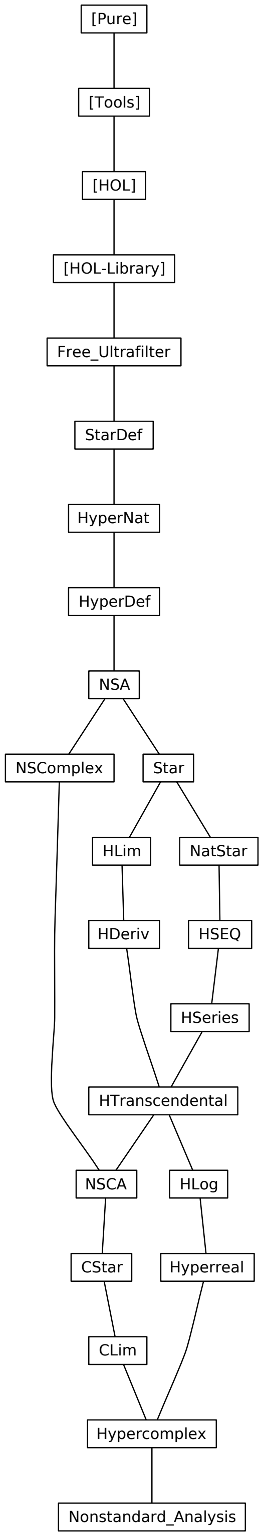
Isabelle/HOL-NSA — Non-Standard Analysis

December 17, 2025

Contents


[Pure]
[Tools]
[HOL]
[HOL-Library]
Free_Ultrafilter
StarDef
HyperNat
HyperDef
NSA
NSComplex
Star
HLim
NatStar
HDeriv
HSEQ
HSeries
HTranscendental
NSCA
HLog
CStar
Hyperreal
CLim
Hypercomplex
Nonstandard_Analysis

1 Filters and Ultrafilters

theory *Free-Ultrafilter*
  imports *HOL−Library.Infinite-Set*
begin

1.1 Definitions and basic properties

1.1.1 Ultrafilters

locale *ultrafilter* =
  fixes *F* :: *'a filter*
  assumes *proper*: $F \neq bot$
  assumes *ultra*: *eventually P F* ∨ *eventually* (λ*x*. ¬ *P x*) *F*
begin

lemma *eventually-imp-frequently*: *frequently P F* ⟹ *eventually P F*
  using *ultra*[*of P*] **by** (*simp add*: *frequently-def*)

lemma *frequently-eq-eventually*: *frequently P F* = *eventually P F*
  using *eventually-imp-frequently eventually-frequently*[*OF proper*] **..**

lemma *eventually-disj-iff*: *eventually* (λ*x*. *P x* ∨ *Q x*) *F* ⟷ *eventually P F* ∨ *eventually Q F*
  unfolding *frequently-eq-eventually*[*symmetric*] *frequently-disj-iff* **..**

lemma *eventually-all-iff*: *eventually* (λ*x*. ∀ *y*. *P x y*) *F* = (∀ *Y*. *eventually* (λ*x*. *P x* (*Y x*)) *F*)
  using *frequently-all*[*of P F*] **by** (*simp add*: *frequently-eq-eventually*)

lemma *eventually-imp-iff*: *eventually* (λ*x*. *P x* ⟶ *Q x*) *F* ⟷ (*eventually P F* ⟶ *eventually Q F*)
  using *frequently-imp-iff*[*of P Q F*] **by** (*simp add*: *frequently-eq-eventually*)

lemma *eventually-iff-iff*: *eventually* (λ*x*. *P x* ⟷ *Q x*) *F* ⟷ (*eventually P F* ⟷ *eventually Q F*)
  unfolding *iff-conv-conj-imp eventually-conj-iff eventually-imp-iff* **by** *simp*

lemma *eventually-not-iff*: *eventually* (λ*x*. ¬ *P x*) *F* ⟷ ¬ *eventually P F*
  unfolding *not-eventually frequently-eq-eventually* **..**

end

1.2 Maximal filter = Ultrafilter

A filter F is an ultrafilter iff it is a maximal filter, i.e. whenever G is a filter and $F \subseteq G$ then $F = G$

