1, \omega_2$ )"
  shows "independent  $\{\omega_1, \omega_2\}$ "
proof
  assume "dependent  $\{\omega_1, \omega_2\}$ "
  then obtain a b where ab: "a  $\cdot_R \omega_1 + b \cdot_R \omega_2 = 0$ " and "a  $\neq$  0  $\vee$  b  $\neq$  0"
  by (subst (asm) real_vector.dependent_finite) (use assms in <auto dest: fundpair_imp_neq>)
  with assms have [simp]: "a  $\neq$  0" "b  $\neq$  0"
  by auto
  from ab have " $\omega_2 / \omega_1 = \text{of\_real } (-a / b)$ "
  using assms by (auto simp: field_simps scaleR_conv_of_real add_eq_0_iff)
  also have "...  $\in \mathbb{R}$ "
  by simp
  finally show False
  using assms by (auto simp: fundpair_def)

```

qed

```
lemma fundpair_imp_basis:
  assumes "fundpair ( $\omega_1$ ,  $\omega_2$ )"
  shows "span { $\omega_1$ ,  $\omega_2$ } = UNIV"
proof -
  have "dim (span { $\omega_1$ ,  $\omega_2$ }) = card { $\omega_1$ ,  $\omega_2$ }"
    using fundpair_imp_independent[OF assms] by (rule dim_span_eq_card_independent)
  hence "dim (span { $\omega_1$ ,  $\omega_2$ }) = DIM(complex)"
    using fundpair_imp_neq(1)[OF assms] by simp
  thus ?thesis
    using dim_eq_full span_span by blast
qed
```

We now introduce the assumption that the generators be independent. This makes $\{\omega_1, \omega_2\}$ a basis of \mathbb{C} (in the sense of an \mathbb{R} -vector space), and we define a few functions to help us convert between these two views.

```
locale complex_lattice = pre_complex_lattice +
  assumes fundpair: "fundpair ( $\omega_1$ ,  $\omega_2$ )"
begin

definition ratio :: complex ("τ") where "τ =  $\omega_2$  /  $\omega_1$ "

lemma ratio_not_in_Reals: "τ  $\notin$   $\mathbb{R}$ "
  unfolding ratio_def using fundpair by (simp add: fundpair_def)

lemma  $\omega_1$ _neq_ $\omega_2$  [simp]: " $\omega_1 \neq \omega_2$ " and  $\omega_2$ _neq_ $\omega_1$  [simp]: " $\omega_2 \neq \omega_1$ "
  using fundpair fundpair_imp_neq by blast+

lemma  $\omega_1$ _nonzero [simp]: " $\omega_1 \neq 0$ " and  $\omega_2$ _nonzero [simp]: " $\omega_2 \neq 0$ "
  using fundpair by auto

lemma lattice0_nonempty [simp]: "lattice0  $\neq$  {}"
proof -
  have " $\omega_1 \in$  lattice0"
    by auto
  thus ?thesis
    by blast
qed

lemma  $\omega_1\omega_2$ _independent': "independent { $\omega_1$ ,  $\omega_2$ }"
  using fundpair by (rule fundpair_imp_independent)

lemma span_ $\omega_1\omega_2$ : "span { $\omega_1$ ,  $\omega_2$ } = UNIV"
  using fundpair by (rule fundpair_imp_basis)

The following converts complex numbers into lattice coordinates, i.e. as a
linear combination of the two generators.

definition  $\omega_1$ _coord :: "complex  $\Rightarrow$  real" where
```

```

"ω1_coord z = representation {ω1, ω2} z ω1"

definition ω2_coord :: "complex ⇒ real" where
  "ω2_coord z = representation {ω1, ω2} z ω2"

definition ω12_coords :: "complex ⇒ real × real" where
  "ω12_coords z = (ω1_coord z, ω2_coord z)"

sublocale ω1_coord: bounded_linear ω1_coord
  unfolding ω1_coord_def using ω12_independent' span_ω12
  by (rule bounded_linear_representation)

sublocale ω2_coord: bounded_linear ω2_coord
  unfolding ω2_coord_def using ω12_independent' span_ω12
  by (rule bounded_linear_representation)

sublocale ω12_coords: linear ω12_coords
  unfolding ω12_coords_def
  by (auto simp: linear_iff ω1_coord.add ω2_coord.add ω1_coord.scaleR
ω2_coord.scaleR)

sublocale ω12_coords: bounded_linear ω12_coords
  using ω12_coords.linear_axioms linear_conv_bounded_linear by auto

lemmas [continuous_intros] =
  ω1_coord.continuous_on ω1_coord.continuous
  ω2_coord.continuous_on ω2_coord.continuous
  ω12_coords.continuous_on ω12_coords.continuous

lemmas [tendsto_intros] = ω1_coord.tendsto ω2_coord.tendsto ω12_coords.tendsto

lemma ω1_coord_ω1 [simp]: "ω1_coord ω1 = 1"
  and ω1_coord_ω2 [simp]: "ω1_coord ω2 = 0"
  and ω2_coord_ω1 [simp]: "ω2_coord ω1 = 0"
  and ω2_coord_ω2 [simp]: "ω2_coord ω2 = 1"
  unfolding ω1_coord_def ω2_coord_def using ω1_neq_ω2
  by (simp_all add: ω12_independent' representation_basis)

lemma ω12_coords_ω1 [simp]: "ω12_coords ω1 = (1, 0)"
  and ω12_coords_ω2 [simp]: "ω12_coords ω2 = (0, 1)"
  by (simp_all add: ω12_coords_def)

lemma ω12_coords_of_ω12_coords [simp]: "ω12_coords (of_ω12_coords z)
= z"
  by (simp add: of_ω12_coords_def case_prod_unfold ω12_coords.add ω12_coords.scaleR
flip: scaleR_conv_of_real)

lemma ω1_coord_of_ω12_coords [simp]: "ω1_coord (of_ω12_coords z) =
fst z"

```

```

    and  $\omega_2$ _coord_of_ $\omega_12$ _coords [simp]: " $\omega_2$ _coord (of_ $\omega_12$ _coords z) = snd z"
    using  $\omega_12$ _coords_of_ $\omega_12$ _coords[of z]
    by (auto simp del:  $\omega_12$ _coords_of_ $\omega_12$ _coords simp add:  $\omega_12$ _coords_def prod_eq_iff)

```

```

lemma of_ $\omega_12$ _coords_ $\omega_12$ _coords [simp]: "of_ $\omega_12$ _coords ( $\omega_12$ _coords z) = z"

```

```

proof -

```

```

  have " $(\sum b \in \{\omega_1, \omega_2\}. \text{representation } \{\omega_1, \omega_2\} z b *_{\mathbb{R}} b) = z$ "
    by (rule real_vector.sum_representation_eq) (use  $\omega_12$ _independent' span_ $\omega_12$  in simp_all)
  thus ?thesis
    by (simp add:  $\omega_12$ _coords_def of_ $\omega_12$ _coords_def scaleR_conv_of_real  $\omega_1$ _coord_def  $\omega_2$ _coord_def  $\omega_1$ _neq_ $\omega_2$ )

```

```

qed

```

```

lemma  $\omega_12$ _coords_eqI:

```

```

  assumes "of_ $\omega_12$ _coords a = b"
  shows "  $\omega_12$ _coords b = a"
  unfolding assms[symmetric] by auto

```

```

lemmas [simp] =  $\omega_1$ _coord.scaleR  $\omega_2$ _coord.scaleR  $\omega_12$ _coords.scaleR

```

```

lemma  $\omega_12$ _coords_times_ $\omega_1$  [simp]: " $\omega_12$ _coords (of_real a *  $\omega_1$ ) = (a, 0)"

```

```

  and  $\omega_12$ _coords_times_ $\omega_2$  [simp]: " $\omega_12$ _coords (of_real a *  $\omega_2$ ) = (0, a)"

```

```

  and  $\omega_12$ _coords_times_ $\omega_1$ ' [simp]: " $\omega_12$ _coords ( $\omega_1$  * of_real a) = (a, 0)"

```

```

  and  $\omega_12$ _coords_times_ $\omega_2$ ' [simp]: " $\omega_12$ _coords ( $\omega_2$  * of_real a) = (0, a)"

```

```

  and  $\omega_12$ _coords_mult_of_real [simp]: " $\omega_12$ _coords (of_real c * z) = c *_{\mathbb{R}}  $\omega_12$ _coords z"

```

```

  and  $\omega_12$ _coords_mult_of_int [simp]: " $\omega_12$ _coords (of_int i * z) = of_int i *_{\mathbb{R}}  $\omega_12$ _coords z"

```

```

  and  $\omega_12$ _coords_mult_of_nat [simp]: " $\omega_12$ _coords (of_nat n * z) = of_nat n *_{\mathbb{R}}  $\omega_12$ _coords z"

```

```

  and  $\omega_12$ _coords_divide_of_real [simp]: " $\omega_12$ _coords (z / of_real c) =  $\omega_12$ _coords z /_{\mathbb{R}} c"

```

```

  and  $\omega_12$ _coords_mult_numeral [simp]: " $\omega_12$ _coords (numeral num * z) = numeral num *_{\mathbb{R}}  $\omega_12$ _coords z"

```

```

  and  $\omega_12$ _coords_divide_numeral [simp]: " $\omega_12$ _coords (z / numeral num) =  $\omega_12$ _coords z /_{\mathbb{R}} numeral num"

```

```

  by (rule  $\omega_12$ _coords_eqI; simp add: scaleR_conv_of_real field_simps; fail)+

```

```

lemma of_ $\omega_12$ _coords_eq_iff: "of_ $\omega_12$ _coords z1 = of_ $\omega_12$ _coords z2  $\iff$  z1 = z2"

```

```

using  $\omega12\_coords\_eqI$  by blast

lemma  $\omega12\_coords\_eq\_iff$ : " $\omega12\_coords\ z1 = \omega12\_coords\ z2 \iff z1 = z2$ "
  by (metis of_ $\omega12\_coords\_omega12\_coords$ )

lemma of_ $\omega12\_coords\_eq\_0\_iff$  [simp]: "of_ $\omega12\_coords\ z = 0 \iff z = (0,0)$ "
  unfolding zero_prod_def [symmetric]
  by (metis  $\omega12\_coords.zero\ \omega12\_coords\_eqI$  of_ $\omega12\_coords\_omega12\_coords$ )

lemma  $\omega12\_coords\_eq\_0\_0\_iff$  [simp]: " $\omega12\_coords\ x = (0, 0) \iff x = 0$ "
  by (metis  $\omega12\_coords.zero\ \omega12\_coords\_eq\_iff\ zero\_prod\_def$ )

lemma bij_of_ $\omega12\_coords$ : "bij of_ $\omega12\_coords$ "
proof -
  have " $\exists z'. z = of\_omega12\_coords\ z'$ " for z
    by (rule exI[of _ " $\omega12\_coords\ z$ "]) auto
  hence "surj of_ $\omega12\_coords$ "
    by blast
  thus ?thesis
    unfolding bij_def by (auto intro!: injI simp: of_ $\omega12\_coords\_eq\_iff$ )
qed

lemma bij_betw_lattice: "bij_betw of_ $\omega12\_coords\ (\mathbb{Z} \times \mathbb{Z})\ lattice$ "
  unfolding lattice_def using bij_of_ $\omega12\_coords$  unfolding bij_betw_def
  inj_on_def by blast

lemma bij_betw_lattice0: "bij_betw of_ $\omega12\_coords\ (\mathbb{Z} \times \mathbb{Z} - \{(0,0)\})\ lattice0$ "
  unfolding lattice0_def by (intro bij_betw_DiffI bij_betw_singletonI
  bij_betw_lattice) auto

lemma bij_betw_lattice': "bij_betw (of_ $\omega12\_coords \circ map\_prod\ of\_int\ of\_int$ ) UNIV lattice"
  by (rule bij_betw_trans[OF _ bij_betw_lattice]) (auto simp: Ints_def
  bij_betw_def inj_on_def)

lemma bij_betw_lattice0': "bij_betw (of_ $\omega12\_coords \circ map\_prod\ of\_int\ of\_int$ ) (-{(0,0)}) lattice0"
  by (rule bij_betw_trans[OF _ bij_betw_lattice0]) (auto simp: Ints_def
  bij_betw_def inj_on_def)

lemma infinite_lattice: " $\neg finite\ lattice$ "
proof -
  have "finite (UNIV :: (int  $\times$  int) set)  $\iff$  finite lattice"
    by (rule bij_betw_finite[OF bij_betw_lattice'])
  moreover have " $\neg finite\ (UNIV :: (int \times int)\ set)$ "
    by (simp add: finite_prod)
  ultimately show ?thesis

```

```

    by blast
qed

lemma  $\omega_{12}$ _coords_image_eq: " $\omega_{12}$ _coords ' X = of_ $\omega_{12}$ _coords -' X"
  using bij_of_ $\omega_{12}$ _coords image_iff by fastforce

lemma of_ $\omega_{12}$ _coords_image_eq: "of_ $\omega_{12}$ _coords ' X =  $\omega_{12}$ _coords -' X"
  by (metis UNIV_I  $\omega_{12}$ _coords_eqI  $\omega_{12}$ _coords_image_eq bij_betw_imp_surj_on
    bij_of_ $\omega_{12}$ _coords rangeI subsetI subset_antisym surj_image_vimage_eq)

lemma of_ $\omega_{12}$ _coords_linepath:
  " $\omega_{12}$ _coords (linepath a b x) = linepath (of_ $\omega_{12}$ _coords a) (of_ $\omega_{12}$ _coords
  b) x"
  by (simp add: linepath_def scaleR_prod_def scaleR_conv_of_real
    of_ $\omega_{12}$ _coords_def algebra_simps case_prod_unfold)

lemma of_ $\omega_{12}$ _coords_linepath':
  " $\omega_{12}$ _coords o (linepath a b) =
  linepath (of_ $\omega_{12}$ _coords a) (of_ $\omega_{12}$ _coords b)"
  unfolding comp_def using of_ $\omega_{12}$ _coords_linepath
  by auto

lemma  $\omega_{12}$ _coords_linepath:
  " $\omega_{12}$ _coords (linepath a b x) = linepath ( $\omega_{12}$ _coords a) ( $\omega_{12}$ _coords
  b) x"
  by (rule  $\omega_{12}$ _coords_eqI) (simp add: of_ $\omega_{12}$ _coords_linepath)

lemma of_ $\omega_{12}$ _coords_in_lattice_iff:
  " $\omega_{12}$ _coords z  $\in$   $\Lambda$   $\longleftrightarrow$  fst z  $\in$   $\mathbb{Z}$   $\wedge$  snd z  $\in$   $\mathbb{Z}$ "
proof
  assume " $\omega_{12}$ _coords z  $\in$   $\Lambda$ "
  then obtain m n where mn: " $\omega_{12}$ _coords z = of_ $\omega_{12}$ _coords (of_int
  m, of_int n)"
  by (elim latticeE)
  hence "z = (of_int m, of_int n)"
  by (simp only: of_ $\omega_{12}$ _coords_eq_iff)
  thus "fst z  $\in$   $\mathbb{Z}$   $\wedge$  snd z  $\in$   $\mathbb{Z}$ "
  by auto
next
  assume "fst z  $\in$   $\mathbb{Z}$   $\wedge$  snd z  $\in$   $\mathbb{Z}$ "
  thus " $\omega_{12}$ _coords z  $\in$   $\Lambda$ "
  by (simp add: latticeI of_ $\omega_{12}$ _coords_def split_def)
qed

lemma of_ $\omega_{12}$ _coords_in_lattice [simp, intro]:
  "fst z  $\in$   $\mathbb{Z}$   $\implies$  snd z  $\in$   $\mathbb{Z}$   $\implies$  of_ $\omega_{12}$ _coords z  $\in$   $\Lambda$ "
  by (subst of_ $\omega_{12}$ _coords_in_lattice_iff) auto

lemma in_lattice_conv_ $\omega_{12}$ _coords: "z  $\in$   $\Lambda$   $\longleftrightarrow$   $\omega_{12}$ _coords z  $\in$   $\mathbb{Z} \times \mathbb{Z}$ "

```

```

    using of_ω12_coords_in_lattice_iff[of "ω12_coords z"] by (auto simp:
mem_Times_iff)

```

```

lemma ω12_coords_in_Z_times_Z: "z ∈ Λ ⇒ ω12_coords z ∈ ℤ × ℤ"
  by (subst (asm) in_lattice_conv_ω12_coords) auto

```

```

lemma half_periods_notin_lattice [simp]:
  "ω1 / 2 ∉ Λ" "ω2 / 2 ∉ Λ" "(ω1 + ω2) / 2 ∉ Λ"
  by (auto simp: in_lattice_conv_ω12_coords ω12_coords.add)

```

```

end

```

```

locale complex_lattice_swap = complex_lattice
begin

```

```

  sublocale pre_complex_lattice_swap ω1 ω2 .

```

```

  sublocale swap: complex_lattice ω2 ω1
  proof
    show "fundpair (ω2, ω1)"
      using fundpair by (subst (asm) fundpair_swap) auto
  qed

```

```

lemma swap_ω12_coords [simp]: "swap.ω12_coords = prod.swap ∘ ω12_coords"
  by (metis (no_types, lifting) ext comp_apply of_ω12_coords_ω12_coords
pre_complex_lattice_swap.swap_of_ω12_coords swap.ω12_coords_eqI)

```

```

lemma swap_ω1_coord [simp]: "swap.ω1_coord = ω2_coord"
  and swap_ω2_coord [simp]: "swap.ω2_coord = ω1_coord"
  using swap_ω12_coords unfolding swap.ω12_coords_def[abs_def] ω12_coords_def[abs_def]
  by (auto simp: fun_eq_iff)

```

```

end

```

3.2 Period parallelograms

```

context pre_complex_lattice
begin

```

The period parallelogram at a vertex z is the parallelogram with the vertices z , $z + \omega_1$, $z + \omega_2$, and $z + \omega_1 + \omega_2$. For convenience, we define the parallelogram to be contain only two of its four sides, so that one can obtain an exact covering of the complex plane with period parallelograms.

We will occasionally need the full parallelogram with all four sides, or the interior of the parallelogram without its four sides, but these are easily obtained from this using the *closure* and *interior* operators, while the border itself (which is of interest for integration) is obtained with the *frontier* operator.

```

definition period_parallelogram :: "complex  $\Rightarrow$  complex set" where
  "period_parallelogram z = (+) z ` of_ω12_coords ` ({0..<1}  $\times$  {0..<1})"

```

The following is a path along the border of a period parallelogram, starting at the vertex z and going in direction ω_1 .

```

definition period_parallelogram_path :: "complex  $\Rightarrow$  real  $\Rightarrow$  complex" where
  "period_parallelogram_path z  $\equiv$  parallelogram_path z ω1 ω2"

```

```

lemma bounded_period_parallelogram [intro]: "bounded (period_parallelogram z)"
  unfolding period_parallelogram_def
  by (rule bounded_translation bounded_linear_image bounded_Times)+
  (auto intro: of_ω12_coords.bounded_linear_axioms)

```

```

lemma convex_period_parallelogram [intro]:
  "convex (period_parallelogram z)"
  unfolding period_parallelogram_def
  by (intro convex_translation convex_linear_image of_ω12_coords.linear_axioms
convex_Times) auto

```

```

lemma closure_period_parallelogram:
  "closure (period_parallelogram z) = (+) z ` of_ω12_coords ` (cbox (0,0)
(1,1))"
proof -
  have "closure (period_parallelogram z) = (+) z ` closure (of_ω12_coords
` ({0..<1}  $\times$  {0..<1}))"
    unfolding period_parallelogram_def by (subst closure_translation)
  auto
  also have "closure (of_ω12_coords ` ({0..<1}  $\times$  {0..<1})) =
of_ω12_coords ` (closure ({0..<1}  $\times$  {0..<1}))"
    by (rule closure_bounded_linear_image [symmetric])
    (auto intro: bounded_Times of_ω12_coords.linear_axioms)
  also have "closure ({0..<1>::real}  $\times$  {0..<1>::real}) = {0..1}  $\times$  {0..1}"
    by (simp add: closure_Times)
  also have "... = cbox (0,0) (1,1)"
    by auto
  finally show ?thesis .
qed

```

```

lemma compact_closure_period_parallelogram [intro]: "compact (closure
(period_parallelogram z))"
  unfolding closure_period_parallelogram
  by (intro compact_translation compact_continuous_image continuous_intros
compact_Times) auto

```

```

lemma vertex_in_period_parallelogram [simp, intro]: "z  $\in$  period_parallelogram
z"
  unfolding period_parallelogram_def image_image
  by (rule image_eqI[of _ _ "(0,0)"]) auto

```

```

lemma nonempty_period_parallelogram: "period_parallelogram z ≠ {}"
  using vertex_in_period_parallelogram[of z] by blast

end

lemma (in pre_complex_lattice_swap)
  swap_period_parallelogram [simp]: "swap.period_parallelogram = period_parallelogram"
  unfolding swap.period_parallelogram_def period_parallelogram_def swap_of_ω12_coords
    image_comp [symmetric] product_swap ..

context complex_lattice
begin

lemma simple_path_parallelogram: "simple_path (parallelogram_path z ω1
ω2)"
  unfolding parallelogram_path_altdef
proof (rule simple_path_continuous_image)
  let ?h = "λw. z + Re w *R ω1 + Im w *R ω2"
  show "simple_path (rectpath 0 (1 + i))"
    by (intro simple_path_rectpath) auto
  show "continuous_on (path_image (rectpath 0 (1 + i))) ?h"
    by (intro continuous_intros)
  show "inj_on ?h (path_image (rectpath 0 (1 + i)))"
  proof
    fix x y assume "?h x = ?h y"
    hence "of_ω12_coords (Re x, Im x) = of_ω12_coords (Re y, Im y)"
      by (simp add: of_ω12_coords_def scaleR_conv_of_real)
    thus "x = y"
      by (intro complex_eqI) (simp_all add: of_ω12_coords_eq_iff)
  qed
qed

lemma period_parallelogram_altdef:
  "period_parallelogram z = {w. ω12_coords (w - z) ∈ {0..<1} × {0..<1}}"
  unfolding period_parallelogram_def of_ω12_coords_image_eq image_plus_conv_vimage_plus
  by auto

lemma interior_period_parallelogram:
  "interior (period_parallelogram z) = (+) z ' of_ω12_coords ' box (0,0)
(1,1)"
proof -
  have bij: "bij of_ω12_coords"
    by (simp add: bij_of_ω12_coords)
  have "interior (period_parallelogram z) = (+) z ' interior (of_ω12_coords
' ({0..<1} × {0..<1}))"
    unfolding period_parallelogram_def by (subst interior_translation)

```

```

auto
  also have "interior (of_ω12_coords ' ({0..<1} × {0..<1})) =
            of_ω12_coords ' (interior ({0..<1} × {0..<1}))"
    using of_ω12_coords.linear_axioms bij
    by (rule interior_bijective_linear_image)
  also have "interior ({0..<1::real} × {0..<1::real}) = {0<..<1} × {0<..<1}"
    by (subst interior_Times) simp_all
  finally show ?thesis by (auto simp: box_prod)
qed

lemma path_image_parallelogram_path':
  "path_image (parallelogram_path z ω1 ω2) =
    (+) z ' of_ω12_coords ' (cbox (0,0) (1,1) - box (0,0) (1,1))"
proof -
  define f where "f = (λx. (Re x, Im x))"
  have "bij f"
    by (rule bij_betwI[of _ _ _ "λ(x,y). Complex x y"]) (auto simp: f_def)
  hence "inj f" "surj f"
    using bij_is_inj bij_is_surj by auto
  have "path_image (parallelogram_path z ω1 ω2) =
    (λw. z + Re w *R ω1 + Im w *R ω2) ' (cbox 0 (1 + i) - box 0
(1 + i))"
    (is "_ = _ ' ?S")
    unfolding parallelogram_path_altdef period_parallelogram_altdef path_image_compose
    by (subst path_image_rectpath_cbox_minus_box) auto
  also have "(λw. z + Re w *R ω1 + Im w *R ω2) = (+) z ◦ of_ω12_coords
◦ f"
    by (auto simp: of_ω12_coords_def fun_eq_iff scaleR_conv_of_real f_def)
  also have "... ' (cbox 0 (1 + i) - box 0 (1 + i)) =
    (+) z ' of_ω12_coords ' f ' ((cbox 0 (1 + i) - box 0 (1 +
i)))"
    by (simp add: image_image)
  also have "f ' ((cbox 0 (1 + i) - box 0 (1 + i))) = f ' cbox 0 (1 + i)
- f ' box 0 (1 + i)"
    by (rule image_set_diff[OF <inj f>])
  also have "cbox 0 (1 + i) = f -' cbox (0,0) (1,1)"
    by (auto simp: f_def cbox_complex_eq)
  also have "f ' ... = cbox (0,0) (1,1)"
    by (rule surj_image_vimage_eq[OF <surj f>])
  also have "box 0 (1 + i) = f -' box (0,0) (1,1)"
    by (auto simp: f_def box_complex_eq box_prod)
  also have "f ' ... = box (0,0) (1,1)"
    by (rule surj_image_vimage_eq[OF <surj f>])
  finally show ?thesis .
qed

lemma fund_period_parallelogram_in_lattice_iff:
  assumes "z ∈ period_parallelogram 0"
  shows "z ∈ Λ ↔ z = 0"

```

```

proof
  assume "z ∈ Λ"
  then obtain m n where mn: "z = of_ω12_coords (of_int m, of_int n)"
    by (elim latticeE)
  show "z = 0"
    using assms unfolding mn period_parallelogram_altdef by auto
qed auto

lemma path_image_parallelogram_path:
  "path_image (parallelogram_path z ω1 ω2) = frontier (period_parallelogram
z)"
  unfolding frontier_def closure_period_parallelogram interior_period_parallelogram
    path_image_parallelogram_path'
  by (subst image_set_diff) (auto intro!: inj_onI simp: of_ω12_coords_eq_iff)

lemma path_image_parallelogram_subset_closure:
  "path_image (parallelogram_path z ω1 ω2) ⊆ closure (period_parallelogram
z)"
  unfolding path_image_parallelogram_path' closure_period_parallelogram
  by (intro image_mono) auto

lemma path_image_parallelogram_disjoint_interior:
  "path_image (parallelogram_path z ω1 ω2) ∩ interior (period_parallelogram
z) = {}"
  unfolding path_image_parallelogram_path' interior_period_parallelogram
  by (auto simp: of_ω12_coords_eq_iff box_prod)

lemma winding_number_parallelogram_outside:
  assumes "w ∉ closure (period_parallelogram z)"
  shows "winding_number (parallelogram_path z ω1 ω2) w = 0"
  by (rule winding_number_zero_outside[OF _ _ _ assms])
  (use path_image_parallelogram_subset_closure[of z] in auto)

The path we take around the period parallelogram is clearly a simple path,
and its orientation depends on the angle between our generators.

lemma winding_number_parallelogram_inside:
  assumes "w ∈ interior (period_parallelogram z)"
  shows "winding_number (parallelogram_path z ω1 ω2) w = sgn (Im (ω2
/ ω1))"
proof -
  let ?P = "parallelogram_path z ω1 ω2"
  have w: "w ∉ path_image ?P"
    using assms unfolding interior_period_parallelogram path_image_parallelogram_path'
    by (auto simp: of_ω12_coords_eq_iff box_prod)
  define ind where "ind = (λa b. winding_number (linepath (z + a) (z
+b)) w)"
  define u where "u = w - z"
  define x y where "x = ω1_coord u" and "y = ω2_coord u"
  have u_eq: "u = of_ω12_coords (x, y)"

```

```

    by (simp_all add: x_def y_def flip: ω12_coords_def)
  have xy: "x ∈ {0<..<1}" "y ∈ {0<..<1}" using assms(1)
    unfolding interior_period_parallelogram image_plus_conv_vimage_plus
of_ω12_coords_image_eq
    by (auto simp: x_def y_def ω12_coords_def u_def box_prod)
  have w_eq: "w = z + of_ω12_coords (x, y)"
    using u_eq by (simp add: u_def algebra_simps)

  define W where "W = winding_number (parallelogram_path z ω1 ω2) w"
  have Re_W_eq: "Re W = Re (ind 0 ω1) + Re (ind ω1 (ω1 + ω2)) + Re (ind
(ω1 + ω2) ω2) + Re (ind ω2 0)"
    using w unfolding W_def parallelogram_path_def
    by (simp add: winding_number_join ind_def path_image_join add_ac)

show ?thesis
proof (cases "Im (ω2 / ω1)" "0::real" rule: linorder_cases)
  case equal
  hence False
    using fundpair complex_is_Real_iff by (auto simp: fundpair_def)
  thus ?thesis ..

next
  case greater
  have "W = 1"
    unfolding W_def
  proof (rule simple_closed_path_winding_number_pos; (fold W_def)?)
    from greater have neg: "Im (ω1 * cnj ω2) < 0"
      by (subst (asm) Im_complex_div_gt_0) (auto simp: mult_ac)

    have pos1: "Re (ind 0 ω1) > 0"
    proof -
      have "Im ((z + ω1 - (z + 0)) * cnj (z + ω1 - w)) = y * (-Im (ω1
* cnj ω2))"
        by (simp add: w_eq algebra_simps of_ω12_coords_def)
      also have "... > 0"
        using neg xy by (intro mult_pos_pos) auto
      finally show ?thesis
        unfolding ind_def by (rule winding_number_linepath_pos_lt)
    qed

    have pos2: "Re (ind ω1 (ω1 + ω2)) > 0"
    proof -
      have "Im ((z + (ω1 + ω2) - (z + ω1)) * cnj (z + (ω1 + ω2) -
w)) =
        (1 - x) * (-Im (ω1 * cnj ω2))"
        by (simp add: w_eq algebra_simps of_ω12_coords_def)
      also have "... > 0"
        using neg xy by (intro mult_pos_pos) auto
      finally show ?thesis

```

```

      unfolding ind_def by (rule winding_number_linepath_pos_lt)
    qed

  have pos3: "Re (ind (ω1 + ω2) ω2) > 0"
  proof -
    have "Im ((z + ω2 - (z + (ω1 + ω2))) * cnj (z + ω2 - w)) =
      (1 - y) * (-Im (ω1 * cnj ω2))"
      by (simp add: w_eq algebra_simps of_ω12_coords_def)
    also have "... > 0"
      using neg xy by (intro mult_pos_pos) auto
    finally show ?thesis
      unfolding ind_def by (rule winding_number_linepath_pos_lt)
  qed

  have pos4: "Re (ind ω2 0) > 0"
  proof -
    have "Im ((z + 0 - (z + ω2)) * cnj (z + 0 - w)) =
      x * (-Im (ω1 * cnj ω2))"
      by (simp add: w_eq algebra_simps of_ω12_coords_def)
    also have "... > 0"
      using neg xy by (intro mult_pos_pos) auto
    finally show ?thesis
      unfolding ind_def by (rule winding_number_linepath_pos_lt)
  qed

  show "Re W > 0"
    using pos1 pos2 pos3 pos4 unfolding Re_W_eq by linarith
  qed (use w in <auto intro: simple_path_parallelogram>)

  thus ?thesis
    using greater by (simp add: W_def)

next
case less

  have "W = -1"
    unfolding W_def
  proof (rule simple_closed_path_winding_number_neg; (fold W_def)?)
    from less have neg: "Im (ω2 * cnj ω1) < 0"
      by (simp add: Im_complex_div_lt_0)

    have neg1: "Re (ind 0 ω1) < 0"
    proof -
      have "Im ((z + 0 - (z + ω1)) * cnj (z + 0 - w)) = y * (-Im (ω2
* cnj ω1))"
        by (simp add: w_eq algebra_simps of_ω12_coords_def)
      also have "... > 0"
        using neg xy by (intro mult_pos_pos) auto
      finally show ?thesis
    end
  end

```

```

    unfolding ind_def by (rule winding_number_linepath_neg_lt)
  qed

  have neg2: "Re (ind  $\omega_1$  ( $\omega_1 + \omega_2$ )) < 0"
  proof -
    have "Im ((z +  $\omega_1$  - (z + ( $\omega_1 + \omega_2$ ))) * cnj (z +  $\omega_1$  - w)) =
      (1 - x) * (-Im ( $\omega_2$  * cnj  $\omega_1$ ))"
      by (simp add: w_eq algebra_simps of_omega12_coords_def)
    also have "... > 0"
      using neg xy by (intro mult_pos_pos) auto
    finally show ?thesis
      unfolding ind_def by (rule winding_number_linepath_neg_lt)
  qed

  have neg3: "Re (ind ( $\omega_1 + \omega_2$ )  $\omega_2$ ) < 0"
  proof -
    have "Im ((z + ( $\omega_1 + \omega_2$ ) - (z +  $\omega_2$ )) * cnj (z + ( $\omega_1 + \omega_2$ ) -
w)) =
      (1 - y) * (-Im ( $\omega_2$  * cnj  $\omega_1$ ))"
      by (simp add: w_eq algebra_simps of_omega12_coords_def)
    also have "... > 0"
      using neg xy by (intro mult_pos_pos) auto
    finally show ?thesis
      unfolding ind_def by (rule winding_number_linepath_neg_lt)
  qed

  have neg4: "Re (ind  $\omega_2$  0) < 0"
  proof -
    have "Im ((z +  $\omega_2$  - (z + 0)) * cnj (z +  $\omega_2$  - w)) =
      x * (-Im ( $\omega_2$  * cnj  $\omega_1$ ))"
      by (simp add: w_eq algebra_simps of_omega12_coords_def)
    also have "... > 0"
      using neg xy by (intro mult_pos_pos) auto
    finally show ?thesis
      unfolding ind_def by (rule winding_number_linepath_neg_lt)
  qed

  show "Re W < 0"
    using neg1 neg2 neg3 neg4 unfolding Re_W_eq by linarith
  qed (use w in <auto intro: simple_path_parallelagram>)

  thus ?thesis
    using less by (simp add: W_def)
  qed
qed
end

```

3.3 Canonical representatives and the fundamental parallelogram

```
context complex_lattice
begin
```

The following function maps any complex number z to its canonical representative z' in the fundamental period parallelogram.

```
definition to_fund_parallelogram :: "complex  $\Rightarrow$  complex" where
  "to_fund_parallelogram z =
    (case  $\omega_{12}$ _coords z of (a, b)  $\Rightarrow$  of_ $\omega_{12}$ _coords (frac a, frac b))"
```

```
lemma to_fund_parallelogram_in_parallelogram [intro]:
  "to_fund_parallelogram z  $\in$  period_parallelogram 0"
  unfolding to_fund_parallelogram_def
  by (auto simp: period_parallelogram_altdef case_prod_unfold frac_lt_1)
```

```
lemma  $\omega_1$ _coord_to_fund_parallelogram [simp]: " $\omega_1$ _coord (to_fund_parallelogram z) = frac ( $\omega_1$ _coord z)"
  and  $\omega_2$ _coord_to_fund_parallelogram [simp]: " $\omega_2$ _coord (to_fund_parallelogram z) = frac ( $\omega_2$ _coord z)"
  by (auto simp: to_fund_parallelogram_def case_prod_unfold  $\omega_{12}$ _coords_def)
```

```
lemma to_fund_parallelogramE:
  obtains m n where "to_fund_parallelogram z = z + of_int m *  $\omega_1$  + of_int n *  $\omega_2$ "
```

```
proof -
  define m where "m = floor (fst ( $\omega_{12}$ _coords z))"
  define n where "n = floor (snd ( $\omega_{12}$ _coords z))"
  have "z - of_int m *  $\omega_1$  - of_int n *  $\omega_2$  =
    of_ $\omega_{12}$ _coords ( $\omega_{12}$ _coords z) - of_int m *  $\omega_1$  - of_int n *  $\omega_2$ "
    by (simp add: m_def n_def)
  also have "... = to_fund_parallelogram z"
    unfolding of_ $\omega_{12}$ _coords_def
    by (simp add: case_prod_unfold to_fund_parallelogram_def frac_def m_def n_def of_ $\omega_{12}$ _coords_def algebra_simps)
  finally show ?thesis
    by (intro that[of "-m" "-n"]) auto
```

qed

```
lemma rel_to_fund_parallelogram_left: "rel (to_fund_parallelogram z) z"
```

```
proof -
  obtain m n where "to_fund_parallelogram z = z + of_int m *  $\omega_1$  + of_int n *  $\omega_2$ "
  by (elim to_fund_parallelogramE)
  hence "to_fund_parallelogram z - z = of_int m *  $\omega_1$  + of_int n *  $\omega_2$ "
  by Groebner_Basis.algebra
  also have "... = of_ $\omega_{12}$ _coords (of_int m, of_int n)"
```

```

    by (simp add: of_ω12_coords_def)
  also have "... ∈ Λ"
    by (rule of_ω12_coords_in_lattice) auto
  finally show ?thesis
    by (simp add: rel_def)
qed

lemma rel_to_fund_parallelogram_right: "rel z (to_fund_parallelogram
z)"
  using rel_to_fund_parallelogram_left[of z] by (simp add: rel_sym)

lemma rel_to_fund_parallelogram_left_iff [simp]: "rel (to_fund_parallelogram
z) w ⟷ rel z w"
  using rel_sym rel_to_fund_parallelogram_right rel_trans by blast

lemma rel_to_fund_parallelogram_right_iff [simp]: "rel z (to_fund_parallelogram
w) ⟷ rel z w"
  using rel_sym rel_to_fund_parallelogram_left rel_trans by blast

lemma to_fund_parallelogram_in_lattice_iff [simp]:
  "to_fund_parallelogram z ∈ lattice ⟷ z ∈ lattice"
  using pre_complex_lattice.rel_0_left_iff rel_to_fund_parallelogram_right_iff
  by blast

lemma to_fund_parallelogram_in_lattice [lattice_intros]:
  "z ∈ lattice ⟹ to_fund_parallelogram z ∈ lattice"
  by simp

to_fund_parallelogram is a bijective map from any period parallelogram to
the standard period parallelogram:

lemma bij_betw_to_fund_parallelogram:
  "bij_betw to_fund_parallelogram (period_parallelogram orig) (period_parallelogram
0)"
proof -
  have "bij_betw (of_ω12_coords ∘ map_prod frac frac ∘ ω12_coords)
(period_parallelogram orig) (period_parallelogram 0)"
  proof (intro bij_betw_trans)
    show "bij_betw of_ω12_coords ({0..<1}×{0..<1}) (period_parallelogram
0)"
      by (rule bij_betwI[of _ _ _ ω12_coords]) (auto simp: period_parallelogram_altdef)
  next
    define a b where "a = ω1_coord orig" "b = ω2_coord orig"
    have orig_eq: "orig = of_ω12_coords (a, b)"
      by (auto simp: a_b_def simp flip: ω12_coords_def)

    show "bij_betw ω12_coords (period_parallelogram orig)
(ω1_coord orig..<ω1_coord orig+1} × {ω2_coord orig..<ω2_coord
orig+1})"
      proof (rule bij_betwI[of _ _ _ of_ω12_coords])

```

```

    show "ω12_coords
        ∈ period_parallelogram orig →
        {ω1_coord orig..<ω1_coord orig + 1} × {ω2_coord orig..<ω2_coord
orig + 1}"
    by (auto simp: orig_eq period_parallelogram_def period_parallelogram_altdef
ω12_coords.add)
    next
    show "of_ω12_coords ∈ {ω1_coord orig..<ω1_coord orig + 1} × {ω2_coord
orig..<ω2_coord orig + 1} →
        period_parallelogram orig"
    proof safe
    fix c d :: real
    assume c: "c ∈ {ω1_coord orig..<ω1_coord orig + 1}"
    assume d: "d ∈ {ω2_coord orig..<ω2_coord orig + 1}"
    have "of_ω12_coords (c, d) = of_ω12_coords (a, b) + of_ω12_coords
(c - a, d - b)"
    by (simp add: of_ω12_coords_def algebra_simps)
    moreover have "(c - a, d - b) ∈ {0..<1} × {0..<1}"
    using c d unfolding a_b_def [symmetric] by auto
    ultimately show "of_ω12_coords (c, d) ∈ period_parallelogram
orig"
    unfolding period_parallelogram_def period_parallelogram_altdef
orig_eq image_image
    by auto
    qed
    qed auto
    next
    show "bij_betw (map_prod frac frac)
        ({ω1_coord orig..<ω1_coord orig + 1} × {ω2_coord orig..<ω2_coord
orig + 1})
        ({0..<1} × {0..<1})"
    by (intro bij_betw_map_prod bij_betw_frac)
    qed
    also have "of_ω12_coords ∘ map_prod frac frac ∘ ω12_coords =
        to_fund_parallelogram"
    by (auto simp: o_def to_fund_parallelogram_def fun_eq_iff case_prod_unfold
map_prod_def)
    finally show ?thesis .
    qed

```

There exists a bijection between any two period parallelograms that always maps points to equivalent points.

lemma *bij_betw_period_parallelograms:*

obtains *f* **where**

"bij_betw *f* (period_parallelogram orig) (period_parallelogram orig)"

" $\wedge z. \text{rel } (f z) z$ "

proof -

define *h* **where** "*h* = inv_into (period_parallelogram orig') to_fund_parallelogram"

show ?thesis

```

proof (rule that[of "h ∘ to_fund_parallelogram"])
  show "bij_betw (h ∘ to_fund_parallelogram)
        (period_parallelogram orig) (period_parallelogram orig'"
    unfolding h_def
    using bij_betw_to_fund_parallelogram bij_betw_inv_into[OF bij_betw_to_fund_parallelogram]
    by (rule bij_betw_trans)
next
  fix z :: complex
  have "rel (to_fund_parallelogram (h (to_fund_parallelogram z))) (h
(to_fund_parallelogram z))"
    by auto
  also have "to_fund_parallelogram (h (to_fund_parallelogram z)) = to_fund_parallelogram
z"
    unfolding h_def using bij_betw_to_fund_parallelogram[of orig']
    by (subst f_inv_into_f[of _ to_fund_parallelogram])
      (simp_all add: bij_betw_def to_fund_parallelogram_in_parallelogram)
  finally have *: "rel (to_fund_parallelogram z) (h (to_fund_parallelogram
z))" .
  have "rel ((to_fund_parallelogram z - z)) (to_fund_parallelogram z
- h (to_fund_parallelogram z))"
    using * diff_in_lattice rel_def rel_to_fund_parallelogram_left by
blast
  thus "rel ((h ∘ to_fund_parallelogram) z) z"
    using * pre_complex_lattice.rel_sym by force
qed
qed

lemma to_fund_parallelogram_0 [simp]: "to_fund_parallelogram 0 = 0"
  by (simp add: to_fund_parallelogram_def zero_prod_def)

lemma to_fund_parallelogram_lattice [simp]: "z ∈ Λ ⇒ to_fund_parallelogram
z = 0"
  by (auto simp: to_fund_parallelogram_def in_lattice_conv_ω12_coords)

lemma to_fund_parallelogram_eq_iff [simp]:
  "to_fund_parallelogram u = to_fund_parallelogram v ↔ rel u v"
proof
  assume "rel u v"
  then obtain z where z: "z ∈ Λ" "v = u + z"
    by (elim relE)
  from this(1) obtain m n where mn: "z = of_ω12_coords (of_int m, of_int
n)"
    by (elim latticeE)
  show "to_fund_parallelogram u = to_fund_parallelogram v" unfolding
z(2)
    by (simp add: to_fund_parallelogram_def ω12_coords.add in_lattice_conv_ω12_coords
mn case_prod_unfold)
next
  assume "to_fund_parallelogram u = to_fund_parallelogram v"

```

```

    thus "rel u v"
      by (metis rel_to_fund_parallelogram_right rel_to_fund_parallelogram_right_iff)
qed

```

```

lemma to_fund_parallelogram_eq_0_iff [simp]: "to_fund_parallelogram u
= 0  $\longleftrightarrow$  u  $\in$   $\Lambda$ "
  using to_fund_parallelogram_eq_iff[of u 0]
  by (simp del: to_fund_parallelogram_eq_iff add: rel_0_right_iff)

```

```

lemma to_fund_parallelogram_of_fund_parallelogram:
  "z  $\in$  period_parallelogram 0  $\implies$  to_fund_parallelogram z = z"
  unfolding to_fund_parallelogram_def period_parallelogram_def
  by (auto simp: of_omega12_coords_eq_iff frac_eq_id)

```

```

lemma to_fund_parallelogram_idemp [simp]:
  "to_fund_parallelogram (to_fund_parallelogram z) = to_fund_parallelogram
z"
  by (rule to_fund_parallelogram_of_fund_parallelogram) auto

```

```

lemma to_fund_parallelogram_unique:
  assumes "rel z z'" "z'  $\in$  period_parallelogram 0"
  shows "to_fund_parallelogram z = z'"
  using assms by (metis to_fund_parallelogram_eq_iff to_fund_parallelogram_of_fund_parallelogram)

```

```

lemma to_fund_parallelogram_unique':
  assumes "rel z z'" "z  $\in$  period_parallelogram 0" "z'  $\in$  period_parallelogram
0"
  shows "z = z'"
  using assms
  by (metis to_fund_parallelogram_eq_iff to_fund_parallelogram_of_fund_parallelogram)

```

The following is the “left half” of the fundamental parallelogram. The bottom border is contained, the top border is not. Of the frontier of this parallelogram only the upper half is

```

definition (in pre_complex_lattice) half_fund_parallelogram where
  "half_fund_parallelogram =
    of_omega12_coords ' {(x,y). x  $\in$  {0..1/2}  $\wedge$  y  $\in$  {0.. $<1$ }  $\wedge$  (x  $\in$  {0, 1/2}
 $\longrightarrow$  y  $\leq$  1/2)}"

```

```

lemma half_fund_parallelogram_altdef:
  "half_fund_parallelogram = omega12_coords - ' {(x,y). x  $\in$  {0..1/2}  $\wedge$  y  $\in$ 
{0.. $<1$ }  $\wedge$  (x  $\in$  {0, 1/2}  $\longrightarrow$  y  $\leq$  1/2)}"
  unfolding half_fund_parallelogram_def by (meson of_omega12_coords_image_eq)

```

```

lemma zero_in_half_fund_parallelogram [simp, intro]: "0  $\in$  half_fund_parallelogram"
  by (auto simp: half_fund_parallelogram_altdef zero_prod_def)

```

```

lemma half_fund_parallelogram_in_lattice_iff:
  assumes "z  $\in$  half_fund_parallelogram"

```

```

shows "z ∈ Λ ↔ z = 0"
proof
  assume "z ∈ Λ"
  then obtain m n where z_eq: "z = of_ω12_coords (of_int m, of_int n)"
    by (elim latticeE)
  thus "z = 0"
    using assms unfolding z_eq half_fund_parallelogram_altdef by auto
qed auto

definition to_half_fund_parallelogram :: "complex ⇒ complex" where
  "to_half_fund_parallelogram z =
    (let (x,y) = map_prod frac frac (ω12_coords z);
      (x',y') = (if x > 1/2 ∨ (x ∈ {0, 1/2} ∧ y > 1 / 2) then (if
x = 0 then 0 else 1 - x, if y = 0 then 0 else 1 - y) else (x, y))
      in of_ω12_coords (x',y'))"

lemma in_Ints_conv_floor: "x ∈ ℤ ↔ x = of_int (floor x)"
  by (metis Ints_of_int of_int_floor)

lemma (in complex_lattice) rel_to_half_fund_parallelogram:
  "rel z (to_half_fund_parallelogram z) ∨ rel z (-to_half_fund_parallelogram
z)"
  unfolding rel_def in_lattice_conv_ω12_coords to_half_fund_parallelogram_def
  Let_def
  ω1_coord.diff ω2_coord.diff ω1_coord.add ω2_coord.add ω1_coord.neg
ω2_coord.neg
  case_prod_unfold Let_def ω12_coords_def frac_def map_prod_def
  by (simp flip: in_Ints_conv_floor)

lemma (in complex_lattice) to_half_fund_parallelogram_in_half_fund_parallelogram
[intro]:
  "to_half_fund_parallelogram z ∈ half_fund_parallelogram"
  unfolding half_fund_parallelogram_altdef to_half_fund_parallelogram_def
to_half_fund_parallelogram_def Let_def
  ω1_coord.diff ω2_coord.diff ω1_coord.add ω2_coord.add ω1_coord.neg
ω2_coord.neg
  case_prod_unfold Let_def ω12_coords_def frac_def map_prod_def
  apply simp
  apply linarith
  done

lemma (in complex_lattice) half_fund_parallelogram_subset_period_parallelogram:
  "half_fund_parallelogram ⊆ period_parallelogram 0"
proof -
  have "of_ω12_coords ' {(x, y). x ∈ {0..1 / 2} ∧ y ∈ {0..<1} ∧ (x ∈
{0,1/2} → y ≤ 1 / 2)} ⊆
  of_ω12_coords ' ({0..<1} × {0..<1})"
    by (intro image_mono) (auto simp: not_less)
  also have "... = (+) 0 ' (of_ω12_coords ' ({0..<1} × {0..<1}))"

```

```

    by simp
  finally show ?thesis
    unfolding period_parallelogram_def half_fund_parallelogram_def .
qed

lemma to_half_fund_parallelogram_in_lattice_iff [simp]: "to_half_fund_parallelogram
z ∈  $\Lambda$   $\leftrightarrow$  z ∈  $\Lambda$ "
  by (metis rel_lattice_trans_left rel_sym rel_to_half_fund_parallelogram
uminus_in_lattice_iff)

lemma rel_in_half_fund_parallelogram_imp_eq:
  assumes "rel z w  $\vee$  rel z (-w)" "z ∈ half_fund_parallelogram" "w ∈
half_fund_parallelogram"
  shows "z = w"
  using assms(1)
proof
  assume "rel z w"
  moreover from assms have "z ∈ period_parallelogram 0" "w ∈ period_parallelogram
0"
  using half_fund_parallelogram_subset_period_parallelogram by blast+
  ultimately show "z = w"
  by (metis to_fund_parallelogram_eq_iff
to_fund_parallelogram_of_fund_parallelogram)
next
  assume "rel z (-w)"
  hence "rel (-w) z"
  by (rule rel_symI)
  then obtain m n where z_eq: "z = -w + of_ $\omega$ 12_coords (of_int m, of_int
n)"
  by (elim relE latticeE) auto
  define x y where "x =  $\omega$ 1_coord w" "y =  $\omega$ 2_coord w"
  have w_eq: "w = of_ $\omega$ 12_coords (x, y)"
  unfolding x_y_def  $\omega$ 12_coords_def [symmetric] by simp
  have 1: "x  $\geq$  0" "y  $\geq$  0" "x  $\leq$  1/2" "y < 1" "x = 0  $\vee$  x = 1/2  $\implies$  y
 $\leq$  1/2"
  using assms(3) unfolding half_fund_parallelogram_altdef w_eq by auto
  have 2: "of_int m - x  $\geq$  0" "of_int n - y  $\geq$  0" "of_int m - x  $\leq$  1/2"
"of_int n - y < 1"
  "of_int m - x = 0  $\vee$  of_int m - x = 1/2  $\implies$  of_int n - y  $\leq$  1/2"
  using assms(2) unfolding half_fund_parallelogram_altdef z_eq w_eq

  by (auto simp:  $\omega$ 12_coords.diff)

  have "m ∈ {0,1}" "n ∈ {0,1}"
  using 1 2 by auto
  hence "real_of_int m = 2 * x  $\wedge$  real_of_int n = 2 * y"
  using 1 2 by auto
  hence " $\omega$ 12_coords z =  $\omega$ 12_coords w"
  unfolding z_eq w_eq  $\omega$ 12_coords.add  $\omega$ 12_coords.diff  $\omega$ 12_coords.neg

```

```

ω12_coords_of_ω12_coords
  by simp
  thus ?thesis
    by (metis of_ω12_coords_ω12_coords)
qed

lemma to_half_fund_parallelogram_of_half_fund_parallelogram:
  assumes "z ∈ half_fund_parallelogram"
  shows "to_half_fund_parallelogram z = z"
  by (metis assms rel_to_half_fund_parallelogram to_half_fund_parallelogram_in_half_fund_pa
rel_in_half_fund_parallelogram_imp_eq)

lemma to_half_fund_parallelogram_idemp [simp]:
  "to_half_fund_parallelogram (to_half_fund_parallelogram z) = to_half_fund_parallelogram
z"
  by (rule to_half_fund_parallelogram_of_half_fund_parallelogram) auto

lemma to_half_fund_parallelogram_unique:
  assumes "rel z z' ∨ rel z (-z'" "z' ∈ half_fund_parallelogram"
  shows "to_half_fund_parallelogram z = z'"
proof (rule rel_in_half_fund_parallelogram_imp_eq)
  show "rel (to_half_fund_parallelogram z) z' ∨ rel (to_half_fund_parallelogram
z) (- z'"
    using rel_to_half_fund_parallelogram[of z] assms rel_sym rel_trans
rel_minus minus_minus
    by metis
qed (use to_half_fund_parallelogram_in_half_fund_parallelogram assms in
auto)

lemma to_half_fund_parallelogram_eq_iff:
  "to_half_fund_parallelogram z = to_half_fund_parallelogram w  $\longleftrightarrow$  rel
z w ∨ rel z (-w)"
proof
  assume eq: "to_half_fund_parallelogram z = to_half_fund_parallelogram
w"
  define u where "u = to_half_fund_parallelogram w"
  have "rel z u ∨ rel z (-u)" "rel w u ∨ rel w (-u)"
    using rel_to_half_fund_parallelogram[of z] rel_to_half_fund_parallelogram[of
w] eq unfolding u_def by auto
  hence "rel z w ∨ (rel z u ∧ rel w (-u) ∨ rel z (-u) ∧ rel w u)"
    using rel_trans rel_sym by blast
  moreover have "rel z (-w)" if "rel z u ∧ rel w (-u) ∨ rel z (-u) ∧
rel w u"
    using that by (metis minus_minus pre_complex_lattice.rel_minus rel_sym
rel_trans)
  ultimately show "rel z w ∨ rel z (-w)" by blast
next
  assume *: "rel z w ∨ rel z (-w)"
  define u where "u = to_half_fund_parallelogram w"

```

```

show "to_half_fund_parallelogram z = u"
proof (rule to_half_fund_parallelogram_unique)
  have "rel w u  $\vee$  rel w (-u)"
    unfolding u_def using rel_to_half_fund_parallelogram[of w] by blast
  with * have "rel z u  $\vee$  (rel (-w) (-u)  $\wedge$  rel z (-w)  $\vee$  rel w (-u)
 $\wedge$  rel z w)"
    using rel_trans rel_sym rel_minus[of w "-u"] rel_minus[of z "-w"]
rel_minus[of w u]
    unfolding minus_minus by blast
  thus "rel z u  $\vee$  rel z (-u)"
    using rel_trans rel_sym by blast
qed (auto simp: u_def)
qed

lemma in_half_fund_parallelogram_imp_half_lattice:
  assumes "z  $\in$  half_fund_parallelogram" "to_fund_parallelogram (-z)  $\in$ 
half_fund_parallelogram"
  shows "2 * z  $\in$   $\Lambda$ "
  using assms
  by (metis rel_in_half_fund_parallelogram_imp_eq diff_minus_eq_add mult_2
pre_complex_lattice.rel_def rel_to_fund_parallelogram_left)

end

```

3.4 Equivalence of fundamental pairs

Two fundamental pairs are called *equivalent* if they generate the same complex lattice.

definition `equiv_fundpair` :: "complex \times complex \Rightarrow complex \times complex \Rightarrow bool" where

```

"equiv_fundpair = ( $\lambda$ ( $\omega$ 1,  $\omega$ 2) ( $\omega$ 1',  $\omega$ 2')) .
pre_complex_lattice.lattice  $\omega$ 1  $\omega$ 2 = pre_complex_lattice.lattice
 $\omega$ 1'  $\omega$ 2'"

```

lemma `equiv_fundpair_iff_aux`:

```

fixes p :: int
assumes "p * c + q * a = 0" "p * d + q * b = 1"
        "r * c + s * a = 1" "r * d + s * b = 0"
shows "|a * d - b * c| = 1"
proof -
  have "r * b * c + s * a * b = b"
    by (metis assms(3) distrib_left mult.left_commute mult.right_neutral)
  moreover have "r * a * d + s * a * b = 0"
    by (metis assms(4) distrib_left mult.commute mult.left_commute mult_zero_right)
  ultimately have "r dvd b"
    by (metis mult.assoc dvd_0_right dvd_add_right_iff dvd_triv_left)
  have "p * r * d + p * s * b = 0"
    by (metis assms(4) distrib_left mult.commute mult.left_commute mult_zero_right)
  moreover have "p * r * d + q * r * b = r"

```

```

    by (metis assms(2) int_distrib(2) mult.assoc mult.left_commute mult.right_neutral)
ultimately have "b dvd r"
    by (metis dvd_0_right mult.commute zdvd_reduce)
have "r * c * d + s * b * c = 0"
    by (metis assms(4) distrib_left mult.commute mult.left_commute mult_zero_right)
moreover have "r * c * d + s * a * d = d"
    by (metis assms(3) distrib_left mult.commute mult.right_neutral)
ultimately have "s dvd d"
    by (metis dvd_0_right dvd_add_times_triv_right_iff mult.assoc mult.commute)
have "q * r * d + q * s * b = 0"
    by (metis mult.assoc assms(4) int_distrib(2) mult_not_zero)
moreover have "p * s * d + q * s * b = s"
    by (metis assms(2) int_distrib(2) mult.assoc mult.left_commute mult.right_neutral)
ultimately have "d dvd s"
    by (metis add.commute dvd_0_right mult.commute zdvd_reduce)
have "|b| = |r|" "|d| = |s|"
    by (meson <b dvd r> <r dvd b> <d dvd s> <s dvd d> zdvd_antisym_abs)+
then show ?thesis
    by (smt (verit, best) assms(3,4) mult.commute mult_cancel_left mult_eq_0_iff
mult_minus_left)
qed

```

The following fact is Theorem 1.2 in Apostol's book: two fundamental pairs are equivalent iff there exists a unimodular transformation that maps one to the other.

theorem equiv_fundpair_iff:

```

    fixes  $\omega_1 \omega_2 \omega_1' \omega_2' :: \text{complex}$ 
    assumes "fundpair ( $\omega_1, \omega_2$ )" "fundpair ( $\omega_1', \omega_2'$ )"
    shows "equiv_fundpair ( $\omega_1, \omega_2$ ) ( $\omega_1', \omega_2'$ )  $\longleftrightarrow$ 
        ( $\exists a b c d. |a*d - b*c| = 1 \wedge$ 
             $\omega_2' = \text{of\_int } a * \omega_2 + \text{of\_int } b * \omega_1 \wedge \omega_1' = \text{of\_int } c * \omega_2 + \text{of\_int } d * \omega_1$ )"
    (is "?lhs = ?rhs")

```

proof -

```

    interpret gl: complex_lattice  $\omega_1 \omega_2$ 
    by standard fact
    interpret gl': complex_lattice  $\omega_1' \omega_2'$ 
    by standard fact
    show ?thesis
    proof
        assume L: ?lhs
        hence lattices_eq: "gl.lattice = gl'.lattice"
            by (simp add: equiv_fundpair_def)

        have " $\omega_1' \in \text{gl'.lattice}$ " " $\omega_2' \in \text{gl'.lattice}$ "
            by auto
        hence " $\omega_1' \in \text{gl.lattice}$ " " $\omega_2' \in \text{gl.lattice}$ "
            by (simp_all add: lattices_eq)
        then obtain a b c d

```

```

where ab: "ω2' = of_int b * ω1 + of_int a * ω2"
      and cd: "ω1' = of_int d * ω1 + of_int c * ω2"
by (elim gl.latticeE) (auto simp: gl.of_ω12_coords_def scaleR_conv_of_real)

have "ω1 ∈ gl.lattice" "ω2 ∈ gl.lattice"
  by auto
hence "ω1 ∈ gl'.lattice" "ω2 ∈ gl'.lattice"
  by (simp_all add: lattices_eq)
then obtain p q r s
  where pq: "ω1 = of_int p * ω1' + of_int q * ω2'" and
        rs: "ω2 = of_int r * ω1' + of_int s * ω2'"
  by (elim gl'.latticeE) (auto simp: gl'.of_ω12_coords_def scaleR_conv_of_real)

have "ω1 = p * (c * ω2 + d * ω1) + q * (a * ω2 + b * ω1)"
  using pq cd ab add.commute by metis
also have "... = (p * c + q * a) * ω2 + (p * d + q * b) * ω1"
  by (simp add: algebra_simps)
finally have "gl.of_ω12_coords (1, 0) = gl.of_ω12_coords (p*d+q*b,
p*c+q*a)"
  by (simp_all add: gl.of_ω12_coords_def add_ac)
hence pc: "(p * c + q * a) = 0 ∧ p * d + q * b = 1"
  unfolding gl.of_ω12_coords_eq_iff prod.case prod.inject by linarith

have "ω2 = r * (c * ω2 + d * ω1) + s * (a * ω2 + b * ω1)"
  using cd rs ab add.commute by metis
also have "... = (r * c + s * a) * ω2 + (r * d + s * b) * ω1"
  by (simp add: algebra_simps)
finally have "gl.of_ω12_coords (0, 1) = gl.of_ω12_coords (r*d+s*b,
r*c+s*a)"
  by (simp_all add: gl.of_ω12_coords_def add_ac)
hence rc: "r * c + s * a = 1 ∧ r * d + s * b = 0"
  unfolding gl.of_ω12_coords_eq_iff prod.case prod.inject by linarith
with pc have "|a*d - b*c| = 1"
  by (meson equiv_fundpair_iff_aux)

hence "|a*d - b*c| = 1 ∧
      ω2' = of_int a * ω2 + of_int b * ω1 ∧ ω1' = of_int c * ω2
+ of_int d * ω1"
  using cd ab by auto
thus ?rhs
  by blast
next
assume ?rhs
then obtain a b c d :: int where 1: "|a * d - b * c| = 1"
  and eq: "ω2' = of_int a * ω2 + of_int b * ω1" "ω1' = of_int c
* ω2 + of_int d * ω1"
  by blast
define det where "det = a * d - b * c"
define a' b' c' d' where "a' = det * d" "b' = -det * b" "c' = -det

```

```

* c" "d' = det * a"
  have "|det| = 1"
    using 1 by (simp add: det_def)
  hence det_square: "det ^ 2 = 1"
    using abs_square_eq_1 by blast

  have eq': "ω2 = of_int a' * ω2' + of_int b' * ω1'" "ω1 = of_int
c' * ω2' + of_int d' * ω1'"
  proof -
    have "of_int a' * ω2' + of_int b' * ω1' = det^2 * ω2"
      by (simp add: eq algebra_simps det_def a'_b'_c'_d'_def power2_eq_square)
    thus "ω2 = of_int a' * ω2' + of_int b' * ω1'"
      by (simp add: det_square)
  next
    have "of_int c' * ω2' + of_int d' * ω1' = det^2 * ω1"
      by (simp add: eq algebra_simps det_def a'_b'_c'_d'_def power2_eq_square)
    thus "ω1 = of_int c' * ω2' + of_int d' * ω1'"
      by (simp add: det_square)
  qed

  have "gl'.lattice ⊆ gl.lattice"
    by (safe elim!: gl'.latticeE, unfold gl'.of_ω12_coords_def)
      (auto simp: eq ring_distrib intro!: gl'.lattice_intros)
  moreover have "gl.lattice ⊆ gl'.lattice"
    by (safe elim!: gl.latticeE, unfold gl.of_ω12_coords_def)
      (auto simp: eq' ring_distrib intro!: gl.lattice_intros)
  ultimately show ?lhs
    unfolding equiv_fundpair_def by auto
  qed
qed

```

We will now look at the triangle spanned by the origin and the generators. We will prove that the only points that lie in or on this triangle are its three vertices.

Moreover, we shall prove that for any lattice Λ , if we have two points $\omega_1', \omega_2' \in \Lambda$ then these two points generate Λ if and only if the triangle spanned by $0, \omega_1'$, and ω_2' contains no other lattice points except $0, \omega_1'$, and ω_2' .

```

context complex_lattice
begin

```

```

lemma in_triangle_iff:

```

```

  fixes x
  defines "a ≡ ω1_coord x" and "b ≡ ω2_coord x"
  shows "x ∈ convex_hull {0, ω1, ω2} ↔ a ≥ 0 ∧ b ≥ 0 ∧ a + b ≤ 1"

```

```

proof

```

```

  assume "x ∈ convex_hull {0, ω1, ω2}"
  then obtain t u where tu: "u ≥ 0" "t ≥ 0" "u + t ≤ 1" "x = of_ω12_coords

```

```

(u, t)"
  unfolding convex_hull_3_alt by (auto simp: of_ω12_coords_def scaleR_conv_of_real)
  have "a = u" "b = t"
  by (auto simp: a_def b_def tu(4))
  with tu show "a ≥ 0 ∧ b ≥ 0 ∧ a + b ≤ 1"
  by auto
next
  assume ab: "a ≥ 0 ∧ b ≥ 0 ∧ a + b ≤ 1"
  have "x = of_ω12_coords (a, b)"
  by (auto simp: a_def b_def simp flip: ω12_coords_def)
  hence "x = 0 + a *R (ω1 - 0) + b *R (ω2 - 0)" "0 ≤ a ∧ 0 ≤ b ∧ a
+ b ≤ 1"
  using ab by (auto simp: a_def b_def of_ω12_coords_def scaleR_conv_of_real)
  thus "x ∈ convex hull {0, ω1, ω2}"
  unfolding convex_hull_3_alt by blast
qed

```

The only lattice points inside the fundamental triangle are the generators and the origin.

lemma *lattice_Int_triangle*: "convex hull {0, ω1, ω2} ∩ Λ = {0, ω1, ω2}"

proof (intro equalityI subsetI)

```

  fix x assume x: "x ∈ convex hull {0, ω1, ω2} ∩ Λ"
  then obtain a b :: real where ab: "a ≥ 0" "b ≥ 0" "a + b ≤ 1" "x
= of_ω12_coords (a, b)"
  unfolding convex_hull_3_alt by (auto simp: of_ω12_coords_def scaleR_conv_of_real)
  from ab(4) and x have "a ∈ ℤ" "b ∈ ℤ"
  by (auto simp: of_ω12_coords_in_lattice_iff)
  with ab(1-3) have "(a, b) ∈ {(0, 0), (0, 1), (1, 0)}"
  by (auto elim!: Ints_cases)
  with ab show "x ∈ {0, ω1, ω2}"
  by auto
qed (auto intro: hull_inc)

```

The following fact is Theorem 1.1 in Apostol's book: given a fixed lattice Λ , a pair of non-collinear period vectors ω_1, ω_2 is fundamental (i.e. generates Λ) iff the triangle spanned by $0, \omega_1, \omega_2$ contains no lattice points other than its three vertices.

lemma *equiv_fundpair_iff_triangle*:

```

  assumes "fundpair (ω1', ω2'" "ω1' ∈ Λ" "ω2' ∈ Λ"
  shows "equiv_fundpair (ω1, ω2) (ω1', ω2') ↔ convex hull {0, ω1',
ω2'} ∩ Λ = {0, ω1', ω2'}"

```

proof -

```

  interpret lattice': complex_lattice ω1' ω2'

```

```

  by standard fact

```

```

  show ?thesis

```

proof

```

  assume "equiv_fundpair (ω1, ω2) (ω1', ω2'"

```

```

  thus "convex hull {0, ω1', ω2'} ∩ Λ = {0, ω1', ω2'}"

```

```

    using lattice'.lattice_Int_triangle by (simp add: equiv_fundpair_def)
next
  assume triangle: "convex hull {0,  $\omega_1'$ ,  $\omega_2'$ }  $\cap$   $\Lambda$  = {0,  $\omega_1'$ ,  $\omega_2'$ }"

  show "equiv_fundpair ( $\omega_1$ ,  $\omega_2$ ) ( $\omega_1'$ ,  $\omega_2'$ )"
    unfolding equiv_fundpair_def prod.case
  proof
    show "lattice'.lattice  $\subseteq$   $\Lambda$ "
      by (intro subsetI, elim lattice'.latticeE)
        (auto simp: lattice'.of_ $\omega_1_2$ _coords_def intro!: lattice_intros
  assms)
    next
      show " $\Lambda \subseteq$  lattice'.lattice"
        proof
          fix x assume x: "x  $\in$   $\Lambda$ "
          define y where "y = lattice'.to_fund_parallelogram x"
          have y: "y  $\in$   $\Lambda$ "
          proof -
            obtain a b where y_eq: "y = x + of_int a *  $\omega_1'$  + of_int b *
 $\omega_2'$ "
              using lattice'.to_fund_parallelogramE[of x] unfolding y_def
            by blast
            show ?thesis by (auto simp: y_eq intro!: lattice_intros assms
  x)
          qed

          have "y  $\in$  lattice'.lattice"
          proof (cases "y  $\in$  convex hull {0,  $\omega_1'$ ,  $\omega_2'$ }")
            case True
              hence "y  $\in$  convex hull {0,  $\omega_1'$ ,  $\omega_2'$ }  $\cap$   $\Lambda$ "
                using y by auto
              also note triangle
              finally show ?thesis
                by auto
            next
              case False
                define y' where "y' =  $\omega_1'$  +  $\omega_2'$  - y"
                have y_conv_y': "y =  $\omega_1'$  +  $\omega_2'$  - y'"
                  by (simp add: y'_def)
                define a b where "a = lattice'. $\omega_1$ _coord x" and "b = lattice'. $\omega_2$ _coord
  x"
                  have [simp]: "lattice'. $\omega_1$ _coord y = frac a" "lattice'. $\omega_2$ _coord
  y = frac b"
                    by (simp_all add: y_def a_def b_def)
                  from False have "frac a + frac b > 1"
                    by (auto simp: lattice'.in_triangle_iff y'_def)
                  hence "y'  $\in$  convex hull {0,  $\omega_1'$ ,  $\omega_2'$ }"
                    by (auto simp: lattice'.in_triangle_iff y'_def less_imp_le[OF
  frac_lt_1]

```

```

        lattice'. $\omega_1$ _coord.add lattice'. $\omega_2$ _coord.add
        lattice'. $\omega_1$ _coord.diff lattice'. $\omega_2$ _coord.diff)
    hence "y'  $\in$  convex hull {0,  $\omega_1'$ ,  $\omega_2'$ }  $\cap$   $\Lambda$ "
      using y assms by (auto simp: y'_def intro!: lattice_intros)
    also note triangle
    finally have "y'  $\in$  {0,  $\omega_1'$ ,  $\omega_2'$ }" .
    thus ?thesis
      by (auto simp: y_conv_y' intro!: lattice'.lattice_intros)
  qed
  thus "x  $\in$  lattice'.lattice"
    by (simp add: y_def)
  qed
  qed
  qed
  qed
end

```

3.5 Additional useful facts

```

context complex_lattice
begin

```

The following partitions the lattice into countably many “layers”, starting from the origin, which is the 0-th layer. The k -th layer consists of precisely those points in the lattice whose lattice coordinates (m, n) satisfy $\max(|m|, |n|) = k$.

```

definition lattice_layer :: "nat  $\Rightarrow$  complex set" where
  "lattice_layer k =
    of_ $\omega_{12}$ _coords ' map_prod of_int of_int '
    ({int k, -int k}  $\times$  {-int k.. $\text{int } k$ }  $\cup$  {-int k.. $\text{int } k$ }  $\times$  {-int k,
int k})"

```

```

lemma in_lattice_layer_iff:
  "z  $\in$  lattice_layer k  $\iff$ 
     $\omega_{12}$ _coords z  $\in$   $\mathbb{Z} \times \mathbb{Z} \cap$  ({int k, -int k}  $\times$  {-int k.. $\text{int } k$ }  $\cup$  {-int
k.. $\text{int } k$ }  $\times$  {-int k, int k})"
  (is "?lhs = ?rhs")

```

```

proof
  assume ?lhs
  thus ?rhs
    unfolding lattice_layer_def of_ $\omega_{12}$ _coords_image_eq by (auto simp:
case_prod_unfold)
next
  assume ?rhs
  thus ?lhs unfolding lattice_layer_def image_Un map_prod_image of_ $\omega_{12}$ _coords_image_eq
    by (auto elim!: Ints_cases)
qed

```

```

lemma of_ω12_coords_of_int_in_lattice_layer:
  "of_ω12_coords (of_int a, of_int b) ∈ lattice_layer (nat (max |a| |b|))"
  unfolding in_lattice_layer_iff by (auto simp flip: of_int_minus simp:
max_def)

```

```

lemma lattice_layer_covers: "Λ = (⋃k. lattice_layer k)"
proof -
  have "(⋃k. lattice_layer k) = of_ω12_coords ' map_prod real_of_int
real_of_int '
      (⋃k. ({int k, - int k} × {- int k..int k} ∪ {- int k..int
k} × {- int k, int k}))"
    (is "_ = _ ' _ ' (⋃k. ?A k)") unfolding lattice_layer_def by blast
  also have "(⋃k. ?A k) = UNIV"
  proof safe
    fix a b :: int
    have "(a, b) ∈ ?A (nat (max |a| |b|))"
      unfolding max_def by simp linarith
    thus "(a, b) ∈ (⋃k. ?A k)"
      by blast
  qed blast+
  also have "range (map_prod real_of_int real_of_int) = ℤ × ℤ"
    by (auto elim!: Ints_cases)
  finally show ?thesis
    by (simp add: lattice_def)
qed

```

```

lemma finite_lattice_layer: "finite (lattice_layer k)"
  unfolding lattice_layer_def by auto

```

```

lemma lattice_layer_0: "lattice_layer 0 = {0}"
  by (auto simp: lattice_layer_def)

```

```

lemma zero_in_lattice_layer_iff [simp]: "0 ∈ lattice_layer k ⟷ k =
0"
  by (auto simp: in_lattice_layer_iff zero_prod_def)

```

```

lemma lattice_layer_disjoint:
  assumes "m ≠ n"
  shows "lattice_layer m ∩ lattice_layer n = {}"
  using assms by (auto simp: lattice_layer_def of_ω12_coords_eq_iff)

```

```

lemma lattice0_conv_layers: "Λ* = (⋃i∈{0<..}. lattice_layer i)" (is
"?lhs = ?rhs")

```

```

proof -
  have "Λ* = (⋃i∈UNIV. lattice_layer i) - lattice_layer 0"
    by (simp add: lattice0_def lattice_layer_covers lattice_layer_0)
  also have "... = (⋃i∈UNIV-{0}. lattice_layer i)"
    using lattice_layer_disjoint by blast

```

```

    also have "UNIV-{0:nat} = {0<..}"
      by auto
    finally show ?thesis .
qed

lemma card_lattice_layer:
  assumes "k > 0"
  shows "card (lattice_layer k) = 8 * k"
proof -
  define f where "f = of_ω12_coords ∘ map_prod real_of_int real_of_int"
  have "lattice_layer k = f ` ({int k, -int k} × {-int k..int k} ∪
{-int k..int k} × {-int k, int k})"
    (is "_ = _ ' ?A") unfolding lattice_layer_def f_def image_image o_def
  ..
  also have "card ... = card ?A"
    by (intro card_image)
      (auto simp: inj_on_def of_ω12_coords_eq_iff f_def map_prod_def
case_prod_unfold)
  also have "?A = {int k, -int k} × {-int k..int k} ∪ {-int k+1..int
k-1} × {-int k, int k}"
    by auto
  also have "card ... = 8 * k" using <k > 0>
    by (subst card_Un_disjoint)
      (auto simp: nat_diff_distrib nat_add_distrib nat_mult_distrib Suc_diff_Suc)
  finally show ?thesis .
qed

lemma lattice_layer_nonempty: "lattice_layer k ≠ {}"
  by (auto simp: lattice_layer_def)

definition lattice_layer_path :: "complex set" where
  "lattice_layer_path = of_ω12_coords ` ({1, -1} × {-1..1} ∪ {-1..1}
× {-1, 1})"

lemma in_lattice_layer_path_iff:
  "z ∈ lattice_layer_path ↔ ω12_coords z ∈ ({1, -1} × {-1..1} ∪ {-1..1}
× {-1, 1})"
  unfolding lattice_layer_path_def of_ω12_coords_image_eq by blast

lemma lattice_layer_path_nonempty: "lattice_layer_path ≠ {}"
proof -
  have "ω1 ∈ lattice_layer_path"
    by (auto simp: in_lattice_layer_path_iff)
  thus ?thesis by blast
qed

lemma compact_lattice_layer_path [intro]: "compact lattice_layer_path"
  unfolding lattice_layer_path_def of_ω12_coords_def case_prod_unfold
  by (intro compact_continuous_image continuous_intros compact_Un compact_Times)

```

auto

```

lemma lattice_layer_subset: "lattice_layer k  $\subseteq$  (*) (of_nat k) ' lattice_layer_path"
proof
  fix x
  assume "x  $\in$  lattice_layer k"
  then obtain m n where x: "x = of_omega12_coords (of_int m, of_int n)"
    "(m, n)  $\in$  ({int k, -int k}  $\times$  {-int k..int k}  $\cup$  {-int k..int k}  $\times$ 
  {-int k, int k})"
    unfolding lattice_layer_def by blast

  show "x  $\in$  (*) (of_nat k) ' lattice_layer_path"
  proof (cases "k > 0")
    case True
    have "x = of_nat k * of_omega12_coords (of_int m / of_int k, of_int n
  / of_int k)"
      "(of_int m / of_int k, of_int n / of_int k)  $\in$  {1::real, -1}  $\times$ 
  {-1..1}  $\cup$  {-1..1}  $\times$  {-1::real, 1}"
      using x True by (auto simp: divide_simps of_omega12_coords_def)
    thus ?thesis
      unfolding lattice_layer_path_def by blast
  qed (use x in <auto simp: lattice_layer_path_def image_iff
    intro!: exI[of _ "omega1"] bexI[of _ "(1, 0)"]>)
qed

```

The shortest and longest distance of any point on the first layer from the origin, respectively.

definition *Inf_para* :: real where — *r* in the proof of Lemma 1
"Inf_para \equiv Inf (norm ' lattice_layer_path)"

```

lemma Inf_para_pos: "Inf_para > 0"
proof -
  have "compact (norm ' lattice_layer_path)"
    by (intro compact_continuous_image continuous_intros) auto
  hence "Inf_para  $\in$  norm ' lattice_layer_path"
    unfolding Inf_para_def
    by (intro closed_contains_Inf)
      (use lattice_layer_path_nonempty in <auto simp: compact_imp_closed
  bdd_below_norm_image>)
  moreover have " $\forall x \in$  norm ' lattice_layer_path. x > 0"
    by (auto simp: in_lattice_layer_path_iff zero_prod_def)
  ultimately show ?thesis
    by blast
qed

```

lemma *Inf_para_nonzero* [*simp*]: "*Inf_para* \neq 0"
 using *Inf_para_pos* by *linarith*

lemma *Inf_para_le*:

```

    assumes "z ∈ lattice_layer_path"
    shows "Inf_para ≤ norm z"
    unfolding Inf_para_def by (rule cInf_lower) (use assms bdd_below_norm_image
in auto)

```

```

lemma lattice_layer_le_norm:
  assumes "ω ∈ lattice_layer k"
  shows "k * Inf_para ≤ norm ω"
proof -
  obtain z where z: "z ∈ lattice_layer_path" "ω = of_nat k * z"
    using lattice_layer_subset[of k] assms by auto
  have "real k * Inf_para ≤ real k * norm z"
    by (intro mult_left_mono Inf_para_le z) auto
  also have "... = norm ω"
    by (simp add: z norm_mult)
  finally show ?thesis .
qed

```

```

corollary Inf_para_le_norm:
  assumes "ω ∈ Λ*"
  shows "Inf_para ≤ norm ω"
proof -
  from assms obtain k where ω: "ω ∈ lattice_layer k" and "k ≠ 0"
    unfolding lattice0_def by (metis DiffE UN_iff lattice_layer_0 lattice_layer_covers)
  with Inf_para_pos have "Inf_para ≤ real k * Inf_para"
    by auto
  then show ?thesis
    using ω lattice_layer_le_norm by force
qed

```

One easy corollary is now that our lattice is discrete in the sense that there is a positive real number that bounds the distance between any two points from below.

```

lemma Inf_para_le_dist:
  assumes "x ∈ Λ" "y ∈ Λ" "x ≠ y"
  shows "dist x y ≥ Inf_para"
proof -
  have "x - y ∈ Λ" "x - y ≠ 0"
    using assms by (auto intro: diff_in_lattice)
  hence "x - y ∈ Λ*"
    by auto
  hence "Inf_para ≤ norm (x - y)"
    by (rule Inf_para_le_norm)
  thus ?thesis
    by (simp add: dist_norm)
qed

```

definition Sup_para :: real where — R in the proof of Lemma 1

```

"Sup_para  $\equiv$  Sup (norm ' lattice_layer_path)"

lemma Sup_para_ge:
  assumes "z  $\in$  lattice_layer_path"
  shows "Sup_para  $\geq$  norm z"
  unfolding Sup_para_def
proof (rule cSup_upper)
  show "bdd_above (norm ' lattice_layer_path)"
    unfolding bdd_above_norm by (rule compact_imp_bounded) auto
qed (use assms in auto)

lemma Sup_para_pos: "Sup_para  $>$  0"
proof -
  have "0  $<$  norm  $\omega_1$ "
    using  $\omega_1$ _nonzero by auto
  also have "...  $\leq$  Sup_para"
    by (rule Sup_para_ge) (auto simp: in_lattice_layer_path_iff)
  finally show ?thesis .
qed

lemma Sup_para_nonzero [simp]: "Sup_para  $\neq$  0"
  using Sup_para_pos by linarith

lemma lattice_layer_ge_norm:
  assumes " $\omega \in$  lattice_layer k"
  shows "norm  $\omega \leq$  k * Sup_para"
proof -
  obtain z where z: "z  $\in$  lattice_layer_path" " $\omega =$  of_nat k * z"
    using lattice_layer_subset[of k] assms by auto
  have "norm  $\omega =$  real k * norm z"
    by (simp add: z norm_mult)
  also have "...  $\leq$  real k * Sup_para"
    by (intro mult_left_mono Sup_para_ge z) auto
  finally show ?thesis .
qed

We can now easily show that our lattice is a sparse set (i.e. it has no limit
points). This also implies that it is closed.

lemma not_islimpt_lattice: " $\neg$ z islimpt  $\Lambda$ "
proof (rule discrete_imp_not_islimpt[of Inf_para])
  fix x y assume "x  $\in$   $\Lambda$ " "y  $\in$   $\Lambda$ " "dist x y  $<$  Inf_para"
  with Inf_para_le_dist[of x y] show "x = y"
    by (cases "x = y") auto
qed (fact Inf_para_pos)

lemma closed_lattice: "closed lattice"
  unfolding closed_limpt by (auto simp: not_islimpt_lattice)

lemma lattice_sparse: " $\Lambda$  sparse_in UNIV"

```

```

using not_islimpt_lattice sparse_in_def by blast

lemma eventually_not_in_lattice_cosparse:
  "eventually ( $\lambda z. z \notin \Lambda$ ) (cosparse UNIV)"
using eventually_not_in_cosparse lattice_sparse by blast

lemma eventually_not_rel_cosparse:
  "eventually ( $\lambda z. \neg \text{rel } z \ w$ ) (cosparse UNIV)"
proof -
  have "eventually ( $\lambda z. \neg \text{rel } z \ w$ ) (cosparse UNIV)  $\longleftrightarrow$ 
    eventually ( $\lambda z. \neg z - w \in \Lambda$ ) (cosparse UNIV)"
    by (simp add: rel_def)
  also have "...  $\longleftrightarrow$  eventually ( $\lambda z. \neg z \in \Lambda$ ) (filtermap ((+) (-w)) (cosparse UNIV))"
    by (simp add: eventually_filtermap)
  also have "filtermap ((+) (-w)) (cosparse UNIV) = cosparse UNIV"
    by (simp add: filtermap_cosparse_translate)
  finally show ?thesis
    using eventually_not_in_lattice_cosparse by blast
qed

```

Any non-empty set of lattice points has one lattice point that is closer to the origin than all others.

```

lemma shortest_lattice_vector_exists:
  assumes "X  $\subseteq \Lambda$ " "X  $\neq \{\}$ "
  obtains x where "x  $\in X$ " " $\bigwedge y. y \in X \implies \text{norm } x \leq \text{norm } y$ "
proof -
  obtain x0 where x0: "x0  $\in X$ "
  using assms by auto
  have " $\neg z$  islimpt X" for z
  using not_islimpt_lattice assms(1) islimpt_subset by blast
  hence "finite (cball 0 (norm x0)  $\cap X$ )"
  by (intro finite_not_islimpt_in_compact) auto
  moreover have "x0  $\in$  cball 0 (norm x0)  $\cap X$ "
  using x0 by auto
  ultimately obtain x where x: "is_arg_min norm ( $\lambda x. x \in$  cball 0 (norm x0)  $\cap X$ ) x"
  using ex_is_arg_min_if_finite[of "cball 0 (norm x0)  $\cap X$ " norm] by blast
  thus ?thesis
  by (intro that[of x]) (auto simp: is_arg_min_def)
qed

```

If x is a non-zero lattice point then there exists another lattice point that is not collinear with x , i.e. that does not lie on the line through 0 and x .

```

lemma noncollinear_lattice_point_exists:
  assumes "x  $\in \Lambda^*$ "
  obtains y where "y  $\in \Lambda^*$ " "y / x  $\notin \mathbb{R}$ "
proof -

```

```

from assms obtain m n where x: "x = of_ω12_coords (of_int m, of_int
n)" and "x ≠ 0"
  by (elim latticeE latticeOE DiffE) auto
define y where "y = of_ω12_coords (of_int (-n), of_int m)"
have "y ∈ Λ"
  by (auto simp: y_def)
moreover have "y ≠ 0"
  using <x ≠ 0> by (auto simp: x y_def of_ω12_coords_eq_0_iff prod_eq_iff)
moreover have "y / x ∉ ℝ"
proof
  assume "y / x ∈ ℝ"
  then obtain a where "y / x = of_real a"
    by (elim Reals_cases)
  hence y: "y = a *R x"
    using assms <x ≠ 0> by (simp add: field_simps scaleR_conv_of_real)
  have "of_ω12_coords (-real_of_int n, real_of_int m) =
    of_ω12_coords (a * real_of_int m, a * real_of_int n)"
    using y by (simp add: x y_def algebra_simps flip: of_ω12_coords.scaleR)
  hence eq: "-real_of_int n = a * real_of_int m" "real_of_int m = a
* real_of_int n"
    unfolding of_ω12_coords_eq_iff prod_eq_iff fst_conv snd_conv by
blast+
  have "m ≠ 0 ∨ n ≠ 0"
    using <x ≠ 0> by (auto simp: x)
  with eq[symmetric] have nz: "m ≠ 0" "n ≠ 0"
    by auto
  have "a ^ 2 * real_of_int m = a * (a * real_of_int m)"
    by (simp add: power2_eq_square algebra_simps)
  also have "... = (-1) * real_of_int m"
    by (simp flip: eq)
  finally have "a ^ 2 = -1"
    using <m ≠ 0> by (subst (asm) mult_right_cancel) auto
  moreover have "a ^ 2 ≥ 0"
    by simp
  ultimately show False
    by linarith
qed
ultimately show ?thesis
  by (intro that) auto
qed

```

We can always easily find a period parallelogram whose border does not touch any given set of points we want to avoid, as long as that set is sparse.

```

lemma shifted_period_parallelogram_avoid:
  assumes "countable avoid"
  obtains orig where "path_image (parallelogram_path orig ω1 ω2) ∩ avoid
= {}"
proof -
  define avoid' where "avoid' = ω12_coords ' avoid"

```

```

define avoid1 where "avoid1 = fst ' avoid'"
define avoid2 where "avoid2 = snd ' avoid'"
define avoid''
  where "avoid'' = (avoid1  $\cup$  ( $\lambda x. x - 1$ ) ' avoid1)  $\times$  UNIV  $\cup$  UNIV  $\times$ 
(avoid2  $\cup$  ( $\lambda x. x - 1$ ) ' avoid2)"

obtain orig where orig: "orig  $\notin$  avoid'"
proof -
  have *: "avoid1  $\cup$  ( $\lambda x. x - 1$ ) ' avoid1  $\in$  null_sets lborel"
    "avoid2  $\cup$  ( $\lambda x. x - 1$ ) ' avoid2  $\in$  null_sets lborel"
  by (rule null_sets.Un; rule countable_imp_null_set_lborel;
    use assms in <force simp: avoid1_def avoid2_def avoid'_def>)+
  have "avoid''  $\in$  null_sets lborel"
    unfolding lborel_prod[symmetric] avoid''_def using * by (intro null_sets.Un)
auto
  hence "AE z in lborel. z  $\notin$  avoid'"
    using AE_not_in by blast
  from eventually_happens[OF this] show ?thesis using that
    by (auto simp: ae_filter_eq_bot_iff)
qed

have *: " $(\lambda x. x - 1)$  ' X = ((+) 1) -' X" for X :: "real set"
  by force

have fst_orig: "fst z  $\notin$  {fst orig, fst orig + 1}" if "z  $\in$  avoid'" for
z
proof
  assume "fst z  $\in$  {fst orig, fst orig + 1}"
  hence "orig  $\in$  (avoid1  $\cup$  ( $\lambda x. x - 1$ ) ' avoid1)  $\times$  UNIV"
    using that unfolding avoid1_def * by (cases orig; cases z) force
  thus False using orig
    by (auto simp: avoid''_def)
qed

have snd_orig: "snd z  $\notin$  {snd orig, snd orig + 1}" if "z  $\in$  avoid'" for
z
proof
  assume "snd z  $\in$  {snd orig, snd orig + 1}"
  hence "orig  $\in$  UNIV  $\times$  (avoid2  $\cup$  ( $\lambda x. x - 1$ ) ' avoid2)"
    using that unfolding avoid2_def * by (cases orig; cases z) force

  thus False using orig
    by (auto simp: avoid''_def)
qed

show ?thesis
proof (rule that[of "of_omega12_coords orig"], safe)
  fix z assume z: "z  $\in$  path_image (parallelogram_path (of_omega12_coords

```

```

orig)  $\omega_1 \omega_2$ )" "z  $\in$  avoid"
  have " $\omega_{12\_coords} z \in \omega_{12\_coords}$  'path_image (parallelogram_path (of_ $\omega_{12\_coords}$ 
orig)  $\omega_1 \omega_2$ )'"
    using z(1) by blast
    thus "z  $\in$  {}" using z(2) fst_orig[of " $\omega_{12\_coords} z$ "] snd_orig[of
" $\omega_{12\_coords} z$ "]
      unfolding path_image_parallelogram_path'
      by (auto simp: avoid'_def  $\omega_{12\_coords}$ .add box_prod)
  qed
qed

```

We can also prove a rule that allows us to prove a property about period parallelograms while assuming w.l.o.g. that the border of the parallelogram does not touch an arbitrary sparse set of points we want to avoid and the property we want to prove is invariant under shifting the parallelogram by an arbitrary amount.

This will be useful later for the use case of showing that any period parallelograms contain the same number of zeros as poles, which is proven by integrating along the border of a period parallelogram that is assume w.l.o.g. not to have any zeros or poles on its border.

```

lemma shifted_period_parallelogram_avoid_wlog [consumes 1, case_names
shift avoid]:
  assumes " $\bigwedge z. \neg z \text{ islimpt avoid}$ "
  assumes " $\bigwedge \text{orig } d. \text{finite (closure (period\_parallelogram orig) } \cap \text{avoid)}$ "
 $\implies$ 
  finite (closure (period\_parallelogram (orig + d))
 $\cap$  avoid)  $\implies$ 
  P orig  $\implies$  P (orig + d)"
  assumes " $\bigwedge \text{orig}. \text{finite (closure (period\_parallelogram orig) } \cap \text{avoid)}$ "
 $\implies$ 
  path_image (parallelogram_path orig  $\omega_1 \omega_2$ )  $\cap$  avoid
= {}  $\implies$ 
  P orig"
  shows "P orig"
proof -
  from assms have countable: "countable avoid"
    using no_limpt_imp_countable by blast

  from shifted_period_parallelogram_avoid[OF countable]
  obtain orig' where orig': "path_image (parallelogram_path orig'  $\omega_1$ 
 $\omega_2$ )  $\cap$  avoid = {}"
    by blast
  define d where "d =  $\omega_{12\_coords}$  (orig - orig')"

  have "compact (closure (period\_parallelogram orig))" for orig
    by (rule compact_closure_period_parallelogram)
  hence fin: "finite (closure (period\_parallelogram orig)  $\cap$  avoid)" for
orig

```

```

    using assms by (intro finite_not_islimpt_in_compact) auto

from orig' have "P orig'"
  by (intro assms fin)
have "P (orig' + (orig - orig'))"
  by (rule assms(2)) fact+
thus ?thesis
  by (simp add: algebra_simps)
qed

end

The standard lattice is one that has been rotated and scaled such that the
first generator is 1 and the second generator  $\tau$  lies in the upper half plane.

locale std_complex_lattice =
  fixes  $\tau$  :: complex (structure)
  assumes Im_ $\tau$ _pos: "Im  $\tau$  > 0"
begin

sublocale complex_lattice 1  $\tau$ 
  by standard (use Im_ $\tau$ _pos in <auto elim!: Reals_cases simp: fundpair_def>)

lemma winding_number_parallelogram_inside':
  assumes "w  $\in$  interior (period_parallelogram z)"
  shows "winding_number (parallelogram_path z 1  $\tau$ ) w = 1"
  using winding_number_parallelogram_inside[OF assms] Im_ $\tau$ _pos by simp

end

```

3.6 Doubly-periodic functions

The following locale can be useful to prove that certain things respect the equivalence relation defined by the lattice: it shows that a doubly periodic function gives the same value for all equivalent points. Note that this is useful even for functions f that are only doubly quasi-periodic, since one might then still be able to prove that the function $\lambda z. f z = 0$ or $zorder f$ or $is_pole f$ are doubly periodic, so the zeros and poles of f are distributed according to the lattice symmetry.

```

locale pre_complex_lattice_periodic = pre_complex_lattice +
  fixes f :: "complex  $\Rightarrow$  'a"
  assumes f_periodic: "f (z +  $\omega$ 1) = f z" "f (z +  $\omega$ 2) = f z"
begin

lemma lattice_cong:
  assumes "rel x y"
  shows "f x = f y"
proof -
  define z where "z = y - x"

```

```

    from assms have z: "z ∈ Λ"
      using pre_complex_lattice.rel_def pre_complex_lattice.rel_sym z_def
  by blast
  have "f (x + z) = f x"
    using z
  proof (induction arbitrary: x rule: lattice_induct)
    case (uminus w x)
    show ?case
      using uminus[of "x - w"] by simp
  qed (auto simp: f_periodic simp flip: add.assoc)
  thus ?thesis
    by (simp add: z_def)
qed

end

locale complex_lattice_periodic =
  complex_lattice ω1 ω2 + pre_complex_lattice_periodic ω1 ω2 f
  for ω1 ω2 :: complex and f :: "complex ⇒ 'a"
begin

lemma eval_to_fund_parallelogram: "f (to_fund_parallelogram z) = f z"
  by (rule lattice_cong) auto

end

locale complex_lattice_periodic_compose =
  complex_lattice_periodic ω1 ω2 f for ω1 ω2 :: complex and f :: "complex
  ⇒ 'a" +
  fixes h :: "'a ⇒ 'b"
begin

sublocale compose: complex_lattice_periodic ω1 ω2 "λz. h (f z)"
  by standard (auto intro!: arg_cong[of _ _ h] lattice_cong simp: rel_def)

end

end

```

4 Subgroups of the modular group

```

theory Modular_Subgroups
  imports Modular_Group
begin

```

4.1 Definition and group action on the upper half plane

```

locale modgrp_subgroup =
  fixes G :: "modgrp set"

```

```

    assumes one_in_G [simp, intro]: "1 ∈ G"
    assumes times_in_G [simp, intro]: "x ∈ G ⇒ y ∈ G ⇒ x * y ∈ G"
    assumes inverse_in_G [simp, intro]: "x ∈ G ⇒ inverse x ∈ G"
begin

lemma divide_in_G [intro]: "f ∈ G ⇒ g ∈ G ⇒ f / g ∈ G"
  unfolding divide_modgrp_def by (intro times_in_G inverse_in_G)

lemma power_in_G [intro]: "f ∈ G ⇒ f ^ n ∈ G"
  by (induction n) auto

lemma power_int_in_G [intro]: "f ∈ G ⇒ f powi n ∈ G"
  by (auto simp: power_int_def)

lemma prod_list_in_G [intro]: "(∧x. x ∈ set xs ⇒ x ∈ G) ⇒ prod_list
xs ∈ G"
  by (induction xs) auto

lemma inverse_in_G_iff [simp]: "inverse f ∈ G ↔ f ∈ G"
  by (metis inverse_in_G modgrp.inverse_inverse)

definition rel :: "complex ⇒ complex ⇒ bool" where
  "rel x y ↔ Im x > 0 ∧ Im y > 0 ∧ (∃f∈G. apply_modgrp f x = y)"

definition orbit :: "complex ⇒ complex set" where
  "orbit x = {y. rel x y}"

lemma Im_nonpos_imp_not_rel: "Im x ≤ 0 ∨ Im y ≤ 0 ⇒ ¬rel x y"
  by (auto simp: rel_def)

lemma orbit_empty: "Im x ≤ 0 ⇒ orbit x = {}"
  by (auto simp: orbit_def Im_nonpos_imp_not_rel)

lemma rel_imp_Im_pos [dest]:
  assumes "rel x y"
  shows "Im x > 0" "Im y > 0"
  using assms by (auto simp: rel_def)

lemma rel_refl [simp]: "rel x x ↔ Im x > 0"
  by (auto simp: rel_def intro!: bexI[of _ 1])

lemma rel_sym:
  assumes "rel x y"
  shows "rel y x"
proof -
  from assms obtain f where f: "f ∈ G" "Im x > 0" "Im y > 0" "apply_modgrp
f x = y"
  by (auto simp: rel_def intro!: bexI[of _ 1])

```

```

from this have "apply_modgrp (inverse f) y = x"
  using pole_modgrp_in_Reals[of f, where ?'a = complex]
  by (intro apply_modgrp_inverse_eqI) (auto simp: complex_is_Real_iff)
moreover have "inverse f ∈ G"
  using f by auto
ultimately show ?thesis
  using f by (auto simp: rel_def)
qed

lemma rel_commutes: "rel x y = rel y x"
  using rel_sym by blast

lemma rel_trans [trans]:
  assumes "rel x y" "rel y z"
  shows "rel x z"
proof -
  from assms obtain f where f: "f ∈ G" "Im x > 0" "Im y > 0" "apply_modgrp
f x = y"
  by (auto simp: rel_def intro!: bexI[of _ 1])
  from assms obtain g where g: "Im z > 0" "g ∈ G" "apply_modgrp g y
= z"
  by (auto simp: rel_def intro!: bexI[of _ 1])
  have "apply_modgrp (g * f) x = z"
  using f g pole_modgrp_in_Reals[of f, where ?'a = complex]
  by (subst apply_modgrp_mult) (auto simp: complex_is_Real_iff)
  with f g show ?thesis
  unfolding rel_def by blast
qed

lemma relI1 [intro]: "rel x y ⇒ f ∈ G ⇒ Im x > 0 ⇒ rel x (apply_modgrp
f y)"
  using modgrp.Im_transform_pos_iff rel_def rel_trans by blast

lemma relI2 [intro]: "rel x y ⇒ f ∈ G ⇒ Im x > 0 ⇒ rel (apply_modgrp
f x) y"
  by (meson relI1 rel_commutes rel_def)

lemma relE:
  assumes "rel x y"
  obtains h where "Im x > 0" "Im y > 0" "h ∈ G" "y = apply_modgrp h x"
  using assms by (auto simp: rel_def)

lemma relE':
  assumes "rel x y"
  obtains h where "Im x > 0" "Im y > 0" "h ∈ G" "x = apply_modgrp h y"
  using rel_sym[OF assms] by (elim relE)

lemma rel_apply_modgrp_left_iff [simp]:
  assumes "f ∈ G"

```

```

  shows "rel (apply_modgrp f x) y  $\longleftrightarrow$  Im x > 0  $\wedge$  rel x y"
proof safe
  assume "rel (apply_modgrp f x) y"
  thus "rel x y"
    by (meson assms modgrp.Im_transform_pos_iff rel_def rel_trans)
next
  assume "rel x y" "Im x > 0"
  thus "rel (apply_modgrp f x) y"
    by (meson assms relI2 rel_trans)
qed auto

lemma rel_apply_modgrp_right_iff [simp]:
  assumes "f  $\in$  G"
  shows "rel y (apply_modgrp f x)  $\longleftrightarrow$  Im x > 0  $\wedge$  rel y x"
  using assms by (metis rel_apply_modgrp_left_iff rel_sym)

lemma orbit_refl_iff: "x  $\in$  orbit x  $\longleftrightarrow$  Im x > 0"
  by (auto simp: orbit_def)

lemma orbit_refl: "Im x > 0  $\implies$  x  $\in$  orbit x"
  by (auto simp: orbit_def)

lemma orbit_cong: "rel x y  $\implies$  orbit x = orbit y"
  using rel_trans rel_commutes unfolding orbit_def by blast

lemma orbit_empty_iff [simp]: "orbit x = {}  $\longleftrightarrow$  Im x  $\leq$  0" "{} = orbit
x  $\longleftrightarrow$  Im x  $\leq$  0"
  using orbit_refl orbit_empty by force+

lemmas [simp] = orbit_refl_iff

lemma orbit_eq_iff: "orbit x = orbit y  $\longleftrightarrow$  Im x  $\leq$  0  $\wedge$  Im y  $\leq$  0  $\vee$  rel
x y"
proof (cases "Im y  $\leq$  0  $\vee$  Im x  $\leq$  0")
  case True
  thus ?thesis
    by (auto simp: orbit_empty)
next
  case False
  have "( $\forall$ z. rel x z  $\longleftrightarrow$  rel y z)  $\longleftrightarrow$  rel x y"
    by (meson False not_le rel_commutes rel_refl rel_trans)
  thus ?thesis
    using False unfolding orbit_def by blast
qed

lemma orbit_apply_modgrp [simp]: "f  $\in$  G  $\implies$  orbit (apply_modgrp f z)
= orbit z"
  by (subst orbit_eq_iff) auto

```

```

lemma apply_modgrp_in_orbit_iff [simp]: "f ∈ G ⇒ apply_modgrp f z
∈ orbit y ↔ z ∈ orbit y"
  by (auto simp: orbit_def rel_commutes)

lemma orbit_imp_Im_pos: "x ∈ orbit y ⇒ Im x > 0"
  by (auto simp: orbit_def)

end

interpretation modular_group: modgrp_subgroup UNIV
  by unfold_locales auto

notation modular_group.rel (infixl "∼Γ" 49)

lemma (in modular_group) rel_imp_rel: "rel x y ⇒ x ∼Γ y"
  unfolding rel_def modular_group.rel_def by auto

lemma modular_group_rel_plus_int_iff_right1 [simp]:
  assumes "z ∈ ℤ"
  shows "x ∼Γ y + z ↔ x ∼Γ y"
proof -
  from assms obtain n where "z = of_int n"
  by (elim Ints_cases)
  have "x ∼Γ apply_modgrp (shift_modgrp n) y ↔ x ∼Γ y"
  by (subst modular_group.rel_apply_modgrp_right_iff) auto
  thus ?thesis
  by (simp add: n)
qed

lemma
  assumes "z ∈ ℤ"
  shows modular_group_rel_plus_int_iff_right2 [simp]: "x ∼Γ z + y ↔
x ∼Γ y"
  and modular_group_rel_plus_int_iff_left1 [simp]: "z + x ∼Γ y ↔
x ∼Γ y"
  and modular_group_rel_plus_int_iff_left2 [simp]: "x + z ∼Γ y ↔
x ∼Γ y"
  using modular_group_rel_plus_int_iff_right1[OF assms] modular_group.rel_commutes
add commute
  by metis+

lemma modular_group_rel_S_iff_right [simp]: "x ∼Γ -(1/y) ↔ x ∼Γ y"
proof -
  have "x ∼Γ apply_modgrp S_modgrp y ↔ x ∼Γ y"
  by (subst modular_group.rel_apply_modgrp_right_iff) auto
  thus ?thesis
  by simp
qed

```

```
lemma modular_group_rel_S_iff_left [simp]: " $-(1/x) \sim_{\Gamma} y \iff x \sim_{\Gamma} y$ "
  using modular_group_rel_S_iff_right[of y x] by (metis modular_group.rel_commutes)
```

The index of a subgroup is the number of cosets.

```
definition index_modgrp :: "modgrp set  $\Rightarrow$  nat" where
  "index_modgrp G = card (range ( $\lambda x. (*) x ' G$ ))"
```

The following defines the group generated by a set of elements.

```
inductive_set generate_modgrp :: "modgrp set  $\Rightarrow$  modgrp set" for X :: "modgrp
set" where
  "x  $\in$  X  $\implies$  x  $\in$  generate_modgrp X"
| "1  $\in$  generate_modgrp X"
| "x  $\in$  generate_modgrp X  $\implies$  y  $\in$  generate_modgrp X  $\implies$  x * y  $\in$  generate_modgrp
X"
| "x  $\in$  generate_modgrp X  $\implies$  inverse x  $\in$  generate_modgrp X"
```

```
lemma modgrp_subgroup_generate: "modgrp_subgroup (generate_modgrp X)"
  by standard (auto intro: generate_modgrp.intros)
```

```
lemma (in modgrp_subgroup) generate_modgrp_subsetI:
  assumes "X  $\subseteq$  G"
  shows "generate_modgrp X  $\subseteq$  G"
```

```
proof
  fix x assume "x  $\in$  generate_modgrp X"
  thus "x  $\in$  G"
    by induction (use assms in auto)
qed
```

4.2 Conjugation

The conjugation of a subgroup G w.r.t. some $h \in \Gamma$ is $h^{-1}Gh$.

```
definition conj_modgrp :: "modgrp  $\Rightarrow$  modgrp set  $\Rightarrow$  modgrp set" where
  "conj_modgrp x G = ( $\lambda y. inverse x * y * x$ ) ' G"
```

```
lemma conj_modgrp_mono: "G  $\subseteq$  H  $\implies$  conj_modgrp x G  $\subseteq$  conj_modgrp x
H"
  by (auto simp: conj_modgrp_def)
```

```
lemma conj_modgrp_altdef: "conj_modgrp x G = ( $\lambda y. x * y * inverse x$ )
- ' G"
```

```
proof -
  have bij: "bij ( $\lambda y. x * y * inverse x$ )"
    by (rule bij_betwI[of _ _ _ " $\lambda y. inverse x * y * x$ "]) (auto simp:
mult.assoc)
  hence " $(\lambda y. x * y * inverse x) - ' G = Hilbert_Choice.inv (\lambda y. x * y
* inverse x) ' G$ "
    by (rule bij_vimage_eq_inv_image)
```

```

    also have "Hilbert_Choice.inv (λy. x * y * inverse x) = (λy. inverse
x * y * x)"
      by (intro inj_imp_inv_eq bij_is_inj bij) (auto simp: mult.assoc)
    finally show ?thesis
      by (simp add: conj_modgrp_def)
qed

lemma conj_modgrp_UNIV [simp]: "conj_modgrp x UNIV = UNIV"
  by (simp add: conj_modgrp_altdef)

lemma in_conj_modgrp_iff: "z ∈ conj_modgrp x G ↔ x * z * inverse x
∈ G"
  by (auto simp: conj_modgrp_altdef)

lemma conj_modgrp_mult: "conj_modgrp (g * f) G = conj_modgrp f (conj_modgrp
g G)"
  by (auto simp: in_conj_modgrp_iff modgrp.inverse_distrib_swap mult.assoc)

lemma conj_modgrp_1 [simp]: "conj_modgrp 1 G = G"
  by (simp add: conj_modgrp_altdef)

context modgrp_subgroup
begin

lemma conj_modgrp_id:
  assumes "x ∈ G"
  shows "conj_modgrp x G = G"
proof (intro equalityI subsetI)
  fix h assume "h ∈ conj_modgrp x G"
  thus "h ∈ G"
    using assms by (auto simp: conj_modgrp_def)
next
  fix h assume "h ∈ G"
  thus "h ∈ conj_modgrp x G"
    using assms by (auto simp: conj_modgrp_altdef)
qed

lemma modgrp_subgroup_conj: "modgrp_subgroup (conj_modgrp x G)"
proof
  fix f g
  assume "f ∈ conj_modgrp x G" "g ∈ conj_modgrp x G"
  hence "x * f * inverse x ∈ G" "x * g * inverse x ∈ G"
    by (auto simp: in_conj_modgrp_iff)
  from times_in_G[OF this] show "f * g ∈ conj_modgrp x G"
    by (simp add: in_conj_modgrp_iff mult.assoc)
next
  fix f assume "f ∈ conj_modgrp x G"
  hence "x * f * inverse x ∈ G"
    by (simp add: in_conj_modgrp_iff)

```

```

    from inverse_in_G[OF this] show "inverse f ∈ conj_modgrp x G"
      by (simp add: in_conj_modgrp_iff mult.assoc modgrp.inverse_distrib_swap
del: inverse_in_G_iff)
qed (auto simp: in_conj_modgrp_iff)

end

```

4.3 Elliptic points

The elliptic order of a point in the complex plane is the size of its stabiliser group (modulo $\pm I$).

```

definition ellorder_modgrp :: "modgrp set ⇒ complex ⇒ nat" where
  "ellorder_modgrp G z =
    (if Im z > 0 then card {h∈G. apply_modgrp h z = z} div card (G ∩
{1, -1}) else 0)"

```

```

lemma (in modgrp_subgroup) ellorder_modgrp_cong:

```

```

  assumes "rel w z"

```

```

  shows "ellorder_modgrp G w = ellorder_modgrp G z"

```

```

proof -

```

```

  from assms obtain h where wz: "h ∈ G" "Im w > 0" "Im z > 0" "apply_modgrp
h w = z"

```

```

    by (auto simp: rel_def)

```

```

  have "ellorder_modgrp G z =

```

```

    card {g ∈ G. apply_modgrp (g * h) w = apply_modgrp h w} div
card (G ∩ {1, -1})"

```

```

    unfolding ellorder_modgrp_def wz(4)[symmetric]

```

```

    by (subst apply_modgrp_mult [symmetric]) (use wz in auto)

```

```

  also have "(λg. apply_modgrp (g * h) w = apply_modgrp h w) =
    (λg. apply_modgrp (inverse h * (g * h)) w = w)"

```

```

    by (subst apply_modgrp_eq_apply_modgrp_iff) (use wz in auto)

```

```

  also have "bij_betw (λg. inverse h * (g * h))

```

```

    {g ∈ G. apply_modgrp (inverse h * (g * h)) w = w}

```

```

    {g ∈ G. apply_modgrp g w = w}"

```

```

    by (rule bij_betwI[of _ _ _ "λg. h * g * inverse h"])

```

```

    (use wz in <auto simp flip: apply_modgrp_mult simp: mult.assoc>)

```

```

  hence "card {g ∈ G. apply_modgrp (inverse h * (g * h)) w = w} =

```

```

    card {g ∈ G. apply_modgrp g w = w}"

```

```

    by (rule bij_betw_same_card)

```

```

  also have "... div card (G ∩ {1, -1}) = ellorder_modgrp G w"

```

```

    using wz by (simp add: ellorder_modgrp_def)

```

```

  finally show ?thesis ..

```

```

qed

```

We define the number of elliptic points of a given order, and the number of cusps (sometimes seen as elliptic points of order ∞).

```

definition ellcount_modgrp :: "modgrp set ⇒ nat ⇒ nat" where
  "ellcount_modgrp G k =

```

```
card ({z. Im z > 0 ∧ ellorder_modgrp G z = k} // {(w,z). modgrp_subgroup.rel
G w z})"
```

```
definition cusp_count_modgrp :: "modgrp set ⇒ nat" where
  "cusp_count_modgrp G = 1 + card ((-pole_modgrp ' G) // {(w::rat,z).
  ∃h. w = apply_modgrp h z})"
```

4.4 Subgroups containing shifts

We will now look at subgroups that contain some shift operator T^n for $n > 0$. The cusp width at infinity is the smallest such n (or equivalently the GCD of all such n).

The cusp width at the other cusps (i.e. a rational numbers) is defined in the same way after conjugation with a modular transformation that sends the rational number to infinity.

```
definition cusp_width_at_ii_inf :: "modgrp set ⇒ nat" ("cusp'_width∞")
where
```

```
"cusp_width_at_ii_inf G = nat (Gcd {n. shift_modgrp n ∈ G})"
```

```
definition cusp_width_modgrp :: "modgrp ⇒ modgrp set ⇒ nat" where
  "cusp_width_modgrp h G = cusp_width∞ (conj_modgrp h G)"
```

```
lemma of_nat_cusp_width_at_ii_inf:
  "of_nat (cusp_width_at_ii_inf G) = Gcd {n. shift_modgrp n ∈ G}"
  unfolding cusp_width_at_ii_inf_def by simp
```

```
lemma cusp_width_at_ii_inf_UNIV [simp]: "cusp_width_at_ii_inf UNIV =
Suc 0"
  by (simp add: cusp_width_at_ii_inf_def)
```

```
lemma (in modgrp_subgroup) shift_modgrp_in_G_iff:
  "shift_modgrp n ∈ G ↔ int (cusp_width_at_ii_inf G) dvd n"
```

```
proof -
```

```
  let ?A = "{n. shift_modgrp n ∈ G}"
  have "?A = {n. int (cusp_width_at_ii_inf G) dvd n}"
    unfolding of_nat_cusp_width_at_ii_inf
    by (rule ideal_int_conv_Gcd) (auto simp: shift_modgrp_add simp flip:
  shift_modgrp_power_int)
  thus ?thesis
    by auto
```

```
qed
```

```
locale modgrp_subgroup_periodic = modgrp_subgroup +
  assumes periodic': "∃n>0. shift_modgrp n ∈ G"
begin
```

```
lemma cusp_width_at_ii_inf_pos: "cusp_width_at_ii_inf G > 0"
proof -
```

```

have "Gcd {n. shift_modgrp n ∈ G} ≠ 0"
  using periodic' by (auto intro!: Nat.grOI simp: Gcd_0_iff)
moreover have "Gcd {n. shift_modgrp n ∈ G} ≥ 0"
  by simp
ultimately show ?thesis
  unfolding cusp_width_at_ii_inf_def by linarith
qed

```

```

lemma shift_modgrp_in_G_period [intro, simp]:
  "shift_modgrp (int (cusp_width_at_ii_inf G)) ∈ G"
  by (subst shift_modgrp_in_G_iff) auto

```

```

lemma shift_modgrp_in_G [intro]:
  "int (cusp_width_at_ii_inf G) dvd n ⇒ shift_modgrp n ∈ G"
  by (subst shift_modgrp_in_G_iff) auto

```

end

4.4.1 Congruence subgroups

The principal congruence subgroup $\Gamma(N)$ consists of those modular transformations A for which $A = I \pmod{N}$ (i.e. they look like the identity modulo N).

