

Interactive Theorem Proving Assignment 2
Martin Skilleter (u6679334)
8 April 2019

I collaborated with the following people while completing this assignment: Isabel Longbottom (u6702144) and Aaron Bryce (u6677442).

Question 1

```
import data.real.basic algebra.field_power
import algebra.pi_instances
import algebra.module
import ring_theory.algebra

-- Question 2a
structure quaternion : Type :=
  (re : ℝ) (i : ℝ) (j : ℝ) (k : ℝ)

notation `H` := quaternion

namespace quaternion

@[simp] theorem eta : ∀ p : H, quaternion.mk p.re p.i p.j p.k = p
| ⟨a, b, c, d⟩ := rfl

@[extensionality] theorem ext : ∀ {p q : H}, p.re = q.re → p.i = q.i →
  p.j = q.j → p.k = q.k → p = q
| ⟨pr, pi, pj, pk⟩ ⟨_, _, _, _⟩ rfl rfl rfl rfl := rfl

theorem ext_iff {p q : H} : p = q ↔ p.re = q.re ∧ p.i = q.i ∧ p.j = q.j ∧
  p.k = q.k :=
  ⟨λ H, by simp [H], by {rintros ⟨a, ⟨b, ⟨c, d⟩⟩, apply ext; assumption}⟩

def of_real (r : ℝ) : H := ⟨r, 0, 0, 0⟩
instance : has_coe ℝ H := ⟨of_real⟩
@[simp] lemma of_real_eq_coe (r : ℝ) : of_real r = r := rfl

@[simp] lemma of_real_re (r : ℝ) : (r : H).re = r := rfl
@[simp] lemma of_real_i (r : ℝ) : (r : H).i = 0 := rfl
@[simp] lemma of_real_j (r : ℝ) : (r : H).j = 0 := rfl
@[simp] lemma of_real_k (r : ℝ) : (r : H).k = 0 := rfl

@[simp] theorem of_real_inj {z w : ℝ} : (z : H) = w ↔ z = w :=
  ⟨congr_arg re, congr_arg _⟩

instance : has_zero H := ⟨(0 : ℝ)⟩
instance : inhabited H := ⟨0⟩
```

```

@[simp] lemma zero_re : (0 : ℍ).re = 0 := rfl
@[simp] lemma zero_i : (0 : ℍ).i = 0 := rfl
@[simp] lemma zero_j : (0 : ℍ).j = 0 := rfl
@[simp] lemma zero_k : (0 : ℍ).k = 0 := rfl
@[simp] lemma of_real_zero : ((0 : ℝ) : ℍ) = 0 := rfl

@[simp] theorem of_real_eq_zero {z : ℝ} : (z : ℍ) = 0 ↔ z = 0 :=
of_real_inj
@[simp] theorem of_real_ne_zero {z : ℝ} : (z : ℍ) ≠ 0 ↔ z ≠ 0 := not_congr
of_real_eq_zero

instance : has_one ℍ := ⟨(1 : ℝ)⟩

@[simp] lemma one_re : (1 : ℍ).re = 1 := rfl
@[simp] lemma one_i : (1 : ℍ).i = 0 := rfl
@[simp] lemma one_j : (1 : ℍ).j = 0 := rfl
@[simp] lemma one_k : (1 : ℍ).k = 0 := rfl
@[simp] lemma of_real_one : ((1 : ℝ) : ℍ) = 1 := rfl

def I : ℍ := ⟨0, 1, 0, 0⟩
def J : ℍ := ⟨0, 0, 1, 0⟩
def K : ℍ := ⟨0, 0, 0, 1⟩

@[simp] lemma I_re : I.re = 0 := rfl
@[simp] lemma I_i : I.i = 1 := rfl
@[simp] lemma I_j : I.j = 0 := rfl
@[simp] lemma I_k : I.k = 0 := rfl

@[simp] lemma J_re : J.re = 0 := rfl
@[simp] lemma J_i : J.i = 0 := rfl
@[simp] lemma J_j : J.j = 1 := rfl
@[simp] lemma J_k : J.k = 0 := rfl

@[simp] lemma K_re : K.re = 0 := rfl
@[simp] lemma K_i : K.i = 0 := rfl
@[simp] lemma K_j : K.j = 0 := rfl
@[simp] lemma K_k : K.k = 1 := rfl

instance : has_add ℍ := ⟨λ z w, ⟨z.re + w.re, z.i + w.i, z.j + w.j, z.k +
w.k⟩⟩