Lemma that shows existence of an extension to what was assumed to be a maximal filter. Will be used to derive contradiction in proof of property of ultrafilter.

```
lemma extend-filter: frequently P F ⟹ inf F (principal {x. P x}) ≠ bot
  by (simp add: trivial-limit-def eventually-inf-principal not-eventually)

lemma max-filter-ultrafilter:
  assumes F ≠ bot
  assumes max: ⋀G. G ≠ bot ⟹ G ≤ F ⟹ F = G
  shows ultrafilter F
proof
  show eventually P F ∨ (∀F x in F. ¬ P x) for P
  proof (rule disjCI)
    assume ¬ (∀F x in F. ¬ P x)
    then have inf F (principal {x. P x}) ≠ bot
      by (simp add: not-eventually extend-filter)
    then have F: F = inf F (principal {x. P x})
      by (rule max) simp
    show eventually P F
      by (subst F) (simp add: eventually-inf-principal)
  qed
qed fact

lemma le-filter-frequently: F ≤ G ⟷ (∀P. frequently P F ⟶ frequently P G)
  unfolding frequently-def le-filter-def
  apply auto
  apply (erule-tac x=λx. ¬ P x in allE)
  apply auto
  done

lemma (in ultrafilter) max-filter:
  assumes G: G ≠ bot
    and sub: G ≤ F
  shows F = G
proof (rule antisym)
  show F ≤ G
    using sub
     by (auto simp: le-filter-frequently[of F] frequently-eq-eventually le-filter-def[of
G]
             intro!: eventually-frequently G proper)
qed fact
```

1.3 Ultrafilter Theorem

```
lemma ex-max-ultrafilter:
  fixes F :: 'a filter
  assumes F: F ≠ bot
  shows ∃U≤F. ultrafilter U
proof -
  let ?X = {G. G ≠ bot ∧ G ≤ F}
  let ?R = {(b, a). a ≠ bot ∧ a ≤ b ∧ b ≤ F}
```

```
  have bot-notin-R: c ∈ Chains ?R ⟹ bot ∉ c for c
    by (auto simp: Chains-def)

  have [simp]: Field ?R = ?X
    by (auto simp: Field-def bot-unique)

  have ∃m∈Field ?R. ∀a∈Field ?R. (m, a) ∈ ?R ⟶ a = m (is ∃m∈?A. ?B m)
  proof (rule Zorns-po-lemma)
    show Partial-order ?R
      by (auto simp: partial-order-on-def preorder-on-def
          antisym-def refl-on-def trans-def Field-def bot-unique)
    show ∃u∈Field ?R. ∀a∈C. (a, u) ∈ ?R if C: C ∈ Chains ?R for C
    proof (simp, intro exI conjI ballI)
      have Inf-C: Inf C ≠ bot Inf C ≤ F if C ≠ {}
      proof -
        from C that have Inf C = bot ⟷ (∃x∈C. x = bot)
            unfolding trivial-limit-def by (intro eventually-Inf-base) (auto simp:
Chains-def)
        with C show Inf C ≠ bot
          by (simp add: bot-notin-R)
        from that obtain x where x ∈ C by auto
        with C show Inf C ≤ F
          by (auto intro!: Inf-lower2[of x] simp: Chains-def)
      qed
      then have [simp]: inf F (Inf C) = (if C = {} then F else Inf C)
        using C by (auto simp add: inf-absorb2)
      from C show inf F (Inf C) ≠ bot
        by (simp add: F Inf-C)
      from C show inf F (Inf C) ≤ F
        by (simp add: Chains-def Inf-C F)
      with C show inf F (Inf C) ≤ x x ≤ F if x ∈ C for x
        using that  by (auto intro: Inf-lower simp: Chains-def)
    qed
  qed
  then obtain U where U: U ∈ ?A ?B U ..
  show ?thesis
  proof
    from U show U ≤ F ∧ ultrafilter U
      by (auto intro!: max-filter-ultrafilter)
  qed
qed
```