```

lift_definition modgrps_pcong :: "int ⇒ modgrp set" ("(<notation=<mixfix
modgrps_pcong>>Γ'(_')") is
  "λN. {(a,b,c,d) :: (int × int × int × int) | a b c d.
    a * d - b * c = 1 ∧ [a = 1] (mod N) ∧ [d = 1] (mod N) ∧ N dvd
b ∧ N dvd c}"
  by (auto simp: rel_set_def)

```

```

lemma modgrps_pcong_altdef:
  "modgrps_pcong N = {f. [f = 1] (modΓ N)}"
  unfolding cong_modgrp_def by transfer (auto simp: cong_0_iff)

```

```

lemma modgrps_pcong_abs [simp]: "modgrps_pcong (abs n) = modgrps_pcong
n"
  by (auto simp: modgrps_pcong_altdef)

```

```

lemma modgrp_in_modgrps_pcong_iff:
  assumes "a * d - b * c = 1"
  shows "modgrp a b c d ∈ modgrps_pcong N ↔ [a = 1] (mod N) ∧ [d
= 1] (mod N) ∧ N dvd b ∧ N dvd c"
  using assms
  by (auto simp: modgrps_pcong_altdef modgrp_c_code modgrp_a_code modgrp_b_code
modgrp_d_code
    cong_modgrp_def cong_0_iff)

```

```

lemma modgrp_in_modgrps_pcong:

```

```

    assumes "[a = 1] (mod N)" "[d = 1] (mod N)" "N dvd b" "N dvd c" "a
* d - b * c = 1"
    shows "modgrp a b c d ∈ modgrps_pcong N"
    using assms
    by (auto simp: modgrps_pcong_altdef modgrp_c_code modgrp_a_code modgrp_b_code
modgrp_d_code
        cong_modgrp_def cong_0_iff)

lemma modgrp_pcong_0 [simp]: "modgrps_pcong 0 = {1}"
  by (auto simp: modgrps_pcong_altdef modgrp_eq_iff cong_modgrp_def cong_0_iff)

lemma modgrp_pcong_1 [simp]: "modgrps_pcong 1 = UNIV"
  by (auto simp: modgrps_pcong_altdef modgrp_eq_iff cong_modgrp_def cong_0_iff)

lemma modgrps_pcong_mono: "n dvd m ⇒ modgrps_pcong m ⊆ modgrps_pcong
n"
  unfolding modgrps_pcong_altdef by (auto intro: cong_dvd_modulus simp:
cong_modgrp_def cong_0_iff)

lemma modgrps_pcong_subset_iff: "modgrps_pcong m ⊆ modgrps_pcong n ↔
n dvd m"
proof
  assume "modgrps_pcong m ⊆ modgrps_pcong n"
  have "shift_modgrp m ∈ modgrps_pcong m"
    by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)
  also note <modgrps_pcong m ⊆ modgrps_pcong n>
  finally show "n dvd m"
    by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)
qed (use modgrps_pcong_mono in auto)

lemma shift_in_modgrps_pcong_iff: "shift_modgrp n ∈ modgrps_pcong d
↔ d dvd n"
  by (auto simp: modgrps_pcong_altdef simp: cong_modgrp_def cong_0_iff)

interpretation modgrps_pcong: modgrp_subgroup "modgrps_pcong N"
proof
  show "inverse x ∈ modgrps_pcong N" if "x ∈ modgrps_pcong N" for x
    using that by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)
next
  show "x * y ∈ modgrps_pcong N" if "x ∈ modgrps_pcong N" "y ∈ modgrps_pcong
N" for x y
  proof -
    from that have "N dvd modgrp_c x" "N dvd modgrp_c y"
      by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)
    have "[modgrp_a x * modgrp_a y + modgrp_b x * modgrp_c y = 1 * 1 +
0 * 0] (mod N)"
      by (intro cong_add cong_mult)
    (use that in <auto simp: modgrps_pcong_altdef cong_0_iff cong_modgrp_def>)
    moreover have "[modgrp_c x * modgrp_b y + modgrp_d x * modgrp_d y

```

```

= 0 * 0 + 1 * 1] (mod N)"
  by (intro cong_add cong_mult)
    (use that in <auto simp: modgrps_pcong_altdef cong_0_iff cong_modgrp_def>)
  ultimately show ?thesis using that
    by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)
qed
qed (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)

```

```

lemma infinite_modgrp_pcong:
  assumes "n ≠ 0"
  shows "infinite (modgrps_pcong n)"
proof -
  have "infinite (UNIV :: int set)"
  by simp
  moreover have "inj (λk. shift_modgrp (k * n))"
  by (rule injI) (use assms in <auto simp: modgrp_eq_iff>)
  ultimately have "infinite (range (λk. shift_modgrp (k * n)))"
  using finite_image_iff by blast
  moreover have "range (λk. shift_modgrp (k * n)) ⊆ modgrps_pcong n"
  by (auto simp: shift_in_modgrps_pcong_iff)
  ultimately show ?thesis
  using finite_subset by blast
qed

```

The level of a subgroup is the smallest n such that it contains $\Gamma(n)$. Equivalently (and more elegantly), it is the GCD of all those numbers n .

definition *level_modgrp* where "level_modgrp $G = \text{Gcd } \{n. \text{modgrps_pcong } n \subseteq G\}$ "

```

lemma level_modgrp_nonneg: "level_modgrp  $G \geq 0$ "
  by (auto simp: level_modgrp_def)

```

```

lemma level_modgrp_UNIV [simp]: "level_modgrp UNIV = 1"
  by (simp add: level_modgrp_def)

```

$\Gamma(n)$ is normal.

```

lemma conj_modgrps_pcong: "conj_modgrp  $x$  (modgrps_pcong  $n$ ) = modgrps_pcong  $n$ "

```

```

proof -
  have *: "modgrps_pcong  $n \subseteq \text{conj\_modgrp } x$  (modgrps_pcong  $n$ )" for  $x$ 
  proof
    fix  $f$  assume  $f: "f \in \text{modgrps\_pcong } n"$ 
    hence "[ $x * f * \text{inverse } x = x * 1 * \text{inverse } x$ ] (mod $_{\Gamma} n$ )"
      by (intro cong_modgrp_mult) (auto simp: modgrps_pcong_altdef)
    thus " $f \in \text{conj\_modgrp } x$  (modgrps_pcong  $n$ )"
      by (simp add: in_conj_modgrp_iff modgrps_pcong_altdef)
  qed

```

```

show ?thesis

```

```

proof
  show "modgrps_pcong n  $\subseteq$  conj_modgrp x (modgrps_pcong n)"
    by (rule *)
next
  have "conj_modgrp x (modgrps_pcong n)  $\subseteq$ 
        conj_modgrp x (conj_modgrp (inverse x) (modgrps_pcong n))"
    by (intro conj_modgrp_mono *)
  also have "... = modgrps_pcong n"
    by (simp flip: conj_modgrp_mult)
  finally show "conj_modgrp x (modgrps_pcong n)  $\subseteq$  modgrps_pcong n" .
qed
qed

```

The level of $\Gamma(N)$ is, unsurprisingly, N .

```

lemma level_conj_modgrp [simp]: "level_modgrp (conj_modgrp x G) = level_modgrp G"

```

```

proof -

```

```

  have "modgrps_pcong n  $\subseteq$  conj_modgrp x G  $\longleftrightarrow$  modgrps_pcong n  $\subseteq$  G" for
  n

```

```

proof

```

```

  assume "modgrps_pcong n  $\subseteq$  G"

```

```

  hence "conj_modgrp x (modgrps_pcong n)  $\subseteq$  conj_modgrp x G"

```

```

    by (intro conj_modgrp_mono)

```

```

  thus "modgrps_pcong n  $\subseteq$  conj_modgrp x G"

```

```

    by (simp add: conj_modgrps_pcong)

```

```

next

```

```

  assume "modgrps_pcong n  $\subseteq$  conj_modgrp x G"

```

```

  hence "conj_modgrp (inverse x) (modgrps_pcong n)  $\subseteq$  conj_modgrp (inverse
x) (conj_modgrp x G)"

```

```

    by (intro conj_modgrp_mono)

```

```

  thus "modgrps_pcong n  $\subseteq$  G"

```

```

    by (simp flip: conj_modgrp_mult add: conj_modgrps_pcong)

```

```

qed

```

```

thus ?thesis

```

```

  by (simp add: level_modgrp_def)

```

```

qed

```

```

lemma cong_lcm_iff:

```

```

  "[a = (b :: 'a :: {unique_euclidean_ring, ring_gcd})] (mod lcm m n)
 $\longleftrightarrow$ "

```

```

  [a = b] (mod m)  $\wedge$  [a = b] (mod n)"

```

```

proof

```

```

  assume "[a = b] (mod lcm m n)"

```

```

  thus "[a = b] (mod m)  $\wedge$  [a = b] (mod n)"

```

```

    by (metis cong_dvd_mono_modulus dvd_lcm1 lcm commute)

```

```

next

```

```

  assume "[a = b] (mod m)  $\wedge$  [a = b] (mod n)"

```

```

  thus "[a = b] (mod lcm m n)"

```

```

    by (auto simp: cong_iff_dvd_diff)
qed

```

Next we investigate $\Gamma(\text{lcm}(n, m))$ and $\Gamma(\text{gcd}(n, m))$. The former is very easy.

```

lemma modgrps_pcong_lcm: "modgrps_pcong (lcm n m) = modgrps_pcong n  $\cap$ 
modgrps_pcong m"
  by (auto simp: modgrps_pcong_altdef cong_lcm_iff cong_modgrp_def)

```

The case for the GCD is slightly more difficult and requires the Chinese Remainder Theorem and the surjectivity of the reduction map.

```

lemma modgrps_pcong_gcd_aux:
  assumes "h  $\in$  modgrps_pcong (gcd m n)"
  obtains f g where "f  $\in$  modgrps_pcong m" "g  $\in$  modgrps_pcong n" "h =
f * g"
proof -
  define D where "D = gcd m n"
  define a b c d where abcd: "a = modgrp_a h" "b = modgrp_b h" "c = modgrp_c
h" "d = modgrp_d h"
  have abcd_cong: "[a = 1] (mod D)" "D dvd b" "D dvd c" "[d = 1] (mod
D)"
    using assms by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff
abcd D_def)
  have det: "a * d - b * c = 1"
    unfolding abcd by (rule modgrp_abcd_det)

  obtain a1 where a1: "[a1 = a] (mod m)" "[a1 = 1] (mod n)"
    using chinese_remainder_theorem_gen[of a 1 m n] abcd_cong by (auto
simp: D_def)
  obtain b1 where b1: "[b1 = b] (mod m)" "[b1 = 0] (mod n)"
    using chinese_remainder_theorem_gen[of b 0 m n] abcd_cong by (auto
simp: D_def cong_0_iff)
  obtain c1 where c1: "[c1 = c] (mod m)" "[c1 = 0] (mod n)"
    using chinese_remainder_theorem_gen[of c 0 m n] abcd_cong by (auto
simp: D_def cong_0_iff)
  obtain d1 where d1: "[d1 = d] (mod m)" "[d1 = 1] (mod n)"
    using chinese_remainder_theorem_gen[of d 1 m n] abcd_cong by (auto
simp: D_def)
  have det1: "[a1 * d1 - b1 * c1 = 1] (mod lcm m n)"
  proof (rule cong_cong_lcm_int)
    have "[a1 * d1 - b1 * c1 = a * d - b * c] (mod m)"
      by (intro cong_diff cong_mult a1 b1 c1 d1)
    thus "[a1 * d1 - b1 * c1 = 1] (mod m)"
      using det by simp
  next
    have "[a1 * d1 - b1 * c1 = 1 * 1 - 0 * 0] (mod n)"
      by (intro cong_diff cong_mult a1 b1 c1 d1)
    thus "[a1 * d1 - b1 * c1 = 1] (mod n)"
      using det by simp
  qed

```

```

    obtain f where "[modgrp_a f = a1] (mod lcm m n)" "[modgrp_b f = b1]
(mod lcm m n)"
                "[modgrp_c f = c1] (mod lcm m n)" "[modgrp_d f = d1]
(mod lcm m n)"
    using modgrp_reduction_surj[OF det1] by blast
    hence f: "[f = h] (mod $\Gamma$  m)" "[f = 1] (mod $\Gamma$  n)"
    using a1 b1 c1 d1 by (auto simp: cong_modgrp_def cong_lcm_iff simp
flip: abcd intro: cong_trans)

    define g where "g = h * inverse f"
    have "[g = h * inverse h] (mod $\Gamma$  m)"
    unfolding g_def by (intro cong_modgrp_mult cong_modgrp_inverse f cong_modgrp_refl)
    hence g: "[g = 1] (mod $\Gamma$  m)"
    by simp
    have gf: "g * f = h"
    by (auto simp: g_def mult.assoc)

    show ?thesis
    using f g gf by (intro that[of g f]) (auto simp: modgrps_pcong_altdef)
qed

lemma (in modgrp_subgroup) modgrps_pcong_gcd:
  fixes m n :: int
  defines "H  $\equiv$  generate_modgrp (modgrps_pcong m  $\cup$  modgrps_pcong n)"
  shows "modgrps_pcong (gcd m n) = H"
proof
  show "H  $\subseteq$  modgrps_pcong (gcd m n)"
  unfolding H_def by (intro modgrps_pcong.generate_modgrp_subsetI Un_least
modgrps_pcong_mono) auto
next
  show "modgrps_pcong (gcd m n)  $\subseteq$  H"
proof
  fix h assume h: "h  $\in$  modgrps_pcong (gcd m n)"
  then obtain f g where fg: "h = f * g" "f  $\in$  modgrps_pcong m" "g  $\in$ 
modgrps_pcong n"
  using modgrps_pcong_gcd_aux[of h m n] h by metis
  thus "h  $\in$  H" unfolding H_def fg(1)
  by (intro generate_modgrp.intros(2-)) (auto intro: generate_modgrp.intros(1))
qed
qed

```

Finally we prove a key lemma: $\Gamma(N)$ is contained in a subgroup iff N is a multiple of the level.

```

lemma (in modgrp_subgroup) contains_modgrps_pcong_iff:
  "modgrps_pcong n  $\subseteq$  G  $\longleftrightarrow$  level_modgrp G dvd n"
proof -
  have "{n. modgrps_pcong n  $\subseteq$  G} = {n. level_modgrp G dvd n}"
  unfolding level_modgrp_def
  proof (rule Gcd_gcd_closed_set_int)

```

```

    show "0 ∈ {n. modgrps_pcong n ⊆ G}"
      by auto
  next
    fix x y :: int
    assume "x ∈ {n. modgrps_pcong n ⊆ G}"
    thus "x * y ∈ {n. modgrps_pcong n ⊆ G}"
      using modgrps_pcong_mono[of x "x * y"] by auto
  next
    fix x y
    assume xy: "x ∈ {n. modgrps_pcong n ⊆ G}" "y ∈ {n. modgrps_pcong
n ⊆ G}"
    have "modgrps_pcong (gcd x y) = generate_modgrp (modgrps_pcong x ∪
modgrps_pcong y)"
      by (rule modgrps_pcong_gcd)
    also have "... ⊆ G"
      by (intro generate_modgrp_subsetI) (use xy in auto)
    finally show "gcd x y ∈ {n. modgrps_pcong n ⊆ G}"
      by simp
  qed
  thus ?thesis
    by auto
qed

```

It is also easy to see that the level is a multiple of all cusp widths. It is in fact even exactly the LCM of the cusp widths (as shown by Wohlfahrt), but we do not show this here.

```

lemma (in modgrp_subgroup) cusp_width_at_ii_inf_dvd_level:
  "cusp_width_at_ii_inf G dvd level_modgrp G"
proof -
  have "Gcd {n. shift_modgrp n ∈ G} dvd level_modgrp G"
  proof (rule Gcd_dvd)
    have "shift_modgrp (level_modgrp G) ∈ modgrps_pcong (level_modgrp
G)"
      by (auto simp: modgrps_pcong_altdef cong_modgrp_def cong_0_iff)
    also have "modgrps_pcong (level_modgrp G) ⊆ G"
      by (auto simp: contains_modgrps_pcong_iff)
    finally show "level_modgrp G ∈ {n. shift_modgrp n ∈ G}"
      by simp
  qed
  thus ?thesis
    by (simp add: cusp_width_at_ii_inf_def)
qed

```

```

lemma level_modgrps_pcong [simp]: "level_modgrp (modgrps_pcong n) = abs
n"
proof -
  have "level_modgrp (modgrps_pcong n) dvd n"
    using level_modgrp_def by simp
  moreover have "modgrps_pcong (level_modgrp (modgrps_pcong n)) ⊆ modgrps_pcong

```

```

n"
  by (simp add: modgrps_pcong.contains_modgrps_pcong_iff)
hence "n dvd level_modgrp (modgrps_pcong n)"
  by (simp add: modgrps_pcong_subset_iff)
ultimately have "abs (level_modgrp (modgrps_pcong n)) = abs n"
  by (intro zdvd_antisym_abs)
thus ?thesis
  using level_modgrp_nonneg[of "modgrps_pcong n"] by simp
qed

```

A *congruence subgroup* is a subgroup of the modular group that contains $\Gamma(N)$ for some $N > 0$.

```

locale cong_subgroup = modgrp_subgroup +
  assumes level_pos: "level_modgrp G > 0"
begin

```

```

definition cusp_width :: "rat  $\Rightarrow$  nat" where
  "cusp_width x = (let f = modgrp_of_rat x in Gcd {n. inverse f * shift_modgrp
(int n) * f  $\in$  G})"

```

```

lemma cong_subgroup_conj: "cong_subgroup (conj_modgrp x G)"
proof -

```

```

  interpret conj: modgrp_subgroup "conj_modgrp x G"
  by (rule modgrp_subgroup_conj)
  show ?thesis
  proof
    show "level_modgrp (conj_modgrp x G) > 0"
      by (simp add: level_pos)
  qed
qed

```

```

sublocale modgrp_subgroup_periodic G
proof

```

```

  have "cusp_width $_{\infty}$  G dvd level_modgrp G"
  by (rule cusp_width_at_ii_inf_dvd_level)
  with level_pos have "cusp_width $_{\infty}$  G > 0"
  by (intro Nat.gr0I) auto
  thus " $\exists n > 0. \text{shift\_modgrp } n \in G$ "
  by (intro exI[of _ "int (cusp_width_at_ii_inf G)"]) (auto simp: shift_modgrp_in_G_iff)
qed

```

```

lemma infinite: "infinite G"

```

```

proof -
  have "modgrps_pcong (level_modgrp G)  $\subseteq$  G"
  using contains_modgrps_pcong_iff by simp
  moreover have "infinite (modgrps_pcong (level_modgrp G))"
  by (intro infinite_modgrp_pcong) (use level_pos in auto)
  ultimately show ?thesis
  using finite_subset by blast

```

qed

end

lemma *cong_subgroup_pcong*: " $N > 0 \implies \text{cong_subgroup } (\text{modgrps_pcong } N)$ "
by *standard auto*

interpretation *modular_group*: *cong_subgroup UNIV*
rewrites "*cuspid_width_at_ii_inf UNIV = Suc 0*"
by *unfold_locales (auto intro: exI[of _ 1] simp: cuspid_width_at_ii_inf_def)*

lemma *cong_subgroup_UNIV* [*intro*]: "*cong_subgroup UNIV*" ..

4.4.2 Hecke subgroups $\Gamma_0(N)$

The Hecke subgroup of level N , $\Gamma_0(N)$, is a superset of $\Gamma(N)$. It only requires that $c \equiv 0 \pmod{N}$, i.e. it consists of the matrices that are lower triangular matrices modulo N .

All cusps have width 1 in $\Gamma_0(N)$.

lift_definition *modgrps_hecke* :: " $\text{int} \implies \text{modgrp set}$ " (" $\langle \text{notation} = \langle \text{mixfix modgrps_hecke} \rangle \Gamma_0'(_)$ ") is
" $\lambda N. \{(a,b,c,d) :: (\text{int} \times \text{int} \times \text{int} \times \text{int}) \mid a \ b \ c \ d. \ a * d - b * c = 1 \wedge N \ \text{dvd} \ c\}$ "
by (*auto simp: rel_set_def*)

lemma *modgrps_hecke_altdef*: "*modgrps_hecke* $q = \{f. q \ \text{dvd} \ \text{modgrp_c } f\}$ "
by *transfer' auto*

lemma *modgrp_in_modgrps_hecke_iff*:
assumes " $a * d - b * c = 1$ "
shows " $\text{modgrp } a \ b \ c \ d \in \text{modgrps_hecke } q \iff q \ \text{dvd} \ c$ "
using *assms* by (*auto simp: modgrps_hecke_altdef modgrp_c_code*)

lemma *modgrp_in_modgrps_hecke*:
assumes " $q \ \text{dvd} \ c$ " " $a * d - b * c = 1$ "
shows " $\text{modgrp } a \ b \ c \ d \in \text{modgrps_hecke } q$ "
using *assms* by (*auto simp: modgrps_hecke_altdef modgrp_c_code*)

lemma *shift_in_modgrps_hecke* [*simp*]: "*shift_modgrp* $n \in \text{modgrps_hecke } q$ "
by (*auto simp: modgrps_hecke_altdef*)

lemma *cuspid_width_at_ii_inf_hecke* [*simp*]: "*cuspid_width* $_{\infty} (\text{modgrps_hecke } q) = 1$ "
by (*simp add: cuspid_width_at_ii_inf_def*)

lemma *S_in_modgrps_hecke_iff* [*simp*]: "*S_modgrp* $\in \text{modgrps_hecke } q \iff \text{is_unit } q$ "

```

    by (auto simp: modgrps_hecke_altdef)

lemma level_modgrps_hecke [simp]: "level_modgrp (modgrps_hecke N) = abs
N"
proof -
  have "modgrps_pcong N  $\subseteq$  modgrps_hecke N"
    by (auto simp: modgrps_pcong_altdef modgrps_hecke_altdef cong_modgrp_def
cong_0_iff)
  hence "level_modgrp (modgrps_hecke N) dvd N"
    by (simp add: level_modgrp_def)
  moreover have "N dvd level_modgrp (modgrps_hecke N)"
    unfolding level_modgrp_def
  proof (rule Gcd_greatest, safe)
    fix n assume "modgrps_pcong n  $\subseteq$  modgrps_hecke N"
    have "modgrp 1 0 n 1  $\in$  modgrps_pcong n"
      by (auto simp: modgrps_pcong_altdef cong_modgrp_def modgrp_abcd_modgrp
cong_0_iff)
    also note <modgrps_pcong n  $\subseteq$  modgrps_hecke N>
    finally show "N dvd n"
      by (auto simp: modgrps_hecke_altdef modgrp_c_modgrp)
  qed
  ultimately have "abs (level_modgrp (modgrps_hecke N)) = abs N"
    using zdvd_antisym_abs by blast
  thus ?thesis
    using level_modgrp_nonneg[of "modgrps_hecke N"] by simp
qed

```

```

locale hecke_subgroup =
  fixes q :: int
  assumes q_pos: "q > 0"
begin

```

```

definition subgrp (" $\Gamma$ '") where "subgrp = modgrps_hecke q"

```

```

lemma S_in_subgrp_iff [simp]: "S_modgrp  $\in$  subgrp  $\longleftrightarrow$  q = 1"
  using q_pos by (auto simp: subgrp_def)

```

```

sublocale modgrp_subgroup  $\Gamma$ '

```

```

proof
  show "inverse x  $\in$   $\Gamma$ '" if "x  $\in$   $\Gamma$ '" for x
  proof -
    from that have "q dvd modgrp_c x"
      by (simp add: modgrps_hecke_altdef subgrp_def)
    hence "q dvd modgrp_c (inverse x)"
      by transfer auto
    thus "inverse x  $\in$   $\Gamma$ '"
      by (simp add: modgrps_hecke_altdef subgrp_def)
  qed

```

```

next
  show "x * y ∈ Γ'" if "x ∈ Γ'" "y ∈ Γ'" for x y
  proof -
    from that have "q dvd modgrp_c x" "q dvd modgrp_c y"
      by (auto simp: modgrps_hecke_altdef subgrp_def)
    hence "q dvd modgrp_c (x * y)"
      by transfer auto
    thus ?thesis
      by (auto simp: modgrps_hecke_altdef subgrp_def)
  qed
qed (auto simp: subgrp_def modgrps_hecke_altdef)

sublocale cong_subgroup Γ'
proof
  show "level_modgrp Γ' > 0"
    using q_pos by (simp add: subgrp_def)
qed

end

```

Next, we focus on the Hecke subgroups of prime d .

```

locale hecke_prime_subgroup =
  fixes p :: int
  assumes p_prime: "prime p"
begin

lemma p_pos: "p > 0"
  using p_prime by (simp add: prime_gt_0_int)

lemma p_not_1 [simp]: "p ≠ 1"
  using p_prime by auto

sublocale hecke_subgroup p
  by standard (rule p_pos)

notation subgrp ("Γ'")

definition S_shift_modgrp where "S_shift_modgrp n = S_modgrp * shift_modgrp
n"

Every transformation  $f \in \Gamma$  that is not in the subgroup can be written as a
product  $f = gST^k$ , where  $g$  is in the subgroup.

lemma modgrp_decompose:
  assumes "f ∉ Γ'"
  obtains g k where "g ∈ Γ'" "k ∈ {0..<p}" "f = g * S_modgrp * shift_modgrp
k"
proof -
  define a b c d where "a = modgrp_a f" "b = modgrp_b f" "c = modgrp_c
f" "d = modgrp_d f"

```

```

have det: "a * d - b * c = 1"
  unfolding a_b_c_d_def using modgrp_abcd_det[of f] by simp
have "-p dvd c"
  unfolding a_b_c_d_def using assms by (auto simp: subgrp_def modgrps_hecke_altdef)
hence "coprime p c"
  using p_prime by (intro prime_imp_coprime) auto
define k where "k = (modular_inverse p c * d) mod p"
have "[k * c = (modular_inverse p c * d) mod p * c] (mod p)"
  by (simp add: k_def)
also have "[ (modular_inverse p c * d) mod p * c = modular_inverse p
c * d * c] (mod p)"
  by (intro cong_mult cong_mod_leftI cong_refl)
also have "modular_inverse p c * d * c = modular_inverse p c * c * d"
  by (simp add: mult_ac)
also have "[... = 1 * d] (mod p)" using <coprime p c>
  by (intro cong_mult cong_refl cong_modular_inverse2) (auto simp: coprime_commute)
finally have "[k * c = d] (mod p)"
  by simp
hence dvd: "p dvd k * c - d"
  by (simp add: cong_iff_dvd_diff)

have det': "(k * a - b) * c - a * (k * c - d) = 1"
  using det by (simp add: algebra_simps)
define g where "g = modgrp (k * a - b) a (k * c - d) c"

show ?thesis
proof (rule that)
  show "g ∈ Γ'"
    unfolding subgrp_def g_def using det' dvd
    by (intro modgrp_in_modgrps_hecke) auto
next
  show "k ∈ {0..<p}"
    unfolding k_def using p_pos by simp
next
  have "g * S_modgrp * shift_modgrp k = modgrp a b c d" using det
    by (auto simp: g_def S_modgrp_code shift_modgrp_code times_modgrp_code
algebra_simps)
  also have "... = f"
    by (simp add: a_b_c_d_def)
  finally show "f = g * S_modgrp * shift_modgrp k" ..
qed
qed

lemma modgrp_decompose':
  obtains g h
  where "g ∈ Γ'" "h = 1 ∨ (∃k∈{0..<p}. h = S_shift_modgrp k)" "f
= g * h"
proof (cases "f ∈ Γ'")
  case True

```

```

    thus ?thesis
      using that[of f 1] by auto
next
  case False
  thus ?thesis
    using modgrp_decompose[of f] that modgrp.assoc unfolding S_shift_modgrp_def
    by metis
qed

end

```

4.5 The subgroups $\Gamma_1(N)$

The following subgroups lie inbetween $\Gamma(N)$ and $\Gamma_0(N)$. They consist of those matrices that become upper triangular matrices with 1 on the diagonal when reduced modulo N .

These groups do not seem to have a name in the literature. We call them the “unipotent subgroups modulo N ” (since upper triangular matrices with 1 on the diagonal are exactly the unipotent matrices).

Again, the level of $\Gamma(N)$ is N and the cusp widths are all 1.

```

lift_definition modgrps_unip :: "int  $\Rightarrow$  modgrp set" ("(<notation=<mixfix
modgrps_unip>> $\Gamma_1$ '(_')") is
  " $\lambda N. \{(a,b,c,d) :: (int \times int \times int \times int) \mid a \ b \ c \ d. a * d - b *
c = 1 \wedge$ 
    [a = 1] (mod N)  $\wedge$  N dvd c  $\wedge$  [d = 1] (mod N)}"
  by auto

```

```

lemma modgrps_unip_altdef:
  "modgrps_unip N = {f. [modgrp_a f = 1] (mod N)  $\wedge$  [modgrp_d f = 1] (mod
N)  $\wedge$  N dvd modgrp_c f}"
  by transfer auto

```

```

lemma modgrps_unip_subset_hecke: "modgrps_unip N  $\subseteq$  modgrps_hecke N"
  by (auto simp: modgrps_unip_def modgrps_hecke_def)

```

```

lemma modgrp_in_modgrps_unip_iff:
  assumes "a * d - b * c = 1"
  shows "modgrp a b c d  $\in$  modgrps_unip N  $\longleftrightarrow$  [a = 1] (mod N)  $\wedge$  [d
= 1] (mod N)  $\wedge$  N dvd c"
  using assms by (auto simp: modgrps_unip_altdef modgrp_c_code modgrp_a_code
modgrp_d_code)

```

```

lemma shift_in_modgrps_unip [simp]: "shift_modgrp n  $\in$  modgrps_unip N"
  by (auto simp: modgrps_unip_altdef)

```

```

lemma cusp_width_at_ii_inf_unip [simp]: "cusp_width $_{\infty}$  (modgrps_unip N)
= 1"
  by (simp add: cusp_width_at_ii_inf_def)

```

```

lemma S_in_modgrps_unip_iff [simp]: "S_modgrp ∈ modgrps_unip N ↔ is_unit
N"
  by (auto simp: modgrps_unip_altdef cong_iff_dvd_diff)

lemma level_modgrps_unip [simp]: "level_modgrp (modgrps_unip N) = abs
N"
proof -
  have "modgrps_pcong N ⊆ modgrps_unip N"
    by (auto simp: modgrps_pcong_altdef modgrps_unip_altdef cong_modgrp_def
cong_0_iff)
  hence "level_modgrp (modgrps_unip N) dvd N"
    by (simp add: level_modgrp_def)
  moreover have "N dvd level_modgrp (modgrps_unip N)"
    unfolding level_modgrp_def
  proof (rule Gcd_greatest, safe)
    fix n assume "modgrps_pcong n ⊆ modgrps_unip N"
    have "modgrp 1 0 n 1 ∈ modgrps_pcong n"
      by (auto simp: modgrps_pcong_altdef cong_modgrp_def modgrp_abcd_modgrp
cong_0_iff)
    also note <modgrps_pcong n ⊆ modgrps_unip N>
    finally show "N dvd n"
      by (auto simp: modgrps_unip_altdef modgrp_c_modgrp)
  qed
  ultimately have "abs (level_modgrp (modgrps_unip N)) = abs N"
    using zdvd_antisym_abs by blast
  thus ?thesis
    using level_modgrp_nonneg[of "modgrps_unip N"] by simp
qed

locale unip_subgroup =
  fixes N :: int
  assumes N_pos: "N > 0"
begin

definition subgrp ("Γ'") where "subgrp = modgrps_unip N"

lemma S_in_subgrp_iff [simp]: "S_modgrp ∈ subgrp ↔ N = 1"
  using N_pos by (auto simp: subgrp_def)

sublocale modgrp_subgroup Γ'
proof
  show "inverse x ∈ Γ'" if "x ∈ Γ'" for x
    using that by (auto simp: modgrps_unip_altdef subgrp_def)
next
  show "x * y ∈ Γ'" if "x ∈ Γ'" "y ∈ Γ'" for x y
  proof -
    have c: "N dvd modgrp_c (x * y)"

```

```

    using that by (auto simp: subgrp_def modgrps_unip_altdef)
    have "[modgrp_a x * modgrp_a y + modgrp_b x * modgrp_c y = 1 * 1 +
modgrp_b x * 0] (mod N)"
      by (intro cong_add cong_mult)
      (use that in <auto simp: subgrp_def modgrps_unip_altdef cong_0_iff>)
    hence a: "[modgrp_a (x * y) = 1] (mod N)"
      by simp
    have "[modgrp_c x * modgrp_b y + modgrp_d x * modgrp_d y = 0 * modgrp_b
y + 1 * 1] (mod N)"
      by (intro cong_add cong_mult)
      (use that in <auto simp: subgrp_def modgrps_unip_altdef cong_0_iff>)
    hence d: "[modgrp_d (x * y) = 1] (mod N)"
      by simp
    show ?thesis using a c d
      by (auto simp: modgrps_unip_altdef subgrp_def)
  qed
qed (auto simp: subgrp_def modgrps_unip_altdef)

```

```

sublocale cong_subgroup  $\Gamma'$ 
proof
  show "level_modgrp  $\Gamma'$  > 0"
    using N_pos by (simp add: subgrp_def)
qed

```

end

```

bundle modgrp_notation
begin

```

```

notation modular_group.rel (infixl " $\sim_{\Gamma}$ " 49)
notation modgrps_pcong (" $\langle$ notation= $\langle$ mixfix modgrps_pcong $\rangle\rangle\Gamma'(_')$ ")
notation modgrps_hecke (" $\langle$ notation= $\langle$ mixfix modgrps_hecke $\rangle\rangle\Gamma_0'(_')$ ")
notation modgrps_unip (" $\langle$ notation= $\langle$ mixfix modgrps_unip $\rangle\rangle\Gamma_1'(_')$ ")

```

end

```

unbundle no modgrp_notation

```

end

5 Fundamental regions of the modular group

```

theory Modular_Fundamental_Region
  imports Modular_Subgroups "Elliptic_Functions.Complex_Lattices" "HOL-Library.Real_Mod"
begin

unbundle modgrp_notation

```

5.1 Definition

A fundamental region of a subgroup of the modular group is an open subset of the upper half of the complex plane that contains at most one representative of every equivalence class and whose closure contains at least one representative of every equivalence class.

```

locale fundamental_region = modgrp_subgroup +
  fixes R :: "complex set"
  assumes "open": "open R"
  assumes subset: "R  $\subseteq$  {z. Im z > 0}"
  assumes unique: " $\bigwedge x y. x \in R \implies y \in R \implies \text{rel } x y \implies x = y$ "
  assumes equiv_in_closure: " $\bigwedge x. \text{Im } x > 0 \implies \exists y \in \text{closure } R. \text{rel } x y$ "
  "

```

begin

The uniqueness property can be extended to the closure of R:

lemma unique':

```

  assumes "x  $\in$  R" "y  $\in$  closure R" "rel x y" "Im y > 0"
  shows "x = y"
proof (cases "y  $\in$  R")
  case False
  show ?thesis
  proof (rule ccontr)
    assume xy: "x  $\neq$  y"
    from assms have "rel y x"
      by (simp add: rel_commutes)
    then obtain f where f: "x = apply_modgrp f y" "f  $\in$  G"
      unfolding rel_def by blast

    have "continuous_on {z. Im z > 0} (apply_modgrp f)"
      by (intro continuous_intros) auto
    hence "isCont (apply_modgrp f) y"
      using open_halfspace_Im_gt[of 0] assms continuous_on_eq_continuous_at
  by blast
    hence lim: "apply_modgrp f  $\rightarrow$  x"
      using f by (simp add: isCont_def)

    define  $\varepsilon$  where " $\varepsilon = \text{dist } x y / 2$ "
    have  $\varepsilon$ : " $\varepsilon > 0$ "
      using xy by (auto simp:  $\varepsilon$ _def)

    have "eventually ( $\lambda w. w \in \text{ball } x \ \varepsilon \cap R$ ) (nhds x)"
      by (intro eventually_nhds_in_open) (use assms  $\varepsilon$  "open" in auto)
    from this and lim have "eventually ( $\lambda z. \text{apply\_modgrp } f \ z \in \text{ball } x \ \varepsilon \cap R$ ) (at y)"
      by (rule eventually_compose_filterlim)
    moreover have "eventually ( $\lambda z. z \in \text{ball } y \ \varepsilon$ ) (nhds y)"
      using assms  $\varepsilon$  by (intro eventually_nhds_in_open) auto
    hence "eventually ( $\lambda z. z \in \text{ball } y \ \varepsilon$ ) (at y)"

```

```

    unfolding eventually_at_filter by eventually_elim auto
    ultimately have "eventually ( $\lambda z. z \in \text{ball } y \ \varepsilon \wedge \text{apply\_modgrp } f \ z \in R \cap \text{ball } x \ \varepsilon$ ) (at y)"
    by eventually_elim auto
    moreover have "y islimpt R"
    using  $\langle y \in \text{closure } R \rangle \langle y \notin R \rangle$  by (auto simp: closure_def)
    hence "frequently ( $\lambda z. z \in R$ ) (at y)"
    using islimpt_conv_frequently_at by blast
    ultimately have "frequently ( $\lambda z. (z \in \text{ball } y \ \varepsilon \wedge \text{apply\_modgrp } f \ z \in R \cap \text{ball } x \ \varepsilon) \wedge z \in R$ ) (at y)"
    by (intro frequently_eventually_conj)
    hence "frequently ( $\lambda z. \text{False}$ ) (at y)"
    proof (rule frequently_elim1)
      fix z assume z: " $(z \in \text{ball } y \ \varepsilon \wedge \text{apply\_modgrp } f \ z \in R \cap \text{ball } x \ \varepsilon) \wedge z \in R$ "
      have " $\text{ball } y \ \varepsilon \cap \text{ball } x \ \varepsilon = \{\}$ "
      by (intro disjoint_ballI) (auto simp:  $\varepsilon$ _def dist_commute)
      with z have " $\text{apply\_modgrp } f \ z \neq z$ "
      by auto
      with z f subset show False
      using unique[of z " $\text{apply\_modgrp } f \ z$ "] by auto
    qed
    thus False
    by simp
  qed
qed (use assms unique in auto)

```

lemma

```

pole_modgrp_not_in_region [simp]: "pole_modgrp f  $\notin R$ " and
pole_image_modgrp_not_in_region [simp]: "pole_image_modgrp f  $\notin R$ "
using subset by force+

```

end

5.2 The standard fundamental region

The standard fundamental region \mathcal{R}_Γ consists of all the points z in the upper half plane with $|z| > 1$ and $|\text{Re}(z)| < \frac{1}{2}$.

definition *std_fund_region* :: "complex set" (" \mathcal{R}_Γ ") where
 $\mathcal{R}_\Gamma = \text{-cball } 0 \ 1 \cap \text{Re } -' \{-1/2 < .. < 1/2\} \cap \{z. \text{Im } z > 0\}$

The following version of \mathcal{R}_Γ is what Apostol refers to as the closure of \mathcal{R}_Γ , but it is actually only part of the closure: since each point at the border of the fundamental region is equivalent to its mirror image w.r.t. the $\text{Im}(z) = 0$ axis, we only want one of these copies to be in \mathcal{R}_Γ' , and we choose the left one.

So \mathcal{R}_Γ' is actually \mathcal{R}_Γ plus all the points on the left border plus all points on the left half of the semicircle.

definition `std_fund_region'` :: "complex set" (" \mathcal{R}_Γ' ") where
`" \mathcal{R}_Γ' " = $\mathcal{R}_\Gamma \cup (-\text{ball } 0 \ 1 \cap \text{Re } -' \{-1/2..0\} \cap \{z. \text{Im } z > 0\})$ "`

lemma `std_fund_region_altdef`:
`" \mathcal{R}_Γ " = $\{z. \text{norm } z > 1 \wedge \text{norm } (z + \text{cnj } z) < 1 \wedge \text{Im } z > 0\}$ "
 by (auto simp: std_fund_region_def complex_add_cnj)`

lemma `in_std_fund_region_iff`:
`" $z \in \mathcal{R}_\Gamma \iff \text{norm } z > 1 \wedge \text{Re } z \in \{-1/2<.. $1/2\} \wedge \text{Im } z > 0$ "$`
 by (auto simp: `std_fund_region_def` `field_simps`)

lemma `in_std_fund_region'_iff`:
`" $z \in \mathcal{R}_\Gamma' \iff \text{Im } z > 0 \wedge ((\text{norm } z > 1 \wedge \text{Re } z \in \{-1/2<.. $1/2\}) \vee (\text{norm } z = 1 \wedge \text{Re } z \in \{-1/2..0\}))$ "$`
 by (auto simp: `std_fund_region'_def` `std_fund_region_def` `field_simps`)

lemma `open_std_fund_region` [`simp`, `intro`]: "`open \mathcal{R}_Γ "`
 unfolding `std_fund_region_def`
 by (intro `open_Int` `open_vimage` `continuous_intros` `open_halfspace_Im_gt`)
 auto

lemma `Im_std_fund_region`: " `$z \in \mathcal{R}_\Gamma \implies \text{Im } z > 0$ "`
 by (auto simp: `std_fund_region_def`)

We now show that the closure of the standard fundamental region contains exactly those points z with $|z| \geq 1$ and $|\text{Re}(z)| \leq \frac{1}{2}$.

context
 fixes `S S'` :: "(real \times real) set" and `T` :: "complex set"
 fixes `f` :: "real \times real \Rightarrow complex" and `g` :: "complex \Rightarrow real \times real"
 defines "`f` $\equiv (\lambda(x,y). \text{Complex } x (y + \text{sqrt } (1 - x^2)))$ "
 defines "`g` $\equiv (\lambda z. (\text{Re } z, \text{Im } z - \text{sqrt } (1 - \text{Re } z^2)))$ "
 defines "`S` $\equiv (\{-1/2<.. $1/2\} \times \{0<..\})$ "
 defines "`S'` $\equiv (\{-1/2..1/2\} \times \{0..})$ "
 defines "`T` $\equiv \{z. \text{norm } z \geq 1 \wedge \text{Re } z \in \{-1/2..1/2\} \wedge \text{Im } z \geq 0\}$ "
begin$

lemma `image_subset_std_fund_region`: " `$f ' S \subseteq \mathcal{R}_\Gamma$ "`
 unfolding `subset_iff_in_std_fund_region_iff` `S_def`
proof `safe`

fix `a b` :: real
 assume `ab`: " `$a \in \{-1/2<.. $1/2\}$ " $b > 0$ "$`
 have " `$|a|^2 \leq (1/2)^2$ "`
 using `ab` by (intro `power_mono`) auto
 hence " `$a^2 \leq 1/4$ "`
 by (simp add: `power2_eq_square`)
 hence " `$a^2 \leq 1$ "`
 by simp

```

show "Im (f (a, b)) > 0"
  using ab <a ^ 2 ≤ 1 / 4> by (auto simp: f_def intro: add_pos_nonneg)

show "Re (f (a, b)) ∈ {-1/2<..<1/2}"
  using ab by (simp add: f_def)

have "1 ^ 2 = a^2 + (0 + sqrt (1 - a^2)) ^ 2"
  using <a ^ 2 ≤ 1 / 4> by (simp add: power2_eq_square algebra_simps)
also have "a^2 + (0 + sqrt (1 - a^2)) ^ 2 < a^2 + (b + sqrt (1 - a^2)) ^
2"
  using ab <a ^ 2 ≤ 1> by (intro add_strict_left_mono power2_mono power2_strict_mono)
auto
also have "... = norm (f (a, b)) ^ 2"
  by (simp add: f_def norm_complex_def)
finally show "norm (f (a, b)) > 1"
  by (rule power2_less_imp_less) auto
qed

lemma image_std_fund_region_subset: "g ' R_Γ ⊆ S"
  unfolding subset_iff g_def S_def
proof safe
  fix z :: complex
  assume "z ∈ R_Γ"
  hence z: "norm z > 1" "Re z ∈ {-1/2<..<1/2}" "Im z > 0"
    by (auto simp: in_std_fund_region_iff)

  have "|Re z| ^ 2 ≤ (1 / 2) ^ 2"
    using z by (intro power_mono) auto
  hence "Re z ^ 2 ≤ 1 / 4"
    by (simp add: power2_eq_square)
  hence "Re z ^ 2 ≤ 1"
    by simp

  from z show "Re z ∈ {- 1 / 2<..<1 / 2}"
    by auto

  have "sqrt (1 - Re z ^ 2) ^ 2 = 1 - Re z ^ 2"
    using <Re z ^ 2 ≤ 1> by simp
  also have "... < Im z ^ 2"
    using z by (simp add: norm_complex_def algebra_simps)
  finally have "sqrt (1 - Re z ^ 2) < Im z"
    by (rule power2_less_imp_less) (use z in auto)
  thus "Im z - sqrt (1 - Re z ^ 2) > 0"
    by simp
qed

lemma std_fund_region_map_inverses: "f (g x) = x" "g (f y) = y"
  by (simp_all add: f_def g_def case_prod_unfold)

```

```

lemma bij_betw_std_fund_region1: "bij_betw f S  $\mathcal{R}_\Gamma$ "
  using image_std_fund_region_subset image_subset_std_fund_region
  by (intro bij_betwI[of _ _ _ g]) (auto simp: std_fund_region_map_inverses)

lemma bij_betw_std_fund_region2: "bij_betw g  $\mathcal{R}_\Gamma$  S"
  using image_std_fund_region_subset image_subset_std_fund_region
  by (intro bij_betwI[of _ _ _ f]) (auto simp: std_fund_region_map_inverses)

lemma image_subset_std_fund_region': "f ' S'  $\subseteq$  T"
  unfolding subset_iff S'_def T_def
proof safe
  fix a b :: real
  assume ab: "a  $\in$   $\{-1/2..1/2\}$ " "b  $\geq$  0"
  have "|a| ^ 2  $\leq$  (1 / 2) ^ 2"
    using ab by (intro power_mono) auto
  hence "a ^ 2  $\leq$  1 / 4"
    by (simp add: power2_eq_square)
  hence "a ^ 2  $\leq$  1"
    by simp

  show "Im (f (a, b))  $\geq$  0"
    using ab <a ^ 2  $\leq$  1 / 4> by (auto simp: f_def intro: add_pos_nonneg)

  show "Re (f (a, b))  $\in$   $\{-1/2..1/2\}$ "
    using ab by (simp add: f_def)

  have "1 ^ 2 = a^2 + (0 + sqrt (1 - a^2)) ^ 2"
    using <a ^ 2  $\leq$  1 / 4> by (simp add: power2_eq_square algebra_simps)
  also have "a^2 + (0 + sqrt (1 - a^2)) ^ 2  $\leq$  a^2 + (b + sqrt (1 - a^2))
  ^ 2"
    using ab <a ^ 2  $\leq$  1> by (intro add_left_mono power2_mono power2_strict_mono)
  auto
  also have "... = norm (f (a, b)) ^ 2"
    by (simp add: f_def norm_complex_def)
  finally show "norm (f (a, b))  $\geq$  1"
    by (rule power2_le_imp_le) auto
qed

lemma image_std_fund_region_subset': "g ' T  $\subseteq$  S'"
  unfolding subset_iff g_def S'_def
proof safe
  fix z :: complex
  assume "z  $\in$  T"
  hence z: "norm z  $\geq$  1" "Re z  $\in$   $\{-1/2..1/2\}$ " "Im z  $\geq$  0"
    by (auto simp: T_def)

  have "|Re z| ^ 2  $\leq$  (1 / 2) ^ 2"
    using z by (intro power_mono) auto

```

```

hence "Re z ^ 2 ≤ 1 / 4"
  by (simp add: power2_eq_square)
hence "Re z ^ 2 ≤ 1"
  by simp

from z show "Re z ∈ {-1/2..1/2}"
  by auto

have "sqrt (1 - Re z ^ 2) ^ 2 = 1 - Re z ^ 2"
  using <Re z ^ 2 ≤ 1> by simp
also have "... ≤ Im z ^ 2"
  using z by (simp add: norm_complex_def algebra_simps)
finally have "sqrt (1 - Re z ^ 2) ≤ Im z"
  by (rule power2_le_imp_le) (use z in auto)
thus "Im z - sqrt (1 - Re z ^ 2) ≥ 0"
  by simp
qed

lemma bij_betw_std_fund_region1': "bij_betw f S' T"
  using image_std_fund_region_subset' image_subset_std_fund_region'
  by (intro bij_betwI[of _ _ g]) (auto simp: std_fund_region_map_inverses)

lemma bij_betw_std_fund_region2': "bij_betw g T S'"
  using image_std_fund_region_subset' image_subset_std_fund_region'
  by (intro bij_betwI[of _ _ f]) (auto simp: std_fund_region_map_inverses)

lemma closure_std_fund_region: "closure  $\mathcal{R}_\Gamma = T$ "
proof -
  have homeo: "homeomorphism S  $\mathcal{R}_\Gamma$  f g"
    using image_std_fund_region_subset image_subset_std_fund_region
    by (intro homeomorphismI)
    (auto simp: g_def f_def case_prod_unfold intro!: continuous_intros)

  have "closure  $\mathcal{R}_\Gamma = \text{closure } (f \text{ ' } S)$ "
    using bij_betw_std_fund_region1' by (simp add: bij_betw_def)
  also have "... = f \text{ ' } \text{closure } S"
    using bij_betw_std_fund_region1' homeo
  proof (rule closure_bij_homeomorphic_image_eq)
    show "continuous_on UNIV f" "continuous_on UNIV g"
      by (auto simp: f_def g_def case_prod_unfold intro!: continuous_intros)
  qed (auto simp: std_fund_region_map_inverses)
  also have "closure S = {-1 / 2..1 / 2} × {0..}"
    by (simp add: S_def closure_Times)
  also have "... = S'"
    by (simp add: S'_def)
  also have "f \text{ ' } S' = T"
    using bij_betw_std_fund_region1' by (simp add: bij_betw_def)
  finally show ?thesis .
qed

```

```

lemma in_closure_std_fund_region_iff:
  "x ∈ closure  $\mathcal{R}_\Gamma \iff \text{norm } x \geq 1 \wedge \text{Re } x \in \{-1/2..1/2\} \wedge \text{Im } x \geq 0$ "
  by (simp add: closure_std_fund_region T_def)

lemma frontier_std_fund_region:
  "frontier  $\mathcal{R}_\Gamma =$ 
  {z. norm z ≥ 1 ∧ Im z > 0 ∧ |Re z| = 1 / 2} ∪
  {z. norm z = 1 ∧ Im z > 0 ∧ |Re z| ≤ 1 / 2}" (is "_ = ?rhs")
proof -
  have [simp]: " $x^2 \geq 1 \iff x \geq 1 \vee x \leq -1$ " for x :: real
  using abs_le_square_iff[of 1 x] by auto
  have "frontier  $\mathcal{R}_\Gamma = \text{closure } \mathcal{R}_\Gamma - \mathcal{R}_\Gamma$ "
  unfolding frontier_def by (subst interior_open) simp_all
  also have "... = ?rhs"
  unfolding closure_std_fund_region unfolding std_fund_region_def
  by (auto simp: cmod_def T_def)
  finally show ?thesis .
qed

lemma std_fund_region'_subset_closure: " $\mathcal{R}_\Gamma' \subseteq \text{closure } \mathcal{R}_\Gamma$ "
  by (auto simp: in_std_fund_region'_iff in_closure_std_fund_region_iff)

lemma std_fund_region'_superset: " $\mathcal{R}_\Gamma \subseteq \mathcal{R}_\Gamma'$ "
  by (auto simp: in_std_fund_region'_iff in_std_fund_region_iff)

lemma in_std_fund_region'_not_on_frontier_iff:
  assumes "z ∉ frontier  $\mathcal{R}_\Gamma$ "
  shows "z ∈  $\mathcal{R}_\Gamma' \iff z \in \mathcal{R}_\Gamma$ "
proof
  assume "z ∈  $\mathcal{R}_\Gamma'$ "
  hence "z ∈ closure  $\mathcal{R}_\Gamma$ "
  using std_fund_region'_subset_closure by blast
  thus "z ∈  $\mathcal{R}_\Gamma$ "
  using assms by (auto simp: frontier_def interior_open)
qed (use std_fund_region'_superset in auto)

lemma simply_connected_std_fund_region: "simply_connected  $\mathcal{R}_\Gamma$ "
proof (rule simply_connected_retraction_gen)
  show "simply_connected S"
  unfolding S_def by (intro convex_imp_simply_connected convex_Times)
auto
  show "continuous_on S f"
  unfolding f_def S_def case_prod_unfold by (intro continuous_intros)
  show "continuous_on  $\mathcal{R}_\Gamma$  g"
  unfolding g_def case_prod_unfold by (intro continuous_intros)
  show "f ' S =  $\mathcal{R}_\Gamma$ "
  using bij_betw_std_fund_region1 by (simp add: bij_betw_def)
  show "g ∈  $\mathcal{R}_\Gamma \rightarrow S$ "

```

```

    using bij_betw_std_fund_region2 bij_betw_imp_funcset by blast
  show "f (g x) = x" for x
    by (simp add: f_def g_def)
qed

lemma simply_connected_closure_std_fund_region: "simply_connected (closure
 $\mathcal{R}_\Gamma$ )"
proof (rule simply_connected_retraction_gen)
  show "simply_connected S'"
    unfolding S'_def by (intro convex_imp_simply_connected convex_Times)
  auto
  show "continuous_on S' f"
    unfolding f_def S'_def case_prod_unfold by (intro continuous_intros)
  show "continuous_on (closure  $\mathcal{R}_\Gamma$ ) g"
    unfolding g_def case_prod_unfold by (intro continuous_intros)
  show "f ' S' = closure  $\mathcal{R}_\Gamma$ "
    using bij_betw_std_fund_region1' by (simp add: bij_betw_def closure_std_fund_region)
  show "g  $\in$  closure  $\mathcal{R}_\Gamma \rightarrow S$ "
    using bij_betw_std_fund_region2' bij_betw_imp_funcset closure_std_fund_region
  by blast
  show "f (g x) = x" for x
    by (simp add: f_def g_def)
qed

lemma std_fund_region'_subset: " $\mathcal{R}_\Gamma' \subseteq \text{closure } \mathcal{R}_\Gamma$ "
  unfolding std_fund_region'_def closure_std_fund_region T_def unfolding
  std_fund_region_def
  by auto

lemma closure_std_fund_region_Im_pos: "closure  $\mathcal{R}_\Gamma \subseteq \{z. \text{Im } z > 0\}$ "
  unfolding closure_std_fund_region
  by (auto intro!: neq_le_trans simp: norm_complex_def field_simps power2_ge_1_iff
  T_def)

lemma closure_std_fund_region_Im_ge: "closure  $\mathcal{R}_\Gamma \subseteq \{z. \text{Im } z \geq \text{sqrt }
3 / 2\}$ "
proof
  fix z assume "z  $\in$  closure  $\mathcal{R}_\Gamma$ "
  hence *: "norm z  $\geq$  1" "|Re z|  $\leq$  1 / 2" "Im z  $\geq$  0"
    by (auto simp: closure_std_fund_region T_def)
  have "1  $\leq$  norm z  $^2$ "
    using * by simp
  also have "norm z  $^2 \leq$  (1 / 2)  $^2$  + Im z  $^2$ "
    unfolding cmod_power2 by (intro add_right_mono power2_mono) (use *
  in auto)
  finally have "Im z  $^2 \geq$  (sqrt 3 / 2)  $^2$ "
    by (simp add: power2_eq_square)
  hence "Im z  $\geq$  sqrt 3 / 2"
    by (subst (asm) abs_le_square_iff [symmetric]) (use * in auto)

```

```

    thus "z ∈ {z. Im z ≥ sqrt 3 / 2}"
      by simp
qed

lemma std_fund_region'_minus_std_fund_region:
  "ℛΓ' - ℛΓ =
    {z. norm z = 1 ∧ Im z > 0 ∧ Re z ∈ {-1/2..0}} ∪ {z. Re z = -1
/ 2 ∧ Im z ≥ sqrt 3 / 2}"
  (is "?lhs = ?rhs")
proof (intro equalityI subsetI)
  fix z assume z: "z ∈ ?lhs"
  from z have "Im z ≥ sqrt 3 / 2"
    using closure_std_fund_region_Im_ge std_fund_region'_subset by auto
  thus "z ∈ ?rhs" using z
    by (auto simp: std_fund_region'_def std_fund_region_def not_less)
next
  fix z assume z: "z ∈ ?rhs"
  have "sqrt 3 / 2 > 0"
    by simp
  have "Im z > 0"
    using z less_le_trans[OF <sqrt 3 / 2 > 0>, of "Im z"] by auto
  moreover have "norm z ≥ 1"
    using z
  proof
    assume "z ∈ {z. Re z = - 1 / 2 ∧ sqrt 3 / 2 ≤ Im z}"
    hence "norm z ^ 2 ≥ (-1/2) ^ 2 + (sqrt 3 / 2) ^ 2"
      unfolding cmod_power2 by (intro add_mono power2_mono) auto
    also have "(-1/2) ^ 2 + (sqrt 3 / 2) ^ 2 = 1"
      by (simp add: field_simps power2_eq_square)
    finally show "norm z ≥ 1"
      by (simp add: power2_nonneg_ge_1_iff)
  qed auto
  ultimately show "z ∈ ?lhs" using z
    by (auto simp: std_fund_region'_def std_fund_region_def)
qed

lemma closure_std_fund_region_minus_std_fund_region':
  "closure ℛΓ - ℛΓ' =
    {z. norm z = 1 ∧ Im z > 0 ∧ Re z ∈ {0<..1/2}} ∪ {z. Re z = 1 /
2 ∧ Im z ≥ sqrt 3 / 2}"
  (is "?lhs = ?rhs")
proof (intro equalityI subsetI)
  fix z assume z: "z ∈ closure ℛΓ - ℛΓ'"
  have "norm z ≥ 1"
    using z by (auto simp: closure_std_fund_region_in_std_fund_region'_iff
not_le T_def)
  from z have "Im z > 0" "Im z ≥ sqrt 3 / 2"
    using closure_std_fund_region_Im_pos closure_std_fund_region_Im_ge
  by blast+

```

```

    thus "z ∈ ?rhs" using z
      by (auto simp: closure_std_fund_region in_std_fund_region'_iff not_le
T_def)
next
  fix z assume "z ∈ ?rhs"
  thus "z ∈ ?lhs"
  proof
    assume "z ∈ {z. cmod z = 1 ∧ 0 < Im z ∧ Re z ∈ {0<..1 / 2}}"
    thus "z ∈ ?lhs"
      by (auto simp: closure_std_fund_region in_std_fund_region'_iff not_le
T_def)
  next
    assume z: "z ∈ {z. Re z = 1 / 2 ∧ sqrt 3 / 2 ≤ Im z}"
    have "0 < sqrt 3 / 2"
      by simp
    also have "... ≤ Im z"
      using z by auto
    finally have "Im z > 0" .
    have "norm z ^ 2 ≥ (1 / 2) ^ 2 + (sqrt 3 / 2) ^ 2"
      unfolding cmod_power2 by (intro add_mono power2_mono) (use z in
auto)
    also have "(1 / 2) ^ 2 + (sqrt 3 / 2) ^ 2 = 1"
      by (simp add: power2_eq_square)
    finally have "norm z ≥ 1"
      by (simp add: power2_nonneg_ge_1_iff)
    from this and <Im z > 0 and z show "z ∈ ?lhs"
      by (auto simp: closure_std_fund_region in_std_fund_region'_iff not_le
T_def)
  qed
qed

```

lemma cis_in_std_fund_region'_iff:

```

  assumes "φ ∈ {0..pi}"
  shows "cis φ ∈ ℛΓ' ↔ φ ∈ {pi/2..2*pi/3}"
proof
  assume φ: "φ ∈ {pi/2..2*pi/3}"
  have "φ > 0"
    by (rule less_le_trans[of _ "pi / 2"]) (use φ in auto)
  moreover have "φ < pi"
    by (rule le_less_trans[of _ "2 * pi / 3"]) (use φ in auto)
  ultimately have "sin φ > 0"
    by (intro sin_gt_zero) auto
  moreover have "cos φ ≥ cos (2 * pi / 3)"
    using φ by (intro cos_monotone_0_pi_le) auto
  moreover have "cos φ ≤ cos (pi / 2)"
    using φ by (intro cos_monotone_0_pi_le) auto
  ultimately show "cis φ ∈ ℛΓ'"
    by (auto simp: in_std_fund_region'_iff cos_120)
next

```

```

assume "cis  $\varphi \in \mathcal{R}_\Gamma$ "
hence *: "cos  $\varphi \geq \cos (2 * \text{pi} / 3)$ " "cos  $\varphi \leq \cos (\text{pi} / 2)$ "
  by (auto simp: in_std_fund_region'_iff cos_120)
have " $\varphi \leq 2 * \text{pi} / 3$ "
  using *(1) assms by (subst (asm) cos_mono_le_eq) auto
moreover have " $\varphi \geq \text{pi} / 2$ "
  using *(2) assms by (subst (asm) cos_mono_le_eq) auto
ultimately show " $\varphi \in \{\text{pi}/2..2*\text{pi}/3\}$ "
  by auto

```

qed

```

lemma imag_axis_in_std_fund_region'_iff: "y *R i  $\in \mathcal{R}_\Gamma$ '  $\longleftrightarrow y \geq 1$ "
  by (auto simp: in_std_fund_region'_iff)

```

```

lemma vertical_left_in_std_fund_region'_iff:
  "-1 / 2 + y *R i  $\in \mathcal{R}_\Gamma$ '  $\longleftrightarrow y \geq \text{sqrt } 3 / 2$ "

```

proof

```

assume y: "y  $\geq \text{sqrt } 3 / 2$ "
have "1 = (1 / 2) ^ 2 + (sqrt 3 / 2) ^ 2"
  by (simp add: power2_eq_square)
also have "...  $\leq (1 / 2) ^ 2 + y ^ 2$ "
  using y by (intro add_mono power2_mono) auto
also have "... = norm (y *R i - 1 / 2) ^ 2"
  unfolding cmod_power2 by simp
finally have "norm (y *R i - 1 / 2)  $\geq 1$ "
  by (simp add: power2_nonneg_ge_1_iff)
moreover have "y > 0"
  by (rule less_le_trans[OF _ y]) auto
ultimately show "-1 / 2 + y *R i  $\in \mathcal{R}_\Gamma$ '"
  using y by (auto simp: in_std_fund_region'_iff)

```

next

```

assume *: "-1 / 2 + y *R i  $\in \mathcal{R}_\Gamma$ '"
hence "y > 0"
  by (auto simp: in_std_fund_region'_iff)
from * have "1  $\leq \text{norm } (y *R i - 1 / 2)$ "
  by (auto simp: in_std_fund_region'_iff)
hence "1  $\leq \text{norm } (y *R i - 1 / 2) ^ 2$ "
  by (simp add: power2_nonneg_ge_1_iff)
also have "... = (1 / 2) ^ 2 + y ^ 2"
  unfolding cmod_power2 by simp
finally have "y ^ 2  $\geq (\text{sqrt } 3 / 2) ^ 2$ "
  by (simp add: algebra_simps power2_eq_square)
hence "y  $\geq \text{sqrt } 3 / 2$ "
  by (rule power2_le_imp_le) (use <y > 0 in auto)
thus "y  $\geq \text{sqrt } 3 / 2$ " using *
  by (auto simp: in_std_fund_region'_iff)

```

qed

```

lemma std_fund_region'_border_aux1:

```

```

"{z. norm z = 1 ∧ 0 < Im z ∧ Re z ∈ {-1/2..0}} = cis ' {pi / 2..2 /
3 * pi}"
proof safe
  fix z :: complex assume z: "norm z = 1" "Im z > 0" "Re z ∈ {-1/2..0}"
  show "z ∈ cis ' {pi/2..2/3*pi}"
  proof (rule rev_image_eqI)
    from z have [simp]: "z ≠ 0"
      by auto
    have [simp]: "Arg z ≥ 0"
      using z by (auto simp: Arg_less_0)
    have z_eq: "cis (Arg z) = z"
      using z by (auto simp: cis_Arg complex_sgn_def)
    thus "z = cis (Arg z)"
      by simp
    have "Re (cis (Arg z)) ≥ -1/2"
      using z by (subst z_eq) auto
    hence "cos (Arg z) ≥ cos (2/3*pi)"
      by (simp add: cos_120 cos_120')
    hence "Arg z ≤ 2 / 3 * pi"
      using Arg_le_pi by (subst (asm) cos_mono_le_eq) auto
    moreover have "Re (cis (Arg z)) ≤ 0"
      using z by (subst z_eq) auto
    hence "cos (Arg z) ≤ cos (pi / 2)"
      by simp
    hence "Arg z ≥ pi / 2"
      using Arg_le_pi by (subst (asm) cos_mono_le_eq) auto
    ultimately show "Arg z ∈ {pi/2..2/3*pi}"
      by simp
  qed
next
  fix t :: real assume t: "t ∈ {pi/2..2/3*pi}"
  have "t > 0"
    by (rule less_le_trans[of _ "pi/2"]) (use t in auto)
  have "t < pi"
    by (rule le_less_trans[of _ "2/3*pi"]) (use t in auto)
  have "sin t > 0"
    using <t > 0 <t < pi> by (intro sin_gt_zero) auto
  moreover have "cos t ≤ cos (pi / 2)"
    using t <t < pi> by (intro cos_monotone_0_pi_le) auto
  moreover have "cos t ≥ cos (2*pi/3)"
    using t by (intro cos_monotone_0_pi_le) auto
  ultimately show "norm (cis t) = 1" "Im (cis t) > 0" "Re (cis t) ∈ {-1/2..0}"
    by (auto simp: cos_120 cos_120')
qed

lemma std_fund_region'_border_aux2:
  "{z. Re z = - 1 / 2 ∧ sqrt 3 / 2 ≤ Im z} = (λx. - 1 / 2 + x *R i) '
  {sqrt 3 / 2..}"
  by (auto simp: complex_eq_iff)

```

```

lemma compact_std_fund_region:
  assumes "B > 1"
  shows "compact (closure  $\mathcal{R}_\Gamma \cap \{z. \text{Im } z \leq B\})"$ 
  unfolding compact_eq_bounded_closed
proof
  show "closed (closure  $\mathcal{R}_\Gamma \cap \{z. \text{Im } z \leq B\})"$ 
    by (intro closed_Int closed_halfspace_Im_le) auto
next
  show "bounded (closure  $\mathcal{R}_\Gamma \cap \{z. \text{Im } z \leq B\})"$ 
  proof -
    have "closure  $\mathcal{R}_\Gamma \cap \{z. \text{Im } z \leq B\} \subseteq \text{cbox } (-1/2) (1/2 + i * B)"$ 
      by (auto simp: in_closure_std_fund_region_iff in_cbox_complex_iff)
    moreover have "bounded (cbox  $(-1/2) (1/2 + i * B)$ )"
      by simp
    ultimately show ?thesis
      using bounded_subset by blast
  qed
qed
end

```

5.3 Proving that the standard region is fundamental

```

lemma norm_open_segment_less:
  fixes x y z :: "'a :: euclidean_space"
  assumes "norm x ≤ norm y" "z ∈ open_segment x y"
  shows "norm z < norm y"
  using assms
  by (metis (no_types, opaque_lifting) diff_zero dist_decreases_open_segment
      dist_norm norm_minus_commute order_less_le_trans)

```

Lemma 1

```

lemma (in complex_lattice) std_fund_region_fundamental_lemma1:
  obtains  $\omega_1' \omega_2' :: \text{complex}$  and  $a b c d :: \text{int}$ 
  where " $|a * d - b * c| = 1$ "
    " $\omega_2' = \text{of\_int } a * \omega_2 + \text{of\_int } b * \omega_1$ "
    " $\omega_1' = \text{of\_int } c * \omega_2 + \text{of\_int } d * \omega_1$ "
    " $\text{Im } (\omega_2' / \omega_1') \neq 0$ "
    " $\text{norm } \omega_1' \leq \text{norm } \omega_2'$ " " $\text{norm } \omega_2' \leq \text{norm } (\omega_1' + \omega_2')$ " " $\text{norm } \omega_2' \leq \text{norm } (\omega_1' - \omega_2')$ "
proof -
  have " $\Lambda^* \subseteq \Lambda$ " " $\Lambda^* \neq \{\}$ "
    by auto
  then obtain  $\omega_1'$  where  $\omega_1': \omega_1' \in \Lambda^*$  " $\bigwedge y. y \in \Lambda^* \implies \text{norm } \omega_1' \leq \text{norm } y$ "
    using shortest_lattice_vector_exists by blast
  define X where " $X = \{y. y \in \Lambda^* \wedge y / \omega_1' \notin \mathbb{R}\}$ "

```

```

have "X ⊆ Λ"
  by (auto simp: X_def lattice0_def)
moreover have "X ≠ {}"
  using noncollinear_lattice_point_exists[of ω1'] ω1'(1) unfolding
X_def by force
ultimately obtain ω2' where ω2': "ω2' ∈ X" "∧z. z ∈ X ⇒ norm ω2'
≤ norm z"
  using shortest_lattice_vector_exists by blast

have [simp]: "ω1' ≠ 0" "ω2' ≠ 0"
  using ω1' ω2' by (auto simp: lattice0_def X_def)
have noncollinear: "ω2' / ω1' ∉ ℝ"
  using ω2' by (auto simp: X_def)
hence fundpair': "fundpair (ω1', ω2')"
  unfolding fundpair_def prod.case by simp
have Im_nz: "Im (ω2' / ω1') ≠ 0"
  using noncollinear by (auto simp: complex_is_Real_iff)

have "norm ω1' ≤ norm ω2'"
  by (intro ω1') (use ω2' in <auto simp: X_def>)

have triangle: "z ∉ Λ" if z: "z ∈ convex hull {0, ω1', ω2'}" "z ∉
{0, ω1', ω2'}" for z
  proof
    assume "z ∈ Λ"
    hence "z ∈ Λ*"
      using z by (auto simp: lattice0_def)
    from that obtain a b where ab: "a ≥ 0" "b ≥ 0" "a + b ≤ 1" "z
= a *ℝ ω1' + b *ℝ ω2'"
      unfolding convex_hull_3_alt by (auto simp: scaleR_conv_of_real)

    have "norm z ≤ norm (a *ℝ ω1') + norm (b *ℝ ω2')"
      unfolding ab using norm_triangle_ineq by blast
    also have "... = a * norm ω1' + b * norm ω2'"
      using ab by simp
    finally have norm_z_le: "norm z ≤ a * norm ω1' + b * norm ω2'" .

    also have "... ≤ a * norm ω2' + b * norm ω2'"
      using ab <norm ω1' ≤ norm ω2'> by (intro add_mono mult_left_mono)
  auto
  also have "... = (a + b) * norm ω2'"
    by (simp add: algebra_simps)
  finally have norm_z_le': "norm z ≤ (a + b) * norm ω2'" .

have "z / ω1' ∉ ℝ"
  proof
    assume real: "z / ω1' ∈ ℝ"
    show False
      proof (cases "b = 0")

```

```

case False
hence " $\omega_2' / \omega_1' = (z / \omega_1' - \text{of\_real } a) / \text{of\_real } b$ "
  by (simp add: ab field_simps scaleR_conv_of_real)
also have "...  $\in \mathbb{R}$ "
  using real by (auto intro: Reals_divide Reals_diff)
finally show False
  using noncollinear by contradiction
next
case True
hence " $z = a *_{\mathbb{R}} \omega_1'$ "
  using ab by simp
from this and z have " $a \neq 1$ "
  by auto
hence " $a < 1$ "
  using ab by simp
have " $\text{norm } z = a * \text{norm } \omega_1'$ "
  using <z = a *R  $\omega_1'$ > <a  $\geq 0$ > by simp
also have "...  $< 1 * \text{norm } \omega_1'$ "
  using <a < 1> by (intro mult_strict_right_mono) auto
finally have " $\text{norm } z < \text{norm } \omega_1'$ "
  by simp
moreover have " $\text{norm } z \geq \text{norm } \omega_1'$ "
  by (intro  $\omega_1'$ ) (use z <z  $\in \Lambda^*$ > in auto)
ultimately show False
  by simp
qed
qed
hence " $z \in X$ "
  using <z  $\in \Lambda^*$ > by (auto simp: X_def)
hence " $\text{norm } z \geq \text{norm } \omega_2'$ "
  by (intro  $\omega_2'$ )

moreover have " $\text{norm } z \leq \text{norm } \omega_2'$ "
proof -
  have " $\text{norm } z \leq (a + b) * \text{norm } \omega_2'$ "
    by (rule norm_z_le')
  also have "...  $\leq 1 * \text{norm } \omega_2'$ "
    using ab by (intro mult_right_mono) auto
  finally show " $\text{norm } z \leq \text{norm } \omega_2'$ "
    by simp
qed

ultimately have norm_z: " $\text{norm } z = \text{norm } \omega_2'$ "
  by linarith

have " $\neg(a + b < 1)$ "
proof
  assume *: " $a + b < 1$ "
  have " $\text{norm } z \leq (a + b) * \text{norm } \omega_2'$ "

```

```

    by (rule norm_z_le')
  also have "... < 1 * norm ω2'"
    by (intro mult_strict_right_mono *) auto
  finally show False
    using norm_z by simp
qed
with ab have b_eq: "b = 1 - a"
  by linarith

have "norm z < norm ω2'"
proof (rule norm_open_segment_less)
  have "a ≠ 0" "a ≠ 1"
    using z ab by (auto simp: b_eq)
  hence "∃u. u > 0 ∧ u < 1 ∧ z = (1 - u) *R ω1' + u *R ω2'"
    using ab by (intro exI[of _ b]) (auto simp: b_eq)
  thus "z ∈ open_segment ω1' ω2'"
    using z ab noncollinear unfolding in_segment by auto
next
  show "norm ω1' ≤ norm ω2'"
    by fact
qed
with norm_z show False
  by simp
qed
hence "convex hull {0, ω1', ω2'} ∩ Λ ⊆ {0, ω1', ω2'}"
  by blast
moreover have "{0, ω1', ω2'} ⊆ convex hull {0, ω1', ω2'} ∩ Λ"
  using ω1' ω2' by (auto intro: hull_inc simp: X_def)
ultimately have "convex hull {0, ω1', ω2'} ∩ Λ = {0, ω1', ω2'}"
  by blast

hence "equiv_fundpair (ω1, ω2) (ω1', ω2')"
  using fundpair' ω1' ω2' by (subst equiv_fundpair_iff_triangle) (auto
simp: X_def)
then obtain a b c d :: int where
  det: "|a * d - b * c| = 1" and
  abcd: "ω2' = of_int a * ω2 + of_int b * ω1" "ω1' = of_int c * ω2
+ of_int d * ω1"
  using fundpair fundpair' by (subst (asm) equiv_fundpair_iff) auto

have *: "norm (ω1' + of_int n * ω2') ≥ norm ω2'" if n: "n ≠ 0" for
n
proof (rule ω2')
  define z where "z = ω1' + of_int n * ω2'"
  have "z ∈ Λ"
    unfolding z_def using ω1'(1) ω2'(1) by (auto intro!: lattice_intros
simp: X_def)
  moreover have "z / ω1' ∉ ℝ"
  proof

```

```

    assume "z / ω1' ∈ ℝ"
    hence "(z / ω1' - 1) / of_int n ∈ ℝ"
      by auto
    also have "(z / ω1' - 1) / of_int n = ω2' / ω1'"
      using n by (simp add: field_simps z_def)
    finally show False
      using noncollinear by contradiction
  qed
  moreover from this have "z ≠ 0"
    by auto
  ultimately show "z ∈ X"
    by (auto simp: X_def lattice0_def)
  qed
  have norms: "norm (ω1' + ω2') ≥ norm ω2'" "norm (ω1' - ω2') ≥ norm
ω2'"
    using *[of 1] and *[of "-1"] by simp_all

  show ?thesis
    using det norms abcd noncollinear <norm ω1' ≤ norm ω2'>
    by (intro that[of a d b c ω2' ω1']) (simp_all add: complex_is_Real_iff)
  qed

lemma (in complex_lattice) std_fund_region_fundamental_lemma2:
  obtains ω1' ω2' :: complex and a b c d :: int
  where "a * d - b * c = 1"
        "ω2' = of_int a * ω2 + of_int b * ω1"
        "ω1' = of_int c * ω2 + of_int d * ω1"
        "Im (ω2' / ω1') ≠ 0"
        "norm ω1' ≤ norm ω2'" "norm ω2' ≤ norm (ω1' + ω2')" "norm ω2'
≤ norm (ω1' - ω2')"
  proof -
    obtain ω1' ω2' a b c d
      where abcd: "|a * d - b * c| = 1"
        and eq: "ω2' = of_int a * ω2 + of_int b * ω1" "ω1' = of_int c
* ω2 + of_int d * ω1"
        and nz: "Im (ω2' / ω1') ≠ 0"
        and norms: "norm ω1' ≤ norm ω2'" "norm ω2' ≤ norm (ω1' + ω2')"
        "norm ω2' ≤ norm (ω1' - ω2'"
      using std_fund_region_fundamental_lemma1 .

    show ?thesis
  proof (cases "a * d - b * c = 1")
    case True
      thus ?thesis
        by (intro that[of a d b c ω2' ω1'] eq nz norms)
    next
      case False
      show ?thesis
        proof (intro that[of a "-d" b "-c" ω2' "-ω1'])

```

```

    from False have "a * d - b * c = -1"
      using abcd by linarith
    thus "a * (-d) - b * (-c) = 1"
      by simp
  next
    show "ω2' = of_int a * ω2 + of_int b * ω1"
      "-ω1' = of_int (-c) * ω2 + of_int (-d) * ω1"
      using eq by (simp_all add: algebra_simps)
    qed (use norms nz in <auto simp: norm_minus_commute add.commute>)
  qed
qed

```

Theorem 2.2

lemma std_fund_region_fundamental_aux1:

```

  assumes "Im τ' > 0"
  obtains τ where "Im τ > 0" "τ ~Γ τ'" "norm τ ≥ 1" "norm (τ + 1) ≥
norm τ" "norm (τ - 1) ≥ norm τ"
proof -
  interpret std_complex_lattice τ'
  using assms by unfold_locales (auto simp: complex_is_Real_iff)
  obtain ω1 ω2 a b c d
  where abcd: "a * d - b * c = 1"
    and eq: "ω2 = of_int a * τ' + of_int b * 1" "ω1 = of_int c * τ'
+ of_int d * 1"
    and nz: "Im (ω2 / ω1) ≠ 0"
    and norms: "norm ω1 ≤ norm ω2" "norm ω2 ≤ norm (ω1 + ω2)" "norm
ω2 ≤ norm (ω1 - ω2)"
  using std_fund_region_fundamental_lemma2 .
  from nz have [simp]: "ω1 ≠ 0" "ω2 ≠ 0"
  by auto

  interpret unimodular_moebius_transform a b c d
  by unfold_locales fact

  define τ where "τ = ω2 / ω1"
  have τ_eq: "τ = φ τ'"
  by (simp add: moebius_def τ_def eq add_ac φ_def)

  show ?thesis
  proof (rule that[of τ])
    show "Im τ > 0"
      using assms τ_eq by (simp add: Im_transform_pos)
  next
    show "norm τ ≥ 1" "norm (τ + 1) ≥ norm τ" "norm (τ - 1) ≥ norm
τ"
      using norms by (simp_all add: τ_def norm_divide field_simps norm_minus_commute)
  next
    have "τ = apply_modgrp as_modgrp τ'"
      using τ_eq by (simp add: φ_as_modgrp)

```

```

    thus " $\tau \sim_{\Gamma} \tau'$ " using <Im  $\tau' > 0$ >
      by auto
  qed
qed

lemma std_fund_region_fundamental_aux2:
  assumes "norm (z + 1)  $\geq$  norm z" "norm (z - 1)  $\geq$  norm z"
  shows "Re z  $\in$   $\{-1/2..1/2\}$ "
proof -
  have "0  $\leq$  norm (z + 1)  $^2$  - norm z  $^2$ "
    using assms by simp
  also have "... = (Re z + 1) $^2$  - (Re z) $^2$ "
    unfolding norm_complex_def by simp
  also have "... = 1 + 2 * Re z"
    by (simp add: algebra_simps power2_eq_square)
  finally have "Re z  $\geq$  -1/2"
    by simp

  have "0  $\leq$  norm (z - 1)  $^2$  - norm z  $^2$ "
    using assms by simp
  also have "... = (Re z - 1) $^2$  - (Re z) $^2$ "
    unfolding norm_complex_def by simp
  also have "... = 1 - 2 * Re z"
    by (simp add: algebra_simps power2_eq_square)
  finally have "Re z  $\leq$  1/2"
    by simp

  with <Re z  $\geq$  -1/2> show ?thesis
    by simp
qed

lemma std_fund_region_fundamental_aux3:
  fixes x y :: complex
  assumes xy: "x  $\in \mathcal{R}_{\Gamma}$ " "y  $\in \mathcal{R}_{\Gamma}$ "
  assumes f: "y = apply_modgrp f x"
  defines "c  $\equiv$  modgrp_c f"
  defines "d  $\equiv$  modgrp_d f"
  assumes c: "c  $\neq$  0"
  shows "Im y < Im x"
proof -
  have ineq1: "norm (c * x + d)  $^2$  > c  $^2$  - |c * d| + d  $^2$ "
  proof -
    have "of_int |c| * 1 < of_int |c| * norm x"
      using xy c by (intro mult_strict_left_mono) (auto simp: std_fund_region_def)
    hence "of_int c  $^2$  < (of_int c * norm x)  $^2$ "
      by (intro power2_strict_mono) auto
    also have "... - |c * d| * 1 + d  $^2$   $\leq$ 
      (of_int c * norm x)  $^2$  - |c * d| * (2 * |Re x|) + d  $^2$ "
      using xy unfolding of_int_add of_int_mult of_int_power of_int_diff

```

```

    by (intro add_mono diff_mono mult_left_mono) (auto simp: std_fund_region_def)
  also have "... = c ^ 2 * norm x ^ 2 - |2 * c * d * Re x| + d ^ 2"
    by (simp add: power_mult_distrib abs_mult)
  also have "... ≤ c ^ 2 * norm x ^ 2 + 2 * c * d * Re x + d ^ 2"
    by linarith
  also have "... = norm (c * x + d) ^ 2"
    unfolding cmod_power2 by (simp add: algebra_simps power2_eq_square)
  finally show "norm (c * x + d) ^ 2 > c ^ 2 - |c * d| + d ^ 2"
    by simp
qed

have "Im y = Im x / norm (c * x + d) ^ 2"
  using f by (simp add: modgrp.Im_transform c_def d_def)

also have "norm (c * x + d) ^ 2 > 1"
proof (cases "d = 0")
  case [simp]: True
  have "0 < c ^ 2"
    using c by simp
  hence "1 ≤ real_of_int (c ^ 2) * 1"
    by linarith
  also have "... < of_int (c ^ 2) * norm x ^ 2"
    using xy c by (intro mult_strict_left_mono) (auto simp: std_fund_region_def)
  also have "... = norm (c * x + d) ^ 2"
    by (simp add: norm_mult power_mult_distrib)
  finally show ?thesis .
next
  case False
  have "0 < |c * d|"
    using c False by auto
  hence "1 ≤ |c * d|"
    by linarith
  also have "... ≤ |c * d| + (|c| - |d|) ^ 2"
    by simp
  also have "... = c ^ 2 - |c * d| + d ^ 2"
    by (simp add: algebra_simps power2_eq_square abs_mult)
  finally have "1 ≤ real_of_int (c ^ 2 - |c * d| + d ^ 2)"
    by linarith
  also have "... < norm (c * x + d) ^ 2"
    using ineq1 False by simp
  finally show ?thesis .
qed

hence "Im x / norm (c * x + d) ^ 2 < Im x / 1"
  using xy by (intro divide_strict_left_mono) (auto simp: std_fund_region_def)
finally show ?thesis
  by simp
qed

```

```

lemma std_fund_region_fundamental_aux4:
  fixes x y :: complex
  assumes xy: "x ∈  $\mathcal{R}_\Gamma$ " "y ∈  $\mathcal{R}_\Gamma$ "
  assumes f: "y = apply_modgrp f x"
  shows "|f| = 1"
proof -
  define a where "a = modgrp_a f"
  define b where "b = modgrp_b f"
  define c where "c = modgrp_c f"
  define d where "d = modgrp_d f"

  have c: "c = 0"
  proof (rule ccontr)
    assume c: "c ≠ 0"
    have "Im y < Im x" using xy f c
      by (intro std_fund_region_fundamental_aux3[where f = f]) (auto simp: c_def)
    moreover have "Im y > Im x"
    proof (rule std_fund_region_fundamental_aux3[where f = "inverse f"])
      have "Im x > 0"
        using xy by (auto simp: std_fund_region_def)
      hence "x ≠ pole_modgrp f"
        using pole_modgrp_in_Reals[of f, where ?'a = complex]
        by (auto simp: complex_is_Real_iff)
      with f have "apply_modgrp (inverse f) y = x"
        by (intro apply_modgrp_inverse_eqI) auto
      thus "x = apply_modgrp (inverse f) y" ..
    next
      have "is_singular_modgrp f"
        using c by (simp add: is_singular_modgrp_altdef c_def)
      hence "is_singular_modgrp (inverse f)"
        by simp
      thus "modgrp_c (inverse f) ≠ 0"
        unfolding is_singular_modgrp_altdef by simp
    qed (use xy c in <auto simp: c_def>)
    ultimately show False
      by simp
  qed

  define n where "n = sgn a * b"
  from c have "¬is_singular_modgrp f"
    by (auto simp: is_singular_modgrp_altdef c_def)
  hence "|f| = |shift_modgrp n|"
    using not_is_singular_modgrpD[of f] by (simp add: n_def a_def b_def)
  hence f_eq: "f = shift_modgrp n ∨ f = -shift_modgrp n"
    by (elim abs_modgrp_eqE) auto
  from xy f f_eq have "n = 0"
    by (auto simp: std_fund_region_def)
  thus "|f| = 1"

```

```

    using f_eq by auto
qed

```

Theorem 2.3

```

interpretation std_fund_region: fundamental_region UNIV std_fund_region
proof

```

```

  show "std_fund_region  $\subseteq$  {z. Im z > 0}"
    using Im_std_fund_region by blast
next
  fix x y :: complex
  assume xy: "x  $\in$   $\mathcal{R}_\Gamma$ " "y  $\in$   $\mathcal{R}_\Gamma$ " "x  $\sim_\Gamma$  y"
  then obtain f where f: "y = apply_modgrp f x"
    by (auto simp: modular_group.rel_def)
  with xy have "|f| = 1"
    using std_fund_region_fundamental_aux4 by blast
  with f xy show "x = y"
    by (auto simp: abs_modgrp_eq_1_iff)
next
  fix x :: complex
  assume x: "Im x > 0"
  obtain y where y: "Im y > 0" "y  $\sim_\Gamma$  x"
    "norm y  $\geq$  1" "norm (y + 1)  $\geq$  norm y" "norm (y - 1)  $\geq$  norm y"
    using std_fund_region_fundamental_aux1[OF x] by blast
  from y have "Re y  $\in$  {-1/2..1/2}"
    by (intro std_fund_region_fundamental_aux2)
  with y show " $\exists y \in \text{closure std\_fund\_region. } x \sim_\Gamma y$ "
    using x unfolding closure_std_fund_region by (auto simp: modular_group.rel_commutes)
qed auto

```

Every point in the upper half plane has a canonical representative w.r.t. the full modular group in the standard fundamental region.

```

lemma canonical_point_in_std_fund_region':

```

```

  assumes "Im z > 0"
  obtains z' where "z'  $\in$   $\mathcal{R}_\Gamma$ " "z  $\sim_\Gamma$  z'"
proof -
  obtain z' where z': "z'  $\in$  closure  $\mathcal{R}_\Gamma$ " "z  $\sim_\Gamma$  z'"
    using std_fund_region.equiv_in_closure[of z] assms by blast
  show ?thesis
  proof (cases "z'  $\in$   $\mathcal{R}_\Gamma$ ")
    case True
    thus ?thesis
      by (intro that[of z']) (use z' in auto)
  next
    case False
    with z' have "z'  $\in$  closure  $\mathcal{R}_\Gamma - \mathcal{R}_\Gamma$ "
      by blast
    hence "norm z' = 1  $\wedge$  Im z' > 0  $\wedge$  Re z'  $\in$  {0<..1/2}  $\vee$  Re z' = 1 / 2  $\wedge$  Im z'  $\geq$  sqrt 3 / 2"

```

```

    by (subst (asm) closure_std_fund_region_minus_std_fund_region')
simp_all
  thus ?thesis
  proof
    assume *: "norm z' = 1 ∧ Im z' > 0 ∧ Re z' ∈ {0 <.. 1/2}"
    define z'' where "z'' = apply_modgrp S_modgrp z'"
    have "z ~Γ z''"
      unfolding z''_def using z'(2) by blast
    moreover have "z'' ∈  $\mathcal{R}_\Gamma$ "
      using * by (auto simp: z''_def norm_divide divide_conv_cnj in_std_fund_region'_iff)
    ultimately show ?thesis
      by (intro that[of z''])
  next
    assume *: "Re z' = 1 / 2 ∧ Im z' ≥ sqrt 3 / 2"
    have [simp]: "Re z' = 1 / 2"
      using * by blast
    define z'' where "z'' = apply_modgrp (shift_modgrp (-1)) z'"
    have "z ~Γ z''"
      unfolding z''_def using z'(2) by blast
    moreover have "norm z'' = norm z'"
      by (auto simp: z''_def cmod_def)
    hence "z'' ∈  $\mathcal{R}_\Gamma$ " using z'
      by (auto simp: z''_def in_std_fund_region'_iff in_closure_std_fund_region_iff
cmod_def)
    ultimately show ?thesis
      by (intro that[of z''])
  qed
qed
qed

theorem std_fund_region_no_fixed_point:
  assumes "z ∈  $\mathcal{R}_\Gamma$ "
  assumes "apply_modgrp f z = z"
  shows "|f| = 1"
  using std_fund_region_fundamental_aux4[of z "apply_modgrp f z" f] assms
  by auto

lemma std_fund_region_no_fixed_point':
  assumes "z ∈  $\mathcal{R}_\Gamma$ "
  assumes "apply_modgrp f z = apply_modgrp g z"
  shows "|f| = |g|"
proof -
  have z: "Im z > 0"
    using assms by (auto simp: in_std_fund_region_iff)
  have "apply_modgrp (inverse f) (apply_modgrp g z) = apply_modgrp (inverse
f) (apply_modgrp f z)"
    by (simp only: assms(2))
  also have "... = z"
    using z by (intro apply_modgrp_inverse_eqI) auto

```

```

    also have "apply_modgrp (inverse f) (apply_modgrp g z) = apply_modgrp
(inverse f * g) z"
      by (rule apply_modgrp_mult [symmetric]) (use z in auto)
    finally have "|inverse f * g| = 1"
      using assms by (intro std_fund_region_no_fixed_point) auto
    thus ?thesis
      by (metis abs_uminus_modgrp minus_minus_modgrp modgrp.inverse_inverse
modgrp.inverse_unique
times_modgrp_uminus_right abs_modgrp_eq_1_iff)
qed

```

The image of the fundamental region under a unimodular transformation is again a fundamental region.

```

locale std_fund_region_image =
  fixes f :: modgrp and R :: "complex set"
  defines "R ≡ apply_modgrp f '  $\mathcal{R}_\Gamma$ "
begin

lemma R_altdef: "R = {z. Im z > 0} ∩ apply_modgrp (inverse f) -'  $\mathcal{R}_\Gamma$ "
  unfolding R_def
proof safe
  fix z assume z: "z ∈  $\mathcal{R}_\Gamma$ "
  thus "Im (apply_modgrp f z) > 0"
    by (auto simp: Im_std_fund_region)
  have "apply_modgrp (inverse f) (apply_modgrp f z) ∈  $\mathcal{R}_\Gamma$ "
    by (subst apply_modgrp_mult [symmetric]) (use z in auto)
  thus "apply_modgrp f z ∈ apply_modgrp (inverse f) -'  $\mathcal{R}_\Gamma$ "
    by (auto simp flip: apply_modgrp_mult)
next
  fix z assume z: "apply_modgrp (inverse f) z ∈  $\mathcal{R}_\Gamma$ " "Im z > 0"
  have "z = apply_modgrp f (apply_modgrp (inverse f) z)"
    by (subst apply_modgrp_mult [symmetric]) (use z(2) in auto)
  with z show "z ∈ apply_modgrp f '  $\mathcal{R}_\Gamma$ "
    by blast
qed

```

```

lemma R_altdef': "R = apply_modgrp (inverse f) -'  $\mathcal{R}_\Gamma$ "
  unfolding R_altdef by (auto simp: in_std_fund_region_iff)

```

```

sublocale fundamental_region UNIV R
proof
  show "open R"
    unfolding R_altdef
    by (intro continuous_open_preimage continuous_intros) (auto simp:
open_halfspace_Im_gt )
  show "R ⊆ {z. 0 < Im z}"
    unfolding R_altdef by auto
  show "x = y" if "x ∈ R" "y ∈ R" "x  $\sim_\Gamma$  y" for x y
    using that unfolding R_def by (auto dest: std_fund_region.unique)

```

```

show "∃y∈closure R. x ~Γ y" if "Im x > 0" for x
proof -
  define x' where "x' = apply_modgrp (inverse f) x"
  have x': "Im x' > 0"
    using that by (simp add: x'_def)
  then obtain y where y: "y ∈ closure  $\mathcal{R}_\Gamma$ " "x' ~Γ y"
    using std_fund_region.equiv_in_closure[of x'] by blast
  define y' where "y' = apply_modgrp f y"
  have "y islimpt  $\mathcal{R}_\Gamma$ "
    using y by (meson islimpt_closure_open limpt_of_closure open_std_fund_region)
  then obtain h :: "nat ⇒ complex" where h: " $\bigwedge n. h\ n \in \mathcal{R}_\Gamma - \{y\}$ "
    "h → y"
    unfolding islimpt_sequential by blast
  have "(apply_modgrp f ∘ h) n ∈ R - {y'}" for n
  proof -
    have Ims: "Im y > 0" "Im (h n) > 0"
      using y h(1)[of n] by (auto simp: in_std_fund_region_iff)
    have "apply_modgrp f (h n) ∈ R" "h n ≠ y"
      using h(1)[of n] by (auto simp: R_def)
    moreover have "apply_modgrp f (h n) ≠ y'"
      unfolding y'_def using y <h n ≠ y> Ims by (subst apply_modgrp_eq_iff)
  auto
    ultimately show ?thesis
      by auto
  qed
  moreover have "(apply_modgrp f ∘ h) → y'"
    unfolding y'_def using y by (auto intro!: tendsto_compose_at[OF
h(2)] tendsto_eq_intros)+
  ultimately have "y' islimpt R"
    unfolding islimpt_sequential by blast
  hence "y' ∈ closure R"
    by (simp add: closure_def)

  moreover have "x ~Γ y'"
    using x' that y unfolding y'_def x'_def
    by (auto simp: modular_group.rel_commutes)
  ultimately show ?thesis
    by blast
  qed
qed
end

```

5.4 The corner point of the standard fundamental region

The point $\rho = \exp(2/3\pi) = -\frac{1}{2} + \frac{\sqrt{3}}{2}i$ is the left corner of the standard fundamental region, and its reflection on the imaginary axis (which is the same as its image under $z \mapsto -1/z$) forms the right corner.

```

definition modfun_rho (" $\rho$ ") where
  " $\rho = \text{cis } (2 / 3 * \text{pi})$ "

lemma modfun_rho_altdef: " $\rho = -1 / 2 + \text{sqrt } 3 / 2 * i$ "
  by (simp add: complex_eq_iff modfun_rho_def Re_exp Im_exp sin_120 cos_120)

lemma Re_modfun_rho [simp]: " $\text{Re } \rho = -1 / 2$ "
  and Im_modfun_rho [simp]: " $\text{Im } \rho = \text{sqrt } 3 / 2$ "
  by (simp_all add: modfun_rho_altdef)

lemma norm_modfun_rho [simp]: " $\text{norm } \rho = 1$ "
  by (simp add: modfun_rho_def)

lemma modfun_rho_plus_1_eq: " $\rho + 1 = \text{exp } (\text{pi} / 3 * i)$ "
  by (simp add: modfun_rho_altdef complex_eq_iff Re_exp Im_exp sin_60
  cos_60)

lemma norm_modfun_rho_plus_1 [simp]: " $\text{norm } (\rho + 1) = 1$ "
  by (simp add: modfun_rho_plus_1_eq)

lemma cnj_modfun_rho: " $\text{cnj } \rho = -\rho - 1$ "
  and cnj_modfun_rho_plus1: " $\text{cnj } (\rho + 1) = -\rho$ "
  by (auto simp: complex_eq_iff)

lemma modfun_rho_cube: " $\rho ^ 3 = 1$ "
  by (simp add: modfun_rho_def Complex.DeMoivre)

lemma modfun_rho_power_mod3_reduce: " $\rho ^ n = \rho ^ (n \text{ mod } 3)$ "
proof -
  have " $\rho ^ n = \rho ^ (3 * (n \text{ div } 3) + (n \text{ mod } 3))$ "
    by simp
  also have "... =  $(\rho ^ 3) ^ (n \text{ div } 3) * \rho ^ (n \text{ mod } 3)$ "
    by (subst power_add) (simp add: power_mult)
  also have "... =  $\rho ^ (n \text{ mod } 3)$ "
    by (simp add: modfun_rho_cube)
  finally show ?thesis .
qed

lemma modfun_rho_power_mod3_reduce': " $n \geq 3 \implies \rho ^ n = \rho ^ (n \text{ mod } 3)$ "
  by (rule modfun_rho_power_mod3_reduce)

lemmas [simp] = modfun_rho_power_mod3_reduce' [of "numeral num" for num]

lemma modfun_rho_square: " $\rho ^ 2 = -\rho - 1$ "
  by (simp add: modfun_rho_altdef power2_eq_square field_simps flip: of_real_mult)

lemma modfun_rho_not_real [simp]: " $\rho \notin \mathbb{R}$ "
  by (simp add: modfun_rho_altdef complex_is_Real_iff)

```

```

lemma modfun_rho_nonzero [simp]: " $\rho \neq 0$ "
  by (simp add: modfun_rho_def)

lemma modfun_rho_not_one [simp]: " $\rho \neq 1$ "
  by (simp add: complex_eq_iff modfun_rho_altdef)

lemma i_neq_modfun_rho [simp]: " $i \neq \rho$ "
  and i_neq_modfun_rho_plus1 [simp]: " $i \neq \rho + 1$ "
  and modfun_rho_neg_i [simp]: " $\rho \neq i$ "
  and modfun_rho_plus1_neg_i [simp]: " $\rho + 1 \neq i$ "
  by (auto simp: complex_eq_iff)

lemma i_in_closure_std_fund_region [intro, simp]: " $i \in \text{closure } \mathcal{R}_\Gamma$ "
  and i_in_std_fund_region' [intro, simp]: " $i \in \mathcal{R}_\Gamma$ "
  and modfun_rho_in_closure_std_fund_region [intro, simp]: " $\rho \in \text{closure } \mathcal{R}_\Gamma$ "
  and modfun_rho_in_std_fund_region' [intro, simp]: " $\rho \in \mathcal{R}_\Gamma$ "
  and modfun_rho_plus1_notin_closure_std_fund_region [intro, simp]: " $\rho + 1 \in \text{closure } \mathcal{R}_\Gamma$ "
  and modfun_rho_plus1_notin_std_fund_region' [intro, simp]: " $\rho + 1 \notin \mathcal{R}_\Gamma$ "
  by (simp_all add: closure_std_fund_region std_fund_region'_def in_std_fund_region_iff)

lemma modfun_rho_power_eq_1_iff: " $\rho^n = 1 \iff 3 \text{ dvd } n$ "
proof -
  have " $\rho^n = 1 \iff (\exists k. \text{real } n = 3 * \text{real\_of\_int } k)$ "
    by (simp add: modfun_rho_def Complex.DeMoivre cis_eq_1_iff)
  also have " $(\lambda k. \text{real } n = 3 * \text{real\_of\_int } k) = (\lambda k. \text{int } n = 3 * k)$ "
    by (rule ext) linarith
  also have " $(\exists k. \text{int } n = 3 * k) \iff 3 \text{ dvd } n$ "
    by presburger
  finally show ?thesis .
qed

```

5.5 The elliptic points of the modular group

We will now explicitly compute the stabilisers of i and ρ and, from that, see that they are elliptic points of order 2 and 3, respectively.

We will later see that these are the only elliptic points of the modular group (up to equivalence w.r.t. unimodular transformations, of course).

```

lemma abs_le_1_iff_int: " $\text{abs } n \leq 1 \iff n = 0 \vee n = 1 \vee n = (-1::\text{int})$ "
  by linarith

```

```

lemma abs_eq_1_iff: " $\text{abs } (x :: 'a :: \text{linordered\_idom}) = 1 \iff x = 1 \vee x = -1$ "
  by (auto simp: abs_if)

```

The stabilisers of two equivalent points are related by conjugation.

```

lemma bij_betw_stabiliser_modgrp_apply_modgrp:
  fixes h :: modgrp
  assumes x: "Im x > 0"
  defines "Stab  $\equiv$  ( $\lambda x. \{h. \text{apply\_modgrp } h \ x = x\}$ )"
  defines "y  $\equiv$  apply_modgrp h x"
  shows "bij_betw ( $\lambda f. h * f * \text{inverse } h$ ) (Stab x) (Stab y)"
proof (rule bij_betwI)
  show " $(\lambda f. h * f * \text{inverse } h) \in \text{Stab } x \rightarrow \text{Stab } y$ "
    using x unfolding Stab_def y_def Pi_def mem_Collect_eq
    by (metis Im_pole_modgrp apply_modgrp_inverse_eqI
        apply_modgrp_mult assms(3) linorder_not_le
        modgrp.Im_transform_zero_iff order.refl)
next
  show " $(\lambda f. \text{inverse } h * f * h) \in \text{Stab } y \rightarrow \text{Stab } x$ "
    using x unfolding Stab_def y_def Pi_def mem_Collect_eq
    by (metis Im_pole_modgrp apply_modgrp_inverse_eqI
        apply_modgrp_mult assms(3) linorder_not_le
        modgrp.Im_transform_zero_iff order.refl)
next
  show "inverse h * (h * f * inverse h) * h = f" for f
    by (simp add: mult.assoc)
next
  show "h * (inverse h * f * h) * inverse h = f" for f
    by (simp add: mult.assoc)
qed

lemma stabiliser_ii_modgrp: "apply_modgrp h i = i  $\longleftrightarrow$  h  $\in$  {1, -1, S_modgrp,
-S_modgrp}"
proof
  assume h: "h  $\in$  {1, -1, S_modgrp, -S_modgrp}"
  thus "apply_modgrp h i = i" by auto
next
  assume "apply_modgrp h i = i"
  define a b c d :: int
    where abcd_def: "a = modgrp_a h" "b = modgrp_b h" "c = modgrp_c h"
  "d = modgrp_d h"
  have det: "a * d - b * c = 1"
    unfolding abcd_def by (metis modgrp.unimodular)
  define n where "n = (c ^ 2 + d ^ 2)"
  have "c  $\neq$  0  $\vee$  d  $\neq$  0"
    using det by auto
  hence "n > 0"
    using det unfolding n_def by (elim disjE) (auto intro: add_pos_nonneg
add_nonneg_pos)

  have "i = apply_modgrp h i"
    by (rule sym) fact
  also have "apply_modgrp h i = (a * i + b) / (c * i + d)"
    by (simp add: apply_modgrp_altdef moebius_def field_simps abcd_def)

```

```

also have "... = (a * i + b) * (d - c * i) / n"
  by (subst complex_div_cnj) (auto simp: cmod_def n_def)
also have "... = i  $\longleftrightarrow$  (a * i + b) * (d - c * i) = n * i"
  using <n > 0> by (simp add: divide_simps mult_ac)
also have "...  $\longleftrightarrow$  real_of_int (a * c + b * d) = of_int 0  $\wedge$  real_of_int
(a * d - b * c) = of_int n"
  unfolding of_int_add by (simp add: complex_eq_iff algebra_simps)
also have "...  $\longleftrightarrow$  a * c + b * d = 0  $\wedge$  a * d - b * c = n"
  by (simp only: of_int_eq_iff)
also have "a * d - b * c = 1"
  by (rule det)
finally have eq: "a * c + b * d = 0" and "n = 1"
  by auto

```

```

from <n = 1> have "c = 0  $\wedge$  abs d = 1  $\vee$  abs c = 1  $\wedge$  d = 0" unfolding
ing n_def
  by (metis (no_types, lifting) abs_mult abs_zmult_eq_1
    add.right_neutral add_0 add_mono_thms_linordered_field(1) div_0
    nonzero_mult_div_cancel_left power2_abs power2_eq_square
    zabs_less_one_iff zero_less_abs_iff)
with eq det show "h  $\in$  {1, -1, S_modgrp, -S_modgrp}" using det
  unfolding insert_iff modgrp_eq_iff abcd_def [symmetric] abs_eq_1_iff
zmult_eq_1_iff
  by (elim disjE conjE) auto
qed

```

```

lemma ellorder_modgrp_UNIV_ii [simp]: "ellorder_modgrp UNIV i = 2"
proof -
  have "ellorder_modgrp UNIV i = card {h. apply_modgrp h i = i} div card
{1::modgrp, -1}"
    by (simp add: ellorder_modgrp_def)
  also have "card {1::modgrp, -1} = 2"
    by (auto simp: modgrp_eq_iff)
  also have "{h. apply_modgrp h i = i} = {1, -1, S_modgrp, -S_modgrp}"
    by (subst stabiliser_ii_modgrp) auto
  also have "card ... = 4"
    by (simp add: modgrp_eq_iff)
  finally show ?thesis
    by simp
qed

```

```

lemma ellorder_modgrp_UNIV_ii': "z  $\sim_{\Gamma}$  i  $\implies$  ellorder_modgrp UNIV z =
2"
  by (metis ellorder_modgrp_UNIV_ii modular_group.ellorder_modgrp_cong)

```

```

lemma stabiliser_rho_modgrp_aux:
  assumes "a  $\wedge$  2 + a * b + b  $\wedge$  2  $\leq$  (1::int)"
  shows "|b|  $\leq$  1"

```

```

proof -
  have "1 ≥ real_of_int (a ^ 2 + a * b + b ^ 2)"
    using assms by linarith
  also have "real_of_int (a ^ 2 + a * b + b ^ 2) = (a + b / 2) ^ 2 + 3
/ 4 * b ^ 2"
    by (simp add: algebra_simps power2_eq_square)
  finally have "b ^ 2 ≤ 4 / 3 * (1 - (a + b / 2) ^ 2)"
    by (simp add: algebra_simps)
  also have "... ≤ 4 / 3 * (1 - 0) ^ 2"
    by (intro mult_left_mono diff_left_mono) auto
  finally have "real_of_int (b ^ 2) ≤ 4 / 3"
    by simp
  hence "b ^ 2 ≤ 1"
    by linarith
  thus "|b| ≤ 1"
    by (metis abs_square_le_1)
qed

lemma stabiliser_rho_modgrp:
  defines "ST ≡ S_modgrp * T_modgrp"
  shows "apply_modgrp h ϱ = ϱ ↔ h ∈ {1, -1, ST, -ST, ST2, -(ST2)}"
proof
  assume h: "h ∈ {1, -1, ST, -ST, ST2, -(ST2)}"
  show "apply_modgrp h ϱ = ϱ" using h unfolding ST_def
    by (elim insertE)
    (simp_all add: apply_modgrp_mult power2_eq_square complex_eq_iff
Re_divide Im_divide
del: apply_modgrp_abs)
next
  assume "apply_modgrp h ϱ = ϱ"
  define a b c d :: int
    where abcd_def: "a = modgrp_a h" "b = modgrp_b h" "c = modgrp_c h"
"d = modgrp_d h"
  have det: "a * d - b * c = 1"
    unfolding abcd_def by (metis modgrp.unimodular)
  have nz: "c * ϱ + d ≠ 0"
    using det by (auto simp: complex_eq_iff)

  have "ϱ = apply_modgrp h ϱ"
    by (rule sym) fact
  also have "apply_modgrp h ϱ = (a * ϱ + b) / (c * ϱ + d)"
    by (simp add: apply_modgrp_altdef moebius_def field_simps abcd_def)
  also have "... = ϱ ↔ (a * ϱ + b) = c * ϱ2 + d * ϱ"
    using nz by (simp add: field_simps power2_eq_square)
  finally have "(b + c) + ϱ * (a + c - d) = 0"
    by (simp add: modfun_rho_square algebra_simps)

  hence "of_int (c + d + 2 * b) = real_of_int a ∧ of_int (a + c) * sqrt
3 = of_int d * sqrt 3"

```

```

    unfolding of_int_add by (auto simp: modfun_rho_altdef algebra_simps
complex_eq_iff)
    also have "of_int (a + c) * sqrt 3 = of_int d * sqrt 3  $\longleftrightarrow$  of_int (a
+ c) = real_of_int d"
    by (subst mult_cancel_right) auto
    finally have ac: "c = -b" "a = d - c"
    unfolding of_int_eq_iff by (auto simp: algebra_simps)

from ac and det have *: "b ^ 2 + b * d + d ^ 2 = 1"
  by (auto simp: algebra_simps power2_eq_square)
have "|b|  $\leq$  1" "|d|  $\leq$  1"
  using stabiliser_rho_modgrp_aux[of b d] stabiliser_rho_modgrp_aux[of
d b] *
  by (simp_all add: add_ac mult_ac)
with * have "(b, d)  $\in$  {(1, 0), (-1, 0), (0, 1), (0, -1), (1, -1), (-1,
1)}"
  unfolding abs_le_1_iff_int insert_iff prod.inject by (elim disjE)
simp_all
thus "h  $\in$  {1, -1, ST, -ST, ST2, -(ST2)}" using ac
  unfolding insert_iff modgrp_eq_iff abcd_def [symmetric] abs_eq_1_iff
zmult_eq_1_iff ST_def
  by (elim disjE conjE) (auto simp: power2_eq_square)
qed

```

```

lemma ellorder_modgrp_UNIV_rho [simp]: "ellorder_modgrp UNIV  $\varrho$  = 3"
proof -
  define ST where "ST = S_modgrp * T_modgrp"
  have "ellorder_modgrp UNIV  $\varrho$  = card {h. apply_modgrp h  $\varrho$  =  $\varrho$ } div card
{1::modgrp, -1}"
  by (simp add: ellorder_modgrp_def)
  also have "card {1::modgrp, -1} = 2"
  by (auto simp: modgrp_eq_iff)
  also have "{h. apply_modgrp h  $\varrho$  =  $\varrho$ } = {1, -1, ST, -ST, ST2, -(ST2)}"
  by (subst stabiliser_rho_modgrp) (auto simp: ST_def)
  also have "card ... = 6"
  by (simp add: modgrp_eq_iff ST_def power2_eq_square)
  finally show ?thesis
  by simp
qed

```

```

lemma ellorder_modgrp_UNIV_rho': "z  $\sim_{\Gamma}$   $\varrho$   $\implies$  ellorder_modgrp UNIV z
= 3"
  by (metis ellorder_modgrp_UNIV_rho modular_group.ellorder_modgrp_cong)

```

Other than i and ρ , no non-trivial unimodular transformation has any fixed points in the fundamental region.

