

Quaternions

Lawrence C. Paulson

December 14, 2021

Abstract

This theory is inspired by the HOL Light development of quaternions [1], but follows its own route. Quaternions are developed coinductively, as in the existing formalisation of the complex numbers. Quaternions are quickly shown to belong to the type classes of real normed division algebras and real inner product spaces. And therefore they inherit a great body of facts involving algebraic laws, limits, continuity, etc., which must be proved explicitly in the HOL Light version. The development concludes with the geometric interpretation of the product of imaginary quaternions.

Contents

1	Theory of Quaternions	2
1.1	Basic definitions	2
1.2	Addition and Subtraction: An Abelian Group	3
1.3	A Vector Space	3
1.4	Multiplication and Division: A Real Division Algebra	4
1.5	Multiplication and Division: A Real Normed Division Algebra	5
1.6	Conjugate of a quaternion	9
1.7	Linearity and continuity of the components	11
1.8	Quaternionic-specific theorems about sums	12
1.9	Bound results for real and imaginary components of limits	13
1.10	Quaternions for describing 3D isometries	14
	1.10.1 The HIm operator	14
	1.10.2 The Hv operator	15
1.11	Geometric interpretation of the product of imaginary quaternions	16
2	Acknowledgements	18

1 Theory of Quaternions

This theory is inspired by the HOL Light development of quaternions, but follows its own route. Quaternions are developed coinductively, as in the existing formalisation of the complex numbers. Quaternions are quickly shown to belong to the type classes of real normed division algebras and real inner product spaces. And therefore they inherit a great body of facts involving algebraic laws, limits, continuity, etc., which must be proved explicitly in the HOL Light version. The development concludes with the geometric interpretation of the product of imaginary quaternions.

```
theory Quaternions
  imports
    HOL-Analysis.Multivariate-Analysis
begin
```

1.1 Basic definitions

As with the complex numbers, coinduction is convenient

```
codatatype quat = Quat (Re: real) (Im1: real) (Im2: real) (Im3: real)
```

```
lemma quat-eqI [intro?]:  $\llbracket \text{Re } x = \text{Re } y; \text{Im1 } x = \text{Im1 } y; \text{Im2 } x = \text{Im2 } y; \text{Im3 } x = \text{Im3 } y \rrbracket \implies x = y$ 
  by (rule quat.expand simp)
```

```
lemma quat-eq-iff:  $x = y \longleftrightarrow \text{Re } x = \text{Re } y \wedge \text{Im1 } x = \text{Im1 } y \wedge \text{Im2 } x = \text{Im2 } y \wedge \text{Im3 } x = \text{Im3 } y$ 
  by (auto intro: quat.expand)
```

```
context
begin
no-notation Complex.imaginary-unit (i)
```

```
primcorec quat-ii :: quat (i)
  where  $\text{Re } i = 0 \mid \text{Im1 } i = 1 \mid \text{Im2 } i = 0 \mid \text{Im3 } i = 0$ 
```

```
primcorec quat-jj :: quat (j)
  where  $\text{Re } j = 0 \mid \text{Im1 } j = 0 \mid \text{Im2 } j = 1 \mid \text{Im3 } j = 0$ 
```

```
primcorec quat-kk :: quat (k)
  where  $\text{Re } k = 0 \mid \text{Im1 } k = 0 \mid \text{Im2 } k = 0 \mid \text{Im3 } k = 1$ 
```

```
end
```

```
bundle quaternion-syntax begin
notation quat-ii (i)
no-notation Complex.imaginary-unit (i)
end
```

```

bundle no-quaternion-syntax begin
no-notation quat-ii (i)
notation Complex.imaginary-unit (i)
end

```

```

unbundle quaternion-syntax

```

1.2 Addition and Subtraction: An Abelian Group

```

instantiation quat :: ab-group-add
begin

```

```

primcorec zero-quat
  where  $Re\ 0 = 0 \mid Im1\ 0 = 0 \mid Im2\ 0 = 0 \mid Im3\ 0 = 0$ 

```

```

primcorec plus-quat
  where
     $Re\ (x + y) = Re\ x + Re\ y$ 
  |  $Im1\ (x + y) = Im1\ x + Im1\ y$ 
  |  $Im2\ (x + y) = Im2\ x + Im2\ y$ 
  |  $Im3\ (x + y) = Im3\ x + Im3\ y$ 

```

```

primcorec uminus-quat
  where
     $Re\ (-x) = -Re\ x$ 
  |  $Im1\ (-x) = -Im1\ x$ 
  |  $Im2\ (-x) = -Im2\ x$ 
  |  $Im3\ (-x) = -Im3\ x$ 

```

```

primcorec minus-quat
  where
     $Re\ (x - y) = Re\ x - Re\ y$ 
  |  $Im1\ (x - y) = Im1\ x - Im1\ y$ 
  |  $Im2\ (x - y) = Im2\ x - Im2\ y$ 
  |  $Im3\ (x - y) = Im3\ x - Im3\ y$ 

```

```

instance
  by standard (simp-all add: quat-eq-iff)