1.3.1 Free Ultrafilters

There exists a free ultrafilter on any infinite set.

```
locale freeultrafilter = ultrafilter +
  assumes infinite: eventually P F ⟹ infinite {x. P x}
begin
```

```
lemma finite: finite {x. P x} ⟹ ¬ eventually P F
  by (erule contrapos-pn) (erule infinite)

lemma finite′: finite {x. ¬ P x} ⟹ eventually P F
  by (drule finite) (simp add: not-eventually frequently-eq-eventually)

lemma le-cofinite: F ≤ cofinite
  by (intro filter-leI)
    (auto simp add: eventually-cofinite not-eventually frequently-eq-eventually dest!:
finite)

lemma singleton: ¬ eventually (λx. x = a) F
  by (rule finite) simp

lemma singleton′: ¬ eventually ((=) a) F
  by (rule finite) simp

lemma ultrafilter: ultrafilter F ..

end

lemma freeultrafilter-Ex:
  assumes [simp]: infinite (UNIV :: ′a set)
  shows ∃ U::′a filter. freeultrafilter U
proof −
  from ex-max-ultrafilter[of cofinite :: ′a filter]
  obtain U :: ′a filter where U ≤ cofinite ultrafilter U
    by auto
  interpret ultrafilter U by fact
  have freeultrafilter U
  proof
    fix P
    assume eventually P U
    with proper have frequently P U
      by (rule eventually-frequently)
    then have frequently P cofinite
      using ‹U ≤ cofinite› by (simp add: le-filter-frequently)
    then show infinite {x. P x}
      by (simp add: frequently-cofinite)
  qed
  then show ?thesis ..
qed

end
```

2 Construction of Star Types Using Ultrafilters

```
theory StarDef
```

imports *Free-Ultrafilter*
begin

2.1 A Free Ultrafilter over the Naturals

definition *FreeUltrafilterNat* :: *nat filter* (‹$\mathcal{U}$›)
where $\mathcal{U}$ = (*SOME U. freeultrafilter U*)

lemma *freeultrafilter-FreeUltrafilterNat*: *freeultrafilter* $\mathcal{U}$
unfolding *FreeUltrafilterNat-def*
by (*simp add*: *freeultrafilter-Ex someI-ex*)

interpretation *FreeUltrafilterNat*: *freeultrafilter* $\mathcal{U}$
by (*rule freeultrafilter-FreeUltrafilterNat*)

2.2 Definition of *star* type constructor

definition *starrel* :: $((nat \Rightarrow 'a) \times (nat \Rightarrow 'a))$ *set*
where *starrel* = $\{(X, Y).$ *eventually* $(\lambda n.\ X\ n = Y\ n)\ \mathcal{U}\}$

definition *star* = (*UNIV* :: $(nat \Rightarrow 'a)$ *set*) // *starrel*

typedef *'a star* = *star* :: $(nat \Rightarrow 'a)$ *set set*
by (*auto simp*: *star-def intro*: *quotientI*)

definition *star-n* :: $(nat \Rightarrow 'a) \Rightarrow 'a$ *star*
where *star-n X* = *Abs-star* (*starrel* ‘‘ $\{X\}$)

theorem *star-cases* [*case-names star-n*, *cases type*: *star*]:
obtains *X* **where** *x* = *star-n X*
by (*cases x*) (*auto simp*: *star-n-def star-def elim*: *quotientE*)

lemma *all-star-eq*: $(\forall x.\ P\ x) \longleftrightarrow (\forall X.\ P\ (star\text{-}n\ X))$
by (*metis star-cases*)

lemma *ex-star-eq*: $(\exists x.\ P\ x) \longleftrightarrow (\exists X.\ P\ (star\text{-}n\ X))$
by (*metis star-cases*)

Proving that *starrel* is an equivalence relation.

