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

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?*]: ⟦*Re x* = *Re y*; *Im1 x* = *Im1 y*; *Im2 x* = *Im2 y*; *Im3 x* = *Im3 y*⟧ ⟹ *x* = *y*
⟨*proof*⟩

lemma *quat-eq-iff*: *x* = *y* ⟷ *Re x* = *Re y* ∧ *Im1 x* = *Im1 y* ∧ *Im2 x* = *Im2 y* ∧ *Im3 x* = *Im3 y*
⟨*proof*⟩

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

primcorec *quat-ii* :: *quat* (i)
where *Re* i = *0* | *Im1* i = *1* | *Im2* i = *0* | *Im3* i = *0*

primcorec *quat-jj* :: *quat* (j)
where *Re* j = *0* | *Im1* j = *0* | *Im2* j = *1* | *Im3* j = *0*

primcorec *quat-kk* :: *quat* (k)
where *Re* k = *0* | *Im1* k = *0* | *Im2* k = *0* | *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* |*Im1 0 = 0* | *Im2 0 = 0* | *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
  ⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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 div y = x * inverse y* **for** *x y* :: *quat*

instance
⟨*proof*⟩

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 *quat-proj (x+y) i = quat-proj x i + quat-proj y i*
⟨*proof*⟩

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)* ⟷ (∀ *x*∈*U*. *eventually* (λ(*x*′, *y*). *x*′ = *x* ⟶ *y* ∈ *U*) *uniformity*)

lemma *norm-eq-L2*: *norm z = L2-set (quat-proj z)* {*..3*}
⟨*proof*⟩

instance
⟨*proof*⟩

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*⟩

end

lemma *quat-inner-1* [*simp*]: *inner 1 x* = *Re x*
⟨*proof*⟩

lemma *quat-inner-1-right* [*simp*]: *inner x 1* = *Re x*
⟨*proof*⟩

lemma *quat-inner-i-left* [*simp*]: *inner* i *x* = *Im1 x*
⟨*proof*⟩

lemma *quat-inner-i-right* [*simp*]: *inner x* i = *Im1 x*
⟨*proof*⟩

lemma *quat-inner-j-left* [*simp*]: *inner* j *x* = *Im2 x*
⟨*proof*⟩

lemma *quat-inner-j-right* [*simp*]: *inner x* j = *Im2 x*
⟨*proof*⟩

lemma *quat-inner-k-left* [*simp*]: *inner* k *x* = *Im3 x*
⟨*proof*⟩

lemma *quat-inner-k-right* [*simp*]: *inner x* k = *Im3 x*
⟨*proof*⟩

abbreviation *quat-of-real* :: *real* ⇒ *quat*
where *quat-of-real* ≡ *of-real*

lemma *Re-quat-of-real* [*simp*]: *Re*(*quat-of-real a*) = *a*
⟨*proof*⟩

lemma *Im1-quat-of-real* [*simp*]: *Im1*(*quat-of-real a*) = *0*
⟨*proof*⟩

lemma *Im2-quat-of-real* [*simp*]: *Im2*(*quat-of-real a*) = *0*
⟨*proof*⟩

lemma *Im3-quat-of-real* [*simp*]: *Im3*(*quat-of-real a*) = *0*
⟨*proof*⟩

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

lemma *quat-of-real-times-commute*: *quat-of-real r* ∗ *q* = *q* ∗ *of-real r*
⟨*proof*⟩

lemma *quat-of-real-times-left-commute*: *quat-of-real r* ∗ (*p* ∗ *q*) = *p* ∗ (*of-real r* ∗ *q*)