```

```

end

```

1.3 A Vector Space

```

instantiation quat :: real-vector

```

```

begin

```

```

primcorec scaleR-quat
  where
     $Re\ (scaleR\ r\ x) = r * Re\ x$ 

```

| $Im1 (scaleR r x) = r * Im1 x$
| $Im2 (scaleR r x) = r * Im2 x$
| $Im3 (scaleR r x) = r * Im3 x$

instance

by *standard (auto simp: quat-eq-iff distrib-left distrib-right scaleR-add-right)*

end

instantiation *quat :: real-algebra-1*

begin

primcorec *one-quat*

where $Re 1 = 1$ | $Im1 1 = 0$ | $Im2 1 = 0$ | $Im3 1 = 0$

primcorec *times-quat*

where

$Re (x * y) = Re x * Re y - Im1 x * Im1 y - Im2 x * Im2 y - Im3 x * Im3 y$
| $Im1 (x * y) = Re x * Im1 y + Im1 x * Re y + Im2 x * Im3 y - Im3 x * Im2 y$
| $Im2 (x * y) = Re x * Im2 y - Im1 x * Im3 y + Im2 x * Re y + Im3 x * Im1 y$
| $Im3 (x * y) = Re x * Im3 y + Im1 x * Im2 y - Im2 x * Im1 y + Im3 x * Re y$

instance

by *standard (auto simp: quat-eq-iff distrib-left distrib-right Rings.right-diff-distrib Rings.left-diff-distrib)*

end

1.4 Multiplication and Division: A Real Division Algebra

instantiation *quat :: real-div-algebra*

begin

primcorec *inverse-quat*

where

$Re (inverse x) = Re x / ((Re x)^2 + (Im1 x)^2 + (Im2 x)^2 + (Im3 x)^2)$
| $Im1 (inverse x) = - (Im1 x) / ((Re x)^2 + (Im1 x)^2 + (Im2 x)^2 + (Im3 x)^2)$
| $Im2 (inverse x) = - (Im2 x) / ((Re x)^2 + (Im1 x)^2 + (Im2 x)^2 + (Im3 x)^2)$
| $Im3 (inverse x) = - (Im3 x) / ((Re x)^2 + (Im1 x)^2 + (Im2 x)^2 + (Im3 x)^2)$

definition $x \text{ div } y = x * \text{inverse } y$ **for** $x y :: \text{quat}$

instance

proof

show $\bigwedge x :: \text{quat}. x \neq 0 \implies \text{inverse } x * x = 1$

```

    by (auto simp: quat-eq-iff add-nonneg-eq-0-iff
        power2-eq-square add-divide-distrib [symmetric] diff-divide-distrib [symmetric])
  show  $\bigwedge x::\text{quat}. x \neq 0 \implies x * \text{inverse } x = 1$ 
    by (auto simp: quat-eq-iff add-nonneg-eq-0-iff power2-eq-square add-divide-distrib
        [symmetric])
  show  $\bigwedge x y::\text{quat}. x \text{ div } y = x * \text{inverse } y$ 
    by (simp add: divide-quat-def)
  show  $\text{inverse } 0 = (0::\text{quat})$ 
    by (auto simp: quat-eq-iff)
qed

end

```

1.5 Multiplication and Division: A Real Normed Division Algebra

```

fun quat-proj
  where
    quat-proj x 0 = Re x
  | quat-proj x (Suc 0) = Im1 x
  | quat-proj x (Suc (Suc 0)) = Im2 x
  | quat-proj x (Suc (Suc (Suc 0))) = Im3 x

lemma quat-proj-add:
  assumes  $i \leq 3$ 
  shows  $\text{quat-proj } (x+y) i = \text{quat-proj } x i + \text{quat-proj } y i$ 
proof -
  consider  $i = 0 \mid i = 1 \mid i = 2 \mid i = 3$ 
  using assms by linarith
  then show ?thesis
    by cases (auto simp: numeral-2-eq-2 numeral-3-eq-3)
qed

```

```

instantiation quat :: real-normed-div-algebra
begin

```

```

definition norm z = sqrt ((Re z)2 + (Im1 z)2 + (Im2 z)2 + (Im3 z)2)

```

```

definition sgn x = x /R norm x for x :: quat

```

```

definition dist x y = norm (x - y) for x y :: quat

```

```

definition [code del]:

```

```

  (uniformity :: (quat × quat) filter) = (INF e∈{0 <..}. principal {(x, y). dist x y < e})

```

```

definition [code del]:

```

```

  open (U :: quat set)  $\longleftrightarrow (\forall x \in U. \text{eventually } (\lambda(x', y). x' = x \longrightarrow y \in U)$ 
  uniformity)

```

```

lemma norm-eq-L2: norm z = L2-set (quat-proj z) {..3}
  by (simp add: norm-quat-def L2-set-def numeral-3-eq-3)

instance
proof
  fix r :: real and x y :: quat and S :: quat set
  show (norm x = 0)  $\longleftrightarrow$  (x = 0)
    by (simp add: norm-quat-def quat-eq-iff add-nonneg-eq-0-iff)
  have eq: L2-set (quat-proj (x + y)) {..3} = L2-set ( $\lambda i$ . quat-proj x i + quat-proj
y i) {..3}
    by (rule L2-set-cong) (auto simp: quat-proj-add)
  show norm (x + y)  $\leq$  norm x + norm y
    by (simp add: norm-eq-L2 eq L2-set-triangle-ineq)
  show norm (scaleR r x) = |r| * norm x
    by (simp add: norm-quat-def quat-eq-iff power-mult-distrib distrib-left [symmetric]
real-sqrt-mult)
  show norm (x * y) = norm x * norm y
    by (simp add: norm-quat-def quat-eq-iff real-sqrt-mult [symmetric]
power2-eq-square algebra-simps)
qed (rule sgn-quat-def dist-quat-def open-quat-def uniformity-quat-def)+

end