lemma *starrel-iff* [*iff*]: $(X, Y) \in starrel \longleftrightarrow$ *eventually* $(\lambda n.\ X\ n = Y\ n)\ \mathcal{U}$
by (*simp add*: *starrel-def*)

lemma *equiv-starrel*: *equiv UNIV starrel*
proof (*rule equivI*)
show *starrel* $\subseteq$ *UNIV* $\times$ *UNIV* **by** *simp*
show *refl starrel* **by** (*simp add*: *refl-on-def*)
show *sym starrel* **by** (*simp add*: *sym-def eq-commute*)
show *trans starrel* **by** (*intro transI*) (*auto elim*: *eventually-elim2*)
qed

lemmas *equiv-starrel-iff* = *eq-equiv-class-iff* [*OF equiv-starrel UNIV-I UNIV-I*]

lemma *starrel-in-star*: *starrel*‘‘{*x*} ∈ *star*
by (*simp add*: *star-def quotientI*)

lemma *star-n-eq-iff*: *star-n X* = *star-n Y* ⟷ *eventually* (λ*n*. *X n* = *Y n*) 𝒰
by (*simp add*: *star-n-def Abs-star-inject starrel-in-star equiv-starrel-iff*)

2.3 Transfer principle

This introduction rule starts each transfer proof.

lemma *transfer-start*: *P* ≡ *eventually* (λ*n*. *Q*) 𝒰 ⟹ *Trueprop P* ≡ *Trueprop Q*
by (*simp add*: *FreeUltrafilterNat.proper*)

Standard principles that play a central role in the transfer tactic.

definition *Ifun* :: (′*a* ⇒ ′*b*) *star* ⇒ ′*a star* ⇒ ′*b star*
(‹(‹*notation*=‹*infix* ⋆››- ⋆/ -)› [*300*, *301*] *300*)
where *Ifun f* ≡
λ*x*. *Abs-star* (⋃*F*∈*Rep-star f*. ⋃*X*∈*Rep-star x*. *starrel*‘‘{λ*n*. *F n* (*X n*)})

lemma *Ifun-congruent2*: *congruent2 starrel starrel* (λ*F X*. *starrel*‘‘{λ*n*. *F n* (*X n*)})
by (*auto simp add*: *congruent2-def equiv-starrel-iff elim*!: *eventually-rev-mp*)

lemma *Ifun-star-n*: *star-n F* ⋆ *star-n X* = *star-n* (λ*n*. *F n* (*X n*))
by (*simp add*: *Ifun-def star-n-def Abs-star-inverse starrel-in-star*
UN-equiv-class2 [*OF equiv-starrel equiv-starrel Ifun-congruent2*])

lemma *transfer-Ifun*: *f* ≡ *star-n F* ⟹ *x* ≡ *star-n X* ⟹ *f* ⋆ *x* ≡ *star-n* (λ*n*. *F n* (*X n*))
by (*simp only*: *Ifun-star-n*)

definition *star-of* :: ′*a* ⇒ ′*a star*
where *star-of x* ≡ *star-n* (λ*n*. *x*)

Initialize transfer tactic.

ML-file ‹*transfer-principle.ML*›

method-setup *transfer* =
‹*Attrib.thms* >> (*fn ths* => *fn ctxt* => *SIMPLE-METHOD*′ (*Transfer-Principle.transfer-tac ctxt ths*))›
transfer principle

Transfer introduction rules.

lemma *transfer-ex* [*transfer-intro*]:
(⋀*X*. *p* (*star-n X*) ≡ *eventually* (λ*n*. *P n* (*X n*)) 𝒰) ⟹
∃ *x*::′*a star*. *p x* ≡ *eventually* (λ*n*. ∃ *x*. *P n x*) 𝒰

by (*simp only*: *ex-star-eq eventually-ex*)

lemma *transfer-all* [*transfer-intro*]:
$(\bigwedge X.\ p\ (star\text{-}n\ X) \equiv eventually\ (\lambda n.\ P\ n\ (X\ n))\ \mathcal{U}) \Longrightarrow$
$\forall x{::}'a\ star.\ p\ x \equiv eventually\ (\lambda n.\ \forall x.\ P\ n\ x)\ \mathcal{U}$
by (*simp only*: *all-star-eq FreeUltrafilterNat.eventually-all-iff*)