```

```

@[simp] lemma add_re (z w : H) : (z + w).re = z.re + w.re := rfl
@[simp] lemma add_i (z w : H) : (z + w).i = z.i + w.i := rfl
@[simp] lemma add_j (z w : H) : (z + w).j = z.j + w.j := rfl
@[simp] lemma add_k (z w : H) : (z + w).k = z.k + w.k := rfl
@[simp] lemma of_real_add (r s : ℝ) : ((r + s : ℝ) : H) = r + s :=
ext_iff.2 $ by simp

```

```

@[simp] lemma of_real_bit0 (r : ℝ) : ((bit0 r : ℝ) : H) = bit0 r :=
ext_iff.2 $ by simp [bit0]
@[simp] lemma of_real_bit1 (r : ℝ) : ((bit1 r : ℝ) : H) = bit1 r :=
ext_iff.2 $ by simp [bit1]

```

```

instance : has_neg H := ⟨λ z, ⟨-z.re, -z.i, -z.j, -z.k⟩⟩

```

```

@[simp] lemma neg_re (z : H) : (-z).re = -z.re := rfl
@[simp] lemma neg_i (z : H) : (-z).i = -z.i := rfl
@[simp] lemma neg_j (z : H) : (-z).j = -z.j := rfl
@[simp] lemma neg_k (z : H) : (-z).k = -z.k := rfl
@[simp] lemma of_real_neg (r : ℝ) : ((-r : ℝ) : H) = -r := ext_iff.2 $ by
simp

```

```

instance : has_mul H := ⟨λ p q, ⟨p.re*q.re - p.i*q.i - p.j*q.j - p.k*q.k,
p.re*q.i + p.i*q.re + p.j*q.k - p.k*q.j,
p.re*q.j - p.i*q.k + p.j*q.re + p.k*q.i,
p.re*q.k + p.i*q.j - p.j*q.i + p.k*q.re⟩⟩

```

```

@[simp] lemma mul_re (p q : H) : (p*q).re = p.re*q.re - p.i*q.i - p.j*q.j -
p.k*q.k := rfl
@[simp] lemma mul_i (p q : H) : (p*q).i = p.re*q.i + p.i*q.re + p.j*q.k -
p.k*q.j := rfl
@[simp] lemma mul_j (p q : H) : (p*q).j = p.re*q.j - p.i*q.k + p.j*q.re +
p.k*q.i := rfl
@[simp] lemma mul_k (p q : H) : (p*q).k = p.re*q.k + p.i*q.j - p.j*q.i +
p.k*q.re := rfl
@[simp] lemma of_real_mul (r s : ℝ) : ((r * s : ℝ) : H) = r * s :=
ext_iff.2 $ by simp
@[simp] lemma mul_comm_re (r : ℝ) (p : H) : (r : H) * p = p * r :=
by ext; {dsimp, simp [mul_comm]}

```

```

@[simp] lemma I_mul_I : I * I = -1 := ext_iff.2 $ by simp
@[simp] lemma J_mul_J : J * J = -1 := ext_iff.2 $ by simp
@[simp] lemma K_mul_K : K * K = -1 := ext_iff.2 $ by simp