```

```

instantiation quat :: real-inner
begin

```

```

definition inner-quat-def:
  inner x y = Re x * Re y + Im1 x * Im1 y + Im2 x * Im2 y + Im3 x * Im3 y

```

```

instance

```

```

proof
  fix x y z :: quat and r :: real
  show inner x y = inner y x
    unfolding inner-quat-def by (simp add: mult.commute)
  show inner (x + y) z = inner x z + inner y z
    unfolding inner-quat-def by (simp add: distrib-right)
  show inner (scaleR r x) y = r * inner x y
    unfolding inner-quat-def by (simp add: distrib-left)
  show 0  $\leq$  inner x x
    unfolding inner-quat-def by simp
  show inner x x = 0  $\longleftrightarrow$  x = 0
    unfolding inner-quat-def by (simp add: add-nonneg-eq-0-iff quat-eq-iff)
  show norm x = sqrt (inner x x)
    unfolding inner-quat-def norm-quat-def
    by (simp add: power2-eq-square)
qed

```

end

lemma *quat-inner-1* [*simp*]: *inner 1 x = Re x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-1-right* [*simp*]: *inner x 1 = Re x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-i-left* [*simp*]: *inner i x = Im1 x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-i-right* [*simp*]: *inner x i = Im1 x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-j-left* [*simp*]: *inner j x = Im2 x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-j-right* [*simp*]: *inner x j = Im2 x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-k-left* [*simp*]: *inner k x = Im3 x*
unfolding *inner-quat-def* **by** *simp*

lemma *quat-inner-k-right* [*simp*]: *inner x k = Im3 x*
unfolding *inner-quat-def* **by** *simp*

abbreviation *quat-of-real* :: *real* \Rightarrow *quat*
where *quat-of-real* \equiv *of-real*

lemma *Re-quat-of-real* [*simp*]: *Re(quat-of-real a) = a*
by (*simp add: of-real-def*)

lemma *Im1-quat-of-real* [*simp*]: *Im1(quat-of-real a) = 0*
by (*simp add: of-real-def*)

lemma *Im2-quat-of-real* [*simp*]: *Im2(quat-of-real a) = 0*
by (*simp add: of-real-def*)

lemma *Im3-quat-of-real* [*simp*]: *Im3(quat-of-real a) = 0*
by (*simp add: of-real-def*)

lemma *quat-eq-0-iff*: $q = 0 \iff (Re\ q)^2 + (Im1\ q)^2 + (Im2\ q)^2 + (Im3\ q)^2 = 0$
proof

assume $(quat.Re\ q)^2 + (Im1\ q)^2 + (Im2\ q)^2 + (Im3\ q)^2 = 0$

then have $\forall qa. qa - q = qa$

by (*simp add: add-nonneg-eq-0-iff minus-quat.ctr*)

then show $q = 0$

by *simp*

qed *auto*

lemma *quat-of-real-times-commute*: $quat\text{-of-real } r * q = q * of\text{-real } r$
by (*simp add: of-real-def*)

lemma *quat-of-real-times-left-commute*: $quat\text{-of-real } r * (p * q) = p * (of\text{-real } r * q)$
by (*simp add: of-real-def*)

lemma *quat-norm-units* [*simp*]: $norm\ quat\text{-ii} = 1\ norm\ (j::quat) = 1\ norm\ (k::quat) = 1$
by (*auto simp: norm-quat-def*)

lemma *ii-nz* [*simp*]: $quat\text{-ii} \neq 0$
using *quat-ii.simps(2)* **by** *fastforce*

lemma *jj-nz* [*simp*]: $j \neq 0$
using *quat-jj.sel(3)* **by** *fastforce*

lemma *kk-nz* [*simp*]: $k \neq 0$
using *quat-kk.sel(4)* **by** *force*

An "expansion" theorem into the traditional notation

lemma *quat-unfold*:
 $q = of\text{-real}(Re\ q) + i * of\text{-real}(Im1\ q) + j * of\text{-real}(Im2\ q) + k * of\text{-real}(Im3\ q)$
by (*simp add: quat-eq-iff*)

lemma *quat-trad*: $Quat\ x\ y\ z\ w = of\text{-real } x + i * of\text{-real } y + j * of\text{-real } z + k * of\text{-real } w$
by (*simp add: quat-eq-iff*)

lemma *of-real-eq-Quat*: $of\text{-real } a = Quat\ a\ 0\ 0\ 0$
by (*simp add: quat-trad*)

lemma *ii-squared* [*simp*]: $quat\text{-ii}^2 = -1$
by (*simp add: power2-eq-square quat.expand*)