lemma *transfer-not* [*transfer-intro*]: $p \equiv eventually\ P\ \mathcal{U} \Longrightarrow \neg\ p \equiv eventually\ (\lambda n.\ \neg\ P\ n)\ \mathcal{U}$
by (*simp only*: *FreeUltrafilterNat.eventually-not-iff*)

lemma *transfer-conj* [*transfer-intro*]:
$p \equiv eventually\ P\ \mathcal{U} \Longrightarrow q \equiv eventually\ Q\ \mathcal{U} \Longrightarrow p \wedge q \equiv eventually\ (\lambda n.\ P\ n \wedge Q\ n)\ \mathcal{U}$
by (*simp only*: *eventually-conj-iff*)

lemma *transfer-disj* [*transfer-intro*]:
$p \equiv eventually\ P\ \mathcal{U} \Longrightarrow q \equiv eventually\ Q\ \mathcal{U} \Longrightarrow p \vee q \equiv eventually\ (\lambda n.\ P\ n \vee Q\ n)\ \mathcal{U}$
by (*simp only*: *FreeUltrafilterNat.eventually-disj-iff*)

lemma *transfer-imp* [*transfer-intro*]:
$p \equiv eventually\ P\ \mathcal{U} \Longrightarrow q \equiv eventually\ Q\ \mathcal{U} \Longrightarrow p \longrightarrow q \equiv eventually\ (\lambda n.\ P\ n \longrightarrow Q\ n)\ \mathcal{U}$
by (*simp only*: *FreeUltrafilterNat.eventually-imp-iff*)

lemma *transfer-iff* [*transfer-intro*]:
$p \equiv eventually\ P\ \mathcal{U} \Longrightarrow q \equiv eventually\ Q\ \mathcal{U} \Longrightarrow p = q \equiv eventually\ (\lambda n.\ P\ n = Q\ n)\ \mathcal{U}$
by (*simp only*: *FreeUltrafilterNat.eventually-iff-iff*)

lemma *transfer-if-bool* [*transfer-intro*]:
$p \equiv eventually\ P\ \mathcal{U} \Longrightarrow x \equiv eventually\ X\ \mathcal{U} \Longrightarrow y \equiv eventually\ Y\ \mathcal{U} \Longrightarrow$
$(if\ p\ then\ x\ else\ y) \equiv eventually\ (\lambda n.\ if\ P\ n\ then\ X\ n\ else\ Y\ n)\ \mathcal{U}$
by (*simp only*: *if-bool-eq-conj transfer-conj transfer-imp transfer-not*)

lemma *transfer-eq* [*transfer-intro*]:
$x \equiv star\text{-}n\ X \Longrightarrow y \equiv star\text{-}n\ Y \Longrightarrow x = y \equiv eventually\ (\lambda n.\ X\ n = Y\ n)\ \mathcal{U}$
by (*simp only*: *star-n-eq-iff*)

lemma *transfer-if* [*transfer-intro*]:
$p \equiv eventually\ (\lambda n.\ P\ n)\ \mathcal{U} \Longrightarrow x \equiv star\text{-}n\ X \Longrightarrow y \equiv star\text{-}n\ Y \Longrightarrow$
$(if\ p\ then\ x\ else\ y) \equiv star\text{-}n\ (\lambda n.\ if\ P\ n\ then\ X\ n\ else\ Y\ n)$
by (*rule eq-reflection*) (*auto simp*: *star-n-eq-iff transfer-not elim*!: *eventually-mono*)

lemma *transfer-fun-eq* [*transfer-intro*]:
$(\bigwedge X.\ f\ (star\text{-}n\ X) = g\ (star\text{-}n\ X) \equiv eventually\ (\lambda n.\ F\ n\ (X\ n) = G\ n\ (X\ n))\ \mathcal{U}) \Longrightarrow$
$f = g \equiv eventually\ (\lambda n.\ F\ n = G\ n)\ \mathcal{U}$

by (*simp only*: *fun-eq-iff transfer-all*)

lemma *transfer-star-n* [*transfer-intro*]: $star\text{-}n\ X \equiv star\text{-}n\ (\lambda n.\ X\ n)$
by (*rule reflexive*)

lemma *transfer-bool* [*transfer-intro*]: $p \equiv eventually\ (\lambda n.\ p)\ \mathcal{U}$
by (*simp add*: *FreeUltrafilterNat.proper*)