```

```

lemma one_ne_zero : (1 :  $\mathbb{H}$ )  $\neq$  0 := by simp [ext_iff]
lemma I_ne_zero : (I :  $\mathbb{H}$ )  $\neq$  0 := mt (congr_arg i) zero_ne_one.symm
lemma J_ne_zero : (J :  $\mathbb{H}$ )  $\neq$  0 := mt (congr_arg j) zero_ne_one.symm
lemma K_ne_zero : (K :  $\mathbb{H}$ )  $\neq$  0 := mt (congr_arg k) zero_ne_one.symm

lemma mk_eq_add_mul_I_add_mul_J_add_mul_K (a b c d :  $\mathbb{R}$ ) :
quaternion.mk a b c d = a + b * I + c * J + d * K :=
ext_iff.2 $ by simp

@[simp] lemma re_add_i_add_j_add_k (z :  $\mathbb{H}$ ) : (z.re :  $\mathbb{H}$ ) + z.i * I + z.j * J
+ z.k * K = z :=
ext_iff.2 $ by simp

def real_prod_equiv :  $\mathbb{H} = (\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}) :=
{ to_fun :=  $\lambda$  z, <z.re, z.i, z.j, z.k>,
  inv_fun :=  $\lambda$  p, <p.1, p.2.1, p.2.2.1, p.2.2.2>,
  left_inv :=  $\lambda$  <a, b, c, d>, rfl,
  right_inv :=  $\lambda$  <a, b, c, d>, rfl }

@[simp] theorem real_prod_equiv_apply (z :  $\mathbb{H}$ ) : real_prod_equiv z = (z.re,
z.i, z.j, z.k) := rfl

-- Question 2d
def conj (z :  $\mathbb{H}$ ) :  $\mathbb{H}$  := <z.re, -z.i, -z.j, -z.k>

@[simp] lemma conj_re (z :  $\mathbb{H}$ ) : (conj z).re = z.re := rfl
@[simp] lemma conj_i (z :  $\mathbb{H}$ ) : (conj z).i = -z.i := rfl
@[simp] lemma conj_j (z :  $\mathbb{H}$ ) : (conj z).j = -z.j := rfl
@[simp] lemma conj_k (z :  $\mathbb{H}$ ) : (conj z).k = -z.k := rfl
@[simp] lemma conj_of_real (r :  $\mathbb{R}$ ) : conj r = r := ext_iff.2 $ by simp
[conj]

@[simp] lemma conj_zero : conj 0 = 0 := ext_iff.2 $ by simp [conj]
@[simp] lemma conj_one : conj 1 = 1 := ext_iff.2 $ by simp
@[simp] lemma conj_I : conj I = -I := ext_iff.2 $ by simp
@[simp] lemma conj_neg_I : conj (-I) = I := ext_iff.2 $ by simp
@[simp] lemma conj_J : conj J = -J := ext_iff.2 $ by simp
@[simp] lemma conj_neg_J : conj (-J) = J := ext_iff.2 $ by simp
@[simp] lemma conj_K : conj K = -K := ext_iff.2 $ by simp
@[simp] lemma conj_neg_K : conj (-K) = K := ext_iff.2 $ by simp$ 
```

```

@[simp] lemma conj_add (z w : ℍ) : conj (z + w) = conj z + conj w :=
ext_iff.2 $ by simp

@[simp] lemma conj_neg (z : ℍ) : conj (-z) = -conj z := rfl

@[simp] lemma conj_mul (p q : ℍ) : conj (p * q) = conj q * conj p := by
simp [ext_iff, mul_comm, add_comm]

@[simp] lemma conj_conj (z : ℍ) : conj (conj z) = z :=
ext_iff.2 $ by simp

lemma conj_bijective : function.bijective conj :=
⟨function.injective_of_has_left_inverse ⟨conj, conj_conj⟩,
function.surjective_of_has_right_inverse ⟨conj, conj_conj⟩⟩

lemma conj_inj {z w : ℍ} : conj z = conj w ↔ z = w :=
conj_bijective.1.eq_iff

@[simp] lemma conj_eq_zero {z : ℍ} : conj z = 0 ↔ z = 0 :=
by simp using @conj_inj z 0

@[simp] lemma eq_conj_iff_real (z : ℍ) : conj z = z ↔ ∃ r : ℝ, z = r :=
begin
  split,
  intros h,
  use z.re,
  simp [ext_iff] at h,
  rcases h with ⟨hi, ⟨hj, hk⟩⟩,
  ext,
  refl,
  repeat {apply eq_zero_of_neg_eq, assumption},
  intros h,
  cases h with r h,
  simp only [ext_iff] at h,
  rcases h with ⟨hre, ⟨hi, ⟨hj, hk⟩⟩⟩,
  ext,
  refl,
  repeat {simp, try {rw hi}, try {rw hj}, try {rw hk}, simp},
end