lemma *jj-squared* [*simp*]: $j^2 = -1$
by (*simp add: power2-eq-square quat.expand*)

lemma *kk-squared* [*simp*]: $k^2 = -1$
by (*simp add: power2-eq-square quat.expand*)

lemma *inverse-ii* [*simp*]: $inverse\ quat\text{-ii} = -quat\text{-ii}$
by (*simp add: power2-eq-square quat.expand*)

lemma *inverse-jj* [*simp*]: $inverse\ j = -j$
by (*simp add: power2-eq-square quat.expand*)

lemma *inverse-kk* [*simp*]: $inverse\ k = -k$

by (simp add: power2-eq-square quat.expand)

lemma inverse-mult: $\text{inverse } (p * q) = \text{inverse } q * \text{inverse } p$ for $p::\text{quat}$
by (metis inverse-zero mult-not-zero nonzero-inverse-mult-distrib)

lemma quat-of-real-inverse-collapse [simp]:
assumes $c \neq 0$
shows $\text{quat-of-real } c * \text{quat-of-real } (\text{inverse } c) = 1$ $\text{quat-of-real } (\text{inverse } c) * \text{quat-of-real } c = 1$
using assms by auto

1.6 Conjugate of a quaternion

primcorec cnj :: $\text{quat} \Rightarrow \text{quat}$

where

$\text{Re } (\text{cnj } z) = \text{Re } z$
| $\text{Im1 } (\text{cnj } z) = - \text{Im1 } z$
| $\text{Im2 } (\text{cnj } z) = - \text{Im2 } z$
| $\text{Im3 } (\text{cnj } z) = - \text{Im3 } z$

lemma cnj-cancel-iff [simp]: $\text{cnj } x = \text{cnj } y \iff x = y$

proof

show $\text{cnj } x = \text{cnj } y \implies x = y$

by (simp add: quat-eq-iff)

qed auto

lemma cnj-cnj [simp]:

$\text{cnj}(\text{cnj } q) = q$

by (simp add: quat-eq-iff)

lemma cnj-of-real [simp]: $\text{cnj}(\text{quat-of-real } x) = \text{quat-of-real } x$

by (simp add: quat-eqI)

lemma cnj-zero [simp]: $\text{cnj } 0 = 0$

by (simp add: quat-eq-iff)

lemma cnj-zero-iff [iff]: $\text{cnj } z = 0 \iff z = 0$

by (simp add: quat-eq-iff)

lemma cnj-one [simp]: $\text{cnj } 1 = 1$

by (simp add: quat-eq-iff)

lemma cnj-one-iff [simp]: $\text{cnj } z = 1 \iff z = 1$

by (simp add: quat-eq-iff)

lemma quat-norm-cnj [simp]: $\text{norm}(\text{cnj } q) = \text{norm } q$

by (simp add: norm-quat-def)

lemma *cnj-add* [*simp*]: $cnj (x + y) = cnj x + cnj y$
by (*simp add: quat-eq-iff*)

lemma *cnj-sum* [*simp*]: $cnj (sum f S) = (\sum x \in S. cnj (f x))$
by (*induct S rule: infinite-finite-induct*) *auto*

lemma *cnj-diff* [*simp*]: $cnj (x - y) = cnj x - cnj y$
by (*simp add: quat-eq-iff*)

lemma *cnj-minus* [*simp*]: $cnj (- x) = - cnj x$
by (*simp add: quat-eq-iff*)

lemma *cnj-mult* [*simp*]: $cnj (x * y) = cnj y * cnj x$
by (*simp add: quat-eq-iff*)

lemma *cnj-inverse* [*simp*]: $cnj (inverse x) = inverse (cnj x)$
by (*simp add: quat-eq-iff*)

lemma *cnj-divide* [*simp*]: $cnj (x / y) = inverse (cnj y) * cnj x$
by (*simp add: divide-quat-def*)

lemma *cnj-power* [*simp*]: $cnj (x ^ n) = cnj x ^ n$
by (*induct n*) (*auto simp: power-commutes*)

lemma *cnj-of-nat* [*simp*]: $cnj (of-nat n) = of-nat n$
by (*metis cnj-of-real of-real-of-nat-eq*)

lemma *cnj-of-int* [*simp*]: $cnj (of-int z) = of-int z$
by (*metis cnj-of-real of-real-of-int-eq*)

lemma *cnj-numeral* [*simp*]: $cnj (numeral w) = numeral w$
by (*metis of-nat-numeral cnj-of-nat*)

lemma *cnj-neg-numeral* [*simp*]: $cnj (- numeral w) = - numeral w$
by *simp*

lemma *cnj-scaleR* [*simp*]: $cnj (scaleR r x) = scaleR r (cnj x)$
by (*simp add: quat-eq-iff*)

lemma *cnj-units* [*simp*]: $cnj \text{quat-ii} = -i \text{cnj } j = -j \text{cnj } k = -k$
by (*simp-all add: quat-eq-iff*)

lemma *cnj-eq-of-real*: $cnj q = \text{quat-of-real } x \iff q = \text{quat-of-real } x$
proof
show $cnj q = \text{quat-of-real } x \implies q = \text{quat-of-real } x$
by (*metis cnj-of-real cnj-cnj*)
qed *auto*

lemma *quat-add-cnj*: $q + cnj q = \text{quat-of-real}(2 * \text{Re } q) \text{cnj } q + q = \text{quat-of-real}(2$