2.4 Standard elements

definition *Standard* :: *'a star set*
where *Standard* = *range star-of*

Transfer tactic should remove occurrences of *star-of*.

setup ‹*Transfer-Principle.add-const* ***const-name*** ‹*star-of*››

lemma *star-of-inject*: $star\text{-}of\ x = star\text{-}of\ y \longleftrightarrow x = y$
by *transfer* (*rule refl*)

lemma *Standard-star-of* [*simp*]: $star\text{-}of\ x \in Standard$
by (*simp add*: *Standard-def*)

2.5 Internal functions

Transfer tactic should remove occurrences of *Ifun*.

setup ‹*Transfer-Principle.add-const* ***const-name*** ‹*Ifun*››

lemma *Ifun-star-of* [*simp*]: $star\text{-}of\ f \star star\text{-}of\ x = star\text{-}of\ (f\ x)$
by *transfer* (*rule refl*)

lemma *Standard-Ifun* [*simp*]: $f \in Standard \Longrightarrow x \in Standard \Longrightarrow f \star x \in Standard$
by (*auto simp add*: *Standard-def*)

Nonstandard extensions of functions.

definition *starfun* :: $('a \Rightarrow 'b) \Rightarrow 'a\ star \Rightarrow 'b\ star$
(‹(‹*open-block notation=*‹*prefix starfun*››*f* -)› [*80*] *80*)
where $starfun\ f \equiv \lambda x.\ star\text{-}of\ f \star x$

definition *starfun2* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a\ star \Rightarrow 'b\ star \Rightarrow 'c\ star$
(‹(‹*open-block notation=*‹*prefix starfun2*››*f2* -)› [*80*] *80*)
where $starfun2\ f \equiv \lambda x\ y.\ star\text{-}of\ f \star x \star y$

declare *starfun-def* [*transfer-unfold*]
declare *starfun2-def* [*transfer-unfold*]

lemma *starfun-star-n*: $(\ {*f*}\ f)\ (star\text{-}n\ X) = star\text{-}n\ (\lambda n.\ f\ (X\ n))$
by (*simp only*: *starfun-def star-of-def Ifun-star-n*)

lemma *starfun2-star-n*: $(*f2* f)\ (star\text{-}n\ X)\ (star\text{-}n\ Y) = star\text{-}n\ (\lambda n.\ f\ (X\ n)\ (Y\ n))$
  by (*simp only*: *starfun2-def star-of-def Ifun-star-n*)

lemma *starfun-star-of* [*simp*]: $(*f* f)\ (star\text{-}of\ x) = star\text{-}of\ (f\ x)$
  by *transfer* (*rule refl*)

lemma *starfun2-star-of* [*simp*]: $(*f2* f)\ (star\text{-}of\ x) = *f* f\ x$
  by *transfer* (*rule refl*)

lemma *Standard-starfun* [*simp*]: $x \in Standard \Longrightarrow starfun\ f\ x \in Standard$
  by (*simp add*: *starfun-def*)

lemma *Standard-starfun2* [*simp*]: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow starfun2\ f\ x\ y \in Standard$
  by (*simp add*: *starfun2-def*)