```

```
def norm_sq (z : ℍ) : ℝ := z.re * z.re + z.i * z.i + z.j * z.j + z.k * z.k
```

```
@[simp] lemma norm_sq_of_real (r : ℝ) : norm_sq r = r * r :=  
by simp [norm_sq]
```

```
@[simp] lemma norm_sq_zero : norm_sq 0 = 0 := by simp [norm_sq]  
@[simp] lemma norm_sq_one : norm_sq 1 = 1 := by simp [norm_sq]  
@[simp] lemma norm_sq_I : norm_sq I = 1 := by simp [norm_sq]  
@[simp] lemma norm_sq_J : norm_sq J = 1 := by simp [norm_sq]  
@[simp] lemma norm_sq_K : norm_sq K = 1 := by simp [norm_sq]
```

```
lemma norm_sq_nonneg (z : ℍ) : 0 ≤ norm_sq z :=  
add_nonneg ( add_nonneg ( add_nonneg (mul_self_nonneg _) (mul_self_nonneg  
_)) (mul_self_nonneg _) ) (mul_self_nonneg _)
```

```
lemma zero_of_sum_squares_zero {a b c d : ℝ} : a*a + b*b + c*c + d*d = 0 →  
a = 0 ∧ b = 0 ∧ c = 0 ∧ d = 0 :=
```

```
begin  
  intros h,  
  have ha : a*a ≥ 0, by apply mul_self_nonneg,  
  have hb : b*b ≥ 0, by apply mul_self_nonneg,  
  have hc : c*c ≥ 0, by apply mul_self_nonneg,  
  have hd : d*d ≥ 0, by apply mul_self_nonneg,  
  have hcd : c*c + d*d ≥ 0, by exact add_nonneg hc hd,  
  have hbcd : b*b + (c*c + d*d) ≥ 0, by exact add_nonneg hb hcd,  
  rw ←add_assoc at hbcd,  
  have w₁ := (add_eq_zero_iff_eq_zero_and_eq_zero_of_nonneg_of_nonneg ha  
hbcd).1,  
  have h₁ : a*a + b*b + c*c + d*d = a*a + (b*b + c*c + d*d), by ring,  
  rw ←h₁ at w₁,  
  have w₂ := w₁ h,  
  cases w₂,  
  have h₂ : b*b + c*c + d*d = b*b + (c*c + d*d), by ring,  
  have w₃ := (add_eq_zero_iff_eq_zero_and_eq_zero_of_nonneg_of_nonneg hb  
hcd).1,  
  rw ←h₂ at w₃,  
  have w₄ := w₃ w₂_right,  
  cases w₄,  
  have w₅ := (add_eq_zero_iff_eq_zero_and_eq_zero_of_nonneg_of_nonneg hc  
hd).1 w₄_right,  
  cases w₅,  
  repeat {split},
```

```

    exact eq_zero_of_mul_self_eq_zero w2_left,
    exact eq_zero_of_mul_self_eq_zero w4_left,
    exact eq_zero_of_mul_self_eq_zero w5_left,
    exact eq_zero_of_mul_self_eq_zero w5_right,
end

```

```

@[simp] lemma norm_sq_eq_zero {z : ℍ} : norm_sq z = 0 ↔ z = 0 :=
begin
  split,
  intros h,
  dsimp [norm_sq] at h,
  have w : z.re = 0 ∧ z.i = 0 ∧ z.j = 0 ∧ z.k = 0, by exact
zero_of_sum_squares_zero h,
  rcases w with ⟨wre, ⟨wi, ⟨wj, wk⟩⟩⟩,
  ext,
  repeat {assumption},
  intros h,
  dsimp [norm_sq],
  simp only [ext_iff] at h,
  rcases h with ⟨hre, ⟨hi, ⟨hj, hk⟩⟩⟩,
  rw [hre, hi, hj, hk],
  simp,
end