* $Re\ q$
by *simp-all* (*simp-all add: mult.commute mult-2 plus-quat.code*)

lemma *quat-divide-numeral*:
fixes $x::quat$ **shows** $x / numeral\ w = x /_R\ numeral\ w$
unfolding *divide-quat-def*
by (*metis mult.right-neutral mult-scaleR-right of-real-def of-real-inverse of-real-numeral*)

lemma *Re-divide-numeral* [*simp*]: $Re\ (x / numeral\ w) = Re\ x / numeral\ w$
by (*metis divide-inverse-commute quat-divide-numeral scaleR-quat.simps(1)*)

lemma *Im1-divide-numeral* [*simp*]: $Im1\ (x / numeral\ w) = Im1\ x / numeral\ w$
unfolding *quat-divide-numeral* **by** *simp*

lemma *Im2-divide-numeral* [*simp*]: $Im2\ (x / numeral\ w) = Im2\ x / numeral\ w$
unfolding *quat-divide-numeral* **by** *simp*

lemma *Im3-divide-numeral* [*simp*]: $Im3\ (x / numeral\ w) = Im3\ x / numeral\ w$
unfolding *quat-divide-numeral* **by** *simp*

lemma *of-real-quat-re-cnj*: $quat\ of\ real\ (Re\ q) = inverse\ (quat\ of\ real\ 2) * (q + cnj\ q)$
by (*simp add: quat-eq-iff*)

lemma *quat-mult-cnj-commute*: $cnj\ q * q = q * cnj\ q$
by (*simp add: quat-eq-iff*)

lemma *quat-norm-pow-2*: $quat\ of\ real\ (norm\ q) ^ 2 = q * cnj\ q$
by (*simp add: quat-eq-iff norm-quat-def flip: of-real-power*) (*auto simp: power2-eq-square*)

lemma *quat-norm-pow-2-alt*: $quat\ of\ real\ (norm\ q) ^ 2 = cnj\ q * q$
by (*simp add: quat-mult-cnj-commute quat-norm-pow-2*)

lemma *quat-inverse-cnj*: $inverse\ q = inverse\ (quat\ of\ real\ ((norm\ q)^2)) * cnj\ q$
by (*simp add: quat-eq-iff norm-quat-def numeral-Bit0 flip: of-real-power*)

lemma *quat-inverse-eq-cnj*: $norm\ q = 1 \implies inverse\ q = cnj\ q$
by (*metis inverse-1 mult-cancel-left norm-eq-zero norm-one cnj-one quat-norm-pow-2 right-inverse*)

1.7 Linearity and continuity of the components

lemma *bounded-linear-Re*: *bounded-linear Re*
and *bounded-linear-Im1*: *bounded-linear Im1*
and *bounded-linear-Im2*: *bounded-linear Im2*
and *bounded-linear-Im3*: *bounded-linear Im3*
by (*simp-all add: bounded-linear-intro [where K=1] norm-quat-def real-le-rsqrt add.assoc*)

lemmas *Cauchy-Re* = *bounded-linear.Cauchy* [OF *bounded-linear-Re*]
lemmas *Cauchy-Im1* = *bounded-linear.Cauchy* [OF *bounded-linear-Im1*]
lemmas *Cauchy-Im2* = *bounded-linear.Cauchy* [OF *bounded-linear-Im2*]
lemmas *Cauchy-Im3* = *bounded-linear.Cauchy* [OF *bounded-linear-Im3*]
lemmas *tendsto-Re* [*tendsto-intros*] = *bounded-linear.tendsto* [OF *bounded-linear-Re*]
lemmas *tendsto-Im1* [*tendsto-intros*] = *bounded-linear.tendsto* [OF *bounded-linear-Im1*]
lemmas *tendsto-Im2* [*tendsto-intros*] = *bounded-linear.tendsto* [OF *bounded-linear-Im2*]
lemmas *tendsto-Im3* [*tendsto-intros*] = *bounded-linear.tendsto* [OF *bounded-linear-Im3*]
lemmas *isCont-Re* [*simp*] = *bounded-linear.isCont* [OF *bounded-linear-Re*]
lemmas *isCont-Im1* [*simp*] = *bounded-linear.isCont* [OF *bounded-linear-Im1*]
lemmas *isCont-Im2* [*simp*] = *bounded-linear.isCont* [OF *bounded-linear-Im2*]
lemmas *isCont-Im3* [*simp*] = *bounded-linear.isCont* [OF *bounded-linear-Im3*]
lemmas *continuous-Re* [*simp*] = *bounded-linear.continuous* [OF *bounded-linear-Re*]
lemmas *continuous-Im1* [*simp*] = *bounded-linear.continuous* [OF *bounded-linear-Im1*]
lemmas *continuous-Im2* [*simp*] = *bounded-linear.continuous* [OF *bounded-linear-Im2*]
lemmas *continuous-Im3* [*simp*] = *bounded-linear.continuous* [OF *bounded-linear-Im3*]
lemmas *continuous-on-Re* [*continuous-intros*] = *bounded-linear.continuous-on*[OF
bounded-linear-Re]
lemmas *continuous-on-Im1* [*continuous-intros*] = *bounded-linear.continuous-on*[OF
bounded-linear-Im1]
lemmas *continuous-on-Im2* [*continuous-intros*] = *bounded-linear.continuous-on*[OF
bounded-linear-Im2]
lemmas *continuous-on-Im3* [*continuous-intros*] = *bounded-linear.continuous-on*[OF
bounded-linear-Im3]
lemmas *has-derivative-Re* [*derivative-intros*] = *bounded-linear.has-derivative*[OF
bounded-linear-Re]
lemmas *has-derivative-Im1* [*derivative-intros*] = *bounded-linear.has-derivative*[OF
bounded-linear-Im1]
lemmas *has-derivative-Im2* [*derivative-intros*] = *bounded-linear.has-derivative*[OF
bounded-linear-Im2]
lemmas *has-derivative-Im3* [*derivative-intros*] = *bounded-linear.has-derivative*[OF
bounded-linear-Im3]
lemmas *sums-Re* = *bounded-linear.sums* [OF *bounded-linear-Re*]
lemmas *sums-Im1* = *bounded-linear.sums* [OF *bounded-linear-Im1*]
lemmas *sums-Im2* = *bounded-linear.sums* [OF *bounded-linear-Im2*]
lemmas *sums-Im3* = *bounded-linear.sums* [OF *bounded-linear-Im3*]