```

lemma modgrp_fixed_point_trivial_std_fund_region':
  assumes "z  $\in$   $\mathcal{R}_{\Gamma}'$  - {i,  $\varrho$ }" "apply_modgrp h z = z"
  shows "abs h = 1"

```

```

proof (cases "z ∈ RΓ")
  case True
  thus ?thesis
    using assms std_fund_region_no_fixed_point by blast
next
  case False
  define a b c d :: int
    where abcd_def: "a = modgrp_a h" "b = modgrp_b h" "c = modgrp_c h"
    "d = modgrp_d h"
  have h_eq: "h = modgrp a b c d"
    unfolding abcd_def by (rule modgrp.as_modgrp_altdef)
  have det: "a * d - b * c = 1"
    unfolding abcd_def by (metis modgrp.unimodular)

  from assms have "Im z > 0"
    by (auto simp: std_fund_region'_def std_fund_region_def)
  from False have "norm z = 1 ∧ 0 < Im z ∧ Re z ∈ {-1/2..0} ∨ Re z
= -1/2 ∧ sqrt 3 / 2 ≤ Im z"
    using assms std_fund_region'_minus_std_fund_region by blast
  thus ?thesis
  proof
    assume z: "Re z = -1/2 ∧ sqrt 3 / 2 ≤ Im z"
    define y where "y = Im z"
    have "y ≠ sqrt 3 / 2"
      using assms z by (auto simp: complex_eq_iff y_def)
    with z have y: "y > sqrt 3 / 2"
      by (auto simp: y_def)
    have z_eq: "z = -1/2 + i * y"
      using z by (auto simp: y_def complex_eq_iff)

    have nz: "c * z + d ≠ 0"
      using det y by (auto simp: complex_eq_iff z_eq)
    have "y ≠ 0"
      using y by auto

    have "z = apply_modgrp h z"
      by (rule sym) fact
    also have "... = (a * z + b) / (c * z + d)"
      by (simp add: apply_modgrp_altdef moebius_def flip: abcd_def)
    finally have eq: "c * z ^ 2 + (d - a) * z - b = 0"
      using nz by (simp add: field_simps power2_eq_square)

    have "0 = Im (c * z ^ 2 + (d - a) * z - b)"
      by (subst eq) auto
    also have "... = y * (d - a - c)"
      by (simp add: z_eq power2_eq_square field_simps)
    finally have a_eq: "a = d - c"
      using <y ≠ 0> by simp
  end
end

```

```

from det have bc_eq: "b * c = d ^ 2 - c * d - 1"
  by (auto simp: a_eq algebra_simps power2_eq_square)

have "0 = 4 * c * Re (c * z ^ 2 + (d - a) * z - b)"
  by (subst eq) auto
also have "... = -(4 * (b * c) + c ^ 2 * (4 * y ^ 2 + 1))"
  by (simp add: a_eq z_eq power2_eq_square field_simps)
also have "... = 4 - ((c - 2 * d) ^ 2 + (2*c*y)^2)"
  by (subst bc_eq) (auto simp: algebra_simps power2_eq_square)
finally have eq': "(c - 2 * d) ^ 2 + (2 * c * y) ^ 2 = 4"
  by simp

show "abs h = 1"
proof (cases "c = 0")
  case [simp]: True
  from eq' have d: "d = 1 ∨ d = -1"
    by (auto simp: power2_eq_1_iff)
  have "b = 0"
    using d det eq by auto
  with d a_eq show ?thesis
    by (elim disjE) (simp_all add: h_eq modgrp_eq_iff modgrp_abcd_modgrp
abs_modgrp_altdef)
  next
  case c: False
  have "3 * c ^ 2 = (2 * c * (sqrt 3 / 2)) ^ 2"
    by (simp add: power_mult_distrib)
  also have "... ≤ (c - 2 * d) ^ 2 + (2 * c * (sqrt 3 / 2)) ^ 2"
    by simp
  also have "... < (c - 2 * d) ^ 2 + (2 * c * y) ^ 2"
    unfolding power_mult_distrib using c
    by (intro add_strict_left_mono power_strict_mono mult_strict_left_mono
y) auto
  also have "... = 4"
    by (subst eq') auto
  finally have "real_of_int (c ^ 2) < 4 / 3"
    by simp
  hence "c ^ 2 ≤ 1"
    by linarith
  hence "c ^ 2 = 1" "abs c = 1"
    using c by (auto simp: abs_square_le_1 abs_square_eq_1)

  have "real_of_int ((c - 2 * d)^2 + 3) = (c - 2 * d) ^ 2 + (2 * c
* (sqrt 3 / 2)) ^ 2"
    by (simp add: power_mult_distrib <c ^ 2 = 1> flip: of_int_power)
  also have "... < (c - 2 * d) ^ 2 + (2 * c * y) ^ 2"
    unfolding power_mult_distrib using c
    by (intro add_strict_left_mono power_strict_mono mult_strict_left_mono
y) auto
  also have "... = 4"

```

```

    by (subst eq') auto
  finally have "(c - 2 * d) ^ 2 < 1"
    by linarith
  hence "c - 2 * d = 0"
    by (simp add: abs_square_less_1)
  with <abs c = 1> have False
    by presburger
  thus ?thesis ..
qed

next
assume z: "norm z = 1 ∧ Im z > 0 ∧ Re z ∈ {-1/2..0}"

define x y where "x = Re z" and "y = Im z"
have z_eq: "z = Complex x y"
  by (auto simp: complex_eq_iff x_def y_def)
have xy: "x ^ 2 + y ^ 2 = 1" "y > 0" "x ∈ {-1/2..0}"
  using z by (auto simp: x_def y_def cmod_def)

have "x ≠ 0"
  using assms xy by (auto simp: complex_eq_iff power2_eq_1_iff x_def
y_def)
moreover have "x ≠ -1/2"
proof
  assume "x = -1/2"
  hence "y ^ 2 = (sqrt 3 / 2) ^ 2"
    using xy by (auto simp: complex_eq_iff power_divide power2_eq_1_iff
x_def y_def)
  hence "y = sqrt 3 / 2"
    using <y > 0> by (subst (asm) power2_eq_iff_nonneg) auto
  with <x = -1/2> show False
    using assms by (auto simp: complex_eq_iff z_eq)
qed
ultimately have x: "x ∈ {-1/2<..<0}"
  using xy by auto

have nz: "c * z + d ≠ 0"
  using det xy by (auto simp: z_eq complex_eq_iff)
have "z = apply_modgrp h z"
  by (rule sym) fact
also have "... = (a * z + b) / (c * z + d)"
  by (simp add: apply_modgrp_altdef moebius_def flip: abcd_def)
finally have eq: "c * z ^ 2 + (d - a) * z - b = 0"
  using nz by (simp add: field_simps power2_eq_square)

have "0 = Im (c * z ^ 2 + (d - a) * z - b)"
  by (subst eq) auto
also have "... = y * (d - a + 2 * c * x)"
  by (simp add: z_eq power2_eq_square field_simps)

```

```

finally have d_eq: "d = a - 2 * c * x"
  using <y > 0> by simp

have "0 = Re (c * z ^ 2 + (d - a) * z - b)"
  by (subst eq) auto
also have "... = -(b + c * (x ^ 2 + y ^ 2))"
  by (simp add: z_eq power2_eq_square field_simps d_eq)
also have "x ^ 2 + y ^ 2 = 1"
  using xy by simp
finally have b_eq: "b = -c"
  by auto

show ?thesis
proof (cases "c = 0")
  case True
  hence "h = 1 ∨ h = -1"
    using det b_eq xy eq
    by (auto simp: h_eq abs_modgrp_altdef modgrp_abcd_modgrp zmult_eq_1_iff
modgrp_eq_iff)
  thus ?thesis
    by auto
  next
  case False
  with d_eq have x_eq: "x = (a - d) / (2 * c)"
    by auto
  have "|x| ^ 2 * (2*c) ^ 2 ≤ (1/2) ^ 2 * (2*c) ^ 2"
    using xy by (intro mult_right_mono power_mono) auto
  also have "|x| ^ 2 * (2*c) ^ 2 = (a - d) ^ 2"
    using False by (auto simp: x_eq abs_mult power_divide power_mult_distrib)
  also have "(1/2) ^ 2 * (2*c) ^ 2 = c ^ 2"
    by (simp add: power_divide)
  also have "c ^ 2 = 1 - a * d"
    using det by (simp add: b_eq power2_eq_square)
  finally have "(a - d) ^ 2 ≤ 1 - a * d"
    by linarith
  hence ineq: "a ^ 2 - a * d + d ^ 2 ≤ 1"
    by (simp add: power2_eq_square algebra_simps)
  hence "|a| ≤ 1" "|d| ≤ 1"
    using stabiliser_rho_modgrp_aux[of "-a" d] stabiliser_rho_modgrp_aux[of
"-d" a]
    by (simp_all add: mult_ac add_ac)
  hence False
    unfolding abs_le_1_iff_int using det False ineq x
    by (elim disjE) (auto simp: b_eq x_eq power_divide zmult_eq_1_iff)
  thus ?thesis ..
qed
qed
qed

```

Therefore, all points not equivalent to i or ρ have elliptic order 1, i.e. are

not elliptic points.

lemma `ellorder_modgrp_UNIV_eq_1_std_fund_region'`:

assumes " $z \in \mathcal{R}_\Gamma' - \{i, \rho\}$ "
shows "`ellorder_modgrp UNIV z = 1`"

proof -

from `assms` have "`ellorder_modgrp UNIV z = card {h. apply_modgrp h z = z} div 2`"

by (`simp add: ellorder_modgrp_def modgrp_eq_iff numeral_2_eq_2 in_std_fund_region'_iff`)

also have "`{h. apply_modgrp h z = z} = {1, -1}`"

using `modgrp_fixed_point_trivial_std_fund_region'[of z] assms`

by (`auto simp: abs_modgrp_eq_1_iff`)

also have "`card ... = 2`"

by (`simp add: modgrp_eq_iff numeral_2_eq_2`)

finally show `?thesis`

by `simp`

qed

lemma `ellorder_modgrp_UNIV_eq_1`:

assumes " $\text{Im } z > 0$ " " $\neg(z \sim_\Gamma i)$ " " $\neg(z \sim_\Gamma \rho)$ "

shows "`ellorder_modgrp UNIV z = 1`"

proof -

obtain `z'` where `z': "z' \in \mathcal{R}_\Gamma'" "z \sim_\Gamma z'"`

using `canonical_point_in_std_fund_region' assms(1) by blast`

have " $\text{Im } z' > 0$ "

using `z'` by (`auto simp: in_std_fund_region'_iff`)

have "`ellorder_modgrp UNIV z = ellorder_modgrp UNIV z'`"

by (`intro modular_group.ellorder_modgrp_cong`) (`use z' in auto`)

also have "`... = 1`"

by (`rule ellorder_modgrp_UNIV_eq_1_std_fund_region'`) (`use z' assms in auto`)

finally show `?thesis .`

qed

lemma `ellorder_modgrp_UNIV_eq`:

`"ellorder_modgrp UNIV z =`

`(if $\text{Im } z \leq 0$ then 0 else if $z \sim_\Gamma i$ then 2 else if $z \sim_\Gamma \rho$ then 3 else 1)"`

proof (`cases "Im z > 0"`)

case `True`

thus `?thesis`

using `ellorder_modgrp_UNIV_eq_1[of z] ellorder_modgrp_UNIV_ii ellorder_modgrp_UNIV_rho modular_group.ellorder_modgrp_cong[of z]`

by (`auto dest: modular_group.ellorder_modgrp_cong`)

qed (`auto simp: ellorder_modgrp_def`)

A simple consequence: no unimodular transformation sends ρ to i .

lemma `not_rho_equiv_i [simp]: "\(\rho \sim_\Gamma i)"`

proof

assume " $\rho \sim_\Gamma i$ "

```

    hence "ellorder_modgrp UNIV  $\rho$  = ellorder_modgrp UNIV i"
      by (rule modular_group.ellorder_modgrp_cong)
    thus False
      by simp
qed

lemma not_i_equiv_rho [simp]: " $\neg(i \sim_{\Gamma} \rho)$ "
  by (subst modular_group.rel_commutes) simp

lemma not_modular_group_rel_rho_i [simp]: " $z \sim_{\Gamma} \rho \implies \neg z \sim_{\Gamma} i$ "
  by (meson modular_group.rel_sym modular_group.rel_trans not_i_equiv_rho)

lemma modular_group_rel_rho_i_cases [case_names rho i neither invalid]:
  obtains " $z \sim_{\Gamma} \rho$ " " $\neg z \sim_{\Gamma} i$ " | " $z \sim_{\Gamma} i$ " " $\neg z \sim_{\Gamma} \rho$ " | "Im z > 0" " $\neg z \sim_{\Gamma} \rho$ " " $\neg z \sim_{\Gamma} i$ " | "Im z  $\leq$  0"
  by (cases "Im z > 0"; cases "z  $\sim_{\Gamma} \rho$ "; cases "z  $\sim_{\Gamma} i$ ") auto

lemma finite_modgrp_fixpoints:
  assumes "Im z > 0"
  shows "finite {h  $\in$  G. apply_modgrp h z = z}"
proof -
  obtain z' where z': "z'  $\in$   $\mathcal{R}_{\Gamma}$ " "z  $\sim_{\Gamma}$  z'"
    using canonical_point_in_std_fund_region'[of z] assms by blast
  then obtain h where h: "z' = apply_modgrp h z"
    by (elim modular_group.relE)
  have "finite {h. apply_modgrp h z' = z'}"
  proof (rule finite_subset)
    define S where "S = S_modgrp"
    define ST where "ST = S_modgrp * T_modgrp"
    show "{h. apply_modgrp h z' = z'}  $\subseteq$  {1, -1, S, -S, ST, -ST, ST2, -(ST2)}"
      by (cases "z' = i"; cases "z' =  $\rho$ ")
        (use z'(1) stabiliser_ii_modgrp stabiliser_rho_modgrp
          modgrp_fixed_point_trivial_std_fund_region'[of z']
          in <auto simp: S_def ST_def abs_modgrp_eq_1_iff>)
  qed auto
  moreover have "bij_betw ( $\lambda f. h * f * \text{inverse } h$ ) {h. apply_modgrp h z = z} {h. apply_modgrp h z' = z'}"
    unfolding h by (rule bij_betw_stabiliser_modgrp_apply_modgrp) fact
  hence "finite {h. apply_modgrp h z = z}  $\longleftrightarrow$  finite {h. apply_modgrp h z' = z'}"
    by (rule bij_betw_finite)
  ultimately have "finite {h. apply_modgrp h z = z}"
    by blast
  thus ?thesis
    by simp
qed

lemma (in modgrp_subgroup) ellorder_modgrp_pos:

```

```

    assumes "Im z > 0"
    shows "ellorder_modgrp G z > 0"
  proof -
    from assms have "ellorder_modgrp G z = card {h∈G. apply_modgrp h z
= z} div card (G ∩ {1, -1})"
      by (auto simp: ellorder_modgrp_def)
    also have "... ≥ card (G ∩ {1, -1}) div card (G ∩ {1, -1})"
      by (intro div_le_mono card_mono finite_modgrp_fixpoints) (use assms
in auto)
    also have "card (G ∩ {1, -1}) ≥ card {1::modgrp}"
      by (rule card_mono) auto
    hence "card (G ∩ {1, -1}) div card (G ∩ {1, -1}) = 1"
      by simp
    finally show ?thesis
      by simp
  qed

lemma (in modgrp_subgroup)
  assumes "Im z > 0"
  shows "ellorder_modgrp_dvd: \"card (G ∩ {1, -1}) dvd card {h∈G. apply_modgrp
h z = z}\"
    and "card_modgrp_fixpoints:
      \"card {h∈G. apply_modgrp h z = z} = ellorder_modgrp G z *
card (G ∩ {1, -1})\"
  proof -
    define B where "B = {h ∈ G. apply_modgrp h z = z}"
    define n where "n = card (G ∩ {1, -1})"
    show "n dvd card B"
    proof (cases "-1 ∈ G")
      case False
      thus ?thesis by (simp add: n_def)
    next
      case True
      define A where "A = abs ` {h ∈ G. apply_modgrp h z = z}"
      have [simp]: "-f ∈ G ⟷ f ∈ G" for f
        using times_in_G[OF True, of f] times_in_G[OF True, of "-f"] by
auto
      have [simp]: "abs f ∈ G ⟷ f ∈ G" for f
        by (auto simp: abs_modgrp_altdef)
      have **: "-abs f = f" if "abs f ≠ f" for f :: modgrp
        using that by (auto simp: abs_modgrp_altdef split: if_splits)
      have bij: "bij_betw (λ(b,f). if b then -f else f) (UNIV × A) B"
        by (rule bij_betwI[of _ _ _ "λf. (abs f ≠ f, abs f)"])
        (auto simp: A_def B_def ** split: if_splits)
      have "card B = n * card A"
        using bij_betw_same_card[OF bij] by (simp add: card_cartesian_product
n_def)
      thus ?thesis
        by simp
    qed
  end

```

qed

```
have "ellorder_modgrp G z * n = card B div n * n"
  using assms by (simp add: ellorder_modgrp_def B_def n_def)
also have "... = card B"
  using <n dvd card B> by simp
finally show "card B = ellorder_modgrp G z * n" ..
```

qed

A point having an elliptic order of 1 means that the point is fixed only by the trivial maps $\pm I$.

```
lemma (in modgrp_subgroup) ellorder_modgrp_eq_1_iff:
  assumes "Im z > 0"
  shows "ellorder_modgrp G z = 1  $\longleftrightarrow$  ( $\forall f \in G. \text{apply\_modgrp } f \ z = z \longleftrightarrow |f| = 1$ )"
```

proof

```
  assume *: "( $\forall f \in G. \text{apply\_modgrp } f \ z = z \longleftrightarrow |f| = 1$ )"
  have "ellorder_modgrp G z = card {h  $\in$  G. apply_modgrp h z = z} div
    card (G  $\cap$  {1, -1})"
    using assms by (simp add: ellorder_modgrp_def)
  also from * have "{f  $\in$  G. apply_modgrp f z = z} = {f  $\in$  G. abs f = 1}"
    by blast
  also have "... = G  $\cap$  {1, -1}"
    by (auto simp: abs_modgrp_eq_1_iff)
  finally show "ellorder_modgrp G z = 1"
    by simp
```

next

```
  assume *: "ellorder_modgrp G z = 1"
  have "G  $\cap$  {1, -1} = {h  $\in$  G. apply_modgrp h z = z}"
  proof (rule card_subset_eq)
    from * show "card (G  $\cap$  {1, -1}) = card {h  $\in$  G. apply_modgrp h z = z}"
      = z}"
      by (subst card_modgrp_fixpoints) (use assms in auto)
    show "G  $\cap$  {1, -1}  $\subseteq$  {h  $\in$  G. apply_modgrp h z = z}"
      by auto
    show "finite {h  $\in$  G. apply_modgrp h z = z}"
      by (rule finite_modgrp_fixpoints) (use assms in auto)
```

qed

```
  also have "G  $\cap$  {1, -1} = {f  $\in$  G. abs f = 1}"
    by (auto simp: abs_modgrp_eq_1_iff)
  finally show " $\forall f \in G. \text{apply\_modgrp } f \ z = z \longleftrightarrow |f| = 1$ "
    by blast
```

qed

```
lemma modgrp_fixed_point_trivial:
  assumes "Im z > 0" " $\neg z \sim_{\Gamma} i$ " " $\neg z \sim_{\Gamma} 0$ " "apply_modgrp f z = z"
  shows "|f| = 1"
proof -
  have "ellorder_modgrp UNIV z = 1"
```

```

    by (rule ellorder_modgrp_UNIV_eq_1) (use assms in auto)
  thus " $|f| = 1$ "
    by (subst (asm) modular_group.ellorder_modgrp_eq_1_iff) (use assms
in auto)
qed

```

5.6 Fundamental regions for congruence subgroups

```

context hecke_prime_subgroup
begin

```

```

definition std_fund_region_cong ("R") where
  "R = R_Γ ∪ (⋃ k ∈ {0..<p}. (λz. -1 / (z + of_int k)) ' R_Γ)"

```

```

lemma std_fund_region_cong_altdef:
  "R = R_Γ ∪ (⋃ k ∈ {0..<p}. apply_modgrp (S_shift_modgrp k) ' R_Γ)"
proof -
  have "apply_modgrp (S_shift_modgrp k) ' R_Γ = (λz. -1 / (z + of_int
k)) ' R_Γ" for k
    unfolding S_shift_modgrp_def
    by (intro image_cong refl, subst apply_modgrp_mult) auto
  thus ?thesis
    by (simp add: std_fund_region_cong_def)
qed

```

```

lemma closure_UN_finite: "finite A ⇒ closure (⋃ A) = (⋃ X ∈ A. closure
X)"
  by (induction A rule: finite_induct) auto

```

```

sublocale std_region: fundamental_region Γ' R

```

```

proof
  show "open R"
    unfolding std_fund_region_cong_altdef
    by (intro open_Un open_UN ballI open_std_fund_region apply_modgrp_open_map)
auto
next
  show "R ⊆ {z. Im z > 0}"
    by (auto simp: std_fund_region_cong_altdef in_std_fund_region_iff)
next
  fix x assume "Im x > 0"
  then obtain y where y: "y ∈ closure R_Γ" "x ~_Γ y"
    using std_fund_region.equiv_in_closure by blast
  then obtain f where f: "y = apply_modgrp f x" "Im y > 0"
    by (auto simp: modular_group.rel_def)
  obtain g h where gh: "g ∈ Γ'" "h = 1 ∨ (∃ k ∈ {0..<p}. h = S_shift_modgrp
k)" "inverse f = g * h"
    using modgrp_decompose' [of "inverse f"] .
  have inverse_g_eq: "inverse g = h * f"

```

```

using gh(3) by (metis modgrp.assoc modgrp.inverse_unique modgrp.left_inverse)

show "∃y∈closure  $\mathcal{R}$ . rel x y"
  using gh(2)
proof safe
  assume "h = 1"
  thus ?thesis using y f gh
    by (auto simp: std_fund_region_cong_altdef intro!: bexI[of _ y])
next
  fix k assume k: "k ∈ {0..<p>}" "h = S_shift_modgrp k"
  have "apply_modgrp h y ∈ apply_modgrp h 'closure  $\mathcal{R}_\Gamma$ "
    using y by blast
  also have "... ⊆ closure (apply_modgrp h '  $\mathcal{R}_\Gamma$ )"
    by (intro continuous_image_closure_subset[of "{z. Im z > 0}"])
      (auto intro!: continuous_intros closure_std_fund_region_Im_pos)
  also have "apply_modgrp h y = apply_modgrp (inverse g) x"
    unfolding inverse_g_eq using <Im x > 0> f by (subst apply_modgrp_mult)
auto
  finally have "apply_modgrp (inverse g) x ∈ closure (apply_modgrp h
'  $\mathcal{R}_\Gamma$ )" .
  moreover have "rel x (apply_modgrp (inverse g) x)"
    using <Im x > 0> gh by auto
  ultimately show ?thesis
    unfolding std_fund_region_cong_altdef using k by (auto simp: closure_UN_finite)
qed
next
  fix x y assume xy: "x ∈  $\mathcal{R}$ " "y ∈  $\mathcal{R}$ " "rel x y"
  define ST where "ST = (λk. apply_modgrp (S_shift_modgrp k) :: complex
⇒ complex)"

  have 1: False
    if xy: "x ∈  $\mathcal{R}_\Gamma$ " "y ∈  $\mathcal{R}_\Gamma$ " "rel x (ST k y)" and k: "k ∈ {0..<p>}"
for x y k
proof -
  have "x ~ $\Gamma$  ST k y"
    using xy(3) by (rule rel_imp_rel)
  hence "x ~ $\Gamma$  y"
    by (auto simp: ST_def)
  hence [simp]: "x = y"
    using xy(1,2) std_fund_region.unique by blast
  with xy(3) have "rel (ST k x) x"
    by (simp add: rel_commutes)
  then obtain f where f: "f ∈  $\Gamma$ " "apply_modgrp f (ST k x) = x" "Im
x > 0"
    unfolding rel_def by blast

  have "modgrp_c |f| = 1"
proof -
  from f have "apply_modgrp (f * S_shift_modgrp k) x = x"

```

```

    by (subst apply_modgrp_mult) (auto simp: ST_def)
  hence "|f * S_shift_modgrp k| = 1"
    using xy by (intro std_fund_region_no_fixed_point) auto
  hence "|f| = |inverse (S_shift_modgrp k)|"
    by (metis abs_modgrp_congs(3) abs_modgrp_eq_1_iff abs_uminus_modgrp
minus_minus_modgrp
      modgrp.inverse_inverse modgrp.inverse_unique times_modgrp_uminus_right)
  also have "modgrp_c (|inverse (S_shift_modgrp k)|) = 1"
    by (simp add: S_shift_modgrp_def S_modgrp_code shift_modgrp_code
inverse_modgrp_code
      times_modgrp_code modgrp_c_code abs_modgrp_code)
  finally show "modgrp_c |f| = 1" .
qed
hence "|modgrp_c f| = 1"
  by (auto simp: abs_modgrp_altdef split: if_splits)
hence "is_unit (modgrp_c f)"
  by auto
moreover have "p dvd modgrp_c f"
  using <f ∈ Γ> unfolding subgrp_def modgrps_hecke_altdef by auto
ultimately have "is_unit p"
  using dvd_trans by blast
moreover have "¬is_unit p"
  using p_prime using not_prime_unit by blast
ultimately show False
  by contradiction
qed

have "x ∈ R_Γ ∪ (⋃k∈{0..<p>. ST k ' R_Γ)" "y ∈ R_Γ ∪ (⋃k∈{0..<p>.
ST k ' R_Γ)"
  using xy unfolding ST_def std_fund_region_cong_altdef by blast+
thus "x = y"
proof safe
  assume "x ∈ R_Γ" "y ∈ R_Γ"
  thus "x = y"
    using <rel x y> rel_imp_rel std_fund_region.unique by blast
next
  fix k y'
  assume "x ∈ R_Γ" "k ∈ {0..<p>" "y' ∈ R_Γ" "y = ST k y'"
  thus "x = ST k y'"
    using 1[of x y' k] <rel x y> by auto
next
  fix k x'
  assume "x' ∈ R_Γ" "k ∈ {0..<p>" "y ∈ R_Γ" "x = ST k x'"
  thus "ST k x' = y"
    using 1[of y x' k] <rel x y> by (auto simp: rel_commutes)
next
  fix x' y' :: complex and k1 k2 :: int
  assume xy': "x' ∈ R_Γ" "y' ∈ R_Γ" "x = ST k1 x'" "y = ST k2 y'"
  assume k12: "k1 ∈ {0..<p>" "k2 ∈ {0..<p>"

```

```

have x': "Im x' > 0"
  using xy' by (auto simp: in_std_fund_region_iff)
have "ST k1 x' ~Γ ST k2 y'"
  using xy xy' by (intro rel_imp_rel) auto
hence "x' ~Γ y'"
  by (auto simp: ST_def)
hence [simp]: "x' = y'"
  using xy' by (intro std_fund_region.unique)
have "rel (ST k1 x') (ST k2 x')"
  using xy xy' by simp
then obtain f where f: "f ∈ Γ'" "apply_modgrp f (ST k1 x') = ST
k2 y'"
  unfolding rel_def by auto

have "p dvd modgrp_c |f|"
  using f by (auto simp: subgrp_def modgrps_hecke_altdef abs_modgrp_altdef)
also have "modgrp_c |f| = |k2 - k1|"
proof -
  note <apply_modgrp f (ST k1 x') = ST k2 y'>
  also have "apply_modgrp f (ST k1 x') = apply_modgrp (f * S_shift_modgrp
k1) x'"
    unfolding ST_def using x' by (subst apply_modgrp_mult) auto
  finally have "|f * S_shift_modgrp k1| = |S_shift_modgrp k2|"
    unfolding ST_def using xy' by (intro std_fund_region_no_fixed_point'[of
x']) auto
  hence "|f| = |S_shift_modgrp k2 * inverse (S_shift_modgrp k1)|"
    by (smt (verit, ccfv_threshold) abs_modgrp_mult_cong modgrp.assoc
modgrp.right_inverse
mult_1_right)
  also have "... = |S_modgrp * shift_modgrp (k2 - k1) * S_modgrp|"
    using shift_modgrp_add[of k2 "-k1"]
    by (simp add: S_shift_modgrp_def modgrp.inverse_distrib_swap modgrp.assoc
flip: shift_modgrp_minus)
  finally have "|f| = |S_modgrp * shift_modgrp (k2 - k1) * S_modgrp|"
.
  also have "modgrp_c |S_modgrp * shift_modgrp (k2 - k1) * S_modgrp|
= |k2 - k1|"
    by (simp add: S_modgrp_code shift_modgrp_code times_modgrp_code
modgrp_c_code abs_modgrp_code)
  finally show "modgrp_c |f| = |k2 - k1|" .
qed
finally have "p dvd |k2 - k1|" .
moreover from k12 have "|k2 - k1| < p"
  by auto
ultimately have "k1 = k2"
  using zdvd_not_zless[of "|k2 - k1|" p] by (cases "k1 = k2") auto
thus "ST k1 x' = ST k2 y'"
  by simp
qed

```

qed

end

```
bundle modfun_region_notation
begin
notation std_fund_region (" $\mathcal{R}_\Gamma$ ")
notation modfun_rho (" $\rho$ ")
end
```

```
unbundle no_modfun_region_notation
unbundle no_modgrp_notation
```

end

6 Elliptic Functions

```
theory Elliptic_Functions
  imports Complex_Lattices
begin
```

6.1 Definition

In the context of a complex lattice Λ , a function is called *elliptic* if it is meromorphic and periodic w.r.t. the lattice.

```
locale elliptic_function = complex_lattice_periodic  $\omega_1$   $\omega_2$   $f$ 
  for  $\omega_1$   $\omega_2$  :: complex and  $f$  :: "complex  $\Rightarrow$  complex" +
  assumes meromorphic: " $f$  meromorphic_on UNIV"
```

We call a function *nicely elliptic* if it additionally is nicely meromorphic, i.e. it has no removable singularities and returns 0 at each pole. It is easy to convert elliptic functions into nicely elliptic ones using the `remove_sings` operator and lift results from the nicely elliptic setting to the “regular” elliptic one.

```
locale nicely_elliptic_function = complex_lattice_periodic  $\omega_1$   $\omega_2$   $f$ 
  for  $\omega_1$   $\omega_2$  :: complex and  $f$  :: "complex  $\Rightarrow$  complex" +
  assumes nicely_meromorphic: " $f$  nicely_meromorphic_on UNIV"
```

```
locale elliptic_function_remove_sings = elliptic_function
begin
```

```
sublocale remove_sings: nicely_elliptic_function  $\omega_1$   $\omega_2$  "remove_sings
 $f$ "
```

```
proof
```

```
  show "remove_sings  $f$  nicely_meromorphic_on UNIV"
```

```

    using meromorphic by (rule remove_sings_nicely_meromorphic)
  have *: "remove_sings f (z + ω) = remove_sings f z" if ω: "ω ∈ Λ"
for z ω
  proof -
    have "remove_sings f (z + ω) = remove_sings (λw. f (w + ω)) z"
      by (simp add: remove_sings_shift_0' add_ac)
    also have "(λw. f (w + ω)) = f"
      by (intro ext lattice_cong) (auto simp: rel_def ω)
    finally show ?thesis .
  qed
  show "remove_sings f (z + ω1) = remove_sings f z"
    "remove_sings f (z + ω2) = remove_sings f z" for z
    by (rule *; simp; fail)+
qed

end

context elliptic_function
begin

interpretation elliptic_function_remove_sings ..

lemma isolated_singularity [simp, singularity_intros]: "isolated_singularity_at
f z"
  using meromorphic by (simp add: meromorphic_on_altdef)

lemma not_essential [simp, singularity_intros]: "not_essential f z"
  using meromorphic by (simp add: meromorphic_on_altdef)

lemma meromorphic' [meromorphic_intros]: "f meromorphic_on A"
  by (rule meromorphic_on_subset[OF meromorphic]) auto

lemma meromorphic'' [meromorphic_intros]:
  assumes "g analytic_on A"
  shows "(λx. f (g x)) meromorphic_on A"
  by (rule meromorphic_on_compose[OF meromorphic assms]) auto

Due to the lattice-periodicity of  $f$ , its derivative, zeros, poles, multiplicities,
and residues are also all lattice-periodic.

sublocale zeros: complex_lattice_periodic ω1 ω2 "isolated_zero f"
proof
  fix z :: complex
  have *: "isolated_zero f (z + c) ↔ isolated_zero (λz. f (z + c))
z" for c
    by (simp add: isolated_zero_shift' add_ac)
  from this show "isolated_zero f (z + ω1) ↔ isolated_zero f z"
    "isolated_zero f (z + ω2) ↔ isolated_zero f z"
    by (simp_all add: f_periodic)

```

qed

sublocale poles: complex_lattice_periodic ω_1 ω_2 "is_pole f"

proof

fix z :: complex

have *: "is_pole f (z + c) \longleftrightarrow is_pole ($\lambda z. f (z + c)$) z" for c
by (simp add: is_pole_shift_0' add_ac)

from this show "is_pole f (z + ω_1) \longleftrightarrow is_pole f z" "is_pole f (z + ω_2) \longleftrightarrow is_pole f z"

by (simp_all add: f_periodic)

qed

sublocale zorder: complex_lattice_periodic ω_1 ω_2 "zorder f"

proof

fix z :: complex

have *: "zorder f (z + c) = zorder ($\lambda z. f (z + c)$) z" for c
by (simp add: zorder_shift' add_ac)

from this show "zorder f (z + ω_1) = zorder f z" "zorder f (z + ω_2) = zorder f z"

by (simp_all add: f_periodic)

qed

sublocale deriv: complex_lattice_periodic ω_1 ω_2 "deriv f"

proof

fix z :: complex

have *: "deriv f (z + c) = deriv ($\lambda z. f (z + c)$) z" for c
by (simp add: deriv_shift_0' add_ac o_def)

from this show "deriv f (z + ω_1) = deriv f z" "deriv f (z + ω_2) = deriv f z"

by (simp_all add: f_periodic)

qed

sublocale higher_deriv: complex_lattice_periodic ω_1 ω_2 "(deriv ^^ n) f"

proof

fix z :: complex

have *: "(deriv ^^ n) f (z + c) = (deriv ^^ n) ($\lambda z. f (z + c)$) z" for c

by (simp add: higher_deriv_shift_0' add_ac o_def)

from this show "(deriv ^^ n) f (z + ω_1) = (deriv ^^ n) f z"
"(deriv ^^ n) f (z + ω_2) = (deriv ^^ n) f z"

by (simp_all add: f_periodic)

qed

sublocale residue: complex_lattice_periodic ω_1 ω_2 "residue f"

proof

fix z :: complex

have *: "residue f (z + c) = residue ($\lambda z. f (z + c)$) z" for c
by (simp add: residue_shift_0' add_ac o_def)

```

    from this show "residue f (z + ω1) = residue f z" "residue f (z +
ω2) = residue f z"

```

```

    by (simp_all add: f_periodic)

```

```

qed

```

```

lemma eventually_remove_sings_eq: "eventually (λw. remove_sings f w =
f w) (cosparse UNIV)"

```

```

    by (simp add: eventually_remove_sings_eq meromorphic)

```

```

lemma eventually_remove_sings_eq': "eventually (λw. remove_sings f w
= f w) (at z)"

```

```

    using eventually_remove_sings_eq by (simp add: eventually_cosparse_open_eq)

```

```

lemma isolated_zero_analytic_iff:

```

```

    assumes "f analytic_on {z}" "¬(∃z∈UNIV. f z = 0)"

```

```

    shows "isolated_zero f z ↔ f z = 0"

```

```

proof

```

```

    assume "isolated_zero f z"

```

```

    thus "f z = 0"

```

```

        using assms(1) zero_isolated_zero_analytic by blast

```

```

next

```

```

    assume "f z = 0"

```

```

    hence "f -z → 0"

```

```

        using assms(1) by (metis analytic_at_imp_isCont continuous_at)

```

```

    have "eventually (λz. remove_sings f z ≠ 0) (at z)"

```

```

    proof (rule ccontr)

```

```

        assume "¬eventually (λz. remove_sings f z ≠ 0) (at z)"

```

```

        hence "frequently (λz. remove_sings f z = 0) (at z)"

```

```

            by (auto simp: frequently_def)

```

```

        hence "remove_sings f z = 0" for z

```

```

            using remove_sings.nicely_meromorphic by (rule frequently_eq_meromorphic_imp_constant)

```

```

    auto

```

```

        with eventually_remove_sings_eq show False

```

```

            using assms(2) by simp

```

```

    qed

```

```

    hence "eventually (λz. f z ≠ 0) (at z)"

```

```

        using eventually_remove_sings_eq'[of z] by eventually_elim auto

```

```

    with <f -z → 0> show "isolated_zero f z"

```

```

        by (auto simp: isolated_zero_def)

```

```

qed

```

```

end

```

```

context nicely_elliptic_function

```

```

begin

```

```

lemma nicely_meromorphic' [meromorphic_intros]: "f nicely_meromorphic_on

```

```

A"
  by (rule nicely_meromorphic_on_subset[OF nicely_meromorphic]) auto

lemma analytic:
  assumes " $\bigwedge z. z \in A \implies \neg \text{is\_pole } f z$ "
  shows "f analytic_on A"
  using nicely_meromorphic_on_subset[OF nicely_meromorphic]
  by (rule nicely_meromorphic_without_singularities) (use assms in auto)

lemma holomorphic:
  assumes " $\bigwedge z. z \in A \implies \neg \text{is\_pole } f z$ "
  shows "f holomorphic_on A"
  using analytic by (rule analytic_imp_holomorphic) (use assms in blast)

lemma continuous_on:
  assumes " $\bigwedge z. z \in A \implies \neg \text{is\_pole } f z$ "
  shows "continuous_on A f"
  using holomorphic by (rule holomorphic_on_imp_continuous_on) (use assms
in blast)

sublocale elliptic_function  $\omega_1$   $\omega_2$  f
proof
  show "f meromorphic_on UNIV"
    using nicely_meromorphic unfolding nicely_meromorphic_on_def by blast
qed

lemma analytic_at_iff_not_pole: "f analytic_on {z}  $\longleftrightarrow$   $\neg \text{is\_pole } f z$ "
  using analytic analytic_at_imp_no_pole by blast

lemma constant_or_avoid: "f = ( $\lambda_. c$ )  $\vee$  ( $\forall \approx z \in \text{UNIV}. f z \neq c$ )"
  using nicely_meromorphic_imp_constant_or_avoid[OF nicely_meromorphic,
of c] by auto

lemma isolated_zero_iff:
  assumes "f  $\neq$  ( $\lambda_. 0$ )"
  shows "isolated_zero f z  $\longleftrightarrow$   $\neg \text{is\_pole } f z \wedge f z = 0$ "
proof (cases "is_pole f z")
  case True
  thus ?thesis using pole_is_not_zero[of f z]
  by auto
next
  case False
  have " $\neg (\forall \approx z \in \text{UNIV}. f z = 0)$ "
  proof
    assume " $\forall \approx z \in \text{UNIV}. f z = 0$ "
    moreover have " $\forall \approx z \in \text{UNIV}. f z \neq 0$ "
    using constant_or_avoid[of 0] assms by auto
    ultimately have " $\forall \approx z \in (\text{UNIV} :: \text{complex set}). \text{False}$ "
    by eventually_elim auto
  end
end

```

```

      thus False
        by simp
    qed
  moreover have "f analytic_on {z}"
    using False by (subst analytic_at_iff_not_pole)
  ultimately have "isolated_zero f z  $\longleftrightarrow$  f z = 0"
    by (subst isolated_zero_analytic_iff) auto
  thus ?thesis
    using False by simp
qed
end

```

6.2 Basic results about zeros and poles

In this section we will show that an elliptic function has the same number of poles in any period parallelogram. This number is called its *order*. Then we will show that the number of zeros in a period parallelogram is also equal to its order, and that there are no elliptic functions with order 1 and no non-constant elliptic functions with order 0.

```

context elliptic_function
begin

```

Due to its meromorphicity and the fact that the period parallelograms are bounded, an elliptic function can only have a finite number of poles and zeros in a period parallelogram.

```

lemma finite_poles_in_parallelogram: "finite {z $\in$ period_parallelogram
orig. is_pole f z}"
proof (rule finite_subset)
  show "finite (closure (period_parallelogram orig)  $\cap$  {z. is_pole f z})"
  proof (rule finite_not_islimpt_in_compact)
    show " $\neg$ z islimpt {z. is_pole f z}" for z
      by (meson UNIV_I meromorphic meromorphic_on_isolated_singularity
meromorphic_on_subset not_islimpt_poles subsetI)
  qed auto
qed (use closure_subset in auto)

```

```

lemma finite_zeros_in_parallelogram: "finite {z $\in$ period_parallelogram
orig. isolated_zero f z}"
proof (rule finite_subset)
  show "finite (closure (period_parallelogram orig)  $\cap$  {z. isolated_zero
f z})"
  proof (rule finite_not_islimpt_in_compact)
    show " $\neg$ z islimpt {z. isolated_zero f z}" for z
      using meromorphic_on_imp_not_zero_cosparses[OF meromorphic]
      by (auto simp: eventually_cosparses_open_eq islimpt_iff_eventually)
  qed auto
qed (use closure_subset in auto)

```

The *order* of an elliptic function is the number of its poles inside a period parallelogram, with multiplicity taken into account. We will later show that this is also the number of zeros.

```

definition (in complex_lattice) elliptic_order :: "(complex  $\Rightarrow$  complex)
 $\Rightarrow$  nat" where
  "elliptic_order f = ( $\sum$  z | z  $\in$  period_parallelogram 0  $\wedge$  is_pole f z.
  nat (-zorder f z))"

```

```

lemma elliptic_order_const [simp]: "elliptic_order ( $\lambda$ x. c) = 0"
  by (simp add: elliptic_order_def)

```

```

lemma poles_eq_elliptic_order:
  "( $\sum$  z | z  $\in$  period_parallelogram orig  $\wedge$  is_pole f z. nat (-zorder f
  z)) = elliptic_order f"
proof -
  define h where "h = to_fund_parallelogram"
  define P where "P = period_parallelogram"
  have "( $\sum$  z  $\in$  {z $\in$ P orig. is_pole f z}. nat (-zorder f (h z))) =
  ( $\sum$  z  $\in$  {z $\in$ P 0. is_pole f z}. nat (-zorder f z))"
  proof (rule sum_reindex_bij_betw)
    show "bij_betw h {z  $\in$  P orig. is_pole f z} {z  $\in$  P 0. is_pole f z}"
    proof (rule bij_betw_Collect)
      show "bij_betw h (P orig) (P 0)"
      unfolding h_def P_def by (rule bij_betw_to_fund_parallelogram)
    next
      fix z
      show "is_pole f (h z)  $\longleftrightarrow$  is_pole f z"
      unfolding h_def by (simp add: poles.lattice_cong)
    qed
  qed
  also have "( $\sum$  z  $\in$  {z $\in$ P orig. is_pole f z}. nat (-zorder f (h z))) =
  ( $\sum$  z  $\in$  {z $\in$ P orig. is_pole f z}. nat (-zorder f z))"
    using zorder.lattice_cong[of "h z" z for z] by (simp add: h_def)
  finally show ?thesis
    by (simp add: elliptic_order_def P_def)
qed

```

end

```

context nicely_elliptic_function
begin

```

The order of a (nicely) elliptic function is zero iff it is constant. We will later lift this to non-nicely elliptic functions, where we get that the order is zero iff the function is *mostly* constant (i.e. constant except for a sparse set).

In combination with our other results relating *elliptic_order* to the number

of zeros and poles inside period parallelograms, this corresponds to Theorems 1.4 and 1.5 in Apostol's book.

lemma *elliptic_order_eq_0_iff*: "elliptic_order f = 0 \longleftrightarrow f constant_on UNIV"

proof

assume "f constant_on UNIV"

then obtain c where f_eq: "f = (λ _. c)"

by (auto simp: constant_on_def)

have [simp]: " \neg is_pole f z" for z

unfolding f_eq by auto

show "elliptic_order f = 0"

by (simp add: elliptic_order_def)

next

assume "elliptic_order f = 0"

define P where "P = period_parallelogram 0"

have "eventually (λ z. \neg is_pole f z) (cosparse UNIV)"

using meromorphic by (rule meromorphic_on_imp_not_pole_cosparse)

hence "{z. is_pole f z} sparse_in UNIV"

by (simp add: eventually_cosparse)

hence fin: "finite (closure P \cap {z. is_pole f z})"

by (intro sparse_in_compact_finite) (use sparse_in_subset in <auto simp: P_def>)

from <elliptic_order f = 0> have " $\forall z \in \{z \in P. \text{is_pole } f \ z\}. \text{zorder } f \ z \geq 0$ "

unfolding elliptic_order_def

proof (subst (asm) sum_nonneg_eq_0_iff)

show "finite {z \in period_parallelogram 0. is_pole f z}"

by (rule finite_subset[OF _ fin]) (use closure_subset in <auto simp: P_def>)

qed (auto simp: P_def)

moreover have " $\forall z \in \{z \in P. \text{is_pole } f \ z\}. \text{zorder } f \ z < 0$ "

proof safe

fix z assume "is_pole f z"

have "isolated_singularity_at f z"

using meromorphic by (simp add: meromorphic_on_altdef)

with <is_pole f z> show "zorder f z < 0"

using isolated_pole_imp_neg_zorder by blast

qed

ultimately have no_poles_P: " \neg is_pole f z" if "z \in P" for z

using that by force

have no_poles [simp]: " \neg is_pole f z" for z

proof -

have " \neg is_pole f (to_fund_parallelogram z)"

by (intro no_poles_P) (auto simp: P_def)

also have "is_pole f (to_fund_parallelogram z) \longleftrightarrow is_pole f z"

using poles.lattice_cong_rel_to_fund_parallelogram_left by blast

finally show ?thesis .

```

qed

have ana: "f analytic_on UNIV"
  using nicely_meromorphic by (rule nicely_meromorphic_without_singularities)
auto
  have "continuous_on A f" for A
    by (intro analytic_imp_holomorphic holomorphic_on_imp_continuous_on
analytic_on_subset[OF ana])
    auto
  hence "compact (f ` closure P)"
    by (rule compact_continuous_image) (auto simp: P_def)
  hence "bounded (f ` closure P)"
    by (rule compact_imp_bounded)
  hence "bounded (f ` P)"
    by (rule bounded_subset) (use closure_subset in auto)
  also have "f ` P = f ` UNIV"
  proof safe
    fix z show "f z ∈ f ` P"
      by (rule image_eqI[of _ _ "to_fund_parallelogram z"]) (auto simp:
P_def lattice_cong)
  qed auto
  finally show "f constant_on UNIV"
    by (intro Liouville_theorem) (auto intro!: analytic_imp_holomorphic
ana)
qed

```

```

lemma order_pos_iff: "elliptic_order f > 0 ↔ ¬f constant_on UNIV"
  using elliptic_order_eq_0_iff by linarith

```

The following lemma allows us to evaluate an integral of the form $\int_P h(w)f'(w)/f(w) dw$ more easily, where P is the path along the border of a period parallelogram.

Note that this only works if there are no zeros or pole on the border of the parallelogram.

```

lemma argument_principle_f_gen:
  fixes orig :: complex
  defines "γ ≡ parallelogram_path orig ω1 ω2"
  assumes h: "h holomorphic_on UNIV"
  assumes nz: "∧z. z ∈ path_image γ ⇒ f z ≠ 0 ∧ ¬is_pole f z"
  shows "contour_integral γ (λx. h x * deriv f x / f x) =
    contour_integral (linepath orig (orig + ω1))
      (λz. (h z - h (z + ω2)) * deriv f z / f z) -
    contour_integral (linepath orig (orig + ω2))
      (λz. (h z - h (z + ω1)) * deriv f z / f z)"
  proof -
    define h' where "h' = (λx. h (x + orig) * deriv f (x + orig) / f (x
+ orig))"
    have [analytic_intros]: "h analytic_on A" for A
      using h analytic_on_holomorphic by blast
    have f_analytic: "f analytic_on A" if "∧z. z ∈ A ⇒ ¬is_pole f z"

```

```

for A
  by (rule nicely_meromorphic_without_singularities)
    (use that in <auto intro!: nicely_meromorphic_on_subset[OF nicely_meromorphic]>)

  have "contour_integral  $\gamma$  ( $\lambda x. h x * deriv f x / f x$ ) =
        contour_integral (linepath 0  $\omega 1$ ) ( $\lambda x. h' x - h' (x + \omega 2)$ ) -
        contour_integral (linepath 0  $\omega 2$ ) ( $\lambda x. h' x - h' (x + \omega 1)$ )"
(is "_ = ?rhs")
  unfolding  $\gamma\_def$ 
  proof (subst contour_integral_parallelogram_path'; (fold  $\gamma\_def$ )?)
    have "( $\lambda x. h x * deriv f x / f x$ ) analytic_on {z. f z  $\neq$  0  $\wedge$   $\neg$ is_pole f z}"
      by (auto intro!: analytic_intros f_analytic)
    hence "( $\lambda x. h x * deriv f x / f x$ ) analytic_on path_image  $\gamma$ "
      by (rule analytic_on_subset) (use nz in auto)
    thus "continuous_on (path_image  $\gamma$ ) ( $\lambda x. h x * deriv f x / f x$ )" using nz
  by (intro continuous_intros holomorphic_on_imp_continuous_on analytic_imp_holomorphic)
next
  have 1: "linepath orig (orig +  $\omega 1$ ) = (+) orig  $\circ$  linepath 0  $\omega 1$ "
    and 2: "linepath orig (orig +  $\omega 2$ ) = (+) orig  $\circ$  linepath 0  $\omega 2$ "
    by (auto simp: linepath_translate add_ac)
  show "contour_integral (linepath orig (orig +  $\omega 1$ ))
        ( $\lambda x. h x * deriv f x / f x - h (x + \omega 2) * deriv f (x + \omega 2)$ )
/ f (x +  $\omega 2$ ) -
        contour_integral (linepath orig (orig +  $\omega 2$ ))
        ( $\lambda x. h x * deriv f x / f x - h (x + \omega 1) * deriv f (x + \omega 1)$ )
/ f (x +  $\omega 1$ ) = ?rhs"
    unfolding 1 2 contour_integral_translate h'_def by (simp add: algebra_simps)
  qed

  also have "( $\lambda z. h' z - h' (z + \omega 1)$ ) =
        ( $\lambda z. (h (z + orig) - h (z + orig + \omega 1)) * deriv f (z + orig)$ )
/ f (z + orig)"
    (is "?lhs = ?rhs")
  proof
    fix z :: complex
    have "h' z - h' (z +  $\omega 1$ ) =
          h (z + orig) * deriv f (z + orig) / f (z + orig) -
          h (z + orig +  $\omega 1$ ) * deriv f (z + orig +  $\omega 1$ ) / f (z + orig
+  $\omega 1$ )"
      by (simp add: h'_def add_ac)
    also have "... = (h (z + orig) - h (z + orig +  $\omega 1$ )) * deriv f (z +
orig) / f (z + orig)"
      unfolding f_periodic deriv.f_periodic by (simp add: diff_divide_distrib
ring_distrib)
    finally show "?lhs z = ?rhs z" .
  qed
  also have "contour_integral (linepath 0  $\omega 2$ ) ... =

```

```

      contour_integral ((+) orig ◦ linepath 0 ω2) (λz. (h z -
h (z + ω1)) * deriv f z / f z)"
    by (rule contour_integral_translate [symmetric])
  also have "(+) orig ◦ linepath 0 ω2 = linepath orig (orig + ω2)"
    by (subst linepath_translate) (simp_all add: add_ac)

  also have "contour_integral (linepath 0 ω1) (λz. h' z - h' (z + ω2))
=
      contour_integral (linepath 0 ω1)
      (λz. (h (z + orig) - h (z + orig + ω2)) * deriv f (z +
orig) / f (z + orig))"
    (is "contour_integral _ ?lhs = contour_integral _ ?rhs")
  proof (rule contour_integral_cong)
    fix z :: complex
    assume z: "z ∈ path_image (linepath 0 ω1)"
    hence "orig + z ∈ path_image ((+) orig ◦ linepath 0 ω1)"
      by (subst path_image_compose) auto
    also have "(+) orig ◦ linepath 0 ω1 = linepath orig (orig + ω1)"
      by (subst linepath_translate) (auto simp: add_ac)
    finally have "orig + z ∈ path_image γ"
      unfolding γ_def parallelogram_path_def by (auto simp: path_image_join)
    hence nz': "f (orig + z) ≠ 0"
      using nz by blast

    have "h' z - h' (z + ω2) =
      h (z + orig) * deriv f (z + orig) / f (z + orig) -
      h (z + orig + ω2) * (deriv f (z + orig + ω2) / f (z + orig
+ ω2))"
      by (simp add: h'_def add_ac)
    also have "deriv f (z + orig + ω2) / f (z + orig + ω2) =
      deriv f (z + orig) / f (z + orig)"
      unfolding f_periodic deriv.f_periodic using nz'
      by (auto simp: divide_simps add_ac)
    also have "h (z + orig) * deriv f (z + orig) / f (z + orig) - h (z
+ orig + ω2) * ... = ?rhs z"
      by (simp add: ring_distrib diff_divide_distrib)
    finally show "?lhs z = ?rhs z" .
  qed auto
  also have "... = contour_integral ((+) orig ◦ linepath 0 ω1)
      (λz. (h z - h (z + ω2)) * deriv f z / f z)"
    unfolding contour_integral_translate by (simp add: add_ac)
  also have "(+) orig ◦ linepath 0 ω1 = linepath orig (orig + ω1)"
    by (subst linepath_translate) (simp_all add: add_ac)

  finally show ?thesis .
qed

```

Using our lemma with $h(z) = 1$, we immediately get the fact that the integral over $f'(z)/f(z)$ vanishes.

```

lemma argument_principle_f_1:
  fixes orig :: complex
  defines "γ ≡ parallelogram_path orig ω1 ω2"
  assumes nz: "∧z. z ∈ path_image γ ⇒ f z ≠ 0 ∧ ¬is_pole f z"
  shows "contour_integral (parallelogram_path orig ω1 ω2) (λx. deriv
f x / f x) = 0"
  using argument_principle_f_gen[of "λ_. 1" orig] nz by (simp add: γ_def)

```

Using our lemma with $h(z) = z$, we see that the integral over $zf'(z)/f(z)$ does not vanish, but it is of the form $2\pi i\omega$, where $\omega \in \Lambda$.

```

lemma argument_principle_f_z:
  fixes orig :: complex
  defines "γ ≡ parallelogram_path orig ω1 ω2"
  assumes wf: "∧z. z ∈ path_image γ ⇒ f z ≠ 0 ∧ ¬is_pole f z"
  shows "contour_integral γ (λz. z * deriv f z / f z) / (2*pi*i) ∈ Λ"
proof -
  note [holomorphic_intros del] = holomorphic_deriv
  note [holomorphic_intros] = holomorphic holomorphic_on_subset[OF holomorphic_deriv[of
_ UNIV]]
  define γ1 where "γ1 = linepath orig (orig + ω1)"
  define γ2 where "γ2 = linepath orig (orig + ω2)"
  define γ1' where "γ1' = f ∘ γ1"
  define γ2' where "γ2' = f ∘ γ2"
  have path_image_subset: "path_image γ1 ⊆ path_image γ" "path_image
γ2 ⊆ path_image γ"
    by (auto simp: γ_def γ1_def γ2_def path_image_join closed_segment_commute
parallelogram_path_def)

  have "pathstart γ ∈ path_image γ"
    by blast
  from wf[OF this] have [simp]: "f orig ≠ 0"
    by (simp add: γ_def)
  have [simp, intro]: "valid_path γ1" "valid_path γ2"
    by (simp_all add: γ1_def γ2_def)
  have [simp, intro]: "valid_path γ1'" "valid_path γ2'"
    unfolding γ1'_def γ2'_def using wf path_image_subset
    by (auto intro!: valid_path_compose_analytic[of _ _ "path_image γ"]
analytic)
  have [simp]: "pathstart γ1' = f orig" "pathfinish γ1' = f (orig + ω1)"
    "pathstart γ2' = f orig" "pathfinish γ2' = f (orig + ω2)"
    by (simp_all add: γ1'_def γ1_def γ2'_def γ2_def pathstart_compose
pathfinish_compose)

  have wf': "f z ≠ 0 ∧ ¬is_pole f z" if "z ∈ path_image γ1 ∪ path_image
γ2" for z
  proof -
    note <z ∈ path_image γ1 ∪ path_image γ2>
    also have "path_image γ1 ∪ path_image γ2 ⊆ path_image γ"
      using path_image_subset by blast

```

```

    finally show ?thesis
      using wf[of z] by blast
qed
have [simp]: "0 ∉ path_image γ1'" "0 ∉ path_image γ2'"
  using wf' by (auto simp: γ1'_def γ2'_def path_image_compose)

```

The actual proof starts here.

```

define I1 where "I1 = contour_integral γ1"
define I2 where "I2 = contour_integral γ2"

have "winding_number γ1' 0 ∈ ℤ"
  by (rule integer_winding_number) (auto intro!: valid_path_imp_path
simp: f_periodic)
then obtain m where m: "winding_number γ1' 0 = of_int m"
  by (elim Ints_cases)

have "winding_number γ2' 0 ∈ ℤ"
  by (rule integer_winding_number) (auto intro!: valid_path_imp_path
simp: f_periodic)
then obtain n where n: "winding_number γ2' 0 = of_int n"
  by (elim Ints_cases)

have "contour_integral γ (λz. z * deriv f z / f z) / (2*pi*i) =
      (I1 (λz. (-ω2) * (deriv f z / f z)) - I2 (λz. (-ω1) * (deriv
f z / f z))) / (2*pi*i)"
  unfolding γ_def I1_def I2_def γ1_def γ2_def using wf
  by (subst argument_principle_f_gen) (auto simp: diff_divide_distrib
γ_def)
also have "I1 (λz. (-ω2) * (deriv f z / f z)) - I2 (λz. (-ω1) * (deriv
f z / f z)) =
      (-ω2) * I1 (λz. deriv f z / f z) - (-ω1) * I2 (λz. deriv
f z / f z)"
  using wf' unfolding I1_def I2_def γ1_def γ2_def
  by (subst (1 2) contour_integral_lmul)
      (auto intro!: contour_integrable_continuous_linepath holomorphic_on_imp_continuous_on
analytic_imp_holomorphic analytic_intros analytic)
also have "... = ω1 * contour_integral γ2' (λz. 1 / z) -
      ω2 * contour_integral γ1' (λz. 1 / z)"
  unfolding I1_def I2_def γ1'_def γ2'_def using wf'
  by (subst (1 2) contour_integral_comp_analyticW[OF analytic[of "path_image
γ1 ∪ path_image γ2"]])
      (auto simp: γ1_def)
also have "... / (2 * pi * i) = ω1 * winding_number γ2' 0 - ω2 * winding_number
γ1' 0"
  by (subst (1 2) winding_number_valid_path)
      (auto intro!: valid_path_compose_analytic analytic simp: diff_divide_distrib)
also have "... = of_int n * ω1 - of_int m * ω2"
  by (auto simp: m n)
also have "... ∈ Λ"

```

```

    by (auto intro!: lattice_intros)
  finally show ?thesis .
qed

```

By using the fact that the integral $f'(z)/f(z)$ along the border of a period parallelogram vanishes, we get the following fact: The number of zeros in the period parallelogram equals the number of poles, i.e. the order.

The only difficulty left here is to show that 1. the number of zeros is invariant under which period parallelogram we choose, and 2. there is a period parallelogram whose borders do not contain any zeros or poles.

This is essentially Theorem 1.8 in Apostol's book.

lemma `zeros_eq_elliptic_order_aux`:

```

  "( $\sum z \mid z \in \text{period\_parallelogram orig} \wedge \text{isolated\_zero } f z. \text{nat } (\text{zorder } f z)$ ) = elliptic_order f"

```

proof (cases "f = ($\lambda_. 0$)")

```

  case True

```

```

  hence "elliptic_order f = 0"

```

```

    by (subst elliptic_order_eq_0_iff) (auto simp: constant_on_def)

```

```

  moreover have " $\neg \text{isolated\_zero } f z$ " for z

```

```

    by (simp add: isolated_zero_def True)

```

```

  ultimately show ?thesis

```

```

    by simp

```

```

next

```

```

  case False

```

```

  define s where "s = complex_of_real (sgn (Im ( $\omega_2 / \omega_1$ )))"

```

```

  have [simp]: "s  $\neq$  0"

```

```

    using fundpair by (auto simp: s_def sgn_if fundpair_def complex_is_Real_iff)

```

```

  have ev_nonzero: "eventually ( $\lambda z. f z \neq 0$ ) (cosparse UNIV)"

```

```

    using nicely_meromorphic False nicely_meromorphic_imp_constant_or_avoid[of f UNIV 0]

```

```

    by auto

```

```

  have isolated_zero_iff: " $\text{isolated\_zero } f z \iff \neg \text{is\_pole } f z \wedge f z = 0$ " for z

```

```

    using isolated_zero_iff[OF False] by blast

```

```

  define P where "P = period_parallelogram"

```

```

  define avoid where "avoid = {z. isolated_zero f z  $\vee$  is_pole f z}"

```

```

  have sparse_avoid: " $\neg z \text{ islimpt avoid}$ " for z

```

```

  proof -

```

```

    have " $\forall \approx z \in \text{UNIV}. \neg \text{isolated\_zero } f z \wedge \neg \text{is\_pole } f z$ " using meromorphic

```

```

      by (intro meromorphic_on_imp_not_zero_cosparse

```

```

          meromorphic_on_imp_not_pole_cosparse eventually_conj)

```

```

    hence " $\forall \approx z \in \text{UNIV}. z \notin \text{avoid}$ "

```

```

      by eventually_elim (auto simp: avoid_def)

```

```

  thus ?thesis

```

```

    by (auto simp: eventually_cosparse_open_eq islimpt_iff_eventually)

```

```

qed
thus ?thesis
  unfolding P_def [symmetric]
proof (induction orig rule: shifted_period_parallelogram_avoid_wlog)
  case (shift orig d)
  obtain h where h: "bij_betw h (P (orig + d)) (P orig)" "\z. rel
(h z) z"
  using bij_betw_period_parallelograms unfolding P_def by blast
  have "( $\sum z \mid z \in P (orig + d) \wedge isolated\_zero f z. nat (zorder f
(h z))) =$ 
( $\sum z \mid z \in P orig \wedge isolated\_zero f z. nat (zorder f z)$ )"
  by (rule sum.reindex_bij_betw, rule bij_betw_Collect[OF h(1)])
  (auto simp: zeros.lattice_cong[OF h(2)])
  also have "( $\sum z \mid z \in P (orig + d) \wedge isolated\_zero f z. nat (zorder
f (h z))) =$ 
( $\sum z \mid z \in P (orig + d) \wedge isolated\_zero f z. nat (zorder
f z)$ )"
  by (simp add: zorder.lattice_cong[OF h(2)])
  finally show ?case
  using shift by simp
next
case (avoid orig)
define  $\gamma$  where " $\gamma = parallelogram\_path orig \omega1 \omega2$ "
define zeros where " $zeros = \{z \in P orig. isolated\_zero f z\}$ "
define poles where " $poles = \{z \in P orig. is\_pole f z\}$ "
have  $\gamma$ : " $\neg isolated\_zero f z \wedge \neg is\_pole f z$ " if " $z \in path\_image \gamma$ "
for z
  using avoid that by (auto simp: avoid_def  $\gamma\_def$ )
  have "compact (closure (P orig))"
  unfolding P_def by auto
  then obtain R where R: " $closure (P orig) \subseteq ball 0 R$ "
  using compact_imp_bounded bounded_subset_ballD by blast
  define A :: "complex set" where "A = ball 0 R"
  have A: " $closure (P orig) \subseteq A$ "
  using R by (simp add: A_def)

  have fin: "finite {w  $\in$  A. f w = 0  $\vee$  w  $\in$  {z. is_pole f z}}"
  proof (rule finite_subset)
  show "finite (cball 0 R  $\cap$  avoid)"
  by (rule finite_not_islimpt_in_compact) (use sparse_avoid in auto)
  next
  show "{w  $\in$  A. f w = 0  $\vee$  w  $\in$  {z. is_pole f z}}  $\subseteq$  cball 0 R  $\cap$  avoid"
  by (auto simp: avoid_def A_def isolated_zero_iff)
  qed

  have "contour_integral  $\gamma$  ( $\lambda x. deriv f x * 1 / f x$ ) =
of_real (2 * pi) * i * ( $\sum p \in \{w \in A. f w = 0 \vee w \in \{z. is\_pole
f z\}\}. winding\_number \gamma p * 1 * of\_int (zorder f p)$ )"

```

```

proof (rule argument_principle)
  show "open A" "connected A"
    by (auto simp: A_def)
next
  have "f analytic_on A - {z. is_pole f z}"
    using nicely_meromorphic_on_subset[OF nicely_meromorphic]
    by (rule nicely_meromorphic_without_singularities) auto
  thus "f holomorphic_on A - {z. is_pole f z}"
    by (rule analytic_imp_holomorphic)
next
  show "path_image  $\gamma \subseteq A - \{w \in A. f w = 0 \vee w \in \{z. is\_pole f z\}\}"
    using A  $\gamma$  path_image_parallelogram_subset_closure[of orig]
    by (auto simp:  $\gamma\_def$  P_def isolated_zero_iff)
next
  show "finite {w \in A. f w = 0 \vee w \in \{z. is_pole f z\}"
    by (rule finite_subset[OF _ fin]) auto
next
  show " $\forall z. z \notin A \longrightarrow winding\_number \gamma z = 0$ "
    using A unfolding  $\gamma\_def$  P_def by (auto intro!: winding_number_parallelogram_outside)
qed (auto simp:  $\gamma\_def$ )
also have "contour_integral  $\gamma (\lambda x. deriv f x * 1 / f x) = 0$ "
  using argument_principle_f_1[of orig]  $\gamma$  by (auto simp:  $\gamma\_def$  isolated_zero_iff)
finally have " $(\sum z \in \{z \in A. f z = 0 \vee is\_pole f z\}. winding\_number \gamma z * of\_int (zorder f z)) = 0$ "
  by simp

  also have " $(\sum z \in \{z \in A. f z = 0 \vee is\_pole f z\}. winding\_number \gamma z * of\_int (zorder f z)) =$ 
     $(\sum z \in \{z \in P \text{ orig}. f z = 0 \vee is\_pole f z\}. s * of\_int (zorder f z))$ "
proof (intro sum.mono_neutral_cong_right ballI, goal_cases)
  case (3 z)
  hence " $z \notin P \text{ orig} \cup frontier (P \text{ orig})$ "
    using  $\gamma$ [of z] by (auto simp: isolated_zero_iff P_def  $\gamma\_def$  path_image_parallelogram)
  also have " $P \text{ orig} \cup frontier (P \text{ orig}) = closure (P \text{ orig})$ "
    using closure_Un_frontier by blast
  finally have " $winding\_number \gamma z = 0$ "
    unfolding  $\gamma\_def$  P_def using winding_number_parallelogram_outside
by blast
  thus ?case
    by simp
next
  case (4 z)
  hence " $z \in P \text{ orig} - frontier (P \text{ orig})$ "
    using  $\gamma$ [of z] by (auto simp: isolated_zero_iff P_def  $\gamma\_def$  path_image_parallelogram)
  also have " $P \text{ orig} - frontier (P \text{ orig}) = interior (P \text{ orig})$ "
    using interior_subset closure_subset by (auto simp: frontier_def)
  finally have " $winding\_number \gamma z = s$ "$ 
```

```

    unfolding s_def  $\gamma$ _def P_def by (rule winding_number_parallelogram_inside)
  thus ?case
    by simp
qed (use A fin closure_subset in auto)
also have "{z $\in$ P orig. f z = 0  $\vee$  is_pole f z} = zeros  $\cup$  poles"
  by (auto simp: zeros_def poles_def isolated_zero_iff)
also have "(\sum z $\in$ zeros  $\cup$  poles. s * complex_of_int (zorder f z)) =
  s * of_int (\sum z $\in$ zeros  $\cup$  poles. zorder f z)"
  by (simp add: sum_distrib_left)
finally have "(\sum z $\in$ zeros  $\cup$  poles. zorder f z) = 0"
  by (simp del: of_int_sum)

  also have "(\sum z $\in$ zeros  $\cup$  poles. zorder f z) = (\sum z $\in$ zeros. zorder
f z) + (\sum z $\in$ poles. zorder f z)"
  by (subst sum.union_disjoint)
  (use A closure_subset
    in <auto simp: zeros_def poles_def isolated_zero_iff intro!:
finite_subset[OF _ fin]> )
  also have "(\sum z $\in$ zeros. zorder f z) = (\sum z $\in$ zeros. int (nat (zorder
f z)))"
  proof (intro sum.cong)
    fix z assume z: "z  $\in$  zeros"
    hence " $\neg$ is_pole f z"
      by (auto simp: zeros_def isolated_zero_iff)
    hence "f analytic_on {z}"
      using nicely_meromorphic nicely_meromorphic_on_imp_analytic_at
by blast
    hence "zorder f z > 0"
      using z by (intro zorder_isolated_zero_pos) (auto simp: zeros_def)
    thus "zorder f z = int (nat (zorder f z))"
      by simp
  qed auto
  also have "... = int (\sum z $\in$ zeros. nat (zorder f z))"
    by simp

  also have "(\sum z $\in$ poles. zorder f z) = (\sum z $\in$ poles. -int (nat (-zorder
f z)))"
  proof (intro sum.cong)
    fix z assume "z  $\in$  poles"
    hence "zorder f z < 0" using meromorphic
      by (auto intro!: isolated_pole_imp_neg_zorder simp: poles_def
meromorphic_on_altdef)
    thus "zorder f z = - int (nat (- zorder f z))"
      by simp
  qed auto
  also have "... = -int (\sum z $\in$ poles. nat (-zorder f z))"
    by (simp add: sum_negf)

  also have "(\sum z $\in$ poles. nat (-zorder f z)) = elliptic_order f"

```

```

    using poles_eq_elliptic_order[of orig] by (simp add: poles_def P_def)
  finally have "( $\sum z \in \text{zeros. nat (zorder } f \ z)) = \text{elliptic\_order } f$ "
    by linarith
  thus ?case
    by (simp add: zeros_def)
qed
qed

```

In the same vein, we get the following from our earlier result about the integral over $zf'(z)/f(z)$: The sum over all zeros and poles (counted with multiplicity, where poles have negative multiplicity) in a period parallelogram is a lattice point.

This is Exercise 1.2 in Apostol's book.

lemma *sum_zeros_poles_in_lattice_aux*:

```

  defines "Z  $\equiv (\lambda \text{orig. } \{z \in \text{period\_parallelogram } \text{orig. } \text{isolated\_zero } f \ z \vee \text{is\_pole } f \ z\})$ "
  defines "S  $\equiv (\lambda \text{orig. } \sum z \in Z \ \text{orig. } \text{of\_int } (zorder \ f \ z) * z)$ "
  shows "S orig  $\in \Lambda$ "
proof (cases "f = ( $\lambda \_.$  0)")
  case True
  hence "elliptic_order f = 0"
    by (subst elliptic_order_eq_0_iff) (auto simp: constant_on_def)
  moreover have " $\neg \text{isolated\_zero } f \ z$ " " $\neg \text{is\_pole } f \ z$ " for z
    by (auto simp: isolated_zero_def True)
  ultimately show ?thesis
    by (simp add: S_def Z_def)
next
  case False
  define s where "s = complex_of_real (sgn (Im ( $\omega_2 / \omega_1$ )))"
  have s: "s  $\in \{-1, 1\}$ "
    using fundpair by (auto simp: s_def sgn_if fundpair_def complex_is_Real_iff)

  have ev_nonzero: "eventually ( $\lambda z. f \ z \neq 0$ ) (cosparsE UNIV)"
    using nicely_meromorphic False nicely_meromorphic_imp_constant_or_avoid[of f UNIV 0]
    by auto

  have isolated_zero_iff: "isolated_zero f z  $\longleftrightarrow \neg \text{is\_pole } f \ z \wedge f \ z = 0$ " for z
proof
  assume z: " $\neg \text{is\_pole } f \ z \wedge f \ z = 0$ "
  hence "f analytic_on {z}"
    using nicely_meromorphic by (simp add: nicely_meromorphic_on_imp_analytic_at)
  with z have "f  $\rightarrow 0$ "
    by (metis analytic_at_imp_isCont continuous_within)
  moreover have "eventually ( $\lambda z. f \ z \neq 0$ ) (at z)"
    using ev_nonzero by (subst (asm) eventually_cosparsE_open_eq) auto
  ultimately show "isolated_zero f z"
    unfolding isolated_zero_def by blast

```

```

next
  assume "isolated_zero f z"
  thus "¬is_pole f z ∧ f z = 0"
    by (meson iso_tuple_UNIV_I nicely_meromorphic
        nicely_meromorphic_on_imp_analytic_at pole_is_not_zero
        zero_isolated_zero_analytic)
qed

define P where "P = period_parallelogram"
define avoid where "avoid = {z. isolated_zero f z ∨ is_pole f z}"
have sparse_avoid: "¬z islimpt avoid" for z
proof -
  have "∀z∈UNIV. ¬isolated_zero f z ∧ ¬is_pole f z" using meromorphic

    by (intro meromorphic_on_imp_not_zero_cosparse
        meromorphic_on_imp_not_pole_cosparse eventually_conj)
  hence "∀z∈UNIV. z ∉ avoid"
    by eventually_elim (auto simp: avoid_def)
  thus ?thesis
    by (auto simp: eventually_cosparse_open_eq islimpt_iff_eventually)
qed
thus ?thesis
  unfolding P_def [symmetric]
proof (induction orig rule: shifted_period_parallelogram_avoid_wlog)
  case (shift orig d)
  obtain h where h: "bij_betw h (P (orig + d)) (P orig)" "∧z. rel
(h z) z"
    using bij_betw_period_parallelograms unfolding P_def by blast
  have h': "bij_betw h (Z (orig + d)) (Z orig)"
    unfolding Z_def
    by (rule bij_betw_Collect)
    (use h(1) zeros.lattice_cong[OF h(2)] poles.lattice_cong[OF h(2)])
in <auto simp: P_def>

  have "S (orig + d) = (∑ z∈Z (orig + d). of_int (zorder f z) * z)"
    by (simp add: S_def)
  also have "rel (∑ z∈Z (orig + d). of_int (zorder f z) * z)
(∑ z∈Z (orig + d). of_int (zorder f z) * h z)"
    by (intro lattice_intros) (use h(2) in <auto simp: rel_sym>)
  also have "... = (∑ z∈Z (orig + d). of_int (zorder f (h z)) * h z)"
    by (simp add: zorder.lattice_cong[OF h(2)])
  also have "... = (∑ z∈Z orig. of_int (zorder f z) * z)"
    using h' by (rule sum.reindex_bij_betw)
  also have "... = S orig"
    by (simp add: S_def)
  also have "S orig ∈ Λ"
    by fact
  finally show ?case .
next

```

```

case (avoid orig)
define  $\gamma$  where " $\gamma = \text{parallelogram\_path orig } \omega_1 \omega_2$ "
define zeros where " $\text{zeros} = \{z \in P \text{ orig. isolated\_zero } f \ z\}$ "
define poles where " $\text{poles} = \{z \in P \text{ orig. is\_pole } f \ z\}$ "
have  $\gamma$ : " $\neg \text{isolated\_zero } f \ z \wedge \neg \text{is\_pole } f \ z$ " if " $z \in \text{path\_image } \gamma$ "
for z
  using avoid that by (auto simp: avoid_def  $\gamma$ _def)
  have "compact (closure (P orig))"
  unfolding P_def by auto
  then obtain R where R: " $\text{closure } (P \text{ orig}) \subseteq \text{ball } 0 \ R$ "
  using compact_imp_bounded bounded_subset_ballD by blast
  define A :: "complex set" where "A = ball 0 R"
  have A: " $\text{closure } (P \text{ orig}) \subseteq A$ "
  using R by (simp add: A_def)

  have fin: "finite  $\{w \in A. f \ w = 0 \vee w \in \{z. \text{is\_pole } f \ z\}\}$ "
  proof (rule finite_subset)
    show "finite (cball 0 R  $\cap$  avoid)"
    by (rule finite_not_islimpt_in_compact) (use sparse_avoid in auto)
  next
    show " $\{w \in A. f \ w = 0 \vee w \in \{z. \text{is\_pole } f \ z\}\} \subseteq \text{cball } 0 \ R \cap \text{avoid}$ "
    by (auto simp: avoid_def A_def isolated_zero_iff)
  qed

  have "contour_integral  $\gamma$  ( $\lambda x. \text{deriv } f \ x * x / f \ x$ ) =
    of_real (2 * pi) * i * ( $\sum_{p \in \{w \in A. f \ w = 0 \vee w \in \{z. \text{is\_pole } f \ z\}\}.}$ 
      winding_number  $\gamma$  p * p * of_int (zorder f p))"
  proof (rule argument_principle)
    show "open A" "connected A"
    by (auto simp: A_def)
  next
    have "f analytic_on A -  $\{z. \text{is\_pole } f \ z\}$ "
    using nicely_meromorphic_on_subset[OF nicely_meromorphic]
    by (rule nicely_meromorphic_without_singularities) auto
    thus "f holomorphic_on A -  $\{z. \text{is\_pole } f \ z\}$ "
    by (rule analytic_imp_holomorphic)
  next
    show "path_image  $\gamma \subseteq A - \{w \in A. f \ w = 0 \vee w \in \{z. \text{is\_pole } f \ z\}\}$ "
    using A  $\gamma$  path_image_parallelogram_subset_closure[of orig]
    by (auto simp:  $\gamma$ _def P_def isolated_zero_iff)
  next
    show "finite  $\{w \in A. f \ w = 0 \vee w \in \{z. \text{is\_pole } f \ z\}\}$ "
    by (rule finite_subset[OF _ fin]) auto
  next
    show " $\forall z. z \notin A \longrightarrow \text{winding\_number } \gamma \ z = 0$ "
    using A unfolding  $\gamma$ _def P_def by (auto intro!: winding_number_parallelogram_outside)
  qed (auto simp:  $\gamma$ _def)

```

```

    hence "( $\sum_{p \in \{w \in A. f w = 0 \vee \text{is\_pole } f w\}} \text{winding\_number } \gamma p * p$ 
* of_int (zorder f p)) =
      contour_integral  $\gamma (\lambda x. x * \text{deriv } f x / f x) / (2 * \text{pi} * i)$ "
    by (simp add: field_simps)
    also have "...  $\in \Lambda$ "
    unfolding  $\gamma\_def$  by (rule argument_principle_f_z) (use  $\gamma$  in <auto simp:  $\gamma\_def$  isolated_zero_iff>)

    also have "( $\sum_{z \in \{z \in A. f z = 0 \vee \text{is\_pole } f z\}} \text{winding\_number } \gamma z$ 
* z * of_int (zorder f z)) =
      ( $\sum_{z \in Z \text{ orig. } s * \text{of\_int } (zorder f z) * z$ )"
    proof (intro sum.mono_neutral_cong_right ballI, goal_cases)
      case (3 z)
      hence " $z \notin P \text{ orig} \cup \text{frontier } (P \text{ orig})$ "
      using  $\gamma$ [of z] by (auto simp: Z_def isolated_zero_iff P_def  $\gamma\_def$ 
path_image_parallelogram_path)
      also have " $P \text{ orig} \cup \text{frontier } (P \text{ orig}) = \text{closure } (P \text{ orig})$ "
      using closure_Un_frontier by blast
      finally have " $\text{winding\_number } \gamma z = 0$ "
      unfolding  $\gamma\_def$  P_def using winding_number_parallelogram_outside
    by blast
      thus ?case
      by simp
    next
      case (4 z)
      hence " $z \in P \text{ orig} - \text{frontier } (P \text{ orig})$ "
      using  $\gamma$ [of z] by (auto simp: Z_def isolated_zero_iff P_def  $\gamma\_def$ 
path_image_parallelogram_path)
      also have " $P \text{ orig} - \text{frontier } (P \text{ orig}) = \text{interior } (P \text{ orig})$ "
      using interior_subset closure_subset by (auto simp: frontier_def)
      finally have " $\text{winding\_number } \gamma z = s$ "
      unfolding s_def  $\gamma\_def$  P_def by (rule winding_number_parallelogram_inside)
      thus ?case
      by simp
    qed (use A fin closure_subset in <auto simp: Z_def P_def isolated_zero_iff>)
    also have "( $\sum_{z \in Z \text{ orig. } s * \text{of\_int } (zorder f z) * z$ ) = s * S orig"
    by (simp add: S_def sum_distrib_left mult.assoc)
    finally show "S orig  $\in \Lambda$ "
    using s by (auto simp: uminus_in_lattice_iff)
  qed
qed

```

Again, similarly: The residues in a period parallelogram sum to 0.

```

lemma sum_residues_eq_0_aux:
  defines "Q  $\equiv (\lambda \text{orig. } \{z \in \text{period\_parallelogram } \text{orig. is\_pole } f z\})$ "
  defines "S  $\equiv (\lambda \text{orig. } \sum_{z \in Q \text{ orig. } \text{residue } f z)$ "
  shows "S orig  $\in \Lambda$ "
proof (cases "f = ( $\lambda_. 0$ )")
  case True

```

```

hence "elliptic_order f = 0"
  by (subst elliptic_order_eq_0_iff) (auto simp: constant_on_def)
moreover have "¬is_pole f z" for z
  by (auto simp: isolated_zero_def True)
ultimately show ?thesis
  by (simp add: S_def Q_def)
next
case False
define s where "s = complex_of_real (sgn (Im (ω2 / ω1)))"
have s: "s ∈ {-1, 1}"
  using fundpair by (auto simp: s_def sgn_if fundpair_def complex_is_Real_iff)

have ev_nonzero: "eventually (λz. f z ≠ 0) (cosparse UNIV)"
  using nicely_meromorphic False nicely_meromorphic_imp_constant_or_avoid[of
f UNIV 0]
  by auto

define P where "P = period_parallelogram"
define avoid where "avoid = {z. is_pole f z}"
have sparse_avoid: "¬z islimpt avoid" for z
  unfolding avoid_def
  by (meson UNIV_I meromorphic meromorphic_on_isolated_singularity
meromorphic_on_subset not_islimpt_poles subsetI)
thus ?thesis
  unfolding P_def [symmetric]
proof (induction orig rule: shifted_period_parallelogram_avoid_wlog)
case (shift orig d)
obtain h where h: "bij_betw h (P (orig + d)) (P orig)" "∧z. rel
(h z) z"
  using bij_betw_period_parallelograms unfolding P_def by blast
have h': "bij_betw h (Q (orig + d)) (Q orig)"
  unfolding Q_def
  by (rule bij_betw_Collect)
  (use h(1) poles.lattice_cong[OF h(2)] in <auto simp: P_def>)

have "S (orig + d) = (∑ z∈Q (orig + d). residue f z)"
  by (simp add: S_def)
also have "... = (∑ z∈Q (orig + d). residue f (h z))"
  by (simp add: rel_sym residue.lattice_cong[OF h(2)])
also have "... = (∑ z∈Q orig. residue f z)"
  using h' by (rule sum.reindex_bij_betw)
also have "... = S orig"
  by (simp add: S_def)
also have "S orig ∈ Λ"
  by fact
finally show ?case .
next
case (avoid orig)
define γ where "γ = parallelogram_path orig ω1 ω2"

```

```

have  $\gamma$ : " $\neg$ is_pole f z" if " $z \in \text{path\_image } \gamma$ " for z
  using avoid that by (auto simp: avoid_def  $\gamma$ _def)
have "compact (closure (P orig))"
  unfolding P_def by auto
then obtain R where R: "closure (P orig)  $\subseteq$  ball 0 R"
  using compact_imp_bounded bounded_subset_ballD by blast
define A :: "complex set" where "A = ball 0 R"
have A: "closure (P orig)  $\subseteq$  A"
  using R by (simp add: A_def)

have fin: "finite {w  $\in$  A. is_pole f w}"
proof (rule finite_subset)
  show "finite (cball 0 R  $\cap$  avoid)"
    by (rule finite_not_islimpt_in_compact) (use sparse_avoid in auto)
next
  show "{z  $\in$  A. is_pole f z}  $\subseteq$  cball 0 R  $\cap$  avoid"
    by (auto simp: avoid_def A_def)
qed

have "contour_integral  $\gamma$  f =
  of_real (2 * pi) * i * ( $\sum_{p \in \{z \in A. \text{is\_pole } f z\}} \text{winding\_number } \gamma p * \text{residue } f p$ )"
proof (rule Residue_theorem)
  show "open A" "connected A"
    by (auto simp: A_def)
next
  have "f analytic_on A - {z  $\in$  A. is_pole f z}"
    using nicely_meromorphic_on_subset[OF nicely_meromorphic]
    by (rule nicely_meromorphic_without_singularities) auto
  thus "f holomorphic_on A - {z  $\in$  A. is_pole f z}"
    by (rule analytic_imp_holomorphic)
next
  show "path_image  $\gamma \subseteq$  A - {z  $\in$  A. is_pole f z}"
    using A  $\gamma$  path_image_parallelogram_subset_closure[of orig]
    by (auto simp:  $\gamma$ _def P_def)
next
  show "finite {z  $\in$  A. is_pole f z}"
    using fin by simp
next
  show " $\forall z. z \notin A \longrightarrow \text{winding\_number } \gamma z = 0$ "
    using A unfolding  $\gamma$ _def P_def by (auto intro!: winding_number_parallelogram_outside)
qed (auto simp:  $\gamma$ _def)
also have " $(\sum_{p \in \{z \in A. \text{is\_pole } f z\}} \text{winding\_number } \gamma p * \text{residue } f p) =$ 
  ( $\sum_{p \in Q \text{ orig. } s * \text{residue } f p$ )"
proof (intro sum_mono_neutral_cong_right ballI, goal_cases)
  case (3 z)
  hence "z  $\notin$  P orig  $\cup$  frontier (P orig)"
    using  $\gamma$ [of z] by (auto simp: Q_def P_def  $\gamma$ _def path_image_parallelogram_path)

```

```

    also have "P orig  $\cup$  frontier (P orig) = closure (P orig)"
      using closure_Un_frontier by blast
    finally have "winding_number  $\gamma$  z = 0"
      unfolding  $\gamma\_def$  P_def using winding_number_parallelogram_outside
  by blast
    thus ?case
      by simp
  next
    case (4 z)
    hence "z  $\in$  P orig - frontier (P orig)"
      using  $\gamma$ [of z] by (auto simp: Q_def P_def  $\gamma\_def$  path_image_parallelogram_path)
    also have "P orig - frontier (P orig) = interior (P orig)"
      using interior_subset closure_subset by (auto simp: frontier_def)
    finally have "winding_number  $\gamma$  z = s"
      unfolding s_def  $\gamma\_def$  P_def by (rule winding_number_parallelogram_inside)
    thus ?case
      by simp
  qed (use A fin closure_subset in <auto simp: Q_def P_def>)
  also have "... = s * ( $\sum_{z \in Q \text{ orig.}} \text{residue } f \ z$ )"
    by (simp add: sum_distrib_left)
  also have "contour_integral  $\gamma$  f = 0"
    using  $\gamma$  unfolding  $\gamma\_def$ 
    by (subst contour_integral_parallelogram_path')
      (auto simp: f_periodic intro!: continuous_on)
  finally show ?case
    using s by (auto simp: S_def)
qed
qed
end

```

We now lift everything we have done to non-nice elliptic functions.

```

context elliptic_function
begin

```

```

lemma elliptic_order_remove_sings [simp]: "elliptic_order (remove_sings
f) = elliptic_order f"
  unfolding elliptic_order_def by simp

```

```

interpretation elliptic_function_remove_sings ..

```

```

theorem zeros_eq_elliptic_order:

```

```

  " $(\sum z \mid z \in \text{period\_parallelogram orig} \wedge \text{isolated\_zero } f \ z. \text{nat } (z \text{order }
f \ z)) = \text{elliptic\_order } f$ "
  using remove_sings.zeros_eq_elliptic_order_aux by simp

```

```

lemma card_poles_le_order: "card {z  $\in$  period_parallelogram orig. is_pole
f z}  $\leq$  elliptic_order f"

```

```

proof -

```

```

have "zorder f z < 0" if "is_pole f z" for z
  using that by (simp add: isolated_pole_imp_neg_zorder)
hence " $(\sum z \mid z \in \text{period\_parallelogram orig} \wedge \text{is\_pole } f z. 1) \leq$ "
  " $(\sum z \mid z \in \text{period\_parallelogram orig} \wedge \text{is\_pole } f z. \text{nat } (-\text{zorder}$ 
f z))"
  by (intro sum_mono) force+
  thus ?thesis
  by (simp add: poles_eq_elliptic_order)
qed

```

```

lemma card_zeros_le_order: "card {z∈period_parallelogram orig. isolated_zero
f z} ≤ elliptic_order f"

```

```

proof -
  have "zorder f z > 0" if "isolated_zero f z" for z
    using that by (intro zorder_isolated_zero_pos') auto
  hence " $(\sum z \mid z \in \text{period\_parallelogram orig} \wedge \text{isolated\_zero } f z. 1)$ "
  ≤
    " $(\sum z \mid z \in \text{period\_parallelogram orig} \wedge \text{isolated\_zero } f z. \text{nat}$ 
(zorder f z))"
    by (intro sum_mono) force+
  thus ?thesis
    by (simp add: zeros_eq_elliptic_order)
qed

```

```

corollary elliptic_order_eq_0_iff_no_poles: "elliptic_order f = 0 ↔
(∀z. ¬is_pole f z)"

```

```

proof
  assume "elliptic_order f = 0"
  hence "card {z∈period_parallelogram 0. is_pole f z} ≤ 0"
    using card_poles_le_order[of 0] by simp
  hence "{z∈period_parallelogram 0. is_pole f z} = {}"
    using finite_poles_in_parallelogram[of 0] by simp
  hence *: "¬is_pole f z" if "z ∈ period_parallelogram 0" for z
    using that by blast
  hence "¬is_pole f z" for z
    using *[of "to_fund_parallelogram z"] poles.lattice_cong[of z "to_fund_parallelogram
z"]
    by (auto simp: to_fund_parallelogram_in_parallelogram)
  thus "∀z. ¬is_pole f z"
    by blast
qed (simp_all add: elliptic_order_def)

```

```

corollary elliptic_order_eq_0_iff_no_zeros: "elliptic_order f = 0 ↔
(∀z. ¬isolated_zero f z)"

```

```

proof
  assume "elliptic_order f = 0"
  hence "card {z∈period_parallelogram 0. isolated_zero f z} ≤ 0"
    using card_zeros_le_order[of 0] by simp
  hence "{z∈period_parallelogram 0. isolated_zero f z} = {}"

```

```

    using finite_zeros_in_parallelogram[of 0] by simp
  hence *: "¬isolated_zero f z" if "z ∈ period_parallelogram 0" for z
    using that by blast
  hence "¬isolated_zero f z" for z
    using *[of "to_fund_parallelogram z"] zeros.lattice_cong[of z "to_fund_parallelogram
z"]
    by (auto simp: to_fund_parallelogram_in_parallelogram)
  thus "∀z. ¬isolated_zero f z"
    by blast
qed (simp_all flip: zeros_eq_elliptic_order[of 0])

```

```

lemma elliptic_order_eq_0_iff_const_cosparse:
  "elliptic_order f = 0 ⟷ (∃c. ∀≈x∈UNIV. f x = c)"
proof -
  have "elliptic_order f = 0 ⟷ elliptic_order (remove_sings f) = 0"
    by simp
  also have "... ⟷ remove_sings f constant_on UNIV"
    by (subst remove_sings.elliptic_order_eq_0_iff) auto
  also have "... ⟷ (∃c. ∀≈x∈UNIV. f x = c)"
    by (subst remove_sings_constant_on_open_iff) (auto intro: meromorphic)
  finally show ?thesis .
qed

```

```

lemma cosparse_eq_or_avoid: "(∀≈z∈UNIV. f z = c) ∨ (∀≈z∈UNIV. f z
≠ c)"
  by (simp add: Convex.connected_UNIV meromorphic meromorphic_imp_constant_or_avoid)

```

```

lemma frequently_eq_imp_almost_everywhere_eq:
  assumes "frequently (λz. f z = c) (at z)"
  shows "eventually (λz. f z = c) (cosparse UNIV)"
proof -
  have "¬eventually (λz. f z ≠ c) (cosparse UNIV)"
    using assms by (auto simp: eventually_cosparse_open_eq frequently_def)
  thus ?thesis
    using cosparse_eq_or_avoid[of c] by auto
qed

```

```

lemma eventually_eq_imp_almost_everywhere_eq:
  assumes "eventually (λz. f z = c) (at z)"
  shows "eventually (λz. f z = c) (cosparse UNIV)"
  using assms frequently_eq_imp_almost_everywhere_eq eventually_frequently
at_neq_bot by blast

```

```

lemma avoid: "elliptic_order f > 0 ⟹ ∀≈z∈UNIV. f z ≠ c"
  using cosparse_eq_or_avoid elliptic_order_eq_0_iff_const_cosparse by
auto

```

```

lemma avoid': "elliptic_order f > 0 ⟹ eventually (λz. f z ≠ c) (at
z)"

```

```

using avoid eventually_cosparse_imp_eventually_at by blast

theorem sum_zeros_poles_in_lattice:
  fixes orig :: complex
  defines "Z ≡ {z ∈ period_parallelogram orig. isolated_zero f z ∨ is_pole
f z}"
  shows "(∑ z ∈ Z. of_int (zorder f z) * z) ∈ Λ"
  using remove_sings.sum_zeros_poles_in_lattice_aux[of orig]
  by (simp add: Z_def)

```

```

theorem sum_residues_eq_0:
  fixes orig :: complex
  defines "Q ≡ {z ∈ period_parallelogram orig. is_pole f z}"
  shows "(∑ z ∈ Q. residue f z) ∈ Λ"
  using remove_sings.sum_residues_eq_0_aux[of orig] by (simp add: Q_def)

```

An obvious fact that we use at one point: if $\sum_{x \in A} f(x) = 1$ for $f(x)$ in the positive integers, then $A = \{x\}$ for some x and $f(x) = 1$.

```

lemma (in -) sum_nat_eq_1E:
  fixes f :: "'a ⇒ nat"
  assumes sum_eq: "(∑ x ∈ A. f x) = 1"
  assumes pos: "∧ x. x ∈ A ⇒ f x > 0"
  obtains x where "A = {x}" "f x = 1"
proof -
  have [simp, intro]: "finite A"
    by (rule ccontr) (use sum_eq in auto)
  have "A ≠ {}"
    using sum_eq by auto
  then obtain x where x: "x ∈ A"
    by auto
  have "(∑ x ∈ A. f x) = f x + (∑ x ∈ A - {x}. f x)"
    by (meson <finite A> <x ∈ A> sum.remove)
  hence "f x + (∑ x ∈ A - {x}. f x) = 1"
    using sum_eq by simp
  with pos[OF x] have "f x = 1" "(∑ x ∈ A - {x}. f x) = 0"
    by linarith+
  from this have "∀ y ∈ A - {x}. f y = 0"
    by (subst (asm) sum_eq_0_iff) auto
  with pos have "A - {x} = {}"
    by force
  with x have "A = {x}"
    by auto
  with <f x = 1> show ?thesis
    using that[of x] by blast
qed

```

A simple consequence of our result about the sums of poles and zeros being a lattice point is that there are no elliptic functions of order 1.

If there were such a function, it would have only one zero and one pole (both

simple) in the fundamental parallelogram. Since their sum would be a lattice point, they would be equivalent modulo the lattice and thus identical. But a point cannot be both a zero and a pole.

theorem *elliptic_order_neq_1*: "elliptic_order f \neq 1"

proof

 assume "elliptic_order f = 1"

 define P where "P = period_parallelogram 0"

 from \langle elliptic_order f = 1 \rangle have *: " $(\sum z \mid z \in P \wedge \text{is_pole } f \ z. \text{nat } (- \text{zorder } f \ z)) = 1$ "

 by (simp add: elliptic_order_def P_def)

 moreover have "zorder f z < 0" if "is_pole f z" for z using that meromorphic

 by (meson isolated_pole_imp_neg_zorder meromorphic_at_iff meromorphic_on_subset top_greatest)

 ultimately obtain pole where pole: "{z. z \in P \wedge is_pole f z} = {pole}"

"zorder f pole = -1"

 by (elim sum_nat_eq_1E) auto

 from \langle elliptic_order f = 1 \rangle have *: " $(\sum z \mid z \in P \wedge \text{isolated_zero } f \ z. \text{nat } (\text{zorder } f \ z)) = 1$ "

 using zeros_eq_elliptic_order[of 0] by (simp add: P_def)

 moreover have "zorder f z > 0" if "isolated_zero f z" for z using that

 by (simp add: zorder_isolated_zero_pos')

 ultimately obtain zero where zero: "{z. z \in P \wedge isolated_zero f z} = {zero}" "zorder f zero = 1"

 by (elim sum_nat_eq_1E) auto

 have zero': "isolated_zero f zero" and pole': "is_pole f pole"

 using zero pole by blast+

 have "zero \neq pole"

 using zero' pole' pole_is_not_zero by blast

 have " $(\sum z \mid z \in P \wedge (\text{isolated_zero } f \ z \vee \text{is_pole } f \ z). \text{of_int } (\text{zorder } f \ z) * z) \in \Lambda$ "

 unfolding P_def by (rule sum_zeros_poles_in_lattice)

 also have "{z. z \in P \wedge (isolated_zero f z \vee is_pole f z)} = {zero, pole}"

 using zero pole by auto

 finally have "zero - pole \in Λ "

 using \langle zero \neq pole \rangle zero pole by simp

 hence "rel zero pole"

 by (simp add: rel_def)

 thus False

 using zero' pole' pole_is_not_zero poles.lattice_cong by blast

qed

end

```

locale nonconst_nicely_elliptic_function = nicely_elliptic_function +
  assumes order_pos: "elliptic_order f > 0"
begin

lemma isolated_zero_iff': "isolated_zero f z  $\longleftrightarrow$   $\neg$ is_pole f z  $\wedge$  f z
= 0"
proof -
  have "f  $\neq$  ( $\lambda$ _. 0)"
    using order_pos by auto
  thus ?thesis
    using isolated_zero_iff[of z] by simp
qed

end

```

6.3 Even elliptic functions

If an elliptic function is even, i.e. $f(-z) = f(z)$, it is invariant not only under the group generated by $z \mapsto z + \omega_1$ and $z \mapsto z + \omega_2$, but also the additional generator $z \mapsto -z$.

Since our prototypical example of an elliptic function – the Weierstraß \wp function – is even, we will examine these a bit more closely here.

```

locale even_elliptic_function = elliptic_function +
  assumes even: "f (-z) = f z"
begin

```

The Laurent series expansion of an even elliptic function at lattice points and half-lattice points only has even-index coefficients. This also means that, at lattice and half-lattice points, an even elliptic function can only have zeros and poles of even order.

```

lemma
  assumes z: "2 * z  $\in$   $\Lambda$ " and " $\neg$ ( $\forall$   $\approx$  z. f z = 0)"
  shows odd_laurent_coeffs_eq_0: "odd n  $\implies$  fls_nth (laurent_expansion
f z) n = 0"
  and even_zorder: "even (zorder f z)"
proof -
  define F where "F = laurent_expansion f z"
  have F: " $(\lambda$ w. f (z + w)) has_laurent_expansion F"
    unfolding F_def by (simp add: not_essential_has_laurent_expansion)
  have "F  $\neq$  0"
  proof
    assume "F = 0"
    with F have "eventually ( $\lambda$ w. f w = 0) (at z)"
      using has_laurent_expansion_eventually_zero_iff by blast
    thus False
    using assms(2) eventually_eq_imp_almost_everywhere_eq by blast
  qed

```

```

    have "((λw. f (z + w)) ∘ uminus) has_laurent_expansion (fls_compose_fps
F (-fps_X))"
      by (intro laurent_expansion_intros F has_laurent_expansion_fps fps_expansion_intros)
auto
    also have "((λw. f (z + w)) ∘ uminus) = (λw. f (z + w))" (is "?lhs =
?rhs")
    proof
      fix w :: complex
      have "?lhs w = f (z - w)"
        by simp
      also have "... = f (-(z + w))"
        by (rule lattice_cong) (use assms z in <auto simp: rel_def>)
      also have "... = f (z + w)"
        by (rule even)
      finally show "?lhs w = ?rhs w" .
    qed
    finally have "(λw. f (z + w)) has_laurent_expansion fls_compose_fps F
(- fps_X)" .
    with F have "F = fls_compose_fps F (- fps_X)"
      using has_laurent_expansion_unique by blast
    thus even': "fls_nth F n = 0" if "odd n" for n
      using that fls_nth_fls_compose_fps_linear[of "-1" F n]
      by (auto simp: fls_eq_iff power_int_minus_left simp flip: fps_const_neg)

    have "fls_nth F (fls_subdegree F) ≠ 0"
      using <F ≠ 0> by auto
    with even' have "even (fls_subdegree F)"
      by blast
    also have "fls_subdegree F = zorder f z"
      using F <F ≠ 0> by (metis has_laurent_expansion_zorder)
    finally show "even (zorder f z)" .
  qed

lemma lattice_cong': "rel w z ∨ rel w (-z) ⇒ f w = f z"
  using even lattice_cong by metis

lemma eval_to_half_fund_parallelogram: "f (to_half_fund_parallelogram
z) = f z"
  using rel_to_half_fund_parallelogram[of z] even lattice_cong by metis

lemma zorder_to_half_fund_parallelogram: "zorder f (to_half_fund_parallelogram
z) = zorder f z"
proof -
  define z' where "z' = to_half_fund_parallelogram z"
  have "rel z z' ∨ rel z (-z)"
    unfolding z'_def by (rule rel_to_half_fund_parallelogram)
  hence "zorder f z = zorder f z'"
  proof

```

```

    assume "rel z z'"
    thus "zorder f z = zorder f z'"
      by (rule zorder.lattice_cong)
next
  assume "rel z (-z'"
  hence "zorder f z = zorder f (-z'"
    by (rule zorder.lattice_cong)
  also have "... = zorder ( $\lambda z. f (-z)$ ) z'"
    by (rule zorder_uminus [symmetric]) (auto intro!: meromorphic')
  also have "... = zorder f z'"
    by (simp add: even)
  finally show "zorder f z = zorder f z'" .
qed
thus ?thesis
  by (simp add: z'_def)
qed

lemma zorder_uminus: "zorder f (-z) = zorder f z"
  by (metis rel_refl to_half_fund_parallelogram_eq_iff zorder_to_half_fund_parallelogram)

end

```

6.4 Closure properties of the class of elliptic functions

Elliptic functions are closed under all basic arithmetic operations (addition, subtraction, multiplication, division). Additionally, they are closed under derivative, translation ($f(z) \rightsquigarrow f(z+c)$) and scaling with an integer ($f(z) \rightsquigarrow f(nz)$).

Furthermore, constant functions are elliptic.

```

lemma elliptic_function_unop:
  assumes "elliptic_function  $\omega_1 \omega_2 f$ "
  assumes "f meromorphic_on UNIV  $\implies (\lambda z. h (f z))$  meromorphic_on UNIV"
  shows "elliptic_function  $\omega_1 \omega_2 (\lambda z. h (f z))$ "
proof -
  interpret elliptic_function  $\omega_1 \omega_2 f$  by fact
  interpret complex_lattice_periodic_compose  $\omega_1 \omega_2 f h ..$ 
  show ?thesis
    by standard (use assms(2) meromorphic in auto)
qed

```

```

lemma elliptic_function_binop:
  assumes "elliptic_function  $\omega_1 \omega_2 f$ " "elliptic_function  $\omega_1 \omega_2 g$ "
  assumes "f meromorphic_on UNIV  $\implies g$  meromorphic_on UNIV  $\implies (\lambda z. h (f z) (g z))$  meromorphic_on UNIV"
  shows "elliptic_function  $\omega_1 \omega_2 (\lambda z. h (f z) (g z))$ "
proof -
  interpret f: elliptic_function  $\omega_1 \omega_2 f$  by fact
  interpret g: elliptic_function  $\omega_1 \omega_2 g$  by fact

```

```

interpret complex_lattice_periodic  $\omega_1 \omega_2$  " $\lambda z. h (f z) (g z)$ "
  by standard (auto intro!: arg_cong2[of _ _ _ h] f.lattice_cong g.lattice_cong
simp: f.rel_def)
  show ?thesis
  by standard (use assms(3) f.meromorphic g.meromorphic in auto)
qed

context complex_lattice
begin

named_theorems elliptic_function_intros

lemmas (in elliptic_function) [elliptic_function_intros] = elliptic_function_axioms

lemma elliptic_function_const [elliptic_function_intros]:
  "elliptic_function  $\omega_1 \omega_2 (\lambda_. c)$ "
  by standard auto

lemma [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1 \omega_2 f$ "
  shows elliptic_function_cmult_left: "elliptic_function  $\omega_1 \omega_2 (\lambda z. c * f z)$ "
    and elliptic_function_cmult_right: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z * c)$ "
    and elliptic_function_scaleR: "elliptic_function  $\omega_1 \omega_2 (\lambda z. c' *_{\mathbb{R}} f z)$ "
    and elliptic_function_uminus: "elliptic_function  $\omega_1 \omega_2 (\lambda z. -f z)$ "
    and elliptic_function_inverse: "elliptic_function  $\omega_1 \omega_2 (\lambda z. \text{inverse } (f z))$ "
    and elliptic_function_power: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z ^ m)$ "
    and elliptic_function_power_int: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z \text{ powi } n)$ "
  by (rule elliptic_function_unop[OF assms(1)]; force intro!: meromorphic_intros)+

lemma [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1 \omega_2 f$ " "elliptic_function  $\omega_1 \omega_2 g$ "
  shows elliptic_function_cmult_add: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z + g z)$ "
    and elliptic_function_cmult_diff: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z - g z)$ "
    and elliptic_function_cmult_mult: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z * g z)$ "
    and elliptic_function_cmult_divide: "elliptic_function  $\omega_1 \omega_2 (\lambda z. f z / g z)$ "
  by (rule elliptic_function_binop[OF assms(1,2)]; force intro!: meromorphic_intros)+

```

```

lemma elliptic_function_compose_mult_of_int_left:
  assumes "elliptic_function  $\omega_1$   $\omega_2$   $f$ "
  shows "elliptic_function  $\omega_1$   $\omega_2$  ( $\lambda z. f$  (of_int  $n$  *  $z$ ))"
proof -
  interpret elliptic_function  $\omega_1$   $\omega_2$   $f$ 
  by fact
  show ?thesis
  by unfold_locales
  (auto intro!: meromorphic_intros analytic_intros lattice_cong lattice_intros
    simp: rel_def uminus_in_lattice_iff ring_distrib)
qed

lemma elliptic_function_compose_mult_of_nat_left:
  assumes "elliptic_function  $\omega_1$   $\omega_2$   $f$ "
  shows "elliptic_function  $\omega_1$   $\omega_2$  ( $\lambda z. f$  (of_nat  $n$  *  $z$ ))"
  using elliptic_function_compose_mult_of_int_left[OF assms, of "int  $n$ "]
  by simp

lemma elliptic_function_compose_mult_numeral_left:
  assumes "elliptic_function  $\omega_1$   $\omega_2$   $f$ "
  shows "elliptic_function  $\omega_1$   $\omega_2$  ( $\lambda z. f$  (numeral  $n$  *  $z$ ))"
  using elliptic_function_compose_mult_of_int_left[OF assms, of "numeral
 $n$ "] by simp

lemma
  assumes "elliptic_function  $\omega_1$   $\omega_2$   $f$ "
  shows elliptic_function_compose_mult_of_int_right: "elliptic_function
 $\omega_1$   $\omega_2$  ( $\lambda z. f$  ( $z$  * of_int  $n$ ))"
  and elliptic_function_compose_mult_of_nat_right: "elliptic_function
 $\omega_1$   $\omega_2$  ( $\lambda z. f$  ( $z$  * of_nat  $m$ ))"
  and elliptic_function_compose_mult_numeral_right: "elliptic_function
 $\omega_1$   $\omega_2$  ( $\lambda z. f$  ( $z$  * numeral  $num$ ))"
  by (subst mult.commute,
    rule elliptic_function_compose_mult_of_int_left
      elliptic_function_compose_mult_of_nat_left
      elliptic_function_compose_mult_numeral_left,
    rule assms)+

lemma elliptic_function_compose_uminus:
  assumes "elliptic_function  $\omega_1$   $\omega_2$   $f$ "
  shows "elliptic_function  $\omega_1$   $\omega_2$  ( $\lambda z. f$  ( $-z$ ))"
  using elliptic_function_compose_mult_of_int_left[OF assms, of "-1"]
  by simp

lemma elliptic_function_shift:
  assumes "elliptic_function  $\omega_1$   $\omega_2$   $f$ "
  shows "elliptic_function  $\omega_1$   $\omega_2$  ( $\lambda z. f$  ( $z + w$ ))"

```

```

proof -
  interpret elliptic_function  $\omega_1$   $\omega_2$  f by fact
  show ?thesis
  proof
    show "( $\lambda z. f (z + w)$ ) meromorphic_on UNIV"
      using meromorphic by (rule meromorphic_on_compose) (auto intro!:
analytic_intros)
    qed (auto intro!: lattice_cong simp: rel_def)
  qed

definition shift_fun :: "'a  $\Rightarrow$  ('a  $\Rightarrow$  'a)  $\Rightarrow$  'a  $\Rightarrow$  'a :: plus" where
  "shift_fun w f = ( $\lambda z. f (z + w)$ )"

lemma elliptic_function_shift' [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1$   $\omega_2$  f"
  shows "elliptic_function  $\omega_1$   $\omega_2$  (shift_fun w f)"
  unfolding shift_fun_def using assms by (rule elliptic_function_shift)

lemma nicely_elliptic_function_remove_sings [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1$   $\omega_2$  f"
  shows "nicely_elliptic_function  $\omega_1$   $\omega_2$  (remove_sings f)"
proof -
  interpret elliptic_function  $\omega_1$   $\omega_2$  f by fact
  interpret elliptic_function_remove_sings  $\omega_1$   $\omega_2$  f ..
  show ?thesis ..
qed

lemma elliptic_function_remove_sings [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1$   $\omega_2$  f"
  shows "elliptic_function  $\omega_1$   $\omega_2$  (remove_sings f)"
proof -
  interpret elliptic_function  $\omega_1$   $\omega_2$  f by fact
  interpret elliptic_function_remove_sings  $\omega_1$   $\omega_2$  f ..
  show ?thesis ..
qed

lemma elliptic_function_deriv [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1$   $\omega_2$  f"
  shows "elliptic_function  $\omega_1$   $\omega_2$  (deriv f)"
proof -
  interpret elliptic_function  $\omega_1$   $\omega_2$  f by fact
  show ?thesis by standard (auto intro!: meromorphic_intros meromorphic)
qed

lemma elliptic_function_higher_deriv [elliptic_function_intros]:
  assumes "elliptic_function  $\omega_1$   $\omega_2$  f"
  shows "elliptic_function  $\omega_1$   $\omega_2$  ((deriv ^^ n) f)"
  using assms by (induction n) (auto intro!: elliptic_function_intros)

```

```

lemma elliptic_function_sum [elliptic_function_intros]:
  assumes " $\bigwedge x. x \in X \implies \text{elliptic\_function } \omega_1 \ \omega_2 \ (f \ x)$ "
  shows " $\text{elliptic\_function } \omega_1 \ \omega_2 \ (\lambda z. \sum_{x \in X}. f \ x \ z)$ "
  using assms
  by (induction X rule: infinite_finite_induct) (auto intro!: elliptic_function_intros)

lemma elliptic_function_prod [elliptic_function_intros]:
  assumes " $\bigwedge x. x \in X \implies \text{elliptic\_function } \omega_1 \ \omega_2 \ (f \ x)$ "
  shows " $\text{elliptic\_function } \omega_1 \ \omega_2 \ (\lambda z. \prod_{x \in X}. f \ x \ z)$ "
  using assms
  by (induction X rule: infinite_finite_induct) (auto intro!: elliptic_function_intros)

lemma elliptic_function_sum_list [elliptic_function_intros]:
  assumes " $\bigwedge f. f \in \text{set } fs \implies \text{elliptic\_function } \omega_1 \ \omega_2 \ f$ "
  shows " $\text{elliptic\_function } \omega_1 \ \omega_2 \ (\lambda z. \sum_{f \leftarrow fs}. f \ z)$ "
  using assms by (induction fs) (auto intro!: elliptic_function_intros)

lemma elliptic_function_prod_list [elliptic_function_intros]:
  assumes " $\bigwedge f. f \in \text{set } fs \implies \text{elliptic\_function } \omega_1 \ \omega_2 \ f$ "
  shows " $\text{elliptic\_function } \omega_1 \ \omega_2 \ (\lambda z. \prod_{f \leftarrow fs}. f \ z)$ "
  using assms by (induction fs) (auto intro!: elliptic_function_intros)

lemma elliptic_function_sum_mset [elliptic_function_intros]:
  assumes " $\bigwedge f. f \in \# F \implies \text{elliptic\_function } \omega_1 \ \omega_2 \ f$ "
  shows " $\text{elliptic\_function } \omega_1 \ \omega_2 \ (\lambda z. \sum_{f \in \#F}. f \ z)$ "
  using assms by (induction F) (auto intro!: elliptic_function_intros)

lemma elliptic_function_prod_mset [elliptic_function_intros]:
  assumes " $\bigwedge f. f \in \# F \implies \text{elliptic\_function } \omega_1 \ \omega_2 \ f$ "
  shows " $\text{elliptic\_function } \omega_1 \ \omega_2 \ (\lambda z. \prod_{f \in \#F}. f \ z)$ "
  using assms by (induction F) (auto intro!: elliptic_function_intros)

end

```

6.5 Affine transformations and surjectivity

In the following we look at the properties of the elliptic function $af(z) + b$, where $a \neq 0$. Obviously this function inherits many properties from $f(z)$.

```

locale elliptic_function_affine = elliptic_function +
  fixes a b :: complex and g :: "complex  $\Rightarrow$  complex"
  defines "g  $\equiv$   $\lambda z. a * f \ z + b$ "
  assumes nonzero_const: "a  $\neq$  0"
begin

sublocale affine: elliptic_function  $\omega_1 \ \omega_2 \ g$ 
  unfolding g_def by (intro elliptic_function_intros elliptic_function_axioms)

lemma is_pole_affine_iff: "is_pole g z  $\longleftrightarrow$  is_pole f z"

```

```

using nonzero_const by (simp add: g_def flip: is_pole_plus_const_iff)

lemma zorder_pole_affine:
  assumes "is_pole f z"
  shows "zorder g z = zorder f z"
proof -
  note [simp] = nonzero_const
  have "zorder ( $\lambda z. a * f z + b$ ) z = zorder ( $\lambda z. a * f z$ ) z"
  proof (cases "b = 0")
    case [simp]: False
    show ?thesis
    proof (intro zorder_add1)
      from assms have "zorder ( $\lambda z. a * f z$ ) z < 0"
      by (intro isolated_pole_imp_neg_zorder) (auto intro!: singularity_intros)
      thus "zorder ( $\lambda z. a * f z$ ) z < zorder ( $\lambda z. b$ ) z"
      by simp
    next
      have "elliptic_order f > 0"
      using assms elliptic_order_eq_0_iff_no_poles by blast
      hence " $\forall_F z$  in at z.  $a * f z \neq 0$ "
      using avoid'[of 0 z] by auto
      thus " $\exists_F z$  in at z.  $a * f z \neq 0$ "
      using at_neq_bot eventually_frequently by blast
    qed (use nonzero_const in <auto intro!: meromorphic_intros meromorphic>)
  qed auto
  also have "... = zorder f z"
  by (rule zorder_cmult) auto
  finally show ?thesis by (simp only: g_def)
qed

lemma order_affine_eq: "elliptic_order g = elliptic_order f"
  unfolding elliptic_order_def using nonzero_const
  by (intro sum.cong Collect_cong conj_cong refl)
  (auto simp flip: is_pole_plus_const_iff simp: zorder_pole_affine
  is_pole_affine_iff)

end

```

One consequence of the above is that a non-constant elliptic function takes on each value in \mathbb{C} “equally often”. In particular, this means that any non-constant elliptic function is surjective, i.e. for every $c \in \mathbb{C}$ there exists a preimage z with $f(z) = c$ in every period parallelogram.

```

context nonconst_nicely_elliptic_function
begin

```

```

theorem surj:
  fixes c :: complex
  obtains z where "¬is_pole f z" "z ∈ period_parallelogram w" "f z = c"

```

```

proof -
  define g where "g = ( $\lambda z. f z - c$ )"
  interpret elliptic_function_affine  $\omega_1 \omega_2 f 1 "-c" g$ 
    by unfold_locales (auto simp: g_def)

  have "elliptic_order g > 0"
    using order_affine_eq order_pos by simp
  then obtain z where z: "isolated_zero g z"
    using affine.elliptic_order_eq_0_iff_no_zeros by auto
  moreover have " $\neg$ is_pole f z"
    using z pole_is_not_zero is_pole_affine_iff by blast
  hence "f analytic_on {z}"
    by (simp add: analytic_at_iff_not_pole)
  hence "g analytic_on {z}"
    by (auto simp: g_def intro!: analytic_intros)
  ultimately have g: "g z = 0"
    using zero_isolated_zero_analytic by auto

  obtain h where h: "bij_betw h (period_parallelogram z) (period_parallelogram w)"
    " $\bigwedge z. \text{rel } (h z) z$ "
    by (rule bij_betw_period_parallelograms[of z w]) blast

  show ?thesis
  proof (rule that[of "h z"])
    show " $\neg$ is_pole f (h z)"
      using <is_pole f z> h(2)[of z] poles.lattice_cong by blast
  next
    show "h z  $\in$  period_parallelogram w"
      using h(1) by (auto simp: bij_betw_def)
  next
    show "f (h z) = c"
      using g h(2)[of z] unfolding g_def by (simp add: lattice_cong)
  qed
qed

end

end

```

7 The Weierstraß \wp Function

```

theory Weierstrass_Elliptic
imports
  Elliptic_Functions
  Modular_Group
  Theta_Functions.Theta_Nullwert
begin

```

In this section, we will define the Weierstraß \wp function, which is in some

sense the simplest and most fundamental elliptic function. All elliptic functions can be expressed solely in terms of \wp and \wp' .

7.1 Preliminary convergence results

We first examine the uniform convergence of the series

$$\sum_{\omega \in \Lambda^*} \frac{1}{(z - \omega)^n}$$

and

$$\sum_{\omega \in \Lambda} \frac{1}{(z - \omega)^n}$$

for fixed $n \geq 3$.

The second version is an elliptic function that we call the *Eisenstein function* because setting $z = 0$ gives us the Eisenstein series. To our knowledge this function does not have a name of its own in the literature.

This is perhaps because it is up to a constant factor, equal to the $(n - 2)$ -nth derivative of the Weierstraß \wp function (which we will define a bit afterwards).