```

```

@[simp] lemma norm_sq_pos {z : ℍ} : 0 < norm_sq z ↔ z ≠ 0 :=
by rw [lt_iff_le_and_ne, ne, eq_comm]; simp [norm_sq_nonneg]

```

```

@[simp] lemma norm_sq_neg (z : ℍ) : norm_sq (-z) = norm_sq z :=
by simp [norm_sq]

```

```

@[simp] lemma norm_sq_conj (z : ℍ) : norm_sq (conj z) = norm_sq z :=
by simp [norm_sq]

```

```

@[simp] lemma norm_sq_mul (z w : ℍ) : norm_sq (z * w) = norm_sq z * norm_sq
w :=
by dsimp [norm_sq]; ring

```

```

lemma norm_sq_add (z w : ℍ) : norm_sq (z + w) =
norm_sq z + norm_sq w + 2 * (z * conj w).re :=
by dsimp [norm_sq]; ring

```


-- Question 2e

```
theorem mul_conj (z : ℍ) : z * conj z = norm_sq z :=  
ext_iff.2 $ by simp [norm_sq, mul_comm]
```

-- Question 2e

-- I did both because, in general, quaternion multiplication does not commute

```
theorem mul_conj' (z : ℍ) : conj z * z = norm_sq z :=  
ext_iff.2 $ by simp [norm_sq, mul_comm]
```

```
theorem mul_conj_is_re (z : ℍ) : ∃ (r : ℝ), (r : ℍ) = z * conj z :=  
by {use norm_sq z, rw mul_conj}
```

```
theorem mul_conj'_is_re (z : ℍ) : ∃ (r : ℝ), (r : ℍ) = conj z * z :=  
by {use norm_sq z, rw mul_conj'}
```

```
theorem add_conj (z : ℍ) : z + conj z = (2 * z.re : ℝ) :=  
ext_iff.2 $ by simp [two_mul]
```

```
lemma add_assoc (a b c : ℍ) : a + b + c = a + (b + c) :=  
by {ext, repeat {simp}}
```

```
lemma add_comm (a b : ℍ) : a + b = b + a :=  
by {ext, repeat {simp}}
```

```
lemma add_left_neg (a : ℍ) : -a + a = 0 :=  
by {ext, repeat {simp}}
```

```
lemma zero_add (a : ℍ) : 0 + a = a :=  
by {ext, repeat {simp}}
```

```
lemma add_zero (a : ℍ) : a + 0 = a :=  
by {rw add_comm, exact zero_add a}
```

```
instance : add_comm_group ℍ :=  
{add_assoc := add_assoc, zero_add := zero_add,  
add_zero := add_zero, add_left_neg := add_left_neg, add_comm := add_comm,  
..}
```

```
lemma mul_assoc (a b c : ℍ) : a * b * c = a * (b * c) :=  
by {ext; {simp, ring}}
```

```
lemma one_mul (a :  $\mathbb{H}$ ) : 1 * a = a :=
by {ext; {simp}}
```

```
lemma mul_one (a :  $\mathbb{H}$ ) : a * 1 = a :=
by {ext; {simp}}
```

```
lemma left_distrib (a b c :  $\mathbb{H}$ ) : a * (b + c) = a * b + a * c :=
by {ext; {simp, ring}}
```

```
lemma right_distrib (a b c :  $\mathbb{H}$ ) : (a + b) * c = a * c + b * c :=
by {ext; {simp, ring}}
```

```
-- Question 2b
```

```
-- Previous lemmas were written explicitly to make ring instance compile
faster
```

```
instance : ring  $\mathbb{H}$  :=
{mul := (*), mul_assoc := mul_assoc, one := 1, one_mul := one_mul,
  mul_one := mul_one, left_distrib := left_distrib, right_distrib :=
right_distrib, .. (by apply_instance : add_comm_group  $\mathbb{H}$ )}
.
```

```