lemma *Standard-starfun-iff*:
  assumes *inj*: $\bigwedge x\ y.\ f\ x = f\ y \Longrightarrow x = y$
  shows $starfun\ f\ x \in Standard \longleftrightarrow x \in Standard$
proof
  assume $x \in Standard$
  then show $starfun\ f\ x \in Standard$ **by** *simp*
next
  from *inj* **have** *inj'*: $\bigwedge x\ y.\ starfun\ f\ x = starfun\ f\ y \Longrightarrow x = y$
    by *transfer*
  assume $starfun\ f\ x \in Standard$
  then obtain b **where** b: $starfun\ f\ x = star\text{-}of\ b$
    unfolding *Standard-def* **..**
  then have $\exists x.\ starfun\ f\ x = star\text{-}of\ b$ **..**
  then have $\exists a.\ f\ a = b$ **by** *transfer*
  then obtain a **where** $f\ a = b$ **..**
  then have $starfun\ f\ (star\text{-}of\ a) = star\text{-}of\ b$ **by** *transfer*
  with b **have** $starfun\ f\ x = starfun\ f\ (star\text{-}of\ a)$ **by** *simp*
  then have $x = star\text{-}of\ a$ **by** (*rule inj'*)
  then show $x \in Standard$ **by** (*simp add*: *Standard-def*)
qed

lemma *Standard-starfun2-iff*:
  assumes *inj*: $\bigwedge a\ b\ a'\ b'.\ f\ a\ b = f\ a'\ b' \Longrightarrow a = a' \wedge b = b'$
  shows $starfun2\ f\ x\ y \in Standard \longleftrightarrow x \in Standard \wedge y \in Standard$
proof
  assume $x \in Standard \wedge y \in Standard$
  then show $starfun2\ f\ x\ y \in Standard$ **by** *simp*
next
  have *inj'*: $\bigwedge x\ y\ z\ w.\ starfun2\ f\ x\ y = starfun2\ f\ z\ w \Longrightarrow x = z \wedge y = w$
    using *inj* **by** *transfer*
  assume $starfun2\ f\ x\ y \in Standard$
  then obtain c **where** c: $starfun2\ f\ x\ y = star\text{-}of\ c$

unfolding *Standard-def* **..**
then have $\exists\, x\ y.\ starfun2\ f\ x\ y = star\text{-}of\ c$ **by** *auto*
then have $\exists\, a\ b.\ f\ a\ b = c$ **by** *transfer*
then obtain *a b* **where** $f\ a\ b = c$ **by** *auto*
then have $starfun2\ f\ (star\text{-}of\ a)\ (star\text{-}of\ b) = star\text{-}of\ c$ **by** *transfer*
with *c* **have** $starfun2\ f\ x\ y = starfun2\ f\ (star\text{-}of\ a)\ (star\text{-}of\ b)$ **by** *simp*
then have $x = star\text{-}of\ a \wedge y = star\text{-}of\ b$ **by** (*rule inj'*)
then show $x \in Standard \wedge y \in Standard$ **by** (*simp add*: *Standard-def*)
qed

2.6 Internal predicates

definition *unstar* :: *bool star* ⇒ *bool*
where $unstar\ b \longleftrightarrow b = star\text{-}of\ True$

lemma *unstar-star-n*: $unstar\ (star\text{-}n\ P) \longleftrightarrow eventually\ P\ \mathcal{U}$
by (*simp add*: *unstar-def star-of-def star-n-eq-iff*)

lemma *unstar-star-of* [*simp*]: $unstar\ (star\text{-}of\ p) = p$
by (*simp add*: *unstar-def star-of-inject*)

Transfer tactic should remove occurrences of *unstar*.

setup ‹*Transfer-Principle.add-const* ***const-name*** ‹*unstar*››

lemma *transfer-unstar* [*transfer-intro*]: $p \equiv star\text{-}n\ P \Longrightarrow unstar\ p \equiv eventually\ P\ \mathcal{U}$
by (*simp only*: *unstar-star-n*)

definition *starP* :: $('a \Rightarrow bool) \Rightarrow 'a\ star \Rightarrow bool$
(‹(‹*open-block notation*=‹*prefix starP*››*∗p∗* -)› [*80*] *80*)
where $*p*\ P = (\lambda x.\ unstar\ (star\text{-}of\ P \star x))$

definition *starP2* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ star \Rightarrow 'b\ star \Rightarrow bool$
(‹(‹*open-block notation*=‹*prefix starP2*››*∗p2∗* -)› [*80*] *80*)
where $*p2*\ P = (\lambda x\ y.\ unstar\ (star\text{-}of\ P \star x \star y))$