```
lemmas [simp del] = div_mult_self1 div_mult_self2 div_mult_self3 div_mult_self4
```

```
context complex_lattice
begin
```

```
lemma  $\omega_{\text{upper}}$ :
```

```
  assumes " $\omega \in \text{lattice\_layer } k$ " and " $\alpha > 0$ " and " $k > 0$ "
  shows " $\text{norm } \omega \text{ powr } -\alpha \leq (k * \text{Inf\_para}) \text{ powr } -\alpha$ "
  using lattice_layer_le_norm Inf_para_pos assms powr_mono2' by force
```

```
lemma  $\text{sum\_}\omega_{\text{upper}}$ :
```

```
  assumes " $\alpha > 0$ " and " $k > 0$ "
  shows " $(\sum \omega \in \text{lattice\_layer } k. \text{norm } \omega \text{ powr } -\alpha) \leq 8 * k \text{ powr } (1-\alpha)$ "
  * Inf_para powr - $\alpha$ "
  (is " $?lhs \leq ?rhs$ ")
```

```
proof -
```

```
  have " $?lhs \leq (8 * k) * (k * \text{Inf\_para}) \text{ powr } -\alpha$ "
  using sum_bounded_above [OF  $\omega_{\text{upper}}$ ] card_lattice_layer [OF <math>k>0</math>]
```

```
  assms
```

```
  by fastforce
```

```
  also have "... = ?rhs"
```

```
  using Inf_para_pos by (simp add: powr_diff powr_minus powr_mult divide_simps)
```

```
  finally show ?thesis .
```

```
qed
```

```

lemma lattice_layer_lower:
  assumes " $\omega \in \text{lattice\_layer } k$ " and " $k > 0$ "
  shows " $(k * (\text{if } \alpha \geq 0 \text{ then Inf\_para else Sup\_para})) \text{ powr } \alpha \leq \text{norm } \omega \text{ powr } \alpha$ "
proof (cases " $\alpha \geq 0$ ")
  case True
  have [simp]: " $\omega \neq 0$ "
  using assms by auto
  show ?thesis
  by (rule powr_mono2)
  (use True lattice_layer_le_norm[of  $\omega$   $k$ ] Inf_para_pos assms in auto)
next
  case False
  show ?thesis
  by (rule powr_mono2') (use False lattice_layer_ge_norm[of  $\omega$   $k$ ] assms in auto)
qed

lemma sum_lattice_layer_lower:
  fixes  $\alpha :: \text{real}$ 
  assumes " $k > 0$ "
  defines " $C \equiv (\text{if } \alpha \geq 0 \text{ then Sup\_para else Inf\_para})$ "
  shows " $8 * k \text{ powr } (1-\alpha) * C \text{ powr } -\alpha \leq (\sum \omega \in \text{lattice\_layer } k. \text{norm } \omega \text{ powr } -\alpha)$ "
  (is "?lhs  $\leq$  ?rhs")
proof -
  from < $k > 0$ > have "?lhs =  $(\sum \omega \in \text{lattice\_layer } k. (k * C) \text{ powr } -\alpha)$ "
  by (simp add: powr_minus powr_diff field_simps powr_mult card_lattice_layer)
  also have "...  $\leq$  ?rhs"
  unfolding C_def using lattice_layer_lower[of _  $k$  " $-\alpha$ "] < $k > 0$ >
  by (cases " $\alpha > 0$ "; intro sum_mono) (auto split: if_splits)
  finally show ?thesis .
qed

lemma converges_absolutely_iff_aux1:
  fixes  $\alpha :: \text{real}$ 
  assumes " $\alpha > 2$ "
  shows "summable  $(\lambda i. \sum \omega \in \text{lattice\_layer } (\text{Suc } i). 1 / \text{norm } \omega \text{ powr } \alpha)$ "
proof (rule summable_comparison_test')
  show "norm  $(\sum \omega \in \text{lattice\_layer } (\text{Suc } n). 1 / \text{norm } \omega \text{ powr } \alpha) \leq 8 * \text{real } (\text{Suc } n) \text{ powr } (1 - \alpha) * \text{Inf\_para } \text{ powr } -\alpha$ " for  $n$ 
proof -
  have "norm  $(\sum \omega \in \text{lattice\_layer } (\text{Suc } n). 1 / \text{norm } \omega \text{ powr } \alpha) = (\sum \omega \in \text{lattice\_layer } (\text{Suc } n). \text{norm } \omega \text{ powr } -\alpha)$ "
  unfolding real_norm_def
  by (subst abs_of_nonneg) (auto intro!: sum_nonneg simp: powr_minus field_simps)
  also have "...  $\leq 8 * \text{real } (\text{Suc } n) \text{ powr } (1 - \alpha) * \text{Inf\_para } \text{ powr } -\alpha$ "

```

```

    using sum_ω_upper[of α "Suc n"] assms by simp
    finally show ?thesis .
  qed
next
  show "summable (λn. 8 * real (Suc n) powr (1 - α) * Inf_para powr -α)"
    by (subst summable_Suc_iff, intro summable_mult summable_mult2, subst
    summable_real_powr_iff)
    (use assms in auto)
  qed

lemma converges_absolutely_iff_aux2:
  fixes α :: real
  assumes "summable (λi. ∑ ω∈lattice_layer (Suc i). 1 / norm ω powr
  α)"
  shows "α > 2"
proof -
  define C where "C = (if α > 0 then Sup_para else Inf_para)"
  have "C > 0"
    using Sup_para_pos Inf_para_pos by (auto simp: C_def)

  have "summable (λn. 8 * real (Suc n) powr (1 - α) * C powr -α)"
  proof (rule summable_comparison_test')
    show "norm (8 * real (Suc n) powr (1 - α) * C powr -α) ≤
    (∑ ω∈lattice_layer (Suc n). 1 / norm ω powr α)" for n
  proof -
    have "norm (8 * real (Suc n) powr (1 - α) * C powr -α) =
    8 * real (Suc n) powr (1 - α) * C powr -α"
    unfolding real_norm_def by (subst abs_of_nonneg) (auto intro!:
    sum_nonneg)
    also have "... ≤ (∑ ω∈lattice_layer (Suc n). 1 / norm ω powr α)"
    using sum_lattice_layer_lower[of "Suc n" α]
    by (auto simp: powr_minus field_simps C_def split: if_splits)
    finally show ?thesis .
  qed
  qed fact
  hence "summable (λn. 8 * C powr -α * real n powr (1 - α))"
    by (subst (asm) summable_Suc_iff) (simp add: mult_ac)
  hence "summable (λn. real n powr (1 - α))"
    using <C > 0 by (subst (asm) summable_cmult_iff) auto
  thus "α > 2"
    by (subst (asm) summable_real_powr_iff) auto
  qed

```

Apostol's Lemma 1

```

lemma converges_absolutely_iff:
  fixes α :: real
  shows "(λω. 1 / norm ω powr α) summable_on Λ* ↔ α > 2"
    (is "?P ↔ _")
proof -

```

```

have "( $\lambda \omega. 1 / \text{norm } \omega \text{ powr } \alpha$ ) summable_on  $\Lambda^*$   $\longleftrightarrow$ 
      ( $\lambda \omega. 1 / \text{norm } \omega \text{ powr } \alpha$ ) summable_on ( $\bigcup i \in \{0<..\}. \text{lattice\_layer } i$ )"
  by (simp add: lattice0_conv_layers)
also have "...  $\longleftrightarrow$  ( $\lambda i. \sum \omega \in \text{lattice\_layer } i. 1 / \text{norm } \omega \text{ powr } \alpha$ ) summable_on
{0<..}"
  using lattice_layer_disjoint
  by (intro summable_on_Union_iff has_sum_finite finite_lattice_layer refl)
      (auto simp: disjoint_family_on_def)
also have "{0<..} = Suc ' UNIV"
  by (auto simp: image_iff) presburger?
also have "( $\lambda i. \sum \omega \in \text{lattice\_layer } i. 1 / \text{norm } \omega \text{ powr } \alpha$ ) summable_on
Suc ' UNIV  $\longleftrightarrow$ 
      ( $\lambda i. \sum \omega \in \text{lattice\_layer } (\text{Suc } i). 1 / \text{norm } \omega \text{ powr } \alpha$ ) summable_on
UNIV"
  by (subst summable_on_reindex) (auto simp: o_def)
also have "...  $\longleftrightarrow$  summable ( $\lambda i. \sum \omega \in \text{lattice\_layer } (\text{Suc } i). 1 / \text{norm }
\omega \text{ powr } \alpha$ )"
  by (rule summable_on_UNIV_nonneg_real_iff) (auto intro: sum_nonneg)
also have "...  $\longleftrightarrow$   $\alpha > 2$ "
  using converges_absolutely_iff_aux1 converges_absolutely_iff_aux2
by blast
  finally show ?thesis .
qed

lemma bounded_lattice_finite:
  assumes "bounded B"
  shows "finite ( $\Lambda \cap B$ )"
  by (meson not_islimpt_lattice inf.bounded_iff islimpt_subset <bounded
B>
      bounded_infinite_imp_islimpt bounded_subset eq_refl)

lemma closed_subset_lattice: " $\Lambda' \subseteq \Lambda \implies \text{closed } \Lambda'$ "
  unfolding closed_limpt using not_islimpt_lattice islimpt_subset by
blast

corollary closed_lattice0: "closed  $\Lambda^*$ "
  unfolding lattice0_def by (rule closed_subset_lattice) auto

lemma weierstrass_summand_bound:
  assumes " $\alpha \geq 1$ " and " $R > 0$ "
  obtains  $M$  where
    " $M > 0$ "
    " $\bigwedge \omega z. [\omega \in \Lambda; \text{cmod } \omega > R; \text{cmod } z \leq R] \implies \text{norm } (z - \omega) \text{ powr } -\alpha
\leq M * (\text{norm } \omega \text{ powr } -\alpha)$ "
  proof -
    obtain  $m$  where  $m: "of\_int m > R / \text{norm } \omega 1"$  " $m \geq 0$ "
      by (metis ex_less_of_int le_less_trans linear not_le of_int_0_le_iff)
  
```

```

obtain W where W: "W ∈ (Λ - cball 0 R) ∩ cball 0 (norm W)"
proof
  show "of_int m * ω1 ∈ (Λ - cball 0 R) ∩ cball 0 (norm (of_int m
* ω1))"
    using m latticeI [of m 0]
    by (simp add: field_simps norm_mult)
qed
define PF where "PF ≡ (Λ - cball 0 R) ∩ cball 0 (norm W)"
have "finite PF"
  by (simp add: PF_def Diff_Int_distrib2 bounded_lattice_finite)
then have "finite (norm ' PF)"
  by blast
then obtain r where "r ∈ norm ' PF" "r ≤ norm W" and r: "∧x. x
∈ norm ' PF ⇒ r ≤ x"
  using PF_def W Min_le Min_in by (metis empty_iff image_eqI)
then obtain ωr where ωr: "ωr ∈ PF" "norm ωr = r"
  by blast
with assms have "ωr ∈ Λ" "ωr ≠ 0" "r > 0"
  by (auto simp: PF_def)
have minr: "r ≤ cmod ω" if "ω ∈ Λ" "cmod ω > R" for ω
  using r <r ≤ cmod W> that unfolding PF_def by fastforce
have "R < r"
  using ωr by (simp add: PF_def)
with <R > 0> have R_iff_r: "cmod ω > R ⟷ cmod ω ≥ r" if "ω ∈ Λ"
for ω
  using that by (auto simp: minr)
define M where "M ≡ (1 - R/r) powr -α"
have "M > 0"
  unfolding M_def using <R < r> by auto
moreover
have "cmod (z - ω) powr -α ≤ M * cmod ω powr -α"
  if "ω ∈ Λ" "cmod z ≤ R" "R < cmod ω" for z ω
proof -
  have "r ≤ cmod ω"
    using minr that by blast
  then have "ω ≠ 0"
    using <0 < r> that by force
  have "1 - R/r ≤ 1 - norm (z/ω)"
    using that <0 < r> <0 < R> <ω ≠ 0> <r ≤ cmod ω>
    by (simp add: field_simps norm_divide) (metis mult.commute mult_mono
norm_ge_zero)
  also have "... ≤ norm (1 - z/ω)"
    by (metis norm_one norm_triangle_ineq2)
  also have "... = norm ((z - ω) / ω)"
    by (simp add: <ω ≠ 0> norm_minus_commute diff_divide_distrib)
  finally have *: "1 - R/r ≤ norm ((z - ω) / ω)" .
  have ge_rR: "cmod (z - ω) ≥ r - R"
    by (smt (verit) <r ≤ cmod ω> norm_minus_commute norm_triangle_ineq2
that(2))

```

```

have "1/M ≤ norm ((z - ω) / ω) powr α"
proof -
  have "1/M = (1 - R / r) powr α"
    by (simp add: M_def powr_minus_divide)
  also have "... ≤ norm ((z - ω) / ω) powr α"
    using * <0 < r> <R < r> <1 ≤ α> powr_mono2 by force
  finally show ?thesis .
qed
then show ?thesis
  using <R > 0> <M > 0> <ω ≠ 0>
  by (simp add: mult.commute divide_simps powr_divide norm_divide
powr_minus)
qed
ultimately show thesis
  using that by presburger
qed

```

Lemma 2 on Apostol p. 8

```

lemma weierstrass_aux_converges_absolutely_in_disk:
  assumes "α > 2" and "R > 0" and "z ∈ cball 0 R"
  shows "(λω. cmod (z - ω) powr -α) summable_on (Λ - cball 0 R)"
proof -
  have Λ: "Λ - cball 0 R ⊆ Λ*"
    using assms by force
  obtain M where "M > 0" and M: "∧ω. [[ω ∈ Λ; cmod ω > R]] ⇒ cmod(z
- ω) powr -α ≤ M * (cmod ω powr -α)"
    using weierstrass_summand_bound assms
    by (smt (verit, del_insts) less_eq_real_def mem_cball_0 one_le_numeral)
  have §: "(λω. 1 / cmod ω powr α) summable_on Λ*"
    using <α > 2> converges_absolutely_iff by blast
  have "(λω. M * norm ω powr -α) summable_on Λ*"
    using summable_on_cmult_right [OF §] by (simp add: powr_minus field_class.field_divide)
  with Λ have "(λω. M * norm ω powr -α) summable_on Λ - cball 0 R"
    using summable_on_subset_banach by blast
  then show ?thesis
    by (rule summable_on_comparison_test) (use M in auto)
qed

```

```

lemma weierstrass_aux_converges_absolutely_in_disk':
  fixes α :: nat and R :: real and z:: complex
  assumes "α > 2" and "R > 0" and "z ∈ cball 0 R"
  shows "(λω. 1 / norm (z - ω) ^ α) summable_on (Λ - cball 0 R)"
proof -
  have "(λω. norm (z - ω) powr -of_nat α) summable_on (Λ - cball 0 R)"
    using assms by (intro weierstrass_aux_converges_absolutely_in_disk)
  auto
  also have "?this ↔ ?thesis"
    unfolding powr_minus
    by (intro summable_on_cong_refl, subst powr_realpow)

```

```

      (use assms in <auto simp: field_simps>)
    finally show ?thesis .
qed

lemma weierstrass_aux_converges_in_disk':
  fixes  $\alpha :: \text{nat}$  and  $R :: \text{real}$  and  $z :: \text{complex}$ 
  assumes " $\alpha > 2$ " and " $R > 0$ " and " $z \in \text{cball } 0 R$ "
  shows " $(\lambda \omega. 1 / (z - \omega) ^ \alpha)$  summable_on  $(\Lambda - \text{cball } 0 R)$ "
  by (rule abs_summable_summable)
    (use weierstrass_aux_converges_absolutely_in_disk' [OF assms]
     in <simp add: norm_divide norm_power>)

lemma weierstrass_aux_converges_absolutely:
  fixes  $\alpha :: \text{real}$ 
  assumes " $\alpha > 2$ " and " $\Lambda' \subseteq \Lambda$ "
  shows " $(\lambda \omega. \text{norm } (z - \omega) \text{ powr } -\alpha)$  summable_on  $\Lambda'$ "
proof (cases "z = 0")
  case True
  hence " $(\lambda \omega. \text{norm } (z - \omega) \text{ powr } -\alpha)$  summable_on  $\Lambda^*$ "
    using converges_absolutely_iff[of  $\alpha$ ] assms by (simp add: powr_minus
  field_simps)
  hence " $(\lambda \omega. \text{norm } (z - \omega) \text{ powr } -\alpha)$  summable_on  $\Lambda$ "
    by (simp add: lattice_lattice0 summable_on_insert_iff)
  thus ?thesis
    by (rule summable_on_subset_banach) fact
next
  case [simp]: False
  define R where "R = norm z"
  have " $(\lambda \omega. \text{norm } (z - \omega) \text{ powr } -\alpha)$  summable_on  $(\Lambda - \text{cball } 0 R)$ "
    using assms by (intro weierstrass_aux_converges_absolutely_in_disk)
  (auto simp: R_def)
  hence " $(\lambda \omega. \text{norm } (z - \omega) \text{ powr } -\alpha)$  summable_on  $(\Lambda - \text{cball } 0 R) \cup (\Lambda$ 
 $\cap \text{cball } 0 R)$ "
    by (intro bounded_lattice_finite summable_on_union[OF _ summable_on_finite])
  auto
  also have "... =  $\Lambda$ "
    by blast
  finally show ?thesis
    by (rule summable_on_subset) fact
qed

lemma weierstrass_aux_converges_absolutely':
  fixes  $\alpha :: \text{nat}$ 
  assumes " $\alpha > 2$ " and " $\Lambda' \subseteq \Lambda$ "
  shows " $(\lambda \omega. 1 / \text{norm } (z - \omega) ^ \alpha)$  summable_on  $\Lambda'$ "
  using weierstrass_aux_converges_absolutely[of "of_nat  $\alpha$ "  $\Lambda'$  z] assms
  by (simp add: powr_nat' powr_minus field_simps norm_power norm_divide
  powr_realpow')

```

```

lemma weierstrass_aux_converges:
  fixes  $\alpha$  :: real
  assumes " $\alpha > 2$ " and " $\Lambda' \subseteq \Lambda$ "
  shows " $(\lambda\omega. (z - \omega) \text{ powr } -\alpha) \text{ summable\_on } \Lambda'$ "
  by (rule abs_summable_summable)
    (use weierstrass_aux_converges_absolutely[OF assms]
      in <simp add: norm_divide norm_powr_real_powr'>)

lemma weierstrass_aux_converges':
  fixes  $\alpha$  :: nat
  assumes " $\alpha > 2$ " and " $\Lambda' \subseteq \Lambda$ "
  shows " $(\lambda\omega. 1 / (z - \omega) ^ \alpha) \text{ summable\_on } \Lambda'$ "
  using weierstrass_aux_converges[of "of_nat  $\alpha$ "  $\Lambda'$  z] assms
  by (simp add: powr_nat' powr_minus field_simps)

lemma
  fixes  $\alpha$  R :: real
  assumes " $\alpha > 2$ " " $R > 0$ "
  shows weierstrass_aux_converges_absolutely_uniformly_in_disk:
    "uniform_limit (cball 0 R)
      ( $\lambda X z. \sum_{\omega \in X}. \text{norm } ((z - \omega) \text{ powr } -\alpha)$ )
      ( $\lambda z. \sum_{\omega \in \Lambda - \text{cball } 0 R}. \text{norm } ((z - \omega) \text{ powr } -\alpha)$ )
      (finite_subsets_at_top ( $\Lambda - \text{cball } 0 R$ ))" (is
?th1)
  and weierstrass_aux_converges_uniformly_in_disk:
    "uniform_limit (cball 0 R)
      ( $\lambda X z. \sum_{\omega \in X}. (z - \omega) \text{ powr } -\alpha$ )
      ( $\lambda z. \sum_{\omega \in \Lambda - \text{cball } 0 R}. (z - \omega) \text{ powr } -\alpha$ )
      (finite_subsets_at_top ( $\Lambda - \text{cball } 0 R$ ))" (is
?th2)
  proof -
    obtain M where M:
      " $M > 0$ " " $\bigwedge \omega z. [\omega \in \Lambda; \text{norm } \omega > R; \text{norm } z \leq R] \implies \text{norm } (z - \omega) \text{ powr } -\alpha \leq M * \text{norm } \omega \text{ powr } -\alpha$ "
    using weierstrass_summand_bound[of  $\alpha$  R] assms by auto

    have 1: " $(\lambda\omega. M * \text{norm } \omega \text{ powr } -\alpha) \text{ summable\_on } (\Lambda - \text{cball } 0 R)$ "
    proof -
      have " $(\lambda\omega. 1 / \text{norm } \omega \text{ powr } \alpha) \text{ summable\_on } \Lambda^*$ "
      using assms by (subst converges_absolutely_iff) auto
      hence " $(\lambda\omega. M * \text{norm } \omega \text{ powr } -\alpha) \text{ summable\_on } \Lambda^*$ "
      by (intro summable_on_cmult_right) (auto simp: powr_minus field_simps)
      thus " $(\lambda\omega. M * \text{norm } \omega \text{ powr } -\alpha) \text{ summable\_on } (\Lambda - \text{cball } 0 R)$ "
      by (rule summable_on_subset) (use assms in <auto simp: lattice0_def>)
    qed

    have 2: " $\text{norm } ((z - \omega) \text{ powr } -\alpha) \leq M * \text{norm } \omega \text{ powr } -\alpha$ "
      if " $\omega \in \Lambda - \text{cball } 0 R$ " " $z \in \text{cball } 0 R$ " for  $\omega z$ 
    proof -

```

```

    have "norm ((z - ω) powr -α) = norm (z - ω) powr -α"
      by (auto simp add: powr_def)
    also have "... ≤ M * norm ω powr -α"
      using that by (intro M) auto
    finally show "norm ((z - ω) powr -α) ≤ M * norm ω powr -α"
      by simp
  qed

  show ?th1 ?th2
    by (rule Weierstrass_m_test_general[OF _ 1]; use 2 in simp)+
  qed

lemma
  fixes n :: nat and R :: real
  assumes "n > 2" "R > 0"
  shows weierstrass_aux_converges_absolutely_uniformly_in_disk':
    "uniform_limit (cball 0 R)
      (λX z. ∑ ω∈X. norm (1 / (z - ω) ^ n))
      (λz. ∑ ∞ ω∈Λ-cball 0 R. norm (1 / (z - ω) ^
n))
      (finite_subsets_at_top (Λ - cball 0 R))" (is
?th1)
  and weierstrass_aux_converges_uniformly_in_disk':
    "uniform_limit (cball 0 R)
      (λX z. ∑ ω∈X. 1 / (z - ω) powr n)
      (λz. ∑ ∞ ω∈Λ-cball 0 R. 1 / (z - ω) ^ n)
      (finite_subsets_at_top (Λ - cball 0 R))" (is
?th2)
  proof -
    have "uniform_limit (cball 0 R)
      (λX z. ∑ ω∈X. norm ((z - ω) powr -real n))
      (λz. ∑ ∞ ω∈Λ-cball 0 R. norm ((z - ω) powr -real n))
      (finite_subsets_at_top (Λ - cball 0 R))"
      by (rule weierstrass_aux_converges_absolutely_uniformly_in_disk) (use
assms in auto)
    also have "?this ↔ ?th1"
      by (intro uniform_limit_cong eventually_finite_subsets_at_top_weakI
sum.cong ballI)
      (auto simp: norm_divide norm_power powr_minus field_simps powr_nat
intro!: infsum_cong)
    finally show ?th1 .
  next
    have "uniform_limit (cball 0 R)
      (λX z. ∑ ω∈X. (z - ω) powr -of_nat n)
      (λz. ∑ ∞ ω∈Λ-cball 0 R. (z - ω) powr -of_nat n)
      (finite_subsets_at_top (Λ - cball 0 R))"
      using weierstrass_aux_converges_uniformly_in_disk[of "of_nat n" R]
assms by auto
    also have "?this ↔ ?th2"

```

```

    by (intro uniform_limit_cong eventually_finite_subsets_at_top_weakI
sum.cong ballI)
      (auto simp: norm_divide norm_power powr_minus field_simps powr_nat
intro!: infsum_cong)
  finally show ?th2 .
qed

```

```

definition eisenstein_fun_aux :: "nat  $\Rightarrow$  complex  $\Rightarrow$  complex" where
  "eisenstein_fun_aux n z =
    (if n = 0 then -1 else if n < 3  $\vee$  z  $\in$   $\Lambda^*$  then 0 else  $(\sum_{\omega \in \Lambda^*} 1 / (z - \omega)^n)$ )"

```

```

lemma eisenstein_fun_aux_at_pole_eq_0: "n > 0  $\implies$  z  $\in$   $\Lambda^*$   $\implies$  eisenstein_fun_aux
n z = 0"
  by (simp add: eisenstein_fun_aux_def)

```

```

lemma eisenstein_fun_aux_has_sum:
  assumes "n  $\geq$  3" "z  $\notin$   $\Lambda^*$ "
  shows "(( $\lambda \omega. 1 / (z - \omega)^n$ ) has_sum eisenstein_fun_aux n z)  $\Lambda^*$ "
proof -
  have "eisenstein_fun_aux n z =  $(\sum_{\omega \in \Lambda^*} 1 / (z - \omega)^n)$ "
    using assms by (simp add: eisenstein_fun_aux_def)
  also have "(( $\lambda \omega. 1 / (z - \omega)^n$ ) has_sum ...)  $\Lambda^*$ "
    using assms by (intro has_sum_infsum weierstrass_aux_converges') (auto
simp: lattice0_def)
  finally show ?thesis .
qed

```

```

lemma eisenstein_fun_aux_minus: "eisenstein_fun_aux n (-z) = (-1)^n
* eisenstein_fun_aux n z"
proof (cases "n < 3  $\vee$  z  $\in$   $\Lambda^*$ ")
  case False
  have "eisenstein_fun_aux n (-z) =  $(\sum_{\omega \in \Lambda^*} 1 / (-z - \omega)^n)$ "
    using False by (auto simp: eisenstein_fun_aux_def lattice0_def uminus_in_lattice_iff)
  also have "... =  $(\sum_{\omega \in \text{uminus } \Lambda^*} 1 / (\omega - z)^n)$ "
    by (subst infsum_reindex) (auto simp: o_def minus_diff_commute inj_on_def)
  also have "uminus ' $\Lambda^*$  =  $\Lambda^*$ "
    by (auto simp: lattice0_def uminus_in_lattice_iff image_def intro:
bexI[of _ "-x" for x])
  also have "(( $\lambda \omega. 1 / (\omega - z)^n$ ) = ( $\lambda \omega. (-1)^n * (1 / (z - \omega)^n)$ )"
proof
  fix  $\omega ::$  complex
  have "1 / ( $\omega - z$ )^n = (1 / ( $\omega - z$ ))^n"
    by (simp add: power_divide)
  also have "1 / ( $\omega - z$ ) = (-1) / (z -  $\omega$ )"
    by (simp add: divide_simps)
  also have "... ^n = (-1)^n * (1 / (z -  $\omega$ )^n)"
    by (subst power_divide) auto

```

```

    finally show "1 / (ω - z) ^ n = (-1) ^ n * (1 / (z - ω) ^ n)" .
  qed
  also have "(∑∞ ω ∈ Λ*. (-1) ^ n * (1 / (z - ω) ^ n)) = (-1) ^ n * eisenstein_fun_aux
  n z"
    using False by (subst infsum_cmult_right') (auto simp: eisenstein_fun_aux_def)
  finally show ?thesis .
  qed (auto simp: eisenstein_fun_aux_def lattice0_def uminus_in_lattice_iff)

lemma eisenstein_fun_aux_even_minus: "even n ⇒ eisenstein_fun_aux
n (-z) = eisenstein_fun_aux n z"
  by (simp add: eisenstein_fun_aux_minus)

lemma eisenstein_fun_aux_odd_minus: "odd n ⇒ eisenstein_fun_aux n
(-z) = -eisenstein_fun_aux n z"
  by (simp add: eisenstein_fun_aux_minus)

lemma eisenstein_fun_aux_has_field_derivative_aux:
  fixes α :: nat and R :: real
  defines "F ≡ (λα z. ∑∞ ω ∈ Λ - cball 0 R. 1 / (z - ω) ^ α)"
  assumes "α > 2" "R > 0" "w ∈ ball 0 R"
  shows "(F α has_field_derivative -of_nat α * F (Suc α) w) (at w)"
proof -
  define α' where "α' = α - 1"
  have α': "α = Suc α'"
    using assms by (simp add: α'_def)
  have 1: "∀F n in finite_subsets_at_top (Λ - cball 0 R).
    continuous_on (cball 0 R) (λz. ∑ω ∈ n. 1 / (z - ω) ^ α) ∧
    (∀z ∈ ball 0 R. ((λz. ∑ω ∈ n. 1 / (z - ω) ^ α) has_field_derivative
    (∑ω ∈ n. -α / (z - ω) ^ Suc α)) (at z))"

  proof (intro eventually_finite_subsets_at_top_weakI conjI continuous_intros
  derivative_intros ballI)
    fix z::complex and X n
    assume "finite X" "X ⊆ Λ - cball 0 R"
      and "z ∈ ball 0 R" "n ∈ X"
    then show "(λz. 1 / (z - n) ^ α) has_field_derivative of_int (-
  int α) / (z - n) ^ Suc α) (at z)"
      apply (auto intro!: derivative_eq_intros simp: divide_simps)
      apply (simp add: algebra_simps α')
    done
  qed auto

  have "uniform_limit (cball 0 R)
    (λX z. ∑ω ∈ X. (z - ω) powr of_real (-of_nat α))
    (λz. ∑∞ ω ∈ Λ - cball 0 R. (z - ω) powr of_real (-of_nat
  α))
    (finite_subsets_at_top (Λ - cball 0 R))"
    using assms by (intro weierstrass_aux_converges_uniformly_in_disk)

```

```

auto
  also have "?this  $\longleftrightarrow$  uniform_limit (cball 0 R) ( $\lambda X z. \sum_{\omega \in X}. 1 / (z - \omega) ^ \alpha$ ) (F  $\alpha$ )
    (finite_subsets_at_top ( $\Lambda -$  cball 0 R))"
  using assms unfolding F_def
  by (intro uniform_limit_cong eventually_finite_subsets_at_top_weakI)
    (auto simp: powr_minus powr_nat field_simps intro!: sum.cong infsum_cong)
  finally have 2: ... .

  have 3: "finite_subsets_at_top ( $\Lambda -$  cball 0 R)  $\neq$  bot"
    by simp

  obtain g where g: "continuous_on (cball 0 R) (F  $\alpha$ )"
    " $\bigwedge w. w \in$  ball 0 R  $\implies$  (F  $\alpha$  has_field_derivative
g w) (at w)  $\wedge$ 
    (( $\lambda \omega. -$ of_nat  $\alpha / (w - \omega) ^$  Suc  $\alpha$ ) has_sum g
w) ( $\Lambda -$  cball 0 R)"
  unfolding has_sum_def using has_complex_derivative_uniform_limit[OF
1 2 3 <R > 0>] by auto

  have "(( $\lambda \omega. -$ of_nat  $\alpha * (1 / (w - \omega) ^$  Suc  $\alpha$ )) has_sum -of_nat  $\alpha *$ 
F (Suc  $\alpha$ ) w) ( $\Lambda -$  cball 0 R)"
  unfolding F_def using assms
  by (intro has_sum_cmult_right has_sum_infsum weierstrass_aux_converges')
auto
  moreover have "(( $\lambda \omega. -$ of_nat  $\alpha * (1 / (w - \omega) ^$  Suc  $\alpha$ )) has_sum
g w) ( $\Lambda -$  cball 0 R)"
  using g(2)[of w] assms by simp
  ultimately have "g w = -of_nat  $\alpha * F$  (Suc  $\alpha$ ) w"
  by (metis infsumI)
  thus ?thesis
  using g(2)[of w] assms by (simp add: F_def)
qed

lemma eisenstein_fun_aux_has_field_derivative:
  assumes z: "z  $\notin$   $\Lambda^*$ " and n: "n  $\geq$  3"
  shows "(eisenstein_fun_aux n has_field_derivative -of_nat n * eisenstein_fun_aux
(Suc n) z) (at z)"
proof -
  define R where "R = norm z + 1"
  have R: "R > 0" "norm z < R"
  by (auto simp: R_def add_nonneg_pos)
  have "finite ( $\Lambda \cap$  cball 0 R)"
  by (simp add: bounded_lattice_finite)
  moreover have " $\Lambda^* \cap$  cball 0 R  $\subseteq$   $\Lambda \cap$  cball 0 R"
  unfolding lattice0_def by blast
  ultimately have fin: "finite ( $\Lambda^* \cap$  cball 0 R)"
  using finite_subset by blast
  define n' where "n' = n - 1"

```

```

from n have n': "n = Suc n'"
  by (simp add: n'_def)

define F1 where "F1 = (λn z. ∑∞ ω ∈ Λ - cball 0 R. 1 / (z - ω) ^ n)"
define F2 where "F2 = (λn z. ∑ ω ∈ Λ* ∩ cball 0 R. 1 / (z - ω) ^ n)"

have "(F1 n has_field_derivative -of_nat n * F1 (Suc n) z) (at z)"
  unfolding F1_def
  by (rule eisenstein_fun_aux_has_field_derivative_aux) (use n in <auto
simp: R_def add_nonneg_pos>)
moreover have "(F2 n has_field_derivative -of_nat n * F2 (Suc n) z)
(at z)"
  unfolding F2_def sum_distrib_left lattice0_def
  by (rule derivative_eq_intros refl sum.cong | use R z n in <force
simp: lattice0_def>)+
  (simp add: divide_simps power3_eq_cube power4_eq_xxxx n')
ultimately have "((λz. F1 n z + F2 n z) has_field_derivative
(-of_nat n * F1 (Suc n) z) + (-of_nat n * F2 (Suc
n) z)) (at z)"
  by (intro derivative_intros)
also have "?this ↔ (eisenstein_fun_aux n has_field_derivative (-of_nat
n * F1 (Suc n) z) + (-of_nat n * F2 (Suc n) z)) (at z)"
  proof (intro has_field_derivative_cong_ev refl)
    have "eventually (λz'. z' ∈ -Λ*) (nhds z)"
      using z by (intro eventually_nhds_in_open) (auto simp: closed_lattice0)
    thus "∀F x in nhds z. x ∈ UNIV → F1 n x + F2 n x = eisenstein_fun_aux
n x"
      proof eventually_elim
        case (elim z)
          have "((λω. 1 / (z - ω) ^ n) has_sum (F1 n z + F2 n z)) ((Λ - cball
0 R) ∪ (Λ* ∩ cball 0 R))"
            unfolding F1_def F2_def using R fin n
            by (intro has_sum_Un_disjoint[OF has_sum_infsum has_sum_finite]
sumnable_on_subset[OF weierstrass_aux_converges']) auto
          also have "(Λ - cball 0 R) ∪ (Λ* ∩ cball 0 R) = Λ*"
            using R unfolding lattice0_def by auto
          finally show ?case using elim n
            unfolding F1_def F2_def by (simp add: infsumI eisenstein_fun_aux_def)
        qed
      qed auto
    also have "(-of_nat n * F1 (Suc n) z) + (-of_nat n * F2 (Suc n) z) =
-of_nat n * (F1 (Suc n) z + F2 (Suc n) z)"
      by (simp add: algebra_simps)
    also have "F1 (Suc n) z + F2 (Suc n) z = eisenstein_fun_aux (Suc n)
z"
      proof -
        have "((λω. 1 / (z - ω) ^ Suc n) has_sum (F1 (Suc n) z + F2 (Suc
n) z)) ((Λ - cball 0 R) ∪ (Λ* ∩ cball 0 R))"
          unfolding F1_def F2_def using R fin n

```

```

    by (intro has_sum_Un_disjoint[OF has_sum_infsum has_sum_finite]
weierstrass_aux_converges') auto
    also have " $(\Lambda - \text{cball } 0 R) \cup (\Lambda^* \cap \text{cball } 0 R) = \Lambda^*$ "
    using R unfolding lattice0_def by auto
    finally show ?thesis using n z
    unfolding F1_def F2_def eisenstein_fun_aux_def by (simp add: infsumI)
qed
finally show ?thesis .
qed

lemmas eisenstein_fun_aux_has_field_derivative' [derivative_intros] =
  DERIV_chain2[OF eisenstein_fun_aux_has_field_derivative]

lemma higher_deriv_eisenstein_fun_aux:
  assumes z: " $z \notin \Lambda^*$ " and n: " $n \geq 3$ "
  shows " $(\text{deriv } ^{\wedge} m) (\text{eisenstein\_fun\_aux } n) z =$ 
 $(-1)^m * \text{pochhammer } (\text{of\_nat } n) m * \text{eisenstein\_fun\_aux } (n$ 
 $+ m) z$ "
  using z n
proof (induction m arbitrary: z n)
  case 0
  thus ?case by simp
next
  case (Suc m z n)
  have ev: "eventually  $(\lambda z. z \in -\Lambda^*)$  (nhds z)"
  using Suc.prem1 closed_lattice0 by (intro eventually_nhds_in_open)
  auto
  have " $(\text{deriv } ^{\wedge} (\text{Suc } m) (\text{eisenstein\_fun\_aux } n) z = \text{deriv } ((\text{deriv } ^{\wedge} m) (\text{eisenstein\_fun\_aux } n)) z$ "
  by simp
  also have "... =  $\text{deriv } (\lambda z. (-1)^m * \text{pochhammer } (\text{of\_nat } n) m * \text{eisenstein\_fun\_aux } (n + m) z) z$ "
  by (intro deriv_cong_ev eventually_mono[OF ev]) (use Suc in auto)
  also have "... =  $(-1)^{\text{Suc } m} * \text{pochhammer } (\text{of\_nat } n) (\text{Suc } m) * \text{eisenstein\_fun\_aux } (\text{Suc } (n + m)) z$ "
  using Suc.prem1
  by (intro DERIV_imp_deriv)
  (auto intro!: derivative_eq_intros simp: pochhammer_Suc algebra_simps)
  finally show ?case
  by simp
qed

lemma eisenstein_fun_aux_holomorphic: "eisenstein_fun_aux n holomorphic_on  $-\Lambda^*$ "
proof (cases " $n \geq 3$ ")
  case True
  thus ?thesis
  using closed_lattice0
  by (subst holomorphic_on_open) (auto intro!: eisenstein_fun_aux_has_field_derivative)

```

```

next
  case False
  thus ?thesis
    by (cases "n = 0") (auto simp: eisenstein_fun_aux_def [abs_def])
qed

lemma eisenstein_fun_aux_holomorphic' [holomorphic_intros]:
  assumes "f holomorphic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda^*$ "
  shows " $(\lambda z. \text{eisenstein\_fun\_aux } n (f z)) \text{ holomorphic\_on } A$ "
proof -
  have "eisenstein_fun_aux n  $\circ$  f holomorphic_on A"
    by (rule holomorphic_on_compose_gen assms eisenstein_fun_aux_holomorphic)+
  (use assms in auto)
  thus ?thesis by (simp add: o_def)
qed

lemma eisenstein_fun_aux_analytic: "eisenstein_fun_aux n analytic_on
 $-\Lambda^*$ "
  by (simp add: analytic_on_open closed_lattice0 open_Cmpl eisenstein_fun_aux_holomorphic)

lemma eisenstein_fun_aux_analytic' [analytic_intros]:
  assumes "f analytic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda^*$ "
  shows " $(\lambda z. \text{eisenstein\_fun\_aux } n (f z)) \text{ analytic\_on } A$ "
proof -
  have "eisenstein_fun_aux n  $\circ$  f analytic_on A"
    by (rule analytic_on_compose_gen assms eisenstein_fun_aux_analytic)+
  (use assms in auto)
  thus ?thesis by (simp add: o_def)
qed

lemma eisenstein_fun_aux_continuous_on: "continuous_on  $(-\Lambda^*)$  (eisenstein_fun_aux
n)"
  using holomorphic_on_imp_continuous_on eisenstein_fun_aux_holomorphic
  by blast

lemma eisenstein_fun_aux_continuous_on' [continuous_intros]:
  assumes "continuous_on A f" " $\bigwedge z. z \in A \implies f z \notin \Lambda^*$ "
  shows "continuous_on A  $(\lambda z. \text{eisenstein\_fun\_aux } n (f z))$ "
  by (rule continuous_on_compose2[OF eisenstein_fun_aux_continuous_on
assms(1)]) (use assms in auto)

lemma weierstrass_aux_translate:
  fixes  $\alpha :: \text{real}$ 
  assumes " $\alpha > 2$ "
  shows " $(\sum_{\infty \omega \in \Lambda. (z + w - \omega) \text{ powr } -\alpha}) = (\sum_{\infty \omega \in (+) (-w) \in \Lambda. (z
- \omega) \text{ powr } -\alpha})$ "
  by (subst infsum_reindex) (auto simp: o_def algebra_simps)

```

```

lemma weierstrass_aux_holomorphic:
  assumes " $\alpha > 2$ " " $\Lambda' \subseteq \Lambda$ " "finite ( $\Lambda - \Lambda'$ )"
  shows " $(\lambda z. \sum_{\omega \in \Lambda'} 1 / (z - \omega)^\alpha)$  holomorphic_on  $-\Lambda'$ "
proof -
  define M where "M = Max (insert 0 (norm ' ( $\Lambda - \Lambda'$ )))"
  have M: " $M \geq 0$ " " $\bigwedge \omega. \omega \in \Lambda - \Lambda' \implies M \geq \text{norm } \omega$ "
    using assms by (auto simp: M_def)
  have [simp]: "closed  $\Lambda'$ "
    using assms(2) by (rule closed_subset_lattice)

  have *: " $(\lambda z. \sum_{\omega \in \Lambda'} 1 / (z - \omega)^\alpha)$  holomorphic_on ball 0 R -  $\Lambda'$ " if R: " $R > M$ " for R
  proof -
    define F where "F = ( $\lambda \alpha z. \sum_{\omega \in \Lambda - \text{cball } 0 R} 1 / (z - \omega)^\alpha$ )"
    define G where "G = ( $\lambda \alpha z. \sum_{\omega \in \Lambda' \cap \text{cball } 0 R} 1 / (z - \omega)^\alpha$ )"

    have "(F  $\alpha$  has_field_derivative -of_nat  $\alpha$  * F (Suc  $\alpha$ ) z) (at z)"
  if z: " $z \in \text{ball } 0 R$ " for z
    unfolding F_def using assms R M(1) z by (intro eisenstein_fun_aux_has_field_derivative)
  auto
    hence "F  $\alpha$  holomorphic_on ball 0 R -  $\Lambda'$ "
      using holomorphic_on_open <closed  $\Lambda'$ > by blast
    hence " $(\lambda z. F \alpha z + G \alpha z)$  holomorphic_on ball 0 R -  $\Lambda'$ "
      unfolding G_def using assms M R by (intro holomorphic_intros) auto
    also have " $(\lambda z. F \alpha z + G \alpha z) = (\lambda z. \sum_{\omega \in \Lambda'} 1 / (z - \omega)^\alpha)$ "
    proof
      fix z :: complex
      have "finite ( $\Lambda \cap \text{cball } 0 R$ )"
        by (intro bounded_lattice_finite) auto
      moreover have " $\Lambda' \cap \text{cball } 0 R \subseteq \Lambda \cap \text{cball } 0 R$ "
        using assms by blast
      ultimately have "finite ( $\Lambda' \cap \text{cball } 0 R$ )"
        using finite_subset by blast
      hence " $((\lambda \omega. 1 / (z - \omega)^\alpha) \text{ has\_sum } (F \alpha z + G \alpha z)) ((\Lambda - \text{cball } 0 R) \cup (\Lambda' \cap \text{cball } 0 R))$ "
        unfolding F_def G_def using assms
        by (intro has_sum_Un_disjoint[OF has_sum_infsum has_sum_finite]
          weierstrass_aux_converges') auto
      also have " $(\Lambda - \text{cball } 0 R) \cup (\Lambda' \cap \text{cball } 0 R) = \Lambda'$ "
        using M R assms by (force simp: not_le)
      finally show "F  $\alpha z + G \alpha z = (\sum_{\omega \in \Lambda'} 1 / (z - \omega)^\alpha)$ "
        by (simp add: infsumI)
    qed
  qed
  finally show ?thesis .
qed

have " $(\lambda z. \sum_{\omega \in \Lambda'} 1 / (z - \omega)^\alpha)$  holomorphic_on ( $\bigcup_{R \in \{R. R > M\}} \text{ball } 0 R - \Lambda'$ )"
  by (rule holomorphic_on_UN_open) (use * <closed  $\Lambda'$ > in auto)

```

```

also have "... = ( $\bigcup_{R \in \{R. R > M\}} \text{ball } 0 R$ ) -  $\Lambda$ "
  by blast
also have "( $\bigcup_{R \in \{R. R > M\}} \text{ball } 0 R$ ) = (UNIV :: complex set)"
proof (safe intro!: UN_I)
  fix z :: complex
  show "norm z + M + 1 > M" "z  $\in$  ball 0 (norm z + M + 1)"
    using M(1) by (auto intro: add_nonneg_pos)
qed auto
finally show ?thesis
  by (simp add: Compl_eq_Diff_UNIV)
qed

```

```

definition eisenstein_fun :: "nat  $\Rightarrow$  complex  $\Rightarrow$  complex" where
  "eisenstein_fun n z = (if n < 3  $\vee$  z  $\in$   $\Lambda$  then 0 else ( $\sum_{\omega \in \Lambda} 1 / (z - \omega)^n$ ))"

```

```

lemma eisenstein_fun_has_sum:
  "n  $\geq$  3  $\implies$  z  $\notin$   $\Lambda \implies ((\lambda \omega. 1 / (z - \omega)^n)$  has_sum eisenstein_fun n z)  $\wedge$ "
  unfolding eisenstein_fun_def by (auto intro!: has_sum_infsum weierstrass_aux_converges')

```

```

lemma eisenstein_fun_at_pole_eq_0: "z  $\in$   $\Lambda \implies$  eisenstein_fun n z = 0"
  by (simp add: eisenstein_fun_def)

```

```

lemma eisenstein_fun_conv_eisenstein_fun_aux:
  assumes "n  $\geq$  3" "z  $\notin$   $\Lambda$ "
  shows "eisenstein_fun n z = eisenstein_fun_aux n z + 1 / z^n"
proof -
  from assms have "eisenstein_fun n z = ( $\sum_{\omega \in \text{insert } 0 \Lambda^*} 1 / (z - \omega)^n$ )"
  by (simp add: eisenstein_fun_def lattice_lattice0)
  also from assms have "... = ( $\sum_{\omega \in \Lambda^*} 1 / (z - \omega)^n$ ) + 1 / z^n"
  by (subst infsum_insert) (auto intro!: weierstrass_aux_converges' simp: lattice_lattice0)
  also from assms have "... = eisenstein_fun_aux n z + 1 / z^n"
  by (simp add: eisenstein_fun_aux_def lattice_lattice0)
  finally show ?thesis .
qed

```

```

lemma eisenstein_fun_altdef:
  "eisenstein_fun n z = (if n < 3  $\vee$  z  $\in$   $\Lambda$  then 0 else eisenstein_fun_aux n z + 1 / z^n)"
  using eisenstein_fun_conv_eisenstein_fun_aux[of n z]
  by (auto simp: eisenstein_fun_def eisenstein_fun_aux_def lattice0_def)

```

```

lemma eisenstein_fun_minus: "eisenstein_fun n (-z) = (-1)^n * eisenstein_fun n z"
  by (auto simp: eisenstein_fun_altdef eisenstein_fun_aux_minus lattice0_def)

```

```

uminus_in_lattice_iff
  power_minus' divide_simps)
  (auto simp: algebra_simps)

lemma eisenstein_fun_even_minus: "even n  $\implies$  eisenstein_fun n (-z) =
eisenstein_fun n z"
  by (simp add: eisenstein_fun_minus)

lemma eisenstein_fun_odd_minus: "odd n  $\implies$  eisenstein_fun n (-z) = -eisenstein_fun
n z"
  by (simp add: eisenstein_fun_minus)

lemma eisenstein_fun_has_field_derivative:
  assumes "n  $\geq$  3" "z  $\notin$   $\Lambda$ "
  shows "(eisenstein_fun n has_field_derivative -of_nat n * eisenstein_fun
(Suc n) z) (at z)"
proof -
  define n' where "n' = n - 1"
  have n': "n = Suc n'"
    using assms by (simp add: n'_def)
  have ev: "eventually ( $\lambda$ z. z  $\in$   $-\Lambda$ ) (nhds z)"
    using assms closed_lattice by (intro eventually_nhds_in_open) auto

  have "(( $\lambda$ z. eisenstein_fun_aux n z + 1 / z ^ n) has_field_derivative
-of_nat n * eisenstein_fun (Suc n) z) (at z)"
    using assms
    apply (auto intro!: derivative_eq_intros)
    apply (auto simp: eisenstein_fun_conv_eisenstein_fun_aux lattice_lattice0
field_simps n')
  done
  also have "?thesis  $\longleftrightarrow$  ?thesis"
    using assms by (intro has_field_derivative_cong_ev refl eventually_mono[OF
ev])
    (auto simp: eisenstein_fun_conv_eisenstein_fun_aux)
  finally show ?thesis .
qed

lemmas eisenstein_fun_has_field_derivative' [derivative_intros] =
  DERIV_chain2[OF eisenstein_fun_has_field_derivative]

lemma eisenstein_fun_holomorphic: "eisenstein_fun n holomorphic_on  $-\Lambda$ "
proof (cases "n  $\geq$  3")
  case True
  thus ?thesis using closed_lattice
  by (subst holomorphic_on_open) (auto intro!: eisenstein_fun_has_field_derivative)
qed (auto simp: eisenstein_fun_def [abs_def])

lemma higher_deriv_eisenstein_fun:
  assumes z: "z  $\notin$   $\Lambda$ " and n: "n  $\geq$  3"

```

```

shows "(deriv ^^ m) (eisenstein_fun n) z =
      (-1) ^ m * pochhammer (of_nat n) m * eisenstein_fun (n +
m) z"
using z n
proof (induction m arbitrary: z n)
  case 0
  thus ?case by simp
next
  case (Suc m z n)
  have ev: "eventually ( $\lambda z. z \in -\Lambda$ ) (nhds z)"
    using Suc.premis closed_lattice by (intro eventually_nhds_in_open)
  auto
  have "(deriv ^^ Suc m) (eisenstein_fun n) z = deriv ((deriv ^^ m) (eisenstein_fun
n)) z"
    by simp
  also have "... = deriv ( $\lambda z. (-1)^m * pochhammer (of\_nat\ n) m * eisenstein\_fun
(n + m) z$ ) z"
    by (intro deriv_cong_ev eventually_mono[OF ev]) (use Suc in auto)
  also have "... = (-1) ^ Suc m * pochhammer (of_nat n) (Suc m) * eisenstein_fun
(Suc (n + m)) z"
    using Suc.premis
    by (intro DERIV_imp_deriv)
      (auto intro!: derivative_eq_intros simp: pochhammer_Suc algebra_simps)
  finally show ?case
    by simp
qed

```

```

lemma eisenstein_fun_holomorphic' [holomorphic_intros]:
  assumes "f holomorphic_on A" " $\bigwedge z. z \in A \implies n < 3 \vee f z \notin \Lambda$ "
  shows " $(\lambda z. eisenstein\_fun\ n\ (f\ z))$  holomorphic_on A"
proof (cases "n  $\geq$  3")
  case True
  have "eisenstein_fun n  $\circ$  f holomorphic_on A"
    by (rule holomorphic_on_compose_gen assms eisenstein_fun_holomorphic)+
  (use assms True in auto)
  thus ?thesis by (simp add: o_def)
qed (auto simp: eisenstein_fun_def)

```

```

lemma eisenstein_fun_analytic: "eisenstein_fun n analytic_on  $-\Lambda$ "
  by (simp add: analytic_on_open closed_lattice open_Cmpl eisenstein_fun_holomorphic)

```

```

lemma eisenstein_fun_analytic' [analytic_intros]:
  assumes "f analytic_on A" " $\bigwedge z. z \in A \implies n < 3 \vee f z \notin \Lambda$ "
  shows " $(\lambda z. eisenstein\_fun\ n\ (f\ z))$  analytic_on A"
proof (cases "n  $\geq$  3")
  case True
  have "eisenstein_fun n  $\circ$  f analytic_on A"
    by (rule analytic_on_compose_gen assms True eisenstein_fun_analytic)+

```

```

(use assms True in auto)
  thus ?thesis by (simp add: o_def)
qed (auto simp: eisenstein_fun_def)

lemma eisenstein_fun_continuous_on: "n ≥ 3 ⇒ continuous_on (-Λ) (eisenstein_fun
n)"
  using holomorphic_on_imp_continuous_on eisenstein_fun_holomorphic by
blast

lemma eisenstein_fun_continuous_on' [continuous_intros]:
  assumes "continuous_on A f" "∧z. z ∈ A ⇒ n < 3 ∨ f z ∉ Λ"
  shows "continuous_on A (λz. eisenstein_fun n (f z))"
proof (cases "n ≥ 3")
  case True
  show ?thesis
    by (rule continuous_on_compose2[OF eisenstein_fun_continuous_on assms(1)])
      (use assms True in auto)
qed (auto simp: eisenstein_fun_def)

sublocale eisenstein_fun: complex_lattice_periodic ω1 ω2 "eisenstein_fun
n"
proof
  have *: "eisenstein_fun n (w + z) = eisenstein_fun n w" if "z ∈ Λ"
for w z
  proof (cases "n ≥ 3 ∧ w ∉ Λ")
  case True
  show ?thesis
    proof (rule has_sum_unique)
      show "((λω. 1 / (w - ω) ^ n) has_sum eisenstein_fun n w) Λ"
        by (rule eisenstein_fun_has_sum) (use True in auto)
      next
      have "((λω. 1 / (w + z - ω) ^ n) has_sum eisenstein_fun n (w +
z)) Λ"
        by (rule eisenstein_fun_has_sum) (use True that in auto)
      also have "?this ↔ ((λω. 1 / (w - ω) ^ n) has_sum eisenstein_fun
n (w + z)) Λ"
        by (rule has_sum_reindex_bij_witness[of _ "(+) z" "(+) (-z)"])
          (use that in <auto intro!: lattice_intros simp: algebra_simps>)
      finally show "((λω. 1 / (w - ω) ^ n) has_sum eisenstein_fun n (w
+ z)) Λ" .
    qed
  qed (use that in <auto simp: eisenstein_fun_def>)
  show "eisenstein_fun n (z + ω1) = eisenstein_fun n z"
    "eisenstein_fun n (z + ω2) = eisenstein_fun n z" for z
    by (rule *; simp)+
qed

lemma is_pole_eisenstein_fun:
  assumes "n ≥ 3" "z ∈ Λ"

```

```

shows "is_pole (eisenstein_fun n) z"
proof -
  have "eisenstein_fun_aux n - 0 → eisenstein_fun_aux n 0"
    by (rule isContD, rule analytic_at_imp_isCont) (auto intro!: analytic_intros)
  moreover have "is_pole (λw. 1 / w ^ n :: complex) 0"
    using assms is_pole_inverse_power[of n 0] by simp
  ultimately have "is_pole (λw. eisenstein_fun_aux n w + 1 / w ^ n) 0"
    unfolding is_pole_def by (rule tendsto_add_filterlim_at_infinity)
  also have "eventually (λw. w ∉ Λ) (at 0)"
    using not_islimpt_lattice by (auto simp: islimpt_iff_eventually)
  hence "eventually (λw. eisenstein_fun_aux n w + 1 / w ^ n = eisenstein_fun
n w) (at 0)"
    by eventually_elim (use assms in <auto simp: eisenstein_fun_altdef>)
  hence "is_pole (λw. eisenstein_fun_aux n w + 1 / w ^ n) 0 ↔ is_pole
(eisenstein_fun n) 0"
    by (intro is_pole_cong) auto
  also have "eisenstein_fun n = eisenstein_fun n ∘ (λw. w + z)"
    using assms by (auto simp: fun_eq_iff simp: rel_def uminus_in_lattice_iff
eisenstein_fun.lattice_cong)
  also have "is_pole ... 0 ↔ is_pole (eisenstein_fun n) z"
    by (simp add: is_pole_shift_0' o_def add commute)
  finally show ?thesis .
qed

sublocale eisenstein_fun: nicely_elliptic_function ω1 ω2 "eisenstein_fun
n"
proof
  show "eisenstein_fun n nicely_meromorphic_on UNIV"
  proof (cases "n ≥ 3")
    case True
      show ?thesis
      proof (rule nicely_meromorphic_onI_open)
        show "eisenstein_fun n analytic_on UNIV - Λ"
          using eisenstein_fun_analytic[of n] by (simp add: Compl_eq_Diff_UNIV)
        show "is_pole (eisenstein_fun n) z ∧ eisenstein_fun n z = 0" if
"z ∈ Λ" for z
          using that by (simp add: True eisenstein_fun_def is_pole_eisenstein_fun)
        show "isolated_singularity_at (eisenstein_fun n) z" for z
          proof (rule analytic_nhd_imp_isolated_singularity[of _ "-(Λ-{z})"])
            show "open (- (Λ - {z}))"
              by (intro open_Compl closed_subset_lattice) auto
          qed (auto intro!: analytic_intros)
        qed simp
      qed (auto simp: eisenstein_fun_def [abs_def] intro!: analytic_on_imp_nicely_meromorphic_o
qed

lemmas [elliptic_function_intros] =
  eisenstein_fun.elliptic_function_axioms eisenstein_fun.nicely_elliptic_function_axioms

```

end

7.2 Definition and basic properties

The Weierstraß \wp function is in a sense the most basic elliptic function, and we will see later on that all elliptic function can be written as a combination of \wp and \wp' .

Its derivative, as we noted before, is equal to our Eisenstein function for $n = 3$ (up to a constant factor -2). The function \wp itself is somewhat more awkward to define.

`context complex_lattice begin`

`lemma minus_lattice_eq: "uminus ' $\Lambda = \Lambda$ "`

`proof -`

`have "uminus ' $\Lambda \subseteq \Lambda$ "`

`by (auto simp: uminus_in_lattice_iff)`

`then show ?thesis`

`using equation_minus_iff by blast`

`qed`

`lemma minus_latticemz_eq: "uminus ' $\Lambda^* = \Lambda^*$ "`

`by (simp add: lattice0_def inj_on_def image_set_diff minus_lattice_eq)`

`lemma bij_minus_latticemz: "bij_betw uminus $\Lambda^* \Lambda^*$ "`

`by (simp add: bij_betw_def inj_on_def minus_latticemz_eq)`

`definition weierstrass_fun_deriv (" \wp' ") where`

`"weierstrass_fun_deriv z = -2 * eisenstein_fun 3 z"`

`sublocale weierstrass_fun_deriv: elliptic_function $\omega_1 \omega_2$ weierstrass_fun_deriv`

`unfolding weierstrass_fun_deriv_def by (intro elliptic_function_intros)`

`sublocale weierstrass_fun_deriv: nicely_elliptic_function $\omega_1 \omega_2$ weierstrass_fun_deriv`

`proof`

`show " \wp' nicely_meromorphic_on UNIV"`

`using eisenstein_fun.nicely_meromorphic unfolding weierstrass_fun_deriv_def`

`by (intro nicely_meromorphic_on_uminus nicely_meromorphic_on_cmult_left)`

`qed`

`lemmas [elliptic_function_intros] =`

`weierstrass_fun_deriv.elliptic_function_axioms weierstrass_fun_deriv.nicely_elliptic_func`

`lemma weierstrass_fun_deriv_minus [simp]: " $\wp' (-z) = -\wp' z$ "`

`by (simp add: weierstrass_fun_deriv_def eisenstein_fun_odd_minus)`

`lemma weierstrass_fun_deriv_has_field_derivative:`

`assumes "z $\notin \Lambda$ "`

```

shows "( $\wp'$  has_field_derivative 6 * eisenstein_fun 4 z) (at z)"
unfolding weierstrass_fun_deriv_def
using assms by (auto intro!: derivative_eq_intros)

lemma weierstrass_fun_deriv_holomorphic: " $\wp'$  holomorphic_on  $-\Lambda$ "
unfolding weierstrass_fun_deriv_def by (auto intro!: holomorphic_intros)

lemma weierstrass_fun_deriv_holomorphic' [holomorphic_intros]:
assumes "f holomorphic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda$ "
shows "( $\lambda z. \wp' (f z)$ ) holomorphic_on A"
using assms unfolding weierstrass_fun_deriv_def by (auto intro!: holomorphic_intros)

lemma weierstrass_fun_deriv_analytic: " $\wp'$  analytic_on  $-\Lambda$ "
unfolding weierstrass_fun_deriv_def by (auto intro!: analytic_intros)

lemma weierstrass_fun_deriv_analytic' [analytic_intros]:
assumes "f analytic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda$ "
shows "( $\lambda z. \wp' (f z)$ ) analytic_on A"
using assms unfolding weierstrass_fun_deriv_def by (auto intro!: analytic_intros)

lemma weierstrass_fun_deriv_continuous_on: "continuous_on  $(-\Lambda)$   $\wp'$ "
unfolding weierstrass_fun_deriv_def by (auto intro!: continuous_intros)

lemma weierstrass_fun_deriv_continuous_on' [continuous_intros]:
assumes "continuous_on A f" " $\bigwedge z. z \in A \implies f z \notin \Lambda$ "
shows "continuous_on A ( $\lambda z. \wp' (f z)$ )"
using assms unfolding weierstrass_fun_deriv_def by (auto intro!: continuous_intros)

lemma tendsto_weierstrass_fun_deriv [tendsto_intros]:
assumes "(f  $\longrightarrow$  c) F" "c  $\notin \Lambda$ "
shows "(( $\lambda z. \wp' (f z)$ )  $\longrightarrow$   $\wp' c$ ) F"
proof (rule continuous_on_tendsto_compose[OF _ assms(1)])
show "continuous_on  $(-\Lambda)$   $\wp'$ "
by (intro holomorphic_on_imp_continuous_on holomorphic_intros) auto
show "eventually ( $\lambda z. f z \in -\Lambda$ ) F"
by (rule eventually_compose_filterlim[OF _ assms(1)], rule eventually_nhds_in_open)
(use assms(2) in <auto intro: closed_lattice>)
qed (use assms(2) in auto)

```

The following is the Weierstraß function minus its pole at the origin. By convention, it returns 0 at all its remaining poles.

```

definition weierstrass_fun_aux :: "complex  $\Rightarrow$  complex" where
"weierstrass_fun_aux z = (if z  $\in \Lambda^*$  then 0 else ( $\sum_{\infty} \omega \in \Lambda^*. 1 / (z - \omega)^2 - 1 / \omega^2$ ))"

```

This is now the Weierstraß function. Again, it returns 0 at all its poles.

```

definition weierstrass_fun :: "complex  $\Rightarrow$  complex" (" $\wp$ ")
where " $\wp$  z = (if z  $\in \Lambda$  then 0 else  $1 / z^2 + \text{weierstrass\_fun\_aux } z$ )"

```

```

lemma weierstrass_fun_aux_0 [simp]: "weierstrass_fun_aux 0 = 0"
  by (simp add: weierstrass_fun_aux_def)

lemma weierstrass_fun_at_pole: " $\omega \in \Lambda \implies \wp \omega = 0$ "
  by (simp add: weierstrass_fun_def)

lemma
  fixes R :: real
  assumes "R > 0"
  shows weierstrass_fun_aux_converges_absolutely_uniformly_in_disk:
    "uniform_limit (cball 0 R)
      ( $\lambda X z. \sum_{\omega \in X} \text{norm } (1 / (z - \omega)^2 - 1 / \omega^2)$ )
      ( $\lambda z. \sum_{\omega \in \Lambda - \text{cball } 0 R} \text{norm } (1 / (z - \omega)^2 - 1 / \omega^2)$ )
      (finite_subsets_at_top ( $\Lambda - \text{cball } 0 R$ ))" (is
?th1)
  and weierstrass_fun_aux_converges_uniformly_in_disk:
    "uniform_limit (cball 0 R)
      ( $\lambda X z. \sum_{\omega \in X} 1 / (z - \omega)^2 - 1 / \omega^2$ )
      ( $\lambda z. \sum_{\omega \in \Lambda - \text{cball } 0 R} 1 / (z - \omega)^2 - 1 / \omega^2$ )
      (finite_subsets_at_top ( $\Lambda - \text{cball } 0 R$ ))" (is
?th2)
  proof -
    obtain M where M:
      " $M > 0$ " " $\bigwedge \omega z. [\omega \in \Lambda; \text{norm } \omega > R; \text{norm } z \leq R] \implies \text{norm } (z - \omega)$ 
       $\text{powr } -2 \leq M * \text{norm } \omega \text{ powr } -2$ "
      using weierstrass_summand_bound[of 2 R] assms by auto

    have 1: " $(\lambda \omega. 3 * M * R * \text{norm } \omega \text{ powr } -3)$  summable_on ( $\Lambda - \text{cball } 0 R$ )"
    proof -
      have " $(\lambda \omega. 1 / \text{norm } \omega \text{ powr } 3)$  summable_on  $\Lambda^*$ "
        using assms by (subst converges_absolutely_iff) auto
      hence " $(\lambda \omega. 3 * M * R * \text{norm } \omega \text{ powr } -3)$  summable_on  $\Lambda^*$ "
        by (intro summable_on_cmult_right) (auto simp: powr_minus field_simps)
      thus " $(\lambda \omega. 3 * M * R * \text{norm } \omega \text{ powr } -3)$  summable_on ( $\Lambda - \text{cball } 0 R$ )"
        by (rule summable_on_subset) (use assms in <auto simp: lattice0_def>)
    qed

    have 2: " $\text{norm } (1 / (z - \omega)^2 - 1 / \omega^2) \leq 3 * M * R * \text{norm } \omega \text{ powr } -3$ "
      if " $\omega \in \Lambda - \text{cball } 0 R$ " " $z \in \text{cball } 0 R$ " for  $\omega z$ 
    proof -
      from that have nz: " $\omega \neq 0$ " " $\omega \neq z$ "
        using <R > 0> by auto
      hence " $1 / (z - \omega)^2 - 1 / \omega^2 = z * (2 * \omega - z) / (\omega^2 * (z - \omega)^2)$ "
        using that by (auto simp: field_simps) (auto simp: power2_eq_square algebra_simps)
      also have " $\text{norm } \dots = \text{norm } z * \text{norm } (2 * \omega - z) / \text{norm } \omega ^ 2 * \text{norm }$ 
```

```

(z - ω) powr - 2"
  by (simp add: norm_divide norm_mult norm_power powr_minus divide_simps)
  also have "... ≤ R * (2 * norm ω + norm z) / norm ω ^ 2 * (M * norm
ω powr -2)"
    using assms that
    by (intro mult_mono frac_le mult_nonneg_nonneg M order.trans[OF
norm_triangle_ineq4]) auto
  also have "... = M * R * (2 + norm z / norm ω) / norm ω ^ 3"
    using nz by (simp add: field_simps powr_minus power2_eq_square power3_eq_cube)
  also have "... ≤ M * R * 3 / norm ω ^ 3"
    using nz assms M(1) that by (intro mult_left_mono divide_right_mono)
auto
  finally show ?thesis
    by (simp add: field_simps powr_minus)
qed

show ?th1 ?th2 unfolding weierstrass_fun_aux_def
  by (rule Weierstrass_m_test_general[OF _ 1]; use 2 in simp)+
qed

lemma weierstrass_fun_has_field_derivative_aux:
  fixes R :: real
  defines "F ≡ (λz. ∑∞ ω ∈ Λ - cball 0 R. 1 / (z - ω)2 - 1 / ω2)"
  defines "F' ≡ (λz. ∑∞ ω ∈ Λ - cball 0 R. 1 / (z - ω)3)"
  assumes "R > 0" "w ∈ ball 0 R"
  shows "(F has_field_derivative -2 * F' w) (at w)"
proof -
  have 1: "∀F n in finite_subsets_at_top (Λ - cball 0 R).
    continuous_on (cball 0 R) (λz. ∑ω ∈ n. 1 / (z - ω)2 - 1 /
ω2) ∧
    (∀z ∈ ball 0 R. ((λz. ∑ω ∈ n. 1 / (z - ω)2 - 1 / ω2) has_field_derivative
(∑ω ∈ n. -2 / (z - ω)3)) (at z))"
  apply (intro eventually_finite_subsets_at_top_weakI conjI continuous_intros
derivative_intros ballI)
  apply force
  apply (rule derivative_eq_intros refl | force)+
  apply (simp add: divide_simps, simp add: algebra_simps power4_eq_xxxx
power3_eq_cube)
  done

  have "uniform_limit (cball 0 R)
    (λX z. ∑ω ∈ X. 1 / (z - ω)2 - 1 / ω2)
    (λz. ∑∞ ω ∈ Λ - cball 0 R. 1 / (z - ω)2 - 1 / ω2)
    (finite_subsets_at_top (Λ - cball 0 R))"
  using assms by (intro weierstrass_fun_aux_converges_uniformly_in_disk)
auto
  also have "?this ↔ uniform_limit (cball 0 R) (λX z. ∑ω ∈ X. 1 / (z
- ω)2 - 1 / ω2) F"

```

```

      (finite_subsets_at_top (Λ - cball 0 R))"
    using assms unfolding F_def
    by (intro uniform_limit_cong eventually_finite_subsets_at_top_weakI)
      (auto simp: powr_minus powr_nat field_simps intro!: sum.cong infsum_cong)
  finally have 2: ... .

  have 3: "finite_subsets_at_top (Λ - cball 0 R) ≠ bot"
    by simp

  obtain g where g: "continuous_on (cball 0 R) F"
    "∧w. w ∈ ball 0 R ⇒ (F has_field_derivative g
w) (at w) ∧
      ((λω. -2 / (w - ω) ^ 3) has_sum g w) (Λ - cball
0 R)"
    unfolding has_sum_def using has_complex_derivative_uniform_limit[OF
1 2 3 <R > 0>] by auto

  have "((λω. (-2) * (1 / (w - ω) ^ 3)) has_sum (-2) * F' w) (Λ - cball
0 R)"
    unfolding F'_def using assms
    by (intro has_sum_cmult_right has_sum_infsum weierstrass_aux_converges_in_disk')
  auto
  moreover have "((λω. -2 * (1 / (w - ω) ^ 3)) has_sum g w) (Λ - cball
0 R)"
    using g(2)[of w] assms by simp
  ultimately have "g w = -2 * F' w"
    by (metis infsumI)
  thus "(F has_field_derivative -2 * F' w) (at w)"
    using g(2)[of w] assms by simp
qed

lemma norm_summable_weierstrass_fun_aux: "(λω. norm (1 / (z - ω)2 -
1 / ω2)) summable_on Λ"
proof -
  define R where "R = norm z + 1"
  have "(λω. norm (1 / (z - ω)2 - 1 / ω2)) summable_on (Λ - cball 0 R)"
    unfolding summable_iff_has_sum_infsum has_sum_def
    by (rule tendsto_uniform_limitI[OF weierstrass_fun_aux_converges_absolutely_uniformly_i
(auto simp: R_def add_nonneg_pos)
  hence "(λω. norm (1 / (z - ω)2 - 1 / ω2)) summable_on ((Λ - cball 0
R) ∪ (Λ ∩ cball 0 R))"
    by (intro summable_on_union[OF _ summable_on_finite]) (auto simp:
bounded_lattice_finite)
  also have "... = Λ"
    by blast
  finally show ?thesis .
qed

lemma summable_weierstrass_fun_aux: "(λω. 1 / (z - ω)2 - 1 / ω2) summable_on

```

```

 $\Lambda$ "
  using norm_summable_weierstrass_fun_aux by (rule abs_summable_summable)

lemma weierstrass_summable: "( $\lambda\omega. 1 / (z - \omega)^2 - 1 / \omega^2$ ) summable_on
 $\Lambda^*$ "
  by (rule summable_on_subset[OF summable_weierstrass_fun_aux]) (auto
simp: lattice0_def)

lemma weierstrass_fun_aux_has_sum:
  " $z \notin \Lambda^* \implies ((\lambda\omega. 1 / (z - \omega)^2 - 1 / \omega^2)$  has_sum weierstrass_fun_aux
z)  $\Lambda^*$ "
  unfolding weierstrass_fun_aux_def by (simp add: weierstrass_summable)

lemma weierstrass_fun_aux_has_field_derivative:
  defines "F  $\equiv$  weierstrass_fun_aux"
  defines "F'  $\equiv$  ( $\lambda z. \sum_{\omega \in \Lambda^*} 1 / (z - \omega)^3$ )"
  assumes z: " $z \notin \Lambda^*$ "
  shows "(F has_field_derivative -2 * eisenstein_fun_aux 3 z) (at z)"
proof -
  define R where "R = norm z + 1"
  have R: "R > 0" "norm z < R"
    by (auto simp: R_def add_nonneg_pos)
  have "finite ( $\Lambda \cap \text{cball } 0 \text{ } R$ )"
    by (simp add: bounded_lattice_finite)
  moreover have " $\Lambda^* \cap \text{cball } 0 \text{ } R \subseteq \Lambda \cap \text{cball } 0 \text{ } R$ "
    unfolding lattice0_def by blast
  ultimately have fin: "finite ( $\Lambda^* \cap \text{cball } 0 \text{ } R$ )"
    using finite_subset by blast

  define F1 where "F1 = ( $\lambda z. \sum_{\omega \in \Lambda - \text{cball } 0 \text{ } R} 1 / (z - \omega)^2 - 1 / \omega^2$ )"
  define F'1 where "F'1 = ( $\lambda z. \sum_{\omega \in \Lambda - \text{cball } 0 \text{ } R} 1 / (z - \omega)^3$ )"
  define F2 where "F2 = ( $\lambda z. \sum_{\omega \in \Lambda^* \cap \text{cball } 0 \text{ } R} 1 / (z - \omega)^2 - 1 / \omega^2$ )"
  define F'2 where "F'2 = ( $\lambda z. \sum_{\omega \in \Lambda^* \cap \text{cball } 0 \text{ } R} 1 / (z - \omega)^3$ )"

  have "(F1 has_field_derivative -2 * F'1 z) (at z)"
    unfolding F1_def F'1_def
    by (rule weierstrass_fun_has_field_derivative_aux) (auto simp: R_def
add_nonneg_pos)
  moreover have "(F2 has_field_derivative -2 * F'2 z) (at z)"
    unfolding F2_def F'2_def sum_distrib_left lattice0_def
    by (rule derivative_eq_intros refl sum.cong | use R z in <force simp:
lattice0_def>)+
    (simp add: divide_simps power3_eq_cube power4_eq_xxxx)
  ultimately have "(( $\lambda z. F1 z + F2 z$ ) has_field_derivative (-2 * F'1 z)
+ (-2 * F'2 z)) (at z)"
    by (intro derivative_intros)
  also have "?this  $\longleftrightarrow$  (F has_field_derivative (-2 * F'1 z) + (-2 * F'2
z)) (at z)"
  proof (intro has_field_derivative_cong_ev refl)

```

```

have "eventually ( $\lambda z'. z' \in -\Lambda^*$ ) (nhds z)"
  using z by (intro eventually_nhds_in_open) (auto simp: closed_lattice0)
thus " $\forall_F x$  in nhds z.  $x \in UNIV \longrightarrow F1 x + F2 x = F x$ "
proof eventually_elim
  case (elim z)
  have " $((\lambda \omega. 1 / (z - \omega)^2 - 1 / \omega^2)$  has_sum (F1 z + F2 z)) (( $\Lambda$ 
- cball 0 R)  $\cup$  ( $\Lambda^* \cap$  cball 0 R))"
    unfolding F1_def F2_def using R fin
    by (intro has_sum_Un_disjoint[OF has_sum_infsum has_sum_finite]
      summable_on_subset_banach[OF summable_weierstrass_fun_aux])
auto
  also have " $(\Lambda -$  cball 0 R)  $\cup$  ( $\Lambda^* \cap$  cball 0 R) =  $\Lambda^*$ "
    using R unfolding lattice0_def by auto
  finally show ?case using elim
    unfolding F1_def F2_def F_def weierstrass_fun_aux_def by (simp
add: infsumI)
  qed
qed auto
also have " $(-2 * F'1 z) + (-2 * F'2 z) = -2 * (F'1 z + F'2 z)$ "
  by (simp add: algebra_simps)
also have " $F'1 z + F'2 z = F' z$ "
proof -
  have " $((\lambda \omega. 1 / (z - \omega)^3)$  has_sum (F'1 z + F'2 z)) (( $\Lambda -$  cball 0
R)  $\cup$  ( $\Lambda^* \cap$  cball 0 R))"
    unfolding F'1_def F'2_def using R fin
    by (intro has_sum_Un_disjoint [OF has_sum_infsum has_sum_finite]
weierstrass_aux_converges') auto
  also have " $(\Lambda -$  cball 0 R)  $\cup$  ( $\Lambda^* \cap$  cball 0 R) =  $\Lambda^*$ "
    using R unfolding lattice0_def by auto
  finally show " $F'1 z + F'2 z = F' z$ "
    unfolding F'1_def F'2_def F'_def by (simp add: infsumI)
  qed
finally show ?thesis
  using assms by (simp add: eisenstein_fun_aux_def)
qed

lemmas weierstrass_fun_aux_has_field_derivative' [derivative_intros]
=
  weierstrass_fun_aux_has_field_derivative [THEN DERIV_chain2]

lemma weierstrass_fun_aux_holomorphic: "weierstrass_fun_aux holomorphic_on
 $-\Lambda^*$ "
  by (subst holomorphic_on_open)
  (auto intro!: weierstrass_fun_aux_has_field_derivative simp: closed_lattice0)

lemma weierstrass_fun_aux_holomorphic' [holomorphic_intros]:
  assumes "f holomorphic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda^*$ "
  shows " $(\lambda z. \text{weierstrass\_fun\_aux } (f z))$  holomorphic_on A"
proof -

```

```

    have "weierstrass_fun_aux ∘ f holomorphic_on A"
      by (rule holomorphic_on_compose_gen assms weierstrass_fun_aux_holomorphic)+
    (use assms in auto)
    thus ?thesis by (simp add: o_def)
  qed

```

```

lemma weierstrass_fun_aux_continuous_on: "continuous_on (-Λ*) weierstrass_fun_aux"
  using holomorphic_on_imp_continuous_on weierstrass_fun_aux_holomorphic
  by blast

```

```

lemma weierstrass_fun_aux_continuous_on' [continuous_intros]:
  assumes "continuous_on A f" "Λz. z ∈ A ⇒ f z ∉ Λ*"
  shows "continuous_on A (λz. weierstrass_fun_aux (f z))"
  by (rule continuous_on_compose2[OF weierstrass_fun_aux_continuous_on
  assms(1)]) (use assms in auto)

```

```

lemma weierstrass_fun_aux_analytic: "weierstrass_fun_aux analytic_on
-Λ*"
  by (simp add: analytic_on_open closed_lattice0 open_Cmpl weierstrass_fun_aux_holomorphic)

```

```

lemma weierstrass_fun_aux_analytic' [analytic_intros]:
  assumes "f analytic_on A" "Λz. z ∈ A ⇒ f z ∉ Λ*"
  shows "(λz. weierstrass_fun_aux (f z)) analytic_on A"
proof -
  have "weierstrass_fun_aux ∘ f analytic_on A"
    by (rule analytic_on_compose_gen assms weierstrass_fun_aux_analytic)+
  (use assms in auto)
  thus ?thesis by (simp add: o_def)
qed

```

```

lemma deriv_weierstrass_fun_aux:
  "z ∉ Λ* ⇒ deriv weierstrass_fun_aux z = -2 * eisenstein_fun_aux 3
z"
  by (rule DERIV_imp_deriv derivative_eq_intros refl | assumption)+ simp

```

```

lemma weierstrass_fun_has_field_derivative:
  fixes R :: real
  assumes z: "z ∉ Λ"
  shows "(φ has_field_derivative φ' z) (at z)"
proof -
  note [derivative_intros] = weierstrass_fun_aux_has_field_derivative
  from z have [simp]: "z ≠ 0" "z ∉ Λ*"
    by (auto simp: lattice0_def)
  define D where "D = -2 / z ^ 3 - 2 * eisenstein_fun_aux 3 z"

  have "((λz. 1 / z2 + weierstrass_fun_aux z) has_field_derivative D)
(at z)" unfolding D_def

```

```

    by (rule derivative_eq_intros refl | simp)+ (simp add: divide_simps
power3_eq_cube power4_eq_xxxx)
  also have "?this  $\longleftrightarrow$  (weierstrass_fun has_field_derivative D) (at z)"
  proof (intro has_field_derivative_cong_ev refl)
    have "eventually ( $\lambda z. z \in -\Lambda$ ) (nhds z)"
      using closed_lattice z by (intro eventually_nhds_in_open) auto
    thus "eventually ( $\lambda z. z \in UNIV \longrightarrow 1 / z \wedge 2 + \text{weierstrass\_fun\_aux}$ 
 $z = \wp z$ ) (nhds z)"
      by eventually_elim (simp add: weierstrass_fun_def)
  qed auto
  also have "D = -2 * eisenstein_fun 3 z"
    using z by (simp add: eisenstein_fun_conv_eisenstein_fun_aux D_def)
  finally show ?thesis by (simp add: weierstrass_fun_deriv_def)
qed

```

```

lemmas weierstrass_fun_has_field_derivative' [derivative_intros] =
  weierstrass_fun_has_field_derivative [THEN DERIV_chain2]

```

```

lemma weierstrass_fun_holomorphic: " $\wp$  holomorphic_on  $-\Lambda$ "
  by (subst holomorphic_on_open)
  (auto intro!: weierstrass_fun_has_field_derivative simp: closed_lattice)

```

```

lemma weierstrass_fun_holomorphic' [holomorphic_intros]:
  assumes "f holomorphic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda$ "
  shows " $(\lambda z. \text{weierstrass\_fun } (f z))$  holomorphic_on A"
proof -
  have "weierstrass_fun  $\circ$  f holomorphic_on A"
    by (rule holomorphic_on_compose_gen assms weierstrass_fun_holomorphic)+
  (use assms in auto)
  thus ?thesis by (simp add: o_def)
qed

```

```

lemma weierstrass_fun_analytic: " $\wp$  analytic_on  $-\Lambda$ "
  by (simp add: analytic_on_open closed_lattice open_Compl weierstrass_fun_holomorphic)

```

```

lemma weierstrass_fun_analytic' [analytic_intros]:
  assumes "f analytic_on A" " $\bigwedge z. z \in A \implies f z \notin \Lambda$ "
  shows " $(\lambda z. \wp (f z))$  analytic_on A"
proof -
  have " $\wp \circ f$  analytic_on A"
    by (rule analytic_on_compose_gen assms weierstrass_fun_analytic)+
  (use assms in auto)
  thus ?thesis by (simp add: o_def)
qed

```

```

lemma weierstrass_fun_continuous_on: "continuous_on  $(-\Lambda)$  weierstrass_fun"
  using holomorphic_on_imp_continuous_on weierstrass_fun_holomorphic by
blast

```

```

lemma weierstrass_fun_continuous_on' [continuous_intros]:
  assumes "continuous_on A f" "\z. z \in A \implies f z \notin \Lambda"
  shows "continuous_on A (\lambda z. \wp (f z))"
  by (rule continuous_on_compose2[OF weierstrass_fun_continuous_on assms(1)])
  (use assms in auto)

lemma tendsto_weierstrass_fun [tendsto_intros]:
  assumes "(f \longrightarrow c) F" "c \notin \Lambda"
  shows "((\lambda z. \wp (f z)) \longrightarrow \wp c) F"
proof (rule continuous_on_tendsto_compose[OF _ assms(1)])
  show "continuous_on (-\Lambda) \wp"
    by (intro holomorphic_on_imp_continuous_on holomorphic_intros) auto
  show "eventually (\lambda z. f z \in -\Lambda) F"
    by (rule eventually_compose_filterlim[OF _ assms(1)], rule eventually_nhds_in_open)
    (use assms(2) in <auto intro: closed_lattice>)
qed (use assms(2) in auto)

lemma deriv_weierstrass_fun:
  assumes "z \notin \Lambda"
  shows "deriv \wp z = \wp' z"
  by (rule DERIV_imp_deriv weierstrass_fun_has_field_derivative refl assms)+

```

The following identity is to be read with care: for $\omega = 0$ we get a division by zero, so the term $1 / \omega^2$ simply gets dropped.

```

lemma weierstrass_fun_eq:
  assumes "z \notin \Lambda"
  shows "\wp z = (\sum_{\omega \in \Lambda} (1 / (z - \omega)^2) - 1 / \omega^2)"
proof -
  have *: "((\lambda \omega. 1 / (z - \omega)^2 - 1 / \omega^2) has_sum \wp z - 1 / z^2) \Lambda*"
    using has_sum_infsum [OF weierstrass_summable, of z] assms
    by (simp add: weierstrass_fun_def weierstrass_fun_aux_def lattice0_def)
  have "((\lambda \omega. 1 / (z - \omega)^2 - 1 / \omega^2) has_sum ((1 / (z - 0)^2 - 1 / 0^2)
+ (\wp z - 1 / z^2))) \Lambda"
    unfolding lattice_lattice0 by (rule has_sum_insert) (use * in auto)
  then show ?thesis
    by (simp add: infsumI)
qed

```

7.3 Ellipticity and poles

It can easily be seen from its definition that \wp is an even elliptic function with a double pole at each lattice point and no other poles. Thus it has order 2.

Its derivative is consequently an odd elliptic function with a triple pole at each lattice point, no other poles, and order 3.

The results in this section correspond to Apostol's Theorems 1.9 and 1.10.

```

lemma weierstrass_fun_minus: "\wp (-z) = \wp z"

```

```

proof (cases "z ∈ Λ")
  case False
    have "(∑∞ ω ∈ Λ*. 1 / (- z - ω)2 - 1 / ω2) = (∑∞ ω ∈ Λ*. (1 / (z - ω)2 - 1 / ω2))"
      by (rule infsum_reindex_bij_witness[of _ uminus uminus])
      (auto intro!: lattice_intros simp: uminus_in_lattice0_iff power2_commute add_ac)
    thus ?thesis using False
      by (auto simp: weierstrass_fun_def weierstrass_fun_aux_def uminus_in_lattice0_iff lattice_lattice0)
  qed (auto simp: weierstrass_fun_at_pole uminus_in_lattice_iff)

sublocale weierstrass_fun: complex_lattice_periodic ω1 ω2 ϕ
proof
  have *: "ϕ (z + ω) = ϕ z" if ω: "ω ∈ {ω1, ω2}" for ω z
  proof (cases "z ∈ Λ")
    case z: True
      thus ?thesis
        by (subst (1 2) weierstrass_fun_at_pole) (use ω in <auto intro!: lattice_intros>)
    next
      case z: False
        from ω have ω' [simp, intro]: "ω ∈ Λ"
          by auto
        define f where "f = (λz. ϕ (z + ω) - ϕ z)"
        with <z ∉ Λ> have "(f has_field_derivative 0) (at z)" if "z ∉ Λ"
        for z
          proof -
            from that and ω have "(f has_field_derivative (ϕ' (z + ω) - ϕ' z)) (at z)"
              unfolding f_def by (auto intro!: derivative_eq_intros)
            also have "ϕ' (z + ω) = ϕ' z"
              by (rule weierstrass_fun_deriv.lattice_cong) (auto simp: rel_def)
            finally show ?thesis
              by simp
          qed
        hence deriv: "∀x∈UNIV - Λ - {}. (f has_field_derivative 0) (at x)"
          by blast
        have cont: "continuous_on (UNIV - Λ) f"
          by (auto simp: f_def intro!: continuous_intros)

        have *: "connected (UNIV - Λ)" "open (UNIV - Λ)" "countable ({} :: complex set)"
          by (auto simp: closed_lattice intro!: connected_open_diff_countable)

        obtain c where c: "∧z. z ∈ UNIV - Λ ⇒ f z = c"
          using DERIV_zero_connected_constant[OF * cont deriv] by blast

        have "ω / 2 ∉ Λ"

```

```

    using of_ω12_coords_in_lattice_iff ω by (auto simp: in_lattice_conv_ω12_coords)
  hence "f (-ω / 2) = c"
    by (intro c) (auto simp: uminus_in_lattice_iff)
  also have "f (-ω / 2) = 0"
    by (simp add: f_def weierstrass_fun_minus)
  finally have "f z = 0"
    using c that z by auto
  thus ?thesis
    by (simp add: f_def)
qed

show "φ (z + ω1) = φ z" "φ (z + ω2) = φ z" for z
  by (rule *; simp)+
qed

lemma zorder_weierstrass_fun_pole:
  assumes "ω ∈ Λ"
  shows "zorder φ ω = -2"
proof -
  define R where "R = Inf_para"
  have R: "R > 0"
    using Inf_para_pos by (auto simp: R_def)
  have R': "ball 0 R ∩ Λ* = {}"
    using Inf_para_le_norm by (force simp: R_def)

  have "zorder weierstrass_fun ω = zorder (λz. weierstrass_fun (z + ω))
0"
    by (rule zorder_shift)
  also have "(λz. weierstrass_fun (z + ω)) = weierstrass_fun"
    by (intro ext weierstrass_fun.lattice_cong) (auto simp: rel_def assms)
  also have "zorder weierstrass_fun 0 = -2"
proof (rule zorder_eqI)
  show "open (ball 0 R :: complex set)" "(0 :: complex) ∈ ball 0 R"
    using R by auto
  show "(λz. 1 + weierstrass_fun_aux z * z ^ 2) holomorphic_on ball
0 R" using R'
    by (intro holomorphic_intros holomorphic_on_subset[OF weierstrass_fun_aux_holomorphic
auto
show "∧w. [w ∈ ball 0 R; w ≠ 0]
⇒ φ w =
(1 + weierstrass_fun_aux w * w^2) *
(w - 0) powi - 2"
using R'
by (auto simp: weierstrass_fun_def field_simps power_numeral_reduce
powi_minus_numeral_reduce
lattice0_def)
qed auto
finally show ?thesis .

```

qed

lemma is_pole_weierstrass_fun:

assumes $\omega: \omega \in \Lambda$

shows "is_pole $\wp \omega$ "

proof -

have "is_pole $\wp 0$ "

proof -

have "eventually ($\lambda z. z \in -\Lambda^*$) (nhds 0)"

using closed_lattice0 by (intro eventually_nhds_in_open) auto

hence ev: "eventually ($\lambda z. z \notin \Lambda$) (at 0)"

unfolding eventually_at_filter by eventually_elim (auto simp: lattice0_def)

have " $\Lambda - \Lambda^* = \{0\}$ " " $\Lambda^* \subseteq \Lambda$ "

by (auto simp: insert_Diff_if lattice_lattice0)

hence "weierstrass_fun_aux holomorphic_on $-\Lambda^*$ "

by (auto intro!: holomorphic_intros)

hence "continuous_on $(-\Lambda^*)$ weierstrass_fun_aux"

using holomorphic_on_imp_continuous_on by blast

moreover have " $0 \in -\Lambda^*$ "

by (auto simp: lattice0_def)

ultimately have "(weierstrass_fun_aux \longrightarrow weierstrass_fun_aux 0)

(at 0)"

using closed_lattice0 by (metis at_within_open closed_open continuous_on_def)

moreover have "filterlim ($\lambda z::\text{complex}. 1 / z ^ 2$) at_infinity (at

0)"

using is_pole_inverse_power[of 2 0] by (simp add: is_pole_def)

ultimately have "filterlim ($\lambda z. \text{weierstrass_fun_aux } z + 1 / z ^ 2$)

at_infinity (at 0)"

by (rule tendsto_add_filterlim_at_infinity)

also have "?this \longleftrightarrow filterlim weierstrass_fun at_infinity (at 0)"

by (intro filterlim_cong refl eventually_mono[OF ev]) (auto simp:

weierstrass_fun_def)

finally show ?thesis

by (simp add: is_pole_def)

qed

also have " $\wp = \wp \circ (\lambda z. z + \omega)$ "

by (auto simp: fun_eq_iff rel_def assms uminus_in_lattice_iff

intro!: weierstrass_fun.lattice_cong)

also have "is_pole ... 0 \longleftrightarrow is_pole $\wp \omega$ "

by (simp add: o_def is_pole_shift_0' add_ac)

finally show "is_pole $\wp \omega$ " .

qed

sublocale weierstrass_fun: nicely_elliptic_function $\omega_1 \omega_2 \wp$

proof

show " \wp nicely_meromorphic_on UNIV"

proof (rule nicely_meromorphic_onI_open)

show " \wp analytic_on UNIV - Λ "

by (auto intro!: analytic_intros)

```

next
  fix z assume "z ∈ Λ"
  thus "is_pole ϕ z ∧ ϕ z = 0"
    by (auto simp: weierstrass_fun_at_pole is_pole_weierstrass_fun)
next
  show "isolated_singularity_at ϕ z" for z
  proof (rule analytic_nhd_imp_isolated_singularity[of _ "-(Λ-{z})"])
    show "open (- (Λ - {z}))"
      by (intro open_Compl closed_subset_lattice) auto
    qed (auto intro!: analytic_intros)
  qed auto
qed

sublocale weierstrass_fun: even_elliptic_function ω1 ω2 ϕ
  by standard (auto simp: weierstrass_fun_minus)

lemmas [elliptic_function_intros] =
  weierstrass_fun.elliptic_function_axioms
  weierstrass_fun.nicely_elliptic_function_axioms

lemma is_pole_weierstrass_fun_iff: "is_pole ϕ z ↔ z ∈ Λ"
  by (meson ComplI analytic_on_analytic_at is_pole_weierstrass_fun
    nicely_elliptic_function.analytic_at_iff_not_pole
    weierstrass_fun.nicely_elliptic_function_axioms weierstrass_fun_analytic)

lemma is_pole_weierstrass_fun_deriv_iff: "is_pole ϕ' z ↔ z ∈ Λ"
proof -
  have "eventually (λw. w ∉ Λ) (at z)"
    using islimpt_iff_eventually not_islimpt_lattice by auto
  hence "eventually (λw. ϕ' w = deriv ϕ w) (at z)"
    by eventually_elim (simp add: deriv_weierstrass_fun)
  hence "is_pole ϕ' z ↔ is_pole (deriv ϕ) z"
    by (rule is_pole_cong) auto
  also have "... ↔ is_pole ϕ z"
    by (rule is_pole_deriv_iff) (auto intro!: meromorphic_intros)
  also have "... ↔ z ∈ Λ"
    by (rule is_pole_weierstrass_fun_iff)
  finally show ?thesis .
qed

lemma zorder_weierstrass_fun_deriv_pole:
  assumes "z ∈ Λ"
  shows "zorder ϕ' z = -3"
proof -
  have "eventually (λw. w ∉ Λ) (at z)"
    using islimpt_iff_eventually not_islimpt_lattice by auto
  hence "eventually (λw. ϕ' w = deriv ϕ w) (at z)"
    by eventually_elim (simp add: deriv_weierstrass_fun)
  hence "zorder ϕ' z = zorder (deriv ϕ) z"

```

```

    by (rule zorder_cong) auto
  also have "... = zorder  $\wp$  z - 1"
    by (subst zorder_deriv) (auto simp: is_pole_weierstrass_fun_iff assms)
  also have "... = -3"
    using assms by (simp add: zorder_weierstrass_fun_pole)
  finally show ?thesis .
qed

```

```

lemma order_weierstrass_fun [simp]: "elliptic_order  $\wp$  = 2"
proof -
  have "elliptic_order  $\wp$  = ( $\sum z \in \text{period\_parallelogram } 0 \cap \Lambda. \text{nat } (-\text{zorder } \wp z)$ )"
    unfolding elliptic_order_def by (rule sum.cong) (auto simp: is_pole_weierstrass_fun_iff)
  also have "period\_parallelogram  $0 \cap \Lambda = \{0\}$ "
    by (auto elim!: latticeE simp: period\_parallelogram_altdef zero_prod_def)
  finally show ?thesis
    by (simp add: zorder_weierstrass_fun_pole)
qed

```

```

lemma order_weierstrass_fun_deriv [simp]: "elliptic_order  $\wp'$  = 3"
proof -
  have "elliptic_order  $\wp'$  = ( $\sum z \in \text{period\_parallelogram } 0 \cap \Lambda. \text{nat } (-\text{zorder } \wp' z)$ )"
    unfolding elliptic_order_def by (rule sum.cong) (auto simp: is_pole_weierstrass_fun_der)
  also have "period\_parallelogram  $0 \cap \Lambda = \{0\}$ "
    by (auto elim!: latticeE simp: period\_parallelogram_altdef zero_prod_def)
  also have "( $\sum z \in \{0\}. \text{nat } (-\text{zorder } \wp' z)$ ) = nat (-zorder  $\wp' 0)$ "
    by simp
  finally show ?thesis
    by (simp add: zorder_weierstrass_fun_deriv_pole)
qed

```

```

sublocale weierstrass_fun: nonconst_nicely_elliptic_function  $\omega_1 \omega_2 \wp$ 
  by standard auto

```

```

sublocale weierstrass_fun_deriv: nonconst_nicely_elliptic_function  $\omega_1$ 
 $\omega_2$  " $\wp'$ "
  by standard auto

```

7.4 The numbers e_1, e_2, e_3

The values of \wp at the half-periods $\frac{1}{2}\omega_1$, $\frac{1}{2}\omega_2$, and $\frac{1}{2}(\omega_1 + \omega_2)$ are exactly the roots of the polynomial $4X^3 - g_2X - g_3$.

We call these values e_1, e_2, e_3 .

```

definition number_e1:: "complex" (" $e_1$ ") where
  " $e_1 \equiv \wp (\omega_1 / 2)$ "

```

```

definition number_e2:: "complex" (" $e_2$ ") where

```

```

"e2 ≡ ϕ (ω2 / 2)"

definition number_e3 :: "complex" ("e3") where
  "e3 ≡ ϕ ((ω1 + ω2) / 2)"

lemmas number_e123_defs = number_e1_def number_e2_def number_e3_def

definition modulus :: complex where
  "modulus = (e3 - e2) / (e1 - e2)"

The half-lattice points are those that are equivalent to one of the three points
 $\frac{\omega_1}{2}$ ,  $\frac{\omega_2}{2}$ , and  $\frac{\omega_1 + \omega_2}{2}$ .

lemma to_fund_parallelogram_half_period:
  assumes "ω ∉ Λ" "2 * ω ∈ Λ"
  shows "to_fund_parallelogram ω ∈ {ω1 / 2, ω2 / 2, (ω1 + ω2) / 2}"
proof -
  from assms(2) obtain m n where "2 * ω = of_ω12_coords (of_int m, of_int n)"
  by (elim latticeE)
  hence mn: "ω = of_ω12_coords (of_int m / 2, of_int n / 2)"
  by (auto simp: of_ω12_coords_def field_simps)
  have [simp]: "of_ω12_coords (1 / 2, 1 / 2) = (ω1 + ω2) / 2"
  by (simp add: of_ω12_coords_def field_simps)
  have "odd m ∨ odd n"
  using assms(1) unfolding mn by (auto simp: in_lattice_conv_ω12_coords
elim!: evenE)
  thus ?thesis
  by (cases "even m"; cases "even n")
  (auto elim!: evenE oddE simp: mn to_fund_parallelogram_def add_divide_distrib)
qed

lemma rel_half_period:
  assumes "ω ∉ Λ" "2 * ω ∈ Λ"
  shows "∃ ω' ∈ {ω1 / 2, ω2 / 2, (ω1 + ω2) / 2}. rel ω ω'"
proof -
  have "rel ω (to_fund_parallelogram ω)"
  by auto
  with to_fund_parallelogram_half_period[OF assms] show ?thesis
  by blast
qed

lemma weierstass_fun_deriv_half_period_eq_0:
  assumes "ω ∈ Λ"
  shows "ϕ' (ω / 2) = 0"
  using weierstrass_fun_deriv.lattice_cong[of "-ω/2" "ω/2"] <ω ∈ Λ>
  by (simp add: rel_def uminus_in_lattice_iff)

lemma weierstass_fun_deriv_half_root_eq_0 [simp]:
  "ϕ' (ω1 / 2) = 0" "ϕ' (ω2 / 2) = 0" "ϕ' ((ω1 + ω2) / 2) = 0"

```

```

    by (rule weierstrass_fun_deriv_half_period_eq_0; simp)+

lemma weierstrass_fun_at_half_period:
  assumes " $\omega \in \Lambda$ " " $\omega / 2 \notin \Lambda$ "
  shows " $\wp(\omega / 2) \in \{e_1, e_2, e_3\}$ "
proof -
  have " $\exists \omega' \in \{\omega_1 / 2, \omega_2 / 2, (\omega_1 + \omega_2) / 2\}. \text{rel } (\omega / 2) \omega'$ "
    using rel_half_period[of " $\omega / 2$ "] assms by auto
  thus ?thesis
    unfolding number_e123_defs using weierstrass_fun.lattice_cong by
blast
qed

lemma weierstrass_fun_at_half_period':
  assumes " $2 * \omega \in \Lambda$ " " $\omega \notin \Lambda$ "
  shows " $\wp \omega \in \{e_1, e_2, e_3\}$ "
  using weierstrass_fun_at_half_period[of " $2 * \omega$ "] assms by simp

 $\wp'$  has a simple zero at each half-lattice point, and no other zeros.

lemma weierstrass_fun_deriv_eq_0_iff:
  assumes " $z \notin \Lambda$ "
  shows " $\wp' z = 0 \iff 2 * z \in \Lambda$ "
proof
  assume " $\wp' z = 0$ "
  define z' where "z' = to_fund_parallelogram z"
  have z': " $\wp' z' = 0$ " " $z' \notin \Lambda$ " " $z' \in \text{period\_parallelogram } 0$ "
    using < $\wp' z = 0$ > assms by (auto simp: z'_def weierstrass_fun_deriv.eval_to_fund_parallelogram)
  have [simp]: " $\wp' \neq (\lambda_. 0)$ "
    using weierstrass_fun_deriv.elliptic_order_eq_0_iff_no_poles by auto
  have "{ $\omega_1 / 2, \omega_2 / 2, (\omega_1 + \omega_2) / 2, z'$ }  $\subseteq$  { $z \in \text{period\_parallelogram } 0. \text{isolated\_zero } \wp' z$ }"
    using z' by (auto simp: period_parallelogram_altdef  $\omega_1 2$  coords.add
      weierstrass_fun_deriv.isolated_zero_iff is_pole_weierstrass_fun_deriv_iff)

  hence "card { $\omega_1 / 2, \omega_2 / 2, (\omega_1 + \omega_2) / 2, z'$ }  $\leq$  card { $z \in \text{period\_parallelogram } 0. \text{isolated\_zero } \wp' z$ }"
    by (intro card_mono weierstrass_fun_deriv.finite_zeros_in_parallelogram)
  also have "...  $\leq 3$ "
    using weierstrass_fun_deriv.card_zeros_le_order[of 0] by simp
  finally have " $2 * z' \in \{\omega_1, \omega_2, \omega_1 + \omega_2\}$ "
    by (auto simp: card_insert_if split: if_splits)
  also have "...  $\subseteq \Lambda$ "
    by auto
  finally have " $2 * z' \in \Lambda$ " .
  thus " $2 * z \in \Lambda$ " unfolding z'_def
    by (metis rel_to_fund_parallelogram_right mult_2 rel_add rel_lattice_trans_right)
next
  assume " $2 * z \in \Lambda$ "
  thus " $\wp' z = 0$ "

```

```

    using assms weierstrass_fun_deriv_half_period_eq_0[of "2*z"] by auto
qed

lemma zorder_weierstrass_fun_deriv_zero:
  assumes "z ∉ Λ" "2 * z ∈ Λ"
  shows "zorder φ' z = 1"
proof -
  have *: "φ' ≠ (λ_. 0)"
    using is_pole_weierstrass_fun_deriv_iff[of 0] by auto
  note ** [simp] = weierstrass_fun_deriv.isolated_zero_iff[OF *] is_pole_weierstrass_fun_de

  define w1 w2 w3 where "w1 = ω1 / 2" "w2 = ω2 / 2" "w3 = (ω1 + ω2) /
2"
  note w123_defs = this
  have [simp]: "w1 ∉ Λ" "w2 ∉ Λ" "w3 ∉ Λ" "2 * w1 ∈ Λ" "2 * w2 ∈ Λ"
"2 * w3 ∈ Λ"
    and "distinct [w1, w2, w3]"
    by (auto simp: w123_defs add_divide_distrib in_lattice_conv_ω12_coords
ω12_coords.add)
  define A where "A = {w1, w2, w3}"
  have [simp, intro]: "finite A" and [simp]: "card A = 3"
    using <distinct [w1, w2, w3]> by (auto simp: A_def)

  have in_parallelogram: "A ⊆ period_parallelogram 0"
    unfolding A_def period_parallelogram_altdef by (auto simp: w123_defs
ω12_coords.add)
  have [simp]: "φ' w = 0" if "w ∈ A" for w
    by (subst weierstrass_fun_deriv_eq_0_iff) (use that in <auto simp:
A_def>)
  have [intro]: "isolated_zero φ' w" if "w ∈ A" for w
    using that by (subst **) (auto simp: A_def)

  have "(∑ w∈A. nat (zorder φ' w)) ≤ elliptic_order φ'"
    unfolding weierstrass_fun_deriv.zeros_eq_elliptic_order[of 0, symmetric]
using in_parallelogram
    by (intro sum_mono2 weierstrass_fun_deriv.finite_zeros_in_parallelogram)
(auto simp: A_def)
  also have "... = 3"
    by simp
  finally have le_3: "(∑ w∈A. nat (zorder φ' w)) ≤ 3" .

  have pos: "zorder φ' w > 0" if "w ∈ A" for w
    by (rule zorder_isolated_zero_pos) (use that in <auto intro!: analytic_intros
simp: A_def>)

  have zorder_eq_1: "zorder φ' w = 1" if "w ∈ A" for w
proof (rule ccontr)
  assume *: "zorder φ' w ≠ 1"
  have "(∑ w∈A. nat (zorder φ' w)) > (∑ w∈A. 1)"

```

```

proof (rule sum_strict_mono_strong[of _ w])
  show "nat (zorder  $\wp$  w)  $\geq$  1" if "w  $\in$  A" for w
    using pos[of w] that by auto
qed (use pos[of w] * <w  $\in$  A> in auto)
with le_3 show False
  by simp
qed

from assms obtain w where w: "w  $\in$  A" "rel z w"
  using rel_half_period[of z] unfolding A_def w123_defs by metis
with zorder_eq_1[of w] show ?thesis
  using weierstrass_fun_deriv.zorder.lattice_cong by metis
qed

end

```

7.5 Injectivity of \wp

```

context complex_lattice
begin

```

The function \wp is almost injective in the sense that $\wp(u) = \wp(v)$ iff $u \sim v$ or $u \sim -v$. Another way to phrase this is that it is injective inside period half-parallelograms.

This is Exercise 1.3(a) in Apostol's book.

```

theorem weierstrass_fun_eq_iff:
  assumes "u  $\notin$   $\Lambda$ " "v  $\notin$   $\Lambda$ "
  shows "  $\wp$  u =  $\wp$  v  $\iff$  rel u v  $\vee$  rel u (-v)"
proof (intro iffI)
  assume "rel u v  $\vee$  rel u (-v)"
  thus "  $\wp$  u =  $\wp$  v"
    by (metis weierstrass_fun.lattice_cong weierstrass_fun_minus)
next
  assume *: "  $\wp$  u =  $\wp$  v"
  define c where "c =  $\wp$  u"
  define f where "f = ( $\lambda$ z.  $\wp$  z - c)"
  interpret f: elliptic_function_affine  $\omega$ 1  $\omega$ 2  $\wp$  1 "-c" f
  proof -
    show "elliptic_function_affine  $\omega$ 1  $\omega$ 2  $\wp$  1"
      by standard auto
  qed (simp_all add: f_def)
  interpret f: even_elliptic_function  $\omega$ 1  $\omega$ 2 f
    by standard (simp_all add: f_def weierstrass_fun_minus)
  interpret f: elliptic_function_remove_sings  $\omega$ 1  $\omega$ 2 f ..

  have ana: "f analytic_on {x}" if "x  $\notin$   $\Lambda$ " for x using that
    unfolding f_def by (auto intro!: analytic_intros)
  have ana': "remove_sings f analytic_on {x}" if "x  $\notin$   $\Lambda$ " for x using
that

```

```

by (auto simp: f.remove_sings.analytic_at_iff_not_pole f.is_pole_affine_iff

      is_pole_weierstrass_fun_iff)
have [simp]: "remove_sings f x = f x" if "x  $\notin$   $\Lambda$ " for x
  using ana[OF that] by simp
have [simp]: "f u = 0" "f v = 0"
  using * by (auto simp: f_def c_def)

have order: "elliptic_order f = 2"
  by (simp add: f.order_affine_eq)
have nz: "remove_sings f  $\neq$  ( $\lambda$ _. 0)"
proof
  assume "remove_sings f = ( $\lambda$ _. 0)"
  hence "remove_sings f constant_on UNIV"
    by (auto simp: constant_on_def)
  thus False
    using f.remove_sings.elliptic_order_eq_0_iff order by simp
qed
have zero_f_iff [simp]: "isolated_zero f z  $\longleftrightarrow$  f z = 0" if "z  $\notin$   $\Lambda$ "
for z
  using that ana f.affine.isolated_zero_analytic_iff
    f.affine.elliptic_order_eq_0_iff_const_cospars local.order
by auto

define u' where "u' = to_half_fund_parallelogram u"
define v' where "v' = to_half_fund_parallelogram v"
define Z where "Z = {z  $\in$  period_parallelogram 0. z  $\notin$   $\Lambda$   $\wedge$  f z = 0}"
define Z1 where "Z1 = {z  $\in$  half_fund_parallelogram. z  $\notin$   $\Lambda$   $\wedge$  f z =
0}"
define Z2 where "Z2 = to_fund_parallelogram ' uminus ' Z1"

have Z: " $(\sum z \in Z. \text{nat } (zorder f z)) = 2$ " "finite Z"
proof -
  have " $(\sum z \in \{z \in \text{period\_parallelogram } 0. \text{isolated\_zero } (\text{remove\_sings } f) z\}.
\text{nat } (zorder (\text{remove\_sings } f) z)) = 2 \wedge
\text{finite } \{z \in \text{period\_parallelogram } 0. \text{isolated\_zero } (\text{remove\_sings } f) z\}$ "
    using order f.remove_sings.zeros_eq_elliptic_order[of 0]
      f.remove_sings.finite_zeros_in_parallelogram[of 0] by simp
  also have "{z  $\in$  period_parallelogram 0. isolated_zero (remove_sings
f) z} =
{z  $\in$  period_parallelogram 0. z  $\notin$   $\Lambda$   $\wedge$  remove_sings f z =
0}"
    by (subst f.remove_sings.isolated_zero_iff)
      (use nz f.is_pole_affine_iff is_pole_weierstrass_fun_iff in auto)
  also have "... = {z  $\in$  period_parallelogram 0. z  $\notin$   $\Lambda$   $\wedge$  f z = 0}"
    by (intro Collect_cong conj_cong) auto
  finally show " $(\sum z \in Z. \text{nat } (zorder f z)) = 2$ " "finite Z"

```

```

    unfolding Z_def by auto
qed
have "finite Z1"
  by (rule finite_subset[OF _ Z(2)])
    (use half_fund_parallelogram_subset_period_parallelogram in <auto
simp: Z_def Z1_def>)
  hence "finite Z2"
    by (simp_all add: Z2_def)
note finite = <finite Z> <finite Z1> <finite Z2>

have subset: "Z1  $\subseteq$  Z" "Z2  $\subseteq$  Z"
  unfolding Z_def Z1_def Z2_def
  using half_fund_parallelogram_subset_period_parallelogram
  by (auto simp: uminus_in_lattice_iff f.affine.eval_to_fund_parallelogram
f.even)

have "card Z1 + card Z2 = card (Z1  $\cup$  Z2) + card (Z1  $\cap$  Z2)"
  by (rule card_Un_Int) (use finite in auto)
also have "card Z2 = card Z1" unfolding Z2_def image_image
  by (intro card_image inj_onI)
    (auto simp: Z1_def rel_minus_iff intro: rel_in_half_fund_parallelogram_imp_eq)
also have "card (Z1  $\cup$  Z2)  $\leq$  card Z"
  by (intro card_mono finite) (use subset in auto)
also have "... = ( $\sum z \in Z. 1$ )"
  by simp
also have "card (Z1  $\cap$  Z2) = ( $\sum z \in Z1 \cap Z2. 1$ )"
  by simp
also have "... = ( $\sum z \in Z. \text{if } z \in Z1 \cap Z2 \text{ then } 1 \text{ else } 0$ )"
  by (intro sum.mono_neutral_cong_left) (use finite subset in auto)
also have "( $\sum z \in Z. 1$ ) + ... = ( $\sum z \in Z. \text{if } z \in Z1 \cap Z2 \text{ then } 2 \text{ else } 1$ )"
  by (subst sum.distrib [symmetric], intro sum.cong) auto
also have "...  $\leq$  ( $\sum z \in Z. \text{nat } (zorder f z)$ )"
proof (intro sum_mono)
  fix z assume z: "z  $\in$  Z"
  hence "zorder f z > 0"
    by (intro zorder_isolated_zero_pos ana) (auto simp: Z_def)
  moreover have "zorder f z  $\geq$  2" if z: "z  $\in$  Z1  $\cap$  Z2"
  proof -
    obtain z' where "to_fund_parallelogram (-z')  $\in$  half_fund_parallelogram"
    "z'  $\notin$   $\Lambda$ "
      "z = to_fund_parallelogram (-z')" "z'  $\in$  half_fund_parallelogram"
    using z by (auto simp: Z1_def Z2_def uminus_in_lattice_iff)
    hence "2 * z  $\in$   $\Lambda$ "
      by (metis in_half_fund_parallelogram_imp_half_lattice
rel_in_half_fund_parallelogram_imp_eq rel_to_fund_parallelogram_left)
    moreover have " $\neg (\forall \approx z. f z = 0)$ " using nz
      using f.affine.elliptic_order_eq_0_iff_const_cosparses local.order
  by auto
  ultimately have "even (zorder f z)"

```

```

      by (intro f.even_zorder)
      with <zorder f z > 0> show "zorder f z ≥ 2"
      by presburger
    qed
  ultimately show "(if z ∈ Z1 ∩ Z2 then 2 else 1) ≤ nat (zorder f
z)"
    by auto
  qed
  also have "... = 2"
    by (fact Z(1))
  finally have "card Z1 ≤ 1"
    by simp
  moreover have "card {u', v'} ≤ card Z1" using assms
    by (intro card_mono finite) (auto simp: Z1_def u'_def v'_def f.eval_to_half_fund_parallel)
  ultimately have "card {u', v'} ≤ 1"
    by simp
  hence "u' = v'"
    by (cases "u' = v'") simp_all
  thus "rel u v ∨ rel u (-v)"
    unfolding u'_def v'_def by (simp add: to_half_fund_parallelogram_eq_iff)
qed

```

It is also surjective. Together with the fact that it is doubly periodic and even, this means that it takes on every value exactly once inside its period triangles, or twice within its period parallelograms. Note however that the multiplicities of the poles on the lattice points and of the values e_1, e_2, e_3 at the half-lattice points are 2.

```

lemma surj_weierstrass_fun:
  obtains z where "z ∈ period_parallelogram w - Λ" "φ z = c"
  using weierstrass_fun.surj[of w c]
  by (auto simp: is_pole_weierstrass_fun_iff)

```

```

lemma surj_weierstrass_fun_deriv:
  obtains z where "z ∈ period_parallelogram w - Λ" "φ' z = c"
  using weierstrass_fun_deriv.surj[of w c]
  by (auto simp: is_pole_weierstrass_fun_deriv_iff)

```

end

```

context complex_lattice_swap
begin

```

```

lemma weierstrass_fun_aux_swap [simp]: "swap.weierstrass_fun_aux = weierstrass_fun_aux"
  unfolding weierstrass_fun_aux_def [abs_def] swap.weierstrass_fun_aux_def
[abs_def] by auto

```

```

lemma weierstrass_fun_swap [simp]: "swap.weierstrass_fun = weierstrass_fun"
  unfolding weierstrass_fun_def [abs_def] swap.weierstrass_fun_def [abs_def]

```

by auto

```
lemma number_e1_swap [simp]: "swap.number_e1 = number_e2"
  and number_e2_swap [simp]: "swap.number_e2 = number_e1"
  and number_e3_swap [simp]: "swap.number_e3 = number_e3"
  unfolding number_e2_def swap.number_e1_def number_e1_def swap.number_e2_def
    number_e3_def swap.number_e3_def by (simp_all add: add_ac)
```

end

7.6 Invariance under lattice transformations

We show how various concepts related to lattices (e.g. the Weierstraß \wp function, the numbers e_1, e_2, e_3) transform under various transformations of the lattice. Namely: complex conjugation, swapping the generators, stretching/rotation, and unimodular Möbius transforms.