1.8 Quaternionic-specific theorems about sums

lemma *Re-sum* [*simp*]: $\text{Re}(\text{sum } f \ S) = \text{sum } (\lambda x. \ \text{Re}(f \ x)) \ S$ for $f :: 'a \Rightarrow \text{quat}$
by (*induct* S rule: *infinite-finite-induct*) *auto*

lemma *Im1-sum* [*simp*]: $\text{Im1}(\text{sum } f \ S) = \text{sum } (\lambda x. \ \text{Im1}(f \ x)) \ S$
by (*induct* S rule: *infinite-finite-induct*) *auto*

lemma *Im2-sum* [*simp*]: $\text{Im2}(\text{sum } f \ S) = \text{sum } (\lambda x. \ \text{Im2}(f \ x)) \ S$
by (*induct* S rule: *infinite-finite-induct*) *auto*

lemma *Im3-sum* [*simp*]: $Im3(\text{sum } f \ S) = \text{sum } (\lambda x. Im3(f \ x)) \ S$
by (*induct S rule: infinite-finite-induct*) *auto*

lemma *in-Reals-iff-Re*: $q \in Reals \longleftrightarrow \text{quat-of-real}(Re \ q) = q$
by (*metis Re-quat-of-real Reals-cases Reals-of-real*)

lemma *in-Reals-iff-cnj*: $q \in Reals \longleftrightarrow \text{cnj } q = q$
by (*simp add: in-Reals-iff-Re quat-eq-iff*)

lemma *real-norm*: $q \in Reals \implies \text{norm } q = \text{abs}(Re \ q)$
by (*metis in-Reals-iff-Re norm-of-real*)

lemma *norm-power2*: $(\text{norm } q)^2 = Re(\text{cnj } q * q)$
by (*metis Re-quat-of-real of-real-power quat-mult-cnj-commute quat-norm-pow-2*)

lemma *quat-norm-imaginary*: $Re \ x = 0 \implies x^2 = -(\text{quat-of-real } (\text{norm } x))^2$
unfolding *quat-norm-pow-2*
by (*cases x*) (*auto simp: power2-eq-square cnj.ctr times-quat.ctr uminus-quat.ctr*)

1.9 Bound results for real and imaginary components of limits

lemma *Re-tendsto-upperbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. \text{quat.Re } (f \ x) \leq b; \text{net} \neq \text{bot} \rrbracket \implies Re \ l \leq b$
by (*blast intro: tendsto-upperbound [OF tendsto-Re]*)

lemma *Im1-tendsto-upperbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. Im1 \ (f \ x) \leq b; \text{net} \neq \text{bot} \rrbracket \implies Im1 \ l \leq b$
by (*blast intro: tendsto-upperbound [OF tendsto-Im1]*)

lemma *Im2-tendsto-upperbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. Im2 \ (f \ x) \leq b; \text{net} \neq \text{bot} \rrbracket \implies Im2 \ l \leq b$
by (*blast intro: tendsto-upperbound [OF tendsto-Im2]*)

lemma *Im3-tendsto-upperbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. Im3 \ (f \ x) \leq b; \text{net} \neq \text{bot} \rrbracket \implies Im3 \ l \leq b$
by (*blast intro: tendsto-upperbound [OF tendsto-Im3]*)

lemma *Re-tendsto-lowerbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. b \leq \text{quat.Re } (f \ x); \text{net} \neq \text{bot} \rrbracket \implies b \leq Re \ l$
by (*blast intro: tendsto-lowerbound [OF tendsto-Re]*)

lemma *Im1-tendsto-lowerbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. b \leq Im1 \ (f \ x); \text{net} \neq \text{bot} \rrbracket \implies b \leq Im1 \ l$
by (*blast intro: tendsto-lowerbound [OF tendsto-Im1]*)

lemma *Im2-tendsto-lowerbound*:
 $\llbracket (f \longrightarrow l) \ \text{net}; \forall_F \ x \ \text{in } \text{net}. b \leq Im2 \ (f \ x); \text{net} \neq \text{bot} \rrbracket \implies b \leq Im2 \ l$

by (blast intro: tendsto-lowerbound [OF tendsto-Im2])

lemma *Im3-tendsto-lowerbound*:

$\llbracket (f \longrightarrow l) \text{ net}; \forall_F x \text{ in net. } b \leq \text{Im3 } (f x); \text{net} \neq \text{bot} \rrbracket \implies b \leq \text{Im3 } l$

by (blast intro: tendsto-lowerbound [OF tendsto-Im3])

lemma *of-real-continuous-iff*: continuous net $(\lambda x. \text{quat-of-real}(f x)) \longleftrightarrow$ continuous net f

by (simp add: continuous-def tendsto-iff)

lemma *of-real-continuous-on-iff*:

continuous-on $S (\lambda x. \text{quat-of-real}(f x)) \longleftrightarrow$ continuous-on $S f$

using continuous-on-Re continuous-on-of-real by fastforce

1.10 Quaternions for describing 3D isometries

1.10.1 The *HIm* operator

definition *HIm* :: quat \Rightarrow real³ where

$HIm\ q \equiv \text{vector}[\text{Im1 } q, \text{Im2 } q, \text{Im3 } q]$

lemma *HIm-Quat*: $HIm (\text{Quat } w\ x\ y\ z) = \text{vector}[x,y,z]$

by (simp add: HIm-def)

lemma *him-eq*: $HIm\ p = HIm\ q \longleftrightarrow \text{Im1 } p = \text{Im1 } q \wedge \text{Im2 } p = \text{Im2 } q \wedge \text{Im3 } p = \text{Im3 } q$

by (metis HIm-def vector-3)

lemma *him-of-real* [simp]: $HIm(\text{of-real } a) = 0$

by (simp add: of-real-eq-Quat HIm-Quat vec-eq-iff vector-def)

lemma *him-0* [simp]: $HIm\ 0 = 0$

by (metis him-of-real of-real-0)

lemma *him-1* [simp]: $HIm\ 1 = 0$

by (metis him-of-real of-real-1)

lemma *him-cnj*: $HIm(\text{cnj } q) = - HIm\ q$

by (simp add: HIm-def vec-eq-iff vector-def)

lemma *him-mult-left* [simp]: $HIm(\text{of-real } a * q) = a *_R HIm\ q$

by (simp add: HIm-def vec-eq-iff vector-def)

lemma *him-mult-right* [simp]: $HIm(q * \text{of-real } a) = a *_R HIm\ q$

by (simp add: HIm-def vec-eq-iff vector-def)

lemma *him-add* [simp]: $HIm(p + q) = HIm\ p + HIm\ q$

by (simp add: HIm-def vec-eq-iff vector-def)

lemma *him-minus* [simp]: $HIm(-q) = - HIm\ q$

by (simp add: HIm-def vec-eq-iff vector-def)

lemma *him-diff* [simp]: $HIm(p - q) = HIm p - HIm q$
by (simp add: HIm-def vec-eq-iff vector-def)

lemma *him-sum* [simp]: $HIm (\text{sum } f S) = (\sum_{x \in S} HIm (f x))$
by (induct S rule: infinite-finite-induct) auto

lemma *linear-him*: linear HIm

by (metis him-add him-mult-right linearI mult.right-neutral mult-scaleR-right of-real-def)

1.10.2 The Hv operator

definition $Hv :: \text{real}^3 \Rightarrow \text{quat}$ where
 $Hv v \equiv \text{Quat } 0 (v\$1) (v\$2) (v\$3)$

lemma *Re-Hv* [simp]: $Re(Hv v) = 0$
by (simp add: Hv-def)

lemma *Im1-Hv* [simp]: $Im1(Hv v) = v\$1$
by (simp add: Hv-def)

lemma *Im2-Hv* [simp]: $Im2(Hv v) = v\$2$
by (simp add: Hv-def)

lemma *Im3-Hv* [simp]: $Im3(Hv v) = v\$3$
by (simp add: Hv-def)

lemma *hv-vec*: $Hv(\text{vec } r) = \text{Quat } 0 r r r$
by (simp add: Hv-def)

lemma *hv-eq-zero* [simp]: $Hv v = 0 \longleftrightarrow v = 0$
by (simp add: quat-eq-iff vec-eq-iff) (metis exhaust-3)

lemma *hv-zero* [simp]: $Hv 0 = 0$
by simp

lemma *hv-vector* [simp]: $Hv(\text{vector}[x,y,z]) = \text{Quat } 0 x y z$
by (simp add: Hv-def)

lemma *hv-basis*: $Hv(\text{axis } 1 1) = i$ $Hv(\text{axis } 2 1) = j$ $Hv(\text{axis } 3 1) = k$
by (auto simp: Hv-def axis-def quat-ii.code quat-jj.code quat-kk.code)

lemma *hv-add* [simp]: $Hv(x + y) = Hv x + Hv y$
by (simp add: Hv-def quat-eq-iff)