⟨*proof*⟩

lemma *quat-norm-units* [*simp*]: *norm quat-ii* = *1 norm* (j::*quat*) = *1 norm* (k::*quat*) = *1*
⟨*proof*⟩

lemma *ii-nz* [*simp*]: *quat-ii* ≠ *0*
⟨*proof*⟩

lemma *jj-nz* [*simp*]: j ≠ *0*
⟨*proof*⟩

lemma *kk-nz* [*simp*]: k ≠ *0*
⟨*proof*⟩

An "expansion" theorem into the traditional notation

lemma *quat-unfold*:
q = *of-real*(*Re q*) + i ∗ *of-real*(*Im1 q*) + j ∗ *of-real*(*Im2 q*) + k ∗ *of-real*(*Im3 q*)
⟨*proof*⟩

lemma *quat-trad*: *Quat x y z w* = *of-real x* + i ∗ *of-real y* + j ∗ *of-real z* + k ∗ *of-real w*
⟨*proof*⟩

lemma *of-real-eq-Quat*: *of-real a* = *Quat a 0 0 0*
⟨*proof*⟩

lemma *ii-squared* [*simp*]: *quat-ii*^{2} = −*1*
⟨*proof*⟩

lemma *jj-squared* [*simp*]: j^2 = −*1*
⟨*proof*⟩

lemma *kk-squared* [*simp*]: k^2 = −*1*
⟨*proof*⟩

lemma *inverse-ii* [*simp*]: *inverse quat-ii* = −*quat-ii*
⟨*proof*⟩

lemma *inverse-jj* [*simp*]: *inverse* j = −j
⟨*proof*⟩

lemma *inverse-kk* [*simp*]: *inverse* k = −k
⟨*proof*⟩

lemma *inverse-mult*: *inverse* (*p* ∗ *q*) = *inverse q* ∗ *inverse p* **for** *p*::*quat*
⟨*proof*⟩

lemma *quat-of-real-inverse-collapse* [*simp*]:

assumes $c \neq 0$
shows *quat-of-real* $c * quat\text{-}of\text{-}real\ (inverse\ c) = 1\ quat\text{-}of\text{-}real\ (inverse\ c) * quat\text{-}of\text{-}real\ c = 1$
⟨*proof*⟩

1.6 Conjugate of a quaternion

primcorec *cnj* :: *quat* ⇒ *quat*
where
Re (*cnj z*) = *Re z*
| *Im1* (*cnj z*) = − *Im1 z*
| *Im2* (*cnj z*) = − *Im2 z*
| *Im3* (*cnj z*) = − *Im3 z*

lemma *cnj-cancel-iff* [*simp*]: *cnj x* = *cnj y* ⟷ *x* = *y*
⟨*proof*⟩

lemma *cnj-cnj* [*simp*]:
cnj(*cnj q*) = *q*
⟨*proof*⟩

lemma *cnj-of-real* [*simp*]: *cnj*(*quat-of-real x*) = *quat-of-real x*
⟨*proof*⟩

lemma *cnj-zero* [*simp*]: *cnj 0* = *0*
⟨*proof*⟩

lemma *cnj-zero-iff* [*iff*]: *cnj z* = *0* ⟷ *z* = *0*
⟨*proof*⟩

lemma *cnj-one* [*simp*]: *cnj 1* = *1*
⟨*proof*⟩

lemma *cnj-one-iff* [*simp*]: *cnj z* = *1* ⟷ *z* = *1*
⟨*proof*⟩

lemma *quat-norm-cnj* [*simp*]: *norm*(*cnj q*) = *norm q*
⟨*proof*⟩

lemma *cnj-add* [*simp*]: *cnj* (*x* + *y*) = *cnj x* + *cnj y*
⟨*proof*⟩

lemma *cnj-sum* [*simp*]: *cnj* (*sum f S*) = (∑ *x*∈*S*. *cnj* (*f x*))
⟨*proof*⟩

lemma *cnj-diff* [*simp*]: *cnj* (*x* − *y*) = *cnj x* − *cnj y*
⟨*proof*⟩

lemma *cnj-minus* [*simp*]: *cnj* (− *x*) = − *cnj x*
⟨*proof*⟩

lemma *cnj-mult* [*simp*]: *cnj* (*x* ∗ *y*) = *cnj y* ∗ *cnj x*
⟨*proof*⟩

lemma *cnj-inverse* [*simp*]: *cnj* (*inverse x*) = *inverse* (*cnj x*)
⟨*proof*⟩

lemma *cnj-divide* [*simp*]: *cnj* (*x* / *y*) = *inverse* (*cnj y*) ∗ *cnj x*
⟨*proof*⟩