```
locale complex_lattice_cnj = complex_lattice
begin
```

```
sublocale cnj: complex_lattice "cnj  $\omega_1$ " "cnj  $\omega_2$ "
  by unfold_locales (use fundpair in auto)
```

```
lemma bij_betw_lattice_cnj: "bij_betw cnj lattice cnj.lattice"
  by (rule bij_betwI[of _ _ _ cnj])
  (auto elim!: latticeE cnj.latticeE simp: of_ $\omega_{12}$ _coords_def cnj.of_ $\omega_{12}$ _coords_def
    intro!: lattice_intros cnj.lattice_intros)
```

```
lemma bij_betw_lattice0_cnj: "bij_betw cnj lattice0 cnj.lattice0"
  unfolding lattice0_def cnj.lattice0_def
  by (intro bij_betw_DiffI bij_betw_lattice_cnj) auto
```

```
lemma lattice_cnj_eq: "cnj.lattice = cnj ' lattice"
  using bij_betw_lattice_cnj by (auto simp: bij_betw_def)
```

```
lemma lattice0_cnj_eq: "cnj.lattice0 = cnj ' lattice0"
  using bij_betw_lattice0_cnj by (auto simp: bij_betw_def)
```

```
lemma eisenstein_fun_aux_cnj: "cnj.eisenstein_fun_aux n z = cnj (eisenstein_fun_aux
n (cnj z))"
  unfolding eisenstein_fun_aux_def cnj.eisenstein_fun_aux_def
  by (subst infsum_reindex_bij_betw[OF bij_betw_lattice0_cnj, symmetric])
  (auto simp flip: infsum_cnj simp: lattice0_cnj_eq in_image_cnj_iff)
```

```
lemma weierstrass_fun_aux_cnj: "cnj.weierstrass_fun_aux z = cnj (weierstrass_fun_aux
(cnj z))"
  unfolding weierstrass_fun_aux_def cnj.weierstrass_fun_aux_def
  by (subst infsum_reindex_bij_betw[OF bij_betw_lattice0_cnj, symmetric])
  (auto simp flip: infsum_cnj simp: lattice0_cnj_eq in_image_cnj_iff)
```

```

lemma weierstrass_fun_cnj: "cnj.weierstrass_fun z = cnj (weierstrass_fun
(cnj z))"
  unfolding weierstrass_fun_def cnj.weierstrass_fun_def
  by (auto simp: lattice_cnj_eq in_image_cnj_iff weierstrass_fun_aux_cnj)

lemma number_e1_cnj [simp]: "cnj.number_e1 = cnj number_e1"
  and number_e2_cnj [simp]: "cnj.number_e2 = cnj number_e2"
  and number_e3_cnj [simp]: "cnj.number_e3 = cnj number_e3"
  by (simp_all add: number_e1_def cnj.number_e1_def number_e2_def
      cnj.number_e2_def number_e3_def cnj.number_e3_def
  weierstrass_fun_cnj)

lemma modulus_cnj [simp]: "cnj.modulus = cnj modulus"
  by (simp add: modulus_def cnj.modulus_def)

end

locale complex_lattice_stretch = complex_lattice +
  fixes c :: complex
  assumes stretch_nonzero: "c ≠ 0"
begin

sublocale stretched: complex_lattice "c * ω1" "c * ω2"
  by unfold_locales (use fundpair in <auto simp: stretch_nonzero fundpair_def>)

lemma stretched_of_ω12_coords: "stretched.of_ω12_coords ab = c * of_ω12_coords
ab"
  unfolding stretched.of_ω12_coords_def of_ω12_coords_def
  by (auto simp: case_prod_unfold algebra_simps)

lemma stretched_ω12_coords: "stretched.ω12_coords ab = ω12_coords (ab
/ c)"
  using stretch_nonzero stretched_of_ω12_coords
  by (metis mult.commute nonzero_divide_eq_eq of_ω12_coords_ω12_coords
  stretched.ω12_coords_eqI)

lemma stretched_ω1_coord: "stretched.ω1_coord ab = ω1_coord (ab / c)"
  and stretched_ω2_coord: "stretched.ω2_coord ab = ω2_coord (ab / c)"
  using stretched_ω12_coords[of ab] by (simp_all add: stretched.ω12_coords_def
  ω12_coords_def)

lemma mult_into_stretched_lattice: "(*) c ∈ Λ → stretched.lattice"
  by (auto elim!: latticeE simp: stretched.in_lattice_conv_ω12_coords

      stretched_ω12_coords_zero_prod_def)

lemma mult_into_stretched_lattice': "(*) (inverse c) ∈ stretched.lattice

```

```

→  $\Lambda$ "
proof -
  interpret inv: complex_lattice_stretch "c *  $\omega_1$ " "c *  $\omega_2$ " "inverse c"
  by unfold_locales (use stretch_nonzero in auto)
  from inv.mult_into_stretched_lattice show ?thesis
  by (simp add: inv.stretched.lattice_def stretched.lattice_def field_simps
stretch_nonzero)
qed

lemma bij_betw_stretch_lattice: "bij_betw ((* c) lattice stretched.lattice"
proof (rule bij_betwI[of _ _ _ "(* (inverse c)"]])
  show "(* c)  $\in \Lambda \rightarrow$  stretched.lattice"
  by (rule mult_into_stretched_lattice)
  show "(* (inverse c)  $\in$  stretched.lattice  $\rightarrow \Lambda$ "
  by (rule mult_into_stretched_lattice')
qed (auto simp: stretch_nonzero)

lemma bij_betw_stretch_lattice0:
  "bij_betw ((* c) lattice0 stretched.lattice0"
  unfolding lattice0_def stretched.lattice0_def
  by (intro bij_betw_DiffI bij_betw_stretch_lattice) auto

lemma in_stretch_lattice_iff: "z  $\in$  stretched.lattice  $\longleftrightarrow$  z / c  $\in$  lattice"
proof
  assume "z  $\in$  stretched.lattice"
  hence "inverse c * z  $\in$  lattice"
  using mult_into_stretched_lattice' by blast
  thus "z / c  $\in$  lattice"
  by (simp add: field_simps)
next
  assume "z / c  $\in$  lattice"
  hence "c * (z / c)  $\in$  stretched.lattice"
  using mult_into_stretched_lattice by blast
  thus "z  $\in$  stretched.lattice"
  using stretch_nonzero by (auto simp: field_simps)
qed

lemma in_stretch_lattice0_iff: "z  $\in$  stretched.lattice0  $\longleftrightarrow$  z / c  $\in$ 
lattice0"
  by (auto simp: stretched.lattice0_def lattice0_def in_stretch_lattice_iff
stretch_nonzero)

lemma weierstrass_fun_aux_stretch: "stretched.weierstrass_fun_aux z =
weierstrass_fun_aux (z / c) / c ^ 2"
proof (cases "z  $\in$  stretched.lattice0")
  case True
  thus ?thesis using stretch_nonzero
  by (auto simp: in_stretch_lattice0_iff stretched.weierstrass_fun_aux_def
weierstrass_fun_aux_def)

```

```

next
  case False
  hence *: "z / c ∉ lattice0"
    by (auto simp: in_stretch_lattice0_iff stretch_nonzero)
  have "((λω. 1 / (z - ω)2 - 1 / ω2) has_sum stretched.weierstrass_fun_aux
z) stretched.lattice0"
    by (rule stretched.weierstrass_fun_aux_has_sum) fact
  also have "?this ↔ ((λω. 1 / (z - c * ω)2 - 1 / (c * ω)2) has_sum
      stretched.weierstrass_fun_aux z) lattice0"
    by (intro has_sum_reindex_bij_betw [symmetric] bij_betw_stretch_lattice0)
  also have "... ↔ ((λω. (1 / (z / c - ω)2 - 1 / ω2) / c ^ 2) has_sum
      stretched.weierstrass_fun_aux z) lattice0"
    by (intro has_sum_cong) (auto simp: field_simps lattice0_def stretch_nonzero)
  finally have "((λω. (1 / (z / c - ω)2 - 1 / ω2) / c ^ 2) has_sum
      stretched.weierstrass_fun_aux z) lattice0" .
  moreover have "((λω. (1 / (z / c - ω)2 - 1 / ω2) / c ^ 2) has_sum
      weierstrass_fun_aux (z / c) / c ^ 2) lattice0"
    by (intro has_sum_divide_const weierstrass_fun_aux_has_sum) fact
  ultimately show ?thesis
    using has_sum_unique by blast
qed

lemma weierstrass_fun_stretch: "stretched.weierstrass_fun z = weierstrass_fun
(z / c) / c ^ 2"
  by (auto simp: stretched.weierstrass_fun_def weierstrass_fun_def weierstrass_fun_aux_stre
      in_stretch_lattice0_iff divide_simps)

lemma number_e1_stretch [simp]: "stretched.number_e1 = number_e1 / c
^ 2"
  by (simp add: stretched.number_e1_def number_e1_def weierstrass_fun_stretch
stretch_nonzero)

lemma number_e2_stretch [simp]: "stretched.number_e2 = number_e2 / c
^ 2"
  by (simp add: stretched.number_e2_def number_e2_def weierstrass_fun_stretch
stretch_nonzero)

lemma number_e3_stretch [simp]: "stretched.number_e3 = number_e3 / c
^ 2"
  by (simp add: stretched.number_e3_def number_e3_def weierstrass_fun_stretch
      stretch_nonzero add_divide_distrib)

lemma modulus_stretch [simp]: "stretched.modulus = modulus"
  using stretch_nonzero
  unfolding stretched.modulus_def modulus_def number_e1_stretch number_e2_stretch

```

```

number_e3_stretch
  by (simp add: divide_simps)

end

locale unimodular_moebius_transform_lattice = complex_lattice + unimodular_moebius_transform
begin

definition  $\omega_1'$  where " $\omega_1' = \text{of\_int } c * \omega_2 + \text{of\_int } d * \omega_1$ "
definition  $\omega_2'$  where " $\omega_2' = \text{of\_int } a * \omega_2 + \text{of\_int } b * \omega_1$ "

sublocale transformed: complex_lattice  $\omega_1'$   $\omega_2'$ 
proof unfold_locales
  have "Im ( $\varphi \tau$ )  $\neq 0$ "
    using fundpair Im_transform_zero_iff[of  $\tau$ ] unfolding ratio_def
    by (auto simp: fundpair_def complex_is_Real_iff)
  also have " $\varphi \tau = \omega_2' / \omega_1'$ "
    by (simp add:  $\varphi$ _def  $\omega_1'$ _def  $\omega_2'$ _def moebius_def ratio_def divide_simps)
  finally show "fundpair ( $\omega_1'$ ,  $\omega_2'$ )"
    by (simp add:  $\varphi$ _def  $\omega_1'$ _def  $\omega_2'$ _def moebius_def ratio_def field_simps

        fundpair_def complex_is_Real_iff)
qed

lemma transformed_lattice_subset: "transformed.lattice  $\subseteq$  lattice"
proof safe
  fix z assume "z  $\in$  transformed.lattice"
  then obtain m n where mn: "z = of_int m *  $\omega_1'$  + of_int n *  $\omega_2'$ "
    by (elim transformed.latticeE) (auto simp: transformed.of_ $\omega_12$ _coords_def)
  also have "of_int m *  $\omega_1'$  + of_int n *  $\omega_2'$  =
    of_int (d * m + b * n) *  $\omega_1$  + of_int (c * m + a * n) *  $\omega_2$ "
    by (simp add: algebra_simps  $\omega_1'$ _def  $\omega_2'$ _def)
  finally show "z  $\in$  lattice"
    by (auto intro!: lattice_intros simp: ring_distrib mult.assoc)
qed

lemma transformed_lattice_eq: "transformed.lattice = lattice"
proof -
  interpret inverse_unimodular_moebius_transform a b c d ..
  interpret inv: unimodular_moebius_transform_lattice  $\omega_1'$   $\omega_2'$  d "-b" "-c"
  a ..
  have [simp]: "inv. $\omega_1'$  =  $\omega_1$ " "inv. $\omega_2'$  =  $\omega_2$ "
    unfolding inv. $\omega_1'$ _def inv. $\omega_2'$ _def unfolding  $\omega_1'$ _def  $\omega_2'$ _def of_int_minus
  using unimodular
    by (simp_all add: algebra_simps flip: of_int_mult)

  have "inv.transformed.lattice  $\subseteq$  transformed.lattice"
    by (rule inv.transformed_lattice_subset)

```

```

    also have "inv.transformed.lattice = lattice"
      unfolding inv.transformed.lattice_def unfolding lattice_def by simp
    finally show ?thesis
      using transformed_lattice_subset by blast
qed

lemma transformed_lattice0_eq: "transformed.lattice0 = lattice0"
  by (simp add: transformed.lattice0_def lattice0_def transformed_lattice_eq)

lemma eisenstein_fun_aux_transformed [simp]: "transformed.eisenstein_fun_aux
= eisenstein_fun_aux"
  by (intro ext) (simp add: transformed.eisenstein_fun_aux_def eisenstein_fun_aux_def
transformed_lattice0_eq)

lemma weierstrass_fun_aux_transformed [simp]: "transformed.weierstrass_fun_aux
= weierstrass_fun_aux"
  by (intro ext, unfold weierstrass_fun_aux_def transformed.weierstrass_fun_aux_def

transformed_lattice0_eq) simp_all

lemma weierstrass_fun_transformed [simp]: "transformed.weierstrass_fun
= weierstrass_fun"
  by (intro ext, simp add: weierstrass_fun_def transformed.weierstrass_fun_def
transformed_lattice_eq)

end

locale complex_lattice_apply_modgrp = complex_lattice +
  fixes f :: modgrp
begin

sublocale unimodular_moebius_transform_lattice
  ω1 ω2 "modgrp_a f" "modgrp_b f" "modgrp_c f" "modgrp_d f"
  rewrites "modgrp.as_modgrp = (λx. x)" and "modgrp.φ = apply_modgrp"
  by unfold_locales simp_all

end

```

7.7 Construction of arbitrary elliptic functions from \wp

In this section we will show that any elliptic function can be written as a combination of \wp and \wp' . The key step is to show that every even elliptic function can be written as a rational function of \wp .

The first step is to show that if $w \notin \Lambda$, the function $f(z) = \wp(z) - \wp(w)$ has a double zero at w if w is a half-lattice point and simple zeros at $\pm w$ otherwise, and no other zeros.

```

locale weierstrass_fun_minus_const = complex_lattice +

```

```

fixes w :: complex and f :: "complex  $\Rightarrow$  complex"
assumes not_in_lattice: "w  $\notin$   $\Lambda$ "
defines "f  $\equiv$  ( $\lambda z$ .  $\wp z - \wp w$ )"
begin

sublocale elliptic_function_affine  $\omega_1$   $\omega_2$   $\wp$  1 "- $\wp$  w" f
  unfolding f_def by unfold_locales (auto simp: f_def)

lemmas order_eq = order_affine_eq
lemmas is_pole_iff = is_pole_affine_iff
lemmas zorder_pole_eq = zorder_pole_affine

lemma isolated_zero_iff: "isolated_zero f z  $\longleftrightarrow$  rel z w  $\vee$  rel z (-w)"
proof (cases "z  $\in$   $\Lambda$ ")
  case False
  hence "f analytic_on {z}"
    unfolding f_def by (auto intro!: analytic_intros)
  moreover have " $\neg(\forall z. f z = 0)$ "
    using affine.elliptic_order_eq_0_iff_const_cosparse order_affine_eq
  by auto
  ultimately have "isolated_zero f z  $\longleftrightarrow$   $\wp z = \wp w$ "
    by (subst affine.isolated_zero_analytic_iff) (auto simp: f_def)
  also have "...  $\longleftrightarrow$  rel z w  $\vee$  rel z (-w)"
    by (rule weierstrass_fun_eq_iff) (use not_in_lattice False in auto)
  finally show ?thesis .
next
  case True
  thus ?thesis
    using not_in_lattice is_pole_iff is_pole_weierstrass_fun pole_is_not_zero
      pre_complex_lattice.rel_lattice_trans_left uminus_in_lattice_iff
  by blast
qed

lemma zorder_zero_eq:
  assumes "rel z w  $\vee$  rel z (-w)"
  shows "zorder f z = (if 2 * w  $\in$   $\Lambda$  then 2 else 1)"
proof (cases "2 * w  $\in$   $\Lambda$ ")
  case False
  have z: "z  $\notin$   $\Lambda$ "
    using assms not_in_lattice pre_complex_lattice.rel_lattice_trans_left
      uminus_in_lattice_iff by blast
  have z': "2 * z  $\notin$   $\Lambda$ "
    using assms False z
  by (metis minus_zero minus_minus not_in_lattice weierstrass_fun_deriv.lattice_cong
    weierstrass_fun_deriv_eq_0_iff weierstrass_fun_deriv_minus z)
  have "zorder f z = 1"
proof (rule zorder_zero_eqI')
  show "f analytic_on {z}"
    using not_in_lattice z by (auto simp: f_def intro!: analytic_intros)

```

```

next
  show "(deriv ^^ i) f z = 0" if "i < nat 1" for i
    using that z not_in_lattice assms by (auto simp: f_def weierstrass_fun_eq_iff)
next
  have "deriv f z = ϕ' z" unfolding f_def
    by (rule DERIV_imp_deriv) (use z in <auto intro!: derivative_eq_intros>)
  also have "ϕ' z ≠ 0"
    using not_in_lattice False z assms z' by (auto simp: weierstrass_fun_deriv_eq_0_iff)
  finally show "(deriv ^^ nat 1) f z ≠ 0"
    by simp
qed auto
with False show ?thesis
  by simp
next
case True
have z: "z ∉ Λ"
  using assms not_in_lattice pre_complex_lattice.rel_lattice_trans_left
    uminus_in_lattice_iff by blast
have z': "2 * z ∈ Λ"
  using assms True z
  by (metis minus_zero minus_minus not_in_lattice weierstrass_fun_deriv.lattice_cong
    weierstrass_fun_deriv_eq_0_iff weierstrass_fun_deriv_minus z)

have "eventually (λw. f w ≠ 0) (at 0)"
  by (simp add: affine.avoid' order_affine_eq)
moreover have "eventually (λw. w ∉ Λ) (at 0)"
  using islimpt_iff_eventually not_islimpt_lattice by auto
ultimately have ev: "eventually (λw. f w ≠ 0 ∧ w ∉ Λ) (at 0)"
  by eventually_elim auto
obtain z0 where z0: "z0 ∉ Λ" "f z0 ≠ 0"
  using eventually_happens[OF ev] by auto

have "(∑ z ∈ {z}. nat (zorder f z)) ≤
  (∑ z' | z' ∈ period_parallelogram z ∧ isolated_zero f z'. nat
(zorder f z'))"
  using assms by (intro sum_mono2 affine.finite_zeros_in_parallelogram)
(auto simp: isolated_zero_iff)
also have "... = 2"
  using affine.zeros_eq_elliptic_order[of z] order_eq by simp
finally have "zorder f z ≤ 2"
  by simp

moreover have "zorder f z ≥ int 2"
proof (rule zorder_geI)
  show "f holomorphic_on -Λ"
    by (auto intro!: holomorphic_intros simp: f_def)
next
  show "z0 ∈ -Λ" "f z0 ≠ 0"
    using z0 by auto

```

```

next
  show "open (- $\Lambda$ )"
    using closed_lattice by auto
next
  have " $\Lambda$  sparse_in UNIV"
    using not_islimpt_lattice sparse_in_def by blast
  hence "connected (UNIV -  $\Lambda$ )"
    by (intro sparse_imp_connected) auto
  also have "UNIV -  $\Lambda$  = - $\Lambda$ "
    by auto
  finally show "connected (- $\Lambda$ )" .
next
  have "deriv f z =  $\wp$ ' z"
    by (rule DERIV_imp_deriv) (auto simp: f_def intro!: derivative_eq_intros
z)
  also have "... = 0"
    using True z z' weierstrass_fun_deriv_eq_0_iff by blast
  finally have "deriv f z = 0" .
  moreover have "f z = 0"
    using not_in_lattice z z' assms by (simp add: f_def weierstrass_fun_eq_iff)
  ultimately show "(deriv ^^ n) f z = 0" if "n < 2" for n
    using that by (cases n) auto
qed (use z in <auto intro!: analytic_intros simp: f_def>)
ultimately have "zorder f z = 2"
  by linarith
thus ?thesis
  using True by simp
qed

lemma zorder_zero_eq':
  assumes "z  $\notin$   $\Lambda$ "
  shows "zorder f z = (if rel z w  $\vee$  rel z (-w) then if 2 * w  $\in$   $\Lambda$  then
2 else 1 else 0)"
proof (cases "rel z w  $\vee$  rel z (-w)")
  case True
  thus ?thesis
    using zorder_zero_eq[OF True] by auto
next
  case False
  have "f analytic_on {z}"
    using assms by (auto simp: f_def intro!: analytic_intros)
  moreover have "f z  $\neq$  0"
    using assms not_in_lattice False by (auto simp: f_def weierstrass_fun_eq_iff)
  ultimately have "zorder f z = 0"
    by (intro zorder_eq_0I) auto
  thus ?thesis
    using False by simp
qed

```

end

```

lemma (in complex_lattice) zorder_weierstrass_fun_minus_const:
  assumes "w ∉ Λ" "z ∉ Λ"
  shows "zorder (λz. ϕ z - ϕ w) z =
        (if rel z w ∨ rel z (-w) then if 2 * w ∈ Λ then 2 else 1
 else 0)"
proof -
  interpret weierstrass_fun_minus_const ω1 ω2 w "λz. ϕ z - ϕ w"
  by unfold_locales (use assms in auto)
  show ?thesis
  using zorder_zero_eq'[of z] assms by simp
qed

```

We now construct an elliptic function

$$g(z) = \prod_{w \in A} (\wp(z) - \wp(w))^{h(w)}$$

where $A \subseteq \mathbb{C} \setminus \Lambda$ is finite and $h : A \rightarrow \mathbb{Z}$.

We will examine what the zeros and poles of this functions are and what their multiplicities are.

This is roughly Exercise 1.3(b) in Apostol's book.

```

locale elliptic_function_construct = complex_lattice +
  fixes A :: "complex set" and h :: "complex ⇒ int" and g :: "complex
⇒ complex"
  assumes finite [intro]: "finite A" and no_lattice_points: "A ∩ Λ =
{}"
  defines "g ≡ (λz. (∏ w ∈ A. (ϕ z - ϕ w) powi h w))"
begin

```

```

sublocale elliptic_function ω1 ω2 g
  unfolding g_def by (intro elliptic_function_intros)

```

```

sublocale even_elliptic_function ω1 ω2 g
  by standard (simp add: g_def weierstrass_fun_minus)

```

```

lemma no_lattice_points': "w ∉ Λ" if "w ∈ A" for w
  using no_lattice_points that by blast

```

```

lemma eq_0_iff: "g z = 0 ⟷ (∃ w ∈ A. h w ≠ 0 ∧ (rel z w ∨ rel z (-w)))"
if "z ∉ Λ" for z
  using finite that by (auto simp: g_def weierstrass_fun_eq_iff no_lattice_points')

```

```

lemma nonzero_almost_everywhere: "eventually (λz. g z ≠ 0) (cosparse
UNIV)"
proof -

```

```

have "{z. g z = 0} ⊆ Λ ∪ (⋃ w ∈ A. (+) w ' Λ) ∪ (⋃ w ∈ A. (+) (-w) '
Λ)"
  using eq_0_iff by (force simp: rel_def)
moreover have "... sparse_in UNIV"
  by (intro sparse_in_union' sparse_in_UN_finite finite_imageI
      finite_sparse_in_translate_UNIV lattice_sparse)
ultimately have "{z. g z = 0} sparse_in UNIV"
  using sparse_in_subset2 by blast
thus ?thesis
  by (simp add: eventually_cosparse)
qed

lemma eventually_nonzero_at: "eventually (λz. g z ≠ 0) (at z)"
  using nonzero_almost_everywhere by (auto simp: eventually_cosparse_open_eq)

lemma zorder_eq:
  assumes z: "z ∉ Λ"
  shows "zorder g z =
      (∑ w ∈ A. if rel z w ∨ rel z (-w) then if 2*w ∈ Λ then 2
* h w else h w else 0)"
proof -
  have [simp]: "w ∉ Λ" if "w ∈ A" for w
    using no_lattice_points that by blast

  have "zorder g z = (∑ x ∈ A. zorder (λz. (ϕ z - ϕ x) powi h x) z)"
    unfolding g_def
  proof (rule zorder_prod)
    show "∀F z in at z. (∏ x ∈ A. (ϕ z - ϕ x) powi h x) ≠ 0"
      using eventually_nonzero_at[of z] by (simp add: g_def)
  qed (auto intro!: meromorphic_intros)
  also have "... = (∑ w ∈ A. if rel z w ∨ rel z (-w) then if 2*w ∈ Λ then
2 * h w else h w else 0)"
  proof (intro sum.cong refl)
    fix w assume "w ∈ A"
    have "zorder (λz. (ϕ z - ϕ w) powi h w) z = h w * zorder (λz. (ϕ
z - ϕ w)) z"
  proof (cases "h w = 0")
    case [simp]: False
    have "∀F z in at z. ϕ z - ϕ w ≠ 0"
      using eventually_nonzero_at[of z]
      by eventually_elim (use <w ∈ A> finite in <auto simp: g_def>)

    hence "∃F z in at z. ϕ z - ϕ w ≠ 0"
      using eventually_frequently_at_neq_bot by blast
    thus ?thesis
      by (intro zorder_power_int) (auto intro!: meromorphic_intros)
  qed auto
  also have "zorder (λz. ϕ z - ϕ w) z =
      (if rel z w ∨ rel z (-w) then if 2*w ∈ Λ then 2 else

```

```

1 else 0)"
  using zorder_weierstrass_fun_minus_const[of w z] z <w ∈ A> by simp
  also have "h w * ... = (if rel z w ∨ rel z (-w) then if 2*w ∈ Λ then
2 * h w else h w else 0)"
  by auto
  finally show "zorder (λz. (ϕ z - ϕ w) powi h w) z =
  (if rel z w ∨ rel z (-w) then if 2*w ∈ Λ then 2 *
h w else h w else 0)" .
  qed
  finally show "zorder g z = (∑ w∈A. if rel z w ∨ rel z (-w) then if
2*w ∈ Λ then 2 * h w else h w else 0)" .
  qed

end

lemma (in even_elliptic_function) in_terms_of_weierstrass_fun_even_aux:
  assumes nontrivial: "¬eventually (λz. f z = 0) (cosparse UNIV)"
  defines "Z ≡ {z∈half_fund_parallelogram - {0}. is_pole f z ∨ isolated_zero
f z}"
  defines "h ≡ (λz. zorder f z div (if 2 * z ∈ Λ then 2 else 1))"
  obtains c where "eventually (λz. f z = c * (∏ w∈Z. (ϕ z - ϕ w) powi
h w)) (cosparse UNIV)"
  proof -
  define h' where "h' = (λz. if z ∈ Z then h z else 0)"
  define g where "g = (λz. (∏ w∈Z. (ϕ z - ϕ w) powi h w))"
  have [intro]: "finite Z"
  proof (rule finite_subset)
  show "Z ⊆ {z∈period_parallelogram 0. is_pole f z} ∪ {z∈period_parallelogram
0. isolated_zero f z}"
  using half_fund_parallelogram_subset_period_parallelogram by (auto
simp: Z_def)
  show "finite ({z∈period_parallelogram 0. is_pole f z} ∪ {z∈period_parallelogram
0. isolated_zero f z})"
  by (intro finite_UnI finite_poles_in_parallelogram finite_zeros_in_parallelogram)
  qed
  have [simp]: "z ∉ Λ" if "z ∈ Z" for z
  using that half_fund_parallelogram_in_lattice_iff[of z] unfolding
Z_def by auto

  interpret g: elliptic_function_construct ω1 ω2 Z h g
  by unfold_locales (auto simp: g_def)

  have zorder_eq_aux: "zorder g z = zorder f z" if z: "z ∈ half_fund_parallelogram
- Λ" for z
  proof -
  have "zorder g z = (∑ w∈Z. if rel z w ∨ rel z (-w) then if 2*w ∈
Λ then 2 * h w else h w else 0)"
  by (rule g.zorder_eq) (use z in auto)

```

```

also have "... = ( $\sum_{w \in Z \cap \{z\}}$ . if  $2 * w \in \Lambda$  then  $2 * h w$  else  $h w$ )"
proof (intro sum.mono_neutral_cong_right ballI)
  fix w assume w: "w  $\in Z - Z \cap \{z\}$ "
  thus "(if rel z w  $\vee$  rel z (-w) then if  $2 * w \in \Lambda$  then  $2 * h w$  else
h w else 0) = 0"
    using rel_in_half_fund_parallelogram_imp_eq[of z w] z by (auto
simp: Z_def)
  qed auto
also have "... = (if  $2 * z \in \Lambda$  then  $2 * h' z$  else  $h' z$ )"
  by (auto simp: h_def h'_def)
also have "... = (if z  $\in Z$  then zorder f z else 0)"
  using even_zorder[of z] nontrivial by (auto simp: h_def h'_def)
also have "... = zorder f z"
proof (cases "z  $\in Z$ ")
  case False
  hence " $\neg$ is_pole f z  $\wedge$   $\neg$ isolated_zero f z"
  using z by (auto simp: Z_def)
  moreover have "frequently ( $\lambda z. f z \neq 0$ ) (at z)"
  using nontrivial by (metis eventually_eq_imp_almost_everywhere_eq
not_eventually)
  ultimately have "zorder f z = 0"
  by (intro not_pole_not_isolated_zero_imp_zorder_eq_0) (auto intro:
meromorphic')
  thus ?thesis
  by simp
  qed auto
  finally show ?thesis .
qed

have zorder_eq: "zorder g z = zorder f z" if z: "z  $\notin \Lambda$ " for z
proof -
  have "zorder g z = zorder g (to_half_fund_parallelogram z)"
  using g.zorder_to_half_fund_parallelogram by simp
  also have "... = zorder f (to_half_fund_parallelogram z)"
  by (rule zorder_eq_aux) (use z in auto)
  also have "... = zorder f z"
  using zorder_to_half_fund_parallelogram by simp
  finally show ?thesis .
qed

define h where "h = ( $\lambda z. f z / g z$ )"
interpret h: elliptic_function  $\omega_1 \omega_2$  h
  unfolding h_def by (intro elliptic_function_intros)

have h_nonzero: "eventually ( $\lambda z. h z \neq 0$ ) (at z)" for z
proof -
  have "eventually ( $\lambda z. f z \neq 0$ ) (at z)"
  using nontrivial frequently_def frequently_eq_imp_almost_everywhere_eq
by blast

```

```

    thus ?thesis
      using g.eventually_nonzero_at by eventually_elim (auto simp: h_def)
qed

have zorder_h: "zorder h z = 0" if z: "z ∉ Λ" for z
unfolding h_def
proof (subst zorder_divide)
  show "∃F z in at z. f z ≠ 0"
    using nontrivial by (metis eventually_eq_imp_almost_everywhere_eq
not_eventually)
  show "∃F z in at z. g z ≠ 0"
    using eventually_frequently g.eventually_nonzero_at trivial_limit_at
by blast
qed (use z zorder_eq[of z] in <auto intro!: meromorphic' g.meromorphic'>)

have zorder_h': "zorder h z = 0" if z: "z ∈ period_parallelogram 0"
"z ≠ 0" for z
  by (rule zorder_h) (use z fund_period_parallelogram_in_lattice_iff[of
z] in auto)

have "elliptic_order h = 0"
proof -
  have "elliptic_order h = (∑ z ∈ (if is_pole h 0 then {0} else {})).
nat (-zorder h z))"
  unfolding elliptic_order_def
  by (intro sum.mono_neutral_right h.finite_poles_in_parallelogram)
(auto dest: zorder_h')
  also have "... = nat (-zorder h 0)"
  using zorder_neg_imp_is_pole[OF h.meromorphic', of 0] h_nonzero
linorder_not_less[of "zorder h 0" 0] by auto
  finally have *: "elliptic_order h = nat (-zorder h 0)" .

  have "elliptic_order h = (∑ z ∈ (if isolated_zero h 0 then {0} else
{}). nat (zorder h z))"
  unfolding h.zeros_eq_elliptic_order[of 0, symmetric]
  by (intro sum.mono_neutral_right h.finite_zeros_in_parallelogram)
(auto dest: zorder_h')
  also have "... = nat (zorder h 0)"
  using zorder_pos_imp_isolated_zero[OF h.meromorphic', of 0] h_nonzero
linorder_not_less[of 0 "zorder h 0"] by auto
  finally have "elliptic_order h = nat (zorder h 0)" .
  with * show "elliptic_order h = 0"
  by simp
qed

then obtain c where c: "eventually (λz. h z = c) (cosparse UNIV)"
  using h.elliptic_order_eq_0_iff_const_cosparse by blast
moreover have "eventually (λz. g z ≠ 0) (cosparse UNIV)"
  using g.nonzero_almost_everywhere by blast
ultimately have "eventually (λz. f z = c * g z) (cosparse UNIV)"

```

```

    by eventually_elim (auto simp: h_def)
  thus ?thesis
    using that[of c] unfolding g_def by blast
qed

```

Finally, we show that any even elliptic function can be written as a rational function of \wp . This is Exercise 1.4 in Apostol's book.

```

lemma (in even_elliptic_function) in_terms_of_weierstrass_fun_even:
  obtains p q :: "complex poly" where "q ≠ 0" "∀ z. f z = poly p (ϕ z) / poly q (ϕ z)"
proof (cases "eventually (λz. f z = 0) (cospars UNIV)")
  case True
  thus ?thesis
    using that[of 1 0] by simp
next
  case False
  define Z where "Z = {z ∈ half_fund_parallelogram - {0}. is_pole f z ∨ isolated_zero f z}"
  define h where "h = (λz. zorder f z div (if 2 * z ∈ Λ then 2 else 1))"
  obtain c where *: "eventually (λz. f z = c * (∏ w ∈ Z. (ϕ z - ϕ w) powi h w)) (cospars UNIV)"
  using False in_terms_of_weierstrass_fun_even_aux unfolding Z_def h_def
  by metis
  define p where "p = Polynomial.smult c (∏ w ∈ {w ∈ Z. h w ≥ 0}. [:-ϕ w, 1:] ^ nat (h w))"
  define q where "q = (∏ w ∈ {w ∈ Z. h w < 0}. [:-ϕ w, 1:] ^ nat (-h w))"

  have finite: "finite Z"
  proof (rule finite_subset)
    show "Z ⊆ {z ∈ period_parallelogram 0. is_pole f z} ∪ {z ∈ period_parallelogram 0. isolated_zero f z}"
    using half_fund_parallelogram_subset_period_parallelogram by (auto simp: Z_def)
    show "finite ({z ∈ period_parallelogram 0. is_pole f z} ∪ {z ∈ period_parallelogram 0. isolated_zero f z})"
    by (intro finite_UnI finite_poles_in_parallelogram finite_zeros_in_parallelogram)
  qed

  show ?thesis
  proof (rule that[of q p])
    show "q ≠ 0"
    using finite by (auto simp: q_def)
  next
    show "∀ z. f z = poly p (ϕ z) / poly q (ϕ z)"
    using *
  proof eventually_elim
    case (elim z)
    have "f z = c * (∏ w ∈ Z. (ϕ z - ϕ w) powi h w)"
    by (fact elim)
  qed

```

```

also have "Z = {w∈Z. h w ≥ 0} ∪ {w∈Z. h w < 0}"
  by auto
also have "c * (∏ w∈... (φ z - φ w) powi h w) =
  c * (∏ w∈{w∈Z. h w ≥ 0}. (φ z - φ w) powi h w) *
  (∏ w∈{w∈Z. h w < 0}. (φ z - φ w) powi h w)"
  by (subst prod.union_disjoint) (use finite in auto)
also have "(∏ w∈{w∈Z. h w ≥ 0}. (φ z - φ w) powi h w) =
  (∏ w∈{w∈Z. h w ≥ 0}. poly [:-φ w, 1:] (φ z) ^ (nat
(h w)))"
  by (intro prod.cong) (auto simp: power_int_def)
also have "c * ... = poly p (φ z)"
  by (simp add: p_def poly_prod)
also have "(∏ w∈{w∈Z. h w < 0}. (φ z - φ w) powi h w) =
  (∏ w∈{w∈Z. h w < 0}. inverse (poly [:-φ w, 1:] (φ z)
^ (nat (-h w))))"
  by (intro prod.cong) (auto simp: power_int_def field_simps)
also have "... = inverse (poly q (φ z))"
  unfolding q_def poly_prod by (subst prod_inverseq [symmetric])
auto
  finally show ?case
    by (simp add: field_simps)
qed
qed
qed

```

From this, we now show that any elliptic function f can be written in the form $f(z) = g(\varphi(z)) + \varphi'(z)h(\varphi(z))$ where g, h are rational functions.

The proof is fairly simple: We can split $f(z)$ into a sum $f(z) = f_1(z) + f_2(z)$ where f_1 is even and f_2 is odd by defining $f_1(z) = \frac{1}{2}(f(z) + f(-z))$ and $f_2(z) = \frac{1}{2}(f(z) - f(-z))$. We can then further define $f_3(z) = f_2(z)/\varphi'(z)$ so that f_3 is also even.

By our previous result, we know that f_1 and f_3 can be written as rational functions of φ , so by combining everything we get the result we want.

This result is Exercise 1.5 in Apostol's book.

theorem (in *elliptic_function*) *in_terms_of_weierstrass_fun*:

```

  obtains p q r s :: "complex poly" where "q ≠ 0" "s ≠ 0"
    "∀ z. f z = poly p (φ z) / poly q (φ z) + φ' z * poly r (φ z) /
poly s (φ z)"

```

proof -

```

  define f1 where "f1 = (λz. (f z + f (-z)) / 2)"
  define f2 where "f2 = (λz. (f z - f (-z)) / 2)"
  define f2' where "f2' = (λz. f2 z / φ' z)"

```

```

  note [elliptic_function_intros] = elliptic_function_compose_uminus[OF
elliptic_function_axioms]

```

```

  interpret f1: elliptic_function ω1 ω2 f1
    unfolding f1_def by (intro elliptic_function_intros)

```

```

interpret f1: even_elliptic_function ω1 ω2 f1
  by standard (auto simp: f1_def)
obtain p q where pq: "q ≠ 0" "∀z. f1 z = poly p (φ z) / poly q (φ
z)"
  using f1.in_terms_of_weierstrass_fun_even .

interpret f2': elliptic_function ω1 ω2 f2'
  unfolding f2'_def f2_def by (intro elliptic_function_intros)
interpret f2': even_elliptic_function ω1 ω2 f2'
  by standard (auto simp: f2'_def f2_def divide_simps)
obtain r s where rs: "s ≠ 0" "∀z. f2' z = poly r (φ z) / poly s
(φ z)"
  using f2'.in_terms_of_weierstrass_fun_even .

have "eventually (λz. φ' z ≠ 0) (cosparse UNIV)"
  by (simp add: weierstrass_fun_deriv.avoid)
with pq(2) and rs(2) have "∀z. f z = poly p (φ z) / poly q (φ z)
+ φ' z * poly r (φ z) / poly s (φ z)"
proof eventually_elim
  case (elim z)
  have "poly p (φ z) / poly q (φ z) + φ' z * poly r (φ z) / poly s
(φ z) =
    f1 z + φ' z * (poly r (φ z) / poly s (φ z))"
    unfolding elim(1) by (simp add: divide_simps)
  also have "poly r (φ z) / poly s (φ z) = f2' z"
    using elim(2) by simp
  also have "φ' z * f2' z = f2 z"
    using elim(3) by (simp add: f2'_def)
  also have "f1 z + f2 z = f z"
    by (simp add: f1_def f2_def field_simps)
  finally show ?case ..
qed
thus ?thesis
  using that[of q s p r] pq(1) rs(1) by simp
qed

end

```

8 Related facts about Jacobi theta functions

```
theory Theta_Inversion
```

```
imports
```

```
  "Theta_Functions.Jacobi_Triple_Product"
```

```
  "Theta_Functions.Theta_Nullwert"
```

```
  Complex_Lattices
```

```
begin
```

```
lemmas [simp del] = div_mult_self1 div_mult_self2 div_mult_self3 div_mult_self4
```

In this section we will re-use some of the lemmas we proved to study elliptic functions in order to show two non-trivial facts about the Jacobi theta functions. The first one is a uniqueness result:

We know that $\vartheta_{00}(z, t)$, viewed as a function of z for fixed t , is entire, periodic with period 1, and quasi-periodic with period t and factor e^{-2z-t} . We will show that for any fixed t in the upper half plane, $\vartheta_{00}(\cdot, t)$ is actually uniquely defined by these relations, up to a constant factor.

8.1 Uniqueness of quasi-periodic entire functions

We first show a fairly obvious fact: in any complete real normed field, the separable ordinary differential equation $f'(x) = g(x)f(x)$ has at most one solution up to a constant factor in any complex domain.

If a non-vanishing function G satisfies $G'(x) = g(x)G(x)$, then G is that solution. This allows us to “certify” a solution easily.

```

lemma separable_ODE_simple_unique:
  fixes f :: "'a :: {banach, real_normed_field} ⇒ 'a"
  assumes eq: "∧x. x ∈ A ⇒ f' x = g x * f x"
  assumes deriv_f: "∧x. x ∈ A ⇒ (f has_field_derivative f' x) (at
x within A)"
  assumes deriv_g: "∧x. x ∈ A ⇒ (G has_field_derivative (g x * G x))
(at x within A)"
  assumes nonzero [simp]: "∧x. x ∈ A ⇒ G x ≠ 0"
  assumes "convex A"
  shows "∃c. ∀x∈A. f x = c * G x"
proof -
  have "∃c. ∀x∈A. f x / G x = c"
  proof (rule has_field_derivative_zero_constant)
    show "((λx. f x / G x) has_field_derivative 0) (at x within A)" if
x: "x ∈ A" for x
      using x by (auto intro!: derivative_eq_intros deriv_f deriv_g simp:
eq)
  qed fact
  then obtain c where c: "∧x. x ∈ A ⇒ f x / G x = c"
    by blast
  thus ?thesis
    by (auto simp: field_simps)
qed

```

The following locale captures the notion of an entire function in the complex plane that satisfies the same (quasi-)periodicity as the Jacobi theta function ϑ_{00} , namely $f(z+1) = f(z)$ and $f(z+t) = e^{-2z-t}f(z)$ for some fixed t with $\text{Im}(t) > 0$.

We will show that any such function is equal to ϑ_{00} up to a constant factor.

```

locale thetalike_function =
  fixes f :: "complex ⇒ complex" and t :: complex

```

```

    assumes entire: "f holomorphic_on UNIV"
    assumes Im_t: "Im t > 0"
    assumes f_plus_1: "f (z + 1) = f z"
    assumes f_plus_quasiperiod: "f (z + t) = f z / to_nome (2*z+t)"
begin

```

```

lemma holomorphic:
  assumes "g holomorphic_on A"
  shows "(λx. f (g x)) holomorphic_on A"
proof -
  have "(f ∘ g) holomorphic_on A"
    using assms entire by (rule holomorphic_on_compose_gen) auto
  thus ?thesis
    by (simp add: o_def)
qed

```

```

lemma analytic:
  assumes "g analytic_on A"
  shows "(λx. f (g x)) analytic_on A"
proof -
  have *: "f analytic_on UNIV"
    by (subst analytic_on_open) (auto intro!: holomorphic)
  have "(f ∘ g) analytic_on A"
    using assms * by (rule analytic_on_compose_gen) auto
  thus ?thesis
    by (simp add: o_def)
qed

```

We first show some straightforward facts about the behaviour of f on the lattice generated by 1 and t .

```

sublocale lattice: std_complex_lattice t
  by standard (use Im_t in <auto elim!: Reals_cases>)

```

```

lemma f_plus_of_nat: "f (z + of_nat n) = f z"
proof (induction n)
  case (Suc n)
  thus ?case
    using f_plus_1[of "z + of_nat n"] by (simp add: algebra_simps)
qed auto

```

```

lemma f_plus_of_int: "f (z + of_int n) = f z"
  using f_plus_of_nat[of z "nat n"] f_plus_of_nat[of "z + of_int n" "nat (-n)"]
  by (cases "n ≥ 0") (auto simp: algebra_simps)

```

```

lemma f_plus_of_nat_quasiperiod:
  "f (z + of_nat n * t) = f z / to_nome (2 * of_nat n * z + of_nat (n2) * t)"
proof (induction n)

```

```

    case (Suc n)
  thus ?case
    using f_plus_quasiperiod[of "z + of_nat n * t"]
    by (simp add: algebra_simps power2_eq_square flip: to_nome_add)
qed auto

lemma f_plus_of_int_quasiperiod:
  "f (z + of_int n * t) = f z / to_nome (2 * of_int n * z + of_int (n2)
  * t)"
proof (cases "n ≥ 0")
  case True
  thus ?thesis
    using f_plus_of_nat_quasiperiod[of z "nat n"] by (auto simp: sgn_if)
next
  case False
  thus ?thesis
    using f_plus_of_nat_quasiperiod[of "z + of_int n * t" "nat (-n)"]
    by (simp add: field_simps to_nome_add to_nome_diff power2_eq_square
    to_nome_minus)
qed

lemma relE:
  assumes "lattice.rel z z'"
  obtains m n :: int where "z = z' + of_int m + of_int n * t"
  using assms unfolding lattice.rel_def
  by (elim lattice.latticeE) (auto simp: lattice.of_ω12_coords_def algebra_simps)

lemma f_zero_cong_lattice:
  assumes "lattice.rel z z'"
  shows "f z = 0 ↔ f z' = 0"
proof -
  interpret lattice: complex_lattice_periodic 1 t "λz. f z = 0"
  by standard (auto simp: f_plus_1 f_plus_quasiperiod)
  show ?thesis
    using lattice.lattice_cong[OF assms] .
qed

lemma zorder_f_cong_lattice:
  assumes "lattice.rel z z'"
  shows "zorder f z = zorder f z'"
proof -
  from assms obtain m n where mn: "z = z' + of_int m + of_int n * t"
  by (elim relE)
  define h where "h = (λz. z + of_int m + of_int n * t)"
  define g where "g = (λz. to_nome (-(2 * of_int n * (z + of_int m) +
  (of_int n)2 * t)))"
  have "zorder f (h z') = zorder (f ∘ h) z'"
  by (simp add: zorder_shift' h_def algebra_simps)
  also have "... = zorder f z'"

```

```

proof (cases "∀z. f z = 0")
  case True
  hence [simp]: "f = (λ_. 0)"
    by auto
  show ?thesis
    by simp
next
  case False
  have ev_nz: "eventually (λz. f z ≠ 0) (cosparse UNIV)"
  proof -
    have "(∀x∈UNIV. f x = 0) ∨ (∀≈x∈UNIV. f x ≠ 0)"
      by (intro nicely_meromorphic_imp_constant_or_avoid
        analytic_on_imp_nicely_meromorphic_on analytic) auto
    moreover have "¬(∀x∈UNIV. f x = 0)"
      using False by auto
    ultimately show ?thesis
      by blast
  qed
  hence ev: "eventually (λw. f w ≠ 0) (at z')"
    by (rule eventually_cosparse_imp_eventually_at) auto
  have "f ∘ h = (λz. f z * g z)"
    unfolding to_nome_minus g_def
    by (auto simp: h_def fun_eq_iff f_plus_of_int_quasiperiod f_plus_of_int
      divide_inverse)
  hence "zorder (f ∘ h) z' = zorder (λz. f z * g z) z'"
    by (rule arg_cong)
  also have "zorder (λz. f z * g z) z' = zorder f z' + zorder g z'"
    by (rule zorder_times_analytic)
    (use ev in <auto intro!: analytic_intros analytic simp: g_def>)
  also have "zorder g z' = 0"
    by (rule zorder_eq_0I) (auto simp: g_def intro!: analytic_intros)
  finally show ?thesis
    by (simp add: o_def)
  qed
  finally show ?thesis
    by (simp add: h_def mn o_def)
qed

lemma deriv_f_plus_1: "deriv f (z + 1) = deriv f z"
proof -
  have "deriv f (z + 1) = deriv (λz. f (z + 1)) z"
    by (simp add: deriv_shift_0' o_def add_ac)
  also have "(λz. f (z + 1)) = f"
    by (subst f_plus_1) auto
  finally show ?thesis .
qed

lemma deriv_f_plus_quasiperiod:
  "deriv f (z + t) = (deriv f z - 2 * pi * i * f z) / to_nome (2 * z +

```

```

t)"
proof -
  have "deriv f (z + t) = deriv (λz. f (z + t)) z"
    by (simp add: deriv_shift_0' o_def add_ac)
  also have "(λz. f (z + t)) = (λz. f z * to_nome (-2 * z - t))"
    by (subst f_plus_quasiperiod) (auto simp: to_nome_diff to_nome_minus
field_simps to_nome_add)
  also have "deriv ... z = f z * deriv (λz. to_nome (- (2 * z) - t)) z
+ deriv f z * to_nome (- (2 * z) - t)"
    by (subst complex_derivative_mult_at) (auto intro!: analytic_intros
analytic)
  also have "deriv (λz. to_nome (- (2 * z) - t)) z = -2 * pi * i * to_nome
(-(2*z)-t)"
    by (rule DERIV_imp_deriv) (auto intro!: derivative_eq_intros)
  finally show "deriv f (z + t) = (deriv f z - 2 * pi * i * f z) / to_nome
(2 * z + t)"
    by (auto simp: field_simps to_nome_minus to_nome_diff to_nome_add)
qed

```

Next, we will simplify the integral

$$\int_{\gamma} \frac{h(z)f'(z)}{f(z)} dz$$

for an arbitrary function $h(z)$ using the shift relations for f . Here, γ is a counter-clockwise contour along the border of a period parallelogram with lower left corner b and no zeros of f on it.

We find that:

$$\int_{\gamma} \frac{h(z)f'(z)}{f(z)} dz = \int_b^{b+1} (h(z)-h(z+t)) \frac{f'(z)}{f(z)} + 2\pi i h(z+t) dz - \int_b^{b+t} (h(z)-h(z+1)) \frac{f'(z)}{f(z)} dz$$

lemma *argument_principle_f_gen*:

```

fixes orig :: complex
defines "γ ≡ parallelogram_path orig 1 t"
assumes h: "h holomorphic_on UNIV"
assumes nz: "∧z. z ∈ path_image γ ⇒ f z ≠ 0"
shows "contour_integral γ (λx. h x * deriv f x / f x) =
      contour_integral (linepath orig (orig + 1))
      (λz. (h z - h (z + t)) * deriv f z / f z + 2 * pi * i * h
(z + t)) -
      contour_integral (linepath orig (orig + t))
      (λz. (h z - h (z + 1)) * deriv f z / f z)"

```

proof -

```

define h' where "h' = (λx. h (x + orig) * deriv f (x + orig) / f (x
+ orig))"
have [analytic_intros]: "h analytic_on A" for A
  using h analytic_on_holomorphic by blast

```

```

have "contour_integral  $\gamma$  ( $\lambda x. h x * deriv f x / f x$ ) =
      contour_integral (linepath 0 1) ( $\lambda x. h' x - h' (x + t)$ ) -
      contour_integral (linepath 0 t) ( $\lambda x. h' x - h' (x + 1)$ )" (is
"_ = ?rhs")
  unfolding  $\gamma\_def$ 
  proof (subst contour_integral_parallelogram_path'; (fold  $\gamma\_def$ )?)
    show "continuous_on (path_image  $\gamma$ ) ( $\lambda x. h x * deriv f x / f x$ )" using nz
      by (intro continuous_intros holomorphic_on_imp_continuous_on analytic_imp_holomorphic
(auto intro!: analytic_intros analytic))
  next
    have 1: "linepath orig (orig + 1) = (+) orig  $\circ$  linepath 0 1"
      and 2: "linepath orig (orig + t) = (+) orig  $\circ$  linepath 0 t"
      by (auto simp: linepath_translate add_ac)
    show "contour_integral (linepath orig (orig + 1))
          ( $\lambda x. h x * deriv f x / f x - h (x + t) * deriv f (x + t) /$ 
 $f (x + t)$ ) -
          contour_integral (linepath orig (orig + t))
          ( $\lambda x. h x * deriv f x / f x - h (x + 1) * deriv f (x + 1) /$ 
 $f (x + 1)$ ) = ?rhs"
      unfolding 1 2 contour_integral_translate h'_def by (simp add: algebra_simps)
    qed

  also have " $(\lambda z. h' z - h' (z + 1)) =$ 
            ( $\lambda z. (h (z + orig) - h (z + orig + 1)) * deriv f (z + orig)$ 
 $/ f (z + orig)$ )"
            (is "?lhs = ?rhs")
    proof
      fix z :: complex
      have "h' z - h' (z + 1) =
            h (z + orig) * deriv f (z + orig) / f (z + orig) -
            h (z + orig + 1) * deriv f (z + orig + 1) / f (z + orig +
1)"
        by (simp add: h'_def add_ac)
      also have "... = (h (z + orig) - h (z + orig + 1)) * deriv f (z +
orig) / f (z + orig)"
        unfolding f_plus_1 deriv_f_plus_1 by (simp add: diff_divide_distrib
ring_distrib)
      finally show "?lhs z = ?rhs z" .
    qed
    also have "contour_integral (linepath 0 t) ... =
            contour_integral ((+) orig  $\circ$  linepath 0 t) ( $\lambda z. (h z - h$ 
 $(z + 1)) * deriv f z / f z$ )"
      by (rule contour_integral_translate [symmetric])
    also have "(+) orig  $\circ$  linepath 0 t = linepath orig (orig + t)"
      by (subst linepath_translate) (simp_all add: add_ac)

  also have "contour_integral (linepath 0 1) ( $\lambda z. h' z - h' (z + t)$ ) =
            contour_integral (linepath 0 1)

```

```

      (λz. (h (z + orig) - h (z + orig + t)) * deriv f (z + orig)
/ f (z + orig) +
      2 * pi * i * h (z + orig + t))"
  (is "contour_integral _ ?lhs = contour_integral _ ?rhs")
proof (rule contour_integral_cong)
  fix z :: complex
  assume z: "z ∈ path_image (linepath 0 1)"
  hence "orig + z ∈ path_image ((+) orig ∘ linepath 0 1)"
    by (subst path_image_compose) auto
  also have "(+) orig ∘ linepath 0 1 = linepath orig (orig + 1)"
    by (subst linepath_translate) (auto simp: add_ac)
  finally have "orig + z ∈ path_image γ"
    unfolding γ_def parallelogram_path_def by (auto simp: path_image_join)
  hence nz': "f (orig + z) ≠ 0"
    using nz by blast

  have "h' z - h' (z + t) =
      h (z + orig) * deriv f (z + orig) / f (z + orig) -
      h (z + orig + t) * (deriv f (z + orig + t) / f (z + orig +
t))"
    by (simp add: h'_def add_ac)
  also have "deriv f (z + orig + t) / f (z + orig + t) =
      deriv f (z + orig) / f (z + orig) - 2 * pi * i"
    unfolding f_plus_quasiperiod deriv_f_plus_quasiperiod using nz'
    by (auto simp: divide_simps add_ac)
  also have "h (z + orig) * deriv f (z + orig) / f (z + orig) - h (z
+ orig + t) * ... = ?rhs z"
    by (simp add: ring_distrib diff_divide_distrib)
  finally show "?lhs z = ?rhs z" .
qed auto
also have "... = contour_integral ((+) orig ∘ linepath 0 1)
      (λz. (h z - h (z + t)) * deriv f z / f z + 2 * pi
* i * h (z + t))"
  unfolding contour_integral_translate by (simp add: add_ac)
  also have "(+) orig ∘ linepath 0 1 = linepath orig (orig + 1)"
    by (subst linepath_translate) (simp_all add: add_ac)

  finally show ?thesis .
qed

```

We now instantiate the above fact with $h(z) = 1$ and see that the corresponding integral divided by $2\pi i$ evaluates to 1. This will later tell us that every period parallelogram contains exactly one root of f , and that it is a simple root.

```

lemma argument_principle_f_1:
  fixes orig :: complex
  defines "γ ≡ parallelogram_path orig 1 t"
  assumes nz: "∧z. z ∈ path_image γ ⇒ f z ≠ 0"
  shows "contour_integral γ (λz. deriv f z / f z) = 2 * pi * i"

```

using argument_principle_f_gen[OF _ nz, of " $\lambda_. 1$ "] by (simp add: γ_def)

Next, we instantiate the lemma with $h(z) = z$ and see that the integral divided by $2\pi i$ evaluates to some value of the form $\frac{t+1}{2} + m + nt$ for integers m, n . In other words: it evaluates to some value equivalent to $\frac{t+1}{2}$ modulo our lattice.

This will later tell us that the roots of f in any period parallelogram sum to something equivalent to $\frac{t+1}{2}$. Since we know there is only one root and it is simple, this means that the only root in each period parallelogram is the copy of $\frac{t+1}{2}$ contained in it.

lemma argument_principle_f_z:

```

  fixes orig :: complex
  defines " $\gamma \equiv \text{parallelogram\_path orig 1 t}$ "
  assumes nz: " $\bigwedge z. z \in \text{path\_image } \gamma \implies f z \neq 0$ "
  shows "lattice.rel (contour_integral  $\gamma (\lambda z. z * \text{deriv } f z / f z) / (2*\pi*i)) ((t+1)/2)$ "
proof -
  note [holomorphic_intros del] = holomorphic_deriv
  note [holomorphic_intros] = holomorphic holomorphic_on_subset[OF holomorphic_deriv[of
- UNIV]]
  define  $\gamma_1$  where " $\gamma_1 = \text{linepath orig (orig + 1)}$ "
  define  $\gamma_2$  where " $\gamma_2 = \text{linepath orig (orig + t)}$ "
  define  $\gamma_2'$  where " $\gamma_2' = f \circ \gamma_2$ "
  define  $\gamma_3$  where " $\gamma_3 = (\lambda z. f \text{ orig } * \text{to\_nome } z) \circ \text{linepath } 0 (-2 * \text{orig} - t)$ "

  have "pathstart  $\gamma \in \text{path\_image } \gamma$ "
    by blast
  from nz[OF this] have [simp]: " $f \text{ orig} \neq 0$ "
    by (simp add:  $\gamma\_def$ )

  have [simp, intro]: " $\text{valid\_path } \gamma_1$ " " $\text{valid\_path } \gamma_2$ "
    by (simp_all add:  $\gamma_1\_def \gamma_2\_def$ )
  have [simp, intro]: " $\text{valid\_path } \gamma_2'$ "
    unfolding  $\gamma_2'\_def$  by (intro valid_path_compose_analytic[of _ _ UNIV]
analytic) auto
  have [simp, intro]: " $\text{valid\_path } \gamma_3$ "
    unfolding  $\gamma_3\_def$  by (intro valid_path_compose_analytic[of _ _ UNIV]
analytic_intros) auto

  have [simp]: " $\text{pathstart } \gamma_2' = f \text{ orig}$ " " $\text{pathfinish } \gamma_2' = f (\text{orig} + t)$ "
    by (simp_all add:  $\gamma_2'\_def \gamma_2\_def \text{pathstart\_compose pathfinish\_compose}$ )
  have [simp]: " $\text{pathstart } \gamma_3 = f \text{ orig}$ " " $\text{pathfinish } \gamma_3 = f (\text{orig} + t)$ "
    by (simp_all add:  $\gamma_3\_def \text{pathstart\_compose pathfinish\_compose } f\_plus\_quasiperiod$ 
to_nome_diff to_nome_minus to_nome_add field_simps)

  have nz': " $f z \neq 0$ " if " $z \in \text{path\_image } \gamma_1 \cup \text{path\_image } \gamma_2$ " for z
proof -

```

```

note <z ∈ path_image γ1 ∪ path_image γ2>
also have "path_image γ1 ∪ path_image γ2 ⊆ path_image γ"
  by (auto simp: γ1_def γ2_def γ_def path_image_compose path_image_join

        parallelogram_path_def image_Un closed_segment_commute
        simp flip: closed_segment_translation)
finally show ?thesis
  using nz[of z] by blast
qed

have [simp]: "0 ∉ path_image γ3"
  by (auto simp: γ3_def path_image_compose)
have [simp]: "0 ∉ path_image γ2'"
  using nz' unfolding γ2'_def by (auto simp: γ2'_def path_image_compose)

```

The actual proof starts here.

```

define I1 where "I1 = contour_integral γ1"
define I2 where "I2 = contour_integral γ2"

have "winding_number (f ∘ γ1) 0 ∈ ℤ"
proof (rule integer_winding_number)
  have "valid_path (f ∘ γ1)"
    by (intro valid_path_compose_analytic[of _ _ UNIV] analytic) auto
  thus "path (f ∘ γ1)"
    by (rule valid_path_imp_path)
qed (auto simp: path_image_compose nz' pathfinish_def pathstart_def
γ1_def linepath_def f_plus_1)
then obtain m where m: "winding_number (f ∘ γ1) 0 = of_int m"
  by (elim Ints_cases)

have "winding_number γ2' 0 + orig + t / 2 ∈ ℤ"
proof -
  define r where "r = unwinding (i * pi * (- (2 * orig) - t))"
  have "winding_number (γ2' +++ reversepath γ3) 0 ∈ ℤ"
  proof (rule integer_winding_number)
    have "valid_path (γ2' +++ reversepath γ3)"
      by (intro valid_path_join valid_path_compose_analytic[of _ _ UNIV]
analytic)
        (auto simp: pathfinish_compose f_plus_quasiperiod)
    thus "path (γ2' +++ reversepath γ3)"
      by (rule valid_path_imp_path)
  next
    show "pathfinish (γ2' +++ reversepath γ3) =
      pathstart (γ2' +++ reversepath γ3)"
      by (auto simp: pathfinish_compose pathstart_compose γ2_def)
  next
    have "0 ∉ path_image γ2' ∪ path_image γ3"
      by auto
    also have "path_image γ2' ∪ path_image γ3 = path_image (γ2' +++

```

```

reversepath  $\gamma_3$ )"
  by (subst path_image_join)
    (simp_all add: f_plus_quasiperiod)
  finally show "0  $\notin$  path_image ( $\gamma_2'$  +++ reversepath  $\gamma_3$ )" .
qed

also have "winding_number ( $\gamma_2'$  +++ reversepath  $\gamma_3$ ) 0 =
winding_number  $\gamma_2'$  0 + winding_number (reversepath  $\gamma_3$ )
0"
  by (rule winding_number_join) (auto simp: valid_path_imp_path)
also have "winding_number (reversepath  $\gamma_3$ ) 0 = -winding_number  $\gamma_3$ 
0"
  by (subst winding_number_reversepath) (auto simp: valid_path_imp_path)
also have "winding_number  $\gamma_3$  0 = contour_integral  $\gamma_3$  ( $\lambda w. 1 / w$ )
/ (2 * pi * i)"
  by (subst winding_number_valid_path) auto
also have "contour_integral  $\gamma_3$  ( $\lambda w. 1 / w$ ) =
contour_integral (linepath 0 (-2 * orig - t))
( $\lambda w. \text{deriv } (\lambda z. f \text{ orig } * \text{to\_nome } z) w / (f \text{ orig } * \text{to\_nome } w)$ )" unfolding  $\gamma_3\_def$ 
  by (subst contour_integral_comp_analyticW[of _ UNIV]) (auto intro!:
analytic_intros)
also have "( $\lambda w. \text{deriv } (\lambda z. f \text{ orig } * \text{to\_nome } z) w / (f \text{ orig } * \text{to\_nome } w)$ ) = ( $\lambda w. \text{pi } * i$ )"
  proof
    fix w :: complex
    have "deriv ( $\lambda z. f \text{ orig } * \text{to\_nome } z$ ) w = pi * i * to_nome w * f
orig"
      by (rule DERIV_imp_deriv) (auto intro!: derivative_eq_intros)
    thus "deriv ( $\lambda z. f \text{ orig } * \text{to\_nome } z$ ) w / (f orig * to_nome w) =
pi * i"
      by simp
  qed
also have "contour_integral (linepath 0 (-2 * orig - t)) ( $\lambda w. \text{pi} * i$ )
/ (2 * pi * i) = -orig - t / 2"
  by simp
finally show ?thesis
  by (simp add: algebra_simps o_def)
qed
then obtain n where n: "winding_number  $\gamma_2'$  0 + orig + t / 2 = of_int
n"
  by (elim Ints_cases)

have "contour_integral  $\gamma$  ( $\lambda z. z * \text{deriv } f z / f z$ ) / (2 * pi * i) =
I1 ( $\lambda z. -t * \text{deriv } f z / f z + 2 * \text{pi } * i * (z + t)$ ) / (2 * pi * i)
-
I2 ( $\lambda z. -(\text{deriv } f z / f z)$ ) / (2 * pi * i)"
using nz unfolding  $\gamma\_def$  I1_def I2_def  $\gamma_1\_def$   $\gamma_2\_def$ 
  by (subst argument_principle_f_gen) (auto simp: diff_divide_distrib)

```

```

also have "I1 (λz. -t * deriv f z / f z + 2 * pi * i * (z + t)) =
          I1 (λz. (-t) * (deriv f z / f z)) + I1 (λz. 2 * pi * i *
(z + t))"
  using nz' unfolding I1_def γ1_def
  by (subst contour_integral_add)
      (auto intro!: contour_integrable_continuous_linepath holomorphic_on_imp_continuous_on
        holomorphic_intros)
also have "... / (2*pi*i) = I1 (λz. (-t) * (deriv f z / f z)) / (2*pi*i)
+
          I1 (λz. 2 * pi * i * (z + t)) / (2*pi*i)"
  by (simp add: add_divide_distrib)
also have "I1 (λz. 2 * pi * i * (z + t)) = 2 * pi * i * (orig + t + 1
/ 2)"
  unfolding I1_def
proof (rule contour_integral_unique)
  define F where "F = (λz. pi * i * z ^ 2 + 2 * pi * i * t * z)"
  have "((λz. 2 * pi * i * (z + t)) has_contour_integral (F (pathfinish
γ1) - F (pathstart γ1))) γ1"
    by (rule contour_integral_primitive[of UNIV])
      (auto simp: γ1_def F_def field_simps intro!: derivative_eq_intros)
  also have "F (pathfinish γ1) - F (pathstart γ1) = 2 * pi * i * (orig
+ t + 1 / 2)"
    by (simp add: F_def γ1_def power2_eq_square field_simps)
  finally show "((λz. 2 * pi * i * (z + t)) has_contour_integral
(2 * pi * i * (orig + t + 1 / 2))) γ1" .

qed
also have "... / (2*pi*i) = (orig + t + 1/2)"
  by (simp add: divide_simps)
also have "I1 (λz. (-t) * (deriv f z / f z)) = (-t) * I1 (λz. deriv
f z / f z)"
  using nz' unfolding I1_def γ1_def
  by (subst contour_integral_lmul)
      (auto intro!: contour_integrable_continuous_linepath holomorphic_on_imp_continuous_on
        holomorphic_intros)

also have "I1 (λz. deriv f z / f z) = contour_integral (f ∘ γ1) (λz.
1 / z)"
  unfolding I1_def γ1_def by (subst contour_integral_comp_analyticW[OF
analytic[of _ UNIV]]) auto
  also have "-t * ... / (2 * pi * i) = -t * winding_number (f ∘ γ1) 0"
    by (subst winding_number_valid_path)
      (auto simp: path_image_compose nz' intro!: valid_path_compose_analytic
analytic)
  also have "winding_number (f ∘ γ1) 0 = of_int m"
    by (rule m)

also have "I2 (λz. -(deriv f z / f z)) = -I2 (λz. deriv f z / f z)"
  unfolding I2_def by (subst contour_integral_neg) auto
also have "I2 (λz. deriv f z / f z) = contour_integral γ2' (λz. 1 /

```

```

z)"
  unfolding I2_def  $\gamma_2'$ _def  $\gamma_2$ _def
  by (subst contour_integral_comp_analyticW[OF analytic[of _ UNIV]])
auto
  also have "(-...) / (2 * pi * i) = -winding_number  $\gamma_2'$  0"
  by (subst winding_number_valid_path) auto
  also have "-t * of_int m + (orig + t + 1 / 2) - - winding_number  $\gamma_2'$ 
0 =
      (t + 1) / 2 + of_int n + of_int (-m) * t"
  unfolding n [symmetric] by (simp add: field_simps)
  finally have "contour_integral  $\gamma$  ( $\lambda z. z * deriv f z / f z$ ) / (2*pi*i)
- (t+1)/2 =
      complex_of_int n + complex_of_int (- m) * t"
  by simp
  also have "... = lattice.of_omega12_coords (of_int n, of_int (-m))"
  by (simp add: lattice.of_omega12_coords_def)
  also have "... ∈ lattice.lattice"
  by (rule lattice.of_omega12_coords_in_lattice) auto
  finally show ?thesis
  unfolding lattice.rel_def by blast
qed

```

We now tie everything together and prove the fact mentioned above: the zeros of f are precisely the shifted copies of $\frac{t+1}{2}$, and they are all simple. Unless of course $f(z)$ is identically zero.

lemma zero_iff:

```

  assumes " $\neg(\forall z. f z = 0)$ "
  shows "f z = 0  $\longleftrightarrow$  lattice.rel z ((t + 1) / 2)"
  and "lattice.rel z ((t + 1) / 2)  $\implies$  zorder f z = 1"

```

proof -

```

  note [holomorphic_intros] = holomorphic
  define avoid where "avoid = {z. f z = 0}"
  define parallelogram where "parallelogram = ( $\lambda z. \text{closure (lattice.period_parallelogram z)}$ )"
  define parallelogram' where "parallelogram' = ( $\lambda z. \text{interior (lattice.period_parallelogram z)}$ )"

```

```

  have ev_nz: "eventually ( $\lambda z. f z \neq 0$ ) (cosparse UNIV)"

```

proof -

```

  have " $(\forall x \in UNIV. f x = 0) \vee (\forall \approx x \in UNIV. f x \neq 0)$ "
  by (intro nicely_meromorphic_imp_constant_or_avoid
      analytic_on_imp_nicely_meromorphic_on analytic) auto
  moreover have " $\neg(\forall x \in UNIV. f x = 0)$ "
  using assms by auto
  ultimately show ?thesis
  by blast

```

qed

```

  have sparse_avoid: " $\neg z \text{ islimpt avoid}$ " for z

```

```

proof -
  from ev_nz have "eventually ( $\lambda z. f z \neq 0$ ) (at z)"
    by (rule eventually_cosparse_imp_eventually_at) auto
  thus ?thesis
    unfolding avoid_def by (auto simp: islimpt_iff_eventually)
qed

have countable: "countable avoid"
  using sparse_avoid no_limpt_imp_countable by blast
obtain orig where avoid: "path_image (parallelogram_path orig 1 t)
 $\cap$  avoid = {}"
  using lattice.shifted_period_parallelogram_avoid[OF countable] by blast

have "compact (parallelogram orig)"
  unfolding parallelogram_def by (rule lattice.compact_closure_period_parallelogram)
then obtain R where R: "parallelogram orig  $\subseteq$  ball 0 R"
  by (meson bounded_subset_ballD compact_imp_bounded)

define  $\gamma$  where " $\gamma = \text{parallelogram\_path orig 1 t}$ "
have  $\gamma_{\text{subset}}$ : "path_image  $\gamma \subseteq$  parallelogram orig"
  and  $\gamma_{\text{disjoint}}$ : "path_image  $\gamma \cap$  interior (lattice.period_parallelogram
orig) = {}"
  using lattice.path_image_parallelogram_subset_closure[of orig]
    lattice.path_image_parallelogram_disjoint_interior[of orig]
  unfolding parallelogram_def  $\gamma_{\text{def}}$  by blast+
have path_image_ $\gamma_{\text{eq}}$ : "path_image  $\gamma = \text{parallelogram orig} - \text{parallelogram}'$ 
orig"
  unfolding  $\gamma_{\text{def}}$  parallelogram_def parallelogram'_def
    lattice.path_image_parallelogram_path_frontier_def ..
have "parallelogram' orig  $\subseteq$  parallelogram orig"
  unfolding parallelogram_def parallelogram'_def
  by (meson closure_subset dual_order.trans interior_subset)
hence parallelogram_conv_union: "parallelogram orig = parallelogram'
orig  $\cup$  path_image  $\gamma$ "
  using  $\gamma_{\text{subset}}$  path_image_ $\gamma_{\text{eq}}$  by blast

have  $\gamma$ : "valid_path  $\gamma$ " "path  $\gamma$ " "pathfinish  $\gamma = \text{pathstart } \gamma$ "
  "path_image  $\gamma \subseteq$  ball 0 R - {w  $\in$  ball 0 R. f w = 0}"
  using avoid R  $\gamma_{\text{subset}}$  by (auto simp:  $\gamma_{\text{def}}$  avoid_def intro!: valid_path_imp_path)

have inside: "winding_number  $\gamma$  w = 1" if w: "w  $\in$  parallelogram orig"
"f w = 0" for w
  using Im_t w avoid  $\gamma_{\text{disjoint}}$  parallelogram_conv_union unfolding  $\gamma_{\text{def}}$ 
  by (subst lattice.winding_number_parallelogram_inside)
  (auto simp: avoid_def parallelogram'_def parallelogram_def)

have outside: "winding_number  $\gamma$  w = 0"
  if w: "w  $\in$  {w  $\in$  ball 0 R. f w = 0} - parallelogram orig" for w
proof (rule winding_number_zero_outside)

```

```

show "convex (parallelogram orig)"
  unfolding parallelogram_def by auto
show "w ∉ parallelogram orig" using w avoid
  by auto
show "path_image γ ⊆ parallelogram orig"
  using path_image_γ_eq by blast
qed (use γ in auto)

have outside': "∀z. z ∉ ball 0 R → winding_number γ z = 0"
proof safe
  fix z :: complex assume z: "z ∉ ball 0 R"
  show "winding_number γ z = 0"
    by (rule winding_number_zero_outside[of _ "ball 0 R"])
      (use z γ in <auto simp: γ_def>)
qed

have "finite (cball 0 R ∩ {w. f w = 0})"
  by (rule sparse_in_compact_finite)
  (use ev_nz in <auto simp: eventually_cosparse intro: sparse_in_subset>)
hence fin: "finite {w ∈ ball 0 R. f w = 0}"
  by (rule finite_subset [rotated]) auto

have "contour_integral γ (λx. deriv f x / f x) / (2 * pi * i) = 1"
  unfolding γ_def by (subst argument_principle_f_1) (use avoid in <auto
simp: avoid_def>)
also have "contour_integral γ (λx. deriv f x / f x) =
  contour_integral γ (λx. deriv f x * 1 / f x)"
  by simp
also have "... / (2 * pi * i) =
  (∑ p ∈ {w ∈ ball 0 R. f w = 0}. winding_number γ p * 1 *
of_int (zorder f p))"
  by (subst argument_principle[of "ball 0 R"])
  (use γ fin outside' in <auto intro!: holomorphic>)
also have "(∑ p ∈ {w ∈ ball 0 R. f w = 0}. winding_number γ p * 1 *
of_int (zorder f p)) =
  (∑ p ∈ {w ∈ parallelogram orig. f w = 0}. of_int (zorder f
p))"
proof (intro sum.mono_neutral_cong_right ballI)
  show "{w ∈ parallelogram orig. f w = 0} ⊆ {w ∈ ball 0 R. f w = 0}"
    using R by (auto simp: parallelogram_def path_image_γ_eq)
qed (use fin in <auto simp: outside inside>)
finally have "of_int (∑ p ∈ {w ∈ parallelogram orig. f w = 0}. zorder
f p) = complex_of_int 1"
  unfolding of_int_sum by simp
hence sum_zorder_eq: "(∑ p ∈ {w ∈ parallelogram orig. f w = 0}. zorder
f p) = 1"
  by (simp only: of_int_eq_iff)

obtain root where root: "{w ∈ parallelogram orig. f w = 0} = {root}"

```

```

" f root = 0 " " zorder f root = 1 "
proof -
  have "int (card {w ∈ parallelogram orig. f w = 0}) = (∑ p∈{w ∈ parallelogram
orig. f w = 0}. 1)"
    by simp
  also have "... ≤ (∑ p∈{w ∈ parallelogram orig. f w = 0}. zorder f
p)"
    proof (rule sum_mono)
      fix w assume w: "w ∈ {w ∈ parallelogram orig. f w = 0}"
      have "∃F z in at w. f z ≠ 0" using ev_nz
        by (intro eventually_frequently eventually_cosparse_imp_eventually_at[OF
ev_nz]) auto
      hence "zorder f w > 0"
        by (subst zorder_pos_iff') (use w in <auto intro!: analytic>)
      thus "zorder f w ≥ 1"
        by simp
    qed
  also have "... = 1"
    by (fact sum_zorder_eq)
  finally have "card {w ∈ parallelogram orig. f w = 0} ≤ 1"
    by simp
  moreover have "card {w ∈ parallelogram orig. f w = 0} > 0"
  proof (subst card_gt_0_iff, safe)
    assume *: "{w ∈ parallelogram orig. f w = 0} = {}"
    show False
      using sum_zorder_eq unfolding * by simp
  next
    show "finite {w ∈ parallelogram orig. f w = 0}"
      by (rule finite_subset[OF _ fin]) (use R in <auto simp: parallelogram_def
path_image_γ_eq>)
  qed
  ultimately have "card {w ∈ parallelogram orig. f w = 0} = 1"
    by linarith
  then obtain w where w: "{w ∈ parallelogram orig. f w = 0} = {w}"
    by (auto simp: card_1_singleton_iff)
  moreover have "zorder f w = 1"
    using sum_zorder_eq unfolding w by simp
  ultimately show ?thesis
    using that by blast
qed

have "lattice.rel (contour_integral γ (λx. x * deriv f x / f x) / (2
* pi * i)) ((t + 1) / 2)"
  unfolding γ_def by (rule argument_principle_f_z) (use avoid in <auto
simp: avoid_def>)
also have "contour_integral γ (λx. x * deriv f x / f x) =
contour_integral γ (λx. deriv f x * x / f x)"
  by (simp add: mult_ac)

```

```

also have "... / (2 * pi * i) =
      ( $\sum_{p \in \{w \in \text{ball } 0 \text{ R. } f w = 0\}} \text{winding\_number } \gamma p * p * p *
\text{of\_int (zorder f p)}$ )"
  by (subst argument_principle[of "ball 0 R"])
      (use  $\gamma$  fin outside' in <auto intro!: holomorphic>)
also have " $(\sum_{p \in \{w \in \text{ball } 0 \text{ R. } f w = 0\}} \text{winding\_number } \gamma p * p * p *
\text{of\_int (zorder f p)}) =
      (\sum_{p \in \{w \in \text{parallelogram orig. } f w = 0\}} p * \text{of\_int (zorder
f p)})"$ "
proof (intro sum.mono_neutral_cong_right ballI)
  show "{w  $\in$  parallelogram orig. f w = 0}  $\subseteq$  {w  $\in$  ball 0 R. f w = 0}"
    using R by (auto simp: parallelogram_def path_image_γ_eq)
qed (use fin in <auto simp: outside inside>)
also have "... = root"
  unfolding root(1) using root(3) by simp
finally have root': "lattice.rel root ((t + 1) / 2)" .

show "f z = 0  $\longleftrightarrow$  lattice.rel z ((t + 1) / 2)"
proof
  assume "lattice.rel z ((t + 1) / 2)"
  also have "lattice.rel ((t + 1) / 2) root"
    using root' by (simp add: lattice.rel_sym)
  finally show "f z = 0"
    using f_zero_cong_lattice[of z root] using root by simp
next
  assume "f z = 0"
  obtain h where h: "bij_betw h (lattice.period_parallelogram z)
    (lattice.period_parallelogram orig)"
    " $\bigwedge w. \text{lattice.rel (h w) w}$ "
  using lattice.bij_betw_period_parallelograms[of z orig] by blast
  have "z  $\in$  lattice.period_parallelogram z"
  by auto
  hence "h z  $\in$  {z  $\in$  parallelogram orig. f z = 0}"
  using h(1) f_zero_cong_lattice[OF h(2)[of z]] <f z = 0>
  using bij_betw_apply closure_subset parallelogram_def by fastforce
  hence "h z = root"
  unfolding root(1) by simp
  hence "lattice.rel (h z) ((t+1) / 2)"
  using root' by simp
  thus "lattice.rel z ((t+1)/2)"
  using h(2) pre_complex_lattice.rel_symI pre_complex_lattice.rel_trans
by blast
qed

show "zorder f z = 1" if *: "lattice.rel z ((t+1)/2)"
proof -
  have "zorder f z = zorder f ((t+1)/2)"
  by (rule zorder_f_cong_lattice) fact
  also have "... = zorder f root"

```

```

    by (rule sym, rule zorder_f_cong_lattice) fact
    also have "... = 1"
    by fact
    finally show ?thesis .
qed
qed

```

Finally, we conclude that our quasi-periodic function is in fact a multiple of ϑ_{00} .

```

theorem multiple_jacobi_theta_00: "∃ c. ∀ z. f z = c * jacobi_theta_00
z t"

```

```

proof -

```

```

  define g where "g = (λ z. jacobi_theta_00 0 t * f z - f 0 * jacobi_theta_00
z t)"

```

```

  interpret g: thetalike_function g t

```

```

  proof

```

```

    show "g holomorphic_on UNIV"

```

```

    unfolding g_def using Im_t by (auto intro!: holomorphic_intros
holomorphic)

```

```

    show "Im t > 0"

```

```

    by (fact Im_t)

```

```

    show "g (z + 1) = g z" for z

```

```

    unfolding g_def

```

```

    by (simp add: f_plus_1 jacobi_theta_00_left.plus_1)

```

```

    show "g (z + t) = g z / to_nome (2 * z + t)" for z

```

```

    unfolding g_def using f_plus_quasiperiod[of z] jacobi_theta_00_plus_quasiperiod[of
z t]

```

```

    by (simp add: diff_divide_distrib add_ac)

```

```

  qed

```

```

show ?thesis

```

```

proof (rule exI[of _ "f 0 / jacobi_theta_00 0 t"], rule allI)

```

```

  fix z :: complex

```

```

  have "g z = 0"

```

```

  proof (rule ccontr)

```

```

    assume "g z ≠ 0"

```

```

    hence *: "¬(∀ z. g z = 0)"

```

```

    by auto

```

```

    have "g 0 = 0"

```

```

    by (simp add: g_def)

```

```

    also have "?this ↔ lattice.rel 0 ((t + 1) / 2)"

```

```

    by (rule g.zero_iff[OF *])

```

```

    also have "... ↔ (0 - ((t + 1) / 2)) ∈ lattice.lattice"

```

```

    by (simp add: lattice.rel_def)

```

```

    also have "0 - ((t + 1) / 2) = lattice.of_ω12_coords (-1/2, -1/2)"

```

```

    by (simp add: lattice.of_ω12_coords_def field_simps)

```

```

    finally show False

```

```

    by (subst (asm) lattice.of_ω12_coords_in_lattice_iff) auto

```

```

  qed

```

```

      thus "f z = f 0 / jacobi_theta_00 0 t * jacobi_theta_00 z t"
        using Im_t by (auto simp: field_simps jacobi_theta_00_0_left_nonzero
g_def)
      qed
    qed
  end

```

As a side effect, we also now know that the zeros of ϑ_{00} are all simple zeros.

```

lemma jacobi_theta_00_simple_zero:
  assumes "Im t > 0" "jacobi_theta_00 z t = 0"
  shows "zorder (λz. jacobi_theta_00 z t) z = 1"
proof -
  interpret thetalike_function "λz. jacobi_theta_00 z t"
  by unfold_locales
  (use assms(1)
  in <auto simp: jacobi_theta_00_plus_quasiperiod jacobi_theta_00_left.plus_1
add_ac
      intro!: holomorphic_intros>)
  have "¬(∀z. jacobi_theta_00 z t = 0)"
  using jacobi_theta_00_0_left_nonzero assms(1) by blast
  with assms(2) and zero_iff show ?thesis
  by blast
qed

```

8.2 Theta inversion

Using the fact that any quasiperiodic function (in the sense used above) is a multiple of ϑ_{00} and the heat equation for ϑ_{00} , we can now relatively easily prove the theta inversion identity, which describes how ϑ_{00} transforms under the modular transformation $t \mapsto -\frac{1}{t}$:

$$\vartheta_{00}(z, -1/t) = \sqrt{-ite^{i\pi tz^2}} \vartheta_{00}(tz, t)$$

In particular, this means that $\vartheta_{00}(0, t)$ is a modular form of weight $\frac{1}{2}$.

```

theorem jacobi_theta_00_minus_one_over:
  fixes z t :: complex
  assumes t: "Im t > 0"
  shows "jacobi_theta_00 z (-(1/t)) = csqrt (-(i*t)) * to_nome (t*z^2)
* jacobi_theta_00 (t*z) t"
proof -

```

First of all, we look at the quotient

$$\frac{e^{i\pi tz^2} \vartheta_{00}(tz, t)}{\vartheta_{00}(z, -1/t)}$$

and show that it does not depend on z .

The proof works by noting that, for fixed t with $\text{Im}(t) > 0$, the numerator is a theta-like function in z and must therefore be a constant multiple of the denominator.

```

define f where "f = (λz t. to_nome (t * z^2) * jacobi_theta_00 (t*z)
t)"
define g where "g = (λz t. f z t / jacobi_theta_00 z (-1/t))"
define c where "c = (λt. g 0 t)"
have [analytic_intros]: "c analytic_on A" if "A ⊆ {t. Im t > 0}" for
A
  unfolding c_def g_def f_def using that
  by (auto intro!: analytic_intros simp: Im_complex_div_lt_0 jacobi_theta_00_0_left_nonze

have f_eq: "f z t = c t * jacobi_theta_00 z (-1/t)" if t: "Im t > 0"
for z t
proof -
  from t have [simp]: "t ≠ 0"
  by auto
  interpret f: thetalike_function "λz. f z t" "-1/t"
  proof
    show "(λz. f z t) holomorphic_on UNIV"
      using t unfolding f_def by (auto intro!: holomorphic_intros)
  next
    show "Im (-1/t) > 0"
      using t by (simp add: Im_complex_div_lt_0)
  next
    show "f (z + 1) t = f z t" for z
      using jacobi_theta_00_plus_quasiperiod[of "t*z" t]
      by (simp add: f_def ring_distrib power2_eq_square field_simps
to_nome_add)
  next
    show "f (z + (-1/t)) t = f z t / to_nome (2 * z + (-1/t))" for
z
  proof -
    have "f (z + (-1/t)) t = to_nome (t * (z - 1 / t)^2) * jacobi_theta_00
(t * z - 1) t"
      by (simp add: f_def ring_distrib)
    also have "jacobi_theta_00 (t * z - 1) t = jacobi_theta_00 (t
* z) t"
      by (rule jacobi_theta_00_left.minus_1)
    also have "t * (z - 1 / t) ^ 2 = t * z ^ 2 - 2 * z + 1 / t"
      by (simp add: field_simps power2_eq_square)
    also have "to_nome ... * jacobi_theta_00 (t * z) t = f z t / to_nome
(2*z + (-1/t))"
      by (simp add: f_def to_nome_add to_nome_diff)
    finally show ?thesis .
  qed
qed

obtain c' where "f z t = c' * jacobi_theta_00 z (-1/t)" for z

```

```

using f.multiple_jacobi_theta_00 by blast
from this[of 0] and this[of z] show ?thesis
using jacobi_theta_00_0_left_nonzero[of "-1/t"] t
by (auto simp: g_def divide_simps Im_complex_div_lt_0 c_def)
qed

```

Next, we take the equation

$$e^{i\pi tz^2} \vartheta_{00}(tz, t) = c(t) \vartheta_{00}(z, -1/t)$$

and take the derivatives $\frac{\partial^2}{\partial z^2}$ of both sides and then, separately, $\frac{\partial}{\partial t}$ of both sides. We then use the heat equation for ϑ_{00} and combine both equations, which gives us the following ordinary differential equation:

$$c'(t) = -\frac{c(t)}{2t}$$

```

have c_ODE: "deriv c t = (-1 / (2*t)) * c t" if t: "Im t > 0" for t
proof -
  have [simp]: "t ≠ 0"
    using t by auto

  have "(deriv ^^ 2) (λz. f z t) 0 = (deriv ^^ 2) (λz. c t * jacobi_theta_00
z (-1/t)) 0"
    using t by (subst f_eq) auto
  also have "... = c t * (deriv ^^ 2) (λz. jacobi_theta_00 z (- 1 /
t)) 0"
    by (rule higher_deriv_cmult')
    (use t in <auto intro!: analytic_intros simp: Im_complex_div_lt_0>)
  also have "(deriv ^^ 2) (λz. jacobi_theta_00 z (- 1 / t)) 0 =
4 * pi * i * deriv (jacobi_theta_00 0) (-1/t)"
    by (subst jacobi_theta_00_heat_equation) (use t in <auto simp: Im_complex_div_lt_0>)
  also have "(deriv ^^ 2) (λz. f z t) 0 =
deriv (deriv (λz. jacobi_theta_00 (t * z) t)) 0 +
2 * deriv (λz. to_nome (t*z^2)) 0 * deriv (λz. jacobi_theta_00
(t * z) t) 0 +
deriv (deriv (λz. to_nome (t*z^2))) 0 * jacobi_theta_00
0 t"
    unfolding f_def using t
    by (subst higher_deriv_mult[of _ UNIV]) (auto intro!: holomorphic_intros
simp: eval_nat_numeral)
  also have "deriv (λz. to_nome (t*z^2)) = (λz. 2 * pi * i * t * z *
to_nome (t*z^2))"
    by (intro ext DERIV_imp_deriv) (auto intro!: derivative_eq_intros)
  also have "deriv ... = (λz. 2 * pi * i * t * to_nome (t*z^2) - 4 * pi^2
* t^2 * z^2 * to_nome (t*z^2))"
    by (intro ext DERIV_imp_deriv)
    (auto intro!: derivative_eq_intros simp: algebra_simps power2_eq_square)
  also have "deriv (λz. jacobi_theta_00 (t * z) t) = (λz. t * deriv
(λz. jacobi_theta_00 z t) (t*z))"

```

```

proof (rule ext, rule DERIV_imp_deriv)
  fix z :: complex
  have "((λz. jacobi_theta_00 z t) o (λz. t * z) has_field_derivative

      deriv (λz. jacobi_theta_00 z t) (t*z) * t) (at z)"
    by (rule DERIV_chain) (use t in <auto intro!: analytic_derivI
analytic_intros>)
  thus "((λz. jacobi_theta_00 (t*z) t) has_field_derivative
      t * deriv (λz. jacobi_theta_00 z t) (t*z)) (at z)"
    by (simp add: algebra_simps o_def)
qed
also have "deriv ... 0 = t * deriv (λz. deriv (λz. jacobi_theta_00
z t) (t*z)) 0"
  by (subst deriv_cmult') (use t in <auto intro!: analytic_intros
simp: eval_nat_numeral>)
also have "deriv (λz. deriv (λz. jacobi_theta_00 z t) (t*z)) 0 =
      t * (deriv ^^ 2) (λz. jacobi_theta_00 z t) 0"
proof (rule DERIV_imp_deriv)
  have "(deriv (λz. jacobi_theta_00 z t) o (λz. t * z) has_field_derivative

      deriv (deriv (λz. jacobi_theta_00 z t)) 0 * t) (at 0)"
    by (rule DERIV_chain) (use t in <auto intro!: analytic_derivI
analytic_intros>)
  thus "((λz. deriv (λz. jacobi_theta_00 z t) (t * z)) has_field_derivative
      t * (deriv ^^ 2) (λz. jacobi_theta_00 z t) 0) (at 0)"
    by (simp add: algebra_simps o_def eval_nat_numeral)
qed
also have "(deriv ^^ 2) (λz. jacobi_theta_00 z t) 0 = 4 * pi * i *
deriv (jacobi_theta_00 0) t"
  using t by (subst jacobi_theta_00_heat_equation) simp_all
finally have "4 * pi * i * (t^2 * deriv (jacobi_theta_00 0) t +
      t * jacobi_theta_00 0 t / 2) =
      4 * pi * i * (c t * deriv (jacobi_theta_00 0) (- (1 /
t))))"
  unfolding ring_distrib by (simp add: field_simps power2_eq_square)
hence *: "c t * deriv (jacobi_theta_00 0) (-1/t) =
      t^2 * deriv (jacobi_theta_00 0) t + t * jacobi_theta_00
0 t / 2"
  by (subst (asm) mult_cancel_left) auto

have "deriv (f 0) t = deriv (λt. c t * jacobi_theta_00 0 (-1/t)) t"
proof (rule deriv_cong_ev)
  have "eventually (λt. t ∈ {t. Im t > 0}) (nhds t)"
    by (rule eventually_nhds_in_open) (use t in <auto simp: open_halfspace_Im_gt>)
  thus "∀F x in nhds t. f 0 x = c x * jacobi_theta_00 0 (- 1 / x)"
    by eventually_elim (auto simp: f_eq)
qed auto
also have "... = c t * deriv (λt. jacobi_theta_00 0 (- (1 / t))) t
+

```

```

      deriv c t * jacobi_theta_00 0 (- (1 / t))"
    by (subst complex_derivative_mult_at)
      (use t in <auto intro!: analytic_intros simp: Im_complex_div_lt_0>)
  also have "deriv (λt. jacobi_theta_00 0 (- (1 / t))) t =
    deriv (jacobi_theta_00 0) (-1/t) * deriv (λt. -(1/t))
t"
    by (rule deriv_compose_analytic)
      (use t in <auto intro!: analytic_intros simp: Im_complex_div_lt_0>)
  also have "deriv (λt. -(1/t)) t = 1 / t ^ 2"
    using t by (auto intro!: DERIV_imp_deriv derivative_eq_intros simp:
power2_eq_square)
  also have "deriv (f 0) t = deriv (jacobi_theta_00 0) t"
    unfolding f_def by simp
  finally have "deriv (jacobi_theta_00 0) t =
    c t * deriv (jacobi_theta_00 0) (-1/t) / t ^ 2
+
    deriv c t * jacobi_theta_00 0 (-1/t))"
    by simp
  also note *
  finally have "jacobi_theta_00 0 t / (2*t) + deriv c t * jacobi_theta_00
0 (-1/t) = 0"
    by (simp add: add_divide_distrib power2_eq_square)

  also have "jacobi_theta_00 0 t = c t * jacobi_theta_00 0 (- (1 / t))"
    using f_eq[of t 0] t by (simp add: f_def)
  also have "c t * jacobi_theta_00 0 (-1/t) / (2 * t) + deriv c t
* jacobi_theta_00 0 (-1/t) =
    jacobi_theta_00 0 (-1/t) * (c t / (2*t) + deriv c t)"
    by (simp add: algebra_simps)
  finally have "c t / (2*t) + deriv c t = 0"
    unfolding mult_eq_0_iff using jacobi_theta_00_0_left_nonzero[of
"-1/t"] t
    by (simp_all add: Im_complex_div_lt_0)
  thus "deriv c t = (-1 / (2*t)) * c t"
    using t by (simp add: field_simps add_eq_0_iff)
qed

```

This is a particularly simple separable ODE, which has the unique general solution $c(t) = C/\sqrt{t}$ for some constant C .

```

  have "∃C. ∀t∈{t. Im t > 0}. c t = C * (1 / csqrt t)"
    using c_ODE
  proof (rule separable_ODE_simple_unique)
    show "(c has_field_derivative deriv c t) (at t within {t. 0 < Im
t})"
      if "t ∈ {t. Im t > 0}" for t
      by (intro analytic_derivI analytic_intros) (use that in auto)
  next
    show "((λx. 1 / csqrt x) has_field_derivative - 1 / (2 * t) * (1
/ csqrt t))

```

```

      (at t within {t. 0 < Im t})" if "t ∈ {t. Im t > 0}" for t
using that
  by (auto intro!: derivative_eq_intros simp: complex_nonpos_Reals_iff

      simp flip: power2_eq_square)
qed (auto simp: convex_halfspace_Im_gt)
then obtain C where C: "∧t. Im t > 0 ⇒ c t = C / csqrt t"
  by auto

```

Noting that $c(i) = 1$, we find that $C = \sqrt{i}$ and therefore $c(t) = 1/\sqrt{-it}$ as desired.

```

have "c i = C / csqrt i"
  by (rule C) auto
also have "c i = 1"
  by (simp add: c_def g_def f_def jacobi_theta_00_0_left_nonzero)
finally have C_eq: "C = csqrt i"
  by (simp add: field_simps)

have c_eq: "c t = 1 / csqrt (- i * t)" if t: "Im t > 0" for t
proof -
  have [simp]: "t ≠ 0"
    using t by auto
  have "csqrt (i * (-i * t)) = csqrt i * csqrt (-i * t)"
  proof (rule csqrt_mult)
    have "Arg (-i * t) = Arg (-i) + Arg t"
      by (subst Arg_times) (use Arg_lt_pi[of t] t in auto)
    thus "Arg i + Arg (-i * t) ∈ {- pi < . pi}"
      using Arg_lt_pi[of t] t by auto
  qed
  hence "csqrt i / csqrt t = 1 / csqrt (-i * t)"
    by (simp add: field_simps del: csqrt_ii)
  also have "csqrt i / csqrt t = c t"
    by (subst C) (use t in <auto simp: C_eq>)
  finally show ?thesis .
qed

have "to_nome (t * z^2) * jacobi_theta_00 (t * z) t = c t * jacobi_theta_00
z (- 1 / t)"
  using f_eq[of t z] t by (simp add: f_def)
also have "c t = 1 / csqrt (-i * t)"
  using t by (rule c_eq)
finally show ?thesis
  using t by (auto simp add: field_simps)
qed

```

Equivalent identities for the other ϑ_{xx} follow:

```

lemma jacobi_theta_01_minus_one_over:
  fixes z t :: complex
  assumes "Im t > 0"

```

```

shows "jacobi_theta_01 z (-1/t) = csqrt (-i*t) * to_nome (t*z^2)
* jacobi_theta_10 (t*z) t"
by (simp add: jacobi_theta_01_def jacobi_theta_10_def jacobi_theta_00_minus_one_over
ring_distrib power2_eq_square to_nome_add add_divide_distrib
assms)

```

```

lemma jacobi_theta_10_minus_one_over:
  fixes z t :: complex
  assumes "Im t > 0"
  shows "jacobi_theta_10 z (-1/t) = csqrt (-i*t) * to_nome (t*z^2)
* jacobi_theta_01 (t*z) t"
proof -
  have [simp]: "t ≠ 0"
    using assms by auto
  have "jacobi_theta_10 z (-1/t) = csqrt (-i*t) *
    (to_nome (z - 1 / (t * 4)) * to_nome (t * (z - 1 / (t * 2))^2))"
  *
    jacobi_theta_00 (t * z - 1 / 2) t" using assms
  by (simp add: jacobi_theta_10_def jacobi_theta_00_minus_one_over
power2_eq_square ring_distrib mult_ac)
  also have "to_nome (z - 1 / (t * 4)) * to_nome (t * (z - 1 / (t * 2))^2)
=
    to_nome (z - 1 / (t * 4) + t * (z - 1 / (t * 2))^2)"
  by (rule to_nome_add [symmetric])
  also have "z - 1 / (t * 4) + t * (z - 1 / (t * 2))^2 = t * z^2"
  by (auto simp: field_simps power2_eq_square)
  also have "jacobi_theta_00 (t * z - 1 / 2) t = jacobi_theta_01 (t *
z - 1) t"
  by (simp add: jacobi_theta_01_def)
  also have "jacobi_theta_01 (t * z - 1) t = jacobi_theta_01 (t * z) t"
  by (rule jacobi_theta_01_left.minus_1)
  finally show ?thesis by simp
qed

```

```

lemma jacobi_theta_11_minus_one_over:
  fixes z t :: complex
  assumes t: "Im t > 0"
  shows "jacobi_theta_11 z (-1/t) = -i * csqrt (-i*t) * to_nome (t*z^2)
* jacobi_theta_11 (t*z) t"
proof -
  have [simp]: "t ≠ 0"
    using assms by auto
  have "jacobi_theta_11 z (-1/t) = i * csqrt (-i*t) * to_nome (t * z^2)
*
    (to_nome (t*z + t/4 - 1/2) * jacobi_theta_00 (t * z - 1 / 2
+ t / 2) t)" using assms
  by (simp add: jacobi_theta_11_def jacobi_theta_00_minus_one_over add_ac
power2_eq_square ring_distrib mult_ac to_nome_add to_nome_diff)
  also have "to_nome (t * z + t / 4 - 1 / 2) * jacobi_theta_00 (t * z

```

```

- 1 / 2 + t / 2) t =
      jacobi_theta_11 (t*z - 1) t"
  by (simp add: jacobi_theta_11_def algebra_simps)
also have "... = -jacobi_theta_11 (t*z) t"
  by (rule jacobi_theta_11_minus1_left)
finally show ?thesis
  by simp
qed

```

8.3 Theta nullwert inversions in the reals

We can thus translate the above theta inversion identities into the q -disc. For simplicity, we only do this for real q with $0 < q < 1$, and we will focus mostly on the theta nullwert functions, where the identities are particularly nice (and stay within the reals).

We introduce the “ q inversion” function

$$f : [0, 1] \rightarrow [0, 1], \quad f(q) = \exp(\pi^2 / \log q)$$

with the border values $f(0) = 1$ and $f(1) = 0$. This function is a strictly decreasing involution on the real interval $[0, 1]$. It corresponds to translating q from the q -disc to the z -plane, doing the transformation $z \mapsto -1/z$, and then translating the result back into the q -disc.

This is useful for computing $\vartheta_i(q)$, since we can apply the inversion to bring any q in the unit disc very close to 0, where the power series of ϑ_i converges extremely quickly.

definition `q_inversion` :: "real \Rightarrow real" where
 "q_inversion q = (if q = 0 then 1 else if q = 1 then 0 else exp (pi² / ln q))"

lemma `q_inversion_0` [simp]: "q_inversion 0 = 1"
 and `q_inversion_1` [simp]: "q_inversion 1 = 0"
 by (simp_all add: q_inversion_def)

lemma `q_inversion_nonneg`: "q \in {0..1} \implies q_inversion q \geq 0"
 and `q_inversion_le_1`: "q \in {0..1} \implies q_inversion q \leq 1"
 and `q_inversion_pos`: "q \in {0.. $<$ 1} \implies q_inversion q $>$ 0"
 and `q_inversion_less_1`: "q \in {0<..1} \implies q_inversion q $<$ 1"
 by (auto simp: q_inversion_def field_simps)

lemma `q_inversion_strict_antimono`: "strict_antimono_on {0..1} q_inversion"
 unfolding `monotone_on_def` by (auto simp: q_inversion_def field_simps)

lemma `q_inversion_less_iff`:
 assumes "q \in {0..1}" "q' \in {0..1}"
 shows "q_inversion q $<$ q_inversion q' \iff q $>$ q'"
 using `assms` by (auto simp: q_inversion_def field_simps)

```

lemma q_inversion_le_iff:
  assumes "q ∈ {0..1}" "q' ∈ {0..1}"
  shows "q_inversion q ≤ q_inversion q' ↔ q ≥ q'"
  using assms by (auto simp: q_inversion_def field_simps)

lemma q_inversion_eq_iff:
  assumes "q ∈ {0..1}" "q' ∈ {0..1}"
  shows "q_inversion q = q_inversion q' ↔ q = q'"
  using assms by (auto simp: q_inversion_def field_simps)

lemma q_inversion_involution:
  assumes "q ∈ {0..1}"
  shows "q_inversion (q_inversion q) = q"
  using assms by (auto simp: q_inversion_def)

lemma continuous_q_inversion [continuous_intros]:
  assumes q: "q ∈ {0..1}"
  shows "continuous (at q within {0..1}) q_inversion"
proof -
  define f where "f = (λq. exp (pi ^ 2 / ln q))"
consider "q = 0" | "q = 1" | "q ∈ {0<..<1}"
  using q by force
hence "(f → q_inversion q) (at q within {0..1})"
proof cases
  assume q: "q = 0"
  have "(f → 1) (at_right 0)"
    unfolding f_def by real_asymp
  thus ?thesis
    by (simp add: q f_def at_within_Icc_at_right)
next
  assume q: "q = 1"
  have "(f → 0) (at_left 1)"
    unfolding f_def by real_asymp
  thus ?thesis
    by (simp add: q f_def at_within_Icc_at_left)
next
  assume q: "q ∈ {0<..<1}"
  have "isCont f q"
    using q unfolding f_def by (auto intro!: continuous_intros)
  hence "(f → f q) (at q within {0..1})"
    by (meson Lim_at_imp_Lim_at_within continuous_within)
  thus ?thesis
    using q by (simp add: q_inversion_def f_def)
qed
moreover have "eventually (λq. f q = q_inversion q) (at q within {0..1})"
  using eventually_neq_at_within[of 0] eventually_neq_at_within[of 1]
  by eventually_elim (auto simp: q_inversion_def f_def)

```

```

ultimately have "(q_inversion  $\longrightarrow$  q_inversion q) (at q within {0..1})"
  by (rule Lim_transform_eventually)
thus ?thesis
  using continuous_within by blast
qed

```

```

lemma continuous_on_q_inversion [continuous_intros]: "continuous_on {0..1}
q_inversion"
  using continuous_q_inversion continuous_on_eq_continuous_within by blast

```

```

lemma continuous_on_q_inversion' [continuous_intros]:
  assumes "continuous_on A f" " $\bigwedge x. x \in A \implies f x \in \{0..1\}$ "
  shows "continuous_on A ( $\lambda x. q\_inversion (f x)$ )"
  by (rule continuous_on_compose2[OF continuous_on_q_inversion assms(1)])
(use assms(2) in auto)

```

```

definition q_inversion_fixedpoint :: real where
  "q_inversion_fixedpoint = exp (-pi)"

```

```

lemma q_inversion_fixedpoint:
  defines "q0  $\equiv$  q_inversion_fixedpoint"
  shows "q0  $\in$  {0..1}" "q_inversion q0 = q0"
proof -
  show "q0  $\in$  {0..1}"
    by (auto simp: q0_def q_inversion_fixedpoint_def)
  show "q_inversion q0 = q0"
    by (auto simp: q_inversion_def q0_def q_inversion_fixedpoint_def field_simps
power2_eq_square)
qed

```

```

lemma q_inversion_less_self_iff:
  assumes "q  $\in$  {0..1}"
  shows "q_inversion q < q  $\iff$  q > q_inversion_fixedpoint"
  using q_inversion_less_iff[OF assms q_inversion_fixedpoint(1)] q_inversion_fixedpoint(2)
  by auto

```

```

lemma q_inversion_greater_self_iff:
  assumes "q  $\in$  {0..1}"
  shows "q_inversion q > q  $\iff$  q < q_inversion_fixedpoint"
  using q_inversion_less_iff[OF q_inversion_fixedpoint(1) assms] q_inversion_fixedpoint(2)
  by auto

```

From the theta inversion identities, we get three identities of the form $\vartheta_i(f(q)) = \sqrt{-\ln q/\pi} \vartheta_j(q)$. This can be harnessed to evaluate the theta nullwert functions very rapidly: their power series converge extremely quickly for small q , and since $f(q)$ has a unique fixed point $q_0 = e^{-\pi} \approx 0.0432$, we can reduce the computation of theta nullwert functions to computing them for q with $q \leq q_0$ via the inversion formulas.

```

lemma jacobi_theta_nome_inversion_real:
  fixes w q :: real
  assumes q: "q ∈ {0<..<1}" and w: "w > 0"
  shows "jacobi_theta_nome (of_real w) (of_real q) =
    complex_of_real (sqrt (- pi / ln q) * exp (- (ln w)2 / (4 *
ln q))) *
    jacobi_theta_nome (cis (-pi * ln w / ln q)) (of_real (exp (pi2
/ ln q)))"
proof -
  define x where "x = ln (of_real w) / (2 * pi * i)"
  define y where "y = of_real (ln q) / (pi * i)"
  have w_eq: "w = to_nome x ^ 2"
    using w by (simp add: x_def to_nome_def exp_of_real flip: exp_of_nat_mult)
  have q_eq: "q = to_nome y"
    using q by (simp add: y_def to_nome_def exp_of_real)
  have y: "y ≠ 0" "Im y > 0"
    using q by (auto simp: y_def Im_complex_div_gt_0 mult_neg_pos)

  have "jacobi_theta_nome (of_real w) (of_real q) = jacobi_theta_00 x
y"
    by (simp add: w_eq q_eq jacobi_theta_00_def flip: jacobi_theta_nome_of_real)
  also have "... = jacobi_theta_00 x (-1/(-1/y))"
    using <y ≠ 0> by auto
  also have "... = csqrt (i / y) * to_nome (- (x2 / y)) * jacobi_theta_00
(-(x/y)) (-1/y)"
    by (subst jacobi_theta_00_minus_one_over) (use y in <auto simp: Im_complex_div_lt_0>)
  also have "i / y = -of_real (pi / ln q)"
    by (auto simp: y_def field_simps)
  also have "jacobi_theta_00 (-(x/y)) (-1/y) =
    jacobi_theta_nome (inverse (to_nome (x / y)) ^ 2) (inverse
(to_nome (1 / y)))"
    by (simp add: jacobi_theta_00_def to_nome_power to_nome_minus)
  also have "inverse (to_nome (1 / y)) = exp (of_real (pi ^ 2 / ln q))"
    using q by (simp add: to_nome_def y_def field_simps exp_minus power2_eq_square)
  also have "... = of_real (exp (pi ^ 2 / ln q))"
    by (rule exp_of_real)
  also have "inverse (to_nome (x / y)) ^ 2 =
    exp (-i * (complex_of_real pi * Ln (complex_of_real w))
/ complex_of_real (ln q))"
    by (simp add: to_nome_def x_def y_def exp_minus field_simps flip:
exp_of_nat_mult)
  also have "-i * (complex_of_real pi * Ln (complex_of_real w)) / complex_of_real
(ln q) =
    -i * complex_of_real (pi * ln w / ln q)"
    using q w by (subst Ln_of_real') (auto simp: field_simps power2_eq_square)
  also have "exp ... = cis (-pi * ln w / ln q)"
    by (auto simp: exp_diff exp_add cis_conv_exp exp_minus field_simps)
  also have "csqrt (- complex_of_real (pi / ln q)) = csqrt (of_real (-pi
/ ln q))"

```

```

    by simp
  also have "... = of_real (sqrt (-pi / ln q))"
    by (subst of_real_sqrt) (use q in <auto simp: field_simps>)
  also have "to_nome (- (x2 / y)) = exp (-ln (of_real w) ^ 2 / of_real
(4 * ln q))"
    unfolding x_def y_def by (simp add: field_simps to_nome_def power2_eq_square)
  also have "-ln (complex_of_real w) ^ 2 / of_real (4 * ln q) =
of_real (-ln w ^ 2) / (4 * ln q)"
    by (subst Ln_of_real') (use w q in <auto simp: field_simps power2_eq_square>)
  also have "exp ... = complex_of_real (exp (-ln w ^ 2) / (4 * ln q))"
    by (rule exp_of_real)
  finally show ?thesis by simp
qed

```

```

lemma jacobi_theta_nome_1_left_inversion_real:
  assumes q: "q ∈ {0..<1}"
  shows "jacobi_theta_nome 1 (q_inversion q) = sqrt (-ln q / pi) * jacobi_theta_nome
1 q"
proof (cases "q = 0")
  case False
  with assms have q: "q ∈ {0<..<1}"
  by auto
  have "complex_of_real (jacobi_theta_nome 1 q) = jacobi_theta_nome (of_real
1) (of_real q)"
  by (simp flip: jacobi_theta_nome_of_real)
  also have "... = of_real (sqrt (-pi / ln q) * jacobi_theta_nome 1 (exp
(pi2 / ln q)))"
  by (subst jacobi_theta_nome_inversion_real) (use q in <auto simp flip:
jacobi_theta_nome_of_real>)
  finally have "jacobi_theta_nome 1 q = sqrt (-pi / ln q) * jacobi_theta_nome
1 (q_inversion q)"
  unfolding of_real_eq_iff q_inversion_def using q by simp
  thus ?thesis
  using q by (simp add: real_sqrt_divide field_simps real_sqrt_minus)
qed auto

```

```

lemma jacobi_theta_nw_00_inversion_real:
  assumes q: "q ∈ {0..<1::real}"
  shows "jacobi_theta_nw_00 (q_inversion q) = sqrt (-ln q / pi) * jacobi_theta_nw_00
q"
  unfolding jacobi_theta_nome_00_def power_one
  by (subst jacobi_theta_nome_1_left_inversion_real [OF q]) auto

```

```

lemma jacobi_theta_nw_01_inversion_real:
  assumes q: "q ∈ {0..<1::real}"
  shows "jacobi_theta_nw_01 (q_inversion q) = sqrt (-ln q / pi) * jacobi_theta_nw_10
q"
proof (cases "q = 0")
  case False

```

```

with assms have q: "q ∈ {0<..<1}"
  by auto
have "complex_of_real (jacobi_theta_nw_10 q) =
      of_real (q powr (1/4)) * jacobi_theta_nome (of_real q) (of_real
q)"
  by (simp add: jacobi_theta_nome_10_def jacobi_theta_nome_of_real)
  also have "... = of_real (sqrt (-pi/ln q)) *
      jacobi_theta_nome (-1) (complex_of_real (exp (pi2 /
ln q)))"
  by (subst jacobi_theta_nome_inversion_real)
      (use q in <auto simp flip: cis_inverse simp: power2_eq_square powr_def
exp_minus>)
  also have "... = complex_of_real (sqrt (-pi/ln q) * jacobi_theta_nw_01
(exp (pi2 / ln q)))"
  by (simp flip: jacobi_theta_nome_of_real add: jacobi_theta_nome_01_def)
  finally have "jacobi_theta_nw_10 q = sqrt (-pi / ln q) * jacobi_theta_nw_01
(q_inversion q)"
  unfolding of_real_eq_iff q_inversion_def using q by simp
  thus ?thesis
  using q by (simp add: real_sqrt_divide field_simps real_sqrt_minus)
qed auto

lemma jacobi_theta_nw_10_inversion_real:
  assumes q: "q ∈ {0<..<1::real}"
  shows "jacobi_theta_nw_10 (q_inversion q) = sqrt (-ln q / pi) * jacobi_theta_nw_01
q"
proof -
  have "jacobi_theta_nw_01 q = jacobi_theta_nw_01 (q_inversion (q_inversion
q))"
  using q by (simp add: q_inversion_involution)
  also have "... = sqrt (-ln (q_inversion q) / pi) * jacobi_theta_nw_10
(q_inversion q)"
  by (subst jacobi_theta_nw_01_inversion_real)
      (use q in <auto simp: q_inversion_less_1 q_inversion_nonneg>)
  also have "-ln (q_inversion q) / pi = -pi / ln q"
  using q by (simp add: q_inversion_def power2_eq_square)
  finally show ?thesis
  using q by (simp add: real_sqrt_divide field_simps real_sqrt_minus)
qed

end

```

9 Eisenstein series and the differential equations of \wp

```

theory Eisenstein_Series
imports
  Weierstrass_Elliptic

```

```

"Elliptic_Functions.Z_Plane_Q_Disc"
"Polynomial_Factorization.Fundamental_Theorem_Algebra_Factorized"
"Zeta_Function.Zeta_Function"
"Polylog.Polylog"
"Lambert_Series.Lambert_Series"
"Cotangent_PFD_Formula.Cotangent_PFD_Formula"
"Algebraic_Numbers.Bivariate_Polynomials"
Theta_Inversion
begin

unbundle jacobi_theta_notation

lemmas [simp del] = div_mult_self1 div_mult_self2 div_mult_self3 div_mult_self4

```

We define the Eisenstein series G_n , which is the sequence of coefficients of the Laurent series expansion of \wp . Both \wp and G_n (for $n \geq 3$) are invariants of the lattice, i.e. they are independent from the choice of generators.

9.1 Definition

For $n \geq 3$, the Eisenstein series G_n is defined simply as the absolutely convergent sum $\sum_{\omega \in \Lambda^*} \omega^{-n}$. However, we want to stay as general as possible here and therefore define it in such a way that the definition also works for $n = 2$, where the sum is only conditionally convergent and much less well-behaved.

Note that all the Eisenstein series with odd $n \geq 3$ vanish due to the symmetry in the sum. As for $n < 3$, we define $G_1 = 0$ in agreement with the the values for other odd n and $G_0 = 1$ since this makes some later theorem statements regarding modular forms more elegant.