@[simp] lemma sub_re (z w :  $\mathbb{H}$ ) : (z - w).re = z.re - w.re := rfl
```

```
@[simp] lemma sub_i (z w :  $\mathbb{H}$ ) : (z - w).i = z.i - w.i := rfl
```

```
@[simp] lemma sub_j (z w :  $\mathbb{H}$ ) : (z - w).j = z.j - w.j := rfl
```

```
@[simp] lemma sub_k (z w :  $\mathbb{H}$ ) : (z - w).k = z.k - w.k := rfl
```

```
@[simp] lemma of_real_sub (r s :  $\mathbb{R}$ ) : ((r - s :  $\mathbb{R}$ ) :  $\mathbb{H}$ ) = r - s :=
ext_iff.2 $ by simp
```

```
@[simp] lemma of_real_pow (r :  $\mathbb{R}$ ) (n :  $\mathbb{N}$ ) : ((r ^ n :  $\mathbb{R}$ ) :  $\mathbb{H}$ ) = r ^ n :=
by induction n; simp [*, of_real_mul, pow_succ]
```

```
theorem sub_conj (z :  $\mathbb{H}$ ) : z - conj z = (2 * z.i :  $\mathbb{R}$ ) * I + (2 * z.j :  $\mathbb{R}$ ) *
J + (2 * z.k :  $\mathbb{R}$ ) * K :=
ext_iff.2 $ by simp [two_mul]
```

```
lemma norm_sq_sub (z w :  $\mathbb{H}$ ) : norm_sq (z - w) =
norm_sq z + norm_sq w - 2 * (z * conj w).re :=
by rw [sub_eq_add_neg, norm_sq_add]; simp [-mul_re]
```

```
noncomputable instance : has_inv  $\mathbb{H}$  := ⟨λ z, conj z * ((norm_sq z)-1: $\mathbb{R}$ )⟩
```

```

theorem inv_def (z : ℍ) : z-1 = conj z * ((norm_sq z)-1:ℝ) := rfl
@[simp] lemma inv_re (z : ℍ) : (z-1).re = z.re / norm_sq z := by simp
[inv_def, division_def]
@[simp] lemma inv_i (z : ℍ) : (z-1).i = -z.i / norm_sq z := by simp
[inv_def, division_def]
@[simp] lemma inv_j (z : ℍ) : (z-1).j = -z.j / norm_sq z := by simp
[inv_def, division_def]
@[simp] lemma inv_k (z : ℍ) : (z-1).k = -z.k / norm_sq z := by simp
[inv_def, division_def]

@[simp] lemma of_real_inv (r : ℝ) : ((r-1 : ℝ) : ℍ) = r-1 :=
ext_iff.2 $ begin
  simp,
  by_cases r = 0, {simp [h]},
  rw [← div_div_eq_div_mul, div_self h, one_div_eq_inv]
end

protected lemma inv_zero : (0-1 : ℍ) = 0 :=
by rw [← of_real_zero, ← of_real_inv, inv_zero]

theorem mul_inv_cancel (z : ℍ) (h : z ≠ 0) : z * z-1 = 1 :=
by rw [inv_def, ← mul_assoc, mul_conj, ← of_real_mul,
  mul_inv_cancel (mt norm_sq_eq_zero.1 h), of_real_one]

theorem inv_mul_cancel (z : ℍ) (h : z ≠ 0) : z-1 * z = 1 :=
by rw [inv_def, ← mul_comm_re, mul_assoc, mul_conj', ← of_real_mul,
  inv_mul_cancel (mt norm_sq_eq_zero.1 h), of_real_one]

instance re.is_add_group_hom : is_add_group_hom quaternion.re :=
by refine_struct {..}; simp

instance i.is_add_group_hom : is_add_group_hom quaternion.i :=
by refine_struct {..}; simp

instance j.is_add_group_hom : is_add_group_hom quaternion.j :=
by refine_struct {..}; simp

instance k.is_add_group_hom : is_add_group_hom quaternion.k :=
by refine_struct {..}; simp

instance of_real.is_ring_hom : is_ring_hom (coe : ℝ → ℍ) :=
by {constructor, rfl, apply of_real_mul, apply of_real_add}