lemma *hv-minus* [simp]: $Hv(-x) = -Hv x$
by (simp add: Hv-def quat-eq-iff)

lemma *hv-diff* [*simp*]: $Hv(x - y) = Hv\ x - Hv\ y$
by (*simp add: Hv-def quat-eq-iff*)

lemma *hv-cmult* [*simp*]: $Hv(a *_R x) = of\text{-real}\ a * Hv\ x$
by (*simp add: Hv-def quat-eq-iff*)

lemma *hv-sum* [*simp*]: $Hv(\text{sum } f\ S) = (\sum_{x \in S}. Hv\ (f\ x))$
by (*induct S rule: infinite-finite-induct*) *auto*

lemma *hv-inj*: $Hv\ x = Hv\ y \longleftrightarrow x = y$
by (*simp add: Hv-def quat-eq-iff vec-eq-iff*) (*metis (full-types) exhaust-3*)

lemma *linear-hv*: *linear Hv*
by *unfold-locales (auto simp: of-real-def)*

lemma *him-hv* [*simp*]: $HIm(Hv\ x) = x$
using *HIm-def hv-inj quat-eq-iff* **by** *fastforce*

lemma *cnj-hv* [*simp*]: $cnj(Hv\ v) = -Hv\ v$
using *Hv-def cnj.code hv-minus* **by** *auto*

lemma *hv-him*: $Hv(HIm\ q) = Quat\ 0\ (Im1\ q)\ (Im2\ q)\ (Im3\ q)$
by (*simp add: HIm-def*)

lemma *hv-him-eq*: $Hv(HIm\ q) = q \longleftrightarrow Re\ q = 0$
by (*simp add: hv-him quat-eq-iff*)

lemma *dot-hv* [*simp*]: $Hv\ u \cdot Hv\ v = u \cdot v$
by (*simp add: Hv-def inner-quat-def inner-vec-def sum-3*)

lemma *norm-hv* [*simp*]: $norm\ (Hv\ v) = norm\ v$
by (*simp add: norm-eq*)

1.11 Geometric interpretation of the product of imaginary quaternions

context includes *cross3-syntax*
begin

lemma *mult-hv-eq-cross-dot*: $Hv\ x * Hv\ y = Hv(x \times y) - of\text{-real}\ (x \cdot y)$
by (*simp add: quat-eq-iff cross3-simps*)

Representing orthogonal transformations as conjugation or congruence with a quaternion

lemma *orthogonal-transformation-quat-congruence*:
assumes $norm\ q = 1$
shows *orthogonal-transformation* ($\lambda x. HIm(cnj\ q * Hv\ x * q)$)
proof –


```

have nq: (quat.Re q)2 + (Im1 q)2 + (Im2 q)2 + (Im3 q)2 = 1
  using assms norm-quat-def by auto
have Vector-Spaces.linear (*R) (*R) (λx. HIm (cnj q * Hv x * q))
proof
  show ∧r b. HIm (cnj q * Hv (r *R b) * q) = r *R HIm (cnj q * Hv b * q)
  by (metis him-mult-left hv-cmult mult-scaleR-left mult-scaleR-right scaleR-conv-of-real)
qed (simp add: distrib-left distrib-right)
moreover have HIm (cnj q * Hv v * q) · HIm (cnj q * Hv w * q) = ((quat.Re
q)2 + (Im1 q)2 + (Im2 q)2 + (Im3 q)2)2 * (v · w) for v w
  by (simp add: HIm-def inner-vec-def sum-3 power2-eq-square algebra-simps)
ultimately show ?thesis
  by (simp add: orthogonal-transformation-def linear-def nq)
qed

lemma orthogonal-transformation-quat-conjugation:
  assumes q ≠ 0
  shows orthogonal-transformation (λx. HIm(inverse q * Hv x * q))
proof -
  obtain c p where eq: q = of-real c * p and 1: norm p = 1
  proof
    show q = quat-of-real (norm q) * (inverse (of-real (norm q)) * q)
    by (metis assms mult.assoc mult.left-neutral norm-eq-zero of-real-eq-0-iff
right-inverse)
    show norm (inverse (quat-of-real (norm q)) * q) = 1
    using assms by (simp add: norm-mult norm-inverse)
  qed
  have c ≠ 0
  using assms eq by auto
  then have HIm (cnj p * Hv x * p) = HIm (inverse (quat-of-real c * p) * Hv x
* (quat-of-real c * p)) for x
  apply (simp add: inverse-mult mult.assoc flip: quat-inverse-eq-cnj [OF 1]
of-real-inverse)
  using quat-of-real-times-commute quat-of-real-times-left-commute quat-of-real-inverse-collapse
  by (simp add: of-real-def)
  then show ?thesis
  using orthogonal-transformation-quat-congruence [OF 1]
  by (simp add: eq)
qed

unbundle no-quaternion-syntax

end

end

```

2 Acknowledgements

The author was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council.

References

- [1] A. Gabrielli and M. Maggesi. Formalizing basic quaternionic analysis. In M. Ayala-Rincón and C. A. Muñoz, editors, *Interactive Theorem Proving*, pages 225–240. Springer, 2017.