```

context complex_lattice
begin

definition eisenstein_series :: "nat  $\Rightarrow$  complex" where
  "eisenstein_series k = (if k = 0 then -1 else if odd k then 0 else
    2 /  $\omega_1$  ^ k * zeta (of_nat k) + ( $\sum_{\infty n \in -\{0\}}$ .  $\sum_{\infty m}$ . 1 / of_omega12_coords
    (of_int m, of_int n) ^ k))"

notation eisenstein_series ("G")

lemma eisenstein_series_0 [simp]: "eisenstein_series 0 = -1"
  by (auto simp: eisenstein_series_def)

lemma eisenstein_series_odd_eq_0 [simp]: "odd k  $\implies$  eisenstein_series
k = 0"
  by (auto simp: eisenstein_series_def elim!: oddE)

```

```

lemma eisenstein_series_Suc_0 [simp]: "eisenstein_series (Suc 0) = 0"
  by (rule eisenstein_series_odd_eq_0) auto

lemma eisenstein_series_norm_summable:
  assumes "n ≥ 3"
  shows "(λω. 1 / norm ω ^ n) summable_on Λ*"
  using converges_absolutely_iff[of "of_nat n"] assms by (simp add: powr_realpow)

lemma eisenstein_series_summable:
  assumes "n ≥ 3"
  shows "(λω. 1 / ω ^ n) summable_on Λ*"
  by (rule abs_summable_summable)
  (use eisenstein_series_norm_summable[OF assms] in <simp add: norm_divide
norm_power>)

lemma eisenstein_series_has_sum:
  assumes "k ≥ 3"
  shows "((λω. 1 / ω ^ k) has_sum eisenstein_series k) Λ*"
proof (cases "even k")
  case odd: False
  define S where "S = (∑∞ ω ∈ Λ*. 1 / ω ^ k)"
  have sum1: "((λω. 1 / ω ^ k) has_sum S) Λ*"
    using eisenstein_series_summable[OF assms] has_sum_infsun unfolding
    S_def by blast
  also have "?thesis ↔ ((λω. -(1 / ω ^ k)) has_sum S) Λ*"
    using assms odd
    by (intro has_sum_reindex_bij_witness[of _ uminus uminus]) (auto simp:
    uminus_in_lattice0_iff)
  also have "... ↔ ((λω. 1 / ω ^ k) has_sum (-S)) Λ*"
    by (rule has_sum_uminus)
  finally have "-S = S"
    using sum1 has_sum_unique by blast
  hence "S = 0"
    by simp
  thus ?thesis
    using sum1 odd assms by simp
next
  case even: True
  define f where "f = (λ(m,n). 1 / of_ω12_coords (of_int m, of_int n)
  ^ k)"

  note k = <k ≥ 3> even
  have "((λm. 1 / of_int m ^ k) has_sum (2 * zeta (of_nat k))) (-{0})"
    by (rule has_sum_zeta_symmetric) (use k in auto)
  hence "((λm. (1 / ω1 ^ k) * (1 / of_int m ^ k))
    has_sum ((1 / ω1 ^ k) * (2 * zeta (of_nat k)))) (-{0})"
    by (rule has_sum_cmult_right)
  hence "((λm. 1 / (of_int m * ω1) ^ k) has_sum (2 / ω1 ^ k * zeta (of_nat

```

```

k))) (-{0})"
  by (simp add: field_simps)
  also have "?this  $\longleftrightarrow$  (f has_sum (2 /  $\omega_1$  ^ k * zeta (of_nat k))) ((-{0})
 $\times$  {0})"
  by (rule has_sum_reindex_bij_witness[of _ fst " $\lambda m. (m, 0)$ "]) (auto
simp: f_def)
  finally have sum1: "(f has_sum (2 /  $\omega_1$  ^ k * zeta (of_nat k))) ((-{0})
 $\times$  {0})" .

define S where "S = ( $\sum_{\omega \in \Lambda^*} 1 / \omega ^ k$ )"
define T where "T = ( $\lambda n. \sum_{m} 1 / \text{of\_}\omega_{12}\text{\_coords (of\_int m, of\_int
n) ^ k}$ )"

have sum2: "(( $\lambda \omega. 1 / \omega ^ k$ ) has_sum S)  $\Lambda^*$ "
  unfolding S_def using eisenstein_series_summable by (rule has_sum_infsum)
(use k in auto)
  also have "?this  $\longleftrightarrow$  (f has_sum S) (-{(0,0)})"
  by (subst has_sum_reindex_bij_betw[OF bij_betw_lattice0', symmetric])
  (simp_all add: case_prod_unfold map_prod_def f_def)
  finally have "(f has_sum S) (-{(0, 0)})" .
  hence "(f has_sum (S - 2 /  $\omega_1$  ^ k * zeta (of_nat k))) (-{(0, 0)} -
((-{0})  $\times$  {0}))"
  by (intro has_sum_Diff sum1) auto
  also have "-{(0, 0)} - ((-{0})  $\times$  {0}) = (UNIV  $\times$  (-{0})) :: (int  $\times$  int)
set)"
  by auto
  finally have "(f has_sum S - 2 /  $\omega_1$  ^ k * zeta (of_nat k)) (UNIV  $\times$  (-{0}))"
  .

  also have "?this  $\longleftrightarrow$  (( $\lambda (n,m). f (m,n)$ ) has_sum S - 2 /  $\omega_1$  ^ k * zeta
(of_nat k)) ((-{0})  $\times$  UNIV)"
  by (rule has_sum_reindex_bij_witness[of _ prod.swap prod.swap]) auto
  finally have sum3: "(( $\lambda (n,m). f (m,n)$ ) has_sum S - 2 /  $\omega_1$  ^ k * zeta
(of_nat k)) ((-{0})  $\times$  UNIV)" .

have "(T has_sum S - 2 /  $\omega_1$  ^ k * zeta (complex_of_nat k)) (-{0})"
  using sum3
proof (rule has_sum_SigmaD, unfold prod.case)
  fix n :: int assume n: "n  $\in$  -{0}"
  have "( $\lambda m. f (m, n)$ ) summable_on UNIV"
  using summable_on_SigmaD1[OF has_sum_imp_summable[OF sum3, unfolded
f_def], OF n]
  by (simp add: f_def)
  thus "(( $\lambda m. f (m, n)$ ) has_sum T n) UNIV"
  unfolding T_def f_def prod.case by (rule has_sum_infsum)
qed
hence "S = 2 /  $\omega_1$  ^ k * zeta (of_nat k) + ( $\sum_{m \in -\{0\}} T m$ )"
  by (simp add: has_sum_iff)
also have "... = eisenstein_series k"
  using k by (simp add: eisenstein_series_def T_def)

```

```

    finally show ?thesis
      using sum2 by simp
qed

lemma eisenstein_series_altdef:
  assumes "k ≥ 3"
  shows "eisenstein_series k = (∑∞ ω ∈ Λ*. 1 / ω ^ k)"
  using eisenstein_series_has_sum[OF assms] by (simp add: has_sum_iff)

lemma eisenstein_fun_aux_0 [simp]:
  assumes "n ≠ 2"
  shows "eisenstein_fun_aux n 0 = eisenstein_series n"
proof (cases "n ≥ 3")
  case gt3: True
  show ?thesis
  proof (cases "even n")
    case True
    show ?thesis
      using True gt3 by (auto simp: eisenstein_fun_aux_def eisenstein_series_altdef)
  next
    case False
    have "eisenstein_fun_aux n 0 = -(∑∞ ω ∈ Λ*. 1 / ω ^ n)"
      using False gt3
      by (auto simp: eisenstein_fun_aux_def eisenstein_series_def infsum_uminus)
    also have "... = -eisenstein_series n"
      using gt3 by (simp add: eisenstein_series_altdef)
    finally show ?thesis
      using False by (simp add: eisenstein_series_odd_eq_0)
  qed
next
  case False
  hence "n ∈ {0,1,2}"
  by auto
  thus ?thesis using assms
  by (auto simp: eisenstein_fun_aux_def)
qed

```

9.2 The Laurent series expansion of \wp at the origin

```

lemma higher_deriv_weierstrass_fun_aux_0:
  assumes "m > 0"
  shows "(deriv ^^ m) weierstrass_fun_aux 0 = (- 1) ^ m * fact (Suc
m) * G (m + 2)"
proof -
  define n where "n = m - 1"
  have "(deriv ^^ Suc n) weierstrass_fun_aux 0 = (deriv ^^ n) (λw. -2
* eisenstein_fun_aux 3 w) 0"
    unfolding funpow_Suc_right o_def
  proof (rule higher_deriv_cong_ev)

```

```

    have "eventually (λz. z ∈ -Λ*) (nhds 0)"
      using closed_lattice0 by (intro eventually_nhds_in_open) auto
    thus "∀F x in nhds 0. deriv weierstrass_fun_aux x = -2 * eisenstein_fun_aux
3 x"
      by eventually_elim (simp add: deriv_weierstrass_fun_aux)
qed auto
also have "... = -2 * (deriv ^^ n) (eisenstein_fun_aux 3) 0"
  using closed_lattice0
  by (intro higher_deriv_cmult[where A = "-Λ*"]) (auto intro!: holomorphic_intros)
also have "... = (- 1) ^ Suc n * fact (n + 2) * G (n + 3)"
  by (subst higher_deriv_eisenstein_fun_aux)
  (auto simp: algebra_simps pochhammer_rec pochhammer_fact)
finally show ?thesis
  using assms by (simp add: n_def)
qed

```

We now show that the Laurent series expansion of $\wp(z)$ at $z = 0$ has the form

$$z^{-2} + \sum_{n \geq 1} (n+1)G_{n+2}z^n .$$

We choose a different approach to prove this than Apostol: Apostol converts the sum in question into a double sum and then interchanges the order of summation, claiming the double sum to be absolutely convergent. Since we were unable to see why that sum should be absolutely convergent, we were unable to replicate his argument. In any case, arguing about absolute convergence of double sums is always messy.

Our approach instead simply uses the fact that *weierstrass_fun_aux* (the Weierstrass function with its double pole removed) is analytic at 0 and thus has a power series expansion that is valid within any ball around 0 that does not contain any lattice points.

The coefficients of this power series expansion can be determined simply by taking the n -th derivative of *weierstrass_fun_aux* at 0, which is easy to do.

Note that this series converges absolutely in this domain, since it is a power series, but we do not show this here.

```

definition fps_weierstrass :: "complex fps"
  where "fps_weierstrass = Abs_fps (λn. if n = 0 then 0 else of_nat (Suc
n) * G (n + 2))"

```

```

lemma weierstrass_fun_aux_fps_expansion: "weierstrass_fun_aux has_fps_expansion
fps_weierstrass"

```

```

proof -
  define c where "c = (λn. if n = 0 then 0 else of_nat (Suc n) * G (n
+ 2))"
  have "(λn. (deriv ^^ n) weierstrass_fun_aux 0 / fact n) = c"
    (is "?lhs = ?rhs")
  proof

```

```

    fix n :: nat
    show "?lhs n = ?rhs n" unfolding c_def
      by (cases "even n") (auto simp: higher_deriv_weierstrass_fun_aux_0
eisenstein_series_odd_eq_0)
    qed
    hence "fps_weierstrass = fps_expansion weierstrass_fun_aux 0"
      by (intro fps_ext) (auto simp: fps_expansion_def fps_weierstrass_def
c_def fun_eq_iff)
    also have "weierstrass_fun_aux has_fps_expansion ..."
      using closed_lattice0
      by (intro has_fps_expansion_fps_expansion[of "- $\Lambda^*$ "] holomorphic_intros)
    auto
    finally show ?thesis .
  qed

```

```

definition fls_weierstrass :: "complex fls"
  where "fls_weierstrass = fls_X_intpow (-2) + fps_to_fls fps_weierstrass"

```

```

lemma fls_subdegree_weierstrass: "fls_subdegree fls_weierstrass = -2"
  by (intro fls_subdegree_eqI) (auto simp: fls_weierstrass_def)

```

```

lemma fls_weierstrass_nz [simp]: "fls_weierstrass  $\neq$  0"
  using fls_subdegree_weierstrass by auto

```

The following corresponds to Theorem 1.11 in Apostol's book.

```

theorem fls_weierstrass_laurent_expansion [laurent_expansion_intros]:
  " $\wp$  has_laurent_expansion fls_weierstrass"

```

proof -

```

  have "( $\lambda z. z \text{ powi } (-2) + \text{weierstrass\_fun\_aux } z$ ) has_laurent_expansion
fls_weierstrass"

```

```

    unfolding fls_weierstrass_def

```

```

    by (intro laurent_expansion_intros has_laurent_expansion_fps[OF weierstrass_fun_aux_fps]

```

```

    also have "?this  $\longleftrightarrow$  ?thesis"

```

```

  proof (intro has_laurent_expansion_cong refl)

```

```

    have "eventually ( $\lambda z. z \in -\Lambda^* - \{0\}$ ) (at 0)"

```

```

      using closed_lattice0 by (intro eventually_at_in_open) auto

```

```

    thus " $\forall_F x$  in at 0.  $x \text{ powi } -2 + \text{weierstrass\_fun\_aux } x = \wp x$ "

```

```

      by eventually_elim

```

```

        (auto simp: weierstrass_fun_def power_int_minus field_simps lattice_lattice0)

```

qed

```

  finally show ?thesis .

```

qed

```

corollary fls_weierstrass_deriv_laurent_expansion [laurent_expansion_intros]:

```

```

  " $\wp'$  has_laurent_expansion fls_deriv fls_weierstrass"

```

```

  by (rule has_laurent_expansion_deriv'[where A = "- $\Lambda^*$ ", OF fls_weierstrass_laurent_expans

```

```

    (use closed_lattice0 in <auto intro!: derivative_eq_intros simp:

```

```

lattice_lattice0>))

```

```

lemma fls_nth_weierstrass:
  "fls_nth fls_weierstrass n =
    (if n = -2 then 1 else if n > 0 then of_int (n + 1) * G (nat n +
2) else 0)"
  unfolding fls_weierstrass_def fps_weierstrass_def by (auto simp: not_less)

```

9.3 Differential equations for \wp

Using our results on elliptic functions, we can prove the important result that \wp satisfies the ordinary differential equation

$$\wp'^2 = 4\wp^3 - 60G_4\wp - 140G_6 .$$

The proof works by simply subtracting the two sides and then looking at the Laurent series expansion, noting that the poles all cancel out. This means that what remains is an elliptic functions without poles and therefore constant.

The constant can then easily be determined, since it is the 0-th coefficient of said Laurent series.

This is Theorem 1.12 in Apostol's book.

```

theorem weierstrass_fun_ODE1:
  assumes "z ∉ Λ"
  shows "ϖ' z ^ 2 = 4 * ϖ z ^ 3 - 60 * G 4 * ϖ z - 140 * G 6"
proof -
  note [simp] = fls_subdegree_deriv fls_subdegree_weierstrass
  define f where "f = (λz. ϖ' z ^ 2 - 4 * ϖ z ^ 3 + 60 * G 4 * ϖ z)"
  interpret f: elliptic_function ω1 ω2 f
  unfolding f_def by (intro elliptic_function_intros)
  let ?P = fls_weierstrass
  define F :: "complex fls" where "F = fls_deriv ?P ^ 2 - 4 * ?P ^ 3
+ fls_const (60 * G 4) * ?P"

  define g where "g = (λz. if z ∈ Λ then -140 * G 6 else f z)"

  have unwind: "{m..n::int} = insert m {m+1..n}" if "m ≤ n" for m n
  using that by auto
  define fls_terms :: "complex fls ⇒ complex list"
  where "fls_terms = (λF. map (λk. fls_nth F k) [-6,-5,-4,-3,-2,-1,0])"
  have coeffs: "map (λk. fls_nth F k) [-6,-5,-4,-3,-2,-1,0] = [0, 0, 0,
0, 0, 0, - (140 * G 6)]"
  by (simp add: power2_eq_square power3_eq_cube unwind fls_terms_def
fls_times_nth(1)
fls_nth_weierstrass F_def eisenstein_series_odd_eq_0
flip: One_nat_def)

```

```

have F: "f has_laurent_expansion F"
  unfolding f_def F_def by (intro laurent_expansion_intros)

have "fls_subdegree F ≥ 0"
proof (cases "F = 0")
  case False
  thus ?thesis
  proof (rule fls_subdegree_geI)
    show "fls_nth F k = 0" if "k < 0" for k
    proof (cases "k < -6")
      case True
      thus ?thesis
      by (simp add: power2_eq_square power3_eq_cube fls_times_nth(1)
F_def)
    next
    case False
    with <k < 0> have "k ∈ {-6, -5, -4, -3, -2, -1}"
      by auto
    thus ?thesis using <k < 0>
      using coeffs by auto
    qed
  qed
qed auto

interpret g: complex_lattice_periodic ω1 ω2 g
  by standard (auto simp: g_def f.lattice_cong rel_def)

have "f holomorphic_on -Λ* - {0}"
  unfolding f_def by (intro holomorphic_intros) (auto simp: lattice_lattice0)
moreover have "open (-Λ*)"
  using closed_lattice0 by auto
moreover have "f -0→ fls_nth F 0"
  using F <fls_subdegree F ≥ 0> by (meson has_laurent_expansion_imp_tendsto_0)
hence "f -0→ -140 * G 6"
  using coeffs by simp
ultimately have "(λz. if z = 0 then -140 * G 6 else f z) holomorphic_on
-Λ*"
  unfolding g_def by (rule removable_singularity)
hence "(λz. if z = 0 then -140 * G 6 else f z) analytic_on {0}"
  using closed_lattice0 unfolding analytic_on_holomorphic by blast
also have "?this ↔ g analytic_on {0}"
  using closed_lattice0
  by (intro analytic_at_cong refl eventually_mono[OF eventually_nhds_in_open[of
"-Λ*"]])
  (auto simp: g_def f_def lattice_lattice0)
finally have "g analytic_on {0}" .

have "g analytic_on (-Λ ∪ (⋃ω∈Λ. {ω}))"
  unfolding analytic_on_Un analytic_on_UN

```

```

proof safe
  fix  $\omega$  :: complex
  assume  $\omega$ : " $\omega \in \Lambda$ "
  have " $g \circ (\lambda z. z - \omega)$  analytic_on  $\{\omega\}$ "
    using <g analytic_on  $\{0\}$ >
    by (intro analytic_on_compose) (auto intro!: analytic_intros)
  also have " $g \circ (\lambda z. z - \omega) = g$ "
    by (auto simp: fun_eq_iff rel_def uminus_in_lattice_iff  $\omega$  intro!:
g.lattice_cong)
  finally show " $g$  analytic_on  $\{\omega\}$ " .
next
  have " $f$  holomorphic_on  $(-\Lambda)$ "
    unfolding f_def by (auto intro!: holomorphic_intros)
  also have "?this  $\longleftrightarrow$   $g$  holomorphic_on  $(-\Lambda)$ "
    by (intro holomorphic_cong refl) (auto simp: g_def)
  finally show " $g$  analytic_on  $(-\Lambda)$ "
    using closed_lattice by (subst analytic_on_open) auto
qed
also have "... = UNIV"
  by blast
finally have g_ana: " $g$  analytic_on UNIV" .

interpret g: nicely_elliptic_function  $\omega_1$   $\omega_2$  g
  by standard (auto intro!: analytic_on_imp_nicely_meromorphic_on g_ana)
have "elliptic_order  $g = 0$ "
proof (subst g.elliptic_order_eq_0_iff_no_poles, rule allI)
  show " $\neg$ is_pole  $g$   $z$ " for  $z$ 
    by (rule analytic_at_imp_no_pole) (auto intro: analytic_on_subset[OF
g_ana])
qed
hence " $g$  constant_on UNIV"
  by (simp add: g.elliptic_order_eq_0_iff)

then obtain  $c$  where  $c$ : " $g$   $z = c$ " for  $z$ 
  unfolding constant_on_def by blast
from c[of 0] have " $c = -140 * G^6$ "
  by (simp add: g_def)
with c[of  $z$ ] and assms show ?thesis
  by (simp add: g_def f_def algebra_simps)
qed

```

The above ODE of the meromorphic function \wp can now easily be lifted to a formal ODE on the corresponding Laurent series.

```

lemma fls_weierstrass_ODE1:
  defines "P  $\equiv$  fls_weierstrass"
  shows "fls_deriv P ^ 2 = 4 * P ^ 3 - fls_const (60 * G^4) * P - fls_const
(140 * G^6)"
  (is "?lhs = ?rhs")
proof -

```

```

have ev: "eventually ( $\lambda z. z \in -\Lambda^* - \{0\}$ ) (at 0)"
  using closed_lattice0 by (intro eventually_at_in_open) auto
have "( $\lambda z. \wp' z ^ 2$ ) has_laurent_expansion ?lhs"
  unfolding P_def by (intro laurent_expansion_intros)
also have "?this  $\longleftrightarrow$  ( $\lambda z. 4 * \wp z ^ 3 - 60 * G 4 * \wp z - 140 * G 6$ )"
has_laurent_expansion ?lhs"
  by (intro has_laurent_expansion_cong refl eventually_mono[OF ev] weierstrass_fun_ODE1)
  (auto simp: lattice_lattice0)
finally have ... .
moreover have "( $\lambda z. 4 * \wp z ^ 3 - 60 * G 4 * \wp z - 140 * G 6$ ) has_laurent_expansion
?rhs"
  unfolding P_def by (intro laurent_expansion_intros)
ultimately show ?thesis
  using has_laurent_expansion_unique by blast
qed

```

```

lemma fls_weierstrass_ODE2:
  defines "P  $\equiv$  fls_weierstrass"
  shows "fls_deriv (fls_deriv P) = 6 * P ^ 2 - fls_const (30 * G 4)"
proof -
  define d where "d = fls_deriv P"
  have "fls_subdegree d = -3"
    using fls_subdegree_weierstrass unfolding d_def
    by (subst fls_subdegree_deriv) (auto simp: P_def)
  hence nz: "d  $\neq$  0" by auto

  have "2 * d * fls_deriv d = 2 * d * (6 * P2 - 30 * fls_const (G 4))"
    using arg_cong[OF fls_weierstrass_ODE1, of fls_deriv]
    unfolding P_def [symmetric] fls_const_mult_const [symmetric] d_def
    by (simp add: fls_deriv_power algebra_simps)
  then have "fls_deriv d = 6 * P2 - 30 * fls_const (G 4)"
    using nz by simp
  then show ?thesis unfolding d_def
    by (metis fls_const_mult_const fls_const_numeral)
qed

```

```

theorem weierstrass_fun_ODE2:
  assumes "z  $\notin$   $\Lambda$ "
  shows "deriv  $\wp' z = 6 * \wp z ^ 2 - 30 * G 4$ "
proof -
  define P where "P = fls_weierstrass"
  have "( $\lambda z. \text{deriv } \wp' z - 6 * \wp z ^ 2 + 30 * G 4$ ) has_laurent_expansion
(fl_s_deriv (fls_deriv P) - 6 * P ^ 2 + fls_const (30 * G 4))"
    unfolding P_def by (intro laurent_expansion_intros)
  also have "... = 0"
    by (simp add: fls_weierstrass_ODE2 P_def)
  finally have "deriv  $\wp' z - 6 * (\wp z)^2 + 30 * G 4 = 0$ "
  proof (rule has_laurent_expansion_0_analytic_continuation)
    show "( $\lambda z. \text{deriv } \wp' z - 6 * (\wp z)^2 + 30 * G 4$ ) holomorphic_on ((UNIV

```

```

- ( $\Lambda - \{0\}$ ) -  $\{0\}$ )"
  by (auto intro!: holomorphic_intros intro: closed_subset_lattice)
  show "open (UNIV - ( $\Lambda - \{0\}$ ))"
  by (intro open_Diff closed_subset_lattice) auto
  show "connected (UNIV - ( $\Lambda - \{0\}$ ))"
  by (intro sparse_imp_connected sparse_in_subset2[OF lattice_sparse])
auto
qed (use assms in auto)
thus ?thesis
  by (simp add: algebra_simps)
qed

```

```

lemma has_field_derivative_weierstrass_fun_deriv [derivative_intros]:
  assumes "(f has_field_derivative f') (at z within A)" "f z  $\notin$   $\Lambda$ "
  shows "(( $\lambda z. \wp' (f z)$ ) has_field_derivative ((6 *  $\wp (f z)$  ^ 2 - 30 * G 4) * f')) (at z within A)"
proof (rule DERIV_chain' [OF assms(1)])
  have "( $\wp'$  has_field_derivative (deriv  $\wp'$  (f z))) (at (f z))"
  by (rule analytic_derivI) (use assms(2) in <auto intro!: analytic_intros>)
  thus "( $\wp'$  has_field_derivative 6 * ( $\wp (f z)$ )2 - 30 * G 4) (at (f z))"
  using weierstrass_fun_ODE2[OF assms(2)] by simp
qed

```

9.4 Lattice invariants and a recurrence for the Eisenstein series

We will see that G_n can always be expressed in terms of G_4 and G_6 . These values, up to a constant factor, are referred to as g_2 and g_3 .

```

definition invariant_g2:: "complex" ("g2") where
  "g2  $\equiv$  60 * eisenstein_series 4"

```

```

definition invariant_g3:: "complex" ("g3") where
  "g3  $\equiv$  140 * eisenstein_series 6"

```

```

lemma weierstrass_fun_ODE1':
  assumes "z  $\notin$   $\Lambda$ "
  shows " $\wp' z$  ^ 2 = 4 *  $\wp z$  ^ 3 - g2 *  $\wp z$  - g3"
  using weierstrass_fun_ODE1[OF assms] by (simp add: invariant_g2_def invariant_g3_def)

```

This is the ODE obtained by differentiating the first ODE.

```

theorem weierstrass_fun_ODE2':
  assumes "z  $\notin$   $\Lambda$ "
  shows " $\text{deriv } \wp' z = 6 * \wp z$  ^ 2 - g2 / 2"
proof -
  define P where "P = fls_weierstrass"
  have "( $\lambda z. \text{deriv } \wp' z - 6 * \wp z$  ^ 2 + g2 / 2) has_laurent_expansion
    fls_deriv (fls_deriv P) - 6 * P ^ 2 + fls_const (g2 / 2)" (is "?f

```

```

has_laurent_expansion _")
  unfolding P_def by (intro laurent_expansion_intros)
  also have "... = 0"
  by (simp add: fls_weierstrass_ODE2 P_def invariant_g2_def)
  finally have "?f has_laurent_expansion 0" .
  hence "?f z = 0"
  proof (rule has_laurent_expansion_0_analytic_continuation)
    show "?f holomorphic_on UNIV -  $\Lambda^*$  - {0}"
    using closed_lattice0
    by (intro holomorphic_intros) (auto simp: lattice_lattice0)
    show "connected (UNIV -  $\Lambda^*$ )"
    by (intro connected_open_diff_countable countable_subset[OF _ countable_lattice])
      (auto simp: lattice_lattice0)
  qed (use assms closed_lattice0 in <auto simp: lattice_lattice0>)
  thus ?thesis by (simp add: algebra_simps)
qed

```

```

lemma half_period_weierstrass_fun_is_root:
  assumes " $\omega \in \Lambda$ " " $\omega / 2 \notin \Lambda$ "
  defines " $z \equiv \wp (\omega / 2)$ "
  shows " $4 * z^3 - g_2 * z - g_3 = 0$ "
proof -
  have " $\wp' (\omega / 2) = 0$ "
  using weierstrass_fun_deriv.lattice_cong[of " $-\omega/2$ " " $\omega/2$ "] < $\omega \in \Lambda$ >
  by (auto simp: rel_def uminus_in_lattice_iff)
  hence " $0 = \wp' (\omega / 2) \wedge^2$ "
  by simp
  also have "... =  $4 * \wp (\omega / 2) \wedge^3 - g_2 * \wp (\omega / 2) - g_3$ "
  using assms by (subst weierstrass_fun_ODE1') auto
  finally show ?thesis
  by (simp add: z_def)
qed

```

The discriminant of the depressed cubic polynomial $p(x) = cX^3 + aX + b$ is $-4a^3 - 27cb^2$. This is useful since it gives us an algebraic condition for whether p has distinct roots.

```

lemma (in -) depressed_cubic_discriminant:
  fixes a b :: "'a :: idom"
  assumes "[b, a, 0, c] = Polynomial.smult c ([:-x1, 1:] * [:-x2, 1:]
  * [:-x3, 1:])"
  shows " $c \wedge^3 * (x1 - x2)^2 * (x1 - x3)^2 * (x2 - x3)^2 = -4 * a \wedge^3 - 27 * c * b \wedge^2$ "
  using assms by (simp add: algebra_simps) Groebner_Basis.algebra

```

The numbers e_1, e_2, e_3 are all distinct and hence the discriminant $\Delta = g_2^3 - 27g_3^2$ does not vanish. This is the first part of Apostol's Theorem 1.14.

```

theorem distinct_e123: "distinct [e1, e2, e3]"
proof -
  define X where "X = { $\omega_1 / 2, \omega_2 / 2, (\omega_1 + \omega_2) / 2$ }"

```

```

have empty: "X ∩ Λ = {}"
  by (auto simp: X_def)
have roots: "∀x∈X. ϕ' x = 0"
  using weierstass_fun_deriv_half_root_eq_0 unfolding X_def by blast
have X_subset: "X ⊆ period_parallelogram 0"
  unfolding period_parallelogram_altdef of_ω12_coords_image_eq
  by (auto simp: X_def ω12_coords.add)

have *: "ϕ w1 ≠ ϕ w2" if w12: "{w1, w2} ⊆ X" and neq: "w1 ≠ w2" for
w1 w2
proof
  assume eq: "ϕ w1 = ϕ w2"
  have not_in_lattice [simp]: "w1 ∉ Λ" "w2 ∉ Λ"
    using empty that(1) by blast+

  define f where "f = (λz. ϕ z - ϕ w1)"

  have deriv_eq: "deriv f w = ϕ' w" if "w ∉ Λ" for w
    unfolding f_def using that
    by (auto intro!: derivative_eq_intros DERIV_imp_deriv)
  have "deriv f w1 = 0" "deriv f w2 = 0"
    using w12 roots not_in_lattice deriv_eq by (metis insert_subset)+
  moreover have [simp]: "f w1 = 0" "f w2 = 0"
    using eq by (simp_all add: f_def)

  have "f has_laurent_expansion fls_weierstrass - fls_const (ϕ w1)"
    unfolding f_def by (intro laurent_expansion_intros)
  moreover have "fls_subdegree (fls_weierstrass - fls_const (ϕ w1))
= -2"
    by (simp add: fls_subdegree_weierstrass fls_subdegree_diff_eq1)
  ultimately have [simp]: "is_pole f 0" "zorder f 0 = -2"
    by (auto dest: has_laurent_expansion_imp_is_pole_0 has_laurent_expansion_zorder_0)
  have [simp]: "¬f constant_on UNIV"
    using pole_imp_not_constant[OF <is_pole f 0>, of UNIV UNIV] by
auto

  have "¬(∀x∈-Λ. f x = 0)"
  proof
    assume *: "∀x∈-Λ. f x = 0"
    have "eventually (λx. x ∈ -Λ* - {0}) (at 0)"
      using closed_lattice0 by (intro eventually_at_in_open) auto
    hence "eventually (λx. f x = 0) (at 0)"
      by eventually_elim (use * in <auto simp: lattice_lattice0>)
    hence "f -0→ 0"
      by (simp add: tendsto_eventually)
    with <is_pole f 0> show False
      unfolding is_pole_def using not_tendsto_and_filterlim_at_infinity[of
"at 0" f 0] by auto
  qed

```

```

then obtain z0 where z0: "z0 ∉ Λ" "f z0 ≠ 0"
  by auto
then have nconst: "¬ f constant_on (UNIV-Λ)"
  by (metis (mono_tags, lifting) Diff_iff UNIV_I <f w1 = 0>
      constant_on_def not_in_lattice(1))

have zorder_ge: "zorder f w ≥ int 2" if "w ∈ {w1, w2}" for w
proof (rule zorder_geI)
  show "f analytic_on {w}" "f holomorphic_on UNIV - Λ"
    unfolding f_def using that w12 empty
    by (auto intro!: analytic_intros holomorphic_intros)
  show "connected (UNIV - Λ)"
    using countable_lattice by (intro connected_open_diff_countable)
auto
  show "z0 ∈ UNIV - Λ" "f z0 ≠ 0"
    using z0 by auto
next
  fix k :: nat
  assume "k < 2"
  hence "k = 0 ∨ k = 1"
    by auto
  thus "(deriv ^^ k) f w = 0"
    using that w12 empty deriv_eq[of w] roots by auto
qed (use closed_lattice that w12 empty in auto)

have not_pole: "¬ is_pole f w" if "w ∈ {w1, w2}" for w
proof -
  have "f holomorphic_on -Λ"
    unfolding f_def by (intro holomorphic_intros) auto
  moreover have "open (-Λ)" "w ∈ -Λ"
    using that empty w12 closed_lattice by auto
  ultimately show ?thesis
    using not_is_pole_holomorphic by blast
qed

have no_further_poles: "¬ is_pole f z" if "z ∈ period_parallelogram
0 - {0}" for z
proof -
  have "f holomorphic_on -Λ"
    unfolding f_def by (intro holomorphic_intros) auto
  moreover have "z ∉ Λ"
proof
  assume "z ∈ Λ"
  then obtain m n where z_eq: "z = of_ω12_coords (of_int m, of_int
n)"
    by (elim latticeE)
  from that have "m = 0" "n = 0"
    unfolding period_parallelogram_altdef of_ω12_coords_image_eq
    by (auto simp: z_eq)

```

```

    with that show False
      by (auto simp: z_eq)
qed
ultimately show "¬is_pole f z"
  using closed_lattice not_is_pole_holomorphic[of "Λ" z f] by
auto
qed

interpret f: elliptic_function ω1 ω2 f
  unfolding f_def by (intro elliptic_function_intros)

have [intro]: "isolated_zero f w" if "w ∈ {w1, w2}" for w
proof (subst f.isolated_zero_analytic_iff)
  show "f analytic_on {w}"
    unfolding f_def by (intro analytic_intros) (use that in auto)
  show "f w = 0"
    using that by auto
  have "elliptic_order f ≠ 0"
    using <is_pole f 0> f.elliptic_order_eq_0_iff_no_poles by blast
  thus "¬(∀z ∈ UNIV. f z = 0)"
    unfolding f.elliptic_order_eq_0_iff_const_cospars by metis
qed

have "4 = (∑ z ∈ {w1, w2}. 2 :: nat)"
  using neq by simp
also have "... ≤ (∑ z ∈ {w1, w2}. nat (zorder f z))"
  using zorder_ge[of w1] zorder_ge[of w2] by (intro sum_mono) auto
also have "{w1, w2} ⊆ {z ∈ period_parallelogram 0. isolated_zero
f z}"
  using w12 X_subset by auto
hence "(∑ z ∈ {w1, w2}. nat (zorder f z)) ≤ (∑ z ∈ ... . nat (zorder
f z))"
  using f.finite_zeros_in_parallelogram[of 0] by (intro sum_mono2)
auto
also have "... = elliptic_order f"
  by (rule f.zeros_eq_elliptic_order)
also have "elliptic_order f = (∑ z | z ∈ period_parallelogram 0 ∧
is_pole f z. nat (-zorder f z))"
  by (rule f.poles_eq_elliptic_order [symmetric])
also have "... ≤ (∑ z ∈ {0}. nat (-zorder f z))"
proof (rule sum_mono2)
  have "¬is_pole f w" if "w ∈ period_parallelogram 0 - {0}" for w
    using no_further_poles[OF that] is_pole_cong by blast
  then show "{z ∈ period_parallelogram 0. is_pole f z} ⊆ {0}"
    unfolding period_parallelogram_def by auto
qed simp_all
also have "... = 2" by simp
finally show False by simp
qed

```

```

show ?thesis
  by (simp add: number_e1_def number_e2_def number_e3_def; intro conjI
*) (auto simp: X_def)
qed

```

The above implies that the polynomial

$$4(X - e_1)(X - e_2)(X - e_3) = 4X^3 - g_2X - g_3$$

has three distinct roots and therefore its discriminant

$$\Delta = g_2^3 - 27g_3^2$$

is non-zero. This is the second part of Apostol's Theorem 1.14.

From now on, we will refer to Δ as the discriminant of our lattice Λ . We also introduce the related invariant $j = \frac{g_2^3}{\Delta}$.

```

definition discr :: complex where
  "discr =  $g_2^3 - 27 * g_3^2$ "

```

```

definition invariant_J :: complex where
  "invariant_J =  $g_2^3 / \text{discr}$ "

```

theorem

```

  fixes z:: "complex"
  defines "P  $\equiv [-g_3, -g_2, 0, 4:]$ "
  shows discr_nonzero_aux1: "P =  $4 * [:-e_1, 1:] * [:-e_2, 1:] * [:-e_3, 1:]$ "
  and discr_nonzero_aux2: " $4 * (\wp z)^3 - g_2 * (\wp z) - g_3 = 4 * (\wp z - e_1) * (\wp z - e_2) * (\wp z - e_3)$ "
  and discr_nonzero: "discr  $\neq 0$ "
  and discr_altdef: "discr =  $(4 * (e_1 - e_2) * (e_1 - e_3) * (e_2 - e_3))^2$ "
  and invariant_g3_conv_e123: " $g_3 = 4 * e_1 * e_2 * e_3$ "
  and invariant_g2_conv_e123: " $g_2 = -4 * (e_1 * e_2 + e_1 * e_3 + e_2 * e_3)$ "
  and sum_e123_0: " $e_1 + e_2 + e_3 = 0$ "

```

proof -

```

  have zeroI: "poly P ( $\wp (\omega / 2)$ ) = 0" if " $\omega \in \Lambda$ " " $\omega / 2 \notin \Lambda$ " for  $\omega$ 
    using half_period_weierstrass_fun_is_root[OF that]
    by (simp add: P_def power3_eq_cube algebra_simps)

  obtain xs where "length xs = 3" and "Polynomial.smult 4 ( $\prod x \leftarrow xs. [:-x, 1:]$ ) = P"
    using fundamental_theorem_algebra_factorized[of P]
    by (auto simp: P_def numeral_3_eq_3)
  hence P_eq: "P = Polynomial.smult 4 ( $\prod x \leftarrow xs. [:-x, 1:]$ )"
    by simp
  from <length xs = 3> obtain x1 x2 x3 where xs: "xs = [x1, x2, x3]"
    by (auto simp: numeral_3_eq_3 length_Suc_conv)

```

```

have poly_P_eq: "poly P x = 4 * (x - x1) * (x - x2) * (x - x3)" for
x
  by (simp add: algebra_simps P_eq xs)

have "∀x∈{e1, e2, e3}. poly P x = 0"
  by (auto simp: number_e1_def number_e2_def number_e3_def intro!: zeroI
intro: lattice_intros)
hence set_xs: "set xs = {e1, e2, e3}" and distinct: "distinct xs"
  using distinct_e123 by (auto simp: poly_P_eq xs)
have "P = Polynomial.smult 4 (∏ x∈set xs. [:-x, 1:])"
  using distinct_P_eq by (subst prod.distinct_set_conv_list) auto
thus P_eq': "P = 4 * [:-e1, 1:] * [:-e2, 1:] * [:-e3, 1:]"
  unfolding set_xs using distinct_e123 by (simp add: xs numeral_poly
algebra_simps)

have "g3 = -poly.coeff P 0"
  by (simp add: P_def)
also have "... = 4 * e1 * e2 * e3"
  by (simp add: P_eq' numeral_poly)
finally show "g3 = 4 * e1 * e2 * e3" .

have "g2 = -poly.coeff P 1"
  by (simp add: P_def)
also have "... = -4 * (e1 * e2 + e1 * e3 + e2 * e3)"
  by (simp add: P_eq' numeral_poly algebra_simps)
finally show "g2 = -4 * (e1 * e2 + e1 * e3 + e2 * e3)" .

have "0 = -poly.coeff P 2 / 4"
  by (simp add: P_def numeral_2_eq_2)
also have "... = e1 + e2 + e3"
  by (simp add: P_eq' numeral_2_eq_2 numeral_poly algebra_simps)
finally show "e1 + e2 + e3 = 0" ..

show "4 * (ϕ z)^3 - g2 * (ϕ z) - g3 = 4 * (ϕ z - e1) * (ϕ z - e2) *
(ϕ z - e3)"
  using arg_cong[OF P_eq', of "λP. poly P (ϕ z)"]
  by (simp add: P_def numeral_poly algebra_simps power3_eq_cube)

have "discr = (-4 * (-g2) ^ 3 - 27 * 4 * (-g3) ^ 2) / 4"
  unfolding discr_def by simp
also have "-4 * (-g2) ^ 3 - 27 * 4 * (-g3) ^ 2 = 4 ^ 3 * (e1 - e2)^2 *
(e1 - e3)^2 * (e2 - e3)^2"
  by (rule sym, rule depressed_cubic_discriminant, fold P_def) (simp
add: P_eq' numeral_poly)
also have "... / 4 = 4 ^ 2 * (e1 - e2)^2 * (e1 - e3)^2 * (e2 - e3)^2"
  by simp
finally show discr_eq: "discr = (4 * (e1 - e2) * (e1 - e3) * (e2 - e3))
^ 2"
  unfolding power_mult_distrib by simp

```

```

    show "discr ≠ 0"
      by (subst discr_eq) (use distinct_e123 in auto)
qed

corollary modulus_neq_0: "modulus ≠ 0" and modulus_neq_1: "modulus ≠
1"
  using distinct_e123 by (auto simp: modulus_def)

The  $J$  invariant can be expressed as a rational function of the modulus.

corollary invariant_J_conv_modulus:
  defines "x ≡ modulus * (1 - modulus)"
  shows "invariant_J = 4 / 27 * (1 - x) ^ 3 / x ^ 2"
proof -
  define l where "l = modulus"
  have l: "(e1 - e2) * l = e3 - e2"
    unfolding l_def modulus_def using distinct_e123 by (auto simp: field_simps)
  have "256 * (1 - l * (1 - l)) ^ 3 * discr = 1728 * invariant_g2 ^ 3
* (1 * (1 - l)) ^ 2"
    using l sum_e123_0 unfolding modulus_def discr_altdef invariant_g2_conv_e123

    by Groebner_Basis.algebra
  thus "invariant_J = 4 / 27 * (1 - x) ^ 3 / x ^ 2"
    using modulus_neq_0 modulus_neq_1 discr_nonzero
    by (simp add: divide_simps invariant_J_def x_def l_def mult_ac)
qed

end

lemma (in complex_lattice_swap) modulus_swap: "swap.modulus = 1 - modulus"
  using distinct_e123 by (auto simp: swap.modulus_def modulus_def field_simps)

context std_complex_lattice
begin

lemma eisenstein_series_norm_summable':
  "k ≥ 3 ⇒ (λ(m,n). norm (1 / (of_int m + of_int n * τ) ^ k)) summable_on
(-{(0,0)})"
  using eisenstein_series_norm_summable[of k]
    summable_on_reindex_bij_betw[OF bij_betw_lattice0', where f =
"λω. norm (1 / ω ^ k)"]
  by (auto simp: eisenstein_series_def map_prod_def of_ω12_coords_def
case_prod_unfold norm_divide norm_power)

lemma eisenstein_series_2_altdef:
  "eisenstein_series 2 = 2 * zeta 2 + (∑∞n∈-{0}. ∑∞m. 1 / (of_int
m + of_int n * τ) ^ 2)"
  by (simp add: eisenstein_series_def of_ω12_coords_def)

```

```

lemma eisenstein_series_altdef':
  "k ≥ 3 ⇒ eisenstein_series k = (∑∞ (m,n) ∈ -{(0,0)}. 1 / (of_int m
+ of_int n * τ) ^ k)"
  using infsum_reindex_bij_betw[OF bij_betw_lattice0', where f = "λω.
1 / ω ^ k"]
  by (auto simp: eisenstein_series_altdef map_prod_def of_ω12_coords_def
case_prod_unfold)

end

```

9.5 Fourier expansion

In this section we derive the Fourier expansion of the Eisenstein series, following Apostol's Theorem 1.18, but with some alterations. For example, we directly generalise the result in the spirit of Apostol's Exercise 1.11, and we make use of the existing formalisation of Lambert series.

We first define an auxiliary function

$$f_n(z) = \sum_{m \in \mathbb{Z}} (z + m)^{-n} = \frac{1}{(n-1)!} \psi^{(n-1)}(1+z) + \psi^{(n-1)}(1-z) + \frac{1}{z^n}$$

where $\psi^{(n)}$ denotes the Polygamma function. This is well-defined for $n \geq 2$ and $z \in \mathbb{C} \setminus \mathbb{Z}$.

We then prove the Fourier expansion

$$f_{n+1}(z) = \frac{(2i\pi)^{n+1}}{n!} \text{Li}_{-n}(q)$$

where $q = e^{2i\pi z}$ and Li_{-n} denotes the Polylogarithm function.

```

definition eisenstein_fourier_aux :: "nat ⇒ complex ⇒ complex" where
  "eisenstein_fourier_aux n z =
    (Polygamma (n-1) (1 + z) + Polygamma (n-1) (1 - z)) / fact (n - 1)
+ 1 / z ^ n"

```

```

lemma abs_summable_one_over_const_plus_nat_power:
  assumes "n ≥ 2"
  shows "summable (λk. norm (1 / (z + of_nat k :: complex) ^ n))"
proof (rule summable_comparison_test_ev)
  have "eventually (λk. real k > norm z) at_top"
    by real_asymp
  thus "eventually (λk. norm (norm (1 / (z + of_nat k) ^ n)) ≤ 1 / (real
k - norm z) ^ n) at_top"
  proof eventually_elim
    case (elim k)
    have "real k - norm z ≤ norm (z + of_nat k)"
      by (metis add.commute norm_diff_ineq norm_of_nat)

```

```

    hence "1 / norm (z + of_nat k) ^ n ≤ 1 / norm (real k - norm z) ^
n"
      using elim
      by (intro power_mono divide_left_mono Rings.mult_pos_pos zero_less_power)
auto
  thus ?case using elim
  by (simp add: norm_divide norm_power)
qed
next
  show "summable (λk. 1 / (real k - norm z) ^ n)"
  proof (rule summable_comparison_test_bigo)
    show "summable (λk. norm (1 / real k ^ n))"
      using inverse_power_summable[of n] assms
      by (simp add: norm_power norm_divide field_simps)
  next
    show "(λk. 1 / (real k - norm z) ^ n) ∈ O(λk. 1 / real k ^ n)"
      by real_asymp
  qed
qed

lemma abs_summable_one_over_const_minus_nat_power:
  assumes "n ≥ 2"
  shows "summable (λk. norm (1 / (z - of_nat k :: complex) ^ n))"
  proof -
    have "summable (λk. norm (1 / ((-z) + of_nat k :: complex) ^ n))"
      using assms by (rule abs_summable_one_over_const_plus_nat_power)
    also have "(λk. norm (1 / ((-z) + of_nat k :: complex) ^ n)) =
      (λk. norm (1 / (z - of_nat k :: complex) ^ n))"
      by (simp add: norm_divide norm_power norm_minus_commute)
    finally show ?thesis .
  qed

lemma has_sum_eisenstein_fourier_aux:
  assumes "n ≥ 2" and "even n" and "Im z > 0"
  shows "((λm. 1 / (z + of_int m) ^ n) has_sum eisenstein_fourier_aux
n z) UNIV"
  proof -
    define f where "f = (λk. 1 / (z + of_int k) ^ n)"

    from assms have "1 + z ≠ 0"
      by (subst add_eq_0_iff) auto
    have "(λk. 1 / (((1 + z) + of_nat k) ^ n)) sums (Polygamma (n-1) (1+z)
/ fact (n-1))"
      using assms Polygamma_LIMSEQ[of "1 + z" "n - 1"] <1 + z ≠ 0> by (simp
add: field_simps)
    moreover have "summable (λk. cmod (1 / (z + (Suc k)) ^ n))"
      by (subst summable_Suc_iff) (rule abs_summable_one_over_const_plus_nat_power,
fact)
    ultimately have "((λk. 1 / (z + of_nat (Suc k)) ^ n) has_sum (Polygamma

```

```

(n-1) (1+z) / fact (n-1))) UNIV"
  by (intro norm_summable_imp_has_sum) (simp_all add: algebra_simps)
  also have "?this  $\longleftrightarrow$  (f has_sum (Polygamma (n-1) (1+z) / fact (n-1)))
{1..}" unfolding f_def
  by (rule has_sum_reindex_bij_witness[of _ "λk. nat (k - 1)" "λk.
of_int (Suc k)"]) auto
  finally have sum1: "(f has_sum (Polygamma (n-1) (1+z) / fact (n-1)))
{1..}" .

  have "1 - z  $\neq$  0"
  using assms by auto
  have "(λk. 1 / (((1 - z) + of_nat k) ^ n)) sums (Polygamma (n-1) (1-z)
/ fact (n-1))"
  using assms Polygamma_LIMSEQ[of "1 - z" "n - 1"] <1 - z  $\neq$  0>
  by (simp add: field_simps)
  also have "(λk. ((1 - z) + of_nat k) ^ n) = (λk. (z - of_nat (Suc k))
^ n)"
  using assms by (subst even_power_diff_commute) (auto simp: algebra_simps)
  finally have "(λk. 1 / (z - of_nat (Suc k)) ^ n) sums (Polygamma (n-1)
(1-z) / fact (n-1))" .
  moreover have "summable (λk. cmod (1 / (z - (Suc k)) ^ n))"
  by (subst summable_Suc_iff) (rule abs_summable_one_over_const_minus_nat_power,
fact)
  ultimately have "((λk. 1 / (z - of_nat (Suc k)) ^ n) has_sum (Polygamma
(n-1) (1-z) / fact (n-1))) UNIV"
  by (intro norm_summable_imp_has_sum)
  also have "?this  $\longleftrightarrow$  (f has_sum (Polygamma (n-1) (1-z) / fact (n-1)))
{..-1}" unfolding f_def
  by (rule has_sum_reindex_bij_witness[of _ "λk. nat (-k-1)" "λk. -of_int
(Suc k)"])
  (auto simp: algebra_simps)
  finally have sum2: "(f has_sum (Polygamma (n-1) (1-z) / fact (n-1)))
{..-1}" .

  have "(f has_sum (Polygamma (n-1) (1+z) / fact (n-1)) + Polygamma (n-1)
(1-z) / fact (n-1))
  ({1..}  $\cup$  {..-1})"
  by (intro has_sum_Un_disjoint sum1 sum2) auto
  also have "({1..}  $\cup$  {..-1}) = -{0::int}"
  by auto
  finally have "(f has_sum ((Polygamma (n-1) (1+z) + Polygamma (n-1) (1-z))
/ fact (n-1)) (-{0}))"
  by (simp add: add_divide_distrib)
  hence "(f has_sum (f 0 + (Polygamma (n-1) (1+z) + Polygamma (n-1) (1-z))
/ fact (n-1)) (insert 0 (-{0}))"
  by (intro has_sum_insert) auto
  also have "insert 0 (-{0}) = (UNIV :: int set)"
  by auto
  finally show ?thesis

```

```

    by (simp add: eisenstein_fourier_aux_def f_def algebra_simps)
qed

lemma eisenstein_fourier_aux_expansion:
  assumes n: "odd n" and z: "Im z > 0"
  shows "eisenstein_fourier_aux (n + 1) z =
        (2 * i * pi) ^ Suc n / fact n * polylog (-int n) (to_q 1 z)"
proof -
  have eq0: "1 / z + cot_pfd z = -i * pi * (2 * polylog 0 (to_q 1 z) +
  1)"
  if z: "Im z > 0" for z :: complex
  proof -
    define x where "x = exp (2 * pi * i * z)"
    have *: "exp (2 * pi * i * z) = exp (pi * i * z) ^ 2"
      by (subst exp_double [symmetric]) (simp add: algebra_simps)
    have "norm x < 1"
      using z by (auto simp: x_def)
    hence "x ≠ 1"
      by auto

    have "1 / z + cot_pfd z = pi * cot (pi * z)"
      using z by (intro cot_pfd_formula_complex [symmetric]) (auto elim:
  Ints_cases)
    also have "pi * cos (pi * z) * (x - 1) = pi * i * sin (pi * z) * (x
  + 1)"
      using z *
      by (simp add: sin_exp_eq cos_exp_eq x_def exp_minus field_simps
  power2_eq_square
  del: div_mult_self3 div_mult_self4 div_mult_self2 div_mult_self1)
    hence "pi * cot (pi * z) = i * pi * (x + 1) / (x - 1)"
      unfolding cot_def using z by (auto simp: divide_simps sin_eq_0)
    also have "... = -i * pi * (-(x + 1) / (x - 1))"
      by (simp add: field_simps)
    also have "-(x + 1) / (x - 1) = 1 + 2 * x / (1 - x)"
      using <x ≠ 1> by (simp add: field_simps)
    also have "... = 2 * polylog 0 x + 1"
      using <norm x < 1> and <x ≠ 1> by (simp add: polylog_0_left field_simps)
    finally show ?thesis by (simp only: x_def to_q_def) simp
  qed

  define f :: "nat ⇒ complex ⇒ complex" where
    "f = (λn z. if n = 0 then 1 / z + cot_pfd z
      else (-1) ^ n * Polygamma n (1 - z) - Polygamma n (1 +
  z) +
      (-1) ^ n * fact n / z ^ (Suc n))"

  have f_odd_eq: "f n z = -fact n * eisenstein_fourier_aux (n+1) z" if
  "odd n" for n z
    using that by (auto simp add: f_def eisenstein_fourier_aux_def field_simps)

```

```

have DERIV_f: "(f n has_field_derivative f (Suc n) z) (at z)" if "z
∉ ℤ" for n z
proof (cases "n = 0")
  case [simp]: True
    have "((λz. 1 / z + cot_pfd z) has_field_derivative f 1 z) (at z)"
      unfolding f_def by (rule derivative_eq_intros refl | use that in
force)+
    thus ?thesis by (simp add: f_def)
  next
    case False
      have 1: "1 - z ∉ ℤ≤0" "1 + z ∉ ℤ≤0"
        using that by (metis Ints_1 Ints_diff add_diff_cancel_left' diff_diff_eq2
nonpos_Ints_Int)+
      have 2: "z ≠ 0"
        using that by auto

      have "((λz. (-1) ^ n * Polygamma n (1 - z) - Polygamma n (1 + z) +
(-1) ^ n * fact n / z ^ (Suc n))
        has_field_derivative (-1) ^ Suc n * Polygamma (Suc n) (1 -
z) -
          Polygamma (Suc n) (1 + z) + (-1) ^ Suc n * fact (Suc n) /
z ^ (Suc (Suc n))) (at z)"
        by (rule derivative_eq_intros refl | use that 1 2 in force)+
      thus ?thesis
        using that False by (simp add: f_def)
qed

define g :: "nat ⇒ complex ⇒ complex" where
  "g = (λn z. -pi * i * (2 * i * pi) ^ n * (2 * polylog (-n) (to_q 1
z) + (if n = 0 then 1 else 0)))"

have g_eq: "g n z = -((2 * i * pi) ^ Suc n) * polylog (-n) (to_q 1 z)"
if "n > 0" for n z
  using that unfolding g_def by simp

note [derivative_intros del] = DERIV_sum
have DERIV_g: "(g n has_field_derivative g (Suc n) z) (at z)" if z:
"Im z > 0" for z n
proof -
  have "norm (to_q 1 z) = exp (- (2 * pi * Im z))"
    by simp
  also have "... < exp 0"
    using that by (subst exp_less_cancel_iff) auto
  finally have "norm (to_q (Suc 0) z) ≠ 1"
    by auto
  moreover have "norm (1 :: complex) = 1"
    by simp
  ultimately have [simp]: "to_q (Suc 0) z ≠ 1"

```

```

    by metis
  show ?thesis unfolding g_def
    by (rule derivative_eq_intros refl | (use z in simp; fail))+
      (auto simp: algebra_simps minus_diff_commute)
qed

have eq: "f n z = g n z" if "Im z > 0" for z n
  using that
proof (induction n arbitrary: z)
  case (0 z)
  have "norm (to_q 1 z) < 1"
    unfolding to_q_def using 0 by simp
  hence "polylog 0 (to_q 1 z) = to_q 1 z / (1 - to_q 1 z)"
    by (subst polylog_0_left) auto
  thus ?case
    using eq0[of z] 0 by (auto simp: complex_is_Int_iff f_def g_def)
next
  case (Suc n z)
  have "(f n has_field_derivative f (Suc n) z) (at z)"
    by (rule DERIV_f) (use Suc.prem in <auto simp: complex_is_Int_iff>)
  also have "?this  $\longleftrightarrow$  (g n has_field_derivative f (Suc n) z) (at z)"
  proof (rule DERIV_cong_ev)
    have "eventually ( $\lambda z. z \in \{z. \text{Im } z > 0\}$ ) (nhds z)"
      using Suc.prem by (intro eventually_nhds_in_open) (auto simp:
open_halfspace_Im_gt)
    thus "eventually ( $\lambda z. f n z = g n z$ ) (nhds z)"
      by eventually_elim (use Suc.IH in auto)
  qed auto
  finally have "(g n has_field_derivative f (Suc n) z) (at z)" .
  moreover have "(g n has_field_derivative g (Suc n) z) (at z)"
    by (rule DERIV_g) fact
  ultimately show "f (Suc n) z = g (Suc n) z"
    using DERIV_unique by blast
qed

from z have "f n z = g n z"
  by (intro eq) auto
also have "f n z = -fact n * eisenstein_fourier_aux (n+1) z"
  using n by (subst f_odd_eq) auto
also have "g n z = - ((2 * i * pi) ^ Suc n) * polylog (-n) (to_q 1 z)"
  using n by (subst g_eq) (auto elim: oddE)
finally show ?thesis
  by (simp add: to_q_def field_simps)
qed

```

With this, we can now express the Fourier expansion of the Eisenstein series of the lattice $\Lambda(1, \tau)$ with $\text{Im}(\tau) > 0$ in terms of a Lambert series:

$$G_k = 2(\zeta(k) + \frac{(2i\pi)^k}{(k-1)!} L(n^{k-1}, q))$$

Here, as usual, $q = e^{2i\pi\tau}$ and

$$L(n^{k-1}, q) = \sum_{n \geq 1} n^{k-1} \frac{q^n}{1 - q^n} = \sum_{n \geq 1} \sigma_{k-1}(n) q^n$$

```

lemma (in std_complex_lattice) eisenstein_series_conv_lambert:
  assumes k: "k ≥ 2" "even k"
  defines "x ≡ to_q 1 τ"
  shows "eisenstein_series k =
    2 * (zeta k + (2 * i * pi) ^ k / fact (k - 1) * lambert (λn.
of_nat n ^ (k-1)) x)"
proof -
  have x: "norm x < 1"
    using Im_τ_pos by (simp add: x_def to_q_def)

  define g where "g = (λ(n,m). 1 / (of_int m + of_int n * τ) ^ k)"
  define C where "C = (2 * i * pi) ^ k / fact (k-1)"
  define S where "S = lambert (λn. of_nat n ^ (k-1)) x"

  have sum1: "(λm. g (int n, m)) has_sum C * polylog (1 - int k) (x ^
n)) UNIV" if n: "n > 0" for n
  proof -
    have "(λm. g (int n, m)) has_sum eisenstein_fourier_aux k (of_nat
n * τ) UNIV"
      using has_sum_eisenstein_fourier_aux[of k "of_nat n * τ"] n k Im_τ_pos
      by (simp add: g_def add_ac)
    also have "eisenstein_fourier_aux k (of_nat n * τ) =
      C * polylog (1 - int k) (to_q 1 (of_nat n * τ))"
      using eisenstein_fourier_aux_expansion[of "k-1" "of_nat n * τ"]
Im_τ_pos n k
      by (simp add: C_def)
    also have "to_q 1 (of_nat n * τ) = x ^ n"
      by (simp add: x_def to_q_power)
    finally show ?thesis .
  qed

  have "(λn. polylog (1 - int k) (x ^ n)) has_sum S {1..}"
    using lambert_power_int_has_sum_polylog_gen[of x 1 "int k - 1"] x
k
    by (simp add: power_int_def nat_diff_distrib S_def)
  hence "(λn. C * polylog (1 - int k) (x ^ n)) has_sum (C * S) {1..}"
    by (rule has_sum_cmult_right)
  also have "?this ↔ ((λn. (∑∞m. g (int n, m))) has_sum (C * S))
{1..}"
    by (rule has_sum_cong) (use sum1 in <auto simp: has_sum_iff>)
  also have "... ↔ ((λn. (∑∞m. g (n, m))) has_sum (C * S)) {1..}"
    by (rule has_sum_reindex_bij_witness[of _ nat int]) auto
  finally have sum2: "(λn. (∑∞m. g (n, m))) has_sum (C * S) {1..}"
  .

```

```

also have "?this  $\longleftrightarrow$  (( $\lambda n. (\sum_{\infty m. g (-n, m)})$ ) has_sum (C * S)) {..-1}"
  by (rule has_sum_reindex_bij_witness[of _ uminus uminus]) (auto simp:
g_def)
also have "( $\lambda n. \sum_{\infty m. g (-n, m)} = (\lambda n. \sum_{\infty m. g (n, m)}$ )"
proof
  fix n :: int
  show "( $\sum_{\infty m. g (-n, m)} = (\sum_{\infty m. g (n, m)}$ )"
    by (rule infsum_reindex_bij_witness[of _ uminus uminus])
      (use k in <auto simp: g_def even_power_diff_commute>)
qed
finally have "(( $\lambda n. \sum_{\infty m. g (n, m)}$ ) has_sum C * S) {..-1}" .
hence "(( $\lambda n. \sum_{\infty m. g (n, m)}$ ) has_sum (C * S + C * S)) ({..-1}  $\cup$  {1..})"
  by (intro has_sum_Un_disjoint sum2) auto
also have "C * S + C * S = 2 * C * S"
  by simp
also have "{..-1}  $\cup$  {1..} = -{0::int}"
  by auto
finally have sum3: "(( $\lambda n. \sum_{\infty m. g (n, m)}$ ) has_sum 2 * C * S) (-{0})"
.

have "eisenstein_series k = 2 * zeta (of_nat k) + ( $\sum_{\infty n \in -\{0\}. \sum_{\infty m. g (n, m)}$ )"
  using k by (simp add: eisenstein_series_def g_def of_omega12_coords_def)
also have "( $\sum_{\infty n \in -\{0\}. \sum_{\infty m. g (n, m)}$ ) = 2 * C * S"
  using sum3 by (simp add: has_sum_iff)
finally show ?thesis
  by (simp add: C_def S_def)
qed

```

9.6 Behaviour under lattice transformations

In this section, we will show how the Eisenstein series and related lattice properties behave under various lattice operations such as unimodular transformations and stretching.

In particular, we will see that the invariant j is actually invariant under unimodular transformations and stretching. This is Apostol's Theorem 1.16.

```

context complex_lattice_swap
begin

```

```

lemma eisenstein_series_swap [simp]:
  assumes "k  $\neq$  2"
  shows "swap.eisenstein_series k = eisenstein_series k"
proof (cases "k  $\geq$  3")
  case True
  thus ?thesis
    unfolding eisenstein_series_altdef[OF True] swap.eisenstein_series_altdef[OF
True] by simp

```

```

next
  case False
  with assms have "k = 0  $\vee$  k = 1"
  by auto
  thus ?thesis
  by auto
qed

lemma eisenstein_fun_aux_swap [simp]: "swap.eisenstein_fun_aux = eisenstein_fun_aux"
  unfolding eisenstein_fun_aux_def [abs_def] swap.eisenstein_fun_aux_def
  [abs_def] by (auto cong: if_cong)

lemma invariant_g2_swap [simp]: "swap.invariant_g2 = invariant_g2"
  and invariant_g3_swap [simp]: "swap.invariant_g3 = invariant_g3"
  unfolding invariant_g2_def [abs_def] swap.invariant_g2_def [abs_def]
  invariant_g3_def [abs_def] swap.invariant_g3_def [abs_def]
  by auto

lemma discr_swap [simp]: "swap.discr = discr"
  by (simp add: discr_def swap.discr_def)

lemma invariant_J_swap [simp]: "swap.invariant_J = invariant_J"
  by (simp add: invariant_J_def swap.invariant_J_def)

end

context complex_lattice_cnj
begin

lemma eisenstein_series_cnj [simp]: "cnj.eisenstein_series n = cnj (eisenstein_series
n)"
  unfolding eisenstein_series_def cnj.eisenstein_series_def
  by (simp flip: zeta_cnj infsum_cnj add: of_omega12_coords_def cnj.of_omega12_coords_def)

lemma invariant_g2_cnj [simp]: "cnj.invariant_g2 = cnj invariant_g2"
  and invariant_g3_cnj [simp]: "cnj.invariant_g3 = cnj invariant_g3"
  by (simp_all add: cnj.invariant_g2_def invariant_g2_def cnj.invariant_g3_def
invariant_g3_def)

lemma discr_cnj [simp]: "cnj.discr = cnj discr"
  by (simp add: discr_def cnj.discr_def)

lemma invariant_J_cnj [simp]: "cnj.invariant_J = cnj invariant_J"
  by (simp add: invariant_J_def cnj.invariant_J_def)

end

```

```

context complex_lattice_stretch
begin

lemma eisenstein_series_stretch:
  "stretched.eisenstein_series n = c powi (-n) * eisenstein_series n"
proof (cases "n ≥ 3")
  case True
  have "stretched.eisenstein_series n = (∑∞ x ∈ Λ*. c powi (-n) * (1 / x ^ n))"
  using infsum_reindex_bij_betw[OF bij_betw_stretch_lattice0, where
f = "λω. 1 / ω ^ n"] True
  unfolding stretched.eisenstein_series_altdef[OF True] by (simp add:
power_int_minus field_simps)
  also have "... = c powi (-n) * eisenstein_series n"
  using True by (subst infsum_cmult_right') (auto simp: eisenstein_series_altdef)
  finally show ?thesis .
next
  case False
  hence "n = 0 ∨ n = 1 ∨ n = 2"
  by auto
  thus ?thesis
  proof (elim disjE)
    assume [simp]: "n = 2"
    show "stretched.eisenstein_series n = c powi -int n * G n"
    unfolding eisenstein_series_def stretched.eisenstein_series_def
stretched_of_ω12_coords
    apply (simp add: ring_distrib flip: infsum_cmult_right')
    apply (simp add: power_int_minus field_simps)?
    done
  qed auto
qed

lemma invariant_g2_stretch [simp]: "stretched.invariant_g2 = invariant_g2
/ c ^ 4"
and invariant_g3_stretch [simp]: "stretched.invariant_g3 = invariant_g3
/ c ^ 6"
unfolding invariant_g2_def stretched.invariant_g2_def
invariant_g3_def stretched.invariant_g3_def eisenstein_series_stretch
by (simp_all add: power_int_minus field_simps)

lemma discr_stretch [simp]: "stretched.discr = discr / c ^ 12"
unfolding stretched.discr_def discr_def invariant_g2_stretch invariant_g3_stretch
by (simp add: field_simps stretch_nonzero)

lemma invariant_J_stretch [simp]: "stretched.invariant_J = invariant_J"
unfolding stretched.invariant_J_def invariant_J_def invariant_g2_stretch
discr_stretch
by (simp add: field_simps stretch_nonzero)

```

end

```
context unimodular_moebius_transform_lattice
begin
```

```
lemma eisenstein_series_transformed [simp]:
  assumes "k ≠ 2"
  shows "transformed.eisenstein_series k = eisenstein_series k"
proof (cases "k ≥ 3")
  case True
  thus ?thesis
    by (simp add: transformed.eisenstein_series_altdef eisenstein_series_altdef
transformed_lattice0_eq)
next
  case False
  with assms have "k = 0 ∨ k = 1"
  by auto
  thus ?thesis
    by auto
qed
```

```
lemma invariant_g2_transformed [simp]: "transformed.invariant_g2 = invariant_g2"
and invariant_g3_transformed [simp]: "transformed.invariant_g3 = invariant_g3"
  by ((intro ext)?, unfold invariant_g2_def invariant_g3_def number_e1_def
transformed.invariant_g2_def transformed.invariant_g3_def transformed_lattice0_eq
simp_all
```

```
lemma discr_transformed [simp]: "transformed.discr = discr"
  by (simp add: transformed.discr_def discr_def)
```

```
lemma invariant_J_transformed [simp]: "transformed.invariant_J = invariant_J"
  by (simp add: transformed.invariant_J_def invariant_J_def)
```

end

9.7 Recurrence relation

```
context complex_lattice
begin
```

Using our formal ODE from above, we find the following recurrence for G_n . By unfolding this repeatedly, we can write any G_n as a polynomial in G_4 and G_6 – or, equivalently, in g_2 and g_3 .

This is Theorem 1.13 in Apostol's book.

```
lemma eisenstein_series_recurrence_aux:
  defines "b ≡ λn. (2*n + 1) * (G (2*n + 2))"
  shows "b 1 = g2 / 20"
  and "b 2 = g3 / 28"
```

```

    and "\n. n ≥ 3 ⇒ (2 * of_nat n + 3) * (of_nat n - 2) * b n = 3
* (∑ i=1..n-2. b i * b (n - i - 1))"
proof -
  show "b 1 = g2 / 20" "b 2 = g3 / 28"
    by (simp_all add: b_def invariant_g2_def invariant_g3_def)
next
  fix n :: nat assume n: "n ≥ 3"

  define c where "c = fls_nth fls_weierstrass"
  have b_c: "b n = c (2 * n)" if "n > 0" for n
    using that by (simp add: c_def fls_nth_weierstrass nat_add_distrib
b_def nat_mult_distrib)

  have "(2 * of_nat n) * (2 * of_nat n - 1) * b n =
6 * fls_nth (fls_weierstrass2) (2 * int n - 2)"
    using arg_cong[OF fls_weierstrass_ODE2, of "\λF. fls_nth F (2 * (n
- 1))"] n
    by (simp add: algebra_simps of_nat_diff nat_mult_distrib b_c c_def)
  also have "fls_nth (fls_weierstrass2) (2 * int n - 2) =
(∑ i=-2..2 * int n. c i * c (2 * int n - 2 - i))"
    by (simp add: power2_eq_square fls_times_nth(2) fls_subdegree_weierstrass
flip: c_def)
  also have "... = (∑ i∈{-2..2 * int n}-{n. odd n}. c i * c (2 * int n
- 2 - i))"
    by (intro sum.mono_neutral_right)
    (auto simp: c_def fls_nth_weierstrass eisenstein_series_odd_eq_0
even_nat_iff)
  also have "... = (∑ i∈{-1, int n} ∪ {0..<int n}. c (2 * i) * c (2 *
(int n - i - 1)))"
    by (intro sum.reindex_bij_witness[of _ "\λk. 2 * k" "\λk. k div 2"])
(auto simp: algebra_simps)
  also have "... = 2 * b n + (∑ i=0..<int n. c (2 * i) * c (2 * (int n
- i - 1)))"
    using n by (subst sum.union_disjoint) (auto simp: c_def fls_nth_weierstrass
b_c)
  also have "(∑ i=0..<int n. c (2 * i) * c (2 * (int n - i - 1))) =
(∑ i=0..<n. c (2 * int i) * c (2 * (int n - int i - 1)))"
    by (intro sum.reindex_bij_witness[of _ int nat]) (auto simp: of_nat_diff)
  also have "... = (∑ i=1..n-2. c (2 * int i) * c (2 * (int n - int i
- 1)))"
    by (intro sum.mono_neutral_right) (auto simp: c_def fls_nth_weierstrass)
  also have "... = (∑ i=1..n-2. b i * b (n - i - 1))"
    using n by (intro sum.cong) (auto simp: b_c of_nat_diff algebra_simps)
  finally show "(2 * of_nat n + 3) * (of_nat n - 2) * b n
= 3 * (∑ i=1..n-2. b i * b (n - i - 1))"
    by Groebner_Basis.algebra
qed

theorem eisenstein_series_recurrence:

```

```

assumes "n ≥ 2"
shows "G (2*n+4) = 3 / of_nat ((2*n+5) * (n-1) * (2*n+3)) *
      (∑ i≤n-2. of_nat ((2*i+3) * (2*(n-i)-1)) * G (2*i+4) * G (2*(n-2-i)+4))"
proof -
  define c :: nat where "c = (2*(n+1)+3) * ((n+1)-2) * (2*(n+1)+1)"
  have c_altdef: "c = (2*n+5) * (n-1) * (2*n+3)"
    using assms unfolding c_def by (intro arg_cong2[of _ _ _ "(*)"])
auto
  define S where "S = (∑ i≤n-2. of_nat ((2*i+3) * (2*(n-i)-1)) * G (2*i+4)
* G (2*(n-2-i)+4))"
  have [simp]: "c ≠ 0"
    using assms unfolding c_def mult_eq_0_iff by auto
  have *: "n + 1 ≥ 3"
    using assms by linarith
  have "of_nat c * G (2 * (n+1) + 2) =
      3 * (∑ i=1..(n+1) - 2. (2 * i + 1) * (2 * ((n+1) - i - 1) +
1) * G (2 * i + 2) * G (2 * ((n+1) - i - 1) + 2))"
    using eisenstein_series_recurrence_aux(3)[OF *] assms unfolding c_def
    apply simp
    apply (simp add: algebra_simps)
    done
  also have "2 * (n+1) + 2 = 2 * n + 4"
    using assms by simp
  also have "(∑ i=1..(n+1) - 2. (2 * i + 1) * (2 * ((n+1) - i - 1) + 1)
* G (2 * i + 2) * G (2 * ((n+1) - i - 1) + 2)) =
      (∑ i≤(n+1) - 3. (2 * (i+1) + 1) * (2 * ((n+1) - (i+1) -
1) + 1) * G (2 * (i+1) + 2) * G (2 * ((n+1) - (i+1) - 1) + 2))"
    by (intro sum.reindex_bij_witness[of _ "λi. i+1" "λi. i-1"]) (use
assms in auto)
  also have "(n+1) - 3 = n - 2"
    using assms by linarith
  also have "(∑ i≤n-2. (2 * (i+1) + 1) * (2 * ((n+1) - (i+1) - 1) + 1)
* G (2 * (i+1) + 2) * G (2 * ((n+1) - (i+1) - 1) + 2)) = S"
    using assms unfolding S_def
    apply (intro sum.cong refl arg_cong2[of _ _ _ "(*)"] arg_cong[of
_ _ of_nat] arg_cong[of _ _ G])
    apply (auto simp: algebra_simps Suc_diff_Suc)
    done
  finally have "G (2 * n + 4) = 3 / of_nat c * S"
    by (simp add: field_simps)
  thus ?thesis
    unfolding S_def c_altdef .
qed

end

```

With this we can now write some code to compute representations of G_n in terms of G_4 and G_6 . Our code returns a bivariate polynomial with rational coefficients.

```

fun eisenstein_series_poly :: "nat  $\Rightarrow$  rat poly poly" where
  "eisenstein_series_poly n =
    (if n = 0 then [: [:0, 1:] :]
     else if n = 1 then [:0, 1:]
     else
      Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
        ( $\sum$  i<n-2. Polynomial.smult (of_nat ((2*i+3)*(2*(n-i)-1)))
          (eisenstein_series_poly i * eisenstein_series_poly (n-2-i))))"

lemmas [simp del] = eisenstein_series_poly.simps

lemma eisenstein_series_poly_0 [simp]: "eisenstein_series_poly 0 = [:
[:0, 1:] :]"
  and eisenstein_series_poly_1 [simp]: "eisenstein_series_poly (Suc 0)
= [:0, 1:]"
  and eisenstein_series_poly_rec:
    "n  $\geq$  2  $\implies$  eisenstein_series_poly n =
      Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
        ( $\sum$  i<n-2. Polynomial.smult (of_nat ((2*i+3)*(2*(n-i)-1)))
          (eisenstein_series_poly i * eisenstein_series_poly (n-2-i)))"
  by (subst eisenstein_series_poly.simps; simp; fail)+

lemma coeff_0_0_eisenstein_series_poly [simp]:
  "poly.coeff (poly.coeff (eisenstein_series_poly n) 0) 0 = 0"
  by (induction n rule: eisenstein_series_poly.induct; subst eisenstein_series_poly.simps)
    (auto simp: coeff_sum coeff_mult_0)

definition coeff2 where "coeff2 p m n = poly.coeff (poly.coeff p n) m"

The polynomial  $P(X, Y)$  that gives us  $G_{2n+4} = P(G_4, G_6)$  only has mono-
mials of the form  $X^i Y^j$  with  $4i + 6j = 2n + 4$ .

lemma support_eisenstein_series_poly:
  assumes "coeff2 (eisenstein_series_poly n) i j  $\neq$  0"
  shows "4 * i + 6 * j = 2 * n + 4"
  using assms
proof (induction n arbitrary: i j rule: less_induct)
  case (less n i j)
  interpret coeff2: group_add_hom " $\lambda p.$  coeff2 p i j"
    by standard (auto simp: coeff2_def)
  define P where "P  $\equiv$  eisenstein_series_poly"

  consider "n = 0" | "n = 1" | "n  $\geq$  2"
    by linarith
  thus ?case
  proof cases
    assume [simp]: "n = 0"
    thus ?thesis using less.prem
      by (cases i; cases j; auto simp: coeff2_def)

```

```

next
  assume [simp]: "n = 1"
  thus ?thesis using less.prem
    by (cases i; cases j; auto simp: coeff2_def of_bool_def split: if_splits)
next
  assume n: "n ≥ 2"
  have "coeff2 (P n) i j ≠ 0"
    using less.prem by (simp add: P_def)
  also have "coeff2 (P n) i j =
    (3 / ((2 * of_nat n + 5) * of_nat (n - 1) * (2 * of_nat n
+ 3))) *
    (∑ r ≤ n - 2. (2 * of_nat r + 3) * of_nat (2 * (n - r) - 1)
*
    coeff2 (P r * P (n - Suc (Suc r))) i j)"
    unfolding P_def
    by (subst eisenstein_series_poly_rec) (use n in <auto simp: coeff2_def
coeff_sum of_nat_poly>)
  finally have "(∑ r ≤ n - 2. (2 * of_nat r + 3) * of_nat (2 * (n - r)
- 1) *
    coeff2 (P r * P (n - Suc (Suc r))) i j) ≠ 0"
    by simp
  then obtain r where r: "r ∈ {..n - 2}" and
    "(2 * rat_of_nat r + 3) * rat_of_nat (2 * (n - r) - 1) *
    coeff2 (P r * P (n - Suc (Suc r))) i j ≠ 0"
    using sum.not_neutral_contains_not_neutral by blast
  hence "coeff2 (P r * P (n - Suc (Suc r))) i j ≠ 0"
    by simp
  also have "coeff2 (P r * P (n - Suc (Suc r))) i j =
    (∑ j' ≤ j. ∑ i' ≤ i. coeff2 (P r) i' j' * coeff2 (P (n
- Suc (Suc r))) (i - i') (j - j'))"
    by (simp add: coeff2_def coeff_mult coeff_sum)
  finally obtain i' j' where i': "i' ∈ {..i}" and j': "j' ∈ {..j}"
and
    "coeff2 (P r) i' j' * coeff2 (P (n - Suc (Suc r))) (i - i') (j -
j') ≠ 0"
    using sum.not_neutral_contains_not_neutral by meson
  hence nz: "coeff2 (P r) i' j' ≠ 0" "coeff2 (P (n - Suc (Suc r)))
(i - i') (j - j') ≠ 0"
    by auto

  have "4 * i' + 6 * j' = 2 * r + 4"
    by (rule less.IH) (use r i' j' n nz(1) in <auto simp: P_def>)
  moreover have "4 * (i - i') + 6 * (j - j') = 2 * (n - Suc (Suc r))
+ 4"
    by (rule less.IH) (use r i' j' n nz(2) in <auto simp: P_def>)
  ultimately have "(4 * i' + 6 * j') + (4 * (i - i') + 6 * (j - j'))
=
    (2 * r + 4) + (2 * (n - Suc (Suc r)) + 4)"
    by (rule arg_cong2)

```

```

    also have "(4 * i' + 6 * j') + (4 * (i - i') + 6 * (j - j')) = 4 *
i + 6 * j"
      using i' j' by (simp add: algebra_simps)
    also have "(2 * r + 4) + (2 * (n - Suc (Suc r)) + 4) = 2 * n + 4"
      using r n by (simp add: algebra_simps)
    finally show ?thesis .
  qed
qed

```

We now show that the polynomial also gives us the right answer.

```

context complex_lattice
begin

lemma eisenstein_series_poly_correct:
  "poly2 (map_poly2 of_rat (eisenstein_series_poly n)) (G 4) (G 6) = G
(2 * n + 4)"
proof (induction n rule: eisenstein_series_poly.induct)
  case (1 n)
  interpret map1: map_poly_comm_ring_hom "of_rat :: rat ⇒ complex"
    by standard auto
  interpret map2: map_poly_comm_ring_hom "map_poly (of_rat :: rat ⇒ complex)"
    by standard auto
  consider "n = 0" | "n = 1" | "n ≥ 2"
    by linarith
  thus ?case
  proof cases
    case 3
    define c where "c = 3 / complex_of_nat ((2 * n + 5) * (n - 1) * (2
* n + 3))"
    have "poly2 (map_poly2 of_rat (eisenstein_series_poly n)) (G 4) (G
6) =
      c * (∑ i≤n-2. (of_nat ((2 * i + 3) * (2 * (n - i) - 1)) *
G (2 * i + 4) * G (2 * (n - 2 - i) + 4)))"
      using 3 1
      apply (simp add: eisenstein_series_poly_rec map_poly2_def hom_distribs
c_def sum_distrib_left)
      apply (simp add: algebra_simps)?
      done
    also have "... = G (2 * n + 4)"
      unfolding c_def by (rule eisenstein_series_recurrence[OF <n ≥ 2>,
symmetric])
    finally show ?thesis .
  qed (auto simp: map_poly_pCons map_poly2_def)
qed

end

```

We employ memoisation for better performance:

```

definition eisenstein_polys_step :: "rat poly poly list ⇒ rat poly poly

```

```

list" where
  "eisenstein_polys_step ps =
    (let n = length ps
     in ps @ [Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
              (∑ i≤n-2. Polynomial.smult (of_nat ((2*i+3)*(2*(n-i)-1)))
              (ps ! i * ps ! (n-2-i)))]])"

definition eisenstein_series_polys :: "nat ⇒ rat poly poly list" where
  "eisenstein_series_polys n = (eisenstein_polys_step ^^ (n - 2)) [[:0, 1:] :], [[:0, 1:] :]"

lemma eisenstein_polys_step_correct:
  assumes n: "n ≥ 2" and ps_eq: "ps = map eisenstein_series_poly [0..<n]"
  shows "eisenstein_polys_step ps = map eisenstein_series_poly [0..<Suc n]"
proof (rule nth_equalityI)
  fix i assume "i < length (eisenstein_polys_step ps)"
  have length: "length ps = n"
  by (simp add: ps_eq)
  define p where "p = Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
    (∑ i≤n-2. Polynomial.smult (of_nat ((2*i+3)*(2*(n-i)-1)))
    (ps ! i * ps ! (n-2-i)))"
  have step: "eisenstein_polys_step ps = ps @ [p]"
  by (simp add: eisenstein_polys_step_def p_def length Let_def)
  have i: "i ≤ n"
  using <i < _> unfolding ps_eq length eisenstein_polys_step_def by
simp
  show "eisenstein_polys_step ps ! i = map eisenstein_series_poly [0..<Suc n] ! i"
  proof (cases "i = n")
    case False
    thus ?thesis
    using i unfolding step by (auto simp: ps_eq nth_append simp del:
upt_Suc)
  next
  case [simp]: True
  have "eisenstein_polys_step ps ! i = p"
  by (auto simp: step nth_append length)
  also have "p = eisenstein_series_poly n"
  unfolding eisenstein_series_poly_rec[OF n] p_def ps_eq
  by (intro arg_cong2[of _ _ _ Polynomial.smult] refl sum.cong)
(use n in auto)
  also have "... = map eisenstein_series_poly [0..<Suc n] ! i"
  by (simp del: upt_Suc)
  finally show ?thesis .
qed

```

```

qed (auto simp: eisenstein_polys_step_def ps_eq)

lemma eisenstein_series_polys_correct:
  "eisenstein_series_polys n = map eisenstein_series_poly [0..

```

```

lemma funpow_rec_right: "n > 0 ⇒ (f ^^ n) xs = (f ^^ (n-1)) (f xs)"
  by (cases n) (auto simp del: funpow.simps simp: funpow_Suc_right)

```

```

context complex_lattice
begin

```

```

lemma eisenstein_series_polys_correct':
  assumes "eisenstein_series_polys m = ps"
  shows "list_all (λi. G (2*i+4) = poly2 (map_poly2 of_rat (ps ! i)))
  (G 4) (G 6) [0..

```

We now compute the relations up to G_{20} for demonstration purposes. This

could in principle be turned into a proof method as well.

```

lemma eisenstein_series_relations:
  "G 8 = 3 / 7 * G 4 ^ 2" (is ?th8)
  "G 10 = 5 / 11 * G 4 * G 6" (is ?th10)
  "G 12 = 18 / 143 * G 4 ^ 3 + 25 / 143 * G 6 ^ 2" (is ?th12)
  "G 14 = 30 / 143 * G 4 ^ 2 * G 6" (is ?th14)
  "G 16 = 27225 / 668525 * G 4 ^ 4 + 300 / 2431 * G 4 * G 6 ^ 2" (is ?th16)
  "G 18 = 3915 / 46189 * G 4 ^ 3 * G 6 + 2750 / 92378 * G 6 ^ 3" (is ?th18)
  "G 20 = 54 / 4199 * G 4 ^ 5 + 36375 / 508079 * G 4 ^ 2 * G 6 ^ 2" (is
?th20)
proof -
  have eq: "eisenstein_series_polys 9 =
    [[:0, 1:]:], [[:0, [1:]:], [[:0, 0, 3 / 7:]:],
    [[:0, [0, 5 / 11:]:], [[:0, 0, 0, 18 / 143:]:], 0, [[:25
/ 143:]:],
    [[:0, [0, 0, 30 / 143:]:], [[:0, 0, 0, 0, 9 / 221:]:], 0,
[:0, 300 / 2431:]:],
    [[:0, [0, 0, 0, 3915 / 46189:]:], 0, [[:125 / 4199:]:],
    [[:0, 0, 0, 0, 0, 54 / 4199:]:], 0, [[:0, 0, 36375 / 508079:]:]]"
  by (simp add: eisenstein_series_polys_def eisenstein_polys_step_def
    funpow_rec_right numeral_poly smult_add_right smult_diff_right
flip: pCons_one)
  thus ?th8 ?th10 ?th12 ?th14 ?th16 ?th18 ?th20
    using eisenstein_series_polys_correct'[OF eq]
    by (simp_all add: upt_rec map_poly2_def of_rat_divide power_numeral_reduce
field_simps)
qed
end
end

```

10 Addition and duplication theorems for \wp

```

theory Weierstrass_Addition
  imports Eisenstein_Series
begin

```

In this section, we shall derive the addition theorem for \wp , and from it the duplication theorem. The addition theorem is:

$$\wp(w + z) = -\wp(w) - \wp(z) + \frac{1}{4} \left(\frac{\wp'(w) - \wp'(z)}{\wp(w) - \wp(z)} \right)^2$$

We first prove this with the additional assumptions that w and z are in “general position”, i.e. we have neither $w + 2z \in \Lambda$ nor $z + 2w \in \Lambda$.

After that, we will drop these unnecessary assumptions using analytic continuation. Our proof follows Lang’s presentation [2].

```

lemma pos_sum_eq_0_imp_empty:
  fixes f :: "'a ⇒ 'b :: ordered_comm_monoid_add"
  assumes "(∑ x∈A. f x) = 0" "∧x. x ∈ A ⇒ f x > 0" "finite A"
  shows "A = {}"
proof (rule ccontr)
  assume "A ≠ {}"
  hence "(∑ x∈A. f x) > 0"
    by (intro sum_pos) (use assms in auto)
  with assms(1) show False by simp
qed

context complex_lattice
begin

lemma weierstrass_fun_add_aux:
  assumes u12: "u1 ∉ Λ" "u2 ∉ Λ" "¬rel u1 u2" "¬rel u1 (-u2)"
  assumes general_position: "u1 + 2 * u2 ∉ Λ" "2 * u1 + u2 ∉ Λ"
  shows "ϕ (u1 + u2) = -ϕ u1 - ϕ u2 + ((ϕ' u1 - ϕ' u2) / (ϕ u1 - ϕ
u2))2 / 4"
  and "ϕ' (u1 + u2) = -ϕ' u1 + (ϕ u1 - ϕ (u1 + u2)) * (ϕ' u1 - ϕ'
u2) / (ϕ u1 - ϕ u2)"
proof -
  note [simp] = weierstrass_fun.eval_to_fund_parallelogram
    weierstrass_fun_deriv.eval_to_fund_parallelogram

We introduce the copies  $u'_1, u'_2$  of  $u_1$  and  $u_2$  in the fundamental parallelogram
for convenience.

  define u1' where "u1' = to_fund_parallelogram u1"
  define u2' where "u2' = to_fund_parallelogram u2"
  have [simp]: "u1' ≠ u2'" "u2' ≠ u1'"
    using u12 by (auto simp: u1'_def u2'_def rel_sym)

Let  $a$  and  $b$  be such that  $(\varphi(u_1), \varphi'(u_2))$  and  $(\varphi(u_2), \varphi'(u_1))$  lie on the line
 $ax + b = y$ , i.e. such that  $u_1$  and  $u_2$  are both solutions of the linear equation
 $\varphi'(u) = a\varphi(u) + b$ .

  define a where "a = (ϕ' u1 - ϕ' u2) / (ϕ u1 - ϕ u2)"
  define b where "b = ϕ' u1 - a * ϕ u1"
  have ab: "ϕ' u1 = a * ϕ u1 + b" "ϕ' u2 = a * ϕ u2 + b"
  proof -
    show "ϕ' u1 = a * ϕ u1 + b"
      using u12 by (auto simp: b_def)
    have "a * (ϕ u1 - ϕ u2) = (ϕ' u1 - ϕ' u2)"
      unfolding a_def using u12 by (simp add: weierstrass_fun_eq_iff)
    thus "ϕ' u2 = a * ϕ u2 + b"
      by (simp add: algebra_simps b_def)
  qed

We define the function  $f(z) = \varphi'(z) - (a\varphi(z) + b)$  and note that it has a
triple pole at the lattice points and no other poles and therefore order 3.

```

```

define f where "f = remove_sings (λz. ϕ' z - (a * ϕ z + b))"
interpret f: nicely_elliptic_function ω1 ω2 f
  unfolding f_def by (intro elliptic_function_intros)
note [simp] = f.zorder.eval_to_fund_parallelogram f.eval_to_fund_parallelogram
have f_eq: "f z = ϕ' z - (a * ϕ z + b)" if "z ∉ Λ" for z
  unfolding f_def by (rule remove_sings_at_analytic) (auto intro!: analytic_intros
simp: that)

have pole_f: "is_pole f z" "zorder f z = -3" if "z ∈ Λ" for z
proof -
  define F where "F = fls_deriv fls_weierstrass - (fls_const a * fls_weierstrass
+ fls_const b)"
  have F: "f has_laurent_expansion F" unfolding f_def F_def
    by (intro laurent_expansion_intros)

  have "fls_nth F n = 0" if "n < -3" for n
    unfolding F_def using that by (auto simp: fls_weierstrass_def)
  moreover have "fls_nth F (-3) ≠ 0"
    unfolding F_def by (auto simp: fls_weierstrass_def)
  ultimately have "fls_subdegree F = -3"
    using fls_subdegree_eqI by blast
  with F have "zorder f 0 = -3" "is_pole f 0"
    using has_laurent_expansion_imp_is_pole_0 has_laurent_expansion_zorder_0
by fastforce+
  thus "is_pole f z" "zorder f z = -3"
    using f.poles.lattice_cong[of z 0] f.zorder.lattice_cong[of z 0]
that
  by (simp_all add: rel_def)
qed

have is_pole_f_iff: "is_pole f z ↔ z ∈ Λ" for z
proof -
  have "f analytic_on -Λ"
    unfolding f_def by (intro analytic_intros remove_sings_analytic_on)
auto
  with pole_f[of z] show ?thesis
    by (meson ComplI analytic_at_imp_no_pole analytic_on_analytic_at)
qed

have analytic_at_f: "f analytic_on {z}" if "z ∉ Λ" for z
  unfolding f_def using that by (intro analytic_intros remove_sings_analytic_on)
auto

interpret f: nonconst_nicely_elliptic_function ω1 ω2 f
proof
  have "elliptic_order f ≠ 0"
    using pole_f[of 0] by (subst f.elliptic_order_eq_0_iff_no_poles)
auto
  thus "elliptic_order f > 0"

```

```

    by linarith
qed

have order_f: "elliptic_order f = 3"
proof -
  have "elliptic_order f = ( $\sum z \mid z \in \text{period\_parallelogram } 0 \wedge \text{is\_pole } f z. \text{nat } (- \text{zorder } f z)$ )"
    by (rule f.poles_eq_elliptic_order [of 0, symmetric])
  also have "... = ( $\sum z \in \{0::\text{complex}\}. 3$ )"
    proof (intro sum.cong)
      have "{z  $\in$  period_parallelogram 0. is_pole f z}  $\subseteq$  {0}"
        using fund_period_parallelogram_in_lattice_iff by (auto simp:
is_pole_f_iff)
      moreover have "{0}  $\subseteq$  {z  $\in$  period_parallelogram 0. is_pole f z}"
        using pole_f[of 0] by auto
      ultimately show "{z  $\in$  period_parallelogram 0. is_pole f z} = {0}"
        by blast
    qed (use pole_f in auto)
  finally show ?thesis
    by simp
qed

```

We now look at the zeros of f . We know that u_1 and u_2 are zeros.

```

define Z where "Z = {z. z  $\in$  period_parallelogram 0  $\wedge$  isolated_zero
f z}"
have "finite Z"
  unfolding Z_def by (rule f.finite_zeros_in_parallelogram)
have in_Z_iff: "z  $\in$  Z  $\longleftrightarrow$  z  $\in$  period_parallelogram 0 - {0}  $\wedge$  f z =
0" for z
  by (auto simp: f.isolated_zero_iff' Z_def is_pole_f_iff fund_period_parallelogram_in_la

have "{u1', u2'}  $\subseteq$  Z"
  using u12 ab by (auto simp: analytic_at_f is_pole_f_iff u1'_def u2'_def
f_eq in_Z_iff)
have zorder_pos: "zorder f z > 0" if "z  $\in$  Z" for z using that
  by (auto simp: Z_def analytic_at_f u1'_def u2'_def f.isolated_zero_iff'
is_pole_f_iff
  intro!: zorder_isolated_zero_pos)
have zorder_pos': "zorder f u1 > 0" "zorder f u2 > 0"
  using zorder_pos[of u1'] zorder_pos[of u2'] <{u1', u2'}  $\subseteq$  Z> by (auto
simp: u1'_def u2'_def)

```

We know that the sum of the multiplicities of the zeros must be 3. We already split the sum into the contributions of u_1 and u_2 and those of any remaining zeros.

```

have sum_zorder_eq:
  "nat (zorder f u1) + nat (zorder f u2) + ( $\sum z \in Z - \{u1', u2'\}. \text{nat } (\text{zorder } f z)$ ) = 3"
proof -

```

```

have "elliptic_order f = ( $\sum z \in Z. \text{nat } (\text{zorder } f \ z)$ )"
  unfolding Z_def by (rule f.zeros_eq_elliptic_order[of 0, symmetric])
also have "Z = {u1', u2'}  $\cup$  (Z - {u1', u2'})"
  using <{u1', u2'}  $\subseteq$  Z> by auto
also have "( $\sum z \in \dots. \text{nat } (\text{zorder } f \ z)$ ) =
  nat (zorder f u1) + nat (zorder f u2) + ( $\sum z \in Z - \{u1', u2'\}.
\text{nat } (\text{zorder } f \ z)$ )"
  by (subst sum.union_disjoint)
  (use <finite Z> u12 zorder_pos' <{u1', u2'}  $\subseteq$  Z> in <auto simp:
u1'_def u2'_def>)
  finally show ?thesis
  using order_f by simp
qed

```

We also know that the sum of the zeros and poles, weighted by their multiplicity, is a lattice point. Since the pole is at the origin, it does not contribute anything to the sum.

```

have sum_zeros_in_lattice:
  "of_int (zorder f u1) * u1 + of_int (zorder f u2) * u2 +
  ( $\sum z \in Z - \{u1', u2'\}. \text{of\_int } (\text{zorder } f \ z) * z$ )  $\in$   $\Lambda$ "
proof -
  have "( $\sum z \mid z \in \text{period\_parallelogram } 0 \wedge (\text{isolated\_zero } f \ z \vee \text{is\_pole }
f \ z).$ 
  of_int (zorder f z) * z)  $\in$   $\Lambda$ "
  by (rule f.sum_zeros_poles_in_lattice[of 0])
  also have "{z. z  $\in$  period\_parallelogram 0  $\wedge$  (isolated\_zero f z  $\vee$ 
is\_pole f z)} = insert 0 Z"
  using pole_f by (auto simp: Z_def fund\_period\_parallelogram\_in\_lattice\_iff
is\_pole\_f\_iff)
  also have "( $\sum z \in \text{insert } 0 \ Z. \text{of\_int } (\text{zorder } f \ z) * z$ ) = ( $\sum z \in Z. \text{of\_int }
(\text{zorder } f \ z) * z$ )"
  by (rule sum.mono\_neutral\_right) (use <finite Z> in auto)
  also have "Z = {u1', u2'}  $\cup$  (Z - {u1', u2'})"
  using <{u1', u2'}  $\subseteq$  Z> by auto
  also have "( $\sum z \in \dots. \text{of\_int } (\text{zorder } f \ z) * z$ ) =
  of_int (zorder f u1) * u1' + of_int (zorder f u2) * u2'
+
  ( $\sum z \in Z - \{u1', u2'\}. \text{of\_int } (\text{zorder } f \ z) * z$ )"
  by (subst sum.union_disjoint)
  (use <finite Z> u12 zorder_pos' <{u1', u2'}  $\subseteq$  Z> in <auto simp:
u1'_def u2'_def>)
  also have "rel (of_int (zorder f u1) * u1' + of_int (zorder f u2)
* u2' +
  ( $\sum z \in Z - \{u1', u2'\}. \text{of\_int } (\text{zorder } f \ z) * z$ ))
(of_int (zorder f u1) * u1 + of_int (zorder f u2) *
u2 +
  ( $\sum z \in Z - \{u1', u2'\}. \text{of\_int } (\text{zorder } f \ z) * z$ ))"
  unfolding u1'_def u2'_def by (intro lattice_intros) auto
  finally show ?thesis .

```

qed

We now show that the zeros u_1 and u_2 must be simple. If they were not simple, they would be the only zeros and consequently we would have either $2u_1 + u_2$ or $u_1 + 2u_2$, which contradicts our assumption that u_1 and u_2 are in general position.

```

have [simp]: "zorder f u1 = 1"
proof (rule ccontr)
  assume "zorder f u1 ≠ 1"
  hence [simp]: "zorder f u1 = 2" "zorder f u2 = 1"
    using sum_zorder_eq zorder_pos' <{u1', u2'} ⊆ Z>
    by (auto simp add: u1'_def u2'_def)
  have "(∑ z ∈ Z - {u1', u2'}. nat (zorder f z)) = 0"
    using sum_zorder_eq by simp
  hence *: "Z - {u1', u2'} = {}"
    by (rule pos_sum_eq_0_imp_empty) (use <finite Z> in <auto intro:
zorder_pos>)
  have "2 * u1 + u2 ∈ Λ"
    using sum_zeros_in_lattice unfolding * by simp
  with general_position show False by simp
qed

```

```

have [simp]: "zorder f u2 = 1"
proof (rule ccontr)
  assume "zorder f u2 ≠ 1"
  hence [simp]: "zorder f u1 = 1" "zorder f u2 = 2"
    using sum_zorder_eq zorder_pos' <{u1', u2'} ⊆ Z>
    by (auto simp add: u1'_def u2'_def)
  have "(∑ z ∈ Z - {u1', u2'}. nat (zorder f z)) = 0"
    using sum_zorder_eq by simp
  hence *: "Z - {u1', u2'} = {}"
    by (rule pos_sum_eq_0_imp_empty) (use <finite Z> in <auto intro:
zorder_pos>)
  have "u1 + 2 * u2 ∈ Λ"
    using sum_zeros_in_lattice unfolding * by simp
  with general_position show False by simp
qed

```

Thus we can conclude that there must be a third root u_3 . It is simple, and there are no further roots.

```

obtain u3 where u3: "u3 ∈ Z - {u1', u2'}" "zorder f u3 = 1" and Z_eq:
"Z = {u1', u2', u3}"
proof -
  have "(∑ z ∈ Z - {u1', u2'}. nat (zorder f z)) = 1"
    using sum_zorder_eq by simp
  then obtain u3 where u3: "Z - {u1', u2'} = {u3}" "nat (zorder f u3)
= 1"
  proof (rule sum_nat_eq_1E)
    fix u assume u: "u ∈ Z - {u1', u2'}"

```

```

    have "f analytic_on {u}"
      unfolding f_def
      by (intro analytic_intros remove_sings_analytic_on)
        (use u in <auto simp: in_Z_iff fund_period_parallelogram_in_lattice_iff>)
    thus "nat (zorder f u) > 0" using u
      by (auto intro!: zorder_isolated_zero_pos simp: Z_def)
  qed auto
  show ?thesis
  proof (rule that[of u3])
    have "u1' ∈ Z" "u2' ∈ Z"
      using u12 ab by (auto simp: u1'_def u2'_def in_Z_iff f.eval_to_fund_parallelogram
f_eq)
    thus "Z = {u1', u2', u3}"
      using u3(1) by blast
    qed (use u3 in auto)
  qed
  have "u3 ∉ Λ"
    using u3(1) by (auto simp: in_Z_iff fund_period_parallelogram_in_lattice_iff)

```

Since the zeros sum to a lattice point, we have $u_3 \sim -(u_1 + u_2)$.

```

    have rel_u3: "rel u3 (-(u1 + u2))"
      using sum_zeros_in_lattice u3 unfolding Z_eq
      by (simp add: u1'_def u2'_def insert_Diff_if rel_def algebra_simps
split: if_splits)

```

By definition of f , the fact that u_3 is a zero means that $(\wp(u_3), \wp'(u_3))$ also lies on the line $ax + b = 0$.

```

    have "f u3 = 0"
      using u3 f.isolated_zero_iff[of u3] by (auto simp: Z_def)
    hence u3_ab: "ϕ' u3 = a * ϕ u3 + b"
      using <u3 ∉ Λ> by (subst (asm) f_eq) auto

```

From our assumptions, it also follows that \wp takes on a different value for each of u_1, u_2, u_3 .

```

    have inj: "inj_on ϕ {u1', u2', u3}"
    proof -
      have "-(u1 + u2) ∉ Λ"
        using <u3 ∉ Λ> rel_lattice_trans_right rel_u3 by blast
      have "ϕ u1' ≠ ϕ u2'"
        using u12 by (auto simp: u1'_def u2'_def weierstrass_fun_eq_iff)
      moreover have "ϕ u1' ≠ ϕ u3"
    proof
      assume "ϕ u1' = ϕ u3"
      hence "ϕ u1 = ϕ (-(u1 + u2))"
        using weierstrass_fun.lattice_cong[OF rel_u3] by (auto simp: u1'_def)
      thus False
        using u12 general_position <-(u1 + u2) ∉ Λ>
        by (auto simp: weierstrass_fun_eq_iff rel_def uminus_in_lattice_iff)
    qed

```

```

qed
moreover have "ϕ u2' ≠ ϕ u3"
proof
  assume "ϕ u2' = ϕ u3"
  hence "ϕ u2 = ϕ (-(u1 + u2))"
    using weierstrass_fun.lattice_cong[OF rel_u3] by (auto simp: u2'_def)
  thus False
    using u12 general_position <-(u1 + u2) ∉ Λ>
    by (auto simp: weierstrass_fun_eq_iff rel_def uminus_in_lattice_iff
add_ac)
qed
ultimately show ?thesis
  by auto
qed

```

We now define the polynomial $P(X) = 4X^3 - g_2X - g_3 - (aX - b)^2$. Combining the fact that u_1, u_2, u_3 are all solutions of $\varphi'(u) = a\varphi(u) + b$ with the ODE satisfied by φ , we know that $\varphi(u_1), \varphi(u_2), \varphi(u_3)$ are roots of P . Since we also showed that the three are distinct, P has no other roots.

```

define P where "P = [:-g3, -g2, 0, 4:] - [b, a:] ^ 2"
have P_roots: "poly P (ϕ u) = 0" if "u ∉ Λ" "ϕ' u = a * ϕ u + b" for
u
proof -
  have "poly P (ϕ u) = 4 * ϕ u ^ 3 - g2 * ϕ u - g3 - (a * ϕ u + b)
^ 2"
    by (simp add: P_def algebra_simps power3_eq_cube)
  also have "4 * ϕ u ^ 3 - g2 * ϕ u - g3 = ϕ' u ^ 2"
    using that weierstrass_fun_ODE1'[of u] by simp
  also have "a * ϕ u + b = ϕ' u"
    using that by (simp add: algebra_simps)
  finally show ?thesis
    by simp
qed

```

Consequently, we can write P in the form $P(X) = 4(X - \varphi(u_1))(X - \varphi(u_2))(X - \varphi(u_3))$.

```

define Q where "Q = 4 * (∏ u∈{u1',u2',u3}. [:-ϕ u, 1:])"
have P_Q: "P = Q"
proof (rule poly_eqI_degree_lead_coeff)
  have "poly.coeff P 3 = 4"
    by (simp add: P_def power2_eq_square eval_nat_numeral)
  moreover have "lead_coeff Q = 4"
    by (simp add: lead_coeff_mult lead_coeff_prod numeral_poly Q_def)
  moreover have "degree Q = 3"
    using u3 u12 by (auto simp: degree_mult_eq degree_prod_eq_sum_degree
card_insert_if Q_def)
  ultimately show "poly.coeff P 3 = poly.coeff Q 3"
    by simp

```

```

next
  show "degree P ≤ 3"
    by (rule degree_le) (auto simp: P_def eval_nat_numeral Suc_less_eq2
coeff_pCons')
next
  show "degree Q ≤ 3"
    by (auto simp: degree_mult_eq degree_prod_eq_sum_degree card_insert_if
Q_def)
next
  show "3 ≤ card (⊖ ' {u1', u2', u3})"
  proof -
    have "3 ≤ card {u1', u2', u3}"
      using u3 by (auto simp: card_insert_if)
    also have "... = card (⊖ ' {u1', u2', u3})"
      using inj by (rule card_image [symmetric])
    finally show ?thesis .
  qed
next
  show "poly P z = poly Q z"
    if "z ∈ ⊖ ' {u1', u2', u3}" for z
  proof -
    from that obtain u where [simp]: "z = ⊖ u" and u: "u ∈ {u1',
u2', u3}"
      by auto
    have "poly P (⊖ u) = 0"
      by (rule P_roots) (use u u3 u3_ab u12 <u3 ∉ Λ> ab in <auto simp:
u1'_def u2'_def>)
    moreover have "poly Q (⊖ u) = 0"
      using u by (auto simp: poly_prod Q_def)
    ultimately show ?thesis by simp
  qed
qed

```

All that remains now is to compare the coefficient of X^2 in these polynomials for X^2 and simplify.

```

have "⊖ (u1 + u2) = ⊖ (-(u1 + u2))"
  by (simp only: weierstrass_fun.even)
also have "... = ⊖ u3"
  by (rule weierstrass_fun.lattice_cong) (use rel_u3 in <auto simp:
rel_sym>)
also have "... = -⊖ u1' - ⊖ u2' + a ^ 2 / 4"
proof -
  have "poly.coeff P 2 = poly.coeff Q 2"
    by (simp only: P_Q)
  also have "poly.coeff P 2 = -a^2"
    by (simp add: P_def power2_eq_square eval_nat_numeral)
  also have "poly.coeff Q 2 = -4 * (⊖ u1' + ⊖ u2' + ⊖ u3)"
    using u3 by (auto simp: Q_def eval_nat_numeral numeral_poly)
  finally show ?thesis

```

```

    by (simp add: field_simps)
  qed
  finally show " $\wp (u1 + u2) = -\wp u1 - \wp u2 + ((\wp' u1 - \wp' u2) / (\wp u1 - \wp u2))^2 / 4$ "
    by (simp add: u1'_def u2'_def a_def)

  have " $\wp' (u1 + u2) = -\wp' (-(u1 + u2))$ "
    by (metis add.inverse_inverse weierstrass_fun_deriv_minus)
  also have " $\wp' (-(u1 + u2)) = \wp' u3$ "
    by (rule weierstrass_fun_deriv.lattice_cong) (use rel_u3 in <auto simp: rel_sym>)
  also have "... = a *  $\wp u3 + b$ "
    by (fact u3_ab)
  also have " $\wp u3 = \wp (-(u1 + u2))$ "
    by (rule weierstrass_fun.lattice_cong) (use rel_u3 in <auto simp: rel_sym>)
  also have "... =  $\wp (u1 + u2)$ "
    by (simp only: weierstrass_fun.even)
  also have " $a * \wp (u1 + u2) + b = \wp' u1 + (\wp (u1 + u2) - \wp u1) * (\wp' u1 - \wp' u2) / (\wp u1 - \wp u2)$ "
    by (simp add: a_def b_def algebra_simps add_divide_distrib diff_divide_distrib ring_distrib)
  finally show " $\wp' (u1 + u2) = -\wp' u1 + (\wp u1 - \wp (u1 + u2)) * (\wp' u1 - \wp' u2) / (\wp u1 - \wp u2)$ "
    by (simp add: divide_simps) (auto simp: algebra_simps)?
  qed

```

We now use analytic continuation to get rid of the “general position” assumption.

For this purpose, we regard u_1 as fixed and view the left-hand side and the right-hand side as a function of u_2 . The set of values u_2 that we have to exclude is sparse, so analytic continuation works.

```

theorem weierstrass_fun_add:
  assumes u12: " $u1 \notin \Lambda$ " " $u2 \notin \Lambda$ " " $\neg \text{rel } u1 \ u2$ " " $\neg \text{rel } u1 \ (-u2)$ "
  shows " $\wp (u1 + u2) = -\wp u1 - \wp u2 + ((\wp' u1 - \wp' u2) / (\wp u1 - \wp u2))^2 / 4$ "
    (is "?lhs u2 = ?rhs u2")
proof -
  define A where "A =  $-(\Lambda \cup \{z. \text{rel } u1 \ z\} \cup \{z. \text{rel } u1 \ (-z)\})$ "
  define B where "B = A -  $\{z. u1 + 2 * z \in \Lambda\} - \{z. 2 * u1 + z \in \Lambda\}$ "

  have A_altdef: "A = UNIV -  $(\Lambda \cup ((+) (-u1) -' \Lambda) \cup ((+) u1 -' \Lambda))$ "
    by (auto simp: A_def rel_def add_ac diff_in_lattice_commute)
  have B_altdef: "B = A -  $(\lambda z. 2 * z + u1) -' \Lambda - ((+) (2*u1)) -' \Lambda$ "
    unfolding B_def A_altdef by (auto simp: A_def add_ac)

  show ?thesis
proof (rule analytic_continuation_open[where f = ?lhs and g = ?rhs])

```

Our set B can be written as the complex plane minus some shifted and scaled copies of the lattice, i.e. an uncountable set minus a countable set. Therefore, B is clearly non-empty.

```

show "B ≠ {}"
proof -
  have "B = UNIV - (Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ ∪ (λz. 2 * z
+ u1) -' Λ ∪ (+) (-2*u1) ' Λ)"
    unfolding A_altdef B_altdef unfolding image_plus_conv_vimage_plus
by auto
  also have "(λz. 2 * z + u1) -' Λ = (λz. (z - u1) / 2) ' Λ"
  proof safe
    fix z assume "2 * z + u1 ∈ Λ"
    thus "z ∈ (λz. (z - u1) / 2) ' Λ"
      by (intro image_eqI[of _ _ "u1 + 2 * z"]) (auto simp: algebra_simps)
  qed auto
  finally have "B = UNIV - (Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ ∪
(λz. (z - u1) / 2) ' Λ ∪ (+) (-2*u1)
' Λ)" .
  also have "uncountable ..."
    by (intro uncountable_minus_countable countable_Un countable_lattice
countable_image uncountable_UNIV_complex)
  finally have "uncountable B" .
  thus "B ≠ {}"
    by auto
qed
next

```

Similarly, A can be written as the complex plane minus some shifted copies of the lattice, i.e. the complement of a sparse set. Clearly, what remains is still connected.

```

show "connected A"
proof -
  have "(Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ) sparse_in UNIV"
    unfolding A_altdef by (intro sparse_in_union' sparse_in_translate_UNIV
lattice_sparse)
  also have "Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ = Λ ∪ (+) (-u1) -' Λ
∪ (+) u1 -' Λ"
    unfolding image_plus_conv_vimage_plus by (simp add: Un_ac)
  finally have "connected (UNIV - ...)"
    by (intro sparse_imp_connected) auto
  also have "... = A"
    by (auto simp: A_altdef)
  finally show "connected A" .
qed
next

```

A and B can both be written in the form “the complex plane minus continuous deformations of the lattice”. Since the lattice is a closed set, A and B

are open.

```

show "open A" unfolding A_altdef
  by (intro open_Diff closed_Un closed_lattice closed_vimage continuous_intros)
auto?
show "open B" unfolding B_altdef
  by (intro open_Diff <open A> closed_lattice closed_vimage continuous_intros)
auto?
next

```

Finally, we apply the restricted version of the identity we already proved before.

```

show "?lhs u2 = ?rhs u2" if "u2 ∈ B" for u2
  using weierstrass_fun_add_aux(1)[of u1 u2] u12(1) that by (auto
simp: A_def B_def)
qed (use u12 in <auto simp: A_def B_def intro!: holomorphic_intros simp:
rel_def weierstrass_fun_eq_iff>)
qed

```

lemma *weierstrass_fun_diff*:

```

assumes u12: "u1 ∉ Λ" "u2 ∉ Λ" "¬rel u1 u2" "¬rel u1 (-u2)"
shows "ϕ (u1 - u2) = -ϕ u1 - ϕ u2 + ((ϕ' u1 + ϕ' u2) / (ϕ u1 - ϕ
u2))2 / 4"
proof -
  have "ϕ (u1 + (-u2)) = -ϕ u1 - ϕ (- u2) + ((ϕ' u1 + ϕ' u2) / (ϕ u1
- ϕ (- u2)))2 / 4"
  by (subst weierstrass_fun_add) (use u12 in <auto simp: uminus_in_lattice_iff>)
  thus ?thesis
  by (simp add: weierstrass_fun.even)
qed

```

Using the addition theorem for $\wp(z+w)$ and letting $w \rightarrow z$ gives us the duplication theorem: $\wp(2z) = -2\wp(z) + \frac{1}{4}(\wp''(z)/\wp'(z))^2$

This is Exercise 1.9 in Apostol's book.

theorem *weierstrass_fun_double*:

```

assumes z: "2 * z ∉ Λ"
shows "ϕ (2 * z) = -2 * ϕ z + (deriv ϕ' z / ϕ' z)2 / 4"
proof (rule tendsto_unique)
  have z': "z ∉ Λ"
  using z by (metis assms add_in_lattice mult_2)

  show "(λw. -ϕ z - ϕ w + ((ϕ' z - ϕ' w) / (ϕ z - ϕ w))2 / 4) -z→ ϕ
(2 * z)"
  proof (rule Lim_transform_eventually)
    show "(λw. ϕ (z + w)) -z→ ϕ (2 * z)"
    by (rule tendsto_eq_intros refl)+ (use z in auto)
  next
  have "eventually (λw. w ∈ -(Λ ∪ (+) z -' Λ ∪ (+) (-z) -' (Λ-{0}))
- {z}) (at z)"

```

```

    using z z' by (intro eventually_at_in_open open_Cmpl closed_Un
closed_vimage
                    closed_subset_lattice) (auto intro!: continuous_intros)
    thus "eventually ( $\lambda w. \varphi (z + w) = -\varphi z - \varphi w + ((\varphi' z - \varphi' w) / (\varphi z - \varphi w))^2 / 4$ ) (at z)"
    proof eventually_elim
      case (elim w)
      show ?case
        by (subst weierstrass_fun_add) (use z' elim in <auto simp: rel_def
diff_in_lattice_commute>)
    qed
  qed

  show " $(\lambda w. -\varphi z - \varphi w + ((\varphi' z - \varphi' w) / (\varphi z - \varphi w))^2 / 4) -z \rightarrow -2 * \varphi z + (\text{deriv } \varphi' z / \varphi' z) ^ 2 / 4$ "
  proof -
    have *: " $(\lambda w. (\varphi' z - \varphi' w) / (\varphi z - \varphi w)) -z \rightarrow (-\text{deriv } \varphi' z) / (-\varphi' z)$ "
    by (rule lhopital_complex_simple[OF _ _ _ _ refl])
    (use z z' in <auto simp: weierstrass_fun_deriv_eq_0_iff weierstrass_fun_ODE2
                    intro!: derivative_eq_intros>)
    have " $(\lambda w. -\varphi z - \varphi w + ((\varphi' z - \varphi' w) / (\varphi z - \varphi w))^2 / 4) -z \rightarrow -\varphi z - \varphi z + ((-\text{deriv } \varphi' z) / (-\varphi' z)) ^ 2 / 4$ "
    by (rule * tendsto_intros z')+ auto
    thus ?thesis
    by simp
  qed
qed auto

theorem weierstrass_fun_deriv_add:
  assumes u12: "u1  $\notin$   $\Lambda$ " "u2  $\notin$   $\Lambda$ " " $\neg$ rel u1 u2" " $\neg$ rel u1 (-u2)"
  shows " $\varphi' (u1 + u2) = -\varphi' u1 + (\varphi u1 - \varphi (u1 + u2)) * (\varphi' u1 - \varphi' u2) / (\varphi u1 - \varphi u2)$ "
    (is "?lhs u2 = ?rhs u2")
  proof -
    define A where "A =  $-(\Lambda \cup \{z. \text{rel } u1 z\} \cup \{z. \text{rel } u1 (-z)\})$ "
    define B where "B = A -  $\{z. u1 + 2 * z \in \Lambda\} - \{z. 2 * u1 + z \in \Lambda\}$ "

    have A_altdef: "A = UNIV -  $(\Lambda \cup ((+) (-u1) -' \Lambda) \cup ((+) u1 -' \Lambda))$ "
    by (auto simp: A_def rel_def add_ac diff_in_lattice_commute)
    have B_altdef: "B = A -  $(\lambda z. 2 * z + u1) -' \Lambda - ((+) (2*u1)) -' \Lambda$ "
    unfolding B_def A_altdef by (auto simp: A_def add_ac)

    show ?thesis
    proof (rule analytic_continuation_open[where f = ?lhs and g = ?rhs])

```

Our set B can be written as the complex plane minus some shifted and

scaled copies of the lattice, i.e. an uncountable set minus a countable set. Therefore, B is clearly non-empty.