lemma *cnj-power* [*simp*]: *cnj* (*x*^*n*) = *cnj x* ^ *n*
⟨*proof*⟩

lemma *cnj-of-nat* [*simp*]: *cnj* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

lemma *cnj-of-int* [*simp*]: *cnj* (*of-int z*) = *of-int z*
⟨*proof*⟩

lemma *cnj-numeral* [*simp*]: *cnj* (*numeral w*) = *numeral w*
⟨*proof*⟩

lemma *cnj-neg-numeral* [*simp*]: *cnj* (− *numeral w*) = − *numeral w*
⟨*proof*⟩

lemma *cnj-scaleR* [*simp*]: *cnj* (*scaleR r x*) = *scaleR r* (*cnj x*)
⟨*proof*⟩

lemma *cnj-units* [*simp*]: *cnj quat-ii* = −i *cnj* j = −j *cnj* k = −k
⟨*proof*⟩

lemma *cnj-eq-of-real*: *cnj q* = *quat-of-real x* ⟷ *q* = *quat-of-real x*
⟨*proof*⟩

lemma *quat-add-cnj*: *q* + *cnj q* = *quat-of-real*(*2* ∗ *Re q*) *cnj q* + *q* = *quat-of-real*(*2* ∗ *Re q*)
⟨*proof*⟩

lemma *quat-divide-numeral*:
fixes *x*::*quat* **shows** *x* / *numeral w* = *x* /$_R$ *numeral w*
⟨*proof*⟩

lemma *Re-divide-numeral* [*simp*]: *Re* (*x* / *numeral w*) = *Re x* / *numeral w*
⟨*proof*⟩

lemma *Im1-divide-numeral* [*simp*]: *Im1* (*x* / *numeral w*) = *Im1 x* / *numeral w*
⟨*proof*⟩

lemma *Im2-divide-numeral* [*simp*]: *Im2* (*x* / *numeral w*) = *Im2 x* / *numeral w*
⟨*proof*⟩

lemma *Im3-divide-numeral* [*simp*]: *Im3* (*x* / *numeral w*) = *Im3 x* / *numeral w*
⟨*proof*⟩

lemma *of-real-quat-re-cnj*: *quat-of-real*(*Re q*) = *inverse*(*quat-of-real 2*) ∗ (*q* + *cnj q*)
⟨*proof*⟩

lemma *quat-mult-cnj-commute*: *cnj q* ∗ *q* = *q* ∗ *cnj q*
⟨*proof*⟩

lemma *quat-norm-pow-2*: *quat-of-real*(*norm q*) ^ *2* = *q* ∗ *cnj q*
⟨*proof*⟩

lemma *quat-norm-pow-2-alt*: *quat-of-real*(*norm q*) ^ *2* = *cnj q* ∗ *q*
⟨*proof*⟩

lemma *quat-inverse-cnj*: *inverse q* = *inverse* (*quat-of-real*((*norm q*)^2)) ∗ *cnj q*
⟨*proof*⟩

lemma *quat-inverse-eq-cnj*: *norm q* = *1* ⟹ *inverse q* = *cnj q*
⟨*proof*⟩

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*
⟨*proof*⟩

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*]: $Re(sum\ f\ S) = sum\ (\lambda x.\ \ Re(f\ x))\ S$ **for** $f :: 'a \Rightarrow quat$
⟨*proof*⟩

lemma *Im1-sum* [*simp*]: $Im1(sum\ f\ S) = sum\ (\lambda x.\ Im1(f\ x))\ S$
⟨*proof*⟩

lemma *Im2-sum* [*simp*]: $Im2(sum\ f\ S) = sum\ (\lambda x.\ Im2(f\ x))\ S$
⟨*proof*⟩

lemma *Im3-sum* [*simp*]: $Im3(sum\ f\ S) = sum\ (\lambda x.\ Im3(f\ x))\ S$
⟨*proof*⟩

lemma *in-Reals-iff-Re*: $q \in Reals \longleftrightarrow quat\text{-}of\text{-}real(Re\ q) = q$
⟨*proof*⟩

lemma *in-Reals-iff-cnj*: $q \in Reals \longleftrightarrow cnj\ q = q$
⟨*proof*⟩

lemma *real-norm*: $q \in Reals \Longrightarrow norm\ q = abs(Re\ q)$
⟨*proof*⟩

lemma *norm-power2*: $(norm\ q)^2 = Re(cnj\ q * q)$
⟨*proof*⟩

lemma *quat-norm-imaginary*: $Re\ x = 0 \Longrightarrow x^2 = -\ (quat\text{-}of\text{-}real\ (norm\ x))^2$
⟨*proof*⟩