```

```

def smul :  $\mathbb{R} \rightarrow H \rightarrow H := \lambda r z, \text{of\_real } r * z$ 

infix `•` := smul

instance : has_scalar  $\mathbb{R} H := \{\text{smul} := \text{smul}\}$ 

lemma add_smul (r s :  $\mathbb{R}$ ) (z : H) : (r + s) • z = r • z + s • z :=
by {dsimp [(•)], ext; {simp, ring}}

lemma zero_smul (z : H) : 0 • z = 0 :=
by {dsimp [(•)], ext; simp}

lemma one_smul (z : H) : 1 • z = z :=
by {dsimp [(•)], simp}

lemma mul_smul (r s :  $\mathbb{R}$ ) (z : H) : (r * s) • z = r • (s • z) :=
by {dsimp [(•)], ext; {rw [of_real_mul, mul_assoc]}}

lemma smul_add (r :  $\mathbb{R}$ ) (z w : H) : r • (z + w) = r • z + r • w :=
by {dsimp [(•)], ext; {simp, ring}}

lemma smul_zero (r :  $\mathbb{R}$ ) : r • 0 = 0 :=
by {dsimp [(•)], simp}

instance : semimodule  $\mathbb{R} H :=
\{\text{add\_smul} := \text{add\_smul}, \text{zero\_smul} := \text{zero\_smul}, \text{one\_smul} := \text{one\_smul},
\text{mul\_smul} := \text{mul\_smul}, \text{smul\_add} := \text{smul\_add}, \text{smul\_zero} := \text{smul\_zero}, \dots\}$ 

instance : module  $\mathbb{R} H := \text{by constructor}$ 

lemma smul_def' (r :  $\mathbb{R}$ ) (z : H) : r • z = of_real r * z :=
begin
  dsimp [(•)],
  simp,
end

-- Question 2c
instance : algebra  $\mathbb{R} H :=
\{\text{to\_fun} := \text{of\_real}, \text{commutes'} := \text{by simp}, \text{smul\_def'} := \text{smul\_def'}, \text{hom} :=
\text{of\_real.is\_ring\_hom}\}$ 

```

```
-- Question 2g
noncomputable instance : division_ring  $\mathbb{H}$  :=
{inv := has_inv.inv, zero_ne_one := one_ne_zero.symm, mul_inv_cancel :=
mul_inv_cancel,
  inv_mul_cancel := inv_mul_cancel, .. (by apply_instance : ring  $\mathbb{H}$ )}
.

end quaternion
```

Question 2f: Normed Spaces

Note: I endeavoured to complete the proof that the quaternions were a normed vector space over the real numbers but was unable to prove the Triangle Inequality for the norm. The approach I would have taken would be to prove that the norm was induced by the standard inner product on Euclidean space (I had already proven that \mathbb{Q} is isomorphic to \mathbb{R}^4 as real vector spaces) but could not find inner products defined anywhere in MathLib.

```
import .Q2
import analysis.normed_space.basic

noncomputable theory

namespace quaternion

instance : has_norm  $\mathbb{H}$  := ⟨λ z, real.sqrt (norm_sq z)⟩

instance : has_dist  $\mathbb{H}$  := ⟨λ z w, //z-w//⟩

lemma dist_self (z :  $\mathbb{H}$ ) : dist z z = 0 :=
by {dsimp [dist], rw add_right_neg, dsimp [norm], simp}

lemma eq_of_dist_eq_zero (z w :  $\mathbb{H}$ ) : dist z w = 0 → z = w :=
begin
  intros h,
  dsimp [dist, norm] at h,
  have w := (real.sqrt_eq_zero (norm_sq_nonneg (z + -w))).1 h,
  rw norm_sq_eq_zero at w,
  exact eq_of_sub_eq_zero w,
end
```

```

lemma dist_comm (z w :  $\mathbb{H}$ ) : dist z w = dist w z :=
begin
  dsimp [dist, norm],
  have h1 : norm_sq (z-w) ≥ 0 := norm_sq_nonneg (z-w),
  have h2 : norm_sq (w-z) ≥ 0 := norm_sq_nonneg (w-z),
  apply (real.sqrt_inj h1 h2).2,
  dsimp [norm_sq],
  ring,
end
.