```

show "B ≠ {}"
proof -
  have "B = UNIV - (Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ ∪ (λz. 2 * z
+ u1) -' Λ ∪ (+) (-2*u1) ' Λ)"
    unfolding A_altdef B_altdef unfolding image_plus_conv_vimage_plus
by auto
  also have "(λz. 2 * z + u1) -' Λ = (λz. (z - u1) / 2) ' Λ"
  proof safe
    fix z assume "2 * z + u1 ∈ Λ"
    thus "z ∈ (λz. (z - u1) / 2) ' Λ"
      by (intro image_eqI[of _ _ "u1 + 2 * z"]) (auto simp: algebra_simps)
  qed auto
  finally have "B = UNIV - (Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ ∪
(λz. (z - u1) / 2) ' Λ ∪ (+) (-2*u1)
' Λ)" .
  also have "uncountable ..."
    by (intro uncountable_minus_countable countable_Un countable_lattice
countable_image uncountable_UNIV_complex)
  finally have "uncountable B" .
  thus "B ≠ {}"
    by auto
qed
next

```

Similarly, A can be written as the complex plane minus some shifted copies of the lattice, i.e. the complement of a sparse set. Clearly, what remains is still connected.

```

show "connected A"
proof -
  have "(Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ) sparse_in UNIV"
    unfolding A_altdef by (intro sparse_in_union' sparse_in_translate_UNIV
lattice_sparse)
  also have "Λ ∪ (+) u1 ' Λ ∪ (+) (-u1) ' Λ = Λ ∪ (+) (-u1) -' Λ
∪ (+) u1 -' Λ"
    unfolding image_plus_conv_vimage_plus by (simp add: Un_ac)
  finally have "connected (UNIV - ...)"
    by (intro sparse_imp_connected) auto
  also have "... = A"
    by (auto simp: A_altdef)
  finally show "connected A" .
qed
next

```

A and B can both be written in the form “the complex plane minus continuous deformations of the lattice”. Since the lattice is a closed set, A and B are open.

```

    show "open A" unfolding A_altdef
      by (intro open_Diff closed_Un closed_lattice closed_vimage continuous_intros)
auto?
    show "open B" unfolding B_altdef
      by (intro open_Diff <open A> closed_lattice closed_vimage continuous_intros)
auto?
  next

```

Finally, we apply the restricted version of the identity we already proved before.

```

    show "?lhs u2 = ?rhs u2" if "u2 ∈ B" for u2
      using weierstrass_fun_add_aux(2)[of u1 u2] u12(1) that by (auto
simp: A_def B_def)
    qed (use u12 in <auto simp: A_def B_def intro!: holomorphic_intros simp:
rel_def weierstrass_fun_eq_iff>)
qed

```

theorem weierstrass_fun_deriv_double:

```

  assumes z: "z ∈ Λ"
  defines "a ≡ deriv ϕ' z / ϕ' z"
  shows "ϕ' (2 * z) = -ϕ' z + 3 * a * ϕ z - a ^ 3 / 4"
proof (rule tendsto_unique)
  have z': "z ∈ Λ"
    using z by (metis assms add_in_lattice mult_2)
  have z'': "z + z ∈ Λ"
    using z by simp

  show "(λw. -ϕ' z + (ϕ z - ϕ (z + w)) * (ϕ' z - ϕ' w) / (ϕ z - ϕ w))
-z → ϕ' (2 * z)"
  proof (rule Lim_transform_eventually)
    show "(λw. ϕ' (z + w)) -z → ϕ' (2 * z)"
      by (rule tendsto_eq_intros refl)+ (use z in auto)
  next
    have "eventually (λw. w ∈ -(Λ ∪ (+) z -' Λ ∪ (+) (-z) -' (Λ-{0}))
- {z}) (at z)"
      using z z' by (intro eventually_at_in_open open_Cmpl closed_Un
closed_vimage
closed_subset_lattice) (auto intro!: continuous_intros)
    thus "eventually (λw. ϕ' (z + w) = -ϕ' z + (ϕ z - ϕ (z + w)) * (ϕ'
z - ϕ' w) / (ϕ z - ϕ w)) (at z)"
      proof eventually_elim
        case (elim w)
        show ?case
          by (subst weierstrass_fun_deriv_add)
            (use z' elim in <auto simp: rel_def diff_in_lattice_commute>)
      qed
    qed

  show "(λw. -ϕ' z + (ϕ z - ϕ (z + w)) * (ϕ' z - ϕ' w) / (ϕ z - ϕ w))

```

```

-z→
      -φ' z + 3 * a * φ z - a ^ 3 / 4"
  proof -
    have "(λw. (φ' z - φ' w) / (φ z - φ w)) -z→ (-deriv φ' z) / (-φ'
z)"
      by (rule lhopital_complex_simple[OF _ _ _ _ refl])
          (use z z' in <auto simp: weierstrass_fun_deriv_eq_0_iff weierstrass_fun_ODE2
              intro!: derivative_eq_intros>)
    hence *: "(λw. (φ' z - φ' w) / (φ z - φ w)) -z→ a"
      by (simp add: a_def)
    have "(λw. -φ' z + (φ z - φ (z + w)) * ((φ' z - φ' w) / (φ z - φ
w)))
      -z→ -φ' z + (φ z - φ (z + z)) * a"
      by (rule * tendsto_intros z'')+
    also have "φ z - φ (z + z) = 3 * φ z - a ^ 2 / 4"
      using z by (simp add: weierstrass_fun_double a_def)
    also have "... * a = 3 * φ z * a - a ^ 3 / 4"
      by (simp add: algebra_simps power3_eq_cube power2_eq_square)
    finally show ?thesis
      by (simp add: algebra_simps)
  qed
qed auto

end

end

theory FPS_Homomorphism
  imports "HOL-Computational_Algebra.Formal_Laurent_Series" "Polynomial_Interpolation.Ring_
begin

interpretation fps_to_fls: comm_ring_hom "fps_to_fls :: 'a :: comm_ring_1
fps ⇒ 'a fls"
  by standard (auto simp: fls_times_fps_to_fls)

interpretation fps_to_fls: inj_idom_hom "fps_to_fls :: 'a :: idom fps ⇒
'a fls"
  by standard auto

interpretation fps_const: comm_ring_hom "fps_const :: 'a :: comm_ring_1
⇒ 'a fps"
  by standard auto

interpretation fps_const: inj_idom_hom "fps_const :: 'a :: idom ⇒ 'a fps"
  by standard auto

interpretation fls_const: comm_ring_hom "fls_const :: 'a :: comm_ring_1
⇒ 'a fls"
  by standard (auto simp: fls_plus_const)

```

```

interpretation fls_const: inj_idom_hom "fls_const :: 'a :: idom  $\Rightarrow$  'a fls"
  by standard auto

interpretation fls_const: field_hom "fls_const :: 'a :: field  $\Rightarrow$  'a fls"
  by standard auto

interpretation fls_const: field_char_0_hom "fls_const :: 'a :: field_char_0
 $\Rightarrow$  'a fls"
  by standard auto

locale fps_homomorphism =
  fixes f :: "'a :: comm_semiring_1  $\Rightarrow$  'b :: comm_semiring_1"
  assumes f_0: "f 0 = 0"
  assumes f_1: "f 1 = 1"
  assumes f_add: "f (x + y) = f x + f y"
  assumes f_mult: "f (x * y) = f x * f y"
begin

lemma f_sum: "f ( $\sum x \in A. g x$ ) = ( $\sum x \in A. f (g x)$ )"
  by (induction A rule: infinite_finite_induct) (simp_all add: f_0 f_add)

lemma f_prod: "f ( $\prod x \in A. g x$ ) = ( $\prod x \in A. f (g x)$ )"
  by (induction A rule: infinite_finite_induct) (simp_all add: f_1 f_mult)

lemma f_sum_list: "f (sum_list xs) = sum_list (map f xs)"
  by (induction xs) (simp_all add: f_0 f_add)

lemma f_prod_list: "f (prod_list xs) = prod_list (map f xs)"
  by (induction xs) (simp_all add: f_1 f_mult)

lemma f_power: "f (x ^ n) = f x ^ n"
  using f_prod[of " $\lambda_. x$ " "{.. $n$ "}] by simp

lemmas f_simps = f_0 f_1 f_add f_mult f_sum f_prod f_power

definition fps :: "'a fps  $\Rightarrow$  'b fps" where
  "fps F = Abs_fps ( $\lambda n. f (fps\_nth F n)$ )"

lemma fps_nth [simp]: "fps_nth (fps F) n = f (fps_nth F n)"
  by (simp add: fps_def)

lemma Abs_fps [simp]: "fps (Abs_fps g) = Abs_fps ( $\lambda n. f (g n)$ )"
  by (simp add: fps_eq_iff)

lemma fps_0 [simp]: "fps 0 = 0"
  and fps_1 [simp]: "fps 1 = 1"
  and fps_const [simp]: "fps (fps_const c) = fps_const (f c)"

```

```

and fps_add [simp]: "fps (F + G) = fps F + fps G"
and fps_mult [simp]: "fps (F * G) = fps F * fps G"
and fps_shift [simp]: "fps (fps_shift n F) = fps_shift n (fps F)"
by (simp_all add: fps_eq_iff f_simps fps_mult_nth)

lemma fps_sum: "fps ( $\sum x \in A. g x$ ) = ( $\sum x \in A. fps (g x)$ )"
by (induction A rule: infinite_finite_induct) (simp_all add: f_0 f_add)

lemma fps_prod: "fps ( $\prod x \in A. g x$ ) = ( $\prod x \in A. fps (g x)$ )"
by (induction A rule: infinite_finite_induct) (simp_all add: f_1 f_mult)

lemma fps_sum_list: "fps (sum_list xs) = sum_list (map fps xs)"
by (induction xs) (simp_all add: f_0 f_add)

lemma fps_prod_list: "fps (prod_list xs) = prod_list (map fps xs)"
by (induction xs) (simp_all add: f_1 f_mult)

lemma fps_power [simp]: "fps (x ^ n) = fps x ^ n"
using fps_prod[of " $\lambda_. x$ " "{.. $n$ "}] by simp

lemma fps_of_nat [simp]: "fps (of_nat n) = of_nat n"
by (induction n) (simp_all add: f_simps)

lemma fps_numeral [simp]: "fps (numeral num) = numeral num"
by (subst (1 2) of_nat_numeral [symmetric]) (rule fps_of_nat)

end

locale fps_homomorphism_ring =
  fps_homomorphism f for f :: "'a :: comm_ring_1  $\Rightarrow$  'b :: comm_ring_1"
begin

lemma fps_minus [simp]: "fps (-x) = -fps x"
proof -
  have "0 = x + (-x)"
  by simp
  also have "fps ... = fps x + fps (-x)"
  by (subst fps_add) auto
  finally show ?thesis
  by (simp add: algebra_simps add_eq_0_iff)
qed

lemma fps_diff [simp]: "fps (F - G) = fps F - fps G"
proof -
  have "fps (F - G) = fps (F + -G)"
  by simp
  thus ?thesis
  unfolding fps_add by simp

```

qed

```
lemma fps_of_int [simp]: "fps (of_int m) = of_int m"
  using fps_of_nat[of "nat m"] fps_minus[of "of_nat (nat (-m))"] fps_of_nat[of
"nat (-m)"]
  by (cases "m ≥ 0") (simp_all del: fps_minus fps_of_nat)
```

end

```
interpretation of_nat: fps_homomorphism of_nat
  by standard auto
```

```
interpretation of_nat_fps: comm_semiring_hom "of_nat.fps"
  by standard auto
```

```
interpretation of_nat_fps: inj_semiring_hom "of_nat.fps :: nat fps ⇒ 'a
:: {comm_semiring_1, semiring_char_0} fps"
  by standard (auto simp: fps_eq_iff)
```

```
interpretation of_int: fps_homomorphism of_int
  by standard auto
```

```
interpretation of_int: fps_homomorphism_ring of_int
  by standard auto
```

```
interpretation of_int_fps: comm_ring_hom "of_int.fps"
  by standard auto
```

```
interpretation of_int_fps: inj_idom_hom "of_int.fps :: int fps ⇒ 'a ::
{idom, ring_char_0} fps"
  by standard (auto simp: fps_eq_iff)
```

end

11 Connection between complex lattices and theta functions

```
theory Complex_Lattices_Theta
  imports Eisenstein_Series "Theta_Functions.Theta_Nullwert"
begin
```

```
lemmas [simp del] = div_mult_self1 div_mult_self2 div_mult_self3 div_mult_self4
```

```
unbundle jacobi_theta_nw_notation
```

We make the connection to theta functions. In order to do that, we first

assume that the generators ω_1 and ω_2 are such that their ratio $\tau := \omega_2/\omega_1$ has positive imaginary part.

```

locale complex_lattice_Im_pos = complex_lattice +
  assumes Im_ratio_pos: "Im ratio > 0"
begin

```

We fix this ratio τ as the second parameter of the theta functions so that the theta functions become quasi-elliptic functions in one variable z .

```

definition theta_00 ("<notation=<mixfix complex_lattice_Im_pos.theta_00>> $\vartheta_{00}$ '(')"))
  where "theta_00 z  $\equiv$  jacobi_theta_00 (z /  $\omega_1$ )  $\tau$ "

```

```

definition theta_01 ("<notation=<mixfix complex_lattice_Im_pos.theta_00>> $\vartheta_{01}$ '(')"))
  where "theta_01 z  $\equiv$  jacobi_theta_01 (z /  $\omega_1$ )  $\tau$ "

```

```

definition theta_10 ("<notation=<mixfix complex_lattice_Im_pos.theta_00>> $\vartheta_{10}$ '(')"))
  where "theta_10 z  $\equiv$  jacobi_theta_10 (z /  $\omega_1$ )  $\tau$ "

```

```

definition theta_11 ("<notation=<mixfix complex_lattice_Im_pos.theta_00>> $\vartheta_{11}$ '(')"))
  where "theta_11 z  $\equiv$  jacobi_theta_11 (z /  $\omega_1$ )  $\tau$ "

```

```

lemma theta_01_conv_00: "theta_01 z = theta_00 (z +  $\omega_1$  / 2)"
  by (simp add: theta_01_def jacobi_theta_01_def theta_00_def add_divide_distrib)

```

```

lemma theta_10_conv_00: "theta_10 z = to_nome (z /  $\omega_1$  +  $\tau$  / 4) * theta_00
(z +  $\omega_2$  / 2)"
  by (simp add: theta_10_def jacobi_theta_10_def theta_00_def add_divide_distrib
ratio_def mult_ac)

```

```

lemma theta_11_conv_00:
  "theta_11 z = to_nome (z /  $\omega_1$  +  $\tau$  / 4 + 1 / 2) * theta_00 (z + ( $\omega_1$  +
 $\omega_2$ ) / 2)"
  by (simp add: theta_11_def jacobi_theta_11_def theta_00_def add_divide_distrib
ratio_def algebra_simps)

```

The four zeta functions then each have their zeros at various lattice or half-lattice points.

```

lemma theta_00_eq_0_iff: " $\vartheta_{00}(z) = 0 \iff \text{rel } z ((\omega_1 + \omega_2) / 2)$ " for
z

```

proof

```

  assume "theta_00(z) = 0"
  then obtain m n :: int where "z /  $\omega_1$  = (of_int m + 1 / 2) + (of_int
n + 1 / 2) *  $\tau$ "
  using Im_ratio_pos by (auto simp: theta_00_def jacobi_theta_00_eq_0_iff_complex)
  hence "z = ( $\omega_1$  +  $\omega_2$ ) / 2 + of_int m *  $\omega_1$  + of_int n *  $\omega_2$ "
  by (auto simp: ratio_def divide_simps) (auto simp: algebra_simps)?
  also have "rel ... (( $\omega_1$  +  $\omega_2$ ) / 2)"
  by (auto simp: rel_def intro!: lattice_intros)
  finally show "rel z (( $\omega_1$  +  $\omega_2$ ) / 2)" .

```

```

next
  assume "rel z (( $\omega_1 + \omega_2$ ) / 2)"
  then obtain m n :: int where "z = ( $\omega_1 + \omega_2$ ) / 2 + of_int m *  $\omega_1$  +
of_int n *  $\omega_2$ "
    by (auto simp: rel_def of_ $\omega_{12}$ _coords_def field_simps elim!: latticeE)
  also have "... /  $\omega_1$  = (of_int m + 1 / 2) + (of_int n + 1 / 2) *  $\tau$ "
    by (auto simp: ratio_def field_simps)
  also have "jacobi_theta_00 ...  $\tau$  = 0"
    by (rule jacobi_theta_00_eq_0')
  finally show " $\vartheta_{00}(z) = 0$ "
    by (simp add: theta_00_def)
qed

lemma theta_01_eq_0_iff: " $\vartheta_{01}(z) = 0 \iff \text{rel } z (\omega_2 / 2)$ "
  unfolding theta_01_conv_00 theta_00_eq_0_iff rel_def
  by (simp add: add_divide_distrib)

lemma theta_10_eq_0_iff: " $\vartheta_{10}(z) = 0 \iff \text{rel } z (\omega_1 / 2)$ "
  unfolding theta_10_conv_00
  by (simp add: theta_00_eq_0_iff add_divide_distrib rel_def)

lemma theta_11_eq_0_iff: " $\vartheta_{11}(z) = 0 \iff z \in \Lambda$ "
  unfolding theta_11_conv_00
  by (simp add: theta_00_eq_0_iff add_divide_distrib rel_def)

lemma zorder_theta_00: "zorder theta_00 (( $\omega_1 + \omega_2$ ) / 2) = 1"
proof -
  define z0 where "z0 = ( $\omega_1 + \omega_2$ ) / 2"
  have z0_over_ $\omega_1$ : "z0 /  $\omega_1$  = ( $\tau + 1$ ) / 2"
    by (auto simp: z0_def ratio_def field_simps)
  have *: "( $\lambda z. \text{theta\_00 } (z_0 + z)$ ) = (( $\lambda z. \vartheta_{00}((\tau + 1) / 2 + z ; \tau)$ )  $\circ$ 
( $\lambda z. z / \omega_1$ ))"
    unfolding z0_over_ $\omega_1$  [symmetric] by (simp add: theta_00_def [abs_def]
o_def add_divide_distrib)

  define F where "F = fps_expansion ( $\lambda z. \vartheta_{00}(z ; \tau)$ ) (( $\tau + 1$ ) / 2)"
  have F: "( $\lambda z. \vartheta_{00}((\tau + 1) / 2 + z ; \tau)$ ) has_fps_expansion F"
    unfolding F_def by (intro analytic_at_imp_has_fps_expansion analytic_intros
Im_ratio_pos)
  have F': "( $\lambda z. \text{theta\_00 } (z_0 + z)$ ) has_fps_expansion (F oo (fps_X / fps_const
 $\omega_1$ ))"
    unfolding F_def *
    by (intro fps_expansion_intros analytic_at_imp_has_fps_expansion
analytic_intros Im_ratio_pos) auto

  have nz: "F oo (fps_X / fps_const  $\omega_1$ )  $\neq$  0"
proof
  assume "F oo (fps_X / fps_const  $\omega_1$ ) = 0"
  hence "( $\lambda z. \vartheta_{00}(z_0 + z)$ ) has_fps_expansion 0"

```

```

    using F' by simp
    hence " $\vartheta_{00}(z_0 + (-z_0)) = 0$ "
      by (rule has_fps_expansion_0_analytic_continuation[where A = UNIV])
          (auto intro!: holomorphic_intros simp: theta_00_def [abs_def]
    Im_ratio_pos)
    thus False
      by (simp add: theta_00_eq_0_iff rel_def uminus_in_lattice_iff)
    qed
    hence [simp]: " $F \neq 0$ "
      by auto

    have "1 = zorder ( $\lambda z. \vartheta_{00}(z ; \tau)$ ) (( $\tau + 1$ ) / 2)"
      by (subst jacobi_theta_00_simple_zero) (simp_all add: Im_ratio_pos
    jacobi_theta_00_eq_0)
    also have "... = subdegree F"
      using has_fps_expansion_zorder[OF F] by simp
    also have "... = zorder theta_00 z0"
      using has_fps_expansion_zorder[OF F'] nz by simp
    finally show ?thesis
      by (simp add: z0_def)
  qed

```

By comparing the zeros of $\wp(z) - e_2$ and $(\vartheta_{01}(z)/\vartheta_{11}(z))^2$ we find that the two functions are identical up to a constant factor, which we then determine to be $(\pi\vartheta_{10}(0)\vartheta_{00}(0)/\omega_1)^2$ by comparing the Laurent series expansions of the two functions at their pole at the origin.

It follows that we can express \wp in terms of the constant e_2 and the theta functions.

```

lemma weierstrass_fun_conv_theta:
  assumes z: " $z \notin \Lambda$ "
  shows " $\wp z = e_2 + (\pi * \vartheta_{10}(0) * \vartheta_{00}(0) / \omega_1) ^ 2 * \vartheta_{01}(z) ^ 2 / \vartheta_{11}(z) ^ 2$ "
proof -
  define f where "f = ( $\lambda z. \wp z - \text{number\_e2}$ )"
  interpret f: weierstrass_fun_minus_const  $\omega_1 \omega_2$  " $\omega_2 / 2$ " f
    by unfold_locales (auto simp: f_def number_e2_def)
  define g where "g = ( $\lambda z. (\text{theta\_01 } z / \text{theta\_11 } z) ^ 2$ )"

  interpret g: even_elliptic_function  $\omega_1 \omega_2$  g
proof
  fix z :: complex
  show "g (z +  $\omega_1$ ) = g z"
    by (simp add: g_def add_divide_distrib theta_01_def theta_11_def

          jacobi_theta_01_left.plus_1 jacobi_theta_11_plus1_left)
next
  fix z :: complex
  have "g (z +  $\omega_2$ ) = g z * (to_nome ( $\tau + 2 * z / \omega_1$ ) / to_nome (2 *

```

```

z / ω1 + τ)) ^ 2"
  by (simp add: g_def add_divide_distrib ratio_def jacobi_theta_01_plus_quasiperiod

          jacobi_theta_11_plus_quasiperiod power_divide theta_01_def
theta_11_def)
  also have "to_nome (τ + 2 * z / ω1) / to_nome (2 * z / ω1 + τ) = 1"
    unfolding to_nome_diff [symmetric] by simp
  finally show "g (z + ω2) = g z"
    by simp
next
show "g meromorphic_on UNIV"
  unfolding g_def theta_01_def theta_11_def using Im_ratio_pos
  by (intro meromorphic_intros analytic_on_imp_meromorphic_on analytic_intros)
auto
next
fix z show "g (-z) = g z"
  by (auto simp: g_def theta_01_def theta_11_def)
qed

define Z where "Z = {z ∈ half_fund_parallelogram \ {0}. is_pole g z
  ∨ isolated_zero g z}"
define h where "h = (λz. zorder g z div (if 2 * z ∈ Λ then 2 else 1))"
have [analytic_intros]: "g analytic_on A" if "A ∩ Λ = {}" for A
  using Im_ratio_pos that theta_11_eq_0_iff unfolding g_def theta_01_def
theta_11_def
  by (auto intro!: analytic_intros)

have g_nz: "¬(∀ ≈z. g z = 0)"
proof
assume "∀ ≈z. g z = 0"
moreover have "∀ ≈z. g z ≠ 0"
  using eventually_not_rel_cosparse[of "ω2 / 2"] eventually_not_in_lattice_cosparse
  by eventually_elim (auto simp: g_def theta_01_eq_0_iff theta_11_eq_0_iff)
ultimately have "∀ ≈z::complex. False"
  by eventually_elim auto
thus False
  by simp
qed

define z0 where "z0 = ω2 / 2"
have z0: "z0 ∈ half_fund_parallelogram" "z0 ≠ 0" "z0 ∉ Λ" "rel z0
(ω2 / 2)"
  by (auto simp: half_fund_parallelogram_altdef z0_def)

have zero_at_z0: "isolated_zero g z0"
proof (subst g.isolated_zero_analytic_iff)
show "g analytic_on {z0}" using z0
  by (auto intro!: analytic_intros)
next

```

```

    show "g z0 = 0" using z0
      by (auto simp: g_def theta_01_eq_0_iff)
qed (use g_nz in auto)

have Z_eq: "Z = {z0}"
proof (intro equalityI subsetI)
  fix z assume "z ∈ Z"
  hence z: "z ∈ half_fund_parallelogram" "z ≠ 0" and z': "is_pole
g z ∨ isolated_zero g z"
  by (auto simp: Z_def)
  have "z ∉ Λ"
  using z by (metis half_fund_parallelogram_in_lattice_iff)
  hence "g analytic_on {z}"
  by (auto intro!: analytic_intros)
  hence "¬is_pole g z"
  by (rule analytic_at_imp_no_pole)
  with z' have "isolated_zero g z"
  by auto
  hence "g z = 0"
  using <g analytic_on {z}> by (simp add: zero_isolated_zero_analytic)
  with <z ∉ Λ> have "rel z (ω2 / 2)"
  by (auto simp: g_def theta_01_eq_0_iff theta_11_eq_0_iff)
  moreover have "ω2 / 2 ∈ period_parallelogram 0"
  unfolding period_parallelogram_altdef by auto
  moreover have "z ∈ period_parallelogram 0"
  using z(1) half_fund_parallelogram_subset_period_parallelogram by
blast
  ultimately show "z ∈ {z0}"
  using to_fund_parallelogram_unique' unfolding z0_def by blast
next
  fix z assume "z ∈ {z0}"
  thus "z ∈ Z"
  using z0 zero_at_z0 by (auto simp: Z_def)
qed

define A where "A = fps_expansion theta_01 0"
define B where "B = fps_expansion theta_11 0"
have A[fps_expansion_intros]: "theta_01 has_fps_expansion A"
  unfolding A_def theta_01_def
  by (intro analytic_at_imp_has_fps_expansion_0 analytic_intros) (use
Im_ratio_pos in auto)
have B[fps_expansion_intros]: "theta_11 has_fps_expansion B"
  unfolding B_def theta_11_def
  by (intro analytic_at_imp_has_fps_expansion_0 analytic_intros) (use
Im_ratio_pos in auto)

have A0: "fps_nth A 0 = ϑ01(0)"
  using has_fps_expansion_imp_0_eq_fps_nth_0[OF A] by (simp add: theta_01_def)
have [simp]: "A ≠ 0"

```

```

proof -
  have "fps_nth A 0 ≠ 0"
    by (auto simp: A0 theta_01_eq_0_iff rel_def uminus_in_lattice_iff)
  thus "A ≠ 0"
    by auto
qed
have [simp]: "subdegree A = 0"
  by (rule subdegree_eq_0)
  (use theta_01_eq_0_iff[of 0] in <auto simp: A0 rel_def uminus_in_lattice_iff>)

have B0: "fps_nth B 0 = 0"
  using has_fps_expansion_imp_0_eq_fps_nth_0[OF B] by (simp add: theta_11_def)
have B1: "fps_nth B (Suc 0) = deriv theta_11 0"
  using fps_nth_fps_expansion[OF B, of 1] by simp
have B1': "fps_nth B (Suc 0) =
  -(of_real pi * theta_00 0 * theta_01 0 * theta_10 0 / ω1)"
proof -
  have "(((λx. jacobi_theta_11 x τ) ∘ (λx. x / ω1)) has_field_derivative
    (deriv (λx. jacobi_theta_11 x τ) (0 / ω1) * (1 / ω1))) (at
0)"
    unfolding B1 using Im_ratio_pos
    by (intro DERIV_chain analytic_derivI analytic_intros)
    (auto intro!: derivative_eq_intros)
  hence "(theta_11 has_field_derivative (deriv (λx. jacobi_theta_11
x τ) 0 / ω1)) (at 0)"
    by (simp add: o_def theta_11_def [abs_def])
  thus ?thesis
    unfolding B1 using Im_ratio_pos
    by (intro DERIV_imp_deriv)
    (simp add: deriv_jacobi_theta_11_at_0 theta_00_def theta_01_def
theta_10_def mult_ac)
qed
have B1_nz: "fps_nth B (Suc 0) ≠ 0"
  by (auto simp: B1' theta_00_eq_0_iff theta_01_eq_0_iff theta_10_eq_0_iff
rel_def uminus_in_lattice_iff)
have [simp]: "B ≠ 0"
  using B1_nz by auto
have [simp]: "subdegree B = 1"
  by (rule subdegreeI) (auto simp: B1_nz B0)
have B1_nz: "fps_nth B (Suc 0) ≠ 0"
  using nth_subdegree_nonzero[of B] by (simp del: nth_subdegree_nonzero)
obtain c where "∀ z. g z = c * (∏ w∈Z. (φ z - φ w) powi h w)"
  using g.in_terms_of_weierstrass_fun_even_aux[OF g_nz]
  unfolding h_def unfolding Z_def by blast
also have "(λz. c * (∏ w∈Z. (φ z - φ w) powi h w)) = (λz. c * (φ z
- e₂) powi h z0)"
  by (simp add: Z_eq z0_def number_e2_def)

```

```

also have "h z0 = zorder g z0 div 2"
  by (simp add: h_def z0_def)
also have "zorder g z0 = 2"
proof -

  have ev_nz: " $\forall_F z \text{ in at } z0. \vartheta_{01}(z) \neq 0$ " " $\forall_F z \text{ in at } z0. \vartheta_{11}(z) \neq 0$ "
0"
    using eventually_not_rel_cosparse[of " $\omega_2/2$ "] eventually_not_in_lattice_cosparse
    by (auto simp: theta_01_eq_0_iff theta_11_eq_0_iff dest: eventually_cosparse_imp_eventually)
  from z0 have nz: " $\vartheta_{11}(z0) \neq 0$ "
    by (subst theta_11_eq_0_iff) auto

  have " $\forall_F z \text{ in at } z0. \vartheta_{01}(z) / \vartheta_{11}(z) \neq 0$ "
    using ev_nz by eventually_elim auto
  hence "zorder g z0 = 2 * zorder ( $\lambda z. \vartheta_{01}(z) / \vartheta_{11}(z)$ ) z0"
    unfolding g_def using ev_nz nz
    by (subst zorder_power)
    (auto simp: theta_11_def [abs_def] theta_01_def [abs_def] Im_ratio_pos
theta_11_eq_0_iff
  intro!: analytic_on_imp_meromorphic_on analytic_intros
eventually_frequently)
  also have "zorder ( $\lambda z. \vartheta_{01}(z) / \vartheta_{11}(z)$ ) z0 = zorder ( $\lambda z. \vartheta_{01}(z)$ ) z0
- zorder ( $\lambda z. \vartheta_{11}(z)$ ) z0"
    using ev_nz by (subst zorder_divide)
    (auto intro!: analytic_on_imp_meromorphic_on analytic_intros
eventually_frequently
  simp: theta_01_def [abs_def] theta_11_def [abs_def]
Im_ratio_pos)
  also from Im_ratio_pos and nz have "zorder ( $\lambda z. \vartheta_{11}(z)$ ) z0 = 0"
    by (intro zorder_eq_0I) (auto simp: theta_11_def [abs_def] intro!:
analytic_intros)
  also have "zorder theta_01 z0 = zorder theta_00 (( $\omega_1 + \omega_2$ ) / 2)"
    by (simp add: theta_01_conv_00 [abs_def] zorder_shift' z0_def add_ac
add_divide_distrib)
  also have "... = 1"
    by (rule zorder_theta_00)
  finally show "zorder g z0 = 2"
    by simp
qed
finally have g_eq: " $\forall_{\approx} z. g z = c * (\wp z - e_2)$ "
  by simp

have g_eq': "g z = c * ( $\wp z - e_2$ )" if "z  $\notin \Lambda$ " for z
  using g_eq
proof (rule analytic_on_continuation)
  show "z  $\in (-\Lambda) \cap UNIV$ "
    using that by auto
qed (auto intro!: analytic_intros)

```

```

define F where "F = ((fps_to_fls A / fps_to_fls B) ^ 2 - fls_const
c * (fls_weierstrass - fls_const e2))"

have "(λz. g z - c * (ϕ z - e2)) has_laurent_expansion F"
  unfolding F_def g_def
  by (intro laurent_expansion_intros has_laurent_expansion_fps fps_expansion_intros)
also have "?this ↔ (λz. 0) has_laurent_expansion F"
proof (rule has_laurent_expansion_cong)
  have "∀F x in at 0. g x = c * (ϕ x - e2)"
    using g_eq by (auto dest: eventually_cosparsely_imp_eventually_at)
  thus "∀F x in at 0. g x - c * (ϕ x - e2) = 0"
    by eventually_elim auto
qed auto
finally have "F = 0"
  by (rule zero_has_laurent_expansion_imp_eq_0)

have "0 = fls_nth F (-2)"
  by (simp add: <F = 0>)
also have "... = fls_nth ((fps_to_fls A / fps_to_fls B)2) (-2) - c"
  by (simp add: fls_weierstrass_def F_def)
also have "-2 = int 2 * fls_subdegree (fps_to_fls A / fps_to_fls B)"
  by (subst fls_divide_subdegree) (auto simp: fls_subdegree_fls_to_fps)
also have "fls_nth ((fps_to_fls A / fps_to_fls B) ^ 2) ... =
  (fls_nth (fps_to_fls A / fps_to_fls B) (-1))2"
  by (subst fls_pow_base) (auto simp: fls_divide_subdegree fls_subdegree_fls_to_fps)
also have "-1 = fls_subdegree (fps_to_fls A) - fls_subdegree (fps_to_fls
B)"
  by (simp add: fls_subdegree_fls_to_fps)
also have "fls_nth (fps_to_fls A / fps_to_fls B) ... = ϑ01(0) / fps_nth
B (Suc 0)"
  by (subst fls_divide_nth_base)
  (auto simp: fls_subdegree_fls_to_fps A0 B1 theta_01_def)
finally have c_eq: "c = (ϑ01(0) / fps_nth B (Suc 0)) ^ 2"
  by simp

have "ϕ z = e2 + (fps_nth B 1 / ϑ01(0)) ^ 2 * ϑ01(z) ^ 2 / ϑ11(z) ^
2"
  using g_eq'[of z] theta_01_eq_0_iff[of 0] theta_11_eq_0_iff[of z]
z B1_nz
  by (auto simp: c_eq g_def rel_def uminus_in_lattice_iff field_simps)
also have "(fps_nth B 1 / ϑ01(0)) ^ 2 * ϑ01(z) ^ 2 / ϑ11(z) ^ 2 =
  (of_real pi * ϑ10(0) * ϑ00(0) / ω1)2 * ϑ01(z)2 / ϑ11(z)2"
  by (simp add: B1' field_simps theta_01_eq_0_iff rel_def uminus_in_lattice_iff)
finally show ?thesis .
qed

```

By plugging in values into the above identity, we derive expressions for e_1 , e_2 , e_3 and the lattice modulus λ purely in terms of theta functions.

lemma $e_{12_conv_theta}$: " $e_1 - e_2 = (\pi / \omega_1) ^ 2 * \vartheta_{00}(0) ^ 4$ "

```

and e32_conv_theta: "e3 - e2 = (pi / ω1) ^ 2 * ϑ10(0) ^ 4"
and e13_conv_theta: "e1 - e3 = (pi / ω1) ^ 2 * ϑ01(0) ^ 4"
and e1_conv_theta: "e1 = (pi / ω1) ^ 2 / 3 * (ϑ00(0) ^ 4 + ϑ01(0) ^
4)"
and e2_conv_theta: "e2 = -(pi / ω1)^2 / 3 * (ϑ00(0) ^ 4 + ϑ10(0) ^ 4)"
and e3_conv_theta: "e3 = (pi / ω1)^2 / 3 * (ϑ10(0) ^ 4 - ϑ01(0) ^ 4)"
and modulus_conv_theta: "modulus = ϑ10(0) ^ 4 / ϑ00(0) ^ 4"
proof -
  have "e1 - e2 = (of_real pi * ϑ10(0) * ϑ00(0) / ω1)^2 * ϑ01(ω1/2)^2 /
ϑ11(ω1/2)^2"
    using weierstrass_fun_conv_theta[of "ω1 / 2"]
    unfolding number_e1_def by simp
  also have "ϑ01(ω1/2) = ϑ00(0)"
    using jacobi_theta_00_left.plus_1[of 0 τ]
    by (simp add: jacobi_theta_01_def theta_00_def theta_01_def)
  also have "ϑ11(ω1/2) = -ϑ10(0)"
    using jacobi_theta_00_left.plus_1[of "τ / 2" τ]
    by (simp add: jacobi_theta_11_def algebra_simps to_nome_add jacobi_theta_10_def
theta_10_def theta_11_def)
  also have "(of_real pi * ϑ10(0) * ϑ00(0) / ω1)^2 * ϑ00(0)^2 / (- ϑ10(0))^2
=
      (pi / ω1) ^ 2 * ϑ00(0) ^ 4"
    using theta_10_eq_0_iff[of 0] by (simp add: field_simps rel_def uminus_in_lattice_iff)
  finally show e12: "e1 - e2 = (pi / ω1) ^ 2 * ϑ00(0) ^ 4" .

  have "e3 - e2 = (of_real pi * ϑ10(0) * ϑ00(0) / ω1)^2 *
      (ϑ01((ω1 + ω2) / 2) / ϑ11((ω1 + ω2) / 2))^2"
    using weierstrass_fun_conv_theta[of "(ω1 + ω2) / 2"] unfolding number_e3_def
    by (simp add: power_divide)
  also have "ϑ01((ω1 + ω2) / 2) = ϑ00(τ/2 + 1 ; τ)"
    by (simp add: theta_01_conv_00 theta_00_def add_divide_distrib ratio_def
mult_ac add_ac)
  also have "... = ϑ00(τ/2 ; τ)"
    by (subst jacobi_theta_00_left.plus_1) auto
  also have "... = ϑ10(0) * to_nome (-τ/4)"
    by (simp add: theta_10_conv_00 theta_00_def ratio_def mult_ac to_nome_minus)
  also have "ϑ11((ω1 + ω2) / 2) = -ϑ00(0 + τ + 1 ; τ) * to_nome (3/4*τ)"
    by (simp add: theta_11_conv_00 theta_00_def add_divide_distrib ratio_def
mult_ac add_ac to_nome_add)
  also have "... = -ϑ00(0) * to_nome (3/4*τ - τ)"
    unfolding jacobi_theta_00_left.plus_1 jacobi_theta_00_plus_quasiperiod
to_nome_diff theta_00_def
    by simp
  also have "ϑ10(0) * to_nome (-τ/4) / ... = -ϑ10(0) / ϑ00(0) * to_nome
(-τ/4 - 3/4 * τ + τ)"
    unfolding to_nome_diff to_nome_add by (simp add: field_simps)
  also have "... ^ 2 = ϑ10(0) ^ 2 / ϑ00(0) ^ 2"
    by (simp add: power_divide)

```

```

    also have "(of_real pi *  $\vartheta_{10}(0)$  *  $\vartheta_{00}(0)$  /  $\omega_1$ )2 * ( $\vartheta_{10}(0)$ )2 /  $\vartheta_{00}(0)$ 2)
= (pi /  $\omega_1$ )2 *  $\vartheta_{10}(0)$ 4"
    by (simp add: field_simps rel_def uminus_in_lattice_iff to_nome_minus
theta_00_eq_0_iff)
    finally show e32: " $e_3 - e_2 = (\text{pi} / \omega_1)^2 * \vartheta_{10}(0)^4$ " .

    have "( $e_1 - e_2$ ) - ( $e_3 - e_2$ ) = (pi /  $\omega_1$ )2 * ( $\vartheta_{00}(0)$ 4 -  $\vartheta_{10}(0)$ 4)"
    unfolding e32 e12 by (simp add: field_simps)
    also have " $\vartheta_{00}(0)$ 4 -  $\vartheta_{10}(0)$ 4 =  $\vartheta_{01}(0)$ 4"
    using jacobi_theta_xy_0_pow4_complex[of  $\tau$ ] Im_ratio_pos
    by (simp add: theta_00_def theta_01_def theta_10_def algebra_simps)
    finally show e13: " $e_1 - e_3 = (\text{pi} / \omega_1)^2 * \vartheta_{01}(0)^4$ "
    by simp

    have e3_eq: " $e_3 = -(e_1 + e_2)$ "
    using sum_e123_0 by (Groebner_Basis.algebra)
    have " $e_1 = ((e_1 - e_2) + (e_1 - e_3)) / 3$ "
    by (simp add: algebra_simps e3_eq)
    also have "... = (pi /  $\omega_1$ )2 / 3 * ( $\vartheta_{00}(0)$ 4 +  $\vartheta_{01}(0)$ 4)"
    unfolding e12 e13 by (simp add: algebra_simps add_divide_distrib)
    finally show e1: " $e_1 = (\text{pi} / \omega_1)^2 / 3 * (\vartheta_{00}(0)^4 + \vartheta_{01}(0)^4)$ "
    .

    have e2_eq: " $e_2 = -(e_1 + e_3)$ "
    using sum_e123_0 by (Groebner_Basis.algebra)
    have " $e_3 = ((e_1 - e_2) - 2 * (e_1 - e_3)) / 3$ "
    by (simp add: algebra_simps e2_eq)
    also have "... = (pi /  $\omega_1$ )2 / 3 * ( $\vartheta_{00}(0)$ 4 - 2 *  $\vartheta_{01}(0)$ 4)"
    unfolding e12 e13 by (simp add: algebra_simps diff_divide_distrib)
    also have " $\vartheta_{00}(0)$ 4 - 2 *  $\vartheta_{01}(0)$ 4 =  $\vartheta_{10}(0)$ 4 -  $\vartheta_{01}(0)$ 4"
    using jacobi_theta_xy_0_pow4_complex[of  $\tau$ , symmetric] Im_ratio_pos
    by (simp add: theta_00_def theta_01_def theta_10_def)
    finally show e3: " $e_3 = (\text{pi} / \omega_1)^2 / 3 * (\vartheta_{10}(0)^4 - \vartheta_{01}(0)^4)$ " .

    show e2: " $e_2 = -(\text{pi} / \omega_1)^2 / 3 * (\vartheta_{00}(0)^4 + \vartheta_{10}(0)^4)$ "
    unfolding e2_eq e1 e3 by (simp add: field_simps)

    have "modulus = ( $e_3 - e_2$ ) / ( $e_1 - e_2$ )"
    by (simp add: modulus_def)
    also have "... =  $\vartheta_{10}(0)$ 4 /  $\vartheta_{00}(0)$ 4"
    unfolding e32 e12 by simp
    finally show "modulus =  $\vartheta_{10}(0)$ 4 /  $\vartheta_{00}(0)$ 4" .
qed

```

Using this, we also obtain an expression of \wp purely in terms of theta functions. This immediately shows that $\wp(z, \tau)$ (which we have not defined yet) is holomorphic in both z and τ .

lemma weierstrass_fun_conv_theta':

```

    assumes "z ∉ Λ"
    shows "∅ z = (pi / ω1)^2 * (-1/3 * (∅00(0)^4 + ∅10(0)^4) + (∅10(0)
* ∅00(0))^2 * ∅01(z)^2 / ∅11(z)^2)"
    by (subst weierstrass_fun_conv_theta[OF assms], subst e2_conv_theta)
        (simp_all add: field_simps)

```

end

```

context std_complex_lattice
begin

```

```

sublocale complex_lattice_Im_pos 1 τ
  rewrites "ratio = τ"
  by unfold_locales (auto simp: ratio_def Im_τ_pos)

```

end

```

unbundle no_jacobi_theta_nw_notation

```

end

12 Eisenstein series and related invariants as modular forms

```

theory Basic_Modular_Forms
imports
  Eisenstein_Series Modular_Fundamental_Region
  Elliptic_Functions_Library FPS_Homomorphism
  "Kummer_Congruence.Kummer_Library"
  Complex_Lattices_Theta
begin

```

```

lemma zeta_of_nat_eq_0_iff: "zeta (of_nat n) = 0 ⟷ n = 1"

```

```

proof -

```

```

  consider "n = 0" | "n = 1" | "n ≥ 2"

```

```

    by linarith

```

```

  thus ?thesis

```

```

proof cases

```

```

  assume "n ≥ 2"

```

```

  hence "zeta (of_nat n) ≠ 0"

```

```

    by (intro zeta_Re_gt_1_nonzero) auto

```

```

  with <n ≥ 2> show ?thesis

```

```

    by auto

```

```

qed (auto simp: zeta_1)

```

```

qed

```

unbundle modgrp_notation

In a previous section we defined the Eisenstein series g_k , the modular discriminant Δ , and Klein's invariant J in the context of a fixed complex lattice $\Lambda(\omega_1, \omega_2)$.

In this section, we will look at them for the lattice $\Lambda(1, z)$ with variable $z \in \mathbb{C} \setminus \mathbb{R}$. Since $\Lambda(1, z) = \Lambda(1, -z)$, all of these notions are symmetric with respect to negation of z , so we will often assume $\text{Im}(z) > 0$, as is common in the literature.

We will show that all the above notions satisfy simple functional equations with respect to the modular group, namely if $h(z) = \frac{az+b}{cz+d}$ then $f(h(z)) = (cz+d)^k f(z)$ for some integer k specific to f (the *weight* of f).

Meromorphic functions that satisfy such a functional equation and additionally have a meromorphic Fourier expansion at $q = 0$ (i.e. $z \rightarrow i\infty$) are called *meromorphic modular forms*. This notion will be introduced more formally in a future AFP entry, but we already show everything that is required to see that G_k (for $k \geq 3$), Δ , and J are meromorphic modular forms of weight k , 12, and 0, respectively.

12.1 Eisenstein series

First, we look at the Eisenstein series $G_k(z)$, which we define to be the Eisenstein series of the lattice generated by 1 and z . For the case where 1 and z are collinear (i.e. z lying on the real line), we return 0 by convention.

definition `Eisenstein_G` :: "nat \Rightarrow complex \Rightarrow complex" where

```
"Eisenstein_G k z = (if z  $\in$   $\mathbb{R}$  then 0 else complex_lattice.eisenstein_series 1 z k)"
```

lemma (in `complex_lattice`) `eisenstein_series_eq_Eisenstein_G`:

```
"eisenstein_series k = Eisenstein_G k ( $\omega_2 / \omega_1$ ) /  $\omega_1$  ^ k"
```

proof -

```
interpret complex_lattice_stretch  $\omega_1$   $\omega_2$  "1 /  $\omega_1$ "
```

```
by standard auto
```

```
have "stretched.eisenstein_series k =  $\omega_1$  ^ k * eisenstein_series k"
```

```
unfolding eisenstein_series_stretch by (simp add: power_int_minus field_simps)
```

```
also have "stretched.eisenstein_series k = Eisenstein_G k ( $\omega_2 / \omega_1$ )"
```

```
using stretched.fundpair unfolding Eisenstein_G_def fundpair_def by simp
```

```
finally show ?thesis
```

```
by simp
```

qed

lemma (in `complex_lattice`) `invariant_g2_eq_Eisenstein_G`:

```
"invariant_g2 = 60 * Eisenstein_G 4 ( $\omega_2 / \omega_1$ ) /  $\omega_1$  ^ 4"
```

```

unfolding invariant_g2_def eisenstein_series_eq_Eisenstein_G by simp

lemma (in complex_lattice) invariant_g3_eq_Eisenstein_G:
  "invariant_g3 = 140 * Eisenstein_G 6 (ω2 / ω1) / ω1 ^ 6"
  unfolding invariant_g3_def eisenstein_series_eq_Eisenstein_G by simp

lemma Eisenstein_G_real_eq_0 [simp]: "z ∈ ℝ ⇒ Eisenstein_G k z = 0"
  by (simp add: Eisenstein_G_def)

lemma Eisenstein_G_0 [simp]:
  assumes [simp]: "z ∉ ℝ"
  shows "Eisenstein_G 0 z = -1"
proof -
  interpret complex_lattice 1 z
  by unfold_locales (auto simp: fundpair_def)
  show ?thesis
  by (auto simp: Eisenstein_G_def)
qed

lemma Eisenstein_G_cnj: "Eisenstein_G k (cnj z) = cnj (Eisenstein_G k z)"
proof (cases "z ∈ ℝ")
  case False
  interpret complex_lattice 1 z
  using False by unfold_locales (auto simp: fundpair_def)
  interpret complex_lattice_cnj 1 z ..
  show ?thesis
  using eisenstein_series_cnj[of k] eisenstein_series_eq_Eisenstein_G[of k]
  cnj.eisenstein_series_eq_Eisenstein_G[of k] by simp
qed auto

lemma Eisenstein_G_odd [simp]:
  assumes "odd k"
  shows "Eisenstein_G k z = 0"
proof (cases "z ∈ ℝ")
  case [simp]: False
  interpret complex_lattice 1 z
  by unfold_locales (auto simp: fundpair_def)
  show ?thesis using assms
  by (auto simp: Eisenstein_G_def)
qed auto

lemma Eisenstein_G_uminus: "Eisenstein_G k (-z) = Eisenstein_G k z"
proof (cases "z ∈ ℝ")
  case False
  interpret lattice1: complex_lattice 1 z
  by standard (use False in <auto simp: fundpair_def>)
  interpret lattice2: complex_lattice 1 "-z"

```

```

    by standard (use False in <auto simp: fundpair_def>)
  have "lattice1.eisenstein_series k = lattice2.eisenstein_series k"
    unfolding lattice1.eisenstein_series_def lattice2.eisenstein_series_def
    by (auto simp: lattice1.of_ω12_coords_def lattice2.of_ω12_coords_def
        intro!: infsum_reindex_bij_witness[of "-{0}" uminus uminus])
  thus ?thesis
    by (simp add: Eisenstein_G_def)
qed (auto simp: Eisenstein_G_def)

lemma
  assumes "k ≥ 3" "(z::complex) ∉ ℝ"
  shows abs_summable_Eisenstein_G:
    "(λ(m,n). 1 / norm (of_int m + of_int n * z) ^ k) summable_on
  (-{(0,0)})"
  and summable_Eisenstein_G:
    "(λ(m,n). 1 / (of_int m + of_int n * z) ^ k) summable_on (-{(0,0)})"
  and has_sum_Eisenstein_G:
    "((λ(m,n). 1 / (of_int m + of_int n * z) ^ k) has_sum Eisenstein_G
  k z) (-{(0,0)})"
proof -
  from assms interpret complex_lattice 1 z
  by unfold_locales (auto simp: fundpair_def)
  have "(λω. 1 / norm ω ^ k) summable_on lattice0"
  by (rule eisenstein_series_norm_summable) (use assms in auto)
  also have "?this ⟷ (λ(m,n). 1 / norm (of_int m + of_int n * z) ^
  k) summable_on (-{(0,0)})"
  by (subst summable_on_reindex_bij_betw[OF bij_betw_lattice0', symmetric])
  (simp_all add: of_ω12_coords_def case_prod_unfold)
  finally show "(λ(m,n). 1 / norm (of_int m + of_int n * z) ^ k) summable_on
  (-{(0,0)})" .

  have "((λω. 1 / ω ^ k) has_sum G k) lattice0"
  by (rule eisenstein_series_has_sum) (use assms in auto)
  also have "?this ⟷ ((λ(m,n). 1 / (of_int m + of_int n * z) ^ k) has_sum
  eisenstein_series k) (-{(0,0)})"
  by (subst has_sum_reindex_bij_betw[OF bij_betw_lattice0', symmetric])
  (simp_all add: of_ω12_coords_def case_prod_unfold)
  finally show "((λ(m,n). 1 / (of_int m + of_int n * z) ^ k) has_sum Eisenstein_G
  k z) (-{(0,0)})"
  unfolding eisenstein_series_eq_Eisenstein_G by simp
  thus "(λ(m,n). 1 / (of_int m + of_int n * z) ^ k) summable_on (-{(0,0)})"
  by (rule has_sum_imp_summable)
qed

lemma Eisenstein_G_analytic [analytic_intros]:
  assumes "f analytic_on A" "∧z. z ∈ A ⟹ odd k ∨ f z ∉ ℝ"
  shows "(λz. Eisenstein_G k (f z)) analytic_on A"
proof (cases "k = 0 ∨ odd k")

```

```

case True
thus ?thesis
proof
  assume [simp]: "k = 0"
  have "(λ_. -1) holomorphic_on -ℝ"
    by (rule holomorphic_intros)
  also have "?this ↔ Eisenstein_G k holomorphic_on -ℝ"
    by (rule holomorphic_cong) auto
  finally have ana: "Eisenstein_G k analytic_on -ℝ"
    by (subst analytic_on_open) (auto intro!: closed_complex_Reals)
  have "(Eisenstein_G k ∘ f) analytic_on A"
    by (rule analytic_on_compose_gen[OF _ ana]) (use assms in auto)
  thus ?thesis
    by (simp add: o_def)
qed auto
next
case False
hence k: "k ≥ 2" "even k"
  by auto
define g :: "complex ⇒ complex" where
  "g = (λz. 2 * (zeta k + (2 * i * pi) ^ k / fact (k - 1) *
    lambert (λn. of_nat n ^ (k-1)) (to_q 1 z)))"

have "g holomorphic_on {z. Im z > 0}"
  unfolding g_def by (auto intro!: holomorphic_intros simp: lambert_conv_radius_power_of)
also have "?this ↔ Eisenstein_G k holomorphic_on {z. Im z > 0}"
proof (intro holomorphic_cong refl)
  fix z assume z: "z ∈ {z. Im z > 0}"
  interpret std_complex_lattice z
  by standard (use z in auto)
  show "g z = Eisenstein_G k z"
    unfolding g_def using eisenstein_series_conv_lambert[of k] k z
    by (simp add: Eisenstein_G_def complex_is_Real_iff)
qed
finally have "Eisenstein_G k holomorphic_on {z. 0 < Im z}" .
hence ana: "Eisenstein_G k analytic_on {z. 0 < Im z}"
  by (subst analytic_on_open) (simp_all add: open_halfspace_Im_gt)

have "(Eisenstein_G k ∘ uminus) analytic_on {z. Im z < 0}"
  by (rule analytic_on_compose_gen[OF _ ana]) (auto intro!: analytic_intros)
also have "Eisenstein_G k ∘ uminus = Eisenstein_G k"
  by (auto simp: Eisenstein_G_uminus)
finally have "Eisenstein_G k analytic_on ({z. 0 < Im z} ∪ {z. Im z <
0})"
  by (subst analytic_on_Un) (use ana in auto)
also have "... = -ℝ"
  by (auto simp: complex_is_Real_iff)
finally have ana2: "Eisenstein_G k analytic_on -ℝ" .

```

```

have "(Eisenstein_G k ∘ f) analytic_on A"
  by (rule analytic_on_compose_gen[OF _ ana2]) (use assms False in auto)
thus ?thesis
  by (simp add: o_def)
qed

lemma Eisenstein_G_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "∧z. z ∈ A ⇒ odd k ∨ f z ∉ ℝ"
  shows "(λz. Eisenstein_G k (f z)) holomorphic_on A"
proof -
  from assms(1) have "(Eisenstein_G k ∘ f) holomorphic_on A"
    by (rule holomorphic_on_compose)
    (use assms in <auto intro!: analytic_imp_holomorphic analytic_intros>)
  thus ?thesis
    by (simp add: o_def)
qed

lemma Eisenstein_G_meromorphic [meromorphic_intros]:
  assumes "f analytic_on A" "∧z. z ∈ A ⇒ odd k ∨ f z ∉ ℝ"
  shows "(λz. Eisenstein_G k (f z)) meromorphic_on A"
  by (rule meromorphic_on_compose[OF _ assms(1) order.refl])
    (use assms(2) in <auto intro!: analytic_intros analytic_on_imp_meromorphic_on>)

lemma tendsto_Eisenstein_G [tendsto_intros]:
  assumes "(f ⟶ z) F" "odd k ∨ z ∉ ℝ"
  shows "((λz. Eisenstein_G k (f z)) ⟶ Eisenstein_G k z) F"
proof -
  have "Eisenstein_G k analytic_on {z}"
    by (rule analytic_intros) (use assms in auto)
  with assms(1) show ?thesis
    by (metis analytic_at_imp_isCont isCont_tendsto_compose)
qed

lemma continuous_Eisenstein_G [continuous_intros]:
  "continuous (at x within A) f ⇒ odd k ∨ f x ∉ ℝ ⇒
  continuous (at x within A) (λz. Eisenstein_G k (f z))"
unfolding continuous_def
using tendsto_Eisenstein_G[of f "f x" "at x within A"]
by (cases "at x within A = bot") (auto simp: Lim_ident_at)

lemma continuous_on_Eisenstein_G [continuous_intros]:
  "continuous_on A f ⇒ odd k ∨ (∀x∈A. f x ∉ ℝ) ⇒
  continuous_on A (λz. Eisenstein_G k (f z))"
  by (rule continuous_on_compose2[of "if even k then ¬ℝ else UNIV" _
  _ f])
    (auto intro!: holomorphic_on_imp_continuous_on holomorphic_intros)

```

We can also lift our earlier results about the Fourier expansion of G_k to this new viewpoint of $G_k(z)$. This is Theorem 1.18 in Apostol's book.

```

theorem Eisenstein_G_fourier_expansion:
  fixes z :: complex and k :: nat
  assumes z: "Im z > 0"
  assumes k: "k ≥ 2" "even k"
  shows "Eisenstein_G k z =
    2 * (zeta k + (2*i*pi) ^ k / fact (k - 1) * lambert (λn. of_nat
n ^ (k - 1)) (to_q 1 z))"
proof -
  interpret std_complex_lattice z
  by standard fact
  have "Eisenstein_G k z = eisenstein_series k"
  using eisenstein_series_eq_Eisenstein_G[of k] by simp
  also have "... = 2 * (zeta k + (2*i*pi) ^ k / fact (k - 1) *
    lambert (λn. of_nat n ^ (k - 1)) (to_q 1 z))"
  by (rule eisenstein_series_conv_lambert[OF k])
  finally show ?thesis .
qed

```

We explicitly define the q -expansion of G_n as a holomorphic function on the unit disc.

```

definition q_Eisenstein_G :: "nat ⇒ complex ⇒ complex" where
  "q_Eisenstein_G k q = (if k = 0 then -1 else if odd k then 0 else
    2 * (zeta k + (2*i*pi) ^ k / fact (k - 1) * lambert (λn. of_nat n
^ (k - 1)) q))"

```

```

lemma holomorphic_on_q_Eisenstein_G [holomorphic_intros]:
  "f holomorphic_on A ⇒ (λz. z ∈ A ⇒ norm (f z) < 1) ⇒
    (λz. q_Eisenstein_G k (f z)) holomorphic_on A"
  unfolding q_Eisenstein_G_def
  by (cases "k = 0"; cases "even k")
    (force intro!: holomorphic_intros simp: lambert_conv_radius_power_of_nat)+

```

```

lemma analytic_on_q_Eisenstein_G [analytic_intros]:
  "f analytic_on A ⇒ (λz. z ∈ A ⇒ norm (f z) < 1) ⇒
    (λz. q_Eisenstein_G k (f z)) analytic_on A"
  unfolding q_Eisenstein_G_def
  by (cases "k = 0"; cases "even k")
    (force intro!: analytic_intros simp: lambert_conv_radius_power_of_nat)+

```

```

lemma tendsto_q_Eisenstein_G [tendsto_intros]:
  assumes "(f ⟶ q) F" "norm q < 1"
  shows "((λz. q_Eisenstein_G k (f z)) ⟶ q_Eisenstein_G k q) F"
proof -
  have "q_Eisenstein_G k analytic_on {q}"
  by (rule analytic_intros) (use assms in auto)
  with assms(1) show ?thesis
  by (metis analytic_at_imp_isCont isCont_tendsto_compose)
qed

```

```

lemma continuous_q_Eisenstein_G [continuous_intros]:
  "continuous (at x within A) f  $\implies$  norm (f x) < 1  $\implies$ 
    continuous (at x within A) ( $\lambda z$ . q_Eisenstein_G k (f z))"
  unfolding continuous_def
  using tendsto_q_Eisenstein_G[of f "f x" "at x within A"]
  by (cases "at x within A = bot") (auto simp: Lim_ident_at)

```

The following is the formal q -expansion of G_n .

```

definition fps_Eisenstein_G :: "nat  $\Rightarrow$  complex fps" where
  "fps_Eisenstein_G k = (if k = 0 then -1 else if odd k then 0 else
    2 * (fps_const (zeta k) +
      fps_const ((2*i*pi) ^ k / fact (k-1)) * Abs_fps (divisor_sigma
(k-1))))"

```

```

lemma has_fps_expansion_lambert_sigma_power [fps_expansion_intros]:
  "lambert ( $\lambda n$ . of_nat n ^ k :: 'a :: {nat_power_normed_field,banach})
has_fps_expansion
  Abs_fps ( $\lambda n$ . of_nat (divisor_sigma k n))"
proof (rule has_fps_expansionI)
  let ?F = "Abs_fps ( $\lambda n$ . of_nat (divisor_sigma k n))"
  have "eventually ( $\lambda q$ . q  $\in$  ball 0 1) (nhds (0::'a))"
    by (rule eventually_nhds_in_open) auto
  thus "eventually ( $\lambda q$ . ( $\lambda n$ . fps_nth ?F n * q ^ n) sums lambert ( $\lambda n$ . of_nat
n ^ k) q) (nhds (0::'a))"
  proof eventually_elim
    case (elim q)
    thus ?case
      using divisor_sigma_powser_conv_lambert[of q "of_nat k"] by (simp
add: divisor_sigma_of_nat)
  qed
qed

```

```

lemma has_fps_expansion_q_Eisenstein_G [fps_expansion_intros]:
  "q_Eisenstein_G k has_fps_expansion fps_Eisenstein_G k"
proof (cases "k = 0  $\vee$  odd k")
  case True
  thus ?thesis
    by (auto simp: q_Eisenstein_G_def [abs_def] fps_Eisenstein_G_def intro:
fps_expansion_intros)
next
  case k: False
  have "( $\lambda q$ . 2 * (zeta k + (2*i*pi) ^ k / fact (k - 1)) * lambert ( $\lambda n$ .
of_nat n ^ (k - 1)) q)
    has_fps_expansion 2 * (fps_const (zeta k) +
      fps_const ((2*i*pi) ^ k / fact (k-1)) * Abs_fps
(divisor_sigma (k-1)))"
    by (intro fps_expansion_intros)
  thus ?thesis using k
    by (auto simp: q_Eisenstein_G_def [abs_def] fps_Eisenstein_G_def simp

```

```
flip: divisor_sigma_of_nat)
qed
```

```
lemma Eisenstein_G_conv_q:
  assumes z: "Im z > 0"
  shows "Eisenstein_G k z = q_Eisenstein_G k (to_q 1 z)"
proof -
  from assms have [simp]: "z ∉ ℝ"
  by (auto elim!: Reals_cases)
  consider "k = 0" | "odd k" | "even k" "k ≥ 2"
  by force
  thus ?thesis
  proof cases
    assume [simp]: "k = 0"
    thus ?thesis by (use z in <auto simp: q_Eisenstein_G_def>)
  next
    assume "odd k"
    moreover from this have "k > 0"
    by (auto elim!: oddE)
    ultimately show ?thesis
    by (auto simp: q_Eisenstein_G_def)
  next
    assume "even k" "k ≥ 2"
    thus ?thesis
    using Eisenstein_G_fourier_expansion[of z k] z by (auto simp: q_Eisenstein_G_def)
  qed
qed
```

We translate our machinery for deducing representations of G_n in terms of G_4 and G_6 to work for our new views of G_n , including its q -series.

```
lemma eisenstein_series_poly_Eisenstein_G:
  "poly2 (map_poly2 of_rat (eisenstein_series_poly n))
  (Eisenstein_G 4 z) (Eisenstein_G 6 z) = Eisenstein_G (2 * n + 4)
  z"
proof (cases "z ∈ ℝ")
  case True
  thus ?thesis
  by (simp add: Eisenstein_G_def map_poly2_def hom_distrib poly2_def
  poly_0_coeff_0 coeff_map_poly)
next
  case False
  interpret complex_lattice 1 z
  by standard (use False in <auto simp: fundpair_def>)
  show ?thesis
  using False by (simp add: Eisenstein_G_def eisenstein_series_poly_correct)
qed
```

```
lemma eisenstein_series_poly_q_Eisenstein_G:
  assumes "norm q < 1"
```

```

shows "poly2 (map_poly2 of_rat (eisenstein_series_poly n))
      (q_Eisenstein_G 4 q) (q_Eisenstein_G 6 q) = q_Eisenstein_G
(2 * n + 4) q"
      (is "?lhs q = ?rhs q")
proof -
define f where "f = (λq. ?lhs q - ?rhs q)"
have f_eq_0: "f q = 0" if q: "norm q < 1" "q ≠ 0" for q
proof -
  show ?thesis
    using eisenstein_series_poly_Eisenstein_G[of n "of_q 1 q"] q
    by (simp add: f_def Eisenstein_G_conv_q Im_of_q divide_less_0_iff)
qed

show ?thesis
proof (cases "q = 0")
  case True
  have "isCont f 0"
    unfolding f_def by (intro continuous_intros continuous_within_poly2)
auto
  hence "f -0→ f 0"
    by (rule isContD)
  moreover have "eventually (λq. q ∈ ball 0 1 - {0}) (at (0::complex))"
    by (rule eventually_at_in_open) auto
  hence "eventually (λq. f q = 0) (at 0)"
    by eventually_elim (use f_eq_0 in auto)
  hence "f -0→ 0"
    using tendsto_eventually by blast
  ultimately have "f 0 = 0"
    by (rule tendsto_unique[rotated]) auto
  thus ?thesis
    using True by (simp add: f_def)
qed (use f_eq_0[of q] assms in <auto simp: f_def>)
qed

lemma eisenstein_series_poly_fps_Eisenstein_G:
  "poly2 (map_poly2 (fps_const ∘ of_rat) (eisenstein_series_poly n))
    (fps_Eisenstein_G 4) (fps_Eisenstein_G 6) = fps_Eisenstein_G (2 *
n + 4)" (is "?lhs = ?rhs")
proof -
define F where "F = poly2 (map_poly2 fps_const (map_poly2 of_rat (eisenstein_series_poly
n)))
    (fps_Eisenstein_G 4) (fps_Eisenstein_G 6) - fps_Eisenstein_G (2 *
n + 4)"
define f where "f = (λq. poly2 (map_poly2 of_rat (eisenstein_series_poly
n))
    (q_Eisenstein_G 4 q) (q_Eisenstein_G 6 q) - q_Eisenstein_G
(2 * n + 4) q)"
have "f has_fps_expansion F"
  unfolding f_def F_def by (intro fps_expansion_intros)

```

```

moreover have "eventually ( $\lambda q. q \in \text{ball } 0 \ 1$ ) (nhds (0::complex))"
  by (rule eventually_nhds_in_open) auto
hence "eventually ( $\lambda q. f \ q = 0$ ) (nhds 0)"
proof eventually_elim
  case (elim q)
  thus ?case using eisenstein_series_poly_q_Eisenstein_G[of q n] by
(simp add: f_def)
qed
ultimately have "F = 0"
  by (metis fps_expansion_unique_complex has_fps_expansion_0_iff)
also have "F = ?lhs - ?rhs"
  unfolding F_def by (subst map_poly2_compose) auto
finally show ?thesis by simp
qed

```

As an application, we show that the identities for G_n give rise to related identities on the divisor function by looking at the coefficients of the q -expansions, i.e. $G_8 = \frac{3}{7}G_4^2$ gives rise to

$$\sigma_7(n) = \sigma_3(n) + 120 \sum_{k=1}^{n-1} \sigma_3(k)\sigma_3(n-k) .$$

```

theorem divisor_sigma_7_conv_3:
  "divisor_sigma 7 n =
    divisor_sigma 3 n + 120 * ( $\sum_{k=1..<n. \text{divisor\_sigma } 3 \ k * \text{divisor\_sigma } 3 \ (n - k)}$ )"
proof (cases "n > 0")
  case n: True
  define G where "G = fps_Eisenstein_G"
  define a where "a = fps_const (3/7::complex)"
  define b where "b = fps_const (zeta 4)"
  define c where "c = (32 / 315 * pi ^ 8)"
  write divisor_sigma ("σ")

  have "G 8 = a * G 4 ^ 2"
    using eisenstein_series_poly_fps_Eisenstein_G[of 2]
    by (simp add: a_def eisenstein_series_poly_rec numeral_poly G_def
power2_eq_square of_rat_divide mult_ac)
  hence "Re (fps_nth (G 8) n) / c = Re (fps_nth (a * G 4 ^ 2) n) / c"
    by (rule arg_cong)
  also have "fps_nth (G 8) n = of_real (32 / 315 * pi ^ 8 * σ 7 n)"
    using n by (simp add: a_def G_def fps_Eisenstein_G_def fact_numeral
power_mult_distrib
flip: divisor_sigma_of_real)
  also have "a * G 4 ^ 2 = 4 * a * (b + fps_const (240 * zeta 4) * Abs_fps
(σ 3))^2"
    by (simp add: G_def fps_Eisenstein_G_def fact_numeral zeta_even_numeral

```

```

      b_def power2_eq_square algebra_simps)
also have "fps_const (240 * zeta 4) = 240 * b"
  by (simp add: b_def)
also have "4 * a * (b + 240 * b * Abs_fps (σ 3))^2 =
  4 * a * b^2 * (1 + 240 * Abs_fps (σ 3))^2"
  by (simp add: power2_eq_square algebra_simps)
also have "fps_nth ... n =
  of_real pi ^ 8 / 4725 * fps_nth ((1 + 240 * Abs_fps (σ
3)) ^ 2) n"
  by (simp add: a_def b_def mult.assoc zeta_even_numeral power_mult_distrib
fact_numeral power_divide)
also have "fps_nth ((1 + 240 * Abs_fps (σ (3::complex))) ^ 2) n =
  480 * σ 3 n + 57600 * fps_nth (Abs_fps (σ 3) ^ 2) n"
  using n by (simp add: power2_eq_square ring_distrib)
also have "fps_nth (Abs_fps (σ (3::complex)) ^ 2) n = (∑ i=0..n. σ
3 i * σ 3 (n - i))"
  by (simp add: power2_eq_square fps_mult_nth)
also have "... = (∑ i=1..<n. σ 3 i * σ 3 (n - i))"
  by (rule sum.mono_neutral_right) (auto simp: Suc_le_eq)
finally have "(σ 7 n :: real) = σ 3 n + 120 * (∑ k=1..<n. σ 3 k * σ
3 (n - k))"
  by (simp add: c_def)

hence "real (σ 7 n) = real (σ 3 n + 120 * (∑ k=1..<n. σ 3 k * σ 3 (n
- k)))"
  unfolding of_nat_add by (simp flip: divisor_sigma_of_nat)
hence "σ (7::nat) n = σ 3 n + 120 * (∑ k=1..<n. σ 3 k * σ 3 (n - k))"
  by (subst (asm) of_nat_eq_iff)
hence "of_nat (σ 7 n) = (of_nat (σ 3 n + 120 * (∑ k=1..<n. σ 3 k *
σ 3 (n - k))) :: 'a)"
  by (rule arg_cong)
thus ?thesis
  by (simp flip: divisor_sigma_of_nat)
qed auto

```

We now show how the modular group acts on G_k . The case $k = 2$ is more complicated and will be dealt with later.

```

theorem Eisenstein_G_apply_modgrp:
  assumes "k ≠ 2"
  shows "Eisenstein_G k (apply_modgrp f z) = automorphy_factor f z ^
k * Eisenstein_G k z"
proof (cases "z ∈ ℝ")
case False
interpret complex_lattice 1 z
  by standard (use False in <auto simp: fundpair_def>)
interpret complex_lattice_apply_modgrp 1 z f ..

have "Eisenstein_G k (apply_modgrp f z) = Eisenstein_G k (ω2' / ω1')"
  unfolding moebius_def modgrp.φ_def ω1'_def ω2'_def

```

```

    by (simp add: apply_modgrp_altdef moebius_def)
  also have "... = automorphy_factor f z ^ k * transformed.eisenstein_series
k"
    by (subst transformed.eisenstein_series_eq_Eisenstein_G)
      (auto simp:  $\omega_1'$ _def power_int_minus field_simps automorphy_factor_altdef)
  also have "... = automorphy_factor f z ^ k * eisenstein_series k"
    using assms by simp
  also have "... = automorphy_factor f z ^ k * Eisenstein_G k z"
    by (subst eisenstein_series_eq_Eisenstein_G) auto
  finally show ?thesis .
qed auto

lemma Eisenstein_G_plus_int: "Eisenstein_G k (z + of_int m) = Eisenstein_G
k z"
proof (cases "k = 2")
  case False
  thus ?thesis
    using Eisenstein_G_apply_modgrp[of k "shift_modgrp m" z] by simp
next
  case [simp]: True
  have *: "Eisenstein_G 2 (z + of_int m) = Eisenstein_G 2 z" if z: "Im
z > 0" for z m
    using z by (simp add: Eisenstein_G_fourier_expansion to_q_add)

  show ?thesis
  proof (cases "Im z" "0 :: real" rule: linorder_cases)
    case greater
    thus ?thesis by (simp add: *)
  next
    case equal
    hence "z  $\in$  R"
      by (auto simp: complex_is_Real_iff)
    thus ?thesis
      by simp
  next
    case less
    have "Eisenstein_G k (z + of_int m) = Eisenstein_G 2 (-(z + of_int
m))"
      by (subst Eisenstein_G_uminus) auto
    also have "... = Eisenstein_G 2 (-z + of_int (-m))"
      by simp
    also have "... = Eisenstein_G 2 (-z)"
      using less by (intro *) auto
    also have "... = Eisenstein_G k z"
      by (simp add: Eisenstein_G_uminus)
    finally show ?thesis .
  qed
qed

```

lemma Eisenstein_G_plus1: "Eisenstein_G k (z + 1) = Eisenstein_G k z"
 using Eisenstein_G_plus_int[of k z 1] by simp

12.2 The monomials of the Eisenstein polynomial

The polynomial $P(X, Y)$ that gives us $E_{2n+4} = P(E_4, E_6)$ only has monomials of the form $X^i Y^j$ with $4i + 6j = 2n + 4$.

definition coeff2 where "coeff2 p m n = poly.coeff (poly.coeff p n) m"

definition is_46_poly :: "nat \Rightarrow 'a :: comm_ring_1 poly poly \Rightarrow bool" where
 "is_46_poly n p \longleftrightarrow ($\forall i j. \text{coeff2 } p \ i \ j \neq 0 \longrightarrow 4 * i + 6 * j = n$)"

lemma is_46_poly_induct [consumes 1, case_names zero add smult monom]:
 assumes "is_46_poly n p"
 assumes "P 0"
 assumes " $\bigwedge p q. P p \implies P q \implies P (p + q)$ "
 assumes " $\bigwedge p c. P p \implies P (\text{Polynomial.smult } [:c:] p)$ "
 assumes " $\bigwedge i j. 4 * i + 6 * j = n \implies P (\text{Polynomial.monom } (\text{Polynomial.monom } 1 \ i) \ j)$ "
 shows "P p"

proof -

define I where "I = (SIGMA j:{.. $\text{degree } p$ }. {i \in {.. $\text{degree } (\text{poly.coeff } p \ j)$ }. coeff2 p i j \neq 0})"
 have sum: "P ($\sum x \in I. f \ x$)" if " $\bigwedge x. x \in I \implies P (f \ x)$ " for f
 using that by (induction I rule: infinite_finite_induct) (auto intro!: assms)

have "p = ($\sum j \leq \text{degree } p. \sum i \leq \text{degree } (\text{poly.coeff } p \ j). \text{Polynomial.monom } (\text{Polynomial.monom } (\text{coeff2 } p \ i \ j) \ i) \ j)$ "
 by (subst poly_as_sum_of_monoms [symmetric], subst (2) poly_as_sum_of_monoms [symmetric])
 (simp add: monom_sum coeff2_def)

also have "... = ($\sum (i,j) \in (\text{SIGMA } i:\{.. $\text{degree } p$ }. \{.. $\text{degree } (\text{poly.coeff } p \ i)\}$ }).$

Polynomial.monom (Polynomial.monom (coeff2 p j i) j) i)"

by (subst sum.Sigma) (auto simp: case_prod_unfold)

also have "... = ($\sum (i,j) \in I. \text{Polynomial.monom } (\text{Polynomial.monom } (\text{coeff2 } p \ j \ i) \ j) \ i)$ "

by (intro sum.mono_neutral_right) (auto simp: I_def)

also have "... = ($\sum (i,j) \in I. \text{Polynomial.smult } [: \text{coeff2 } p \ j \ i:] (\text{Polynomial.monom } (\text{Polynomial.monom } 1 \ j) \ i)$)"

by (intro sum.cong) (auto simp: smult_monom)

also have "P ..." unfolding case_prod_unfold

by (intro sum assms) (use assms(1) in <auto simp: I_def is_46_poly_def>)

finally show ?thesis .

qed

lemma is_46_poly_0: "is_46_poly n 0"

```

    by (force simp: is_46_poly_def coeff2_def)

lemma is_46_poly_1: "is_46_poly 0 1"
  by (force simp: is_46_poly_def coeff2_def)

lemma is_46_poly_const: "is_46_poly 0 [: [:c:] :]"
  by (auto simp: is_46_poly_def coeff2_def coeff_pCons split: nat.splits)

lemma is_46_poly_x: "is_46_poly 4 [: [: 0, 1 :] :]"
  by (auto simp: is_46_poly_def coeff2_def coeff_pCons split: nat.splits)

lemma is_46_poly_y: "is_46_poly 6 [: 0, [:1:] :]"
  by (auto simp: is_46_poly_def coeff2_def coeff_pCons split: nat.splits)

lemma is_46_poly_x_power: "m = 4 * n  $\implies$  is_46_poly m [: Polynomial.monom c n :]"
  by (auto simp: is_46_poly_def coeff2_def coeff_pCons split: nat.splits)

lemma is_46_poly_y_power: "m = 6 * n  $\implies$  is_46_poly m (Polynomial.monom [:c:] n)"
  by (auto simp: is_46_poly_def coeff2_def coeff_pCons split: nat.splits)

lemma is_46_poly_smult:
  assumes "is_46_poly n p"
  shows "is_46_poly n (Polynomial.smult [:c:] p)"
  using assms by (force simp: is_46_poly_def coeff2_def coeff_smult)

lemma is_46_poly_add:
  assumes "is_46_poly n p" "is_46_poly n q"
  shows "is_46_poly n (p + q)"
  using assms by (force simp: is_46_poly_def coeff2_def)

lemma is_46_poly_diff:
  assumes "is_46_poly n p" "is_46_poly n q"
  shows "is_46_poly n (p - q)"
  using assms by (force simp: is_46_poly_def coeff2_def)

lemma is_46_poly_uminus:
  assumes "is_46_poly n p"
  shows "is_46_poly n (-p)"
  using assms by (force simp: is_46_poly_def coeff2_def)

lemma is_46_poly_sum:
  assumes " $\bigwedge x. x \in A \implies$  is_46_poly n (p x)"
  shows "is_46_poly n ( $\sum_{x \in A. p x}$ )" using assms
  by (induction A rule: infinite_finite_induct) (auto intro: is_46_poly_add is_46_poly_0)

lemma is_46_poly_mult:

```

```

    assumes "is_46_poly n1 p" "is_46_poly n2 q" "n = n1 + n2"
    shows "is_46_poly n (p * q)"
    unfolding is_46_poly_def
  proof safe
    fix i j assume "coeff2 (p * q) i j ≠ 0"
    then obtain i' j' where *: "i' ≤ i" "j' ≤ j" "coeff2 p i' j' ≠ 0"
    "coeff2 q (i - i') (j - j') ≠ 0"
    by (auto simp: is_46_poly_def coeff_mult coeff2_def coeff_sum
        elim!: sum.not_neutral_contains_not_neutral dest!: mult_not_zero)
    with assms have "4 * i' + 6 * j' = n1" "4 * (i - i') + 6 * (j - j')
    = n2"
    unfolding is_46_poly_def by force+
    thus "4 * i + 6 * j = n"
    using *(1,2) <n = n1 + n2> by (simp add: algebra_simps)
  qed

lemma is_46_poly_power:
  assumes "is_46_poly n' p" "n = m * n'"
  shows "is_46_poly n (p ^ m)" using assms(1) unfolding assms(2)
  by (induction m) (auto intro!: is_46_poly_mult is_46_poly_1)

lemma is_46_poly_eisenstein_series_poly: "is_46_poly (2*n+4) (eisenstein_series_poly
n)"
proof (induction n rule: less_induct)
  case (less n)
  consider "n = 0" | "n = 1" | "n ≥ 2"
  by linarith
  thus ?case
  proof cases
    assume n: "n ≥ 2"
    show ?thesis using n
    by (auto intro!: is_46_poly_x is_46_poly_y is_46_poly_smult is_46_poly_sum
    is_46_poly_mult less.IH
    simp: of_nat_poly eisenstein_series_poly_rec simp flip:
    pCons_one)
  qed (use is_46_poly_x is_46_poly_y in auto)
qed

```

12.3 The normalised Eisenstein series

The literature also often defines the *normalised* Eisenstein series E_k , which is G_k divided by the constant $2\zeta(k)$. This leads to the somewhat nicer Fourier expansion

$$E_k(z) = 1 - \frac{2k}{B_k} \sum_{n=1}^{\infty} \sigma_{k-1}(n)q^n .$$

definition $Eisenstein_E :: "nat \Rightarrow complex \Rightarrow complex"$ where
 $"Eisenstein_E\ k\ z = Eisenstein_G\ k\ z / (2 * zeta\ k)"$

```

lemma of_real_of_rat [simp]: "of_real (of_rat x) = of_rat x"
  by (cases x) (auto simp: of_rat_rat hom_distrib)

lemma Eisenstein_E_0 [simp]: "z ∉ ℝ ⇒ Eisenstein_E 0 z = 1"
  by (simp add: Eisenstein_E_def)

lemma Eisenstein_E_real_eq_0 [simp]: "z ∈ ℝ ⇒ Eisenstein_E k z = 0"
  by (simp add: Eisenstein_E_def)

lemma Eisenstein_E_cnj: "Eisenstein_E k (cnj z) = cnj (Eisenstein_E k z)"
  by (simp add: Eisenstein_E_def Eisenstein_G_cnj flip: zeta_cnj)

lemma Eisenstein_E_odd [simp]:
  assumes "odd k"
  shows "Eisenstein_E k z = 0"
  using assms by (auto simp: Eisenstein_E_def elim!: oddE)

lemma Eisenstein_E_plus_int: "Eisenstein_E k (z + of_int m) = Eisenstein_E k z"
  by (auto simp: Eisenstein_E_def Eisenstein_G_plus_int complex_is_Real_iff)

lemma Eisenstein_E_plus1: "Eisenstein_E k (z + 1) = Eisenstein_E k z"
  using Eisenstein_E_plus_int[of k z 1] by simp

lemma Eisenstein_E_uminus: "Eisenstein_E k (-z) = Eisenstein_E k z"
  by (simp add: Eisenstein_E_def Eisenstein_G_uminus)

lemma Eisenstein_E_analytic [analytic_intros]:
  assumes "f analytic_on A" "∧z. z ∈ A ⇒ odd k ∨ f z ∉ ℝ"
  shows "(λz. Eisenstein_E k (f z)) analytic_on A"
proof -
  define c where "c = inverse (2 * zeta (of_nat k) :: complex)"
  have "(λz. c * Eisenstein_G k (f z)) analytic_on A"
    by (intro analytic_intros) (use assms in auto)
  also have "(λz. c * Eisenstein_G k (f z)) = (λz. Eisenstein_E k (f z))"
    by (auto simp: Eisenstein_E_def fun_eq_iff c_def field_simps)
  finally show ?thesis .
qed

lemma Eisenstein_E_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "∧z. z ∈ A ⇒ odd k ∨ f z ∉ ℝ"
  shows "(λz. Eisenstein_E k (f z)) holomorphic_on A"
proof -
  from assms(1) have "(Eisenstein_E k ∘ f) holomorphic_on A"
    by (rule holomorphic_on_compose)

```

```

      (use assms in <auto intro!: analytic_imp_holomorphic analytic_intros>)
    thus ?thesis
      by (simp add: o_def)
qed

lemma Eisenstein_E_meromorphic [meromorphic_intros]:
  assumes "f analytic_on A" "\z. z \in A \implies odd k \vee f z \notin \mathbb{R}"
  shows "(λz. Eisenstein_E k (f z)) meromorphic_on A"
  by (rule meromorphic_on_compose[OF _ assms(1) order.refl])
      (use assms(2) in <auto intro!: analytic_intros analytic_on_imp_meromorphic_on>)

lemma continuous_on_Eisenstein_E [continuous_intros]:
  "continuous_on A f \implies odd k \vee (\forall x \in A. f x \notin \mathbb{R}) \implies
    continuous_on A (λz. Eisenstein_E k (f z))"
  by (rule continuous_on_compose2[of "if even k then \mathbb{R} else UNIV" _
    _ f])
      (auto intro!: holomorphic_on_imp_continuous_on holomorphic_intros)

theorem Eisenstein_E_apply_modgrp:
  assumes "k \neq 2"
  shows "Eisenstein_E k (apply_modgrp f z) = automorphy_factor f z ^
    k * Eisenstein_E k z"
  unfolding Eisenstein_E_def by (subst Eisenstein_G_apply_modgrp) (use
  assms in auto)

The Fourier expansion of  $E_k(z)$ :

definition q_Eisenstein_E :: "nat \Rightarrow complex \Rightarrow complex" where
  "q_Eisenstein_E k q = (if k = 0 then 1 else if odd k then 0 else
    1 - 2 * of_nat k / of_real (bernoulli k) * lambert (λn. of_nat n
    ^ (k - 1)) q)"

lemma q_Eisenstein_G_conv_E: "q_Eisenstein_G k q = 2 * zeta k * q_Eisenstein_E
  k q"
proof (cases "even k \wedge k > 0")
  case True
  have "k \ge 2"
    using True by auto
  from True obtain n where n: "n > 0" and k_eq: "k = 2 * n"
    by (elim conjE evenE) auto
  define L where "L = lambert (λn. of_nat n ^ (k-1)) q"
  have "2 * zeta k * q_Eisenstein_E k q =
    2 * zeta (of_nat k) * (1 - 2 * of_nat k / of_real (bernoulli
  k) * L)"
    using True by (auto simp: q_Eisenstein_E_def L_def)
  also have "2 * of_nat k / complex_of_real (bernoulli k) =
    - ((2 * i * pi) ^ k * (of_nat k / fact k) / zeta (of_nat
  k))"
    using n by (simp add: k_eq zeta_even_nat power_mult_distrib)

```

```

also have "of_nat k / fact k = 1 / (fact (k - 1) :: complex)"
  using True by (simp add: fact_reduce)
also have "2 * zeta (of_nat k) * (1 - ((2 * i * pi) ^ k * (1 / fact
(k - 1))) / zeta (of_nat k)) * L =
  2 * (zeta (of_nat k) + (2*i*pi)^k / fact (k-1) * L)"
  using <k ≥ 2> zeta_Re_gt_1_nonzero[of "of_nat k"] by (simp add: algebra_simps)
also have "... = q_Eisenstein_G k q"
  using True by (simp add: q_Eisenstein_G_def L_def)
finally show ?thesis ..
qed (auto simp: q_Eisenstein_G_def q_Eisenstein_E_def)

lemma Eisenstein_E_conv_q:
  assumes z: "Im z > 0"
  shows "Eisenstein_E k z = q_Eisenstein_E k (to_q 1 z)"
proof (cases "k = 1")
  case False
  thus ?thesis
    using assms zeta_of_nat_eq_0_iff[of k]
    by (auto simp: Eisenstein_E_def Eisenstein_G_conv_q q_Eisenstein_G_conv_E)
qed (auto simp: q_Eisenstein_E_def)

lemma Eisenstein_E_fourier:
  assumes "Im z > 0"
  shows "Eisenstein_E k z = q_Eisenstein_E k (to_q 1 z)"
proof (cases "k ≥ 2 ∧ even k")
  case True
  obtain l where l: "k = 2 * l" "l > 0"
    using assms True by (elim conjE evenE) auto
  have "Eisenstein_E k z = 1 + (2 * i * pi) ^ k / (fact (k - 1) * zeta
k) * lambert (λn. n^(k-1)) (to_q 1 z)"
    using assms True unfolding Eisenstein_E_def
    by (subst Eisenstein_G_fourier_expansion)
    (auto simp: add_divide_distrib zeta_Re_gt_1_nonzero)
  also have "fact (k - 1) = fact k / complex_of_nat k"
    using assms True by (subst fact_reduce) auto
  also have "(2 * i * pi) ^ k / (... * zeta k) = -2 * k / bernoulli k"
    using l(2) by (auto simp: l zeta_even_nat power_mult_distrib)
  finally show ?thesis
    using assms True by (simp add: q_Eisenstein_E_def)
qed (use assms in <auto simp: Eisenstein_E_def q_Eisenstein_E_def complex_is_Real_iff
elim!: oddE Reals_cases>)

```

The formal power series corresponding to this Fourier expansion:

```

definition fps_Eisenstein_E :: "nat ⇒ complex fps" where
  "fps_Eisenstein_E k =
    (if k = 0 then 1 else if odd k then 0
     else 1 - fps_const (2 * of_nat k / bernoulli k) * Abs_fps (λn. of_nat
(divisor_sigma (k-1) n)))"

```

```

lemma fps_nth_0_Eisenstein_E [simp]: "even k  $\implies$  fps_nth (fps_Eisenstein_E k) 0 = 1"
  by (auto simp: fps_Eisenstein_E_def)

lemma fps_Eisenstein_G_conv_E:
  "fps_Eisenstein_G k = fps_const (2 * zeta k) * fps_Eisenstein_E k"
proof (cases "even k  $\wedge$  k > 0")
  case True
  have "k  $\geq$  2"
    using True by auto
  from True obtain n where n: "n > 0" and k_eq: "k = 2 * n"
    by (elim conjE evenE) auto
  define c1 where "c1 = zeta (of_nat k)"
  define c2 where "c2 = ((2 * i * pi) ^ k / fact (k - 1) / zeta (of_nat k))"

  define L where "L = Abs_fps ( $\lambda$ n. of_nat (divisor_sigma (k-1) n) :: complex)"
  have "fps_const (2 * zeta k) * fps_Eisenstein_E k =
    2 * fps_const c1 * (1 - fps_const (2 * of_nat k / of_real (bernoulli k)) * L)"
    using True by (auto simp: fps_Eisenstein_E_def L_def c1_def of_real_bernoulli)
  also have "2 * of_nat k / complex_of_real (bernoulli k) =
    - ((2 * i * pi) ^ k * (of_nat k / fact k) / zeta (of_nat k))"
    using n by (simp add: k_eq zeta_even_nat power_mult_distrib)
  also have "of_nat k / fact k = 1 / (fact (k - 1) :: complex)"
    using True by (simp add: fact_reduce)
  also have "(2 * i * pi) ^ k * (1 / fact (k - 1)) / zeta (of_nat k) = c2"
    by (simp add: c2_def)
  also have "2 * fps_const c1 * (1 - fps_const (-c2) * L) =
    2 * (fps_const c1 + fps_const (c1 * c2) * L)"
    by (simp add: algebra_simps flip: fps_const_neg)
  also have "... = fps_Eisenstein_G k"
    using <k  $\geq$  2> True zeta_Re_gt_1_nonzero[of "of_nat k"]
    by (simp add: c1_def c2_def fps_Eisenstein_G_def L_def flip: divisor_sigma_of_nat)
  finally show ?thesis ..
qed (auto simp: fps_Eisenstein_G_def fps_Eisenstein_E_def)

lemma fps_Eisenstein_E_eq_0_iff [simp]: "fps_Eisenstein_E k = 0  $\iff$  odd k"
proof (cases "even k")
  case True
  hence "fps_nth (fps_Eisenstein_E k) 0  $\neq$  0" "fps_nth (0 :: complex fps) 0 = 0"
  by (auto simp: fps_Eisenstein_E_def)
  thus ?thesis using True
  by metis

```

```

qed (auto simp: fps_Eisenstein_E_def elim!: oddE)

lemma subdegree_fps_Eisenstein_E [simp]: "subdegree (fps_Eisenstein_E
k) = 0"
proof (cases "even k")
  case True
  thus ?thesis
    by (intro subdegree_eq_0) (auto simp: fps_Eisenstein_E_def)
qed (auto simp: fps_Eisenstein_E_def)

lemma has_fps_expansion_q_Eisenstein_E [fps_expansion_intros]:
  "q_Eisenstein_E k has_fps_expansion fps_Eisenstein_E k"
proof (cases "k ≥ 2 ∧ even k")
  case True
  have "(λq. 1 - 2 * of_nat k / of_real (bernoulli k) * lambert (λn. of_nat
n ^ (k - 1) :: complex) q)
    has_fps_expansion (1 - fps_const (2 * of_nat k / of_real (bernoulli
k)) * Abs_fps (λn. of_nat (divisor_sigma (k-1) n)))"
    by (intro fps_expansion_intros)
  thus ?thesis
    using True by (auto simp: q_Eisenstein_E_def [abs_def] fps_Eisenstein_E_def
of_real_bernoulli)
qed (auto simp: q_Eisenstein_E_def[abs_def] fps_Eisenstein_E_def)

lemma holomorphic_on_q_Eisenstein_E [holomorphic_intros]:
  "f holomorphic_on A ⇒ (∧z. z ∈ A ⇒ norm (f z) < 1) ⇒
  (λz. q_Eisenstein_E k (f z)) holomorphic_on A"
  unfolding q_Eisenstein_E_def
  by (cases "k = 0"; cases "even k")
    (force intro!: holomorphic_intros simp: lambert_conv_radius_power_of_nat)+

lemma analytic_on_q_Eisenstein_E [analytic_intros]:
  "f analytic_on A ⇒ (∧z. z ∈ A ⇒ norm (f z) < 1) ⇒
  (λz. q_Eisenstein_E k (f z)) analytic_on A"
  unfolding q_Eisenstein_E_def
  by (cases "k = 0"; cases "even k")
    (force intro!: analytic_intros simp: lambert_conv_radius_power_of_nat
bernoulli_zero_iff)+

lemma tendsto_q_Eisenstein_E [tendsto_intros]:
  assumes "(f ⟶ q) F" "norm q < 1"
  shows "(λz. q_Eisenstein_E k (f z)) ⟶ q_Eisenstein_E k q) F"
proof -
  have "q_Eisenstein_E k analytic_on {q}"
    by (rule analytic_intros) (use assms in auto)
  with assms(1) show ?thesis
    by (metis analytic_at_imp_isCont isCont_tendsto_compose)
qed

```

```

lemma continuous_q_Eisenstein_E [continuous_intros]:
  "continuous (at x within A) f  $\implies$  norm (f x) < 1  $\implies$ 
    continuous (at x within A) ( $\lambda z$ . q_Eisenstein_E k (f z))"
unfolding continuous_def
using tendsto_q_Eisenstein_E[of f "f x" "at x within A"]
by (cases "at x within A = bot") (auto simp: Lim_ident_at)

```

12.4 Identities for normalised Eisenstein series

Just like G_{2n} , E_{2n} can be given as a polynomial in E_4 and E_6 with rational coefficients. The identities resulting from this tend to look a bit nicer than those for G .

```

context
  fixes B :: "nat  $\Rightarrow$  rat"
  defines "B  $\equiv$  bernoulli"
begin

```

```

definition eisenstein_series_poly'_aux :: "nat  $\Rightarrow$  nat  $\Rightarrow$  rat" where
  "eisenstein_series_poly'_aux n i =
    -(B (2 * (i + 2))) * B (2 * (n - i)) / B (2 * (n + 2))) *
    (of_nat (2 * (n + 2) choose 2 * (i + 2)))"

```

```

lemma of_rat_eisenstein_series_poly'_aux:
  assumes "i + 2  $\leq$  n"
  shows "of_rat (eisenstein_series_poly'_aux n i) =
    2 * zeta (2 * of_nat (i+2)) * zeta (2 * of_nat (n-i)) / zeta
    (2 * of_nat (n+2))"

```

proof -

```

  have "2 * zeta (2 * of_nat (i+2)) * zeta (2 * of_nat (n-i)) / zeta (2
  * of_nat (n+2)) =
    complex_of_real (((-1) ^ Suc (i + 2) * (- 1) ^ Suc (n - i)
  / (- 1) ^ Suc (n + 2)) *
    of_rat (B (2 * (i + 2))) * B (2 * (n - i)) / B (2 * (n +
  2))) *
    ((2 * pi) ^ (2 * (i + 2)) * (2 * pi) ^ (2 * (n - i)) /
  (2 * pi) ^ (2 * (n + 2))) *
    ((fact (2 * (n + 2))) / (fact (2 * (i + 2)) * fact (2 *
  (n - i))))))"
  by (subst (1 2 3) zeta_even_nat)
    (simp_all add: mult_ac B_def of_rat_mult of_rat_divide bernoulli_conv_bernoulli'
  of_rat_bernoulli')
  also have "(-1) ^ Suc (i + 2) * (- 1) ^ Suc (n - i) / (- 1) ^ Suc (n
  + 2) = (-1::real)"
  using assms by (auto simp: power_neg_one_If)
  also have "(2 * pi) ^ (2 * (i + 2)) * (2 * pi) ^ (2 * (n - i)) / (2
  * pi) ^ (2 * (n + 2)) =
    (2 * pi) ^ (2 * (i + 2) + 2 * (n - i) - 2 * (n + 2))"
  by (subst power_diff) (simp_all add: power_add)
  also have "2 * (i + 2) + 2 * (n - i) - 2 * (n + 2) = 0"

```

```

    using assms by (auto simp: algebra_simps)
    also have "fact (2 * (n + 2)) / (fact (2 * (i + 2)) * fact (2 * (n -
i))) =
        ((2*(n+2)) choose (2*(i+2)))"
    using assms by (subst binomial_fact) (simp_all add: algebra_simps)
    finally show ?thesis
    by (simp add: power_mult_distrib mult_ac of_rat_mult of_rat_divide
        eisenstein_series_poly'_aux_def of_rat_minus
        del: binomial_Suc_Suc)
qed

fun eisenstein_series_poly' :: "nat ⇒ rat poly poly" where
    "eisenstein_series_poly' n =
        (if n = 0 then [: [:0, 1:] :]
         else if n = 1 then [:0, 1:]
         else
            Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
                (∑ i≤n-2. Polynomial.smult [: of_nat ((2*i+3)*(2*(n-i)-1))
* eisenstein_series_poly'_aux n i:]
                    (eisenstein_series_poly' i * eisenstein_series_poly' (n-2-i))))"

lemmas [simp del] = eisenstein_series_poly'.simps

lemma eisenstein_series_poly'_0 [simp]: "eisenstein_series_poly' 0 =
[: [:0, 1:] :]"
    and eisenstein_series_poly'_1 [simp]: "eisenstein_series_poly' (Suc
0) = [:0, 1:]"
    and eisenstein_series_poly'_rec:
        "n ≥ 2 ⇒ eisenstein_series_poly' n =
            Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
                (∑ i≤n-2. Polynomial.smult [: of_nat ((2*i+3)*(2*(n-i)-1))
* eisenstein_series_poly'_aux n i:]
                    (eisenstein_series_poly' i * eisenstein_series_poly' (n-2-i)))"
    by (subst eisenstein_series_poly'.simps; simp; fail)+

lemma is_46_poly_eisenstein_series_poly': "is_46_poly (2*n+4) (eisenstein_series_poly'
n)"
proof (induction n rule: less_induct)
    case (less n)
    consider "n = 0" | "n = 1" | "n ≥ 2"
    by linarith
    thus ?case
    proof cases
        assume n: "n ≥ 2"
        show ?thesis using n
        by (auto intro!: is_46_poly_x is_46_poly_y is_46_poly_smult is_46_poly_sum
is_46_poly_mult less.IH
            simp: of_nat_poly eisenstein_series_poly'_rec simp flip:
pCons_one)
    qed

```

```

qed (use is_46_poly_x is_46_poly_y in auto)
qed

lemma eisenstein_series_poly'_correct:
  assumes z: "Im z > 0"
  defines "E ≡ (λn. Eisenstein_E n z)"
  shows "poly2 (map_poly2 of_rat (eisenstein_series_poly' n)) (E 4) (E
6) = E (2 * n + 4)"
proof (induction n rule: eisenstein_series_poly.induct)
  case (1 n)
  interpret map1: map_poly_comm_ring_hom "of_rat :: rat ⇒ complex"
    by standard auto
  interpret map2: map_poly_comm_ring_hom "map_poly (of_rat :: rat ⇒ complex)"
    by standard auto
  interpret std_complex_lattice z
    by standard (use z in auto)
  have E_conv_G: "E n = G n / (2 * zeta (of_nat n))" for n
    by (auto simp: E_def Eisenstein_E_def eisenstein_series_eq_Eisenstein_G)

  consider "n = 0" | "n = 1" | "n ≥ 2"
    by linarith
  thus ?case
  proof cases
    case 3
    define c1 where "c1 = inverse ((2 * of_nat n + 5) * (of_nat n - 1)
* (2 * of_nat n + 3) :: complex)"
    define c2 where "c2 = (λi. ((2 * of_nat i + 3) * (2 * of_nat n -
2 * of_nat i - 1)) :: complex)"
    have "poly2 (map_poly2 of_rat (eisenstein_series_poly' n)) (E 4) (E
6) =
      (∑ i ≤ n-2. 3 * (of_rat (eisenstein_series_poly'_aux n i) *
c2 i *
      (E (2 * i + 4) * E (2 * (n - i - 2) + 4))) *
c1)"
      using 3 1 by (subst eisenstein_series_poly'.simps)
      (simp add: hom_distrib map_poly2_def c1_def c2_def
inverse_eq_divide)
    also have "... = 1 / (2 * zeta (2*n+4)) *
      (3 * c1 * (∑ i ≤ n-2. c2 i * (G (4 + i * 2) * G (4
+ 2 * (n - 2 - i)))))"
      unfolding sum_distrib_left
  proof (rule sum.cong, goal_cases)
    case (2 i)
    have "zeta (2 * complex_of_nat n - 2 * complex_of_nat i) ≠ 0"
      using zeta_of_nat_eq_0_iff[of "2 * (n - i)"] 2 3 by simp
    moreover have "zeta (2 * complex_of_nat i + 4) ≠ 0"
      using zeta_of_nat_eq_0_iff[of "2 * i + 4"] by simp
    ultimately show ?case
      using 2 3 by (subst of_rat_eisenstein_series_poly'_aux) (auto

```

```

simp: E_conv_G field_simps)
  qed auto
  also have "3 * c1 * ( $\sum_{i \leq n-2} c2 i * (G (4 + i * 2) * G (4 + 2 * (n - 2 - i)))$ ) = G (2 * n + 4)"
    using 3 by (subst eisenstein_series_recurrence) (auto simp: c1_def c2_def field_simps)
  also have "1 / (2 * zeta (2*n+4)) * G (2 * n + 4) = E (2 * n + 4)"
    by (simp add: E_conv_G)
  finally show ?thesis .
qed (auto simp: map_poly2_def)
qed

end

lemma eisenstein_series_poly_q_Eisenstein_E:
  assumes "norm q < 1"
  shows "poly2 (map_poly2 of_rat (eisenstein_series_poly' n))
    (q_Eisenstein_E 4 q) (q_Eisenstein_E 6 q) = q_Eisenstein_E
    (2 * n + 4) q"
    (is "?lhs q = ?rhs q")
proof -
  define f where "f = ( $\lambda q. ?lhs q - ?rhs q$ )"
  have f_eq_0: "f q = 0" if q: "norm q < 1" "q  $\neq$  0" for q
  proof -
    show ?thesis
      using eisenstein_series_poly'_correct[of "of_q 1 q" n] q
      by (simp add: f_def Eisenstein_E_conv_q Im_of_q divide_less_0_iff)
  qed

  show ?thesis
  proof (cases "q = 0")
    case True
    have "isCont f 0"
      unfolding f_def by (intro continuous_intros continuous_within_poly2)
  auto
  hence "f  $\rightarrow$  0"
    by (rule isContD)
  moreover have "eventually ( $\lambda q. q \in \text{ball } 0 \ 1 - \{0\}$ ) (at (0::complex))"
    by (rule eventually_at_in_open) auto
  hence "eventually ( $\lambda q. f q = 0$ ) (at 0)"
    by eventually_elim (use f_eq_0 in auto)
  hence "f  $\rightarrow$  0"
    using tendsto_eventually by blast
  ultimately have "f 0 = 0"
    by (rule tendsto_unique[rotated]) auto
  thus ?thesis
    using True by (simp add: f_def)
  qed (use f_eq_0[of q] assms in <auto simp: f_def>)
qed

```

```

lemma eisenstein_series_poly'_fps_Eisenstein_E:
  "poly2 (map_poly2 (fps_const ∘ of_rat) (eisenstein_series_poly' n))
    (fps_Eisenstein_E 4) (fps_Eisenstein_E 6) = fps_Eisenstein_E (2 *
n + 4)" (is "?lhs = ?rhs")
proof -
  define F where "F = poly2 (map_poly2 fps_const (map_poly2 of_rat (eisenstein_series_poly'
n)))
    (fps_Eisenstein_E 4) (fps_Eisenstein_E 6) - fps_Eisenstein_E (2 *
n + 4)"
  define f where "f = (λq. poly2 (map_poly2 of_rat (eisenstein_series_poly'
n))
    (q_Eisenstein_E 4 q) (q_Eisenstein_E 6 q) - q_Eisenstein_E
(2 * n + 4) q)"
  have "f has_fps_expansion F"
    unfolding f_def F_def by (intro fps_expansion_intros)
  moreover have "eventually (λq. q ∈ ball 0 1) (nhds (0::complex))"
    by (rule eventually_nhds_in_open) auto
  hence "eventually (λq. f q = 0) (nhds 0)"
  proof eventually_elim
    case (elim q)
    thus ?case using eisenstein_series_poly_q_Eisenstein_E[of q n] by
(simp add: f_def)
  qed
  ultimately have "F = 0"
    by (metis fps_expansion_unique_complex has_fps_expansion_0_iff)
  also have "F = ?lhs - ?rhs"
    unfolding F_def by (subst map_poly2_compose) auto
  finally show ?thesis by simp
qed

```

More efficient computation:

```

definition eisenstein_polys'_step :: "rat poly poly list ⇒ rat poly poly
list" where
  "eisenstein_polys'_step ps =
    (let n = length ps
      in ps @ [Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1) * (2*n+3)):]
        (∑ i≤n-2. Polynomial.smult [: of_nat ((2*i+3)*(2*(n-i)-1))
* eisenstein_series_poly'_aux n i :]
        (ps ! i * ps ! (n-2-i)))]])"

```

```

definition eisenstein_series_polys' :: "nat ⇒ rat poly poly list" where
  "eisenstein_series_polys' n = (eisenstein_polys'_step ^^ (n - 2)) [[:
[:0, 1:] :], [:0, 1:]]"

```

```

lemma eisenstein_polys'_step_correct:
  assumes n: "n ≥ 2" and ps_eq: "ps = map eisenstein_series_poly' [0..<n]"
  shows "eisenstein_polys'_step ps = map eisenstein_series_poly' [0..<Suc
n]"

```

```

proof (rule nth_equalityI)
  fix i assume "i < length (eisenstein_polys'_step ps)"
  have length: "length ps = n"
    by (simp add: ps_eq)
  define p where "p = Polynomial.smult [:3 / of_nat ((2*n+5) * (n-1)
* (2*n+3)):]
    (∑ i≤n-2. Polynomial.smult [:of_nat ((2*i+3)*(2*(n-i)-1))
* eisenstein_series_poly'_aux n i:]
    (ps ! i * ps ! (n-2-i)))"
  have step: "eisenstein_polys'_step ps = ps @ [p]"
    by (simp add: eisenstein_polys'_step_def p_def length Let_def)
  have i: "i ≤ n"
    using <i < _> unfolding ps_eq length eisenstein_polys'_step_def by
simp
  show "eisenstein_polys'_step ps ! i = map eisenstein_series_poly' [0..<Suc
n] ! i"
  proof (cases "i = n")
    case False
    thus ?thesis
      using i unfolding step by (auto simp: ps_eq nth_append simp del:
upt_Suc)
  next
    case [simp]: True
    have "eisenstein_polys'_step ps ! i = p"
      by (auto simp: step nth_append length)
    also have "p = eisenstein_series_poly' n"
      unfolding eisenstein_series_poly'_rec[OF n] p_def ps_eq
      by (intro arg_cong2[of _ _ _ Polynomial.smult] refl sum.cong)
    (use n in auto)
    also have "... = map eisenstein_series_poly' [0..<Suc n] ! i"
      by (simp del: upt_Suc)
    finally show ?thesis .
  qed
qed (auto simp: eisenstein_polys'_step_def ps_eq)

lemma eisenstein_series_polys'_correct:
  "eisenstein_series_polys' n = map eisenstein_series_poly' [0..<max 2
n]"
proof (induction n rule: less_induct)
  case (less n)
  show ?case
  proof (cases "n ≥ 3")
    case False
    thus ?thesis
      by (auto simp: eisenstein_series_polys'_def upt_rec)
  next
    case True
    define m where "m = n - 3"
    have n_eq: "n = Suc (Suc (Suc m))"

```

```

    using True unfolding m_def by simp
    have "eisenstein_polys'_step (eisenstein_series_polys' (n-1)) =
      map eisenstein_series_poly' [0..<Suc (n-1)]"
    proof (rule eisenstein_polys'_step_correct)
      have "eisenstein_series_polys' (n - 1) = map eisenstein_series_poly'
[0..<max 2 (n-1)]"
        by (rule less.IH) (use True in auto)
      thus "eisenstein_series_polys' (n - 1) = map eisenstein_series_poly'
[0..<n-1]"
        using True by simp
    qed (use True in auto)
    also have "eisenstein_polys'_step (eisenstein_series_polys' (n-1))
= eisenstein_series_polys' n"
      by (simp add: eisenstein_series_polys'_def eval_nat_numeral n_eq)
    also have "Suc (n - 1) = max 2 n"
      using True by auto
    finally show ?thesis .
  qed
qed

```

```

lemma eisenstein_series_poly_code [code]:
  "eisenstein_series_poly' n = eisenstein_series_polys' (Suc n) ! n"
  unfolding eisenstein_series_polys'_correct by (subst nth_map) (auto
simp del: upt_Suc)

```

Again, a number of identities for the normalised Eisenstein series follows directly by simply expressing both sides in terms of E_4 and E_6 and simplifying. Corresponding identities for the divisor function can be read off directly from the coefficients of these power series.