1.9 Bound results for real and imaginary components of limits

lemma *Re-tendsto-upperbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ quat.Re\ (f\ x) \leq b; net \neq bot]\!] \Longrightarrow Re\ l \leq b$
⟨*proof*⟩

lemma *Im1-tendsto-upperbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ Im1\ (f\ x) \leq b; net \neq bot]\!] \Longrightarrow Im1\ l \leq b$
⟨*proof*⟩

lemma *Im2-tendsto-upperbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ Im2\ (f\ x) \leq b; net \neq bot]\!] \Longrightarrow Im2\ l \leq b$
⟨*proof*⟩

lemma *Im3-tendsto-upperbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ Im3\ (f\ x) \leq b; net \neq bot]\!] \Longrightarrow Im3\ l \leq b$
⟨*proof*⟩

lemma *Re-tendsto-lowerbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ b \leq quat.Re\ (f\ x); net \neq bot]\!] \Longrightarrow b \leq Re\ l$
⟨*proof*⟩

lemma *Im1-tendsto-lowerbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ b \leq Im1\ (f\ x); net \neq bot]\!] \Longrightarrow b \leq Im1\ l$
⟨*proof*⟩

lemma *Im2-tendsto-lowerbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ b \leq Im2\ (f\ x); net \neq bot]\!] \Longrightarrow b \leq Im2\ l$
⟨*proof*⟩

lemma *Im3-tendsto-lowerbound*:
$[\![(f \longrightarrow l)\ net; \forall_F\ x\ in\ net.\ b \leq Im3\ (f\ x); net \neq bot]\!] \Longrightarrow b \leq Im3\ l$
⟨*proof*⟩

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

lemma *of-real-continuous-on-iff*:
$continuous\text{-}on\ S\ (\lambda x.\ quat\text{-}of\text{-}real(f\ x)) \longleftrightarrow continuous\text{-}on\ S\ f$
⟨*proof*⟩

1.10 Quaternions for describing 3D isometries

1.10.1 The *HIm* operator

definition *HIm* :: *quat* ⇒ *real^3* **where**
HIm q ≡ *vector*[*Im1 q*, *Im2 q*, *Im3 q*]

lemma *HIm-Quat*: *HIm* (*Quat w x y z*) = *vector*[*x*,*y*,*z*]
⟨*proof*⟩

lemma *him-eq*: *HIm p* = *HIm q* ⟷ *Im1 p* = *Im1 q* ∧ *Im2 p* = *Im2 q* ∧ *Im3 p* = *Im3 q*
⟨*proof*⟩

lemma *him-of-real* [*simp*]: *HIm*(*of-real a*) = *0*
⟨*proof*⟩

lemma *him-0* [*simp*]: *HIm 0* = *0*
⟨*proof*⟩

lemma *him-1* [*simp*]: *HIm 1* = *0*
⟨*proof*⟩

lemma *him-cnj*: *HIm*(*cnj q*) = − *HIm q*
⟨*proof*⟩

lemma *him-mult-left* [*simp*]: *HIm*(*of-real a* ∗ *q*) = *a* ∗$_R$ *HIm q*
⟨*proof*⟩

lemma *him-mult-right* [*simp*]: *HIm*(*q* ∗ *of-real a*) = *a* ∗$_R$ *HIm q*
⟨*proof*⟩

lemma *him-add* [*simp*]: *HIm*(*p* + *q*) = *HIm p* + *HIm q*
⟨*proof*⟩

lemma *him-minus* [*simp*]: *HIm*(−*q*) = − *HIm q*
⟨*proof*⟩

lemma *him-diff* [*simp*]: *HIm*(*p* − *q*) = *HIm p* − *HIm q*
⟨*proof*⟩

lemma *him-sum* [*simp*]: *HIm* (*sum f S*) = (∑ *x*∈*S*. *HIm* (*f x*))
⟨*proof*⟩