```

```

lemma le_of_eq_of_le {a b c :  $\mathbb{R}$ } : b = c → a ≤ b → a ≤ c :=
by {intros h w, rw [h] at w, exact w}

```

-- It turned out to be absolutely impossible to prove this without Cauchy-Schwartz.
 -- I would have proven that the norm was induced by an inner product first,
 -- but inner products are not implemented in MathLib anywhere that I could find.

```

lemma norm_triangle (z w :  $\mathbb{H}$ ) : ||z+w|| ≤ ||z||+||w|| :=
begin
  dsimp [norm],
  have h1 := add_nonneg (real.sqrt_nonneg (norm_sq z)) (real.sqrt_nonneg (norm_sq w)),
  rw [real.sqrt_le_left h1, pow_two],
  ring,
  rw [real.ring.right_distrib, ←pow_two, real.sqr_sqrt (norm_sq_nonneg z),
  real.sqr_sqrt (norm_sq_nonneg w)],
  have h2 := real.sqrt_mul (norm_sq_nonneg w) (norm_sq z),
  have h3 := congr_arg (λ (x :  $\mathbb{R}$ ), 2 * x) h2,
  simp at h3,
  rw [←real.ring.mul_assoc] at h3,
  have h4 : norm_sq z + 2 * real.sqrt (norm_sq w) * real.sqrt (norm_sq z) +
  norm_sq w = norm_sq z + 2 * real.sqrt (norm_sq w * norm_sq z) + norm_sq w, by
  {simp, exact h3.symm},
  apply le_of_eq_of_le h4.symm,
  clear h3 h4,
  rw [norm_sq_add],
  simp,
  have zero_lt_two : (0 :  $\mathbb{R}$ ) < 2 := by linarith,
  apply (@mul_le_mul_left _ _ (z.i * w.i + (z.j * w.j + (z.k * w.k + z.re *
  w.re))) _ 2 zero_lt_two).2,
  clear zero_lt_two,
  sorry,

```

end

```
lemma dist_triangle (x y z :  $\mathbb{H}$ ) : dist x z  $\leq$  dist x y + dist y z :=
begin
  dsimp [dist],
  have w := norm_triangle (x-y) (y-z),
  simp at w,
  exact w,
end
```

```
instance : metric_space  $\mathbb{H}$  :=
{dist := dist, dist_self := dist_self, eq_of_dist_eq_zero :=
eq_of_dist_eq_zero,
  dist_comm := dist_comm, dist_triangle := dist_triangle, ..}
```

```
lemma dist_eq (x y :  $\mathbb{H}$ ) : dist x y =  $\|x-y\|$  :=
by {dsimp [dist], refl}
```

```
instance : normed_group  $\mathbb{H}$  :=
{dist := dist, dist_eq := dist_eq, .. (by apply_instance : metric_space  $\mathbb{H}$ )}
```

```
instance : vector_space  $\mathbb{R}$   $\mathbb{H}$  := by constructor
```

```
lemma norm_smul (r :  $\mathbb{R}$ ) (z :  $\mathbb{H}$ ) :  $\|r \bullet z\| = \|r\| * \|z\|$  :=
begin
  dsimp [norm],
  rw  $\leftarrow$  real.sqrt_mul_self_eq_abs,
  rw  $\leftarrow$  (real.sqrt_mul' (r*r) (norm_sq_nonneg z)),
  have w := mul_nonneg (mul_self_nonneg r) (norm_sq_nonneg z),
  rw (real.sqrt_inj (norm_sq_nonneg (r  $\bullet$  z)) w),
  dsimp [norm_sq, ( $\bullet$ )],
  ring,
end
```

-- Question 2f

```
instance : normed_space  $\mathbb{R}$   $\mathbb{H}$  :=
{norm_smul := norm_smul, .. (by apply_instance : vector_space  $\mathbb{R}$   $\mathbb{H}$ )}
```

end quaternion