```

context
  fixes E
  defines "E ≡ fps_Eisenstein_E"
begin

```

```

lemma eisenstein_series_polys'_correct':
  assumes "eisenstein_series_polys' m = ps"
  shows "list_all (λi. E (2*i+4) = poly2 (map_poly2 (fps_const ∘ of_rat)
(ps ! i)) (E 4) (E 6)) [0..<m]"
  unfolding assms [symmetric] eisenstein_series_polys'_correct
  using eisenstein_series_poly'_fps_Eisenstein_E by (auto simp: list_all_iff
E_def)

```

```

lemma Eisenstein_E_identities:
  "E 8 = E 4 ^ 2" (is ?th8)
  "E 10 = E 4 * E 6" (is ?th10)
  "E 12 = 441 / 691 * E 4 ^ 3 + 250 / 691 * E 6 ^ 2" (is ?th12)
  "E 14 = E 4 ^ 2 * E 6" (is ?th14)
  "E 16 = 1617 / 3617 * E 4 ^ 4 + 2000 / 3617 * E 4 * E 6 ^ 2" (is ?th16)
  "E 18 = 38367 / 43867 * E 4 ^ 3 * E 6 + 5500 / 43867 * E 6 ^ 3" (is ?th18)

```

```

"E 20 = 53361 / 174611 * E 4 ^ 5 + 121250 / 174611 * E 4 ^ 2 * E 6 ^
2" (is ?th20)
proof -
  have [simp]: "{..1::nat} = {0,1}"
    by auto
  have eq: "eisenstein_series_polys' 9 = [[[:0, 1:] :], [:0, [:1:] :],
[:[:0, 0, 1:] :],
      [:0, [:0, 1:] :], [[:[:0, 0, 0, 441 / 691:] :], 0, [:250 / 691:] :],
      [:0, [:0, 0, 1:] :], [[:[:0, 0, 0, 0, 1617 / 3617:] :], 0,
      [:0, 2000 / 3617:] :], [[:0, [:0, 0, 0, 38367 / 43867:] :], 0,
[:5500 / 43867:] :],
      [[:[:0, 0, 0, 0, 0, 53361 / 174611:] :], 0, [:0, 0, 121250 /
174611:] :]]]"
    by (simp add: eisenstein_series_polys'_def eisenstein_polys'_step_def
atMost_nat_numeral
        eisenstein_series_poly'_aux_def binomial_fact fact_numeral
        funpow_rec_right numeral_poly smult_add_right smult_diff_right

        flip: pCons_one One_nat_def)
  show ?th8 ?th10 ?th12 ?th14 ?th16 ?th18 ?th20
    using eisenstein_series_polys'_correct'[OF eq]
    by (simp_all add: upt_rec map_poly2_def of_rat_divide power_numeral_reduce
field_simps
        numeral_fps_const flip: E_def pCons_one)
qed
end

```

As an example, we derive the identity that expresses σ_9 in terms of σ_3 and σ_5 :

```

lemma divisor_sigam_9_conv_3_5:
  fixes n :: nat
  assumes n: "n > 0"
  fixes sigma :: "nat  $\Rightarrow$  nat  $\Rightarrow$  int" (" $\sigma$ ")
  defines "sigma  $\equiv$  ( $\lambda$ k n. int (divisor_sigma k n))"
  shows "11 * ( $\sigma_9$  n) = -10 * ( $\sigma_3$  n) + 21 * ( $\sigma_5$  n) +
5040 * ( $\sum_{i \in \{0 <..<n\}} \sigma_3 i * \sigma_5 (n - i)$ )"
proof -
  define F where "F = Abs_fps ( $\lambda$ n. ( $\sigma_3$  n) :: int)"
  define G where "G = Abs_fps ( $\lambda$ n. ( $\sigma_5$  n) :: int)"
  have "fps_nth (fps_Eisenstein_E 10) n = fps_nth (fps_Eisenstein_E 4
* fps_Eisenstein_E 6) n"
    by (subst Eisenstein_E_identities) (rule refl)
  hence "complex_of_int (-264 * ( $\sigma_9$  n)) =
of_int ( $\sigma_3$  n * 240 -  $\sigma_5$  n * 504) +
fps_nth (fps_const (-120960) * (of_int.fps F * of_int.fps
G)) n"
    using n by (simp add: fps_Eisenstein_E_def ring_distrib mult_ac
F_def G_def sigma_def)

```

```

also have "fps_nth (fps_const (-120960) * (of_int.fps F * of_int.fps
G)) n =
      (-120960 :: complex) * fps_nth (of_int.fps (F * G)) n"
  unfolding fps_mult_left_const_nth by (simp add: hom_distrib)
also have "fps_nth (of_int.fps (F * G)) n = of_int (fps_nth (F * G)
n)"
  by (rule of_int.fps_nth)
also have "fps_nth (F * G) n = (∑ i = 0..n. σ₃ i * σ₅ (n - i))"
  by (simp add: fps_mult_nth F_def G_def)
also have "(∑ i = 0..n. σ₃ i * σ₅ (n - i)) = (∑ i ∈ {0<..

```

12.5 The modular discriminant

As before, the modular discriminant is defined as $(60G_4^3) - 27(140G_6)^2$. It is non-zero everywhere and (as we will see later) vanishes at $i\infty$.

definition `modular_discr` :: "complex \Rightarrow complex" where

```
"modular_discr z = (60 * Eisenstein_G 4 z) ^ 3 - 27 * (140 * Eisenstein_G
6 z) ^ 2"
```

lemma `modular_discr_altdef`:

```
"modular_discr z = (4/3)^3 * of_real pi ^ 12 * (Eisenstein_E 4 z ^ 3
- Eisenstein_E 6 z ^ 2)"
```

```
by (auto simp: modular_discr_def Eisenstein_E_def zeta_even_numeral
field_simps fact_numeral)
```

lemma (in `complex_lattice`) `discr_eq_modular_discr`: "discr = modular_discr
(ω_2 / ω_1) / ω_1^{12} "

```
unfolding discr_def modular_discr_def invariant_g2_def invariant_g3_def
eisenstein_series_eq_Eisenstein_G
```

```
by (simp add: field_simps)
```

lemma `modular_discr_real_eq_0` [simp]: "z \in $\mathbb{R} \implies$ modular_discr z = 0"

```
by (simp add: modular_discr_def)
```

lemma `modular_discr_cnj`: "modular_discr (cnj z) = cnj (modular_discr

```

z)"
  by (simp add: modular_discr_def Eisenstein_G_cnj)

lemma modular_discr_analytic [analytic_intros]:
  assumes "f analytic_on A" " $\wedge z. z \in A \implies f z \notin \mathbb{R}$ "
  shows " $(\lambda z. \text{modular\_discr } (f z)) \text{ analytic\_on } A$ "
  unfolding modular_discr_def using assms by (auto intro!: analytic_intros)

lemma modular_discr_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" " $\wedge z. z \in A \implies f z \notin \mathbb{R}$ "
  shows " $(\lambda z. \text{modular\_discr } (f z)) \text{ holomorphic\_on } A$ "
  unfolding modular_discr_def using assms by (auto intro!: holomorphic_intros)

lemma continuous_modular_discr [continuous_intros]:
  "continuous (at x within A) f  $\implies f x \notin \mathbb{R} \implies$ 
  continuous (at x within A)  $(\lambda z. \text{modular\_discr } (f z))$ "
  unfolding modular_discr_def by (intro continuous_intros) auto

lemma continuous_on_modular_discr [continuous_intros]:
  "continuous_on A f  $\implies (\forall x \in A. f x \notin \mathbb{R}) \implies$ 
  continuous_on A  $(\lambda z. \text{modular\_discr } (f z))$ "
  unfolding modular_discr_def by (intro continuous_intros) auto

lemma modular_discr_uminus: "modular_discr (-z) = modular_discr z"
  by (simp add: modular_discr_def Eisenstein_G_uminus)

lemma modular_discr_nonzero:
  assumes "z  $\notin \mathbb{R}$ "
  shows "modular_discr z  $\neq 0$ "
proof -
  interpret complex_lattice 1 z
  by standard (use assms in <auto simp: fundpair_def>)
  have "modular_discr z = discr"
  by (simp add: discr_eq_modular_discr)
  also have "...  $\neq 0$ "
  by (rule discr_nonzero)
  finally show ?thesis .
qed

lemma modular_discr_eq_0_iff: "modular_discr z = 0  $\iff z \in \mathbb{R}$ "
  using modular_discr_nonzero[of z] by auto

theorem modular_discr_apply_modgrp:
  "modular_discr (apply_modgrp f z) = automorphy_factor f z  $^{\wedge} 12 * \text{modular\_discr } z$ "
  by (simp add: modular_discr_def Eisenstein_G_apply_modgrp algebra_simps)

lemma modular_discr_plus_int: "modular_discr (z + of_int m) = modular_discr z"

```

```

using modular_discr_apply_modgrp[of "shift_modgrp m" z] by simp

lemma modular_discr_minus_one_over: "modular_discr  $-(1/z)$  =  $z^{-12}$ 
* modular_discr z"
using modular_discr_apply_modgrp[of "S_modgrp" z] by simp

```

12.6 Ramanujan's tau function

The Ramanujan τ function are the integer coefficients of the normalised modular discriminant $\Delta/(2\pi)^{12}$.

```

definition ramanujan_tau :: "nat  $\Rightarrow$  int" where
  "ramanujan_tau n =
    (let F = ( $\lambda$ k. Abs_fps (divisor_sigma k))
     in fps_nth ((1 + 240 * F 3) ^ 3 - (1 - 504 * F 5) ^ 2) n div 12
    ^ 3)"

```

```

definition fps_modular_discr :: "complex fps" where
  "fps_modular_discr = fps_const ((2 * pi) ^ 12) * of_int.fps (Abs_fps
ramanujan_tau)"

```

```

lemma fps_ramanujan_tau_eq:
  "of_int.fps (Abs_fps ramanujan_tau) =
    fps_const (1 / 12 ^ 3) * (fps_Eisenstein_E 4 ^ 3 - fps_Eisenstein_E
6 ^ 2)"

```

proof -

```

  define c where "c = complex_of_real ((2 * pi) ^ 12)"
  define A :: "int fps" where "A = Abs_fps (divisor_sigma 3)"
  define B :: "int fps" where "B = Abs_fps (divisor_sigma 5)"
  define C :: "int fps" where "C = Abs_fps ( $\lambda$ n. (5 * fps_nth A n + 7
* fps_nth B n) div 12)"
  define D where "D = 100 * A ^ 2 - 147 * B ^ 2 + 8000 * A ^ 3"

  define F0 :: "complex fps" where
    "F0 = (fps_const (complex_of_real (4 / 3 * pi ^ 4)) * of_int.fps (1
+ 240 * A)) ^ 3 -
      27 * (fps_const (complex_of_real (8 / 27 * pi ^ 6)) * of_int.fps
(1 - 504 * B)) ^ 2"
  define F :: "complex fps" where "F = fps_modular_discr"

```

```

have "12 dvd 5 * fps_nth A n + 7 * fps_nth B n" for n :: nat

```

proof -

```

  have "12 dvd 5 * d ^ 3 + 7 * d ^ 5" for d :: nat
  proof -
    have "d mod 12  $\in$  {.. $12$ }"
    by simp
    hence "[0 = 5 * (d mod 12) ^ 3 + 7 * (d mod 12) ^ 5] (mod 12)"
      unfolding lessThan_nat_numeral arith_simps pred_numeral_simps
lessThan_0
      by (elim insertE emptyE) (auto simp: Cong.cong_def)

```

```

    also have "[5 * (d mod 12) ^ 3 + 7 * (d mod 12) ^ 5 = 5 * d ^ 3
+ 7 * d ^ 5] (mod 12)"
      by (intro cong_add cong_mult cong_pow) auto
    finally show ?thesis
      by (metis cong_0_iff cong_iff_lin_nat)
  qed
  hence "int 12 dvd int (∑ d | d dvd n. 5 * d ^ 3 + 7 * d ^ 5)"
    unfolding int_dvd_int_iff by (intro dvd_sum) auto
  also have "... = 5 * fps_nth A n + 7 * fps_nth B n"
    by (simp add: A_def B_def divisor_sigma_def sum.distrib algebra_simps

          sum_distrib_left sum_distrib_right)
  finally show ?thesis by simp
  qed
  hence "5 * A + 7 * B = 12 * C"
    by (auto simp: A_def B_def C_def fps_eq_iff numeral_fps_const)

  have "(1 + 240 * A) ^ 3 - (1 - 504 * B) ^ 2 =
        12 ^ 2 * (5 * A + 7 * B) + 12 ^ 3 * D"
    by (simp add: algebra_simps power2_eq_square power3_eq_cube D_def)
  also have "5 * A + 7 * B = 12 * C"
    by fact
  also have "12 ^ 2 * (12 * C) = 12 ^ 3 * C"
    by (simp add: eval_nat_numeral)
  finally have eq: "(1 + 240 * A) ^ 3 - (1 - 504 * B) ^ 2 = 12 ^ 3 * (C
+ D)"
    by (simp add: algebra_simps)

  have dvd: "12 ^ 3 dvd fps_nth ((1 + 240 * A) ^ 3 - (1 - 504 * B) ^ 2)
n" for n
    by (subst eq) auto

  show "of_int.fps (Abs_fps ramanujan_tau) =
        fps_const (1 / 12 ^ 3) * (fps_Eisenstein_E 4 ^ 3 - fps_Eisenstein_E
6 ^ 2)"
  proof (rule fps_ext)
    fix n :: nat
    have "12 ^ 3 * ramanujan_tau n =
        12 ^ 3 * (fps_nth ((1 + 240 * A) ^ 3 - (1 - 504 * B) ^ 2)
n div 12 ^ 3)"
      by (simp add: ramanujan_tau_def A_def B_def Let_def)
    also have "... = fps_nth ((1 + 240 * A) ^ 3 - (1 - 504 * B) ^ 2) n"
      using dvd[of n] by simp
    also have "of_int ... = fps_nth (of_int.fps ((1 + 240 * A) ^ 3 - (1
- 504 * B) ^ 2)) n"
      by simp
    also have "of_int.fps ((1 + 240 * A) ^ 3 - (1 - 504 * B) ^ 2) =
        of_int.fps (1 + 240 * A) ^ 3 - (of_int.fps (1 - 504 *
B) ^ 2) :: complex fps)"

```

```

    by (simp add: hom_distrib)
  also have "of_int.fps (1 + 240 * A) = fps_Eisenstein_E 4"
    unfolding fps_Eisenstein_E_def by (simp add: hom_distrib A_def)
  also have "of_int.fps (1 - 504 * B) = fps_Eisenstein_E 6"
    unfolding fps_Eisenstein_E_def by (simp add: hom_distrib B_def)
  finally show "fps_nth (of_int.fps (Abs_fps ramanujan_tau)) n =
    fps_nth (fps_const (1/12^3) * (fps_Eisenstein_E 4
    ^ 3 - (fps_Eisenstein_E 6)^2)) n"
    by (simp add: mult_ac)
  qed
qed

```

lemma fps_modular_discr_conv_Eisenstein:

```

  "fps_modular_discr =
    fps_const ((4/3)^3 * of_real pi ^ 12) *
    (fps_Eisenstein_E 4 ^ 3 - fps_Eisenstein_E 6 ^ 2)"
  unfolding fps_modular_discr_def fps_ramanujan_tau_eq
  by (simp add: power_divide flip: mult.assoc)

```

lemma atLeastAtMost_nat_numeral_right:

```

  "a ≤ numeral b ⇒ {(a::nat)..numeral b} = insert (numeral b) {a..pred_numeral
  b}"
  by (auto simp: numeral_eq_Suc)

```

lemma ramanujan_tau_0 [simp]: "ramanujan_tau 0 = 0"

and ramanujan_tau_1 [simp]: "ramanujan_tau (Suc 0) = 1"

and ramanujan_tau_2: "ramanujan_tau 2 = -24"

and ramanujan_tau_3: "ramanujan_tau 3 = 252"

and ramanujan_tau_4: "ramanujan_tau 4 = -1472"

and ramanujan_tau_5: "ramanujan_tau 5 = 4830"

and ramanujan_tau_6: "ramanujan_tau 6 = -6048"

```

  by (simp_all add: ramanujan_tau_def power2_eq_square power3_eq_cube
  fps_mult_nth fps_numeral_nth

```

```

    divisor_sigma_naive fold_atLeastAtMost_nat.simps atLeastAtMost_nat_numeral
    flip: numeral_2_eq_2)

```

lemma fps_modular_discr_nonzero [simp]: "fps_modular_discr ≠ 0"

and subdegree_fps_modular_discr [simp]: "subdegree fps_modular_discr = 1"

proof -

```

  note_simps = fps_modular_discr_def fps_Eisenstein_E_def power2_eq_square
  power3_eq_cube

```

have 0: "fps_nth fps_modular_discr 0 = 0"

by (auto simp: fps_eq_iff_simps)

have 1: "fps_nth fps_modular_discr (Suc 0) = 4096 * of_real pi ^ 12"

by (auto simp: fps_eq_iff_simps)

from 1 show [simp]: "fps_modular_discr ≠ 0"

by auto

```

have "subdegree fps_modular_discr > 0"
  by (rule subdegree_greaterI) (auto simp: 0)
moreover have "subdegree fps_modular_discr ≤ 1"
  by (rule subdegree_leI) (auto simp: 1)
ultimately show "subdegree fps_modular_discr = 1"
  by linarith
qed

```

12.7 The modular λ function

```

definition modular_lambda :: "complex  $\Rightarrow$  complex" where
  "modular_lambda z = (if z  $\in$   $\mathbb{R}$  then 0 else complex_lattice.modulus 1
z)"

```

```

lemma (in complex_lattice) modulus_eq_modular_lambda:
  "modulus = modular_lambda ( $\omega_2 / \omega_1$ )"
proof -
  interpret complex_lattice_stretch  $\omega_1 \omega_2$  "1 /  $\omega_1$ "
  by standard auto
  have "stretched.modulus = modulus"
  by (rule modulus_stretch)
  also have "stretched.modulus = modular_lambda ( $\omega_2 / \omega_1$ )"
  using stretched.fundpair unfolding modular_lambda_def fundpair_def
by simp
  finally show ?thesis
  by simp
qed

```

```

lemma modular_lambda_real_eq_0 [simp]: "z  $\in$   $\mathbb{R}$   $\implies$  modular_lambda z =
0"
  by (simp add: modular_lambda_def)

```

```

lemma modular_lambda_cnj: "modular_lambda (cnj z) = cnj (modular_lambda
z)"
proof (cases "z  $\in$   $\mathbb{R}$ ")
  case False
  interpret complex_lattice 1 z
  using False by unfold_locales (auto simp: fundpair_def)
  interpret complex_lattice_cnj 1 z ..
  show ?thesis
  using modulus_cnj modulus_eq_modular_lambda cnj.modulus_eq_modular_lambda
by simp
qed auto

```

```

lemma modular_lambda_uminus: "modular_lambda (-z) = modular_lambda z"
proof (cases "z  $\in$   $\mathbb{R}$ ")
  case False
  interpret lattice1: complex_lattice 1 z

```

```

    by standard (use False in <auto simp: fundpair_def>)
interpret lattice2: complex_lattice 1 "-z"
    by standard (use False in <auto simp: fundpair_def>)
have *: "fundpair (1, z)" "fundpair (1, -z)"
    using False by (auto simp: fundpair_def)
have "|(-1) * 1 - 0 * 0 :: int| = 1 ∧
      - z = of_int (-1) * z + of_int 0 * 1 ∧ 1 = of_int 0 * z + of_int
1 * 1"
    by auto
hence "equiv_fundpair (1, z) (1, -z)"
    using equiv_fundpair_iff[OF *] by blast
hence [simp]: "lattice2.lattice = lattice1.lattice"
    by (auto simp: equiv_fundpair_def)
hence [simp]: "lattice2.lattice0 = lattice1.lattice0"
    by (auto simp: lattice1.lattice0_def lattice2.lattice0_def)
have **: "lattice2.weierstrass_fun = lattice1.weierstrass_fun"
    unfolding lattice1.weierstrass_fun_def lattice2.weierstrass_fun_def
      lattice1.weierstrass_fun_aux_def lattice2.weierstrass_fun_aux_def

    by (simp cong: if_cong)
have "lattice1.modulus = lattice2.modulus"
    unfolding lattice1.modulus_def lattice2.modulus_def
      lattice1.number_e1_def lattice1.number_e2_def lattice1.number_e3_def
      lattice2.number_e1_def lattice2.number_e2_def lattice2.number_e3_def
**
    by (intro arg_cong2[of _ _ _ _ "(/)"] arg_cong2[of _ _ _ _ "(-)"]
      lattice1.weierstrass_fun.lattice_cong)
      (auto simp: lattice1.rel_def add_divide_distrib diff_divide_distrib
        intro: lattice1.lattice_intros)
    thus ?thesis
      by (simp add: lattice1.modulus_eq_modular_lambda lattice2.modulus_eq_modular_lambda)
qed (auto simp: modular_lambda_def)

lemma modular_lambda_conv_jacobi_theta:
  assumes z: "Im z > 0"
  shows "modular_lambda z = jacobi_theta_10 0 z ^ 4 / jacobi_theta_00
0 z ^ 4"
proof -
  interpret std_complex_lattice z
    by unfold_locales (use z in auto)
  show ?thesis
    using modular_lambda_eq_modular_lambda modulus_conv_theta by (simp add: theta_10_def
theta_00_def)
qed

lemma modular_lambda_analytic [analytic_intros]:
  assumes "f analytic_on A" "∧z. z ∈ A ⇒ f z ∉ ℝ"
  shows "(λz. modular_lambda (f z)) analytic_on A"
proof -

```

```

have "(λz. jacobi_theta_10 0 z ^ 4 / jacobi_theta_00 0 z ^ 4) holomorphic_on
{z. Im z > 0}"
  by (intro holomorphic_intros open_halfspace_Im_gt)
      (use assms in <auto simp: jacobi_theta_00_0_left_nonzero>)
also have "?thesis <math>\longleftrightarrow</math> modular_lambda holomorphic_on {z. Im z > 0}"
  by (rule holomorphic_cong) (use modular_lambda_conv_jacobi_theta in
auto)
finally have *: "modular_lambda analytic_on {z. Im z > 0}"
  by (simp add: analytic_on_open open_halfspace_Im_gt)

have "(modular_lambda o uminus) analytic_on {z. Im z < 0}"
  by (intro analytic_on_compose analytic_intros analytic_on_subset[OF
*]) auto
also have "(modular_lambda o uminus) = modular_lambda"
  by (auto simp: fun_eq_iff modular_lambda_uminus)
finally have "modular_lambda analytic_on ({z. Im z < 0} ∪ {z. Im z >
0})"
  using * analytic_on_Un by blast
also have "{z. Im z < 0} ∪ {z. Im z > 0} = -ℝ"
  by (auto simp: complex_is_Real_iff)
finally have **: "modular_lambda analytic_on - ℝ" .
have "(modular_lambda o f) analytic_on A"
  by (intro analytic_on_compose assms(1) analytic_on_subset[OF **])
(use assms(2) in auto)
thus ?thesis
  by (simp add: o_def)
qed

```

```

lemma modular_lambda_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "∧z. z ∈ A ⇒ f z ∉ ℝ"
  shows "(λz. modular_lambda (f z)) holomorphic_on A"
proof -
  from assms(1) have "(modular_lambda o f) holomorphic_on A"
    by (rule holomorphic_on_compose)
      (use assms in <auto intro!: analytic_imp_holomorphic analytic_intros>)
  thus ?thesis
    by (simp add: o_def)
qed

```

```

lemma modular_lambda_meromorphic [meromorphic_intros]:
  assumes "f analytic_on A" "∧z. z ∈ A ⇒ f z ∉ ℝ"
  shows "(λz. modular_lambda (f z)) meromorphic_on A"
  by (rule meromorphic_on_compose[OF _ assms(1) order.refl])
      (use assms(2) in <auto intro!: analytic_intros analytic_on_imp_meromorphic_on>)

```

```

lemma tendsto_modular_lambda [tendsto_intros]:
  assumes "(f ⟶ z) F" "z ∉ ℝ"
  shows "((λz. modular_lambda (f z)) ⟶ modular_lambda z) F"
proof -

```

```

have "modular_lambda analytic_on {z}"
  by (rule analytic_intros) (use assms in auto)
with assms(1) show ?thesis
  by (metis analytic_at_imp_isCont isCont_tendsto_compose)
qed

```

```

lemma continuous_modular_lambda [continuous_intros]:
  "continuous (at x within A) f  $\implies$  f x  $\notin$   $\mathbb{R}$   $\implies$ 
  continuous (at x within A) ( $\lambda$ z. modular_lambda (f z))"
unfolding continuous_def
using tendsto_modular_lambda[of f "f x" "at x within A"]
by (cases "at x within A = bot") (auto simp: Lim_ident_at)

```

```

lemma continuous_on_modular_lambda [continuous_intros]:
  "continuous_on A f  $\implies$  ( $\bigwedge$ x. x  $\in$  A  $\implies$  f x  $\notin$   $\mathbb{R}$ )  $\implies$ 
  continuous_on A ( $\lambda$ z. modular_lambda (f z))"
by (rule continuous_on_compose2[of "- $\mathbb{R}$ " _ _ f])
(auto intro!: holomorphic_on_imp_continuous_on holomorphic_intros)

```

12.8 Klein's J invariant

Klein's J invariant is defined as $(60G_4)^3/\Delta$, or equivalently $E_4^3/(E_4^3 - E_6^2)$. It is exactly the J invariant of a lattice we saw earlier.

Note that there are a number of different conventions, e.g. sometimes one also calls what we would write as $1728J$ the j invariant (with a lower case j).

definition $Klein_J :: "complex \Rightarrow complex"$ where
 $"Klein_J z = (60 * Eisenstein_G 4 z) ^ 3 / modular_discr z"$

```

lemma (in complex_lattice) invariant_J_eq_Klein_J:
  "invariant_J = Klein_J ( $\omega$ 2 /  $\omega$ 1)"
unfolding invariant_J_def discr_eq_modular_discr Klein_J_def invariant_g2_def
  eisenstein_series_eq_Eisenstein_G by (simp add: field_simps)

```

```

lemma Klein_J_conv_modular_lambda:
  assumes "z  $\notin$   $\mathbb{R}$ "
  defines "x  $\equiv$  modular_lambda z"
  shows "Klein_J z = 4 * (1 - x * (1 - x)) ^ 3 / (27 * (x * (1 - x))
  ^ 2)"
proof -
  interpret complex_lattice 1 z
  using assms(1) by unfold_locales (auto simp: fundpair_def)
  show ?thesis
  using invariant_J_conv_modulus
  by (simp add: modulus_eq_modular_lambda invariant_J_eq_Klein_J flip:
  x_def)
qed

```

```

lemma Klein_J_real_eq_0 [simp]: "z ∈ ℝ ⇒ Klein_J z = 0"
  by (simp add: Klein_J_def)

lemma Klein_J_uminus: "Klein_J (-z) = Klein_J z"
  by (simp add: Klein_J_def modular_discr_uminus Eisenstein_G_uminus)

lemma Klein_J_cnj: "Klein_J (cnj z) = cnj (Klein_J z)"
  by (simp add: Klein_J_def Eisenstein_G_cnj modular_discr_cnj)

lemma Klein_J_analytic [analytic_intros]:
  assumes "f analytic_on A" "∧z. z ∈ A ⇒ f z ∉ ℝ"
  shows "(λz. Klein_J (f z)) analytic_on A"
  unfolding Klein_J_def using assms by (auto intro!: analytic_intros
simp: modular_discr_nonzero)

lemma Klein_J_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "∧z. z ∈ A ⇒ f z ∉ ℝ"
  shows "(λz. Klein_J (f z)) holomorphic_on A"
  unfolding Klein_J_def using assms by (auto intro!: holomorphic_intros
simp: modular_discr_nonzero)

lemma continuous_Klein_J [continuous_intros]:
  "continuous (at x within A) f ⇒ f x ∉ ℝ ⇒
    continuous (at x within A) (λz. Klein_J (f z))"
  unfolding Klein_J_def by (intro continuous_intros) (auto simp: modular_discr_nonzero)

lemma continuous_on_Klein_J [continuous_intros]:
  "continuous_on A f ⇒ (∀x∈A. f x ∉ ℝ) ⇒
    continuous_on A (λz. Klein_J (f z))"
  unfolding Klein_J_def by (intro continuous_intros) (auto simp: modular_discr_nonzero)

It is trivial to show that Klein's  $J$  function is invariant under the modular
group. This is Apostol's Theorem 1.16.

theorem Klein_J_apply_modgrp:
  "Klein_J (apply_modgrp f z) = Klein_J z"
proof (cases "z ∈ ℝ")
  case False
  thus ?thesis
    by (simp add: Klein_J_def Eisenstein_G_apply_modgrp modular_discr_apply_modgrp
algebra_simps
      complex_is_Real_iff)
qed auto

lemma Klein_J_plus_int: "Klein_J (z + of_int m) = Klein_J z"
  using Klein_J_apply_modgrp[of "shift_modgrp m" z] by simp

lemma Klein_J_plus1: "Klein_J (z + 1) = Klein_J z"
  using Klein_J_apply_modgrp[of "T_modgrp" z] by simp

```

```
lemma Klein_J_minus_one_over: "Klein_J (-(1/z)) = Klein_J z"
  using Klein_J_apply_modgrp[of "S_modgrp" z] by simp
```

```
lemma Klein_J_cong:
  assumes "w ~Γ z"
  shows "Klein_J w = Klein_J z"
  using assms by (metis Klein_J_apply_modgrp modular_group.rel_def)
```

12.9 Klein's c

```
definition Klein_c :: "nat ⇒ int" where
  "Klein_c n = (let A = of_nat.fps (Abs_fps (divisor_sigma 3));
                 C = fps_left_inverse (fps_shift 1 (Abs_fps ramanujan_tau))
  1
                 in fps_nth (C * (1 + 240 * A) ^ 3) (n + 1))"
```

```
definition fls_Klein_j :: "complex fls"
  where "fls_Klein_j = fls_X_inv + fps_to_fls (Abs_fps (λx. of_int (Klein_c x)))"
```

```
definition fls_Klein_J :: "complex fls"
  where "fls_Klein_J = fls_const (1 / 1728) * fls_Klein_j"
```

```
lemma fls_Klein_J_conv_modular_discr:
```

```
"fls_Klein_J = fps_to_fls ((60 * fps_Eisenstein_G 4) ^ 3) / fps_to_fls
fps_modular_discr"
```

```
proof -
```

```
  define A :: "complex fps" where "A = of_nat.fps (Abs_fps (divisor_sigma
  3))"
```

```
  define B :: "complex fps" where "B = of_int.fps (Abs_fps ramanujan_tau)"
```

```
  define C :: "complex fps" where
```

```
    "C = of_int.fps (fps_left_inverse (fps_shift 1 (Abs_fps ramanujan_tau))
  1)"
```

```
  have "fps_nth B 1 ≠ 0"
```

```
    by (simp add: B_def ramanujan_tau_1)
```

```
  hence [simp]: "B ≠ 0"
```

```
    by auto
```

```
  define F where "F = fps_to_fls ((60 * fps_Eisenstein_G 4) ^ 3) / fps_to_fls
fps_modular_discr"
```

```
  have "F = fps_to_fls ((60 * fps_Eisenstein_G 4) ^ 3) / fps_to_fls fps_modular_discr"
    by (simp add: F_def)
```

```
  also have "... = fls_const (60^3) * fps_to_fls (fps_Eisenstein_G 4 ^
  3) / fps_to_fls fps_modular_discr"
```

```
    by (simp add: power_mult_distrib hom_distrib)
```

```
  also have "fps_Eisenstein_G 4 = fps_const (of_real pi ^ 4 / 45) * fps_Eisenstein_E
  4"
```

```
    by (subst fps_Eisenstein_G_conv_E) (auto simp: zeta_4)
```

```

also have "... ^ 3 = fps_const (of_real pi ^ 12 / 91125) * fps_Eisenstein_E
4 ^ 3"
  by (simp add: power_mult_distrib power_divide)
also have "fps_Eisenstein_E 4 = 1 + 240 * A"
  by (simp add: fps_Eisenstein_E_def hom_distrib A_def flip: divisor_sigma_of_nat)
also have "fps_modular_discr = fps_const (4096 * of_real pi ^ 12) *
B"
  by (simp add: fps_modular_discr_def B_def)
also have "fls_const (60 ^ 3) * fps_to_fls (fps_const (of_real pi ^
12 / 91125) * (1 + 240 * A) ^ 3) /
      fps_to_fls (fps_const (4096 * of_real pi ^ 12) * B) =
      fls_const (1/12^3) * (fps_to_fls ((1 + 240 * A) ^ 3) / fps_to_fls
B)"
  by (simp add: fps_to_fls.hom_mult field_simps fls_const.hom_div fls_const.hom_power
fls_const.hom_mult)

also have B_eq: "B = fps_X * fps_shift 1 B"
  by (simp add: B_def fps_eq_iff)
also have "fps_to_fls ((1 + 240 * A) ^ 3) / fps_to_fls (fps_X * fps_shift
1 B) =
      1 / fls_X * (fps_to_fls ((1 + 240 * A) ^ 3) * inverse (fps_to_fls
(fps_shift 1 B)))"
  by (simp add: divide_simps fls_times_fps_to_fls del: fls_divide_X)
also have "inverse (fps_to_fls (fps_shift 1 B)) = fps_to_fls (inverse
(fps_shift 1 B))"
  by (intro fls_inverse_fps_to_fls subdegree_eq_0) (auto simp: B_def
ramanujan_tau_1)
also have "fps_left_inverse (fps_shift 1 (Abs_fps ramanujan_tau)) 1
*
      fps_shift 1 (Abs_fps ramanujan_tau) = 1"
  unfolding C_def by (intro fps_left_inverse) (auto simp: ramanujan_tau_1)
hence "of_int.fps (fps_left_inverse (fps_shift 1 (Abs_fps ramanujan_tau))
1 *
      fps_shift 1 (Abs_fps ramanujan_tau)) = of_int.fps 1"
  by (simp only: )
hence "C * fps_shift 1 B = 1"
  unfolding of_int.fps_1 by (simp add: C_def B_def)
hence "inverse (fps_shift 1 B) = C"
  by (metis fps_inverse_unique mult.commute)
also have "fps_to_fls ((1 + 240 * A) ^ 3) * fps_to_fls C = fps_to_fls
(C * (1 + 240 * A) ^ 3)"
  by (simp add: fls_times_fps_to_fls mult_ac)
also have "1 / fls_X * ... = fls_shift 1 (fps_to_fls (C * (1 + 240 *
A) ^ 3))"
  by (simp add: fls_shifted_times_simps(2))
also have "C * (1 + 240 * A) ^ 3 = 1 + fps_X * of_int.fps (Abs_fps Klein_c)"
  by (simp add: Klein_c_def C_def A_def B_def fps_eq_iff fps_nth_power_0
flip: of_int.fps_nth)
also have "fls_shift 1 (fps_to_fls ...) = fls_X_inv + fps_to_fls (Abs_fps

```

```

Klein_c)"
  by (rule fls_eqI) (auto simp: nat_add_distrib)
  also have "... = fls_Klein_j"
  by (simp add: fls_Klein_j_def)
  finally show ?thesis unfolding F_def [symmetric]
  by (simp add: field_simps fls_const.hom_div fls_Klein_J_def)
qed

lemma fls_Klein_J_conv_Eisenstein_E:
  "fls_Klein_J =
    fps_to_fls (fps_Eisenstein_E 4 ^ 3) /
    fps_to_fls (fps_Eisenstein_E 4 ^ 3 - fps_Eisenstein_E 6 ^ 2)"
  using fps_modular_discr_nonzero
  unfolding fls_Klein_J_conv_modular_discr fps_modular_discr_conv_Eisenstein
  by (simp add: fps_to_fls.hom_mult fps_to_fls.hom_power fls_const.hom_mult
    fls_const.hom_power
    fls_const.hom_div fps_Eisenstein_G_conv_E divide_simps
    zeta_4
    del: fls_const_mult_const fps_modular_discr_nonzero)

lemma fps_left_inverse_constructor_numeral:
  "fps_left_inverse_constructor a b (numeral n) =
    - (∑ i = 0..pred_numeral n. fps_left_inverse_constructor a b i *
    a $ (Suc (pred_numeral n) - i)) * b"
  unfolding numeral_eq_Suc fps_left_inverse_constructor.simps ..

lemma Klein_c_0: "Klein_c 0 = 744"
  and Klein_c_1: "Klein_c (Suc 0) = 196884"
  and Klein_c_2: "Klein_c 2 = 21493760"
  and Klein_c_3: "Klein_c 3 = 864299970"
  and Klein_c_4: "Klein_c 4 = 20245856256"
  using ramanujan_tau_2 ramanujan_tau_3 ramanujan_tau_4 ramanujan_tau_5
  ramanujan_tau_6
  by (simp_all add: Klein_c_def divisor_sigma_naive fold_atLeastAtMost_nat.simps
    power3_eq_cube Let_def fps_mult_nth atLeastAtMost_nat_numeral_right
    numeral_fps_const fps_left_inverse_constructor_numeral
    flip: numeral_2_eq_2)

lemma fls_Klein_j_subdegree [simp]: "fls_subdegree fls_Klein_j = -1"
  by (intro fls_subdegree_eqI) (auto simp: fls_Klein_j_def Klein_c_0)

lemma fls_Klein_j_nonzero [simp]: "fls_Klein_j ≠ 0"
proof -
  have "fls_nth fls_Klein_j 0 = 744"
  by (simp add: fls_Klein_j_def Klein_c_0)
  thus ?thesis
  by auto
qed

```

```
lemma fls_Klein_J_subdegree [simp]: "fls_subdegree fls_Klein_J = -1"
  by (simp add: fls_Klein_J_def)
```

```
lemma fls_Klein_J_nonzero [simp]: "fls_Klein_J ≠ 0"
  by (simp add: fls_Klein_J_def)
```

12.10 Values at specific points

```
unbundle modfun_region_notation
```

Let $k \geq 2$. The points i and ρ are fixed points of the modular transformations S and ST , respectively. Using this together with the modular transformation law for G_k , it directly follows that $G_k(i) = 0$ unless k is a multiple of 4 and $G_k(\rho) = 0$ unless k is a multiple of 6.

These facts are part of Apostol's Exercise 1.12 and generalise some facts derived in his Theorem 2.7.

The values $G_2(i) = \pi$ and $G_2(\rho) = \frac{2\pi}{\sqrt{3}}$ can be determined in the same fashion once we have proven the transformation law for G_2 .

```
lemma Eisenstein_G_ii_eq_0:
  assumes "k ≠ 2" "¬4 dvd k"
  shows "Eisenstein_G k i = 0"
```

```
proof (cases "even k")
```

```
  case True
```

```
  thus ?thesis
```

```
    using Eisenstein_G_apply_modgrp[of k S_modgrp "i"] assms
```

```
    by (auto elim!: evenE simp: power_neg_one_If split: if_splits)
```

```
qed auto
```

```
lemma Eisenstein_E_ii_eq_0:
```

```
  assumes "k ≠ 2" "¬4 dvd k"
```

```
  shows "Eisenstein_E k i = 0"
```

```
  using assms Eisenstein_G_ii_eq_0[of k] by (cases "k = 0") (auto simp:
Eisenstein_E_def)
```

```
lemma Eisenstein_G_6_ii [simp]: "Eisenstein_G 6 i = 0"
```

```
  by (rule Eisenstein_G_ii_eq_0) auto
```

```
lemma Eisenstein_E_6_ii [simp]: "Eisenstein_E 6 i = 0"
```

```
  by (rule Eisenstein_E_ii_eq_0) auto
```

```
lemma Eisenstein_G_rho_eq_0:
```

```
  assumes "k ≠ 2" "¬6 dvd k"
```

```
  shows "Eisenstein_G k ρ = 0"
```

```
proof (cases "even k")
```

```
  case True
```

```
  show ?thesis
```

```
  proof (rule ccontr)
```

```

    assume nz: "Eisenstein_G k  $\varrho \neq 0$ "
    have "Eisenstein_G k (- (1 / ( $\varrho + 1$ ))) = ( $\varrho + 1$ ) ^ k * Eisenstein_G
k  $\varrho$ "
      using Eisenstein_G_apply_modgrp[of k "S_modgrp * T_modgrp" " $\varrho$ "]
    assms
      by (auto elim!: evenE simp: power_neg_one_If apply_modgrp_mult split:
if_splits)
      also have "-(1 / ( $\varrho + 1$ )) =  $\varrho$ "
        by (auto simp: field_simps modfun_rho_altdef complex_eq_iff Re_divide
Im_divide)
      finally have " $(\varrho + 1) ^ k = 1$ "
        using nz by simp
      also have " $\varrho + 1 = -1 / \varrho$ "
        by (auto simp: complex_eq_iff modfun_rho_altdef field_simps Re_divide
Im_divide)
      finally have " $\varrho ^ k = 1$ "
        using True by (auto simp: field_simps uminus_power_if)
      hence "3 dvd k"
        by (simp add: modfun_rho_power_eq_1_iff)
      with True have "6 dvd k"
        by presburger
      thus False
        using assms by simp
  qed
qed auto

```

```

lemma Eisenstein_E_rho_eq_0:
  assumes "k  $\neq 2$ " "-6 dvd k"
  shows "Eisenstein_E k  $\varrho = 0$ "
  using assms Eisenstein_G_rho_eq_0[of k] by (cases "k = 0") (auto simp:
Eisenstein_E_def)

```

```

lemma Eisenstein_G_4_rho [simp]: "Eisenstein_G 4  $\varrho = 0$ "
  by (rule Eisenstein_G_rho_eq_0) auto

```

```

lemma Eisenstein_E_4_rho [simp]: "Eisenstein_E 4  $\varrho = 0$ "
  by (rule Eisenstein_E_rho_eq_0) auto

```

```

corollary Eisenstein_G_6_rho_nonzero: "Eisenstein_G 6  $\varrho \neq 0$ "
proof -
  have "modular_discr  $\varrho \neq 0$ "
    by (rule modular_discr_nonzero) auto
  thus ?thesis
    by (auto simp: modular_discr_def)
qed

```

```

corollary Eisenstein_E_6_rho_nonzero: "Eisenstein_E 6  $\varrho \neq 0$ "
  by (auto simp: Eisenstein_E_def Eisenstein_G_6_rho_nonzero zeta_Re_gt_1_nonzero)

```

corollary Eisenstein_G_4_ii_nonzero: "Eisenstein_G 4 i \neq 0"

proof -

 have "modular_discr i \neq 0"
 by (rule modular_discr_nonzero) auto
 thus ?thesis
 by (auto simp: modular_discr_def)

qed

corollary Eisenstein_E_4_ii_nonzero: "Eisenstein_E 4 i \neq 0"

 by (auto simp: Eisenstein_E_def Eisenstein_G_4_ii_nonzero zeta_Re_gt_1_nonzero)

corollary Klein_J_rho [simp]: "Klein_J ρ = 0"

 by (simp add: Klein_J_def)

corollary Klein_J_ii [simp]: "Klein_J i = 1"

 using modular_discr_nonzero[of i]
 by (simp add: Klein_J_def modular_discr_def complex_is_Real_iff)

12.11 Consequences for the fundamental region

One application of the Klein J function: We can show that subgroups of the modular group have discrete orbits. That is: every point has a neighbourhood in which no equivalent points are.

Otherwise, if there were a point x and a sequence x_n of points equivalent to it and converging to it, the function $f(z) = J(z) - J(x)$ would have a zero at every x_n and therefore be identically zero by analytic continuation. This would make J a constant function, but since $J(i) = 1$ and $J(\rho) = 0$, this gives a contradiction.

lemma (in modgrp_subgroup) isolated_in_orbit:

 assumes "Im y > 0"
 shows "¬y islimpt orbit x"

proof

 assume *: "y islimpt orbit x"

 have "Klein_J z - Klein_J x = 0" if z: "Im z > 0" for z

 proof (rule analytic_continuation[of "\lambda z. Klein_J z - Klein_J x"])

 show "(λz . Klein_J z - Klein_J x) holomorphic_on {z. Im z > 0}"

 by (auto intro!: holomorphic_intros simp: complex_is_Real_iff)

 show "open {z. Im z > 0}" and "connected {z. Im z > 0}"

 by (auto simp: open_halfspace_Im_gt intro!: convex_connected convex_halfspace_Im_gt)

 show "y islimpt orbit x" by fact

 show "Klein_J z - Klein_J x = 0" if "z \in orbit x" for z

 proof -

 have "z \sim_{Γ} x"

 using that by (auto simp: orbit_def rel_commutes intro: rel_imp_rel)

 then obtain g where g: "apply_modgrp g z = x" "Im z > 0" "Im x

> 0"

 by (auto simp: modular_group.rel_def)

 show ?thesis

```

        using g(2,3) by (simp add: Klein_J_apply_modgrp flip: g(1))
    qed
    qed (use assms z in <auto simp: orbit_def>)
    from this[of "ρ"] and this[of i] show False
        by simp
qed

lemma (in modgrp_subgroup) discrete_orbit: "discrete (orbit x)"
  unfolding discrete_def
proof safe
  fix y assume y: "y ∈ orbit x"
  hence "Im y > 0"
    by (simp add: orbit_def rel_imp_Im_pos(2))
  have *: "orbit y = orbit x"
    by (intro orbit_cong) (use y in <auto simp: orbit_def rel_commutes>)
  have "y isolated_in orbit y"
    using isolated_in_orbit[of y] y * <Im y > 0> isolated_in_islimpt_iff
  by blast
  thus "y isolated_in orbit x"
    by (simp add: *)
qed

lemma (in modgrp_subgroup) eventually_not_rel_at:
  "Im x > 0 ⇒ eventually (λy. ¬rel y x) (at x)"
  using isolated_in_orbit[of x x]
  by (simp add: islimpt_conv_frequently_at frequently_def orbit_def rel_commutes)

lemma (in modgrp_subgroup) closed_orbit [intro]: "closedin (top_of_set
{z. Im z > 0}) (orbit x)"
  using isolated_in_orbit[of _ x] by (auto simp: closedin_limpt orbit_imp_Im_pos)

unbundle no_modfun_region_notation
unbundle no_modgrp_notation

end

```

13 The Dedekind η function

```

theory Dedekind_Eta
imports
  Bernoulli.Bernoulli
  Theta_Inversion
  Complex_Lattices_Theta
  Basic_Modular_Forms
  Dedekind_Sums.Dedekind_Sums
  Pentagonal_Number_Theorem.Pentagonal_Number_Theorem
begin

```

hide_const (open) Unique_Factorization.coprime

13.1 Definition and basic properties

definition dedekind_eta:: "complex \Rightarrow complex" ("η") where
"η z = to_nome (z / 12) * euler_phi (to_nome (2*z))"

lemma dedekind_eta_nonzero [simp]: "Im z > 0 \implies η z \neq 0"
by (auto simp: dedekind_eta_def norm_to_nome norm_power power_less_one_iff)

lemma holomorphic_dedekind_eta [holomorphic_intros]:
assumes "A \subseteq {z. Im z > 0}"
shows "η holomorphic_on A"
using assms unfolding dedekind_eta_def
by (auto intro!: holomorphic_intros simp: norm_to_nome norm_power power_less_one_iff)

lemma holomorphic_dedekind_eta' [holomorphic_intros]:
assumes "f holomorphic_on A" " \bigwedge z. z \in A \implies Im (f z) > 0"
shows "(λz. η (f z)) holomorphic_on A"
using assms unfolding dedekind_eta_def
by (auto intro!: holomorphic_intros simp: norm_to_nome norm_power power_less_one_iff)

lemma analytic_dedekind_eta [analytic_intros]:
assumes "A \subseteq {z. Im z > 0}"
shows "η analytic_on A"
using assms unfolding dedekind_eta_def
by (auto intro!: analytic_intros simp: norm_to_nome norm_power power_less_one_iff)

lemma analytic_dedekind_eta' [analytic_intros]:
assumes "f analytic_on A" " \bigwedge z. z \in A \implies Im (f z) > 0"
shows "(λz. η (f z)) analytic_on A"
using assms unfolding dedekind_eta_def
by (auto intro!: analytic_intros simp: norm_to_nome norm_power power_less_one_iff)

lemma meromorphic_on_dedekind_eta [meromorphic_intros]:
"f analytic_on A \implies (\bigwedge z. z \in A \implies Im (f z) > 0) \implies (λz. η (f z))
meromorphic_on A"
by (rule analytic_on_imp_meromorphic_on) (auto intro!: analytic_intros)

lemma continuous_on_dedekind_eta [continuous_intros]:
"A \subseteq {z. Im z > 0} \implies continuous_on A η"
unfolding dedekind_eta_def
by (intro continuous_intros) (auto simp: norm_to_nome norm_power power_less_one_iff)

lemma continuous_on_dedekind_eta' [continuous_intros]:
assumes "continuous_on A f" " \bigwedge z. z \in A \implies Im (f z) > 0"
shows "continuous_on A (λz. η (f z))"
using assms unfolding dedekind_eta_def
by (intro continuous_intros) (auto simp: norm_to_nome norm_power power_less_one_iff)

```

lemma tendsto_dedekind_eta [tendsto_intros]:
  assumes "(f ⟶ c) F" "Im c > 0"
  shows   "((λx. η (f x)) ⟶ η c) F"
  unfolding dedekind_eta_def using assms
  by (intro tendsto_intros assms) (auto simp: norm_to_norm norm_power
power_less_one_iff)

lemma tendsto_at_cusp_dedekind_eta [tendsto_intros]: "(η ⟶ 0) at_i∞"
proof -
  have "filterlim (λx::real. x / 12) at_top at_top"
    by real_asymp
  hence "filterlim (λz. Im z / 12) at_top at_i∞"
    by (rule filterlim_at_ii_infl)
  hence *: "(λz. to_norm (z / 12)) ⟶ 0) at_i∞"
    by (intro tendsto_0_to_norm) simp_all

  have "filterlim (λx::real. 2 * x) at_top at_top"
    by real_asymp
  hence "filterlim (λz. 2 * Im z) at_top at_i∞"
    by (rule filterlim_at_ii_infl)
  hence **: "(λz. to_norm (2 * z)) ⟶ 0) at_i∞"
    by (intro tendsto_0_to_norm) simp_all

  have "((λz. to_norm (z / 12) * euler_phi (to_norm (2*z))) ⟶ 0 *
euler_phi 0) at_i∞"
    by (intro tendsto_intros * **) auto
  thus ?thesis
    by (simp add: dedekind_eta_def [abs_def])
qed

lemma dedekind_eta_plus1:
  assumes z: "Im z > 0"
  shows   "η (z + 1) = cis (pi/12) * η z"
proof -
  have "η (z + 1) = to_norm ((z + 1) / 12) * euler_phi (to_norm (2 * (z
+ 1)))"
    by (simp add: dedekind_eta_def)
  also have "to_norm (2 * (z + 1)) = to_norm (2 * z)"
    by (simp add: to_norm_add ring_distrib)
  also have "to_norm ((z + 1) / 12) = to_norm (1/12) * to_norm (z / 12)"
    by (simp add: to_norm_add add_divide_distrib)
  also have "to_norm (1/12) = cis (pi/12)"
    by (simp add: to_norm_def cis_conv_exp)
  also have "... * to_norm (z / 12) * euler_phi (to_norm (2*z)) = cis
(pi/12) * η z"
    by (simp add: dedekind_eta_def mult_ac)
  finally show ?thesis by (simp add: cis_conv_exp mult_ac)
qed

```

```

lemma dedekind_eta_plus_nat:
  assumes z: "Im z > 0"
  shows "η (z + of_nat n) = cis (pi * n / 12) * η z"
proof (induction n)
  case (Suc n)
  have "η (z + of_nat (Suc n)) = η (z + of_nat n + 1)"
    by (simp add: add_ac)
  also have "... = cis (pi/12) * η (z + of_nat n)"
    using z by (intro dedekind_eta_plus1) auto
  also have "η (z + of_nat n) = cis (pi*n/12) * η z"
    by (rule Suc.IH)
  also have "cis (pi/12) * (cis (pi*n/12) * η z) = cis (pi*Suc n/12) *
η z"
    by (simp add: ring_distrib add_divide_distrib exp_add mult_ac cis_mult)
  finally show ?case .
qed auto

```

```

lemma dedekind_eta_plus_int:
  assumes z: "Im z > 0"
  shows "η (z + of_int n) = cis (pi*n/12) * η z"
proof (cases "n ≥ 0")
  case True
  thus ?thesis
    using dedekind_eta_plus_nat[OF assms, of "nat n"] by simp
next
  case False
  thus ?thesis
    using dedekind_eta_plus_nat[of "z + of_int n" "nat (-n)"] assms
    by (auto simp: cis_mult field_simps)
qed

```

The logarithmic derivative of η is, up to a constant factor, the “forbidden” Eisenstein series G_2 . This follows relatively easily from the logarithmic derivative of Euler’s function ϕ and the Fourier expansion of G_2 , both of which involve the Lambert series $\sum_{k=1}^{\infty} k \frac{q^k}{1-q^k}$.

```

theorem deriv_dedekind_eta:
  assumes z: "Im z > 0"
  shows "deriv η z = i / (4 * of_real pi) * Eisenstein_G 2 z * η z"
proof -
  define f :: "complex ⇒ complex" where "f = deriv euler_phi"
  have *: "(euler_phi has_field_derivative f q) (at q within A)" if "norm
q < 1" for q A
    unfolding f_def using that by (auto intro!: analytic_derivI analytic_intros)
  have [derivative_intros]:
    "((λz. euler_phi (g z)) has_field_derivative f (g z) * g') (at z within
A)"
    if "(g has_field_derivative g') (at z within A)" "norm (g z) < 1" for
g g' z A

```

```

    by (rule DERIV_chain'[OF that(1) *]) fact

define L where "L = lambert_complex_of_nat (to_nome (2 * z))"
have L_eq: "L = 1 / 24 - Eisenstein_G 2 z / (8 * pi ^ 2)"
  by (subst Eisenstein_G_fourier_expansion)
    (use z in <simp_all add: L_def zeta_2 field_simps to_q_conv_to_nome>)

have "deriv dedekind_eta z = i * pi * (dedekind_eta z / 12 +
    2 * (to_nome (z / 12) * (to_nome (2*z)
* f (to_nome (2*z))))))"
  by (rule DERIV_imp_deriv)
    (use z in <auto intro!: derivative_eq_intros DERIV_imp_deriv
      simp: norm_to_nome dedekind_eta_def [abs_def] algebra_simps>)
also have "to_nome (2 * z) * f (to_nome (2 * z)) = -L * euler_phi (to_nome
(2 * z))"
  unfolding f_def by (subst deriv_euler_phi) (use z in <auto simp: norm_to_nome
L_def>)
also have "to_nome (z / 12) * ... = -L * dedekind_eta z"
  by (simp add: dedekind_eta_def)
finally have "deriv η z = (i * pi * (1 / 12 - 2 * L)) * η z"
  using z by (simp add: field_simps)
also have "(i * pi * (1 / 12 - 2 * L)) = i / (4 * of_real pi) * Eisenstein_G
2 z"
  unfolding L_eq by (simp add: power2_eq_square)
  finally show ?thesis .
qed

lemma has_field_derivative_dedekind_eta:
  assumes "(f has_field_derivative f') (at x within A)" "Im (f x) > 0"
  shows "((λx. η (f x)) has_field_derivative
    (i / (4 * of_real pi) * Eisenstein_G 2 (f x) * η (f x) * f'))
    (at x within A)"
proof (rule DERIV_chain2[OF _ assms(1)])
  have "(η has_field_derivative deriv η (f x)) (at (f x))"
    by (rule analytic_derivI) (use assms(2) in <auto intro!: analytic_intros>)
  thus "(η has_field_derivative i / (4 * complex_of_real pi) *
    Eisenstein_G 2 (f x) * η (f x)) (at (f x))"
    by (subst (asm) deriv_dedekind_eta) (use assms(2) in auto)
qed

```

13.2 Relation to the Jacobi ϑ functions

```

lemma dedekind_eta_conv_jacobi_theta_01:
  assumes t: "Im t > 0"
  shows "η t = to_nome (t / 12) * jacobi_theta_01 (-t/2) (3 * t)"
proof -
  include qepochhammer_inf_notation
  define q where "q = to_nome t"
  have q: "q ≠ 0" "norm q < 1"

```

```

using t by (auto simp: q_def norm_to_nome)

have "η t = to_nome (t / 12) * euler_phi (q ^ 2)"
  unfolding dedekind_eta_def by (simp add: q_def to_nome_power)
also have "euler_phi (q ^ 2) = ramanujan_theta (- q^2) (-(q ^ 4))"
  by (subst pentagonal_number_theorem_complex)
  (use q in <simp_all add: norm_power power_less_1_iff>)
also have "ramanujan_theta (-(q ^ 2)) (-(q ^ 4)) = jacobi_theta_nome
(-1/q) (q^3)"
  using q by (simp add: jacobi_theta_nome_def eval_nat_numeral mult_ac)
also have "... = jacobi_theta_00 (-t/2 + 1/2) (3 * t)"
  by (simp add: jacobi_theta_00_def q_def to_nome_power to_nome_diff
power_divide diff_divide_distrib)
also have "... = jacobi_theta_01 (-t/2) (3 * t)"
  by (simp add: jacobi_theta_01_def)
finally show ?thesis .
qed

lemma jacobi_theta_01_nw_conv_dedekind_eta:
  assumes t: "Im t > 0"
  shows "jacobi_theta_01 0 t = η (t/2) ^ 2 / η t"
proof -
  include qpochhammer_inf_notation
  define q where "q = to_nome t"
  have q: "q ≠ 0" "norm q < 1"
    using t by (auto simp: q_def norm_to_nome)
  have nz: "(q^2 ; q^2)∞ ≠ 0"
    by (rule qpochhammer_inf_nonzero) (use q in <auto simp: norm_power
power_less_1_iff>)
  have eq: "(q ; q)∞ = (q ; q^2)∞ * (q^2 ; q^2)∞"
    using prod_qpochhammer_inf_group[of q 2 q] q by (simp add: eval_nat_numeral)

  have "jacobi_theta_01 0 t = jacobi_theta_nome (-1) q"
    by (simp add: q_def jacobi_theta_01_def jacobi_theta_00_def)
  also have "... = (q^2 ; q^2)∞ * (q ; q^2)∞ ^ 2"
    by (subst jacobi_theta_nome_triple_product_complex) (use q in <simp_all
add: power2_eq_square>)
  also have "... = euler_phi q ^ 2 / euler_phi (q ^ 2)"
    by (simp add: field_simps euler_phi_def eq power2_eq_square)
  also have "... = η (t/2) ^ 2 / η t"
    by (simp add: dedekind_eta_def power_mult_distrib to_nome_power q_def)
  finally show ?thesis .
qed

lemma jacobi_theta_00_nw_conv_dedekind_eta:
  assumes t: "Im t > 0"
  shows "jacobi_theta_00 0 t = η ((t+1)/2) ^ 2 / η (t+1)"
proof -
  include qpochhammer_inf_notation

```

```

define q where "q = to_nome t"
have q: "q ≠ 0" "norm q < 1"
  using t by (auto simp: q_def norm_to_nome)
have nz: "(q^2 ; q^2)∞ ≠ 0"
  by (rule qpochhammer_inf_nonzero) (use q in <auto simp: norm_power
power_less_1_iff>)
have eq: "(-q ; -q)∞ = (- q ; q^2)∞ * (q^2 ; q^2)∞"
  using prod_qpochhammer_inf_group[of "-q" 2 "-q"] q by (simp add: eval_nat_numeral)

have "jacobi_theta_00 0 t = jacobi_theta_nome 1 q"
  by (simp add: q_def jacobi_theta_00_def)
also have "... = (q^2 ; q^2)∞ * (-q ; q^2)∞ ^ 2"
  by (subst jacobi_theta_nome_triple_product_complex) (use q in <simp_all
add: power2_eq_square>)
also have "... = euler_phi (-q) ^ 2 / euler_phi (q ^ 2)"
  using nz by (simp add: field_simps euler_phi_def power2_eq_square
eq)
also have "... = η ((t+1)/2) ^ 2 / η (t + 1)"
  by (simp add: dedekind_eta_def power_mult_distrib to_nome_power
to_nome_add add_divide_distrib q_def)
finally show ?thesis .
qed

lemma jacobi_theta_00_nw_conv_dedekind_eta':
  assumes t: "Im t > 0"
  shows "jacobi_theta_00 0 t = η t ^ 5 / (η (t/2) * η (2*t)) ^ 2"
proof -
  include qpochhammer_inf_notation
  define q where "q = to_nome t"
  have q: "q ≠ 0" "norm q < 1"
    using t by (auto simp: q_def norm_to_nome)
  have nz': "(q ; q)∞ ≠ 0"
    by (rule qpochhammer_inf_nonzero) (use q in <auto simp: norm_power
power_less_1_iff>)
  have nz'': "(- q^2 ; q^2)∞ ≠ 0"
    by (rule qpochhammer_inf_nonzero) (use q in <auto simp: norm_power
power_less_1_iff>)
  have nz: "(q^2 ; q^2)∞ ≠ 0"
    by (rule qpochhammer_inf_nonzero) (use q in <auto simp: norm_power
power_less_1_iff>)
  have eq: "(-q ; q)∞ = (- q ; q^2)∞ * (-q^2 ; q^2)∞"
    using prod_qpochhammer_inf_group[of q 2 "-q"] q by (simp add: eval_nat_numeral)

  have "η t ^ 5 / (η (t/2) * η (2*t)) ^ 2 =
    to_nome (5 * t / 12) / to_nome (t / 12) / to_nome (t / 3) *
    euler_phi (q ^ 2) ^ 5 / (euler_phi q * euler_phi (q ^ 4))^2"
    by (simp add: dedekind_eta_def power_mult_distrib to_nome_power q_def)
  also have "to_nome (5 * t / 12) / to_nome (t / 12) / to_nome (t / 3)

```

```

=
      to_nome (5 * t / 12 - t / 12 - t / 3)"
    unfolding to_nome_diff ..
  also have "5 * t / 12 - t / 12 - t / 3 = 0"
    by simp
  also have "to_nome 0 * euler_phi (q^2) ^ 5 / (euler_phi q * euler_phi
(q ^ 4))^2 =
      (q^2 ; q^2)_∞ ^ 5 / ((q ; q)_∞ * (q ^ 4 ; q ^ 4)_∞)^2"
    by (simp add: euler_phi_def)
  finally have "η t ^ 5 / (η (t / 2) * η (2 * t))^2 =
      (q^2 ; q^2)_∞ ^ 5 / ((q ; q)_∞ * (q ^ 4 ; q ^ 4)_∞)^2" .
  also have "... = (q^2 ; q^2)_∞ * (q^2 ; q^2)_∞ ^ 2 / ((q ; q)_∞ * ((q ^ 4
; q ^ 4)_∞ / (q^2 ; q^2)_∞))^2"
    by (simp add: eval_nat_numeral)
  also have "(q ^ 4 ; q ^ 4)_∞ / (q^2 ; q^2)_∞ = (-q^2 ; q^2)_∞"
    using qpochhammer_inf_square[of "q^2" "q^2"] q nz
    by (simp add: norm_power power_less_1_iff field_simps)
  also have "(q^2 ; q^2)_∞ * (q^2 ; q^2)_∞ ^ 2 / ((q ; q)_∞ * (-q^2 ; q^2)_∞)^2
=
      (q^2 ; q^2)_∞ * ((q^2 ; q^2)_∞ / (q ; q)_∞ / (-q^2 ; q^2)_∞) ^
2"
    by (simp add: power_divide power_mult_distrib)
  also have "(q^2 ; q^2)_∞ / (q ; q)_∞ = (-q ; q)_∞"
    using qpochhammer_inf_square[of "q" "q"] q nz'
    by (simp add: norm_power power_less_1_iff field_simps)
  also have "(-q ; q)_∞ / (-q^2 ; q^2)_∞ = (-q ; q^2)_∞"
    using eq_nz'' by simp
  also have "(q^2 ; q^2)_∞ * (-q ; q^2)_∞^2 = jacobi_theta_nome 1 q"
    by (subst jacobi_theta_nome_triple_product_complex) (use q in <simp_all
add: power2_eq_square>)
  also have "... = jacobi_theta_00 0 t"
    by (simp add: q_def jacobi_theta_00_def)
  finally show ?thesis ..
qed

```

lemma jacobi_theta_10_nw_conv_dedekind_eta:

```

  assumes t: "Im t > 0"
  shows "jacobi_theta_10 0 t = 2 * η (2*t) ^ 2 / η t"
proof -
  include qpochhammer_inf_notation
  define q where "q = to_nome t"
  have q: "q ≠ 0" "norm q < 1"
    using t by (auto simp: q_def norm_to_nome)
  have nz: "(q^2 ; q^2)_∞ ≠ 0"
    by (rule qpochhammer_inf_nonzero) (use q in <auto simp: norm_power
power_less_1_iff>)

```

```

  have "2 * η (2*t) ^ 2 / η t =
      2 * (to_nome (t / 3) / to_nome (t / 12)) * euler_phi (q^4) ^

```

```

2 / euler_phi (q^2)"
  by (auto simp: dedekind_eta_def power_mult_distrib to_nome_power q_def)
  also have "to_nome (t / 3) / to_nome (t / 12) = to_nome (t / 3 - t /
12)"
  by (subst to_nome_diff) auto
  also have "t / 3 - t / 12 = t / 4"
  by auto
  finally have *: "2 * η (2*t) ^ 2 / η t =
                2 * to_nome (t/4) * euler_phi (q^4) ^ 2 / euler_phi
(q^2)" .

  have "jacobi_theta_10 0 t = to_nome (t / 4) * jacobi_theta_nome q q"
  by (simp add: q_def jacobi_theta_10_def jacobi_theta_00_def to_nome_power)
  also have "... = to_nome (t / 4) * ((q^2 ; q^2)_∞ * (-q^2 ; q^2)_∞ * (-1
; q^2)_∞)"
  by (subst jacobi_theta_nome_triple_product_complex) (use q in <auto
simp: power2_eq_square>)
  also have "(-1 ; q^2)_∞ = 2 * (-q^2 ; q^2)_∞"
  using qpochhammer_inf_mult_q[of "q^2" "-1"] q by (simp add: norm_power
power_less_1_iff)
  also have "(q^2 ; q^2)_∞ * (-q^2 ; q^2)_∞ * (2 * (-q^2 ; q^2)_∞) =
            2 * ((q^2 ; q^2)_∞ * (-q^2 ; q^2)_∞) ^ 2 / (q^2 ; q^2)_∞"
  using nz by (simp add: power2_eq_square)
  also have "... = 2 * (q ^ 4 ; q ^ 4)_∞ ^ 2 / (q^2 ; q^2)_∞"
  by (subst qpochhammer_inf_square) (use q in <auto simp: norm_power
power_less_1_iff>)
  also have "... = 2 * euler_phi (q^4) ^ 2 / euler_phi (q^2)"
  using nz by (simp add: euler_phi_def power2_eq_square)
  also have "to_nome (t / 4) * ... = 2 * η (2*t) ^ 2 / η t"
  using * by simp
  finally show ?thesis .
qed

lemma jacobi_theta_00_01_10_nw_conv_dedekind_eta:
  assumes t: "Im t > 0"
  shows "jacobi_theta_00 0 t * jacobi_theta_01 0 t * jacobi_theta_10
0 t = 2 * η t ^ 3"
  using t by (simp add: jacobi_theta_00_nw_conv_dedekind_eta' field_simps
eval_nat_numeral
                jacobi_theta_01_nw_conv_dedekind_eta jacobi_theta_10_nw_conv_dedekind_eta)

```

Since theta nullwert functions can be expressed as quotients of Dedekind's η function, we also get the following deep connection between the discriminant of a complex lattice and η .

This can also alternatively be derived very elegantly using modular forms. More precisely: η^24 and the modular discriminant are both cusp forms of weight 12 and that the space of cusp forms of weight 12 is one-dimensional. However, since we already have access to the theta functions and the above

connections to the lattice properties, this proof is very simple now as well, without using the heavy tooling of modular forms.

theorem (in *complex_lattice_Im_pos*) *discr_conv_dedekind_eta*:

"discr = 4096 * (pi / ω1) ^ 12 * dedekind_eta τ ^ 24"

proof -

have "discr = (4 * (e₁ - e₂) * (e₁ - e₃) * (e₃ - e₂))²"

by (simp add: *discr_altdef power2_commute power_mult_distrib*)

also have "... = 16 * (pi / ω1) ^ 12 * (ϑ₀₀(0) * ϑ₀₁(0) * ϑ₁₀(0)) ^ 8"

unfolding *discr_altdef unfolding e12_conv_theta e13_conv_theta e32_conv_theta*

by (simp add: *power_mult_distrib power_divide mult_ac*)

also have "ϑ₀₀(0) * ϑ₀₁(0) * ϑ₁₀(0) = 2 * dedekind_eta τ ^ 3 "

by (simp add: *theta_00_def theta_01_def theta_10_def*

jacobi_theta_00_01_10_nw_conv_dedekind_eta Im_ratio_pos)

finally show "discr = 4096 * (pi / ω1) ^ 12 * dedekind_eta τ ^ 24"

by (simp add: *power_mult_distrib*)

qed

13.3 The inversion identity

The inversion identity for Jacobi's ϑ function together with the Jacobi triple product allows us to give a rather short proof of the inversion law for η . This is remarkable: Apostol spends the majority of the chapter on proving this.

We would like to thank Alexey Ustinov, who answered a question of ours on MathOverflow and clarified how to prove the following lemma.

lemma *dedekind_eta_minus_one_over_aux*:

assumes "Im t > 0"

shows "jacobi_theta_10 (1 / 6) (t / 3) =

of_real (sqrt 3) * to_nome (t / 12) * jacobi_theta_01 (-t / 2) (3 * t)"

proof -

include *qepochhammer_inf_notation*

define q where "q = to_nome (t / 12)"

define r where "r = to_nome (1 / 6)"

define r' where "r' = to_nome (2/3)"

have *cos_120'*: "cos (pi * 2 / 3) = -1/2"

using *cos_120* by (simp add: *field_simps*)

have *sin_120'*: "sin (pi * 2 / 3) = sqrt 3 / 2"

using *sin_120* by (simp add: *field_simps*)

have [*simp*]: "q ≠ 0" "r ≠ 0"

by (auto simp: *q_def r_def*)

have q: "norm q < 1"

using *assms* by (simp add: *q_def norm_to_nome*)

have [*simp*]: "norm (q ^ n) < 1 ↔ n > 0" for n

using q by (auto simp: *norm_power power_less_1_iff*)

```

have "jacobi_theta_10 (1 / 6) (t / 3) = r * q * jacobi_theta_nome (r^2
* q^4) (q^4)"
  by (simp add: jacobi_theta_10_def jacobi_theta_00_def power2_eq_square
      q_def r_def to_nome_power add_ac flip: to_nome_add)
also have "... = r * q * ((q^8 ; q^8)_∞ * ((-r^2)) * q^8 ; q ^ 8)_∞
* (-1/r^2) ; q^8)_∞"
  by (subst jacobi_theta_nome_triple_product_complex)
      (auto simp: q norm_power power_less_one_iff algebra_simps)
also have "-(r^2) = r'^2"
proof -
  have "-(r^2) = to_nome (1 + 1/3)"
    unfolding to_nome_add by (simp add: r_def to_nome_power)
  also have "... = r' ^ 2"
    by (simp add: r'_def to_nome_power)
  finally show ?thesis .
qed
also have "-(1/r^2) = r'"
  by (auto simp: r_def r'_def to_nome_power field_simps simp flip: to_nome_add)
also have "(r' ; q ^ 8)_∞ = (r' * q ^ 8 ; q ^ 8)_∞ * (1 - r')"
  by (subst qpochhammer_inf_mult_q) auto
finally have "jacobi_theta_10 (1 / 6) (t / 3) =
  r * (1 - r') * q * (∏ k<3. (r' ^ k * q ^ 8 ; q ^ 8)_∞)"
  by (simp add: numeral_3_eq_3 mult_ac power2_eq_square)
also have "r * (1 - r') = of_real (sqrt 3)"
  by (simp add: r_def r'_def to_nome_def exp_eq_polar complex_eq_iff
      cos_30 sin_30 cos_120' sin_120' field_simps)
also have "(∏ k<3. (r' ^ k * q ^ 8 ; q ^ 8)_∞) = (q ^ 24 ; q ^ 24)_∞"
proof -
  interpret primroot_cis 3 1
  rewrites "cis (2 * pi * 1 / 3) ≡ r'" and
    "cis (2 * pi * j * of_int 1 / of_nat 3) ≡ r' ^ j" for j
  by unfold_locales (auto simp: r'_def cis_conv_to_nome to_nome_power
field_simps)
  show ?thesis
    using prod_qpochhammer_group_cis[of "q^8" "q^8"] by simp
qed
finally have eq1: "jacobi_theta_10 (1 / 6) (t / 3) =
  complex_of_real (sqrt 3) * q * (q ^ 24 ; q ^ 24)_∞"
.

have "jacobi_theta_01 (-t / 2) (3 * t) = jacobi_theta_nome (r^6 / q^12)
(q^36)"
  by (simp add: jacobi_theta_01_def jacobi_theta_00_def power2_eq_square
      q_def r_def to_nome_power flip: to_nome_add to_nome_diff
to_nome_minus)
      (simp add: to_nome_diff to_nome_minus field_simps)?
  also have "... = (q^72; q^72)_∞ * (q^48 ; q^72)_∞ * ((q^36 / q^12) ;
q^72)_∞"

```

```

    by (subst jacobi_theta_nome_triple_product_complex)
      (auto simp: q_norm_power power_less_one_iff to_nome_power field_simps
r_def)
  also have "q36 / q12 = q24"
    by (auto simp: field_simps)
  finally have "jacobi_theta_01 (- t / 2) (3 * t) = (∏k<3. (q24 * (q24)k
; q72)∞)"
    by (simp add: numeral_3_eq_3 mult_ac)
  also have "... = (q24 ; q24)∞"
    using prod_qpochhammer_inf_group[of "q24" 3 "q24"] by simp
  finally have "jacobi_theta_01 (-t/2) (3*t) = (q24; q24)∞" .
  hence eq2: "of_real (sqrt 3) * to_nome (t / 12) * jacobi_theta_01 (-t/2)
(3*t) =
      of_real (sqrt 3) * q * (q24; q24)∞"
    by (simp add: q_def)

  from eq1 eq2 show ?thesis
    by simp
qed

theorem dedekind_eta_minus_one_over:
  assumes t: "Im t > 0"
  shows "η (-1/t) = csqrt (-i*t) * η t"
proof -
  have [simp]: "t ≠ 0"
    using t by auto
  have "η (-1/t) = to_nome (- (1 / (t * 12))) * jacobi_theta_01 (1 / (t
* 2)) (-1 / (t / 3))"
    by (subst dedekind_eta_conv_jacobi_theta_01) (use assms in <auto simp:
Im_complex_div_lt_0>)
  also have "... = csqrt ((1/3) * (-i * t)) * jacobi_theta_10 (1 / 6) (t
/ 3)"
    by (subst jacobi_theta_01_minus_one_over)
      (auto simp: to_nome_diff to_nome_minus to_nome_add field_simps
t power2_eq_square)
  also have "csqrt ((1/3) * (-i * t)) = csqrt (-i * t) / sqrt 3"
    by (subst csqrt_mult) (use Arg_eq_0[of "1/3"] in <auto simp: Arg_bounded
real_sqrt_divide>)
  also have "jacobi_theta_10 (1 / 6) (t / 3) =
      of_real (sqrt 3) * to_nome (t / 12) * jacobi_theta_01 (-t
/ 2) (3 * t)"
    by (rule dedekind_eta_minus_one_over_aux) fact
  also have "... = sqrt 3 * η t"
    using t by (simp add: jacobi_theta_10_def dedekind_eta_conv_jacobi_theta_01)
  finally show ?thesis
    by simp
qed

```

13.4 General transformation law

From our results so far, it is easy to see that η^{24} is a modular form of weight 12. Thus it follows that if $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \text{SL}(2)$ is a modular transformation, then $\eta(Az) = \epsilon(A)\sqrt{z}\eta(z)$, where $\epsilon(A)$ is a 24-th unit root that depends on A but not on z .

In the this section, we will give a concrete definition of this 24-th root ϵ in terms of A .

definition `dedekind_eps` :: "modgrp \Rightarrow complex" (" ϵ ") where

```
" $\epsilon$  f = (let f = abs f in
  (if is_singular_modgrp f then
    cis (pi * ((modgrp_a f + modgrp_d f) / (12 * modgrp_c f) -
      dedekind_sum (modgrp_d f) (modgrp_c f) - 1 / 4))
  else
    cis (pi * modgrp_b f / 12)
  ))"
```

lemma `dedekind_eps_1` [simp]: "`dedekind_eps 1 = 1`"
by (simp add: `dedekind_eps_def`)

lemma `dedekind_eps_shift` [simp]: " ϵ (`shift_modgrp m`) = cis (pi * m / 12)"
by (simp add: `dedekind_eps_def dedekind_sum_def`)

lemma `dedekind_eps_S` [simp]: "`dedekind_eps S_modgrp = cis (-pi / 4)`"
by (simp add: `dedekind_eps_def dedekind_sum_def complex_eq_iff`)

lemma `dedekind_eps_abs` [simp]: "`dedekind_eps (abs f) = dedekind_eps f`"
by (simp add: `dedekind_eps_def`)

lemma `dedekind_eps_uminus` [simp]: "`dedekind_eps (-f) = dedekind_eps f`"
by (simp add: `dedekind_eps_def`)

lemma `is_singular_modgrp_abs_iff` [simp]: "`is_singular_modgrp (abs f)`
 \longleftrightarrow `is_singular_modgrp f`"
by transfer (auto split: `if_splits`)

lemma `dedekind_eps_shift_right` [simp]: " ϵ (`f * shift_modgrp m`) = cis (pi * m / 12) * ϵ f"

proof (cases "`is_singular_modgrp f`")

case True

have [simp]: " $|f * \text{shift_modgrp } m| = |f| * \text{shift_modgrp } m$ "

by (auto simp: `modgrp_eq_iff abs_modgrp_altdef`)

have [simp]: "`modgrp_c (abs f) \neq 0`"

using True by (simp add: `is_singular_modgrp_altdef abs_modgrp_altdef`)

have "`dedekind_sum (modgrp_c |f| * m + modgrp_d |f|) (modgrp_c |f|) =`

```

        dedekind_sum (modgrp_d |f|) (modgrp_c |f|)"
    proof (intro dedekind_sum_cong)
        have "[modgrp_c |f| * m + modgrp_d |f| = 0 + modgrp_d |f|] (mod modgrp_c
|f|)"
            by (intro cong_add) (auto simp: Cong.cong_def)
        thus "[modgrp_c |f| * m + modgrp_d |f| = modgrp_d |f|] (mod modgrp_c
|f|)"
            by simp
    qed (use coprime_modgrp_c_d[of "|f|"] in <auto simp: Rings.coprime_commute>)
    thus ?thesis using True
        by (auto simp add: dedekind_eps_def add_divide_distrib ring_distrib
is_singular_modgrp_times_iff Let_def
            simp flip: cis_mult cis_divide)
    next
    case False
    define n where "n = modgrp_b (abs f)"
    have f: "abs f = shift_modgrp n"
        unfolding n_def using False by (metis not_singular_modgrpD)
    have "abs (f * shift_modgrp m) = shift_modgrp (n + m)"
        using f by (metis abs_modgrp_congs(2) abs_shift_modgrp shift_modgrp_add)
    also have "ε ... = cis (pi * m / 12) * ε f"
        by (simp add: f dedekind_eps_def cis_mult ring_distrib add_divide_distrib
add_ac)
    finally show ?thesis by simp
qed

lemma dedekind_eps_shift_left:
    "ε (shift_modgrp m * f) = cis (pi * m / 12) * ε f"
proof -
    have *: "ε (shift_modgrp m * f) = cis (pi * m / 12) * ε f" if c: "modgrp_c
f ≥ 0" for f
        proof (cases "is_singular_modgrp f")
            case True
            have [simp]: "modgrp_c f ≠ 0"
                using True by (simp add: is_singular_modgrp_altdef)
            have a: "modgrp_a (shift_modgrp m * f) = modgrp_a f + m * modgrp_c
f"
                unfolding shift_modgrp_code using modgrp.unimodular[of f] c
                by (subst (3) modgrp_abcd [symmetric], subst times_modgrp_code)
            (auto simp: modgrp_a_code algebra_simps)
            show ?thesis using True c
                by (auto simp: dedekind_eps_def a add_divide_distrib ring_distrib
Let_def abs_modgrp_altdef
                    simp flip: cis_mult cis_divide)
        next
        case False
        then obtain n where *: "abs f = shift_modgrp n"
            using not_singular_modgrpD by blast
        have "ε (shift_modgrp m * f) = ε |shift_modgrp m * f|"

```

```

    by simp
  also have "|shift_modgrp m * f| = |shift_modgrp m * |f||"
    by (intro abs_modgrp_congs) auto
  also have "... = shift_modgrp (m + n)"
    by (simp add: * flip: shift_modgrp_add)
  also have "ε ... = cis (pi * (real_of_int m + real_of_int n) / 12)"
    by simp
  also have "... = cis (pi * m / 12) * ε |f|"
    unfolding * by (simp add: ring_distrib add_divide_distrib cis_mult)
  finally show ?thesis
    by simp
qed
show ?thesis
  using *[of f] *[of "-f"] by (cases "modgrp_c f ≥ 0") auto
qed

```

```

lemma dedekind_eps_S_right:
  assumes "is_singular_modgrp f" "modgrp_d f ≠ 0"
  shows "ε (f * S_modgrp) = cis (-sgn (modgrp_d f) * sgn (modgrp_c
f) * pi / 4) * ε f"
proof -
  have *: "ε (f * S_modgrp) = cis (-sgn (modgrp_d f) * pi / 4) * ε f"
    if c: "modgrp_c f ≥ 0" and f: "is_singular_modgrp f" "modgrp_d f
≠ 0" for f
  proof -
    note [simp del] = div_mult_self3 div_mult_self4 div_mult_self2 div_mult_self1
    define a b c d where "a = modgrp_a f" "b = modgrp_b f" "c = modgrp_c
f" "d = modgrp_d f"
    have "c > 0"
      using f c unfolding is_singular_modgrp_altdef a_b_c_d_def by auto
    from f have [simp]: "d ≠ 0"
      by (auto simp: a_b_c_d_def)
    have "coprime c d"
      unfolding a_b_c_d_def by (intro coprime_modgrp_c_d)
    have det: "a * d - b * c = 1"
      unfolding a_b_c_d_def by (rule modgrp_abcd_det)
    hence det': "a * d = b * c + 1"
      by linarith

    have "pole_modgrp f ≠ (0 :: real)"
      using f by transfer (auto split: if_splits)
    hence sing: "is_singular_modgrp (f * S_modgrp)"
      by (auto simp: is_singular_modgrp_times_iff)

  show ?thesis
proof (cases d "0 :: int" rule: linorder_cases)
  case greater
    have [simp]: "modgrp_a |f * S_modgrp| = b"

```

```

    using greater unfolding a_b_c_d_def by transfer auto
    have [simp]: "modgrp_b |f * S_modgrp| = -a"
    using greater unfolding a_b_c_d_def by transfer auto
    have [simp]: "modgrp_c |f * S_modgrp| = d"
    using greater unfolding a_b_c_d_def by transfer auto
    have [simp]: "modgrp_d |f * S_modgrp| = -c"
    using greater unfolding a_b_c_d_def by transfer (auto split:
if_splits)

    have "dedekind_sum (-c) d = -dedekind_sum c d"
    using <coprime c d> by (simp add: dedekind_sum_negate)
    also have "... = dedekind_sum d c - c / d / 12 - d / c / 12 + 1
/ 4 - 1 / (12 * c * d)"
    using <c > 0> <d > 0> <coprime c d> by (subst dedekind_sum_reciprocity')
simp_all
    finally have *: "dedekind_sum (-c) d = ..." .
    have [simp]: "cnj (cis (pi / 4)) = 1 / cis (pi / 4)"
    by (subst divide_conv_cnj) auto

    have "ε (f * S_modgrp) = cis (pi * ((b - c) / (12 * d) + c / (12*d)
+
d / (12*c) + 1 / (12 * c * d) - dedekind_sum
d c - 1 / 2))"
    unfolding dedekind_eps_def a_b_c_d_def [symmetric] using <c >
0> <d > 0> sing
    by (simp add: * algebra_simps Let_def)
    also have "(b - c) / (12 * d) + c / (12*d) + d / (12*c) + 1 / (12
* c * d) =
(b * c + 1 + d * d) / (12 * c * d)"
    using <c > 0> <d > 0> by (simp add: field_simps)
    also have "b * c + 1 = a * d"
    using det by (simp add: algebra_simps)
    also have "(a * d + d * d) / (12 * c * d) = (a + d) / (12 * c)"
    using <c > 0> <d > 0> by (simp add: field_simps)
    also have "cis (pi * ((a + d) / (12 * c) - dedekind_sum d c - 1
/ 2)) =
cis (-pi / 4) * ε f"
    unfolding dedekind_eps_def a_b_c_d_def [symmetric] using <c >
0> <d > 0> sing f
    by (auto simp: cis_mult algebra_simps diff_divide_distrib add_divide_distrib

Let_def a_b_c_d_def abs_modgrp_altdef)
    finally show ?thesis
    using <d > 0> by (simp add: a_b_c_d_def)

next
case less
have [simp]: "modgrp_a |f * S_modgrp| = -b"
using less unfolding a_b_c_d_def by transfer auto

```

```

have [simp]: "modgrp_b |f * S_modgrp| = a"
  using less unfolding a_b_c_d_def by transfer auto
have [simp]: "modgrp_c |f * S_modgrp| = -d"
  using less unfolding a_b_c_d_def by transfer auto
have [simp]: "modgrp_d |f * S_modgrp| = c"
  using less unfolding a_b_c_d_def by transfer (auto split: if_splits)

have "dedekind_sum c (-d) =
      -dedekind_sum (-d) c - c / d / 12 - d / c / 12 - 1 / 4 -
1 / (12 * c * d)"
  using <c > 0> <d < 0> <coprime c d> by (subst dedekind_sum_reciprocity')
simp_all
  also have "-dedekind_sum (-d) c = dedekind_sum d c"
    using <coprime c d> by (subst dedekind_sum_negate) (auto simp:
Rings.coprime_commute)
  finally have *: "dedekind_sum c (-d) =
      dedekind_sum d c - c / d / 12 - d / c / 12 - 1
/ 4 - 1 / (12 * c * d)" .

have "ε (f * S_modgrp) =
      cis (pi * (c / (12 * d) + d / (12 * c) + 1 / (12 * c * d)
- (c - b) / (12 * d) -
      dedekind_sum d c))"
  unfolding dedekind_eps_def a_b_c_d_def [symmetric] using <d <
0> sing assms
  by (simp add: * algebra_simps)
  also have "c / (12 * d) + d / (12 * c) + 1 / (12 * c * d) - (c -
b) / (12 * d) =
      (d * d + (1 + b * c)) / (12 * c * d)"
    using <c > 0> <d < 0> by (simp add: field_simps)
  also have "1 + b * c = a * d"
    using det by (simp add: algebra_simps)
  also have "(d * d + a * d) / (12 * c * d) = (a + d) / (12 * c)"
    using <c > 0> <d < 0> by (simp add: field_simps)
  also have "cis (pi * ((a + d) / (12 * c) - dedekind_sum d c)) =
cis (pi / 4) * ε f"
  unfolding dedekind_eps_def a_b_c_d_def [symmetric] using <c >
0> <d < 0> sing f
  by (auto simp: cis_mult algebra_simps diff_divide_distrib add_divide_distrib
Let_def
      a_b_c_d_def abs_modgrp_altdef)
  finally show ?thesis
    using <d < 0> by (simp add: a_b_c_d_def)
qed auto
qed

show ?thesis
  using *[of f] *[of "-f"] assms
  by (cases "modgrp_c f ≥ 0") (auto simp: sgn_if is_singular_modgrp_altdef)

```

qed

lemma dedekind_eps_root_of_unity: " $\varepsilon f ^ 24 = 1$ "

proof -

have not_sing: " $\varepsilon f ^ 24 = 1$ " if " \neg is_singular_modgrp f" for f

proof -

have " $\varepsilon f ^ 24 = \text{cis } (2 * (\text{pi} * \text{real_of_int } (\text{modgrp_b } |f|)))$ "

using that by (auto simp: dedekind_eps_def Complex.DeMoivre)

also have " $2 * (\text{pi} * \text{real_of_int } (\text{modgrp_b } |f|)) = 2 * \text{pi} * \text{real_of_int } (\text{modgrp_b } |f|)$ "

by (simp add: mult_ac)

also have " $\text{cis } \dots = 1$ "

by (rule cis_multiple_2pi) auto

finally show ?thesis .

qed

show ?thesis

proof (induction f rule: modgrp_induct_S_shift')

case (S f)

show ?case

proof (cases "modgrp_d f = 0")

case True

hence " $\varepsilon (f * S_{\text{modgrp}}) ^ 24 = \text{cis } (\text{real_of_int } (\text{modgrp_b } |f * S_{\text{modgrp}}|) * (2 * \text{pi}))$ "

by (auto simp: dedekind_eps_def Complex.DeMoivre mult_ac abs_modgrp_altdef)

also have " $\dots = 1$ "

by (subst cis_power_int [symmetric]) auto

finally show ?thesis

by simp

next

case d: False

show ?thesis

proof (cases "is_singular_modgrp f")

case sing: True

define s where " $s = \text{sgn } (\text{modgrp_c } f) * \text{sgn } (\text{modgrp_d } f)$ "

have " $\varepsilon (f * S_{\text{modgrp}}) ^ 24 = \text{cis } (- (\text{pi} * (\text{real_of_int } s * 6)))$ "

using d sing by (simp add: dedekind_eps_S_right field_simps Complex.DeMoivre S_s_def)

also have " $-(\text{pi} * (\text{real_of_int } s * 6)) = 2 * \text{pi} * \text{of_int } (-3 * s)$ "

by simp

also have " $\text{cis } \dots = 1$ "

by (rule cis_multiple_2pi) auto

finally show ?thesis .

next

case False

then obtain n where [simp]: " $\text{abs } f = \text{shift_modgrp } n$ "

using not_singular_modgrpD by blast

```

    have "ε (f * S_modgrp) = ε |f * S_modgrp|"
      by simp
    also have "|f * S_modgrp| = |shift_modgrp n * S_modgrp|"
      by (intro abs_modgrp_congs) auto
    also have "ε ... ^ 24 = cis (pi * (real_of_int n * 2) - pi * 6)"
      by (subst dedekind_eps_abs, subst dedekind_eps_shift_left)
        (simp add: algebra_simps Complex.DeMoivre cis_mult)
    also have "pi * (real_of_int n * 2) - pi * 6 = (real_of_int (n-3)
* (2 * pi))"
      by (simp add: algebra_simps)
    also have "cis ... = 1"
      by (subst cis_power_int [symmetric]) auto
    finally show ?thesis .
  qed
next
case (shift f n)
have "ε (f * shift_modgrp n) ^ 24 = cis (of_int n * (2 * pi))"
  by (simp add: power_mult_distrib shift Complex.DeMoivre mult_ac)
also have "... = 1"
  by (subst cis_power_int [symmetric]) auto
finally show ?case .
qed auto
qed

```

The following theorem is Apostol's Theorem 3.4: the general functional equation for Dedekind's η function.

Our version is actually more general than Apostol's lemma since it also holds for modular groups with $c = 0$ (i.e. shifts, i.e. T^n). We also use a slightly different definition of ε though, namely the one from Wikipedia. This makes the functional equation look a bit nicer than Apostol's version.

```

theorem dedekind_eta_apply_modgrp:
  assumes "Im z > 0"
  shows "η (apply_modgrp f z) = ε f * csqrt (automorphy_factor |f| z)
* η z"
  using assms
proof (induction f arbitrary: z rule: modgrp_induct_S_shift')
  case id
  thus ?case by simp
next
  case (shift f n z)
  have "η (apply_modgrp (f * shift_modgrp n) z) = η (apply_modgrp f (z
+ of_int n))"
    using shift.prem1 by (subst apply_modgrp_mult) auto
  also have "... = ε f * csqrt (automorphy_factor f (z + of_int n)) *
η (z + of_int n)"
    using shift.prem2 by (subst shift.IH) (use shift.hyps in auto)
  also have "η (z + of_int n) = cis (pi * n / 12) * η z"

```

```

    using shift.premis by (subst dedekind_eta_plus_int) auto
    also have "ε f * csqrt (automorphy_factor f (z + of_int n)) * (cis (pi
* n / 12) * η z) =
        ε (f * shift_modgrp n) * csqrt (automorphy_factor (f * shift_modgrp
n) z) * η z"
    by simp
    also have "f * shift_modgrp n = |f * shift_modgrp n|"
    using shift.hyps by (auto simp: modgrp_eq_iff abs_modgrp_altdef)
    finally show ?case using shift.hyps by simp
next
case (S f z)
note [simp del] = div_mult_self3 div_mult_self4 div_mult_self2 div_mult_self1
define a b c d where "a = modgrp_a f" "b = modgrp_b f" "c = modgrp_c
f" "d = modgrp_d f"
have det: "a * d - b * c = 1"
    using modgrp_abcd_det[of f] by (simp add: a_b_c_d_def)
from S.premis have [simp]: "z ≠ 0"
    by auto
show ?case
proof (cases "is_singular_modgrp f")
case False
hence f: "f = shift_modgrp b"
    unfolding a_b_c_d_def using <|f| = f> not_singular_modgrpD[of f]
by auto
have *: "f * S_modgrp = modgrp b (-1) 1 0"
    unfolding f shift_modgrp_code S_modgrp_code times_modgrp_code by
simp
have [simp]: "modgrp_a (f * S_modgrp) = b"
    "modgrp_b (f * S_modgrp) = -1"
    "modgrp_c (f * S_modgrp) = 1"
    "modgrp_d (f * S_modgrp) = 0"
    by (simp_all add: * modgrp_a_code modgrp_b_code modgrp_c_code modgrp_d_code)
have eps: "ε (f * S_modgrp) = cis (pi * (b / 12 - 1 / 4))"
    unfolding f by (subst dedekind_eps_shift_left) (auto simp: cis_mult
ring_distrib)

have "η (apply_modgrp (f * S_modgrp) z) = η (-1 / z + of_int b)"
    using S.premis by (subst apply_modgrp_mult) (auto simp: f algebra_simps)
also have "... = cis (pi * b / 12) * η (-(1 / z))"
    using S.premis by (subst dedekind_eta_plus_int) auto
also have "... = cis (pi * b / 12) * csqrt (- i * z) * η z"
    using S.premis by (subst dedekind_eta_minus_one_over) auto
also have "... = cis (pi / 4) * csqrt (- i * z) * ε (shift_modgrp b
* S_modgrp) * η z"
    using eps by (auto simp: f ring_distrib simp flip: cis_divide)
also have "csqrt (-i * z) = rcis (norm (csqrt (-i * z))) (Arg (csqrt
(-i * z)))"
    by (rule rcis_cmod_Arg [symmetric])
also have "... = rcis (sqrt (cmod z)) (Arg (- (i * z)) / 2)"

```

```

    by (simp add: norm_mult)
  also have "cis (pi / 4) * ... = rcis (sqrt (norm z)) ((Arg (-i*z) +
pi / 2) / 2)"
    by (simp add: rcis_def cis_mult add_divide_distrib algebra_simps)
  also have "Arg (-i*z) + pi / 2 = Arg z"
  proof (rule cis_Arg_unique [symmetric])
    have "cis (Arg (-i * z) + pi / 2) = - (sgn (i * z) * i)"
      by (simp flip: cis_mult add: cis_Arg)
    also have "... = sgn z"
      by (simp add: complex_sgn_def scaleR_conv_of_real field_simps
norm_mult)
    finally show "sgn z = cis (Arg (-i * z) + pi / 2)" ..
  next
    show "-pi < Arg (-i * z) + pi / 2" "Arg (-i * z) + pi / 2 ≤ pi"
      using Arg_Re_pos[of "-i * z"] S.prems by auto
  qed
  also have "rcis (sqrt (norm z)) (Arg z / 2) = rcis (norm (csqrt z))
(Arg (csqrt z))"
    by simp
  also have "... = csqrt z"
    by (rule rcis_cmod_Arg)
  finally show ?thesis
    by (simp add: f_automorphy_factor_altdef abs_modgrp_altdef)
next
  case sing: True
  hence "c > 0"
    unfolding a_b_c_d_def using <abs f = f>
    by (auto simp: is_singular_modgrp_altdef abs_modgrp_altdef split:
if_splits)
  have "Im (1 / z) < 0"
    using S.prems Im_one_over_neg_iff by blast
  have Arg_z: "Arg z ∈ {0<..

```

```

    unfolding ** by simp
  show ?thesis
    using S.prem1 *** **** unfolding apply_modgrp_abs dedekind_eps_abs
    by (auto simp: dedekind_eta_plus_int **)
next
  case greater
  have "modgrp a b c d * S_modgrp = modgrp b (-a) d (-c)"
    unfolding shift_modgrp_code S_modgrp_code times_modgrp_code det
  by simp
  hence *: "f * S_modgrp = modgrp b (-a) d (-c)"
    by (simp add: a_b_c_d_def)
  have [simp]: "modgrp_a (f * S_modgrp) = b" "modgrp_b (f * S_modgrp)
= -a"
    "modgrp_c (f * S_modgrp) = d" "modgrp_d (f * S_modgrp)
= -c"
    unfolding * modgrp_a_code modgrp_b_code modgrp_c_code modgrp_d_code
    using greater det by auto

  have "η (apply_modgrp (f * S_modgrp) z) = η (apply_modgrp f (- (1
/ z)))"
    using S.prem1 by (subst apply_modgrp_mult) auto
  also have "... = ε f * csqrt (automorphy_factor f (- (1 / z))) *
η (- (1 / z))"
    using S.prem1 S.hyps by (subst S.IH) auto
  also have "automorphy_factor f (- (1 / z)) = d - c / z"
    unfolding automorphy_factor_altdef by (simp add: a_b_c_d_def)
  also have "η (- (1 / z)) = csqrt (-i * z) * η z"
    using S.prem1 by (subst dedekind_eta_minus_one_over) auto
  also have "ε f * csqrt (d - c / z) * (csqrt (-i * z) * η z) =
(csqrt (d - c / z) * csqrt (-i * z)) * ε f * η z"
    by (simp add: mult_ac)
  also have "csqrt (d - c / z) * csqrt (-i * z) = csqrt ((d - c / z)
* (-i * z))"
  proof (rule csqrt_mult [symmetric])
    have "Im (of_int d - of_int c / z) = -of_int c * Im (1 / z)"
      by (simp add: Im_divide)
    hence Im: "Im (of_int d - of_int c / z) > 0"
      using <Im (1 / z) < 0> <c > 0> by (auto simp: mult_less_0_iff)
    hence Arg_pos: "Arg (of_int d - of_int c / z) > 0"
      using Arg_pos_iff by blast

  have "Arg (of_int d - of_int c / z) + Arg z ≤ 3 / 2 * pi"
  proof (cases "Re z ≥ 0")
    case True
    have "Arg (of_int d - of_int c / z) ≤ pi"
      by (rule Arg_le_pi)
    moreover have "Arg z ≤ pi / 2"
      using Arg_Re_nonneg[of z] True by auto
    ultimately show ?thesis by simp

```

```

next
  case False
  have "Re (of_int d - of_int c / z) = of_int d - of_int c * Re
z / norm z ^ 2"
    by (simp add: Re_divide norm_complex_def)
  also have "... ≥ 0 - 0"
    using <d > 0> <c > 0> False
    by (intro diff_mono divide_nonpos_pos mult_nonneg_nonpos)
auto
  finally have "Arg (of_int d - of_int c / z) ≤ pi / 2"
    using Arg_Re_nonneg[of "of_int d - of_int c / z"] by simp
  moreover have "Arg z ≤ pi"
    by (rule Arg_le_pi)
  ultimately show ?thesis by simp
qed
moreover have "Arg (of_int d - of_int c / z) + Arg z > 0 + 0"
  using Arg_z by (intro add_strict_mono Arg_pos) auto
  ultimately show "Arg (of_int d - of_int c / z) + Arg (- i * z)
∈ {-pi<..pi}"
    using Arg_z' by auto
qed
also have "(d - c / z) * (-i * z) = (-i) * (d * z - c)"
  using S.prem by (auto simp: field_simps)
also have "csqrt ... = csqrt (-i) * csqrt (d * z - c)"
proof (intro csqrt_mult)
  have "Arg (of_int d * z - of_int c) > 0"
    using <d > 0> S.prem by (subst Arg_pos_iff) auto
  moreover have "Arg (of_int d * z - of_int c) ≤ pi"
    by (rule Arg_le_pi)
  ultimately show "Arg (-i) + Arg (of_int d * z - of_int c) ∈ {-pi<..pi}"
    by auto
qed
also have "csqrt (-i) = cis (-pi / 4)"
  by (simp add: csqrt_exp_Ln cis_conv_exp)
also have "cis (-pi / 4) * csqrt (d * z - c) * ε f * η z =
ε (f * S_modgrp) * csqrt (d * z - c) * η z"
  using <c > 0> <d > 0> sing by (subst dedekind_eps_S_right) (auto
simp: a_b_c_d_def)
  also have "... = ε (f * S_modgrp) * csqrt (automorphy_factor (f
* S_modgrp) z) * η z"
    unfolding automorphy_factor_altdef by (simp add: a_b_c_d_def)
  also have "f * S_modgrp = |f * S_modgrp|"
    using <c > 0> <d > 0> by (auto simp: abs_modgrp_altdef a_b_c_d_def)
  finally show ?thesis by simp
next
case less
have "modgrp a b c d * S_modgrp = modgrp b (-a) d (-c)"
  unfolding shift_modgrp_code S_modgrp_code times_modgrp_code det
by simp

```

```

hence *: "f * S_modgrp = modgrp b (-a) d (-c)"
  by (simp add: a_b_c_d_def)
have [simp]: "modgrp_a (f * S_modgrp) = b" "modgrp_b (f * S_modgrp)
= -a"
          "modgrp_c (f * S_modgrp) = d" "modgrp_d (f * S_modgrp)
= -c"
  unfolding * modgrp_a_code modgrp_b_code modgrp_c_code modgrp_d_code
  using less det <c > 0> by auto

have "η (apply_modgrp (f * S_modgrp) z) = η (apply_modgrp f (- (1
/ z)))"
  using S.prem1 by (subst apply_modgrp_mult) auto
  also have "... = ε f * csqrt (automorphy_factor f (- (1 / z))) *
η (- (1 / z))"
  using S.prem2 S.hyps by (subst S.IH) auto
  also have "automorphy_factor f (- (1 / z)) = d - c / z"
  unfolding automorphy_factor_altdef by (simp add: a_b_c_d_def)
  also have "η (- (1 / z)) = csqrt (-i * z) * η z"
  using S.prem3 by (subst dedekind_eta_minus_one_over) auto
  also have "ε f * csqrt (d - c / z) * (csqrt (-i * z) * η z) =
(csqrt (d - c / z) * csqrt (-i * z)) * ε f * η z"
  by (simp add: mult_ac)
  also have "csqrt (-i * z) = csqrt (i * -z)"
  by simp
  also have "... = csqrt i * csqrt (-z)"
  proof (rule csqrt_mult)
    show "Arg i + Arg (- z) ∈ {- pi<..pi}"
      using Arg_z by auto
  qed
  also have "csqrt (d - c / z) * ... = csqrt i * (csqrt (d - c / z)
* csqrt (-z))"
  by (simp add: mult_ac)
  also have "csqrt (d - c / z) * csqrt (-z) = csqrt ((d - c / z) *
(-z))"
  proof (rule csqrt_mult [symmetric])
    have "Im (of_int d - of_int c / z) = of_int c * Im z / norm z
^ 2"
      by (simp add: Im_divide norm_complex_def)
    also have "... > 0"
      using S.prem4 <c > 0> by (intro mult_pos_pos divide_pos_pos)
  auto
  finally have "Arg (of_int d - of_int c / z) ∈ {0<..pi}"
    using Arg_lt_pi[of "of_int d - of_int c / z"] by auto
  thus "Arg (of_int d - of_int c / z) + Arg (- z) ∈ {- pi<..pi}"
    using Arg_z by auto
  qed
  also have "(d - c / z) * (-z) = c - d * z"
  using S.prem5 by (simp add: field_simps)
  also have "csqrt i = cis (pi / 4)"

```

```

    by (simp add: csqrt_exp_Ln complex_eq_iff cos_45 sin_45 field_simps)
  also have "cis (pi / 4) * csqrt (c - d * z) * ε f * η z =
    ε (f * S_modgrp) * csqrt (c - d * z) * η z"
    using <c > 0> <d < 0> sing by (subst dedekind_eps_S_right) (auto
simp: a_b_c_d_def)
  also have "... = ε (f * S_modgrp) * csqrt (automorphy_factor (-f
* S_modgrp) z) * η z"
    unfolding automorphy_factor_altdef <d < 0> <c > 0>
    by (auto simp add: a_b_c_d_def abs_modgrp_altdef)
  also have "-f * S_modgrp = |f * S_modgrp|"
    using <c > 0> <d < 0> by (auto simp: abs_modgrp_altdef a_b_c_d_def)
  finally show ?thesis .
qed
qed
qed simp_all

```

```

no_notation dedekind_eta ("η")
no_notation dedekind_eps ("ε")

```

end

13.5 The transformation law for G_2

```

theory Eisenstein_G2
  imports Dedekind_Eta
begin

```

In his book, Apostol derives the inversion law for G_2 in the exercises to Chapter 3 and remarks that it leads to a proof of the inversion law for η . Since we already have a nice and short proof for the inversion law for η , we instead go the other direction. We differentiate the inversion law for η and easily obtain the corresponding law for G_2

```

theorem Eisenstein_G2_minus_one_over:
  assumes t: "Im t > 0"
  shows "Eisenstein_G 2 (-(1/t)) = t^2 * Eisenstein_G 2 t - 2 * pi *
i * t"
proof -
  write dedekind_eta ("η")
  from assms have "t ≠ 0"
  by auto
  note [derivative_intros] = has_field_derivative_dedekind_eta
  have "deriv (λt. η (-(1/t))) t =
    i * Eisenstein_G 2 (-(1 / t)) * η (-(1 / t)) / (4 * pi * t^2)"
  by (rule DERIV_imp_deriv) (use t in <auto intro!: derivative_eq_intros
simp: power2_eq_square>)
  also have "η (-(1 / t)) = csqrt (- (i * t)) * η t"
  by (subst dedekind_eta_minus_one_over) (use t in auto)
  finally have "i * η t * csqrt (-i*t) / (4 * pi * t^2) * Eisenstein_G 2
(-(1 / t)) =

```

```

      deriv (λt. η (-(1/t))) t"
    by simp

  also have "deriv (λt. η (-(1/t))) t = deriv (λt. csqrt (-(i*t)) * η
t) t"
  proof (intro deriv_cong_ev refl)
    have "eventually (λz. z ∈ {z. Im z > 0}) (nhds t)"
      by (rule eventually_nhds_in_open) (use t in <auto simp: open_halfspace_Im_gt>)
    thus "∀F x in nhds t. η (-(1/x)) = csqrt (-(i*x)) * η x"
      by eventually_elim (subst dedekind_eta_minus_one_over, auto)
  qed
  also have "deriv (λt. csqrt (-(i*t)) * η t) t =
      i * η t * (Eisenstein_G 2 t * csqrt (-(i*t)) / (4 * pi) -
1 / (2 * csqrt (-(i*t))))"
    by (rule DERIV_imp_deriv)
      (use t in <auto intro!: derivative_eq_intros simp: complex_nonpos_Reals_iff
field_simps>)
  also have "1 / (2 * csqrt (-(i*t))) = csqrt (-(i*t)) / (2 * (-i * t))"
  proof -
    have *: "-i * t = csqrt (-(i * t)) ^ 2"
      by simp
    show ?thesis
      by (subst (3) *, unfold power2_eq_square) (use <t ≠ 0> in <auto
simp: field_simps>)
  qed
  also have "i * η t * (Eisenstein_G 2 t * csqrt (-(i*t)) / (4 * pi) - ...)
=
      i * η t * csqrt (-(i * t)) / (4 * pi) * (Eisenstein_G 2 t +
2 * pi / (i*t))"
    by (simp add: field_simps)
  also have "... = i * η t * csqrt (-(i * t)) / (4 * pi * t2) * (t2 * Eisenstein_G
2 t - 2 * i * pi * t)"
    using <t ≠ 0> by (simp add: field_simps power2_eq_square)
  finally show "Eisenstein_G 2 (-(1/t)) = t2 * Eisenstein_G 2 t - 2
* pi * i * t"
    by (subst (asm) mult_cancel_left) (use t in auto)
  qed

lemma Eisenstein_E2_minus_one_over:
  assumes t: "Im t > 0"
  shows "Eisenstein_E 2 (-(1/t)) = t2 * Eisenstein_E 2 t - 6 * i / pi
* t"
  using assms
  by (simp add: Eisenstein_E_def Eisenstein_G2_minus_one_over[OF t]
zeta_2 power2_eq_square field_simps)

```

In a similar fashion to the η function, we can prove the general modular transformation law for G_2 :

theorem *Eisenstein_G2_apply_modgrp*:

```

assumes "Im z > 0"
shows "Eisenstein_G 2 (apply_modgrp f z) =
      automorphy_factor f z ^ 2 * Eisenstein_G 2 z -
      2 * i * pi * modgrp_c f * automorphy_factor f z"
using assms
proof (induction f arbitrary: z rule: modgrp_induct_S_shift')
  case id
  thus ?case by simp
next
  case (shift f n z)
  have "Eisenstein_G 2 (apply_modgrp (f * shift_modgrp n) z) =
        Eisenstein_G 2 (apply_modgrp f (z + of_int n))"
    using shift.prem1 by (subst apply_modgrp_mult) auto
  also have "... = (automorphy_factor f (z + of_int n))^2 * Eisenstein_G
2 (z + of_int n) -
        2 * i * of_real pi * of_int (modgrp_c f) * automorphy_factor
f (z + of_int n)"
    using shift.prem2 by (subst shift.IH) auto
  also have "Eisenstein_G 2 (z + of_int n) = Eisenstein_G 2 z"
    by (rule Eisenstein_G_plus_int)
  finally show ?case
    by simp
next
  case (S f z)
  note [simp del] = div_mult_self3 div_mult_self4 div_mult_self2 div_mult_self1
  define a b c d where "a = modgrp_a f" "b = modgrp_b f" "c = modgrp_c
f" "d = modgrp_d f"
  have det: "a * d - b * c = 1"
    using modgrp_abcd_det[of f] by (simp add: a_b_c_d_def)

  from S.prem1 have [simp]: "z ≠ 0"
    by auto
  show ?case
  proof (cases "is_singular_modgrp f")
    case False
    hence f: "f = shift_modgrp b"
      unfolding a_b_c_d_def using S.hyps not_singular_modgrpD[of f] by
auto
    have *: "f * S_modgrp = modgrp b (-1) 1 0"
      unfolding f shift_modgrp_code S_modgrp_code times_modgrp_code by
simp
    have [simp]: "modgrp_a (f * S_modgrp) = b"
      "modgrp_b (f * S_modgrp) = -1"
      "modgrp_c (f * S_modgrp) = 1"
      "modgrp_d (f * S_modgrp) = 0"
      by (simp_all add: * modgrp_a_code modgrp_b_code modgrp_c_code modgrp_d_code)

    have "Eisenstein_G 2 (apply_modgrp (f * S_modgrp) z) = Eisenstein_G
2 (-1 / z + of_int b)"

```

```

    using S.premys by (subst apply_modgrp_mult) (auto simp: f algebra_simps)
  also have "... = Eisenstein_G 2  $-(1/z)$ )"
    by (subst Eisenstein_G_plus_int) auto
  also have "... =  $z^2 * \text{Eisenstein\_G } 2 z - 2 * \pi * i * z$ "
    by (subst Eisenstein_G2_minus_one_over) (use S.premys in auto)
  also have "... = (automorphy_factor (f * S_modgrp) z)2 * Eisenstein_G
2 z -
      2 * i * pi * complex_of_int (modgrp_c (f * S_modgrp))
* automorphy_factor (f * S_modgrp) z"
    by (simp add: automorphy_factor_altdef f)
  finally show ?thesis .
next
  case sing: True
  hence "c > 0" using S.hyps
    unfolding a_b_c_d_def
    by (auto simp: is_singular_modgrp_altdef abs_modgrp_altdef split:
if_splits)
  have "Im (1 / z) < 0"
    using S.premys Im_one_over_neg_iff by blast
  have Arg_z: "Arg z ∈ {0<.. $\pi$ }"
    using S.premys by (simp add: Arg_lt_pi)
  have Arg_z': "Arg (-i * z) =  $-\pi/2 + \text{Arg } z$ "
    using Arg_z by (subst Arg_times) auto
  have [simp]: "Arg (-z) = Arg z -  $\pi$ "
    using Arg_z by (subst Arg_minus) auto

  have "modgrp a b c d * S_modgrp = modgrp b (-a) d (-c)"
    unfolding shift_modgrp_code S_modgrp_code times_modgrp_code det
by simp
  hence *: "f * S_modgrp = modgrp b (-a) d (-c)"
    by (simp add: a_b_c_d_def)
  have [simp]: "modgrp_a (f * S_modgrp) = b" "modgrp_b (f * S_modgrp)
= -a"
    "modgrp_c (f * S_modgrp) = d" "modgrp_d (f * S_modgrp)
= -c"
  unfolding * modgrp_a_code modgrp_b_code modgrp_c_code modgrp_d_code
    using det by auto
  define F where "F = automorphy_factor (f * S_modgrp) z"

  have "Eisenstein_G 2 (apply_modgrp (f * S_modgrp) z) =
      Eisenstein_G 2 (apply_modgrp f (- (1 / z)))"
    using S.premys by (subst apply_modgrp_mult) auto
  also have "... = (automorphy_factor f (- (1 / z)))2 * Eisenstein_G
2 (- (1 / z)) -
      2 * i * complex_of_real pi * c * automorphy_factor
f (- (1 / z))"
    using S.premys by (subst S.IH) (auto simp flip: a_b_c_d_def)
  also have "automorphy_factor f (- (1 / z)) = F / z"
    unfolding F_def automorphy_factor_altdef by (simp add: a_b_c_d_def

```

```

field_simps)
  also have "Eisenstein_G 2 (-(1 / z)) = z^2 * Eisenstein_G 2 z - 2 *
pi * i * z"
    by (subst Eisenstein_G2_minus_one_over) (use S.prem in auto)
  also have "(F / z)^2 * (z^2 * Eisenstein_G 2 z - of_real (2 * pi) *
i * z) =
      F ^ 2 * Eisenstein_G 2 z - 2 * pi * i * F ^ 2 / z"
    using S.prem by (simp add: field_simps power2_eq_square automorphy_factor_def
F_def)
  also have "F^2 * Eisenstein_G 2 z - of_real (2 * pi) * i * F^2 / z -
      2 * i * of_real pi * of_int c * (F / z) =
      F^2 * Eisenstein_G 2 z - 2 * pi * i * ((F^2 + of_int c *
F) / z)"
    by (simp add: field_simps)
  also have "(F^2 + of_int c * F) / z = of_int (modgrp_c (f * S_modgrp))
* F"
    by (simp add: F_def automorphy_factor_altdef field_simps power2_eq_square
      flip: modgrp_c_def a_b_c_d_def)
  finally show ?thesis
    unfolding F_def by simp
qed
qed simp_all

```

```

lemma Eisenstein_E2_apply_modgrp:
  assumes "Im z > 0"
  shows "Eisenstein_E 2 (apply_modgrp f z) =
      automorphy_factor f z ^ 2 * Eisenstein_E 2 z - 6 * i / pi
* modgrp_c f * automorphy_factor f z"
    unfolding Eisenstein_E_def
  by (simp add: Eisenstein_G2_apply_modgrp[OF assms] power2_eq_square
zeta_2 field_simps)

```

We can now also easily derive the values $G_2(i) = \pi$ and $G_2(\rho) = \frac{2\pi}{\sqrt{3}}$ using the same technique we used earlier for general G_k with $k \geq 3$.

```

lemma Eisenstein_G2_ii: "Eisenstein_G 2 i = of_real pi"
  using Eisenstein_G2_minus_one_over[of "i"] by simp

```

```

lemma Eisenstein_E2_ii: "Eisenstein_E 2 i = 3 / of_real pi"
  by (simp add: Eisenstein_G2_ii Eisenstein_E_def zeta_2 power2_eq_square)

```

```

lemma Eisenstein_G2_rho: "Eisenstein_G 2 modfun_rho = of_real (2 / sqrt
3 * pi)"

```

proof -

```

  have "Eisenstein_G 2 (- (1 / modfun_rho)) = modfun_rho^2 * Eisenstein_G
2 modfun_rho -
      complex_of_real (2 * pi) * i * modfun_rho"
    using Eisenstein_G2_minus_one_over[of modfun_rho] by simp
  also have "- (1 / modfun_rho) = modfun_rho + 1"

```

```

    by (auto simp: modfun_rho_altdef field_simps simp flip: of_real_mult)
    also have "modfun_rho ^ 2 = -(modfun_rho + 1)"
    by (auto simp: modfun_rho_altdef field_simps power2_eq_square simp
flip: of_real_mult)
    also have "Eisenstein_G 2 (modfun_rho + 1) = Eisenstein_G 2 modfun_rho"
    using Eisenstein_G_plus_int[of 2 "modfun_rho" 1] by simp
    finally have *: "(modfun_rho + 2) * Eisenstein_G 2 modfun_rho = -2 *
i * pi * modfun_rho"
    unfolding of_real_mult of_real_numeral by Groebner_Basis.algebra

    have "modfun_rho + 2 ≠ 0"
    by (auto simp: modfun_rho_altdef complex_eq_iff)
    hence "Eisenstein_G 2 modfun_rho = (-2 * i * pi * modfun_rho) / (modfun_rho
+ 2)"
    by (subst * [symmetric]) auto
    also have "(-2 * i * pi * modfun_rho) / (modfun_rho + 2) = of_real (2
/ sqrt 3 * pi)"
    by (auto simp: complex_eq_iff modfun_rho_altdef Re_divide Im_divide
field_simps)
    finally show ?thesis .
qed

lemma Eisenstein_E2_rho: "Eisenstein_E 2 modfun_rho = of_real (2 * sqrt
3 / pi)"
  by (simp add: Eisenstein_G2_rho Eisenstein_E_def zeta_2 power2_eq_square
field_simps
      flip: of_real_mult[of "sqrt 3" "sqrt 3"])

end

```

References

- [1] T. M. Apostol. *Modular Functions and Dirichlet Series in Number Theory*. Graduate Texts in Mathematics. Springer New York, 1990.
- [2] S. Lang. *Elliptic Functions*. Graduate Texts in Mathematics. Springer New York, 1973.