lemma *linear-him*: *linear HIm*
⟨*proof*⟩

1.10.2 The *Hv* operator

definition *Hv* :: *real^3* ⇒ *quat* **where**

$Hv\ v \equiv Quat\ 0\ (v\$1)\ (v\$2)\ (v\$3)$

lemma *Re-Hv* [*simp*]: $Re(Hv\ v) = 0$
⟨*proof*⟩

lemma *Im1-Hv* [*simp*]: $Im1(Hv\ v) = v\$1$
⟨*proof*⟩

lemma *Im2-Hv* [*simp*]: $Im2(Hv\ v) = v\$2$
⟨*proof*⟩

lemma *Im3-Hv* [*simp*]: $Im3(Hv\ v) = v\$3$
⟨*proof*⟩

lemma *hv-vec*: $Hv(vec\ r) = Quat\ 0\ r\ r\ r$
⟨*proof*⟩

lemma *hv-eq-zero* [*simp*]: $Hv\ v = 0 \longleftrightarrow v = 0$
⟨*proof*⟩

lemma *hv-zero* [*simp*]: $Hv\ 0 = 0$
⟨*proof*⟩

lemma *hv-vector* [*simp*]: $Hv(vector[x,y,z]) = Quat\ 0\ x\ y\ z$
⟨*proof*⟩

lemma *hv-basis*: $Hv(axis\ 1\ 1) = \mathrm{i}\ Hv(axis\ 2\ 1) = \mathrm{j}\ Hv(axis\ 3\ 1) = \mathrm{k}$
⟨*proof*⟩

lemma *hv-add* [*simp*]: $Hv(x + y) = Hv\ x + Hv\ y$
⟨*proof*⟩

lemma *hv-minus* [*simp*]: $Hv(-x) = -Hv\ x$
⟨*proof*⟩

lemma *hv-diff* [*simp*]: $Hv(x - y) = Hv\ x - Hv\ y$
⟨*proof*⟩

lemma *hv-cmult* [*simp*]: $Hv(a *_R x) = of\text{-}real\ a * Hv\ x$
⟨*proof*⟩

lemma *hv-sum* [*simp*]: $Hv\ (sum\ f\ S) = (\sum x{\in}S.\ Hv\ (f\ x))$
⟨*proof*⟩

lemma *hv-inj*: $Hv\ x = Hv\ y \longleftrightarrow x = y$
⟨*proof*⟩

lemma *linear-hv*: *linear Hv*
⟨*proof*⟩

lemma *him-hv* [*simp*]: $HIm(Hv\ x) = x$
⟨*proof*⟩

lemma *cnj-hv* [*simp*]: $cnj(Hv\ v) = -Hv\ v$
⟨*proof*⟩

lemma *hv-him*: $Hv(HIm\ q) = Quat\ 0\ (Im1\ q)\ (Im2\ q)\ (Im3\ q)$
⟨*proof*⟩

lemma *hv-him-eq*: $Hv(HIm\ q) = q \longleftrightarrow Re\ q = 0$
⟨*proof*⟩

lemma *dot-hv* [*simp*]: $Hv\ u \cdot Hv\ v = u \cdot v$
⟨*proof*⟩

lemma *norm-hv* [*simp*]: $norm\ (Hv\ v) = norm\ v$
⟨*proof*⟩

1.11 Geometric interpretation of the product of imaginary quaternions

context includes *cross3-syntax*
begin

lemma *mult-hv-eq-cross-dot*: $Hv\ x * Hv\ y = Hv(x \times y) - of\text{-}real\ (x \cdot y)$
⟨*proof*⟩

Representing orthogonal transformations as conjugation or congruence with a quaternion

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

lemma *orthogonal-transformation-quat-conjugation*:
assumes $q \neq 0$
shows $orthogonal\text{-}transformation\ (\lambda x.\ HIm(inverse\ q * Hv\ x * q))$
⟨*proof*⟩

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.