declare *starP-def* [*transfer-unfold*]
declare *starP2-def* [*transfer-unfold*]

lemma *starP-star-n*: $(\ *p*\ P)\ (star\text{-}n\ X) = eventually\ (\lambda n.\ P\ (X\ n))\ \mathcal{U}$
by (*simp only*: *starP-def star-of-def Ifun-star-n unstar-star-n*)

lemma *starP2-star-n*: $(\ *p2*\ P)\ (star\text{-}n\ X)\ (star\text{-}n\ Y) = (eventually\ (\lambda n.\ P\ (X\ n)\ (Y\ n))\ \mathcal{U})$
by (*simp only*: *starP2-def star-of-def Ifun-star-n unstar-star-n*)

lemma *starP-star-of* [*simp*]: $(\ *p*\ P)\ (star\text{-}of\ x) = P\ x$
by *transfer* (*rule refl*)

lemma *starP2-star-of* [*simp*]: (*∗p2∗ P*) (*star-of x*) = *∗p∗ P x*
  by *transfer* (*rule refl*)

2.7 Internal sets

definition *Iset* :: *'a set star* ⇒ *'a star set*
  where *Iset A* = {*x*. (*∗p2∗* (∈)) *x A*}

lemma *Iset-star-n*: (*star-n X* ∈ *Iset* (*star-n A*)) = (*eventually* (λ*n*. *X n* ∈ *A n*) 𝒰)
  by (*simp add*: *Iset-def starP2-star-n*)

Transfer tactic should remove occurrences of *Iset*.

setup ‹*Transfer-Principle.add-const* ***const-name*** ‹*Iset*››

lemma *transfer-mem* [*transfer-intro*]:
  x ≡ *star-n X* ⟹ *a* ≡ *Iset* (*star-n A*) ⟹ *x* ∈ *a* ≡ *eventually* (λ*n*. *X n* ∈ *A n*) 𝒰
  by (*simp only*: *Iset-star-n*)

lemma *transfer-Collect* [*transfer-intro*]:
  (⋀*X*. *p* (*star-n X*) ≡ *eventually* (λ*n*. *P n* (*X n*)) 𝒰) ⟹
    Collect p ≡ *Iset* (*star-n* (λ*n*. *Collect* (*P n*)))
  by (*simp add*: *atomize-eq set-eq-iff all-star-eq Iset-star-n*)

lemma *transfer-set-eq* [*transfer-intro*]:
  a ≡ *Iset* (*star-n A*) ⟹ *b* ≡ *Iset* (*star-n B*) ⟹ *a* = *b* ≡ *eventually* (λ*n*. *A n* = *B n*) 𝒰
  by (*simp only*: *set-eq-iff transfer-all transfer-iff transfer-mem*)

lemma *transfer-ball* [*transfer-intro*]:
  a ≡ *Iset* (*star-n A*) ⟹ (⋀*X*. *p* (*star-n X*) ≡ *eventually* (λ*n*. *P n* (*X n*)) 𝒰) ⟹
    ∀ *x*∈*a*. *p x* ≡ *eventually* (λ*n*. ∀ *x*∈*A n*. *P n x*) 𝒰
  by (*simp only*: *Ball-def transfer-all transfer-imp transfer-mem*)

lemma *transfer-bex* [*transfer-intro*]:
  a ≡ *Iset* (*star-n A*) ⟹ (⋀*X*. *p* (*star-n X*) ≡ *eventually* (λ*n*. *P n* (*X n*)) 𝒰) ⟹
    ∃ *x*∈*a*. *p x* ≡ *eventually* (λ*n*. ∃ *x*∈*A n*. *P n x*) 𝒰
  by (*simp only*: *Bex-def transfer-ex transfer-conj transfer-mem*)

lemma *transfer-Iset* [*transfer-intro*]: *a* ≡ *star-n A* ⟹ *Iset a* ≡ *Iset* (*star-n* (λ*n*. *A n*))
  by *simp*

Nonstandard extensions of sets.

definition *starset* :: *'a set* ⇒ *'a star set*
    (‹(‹*open-block notation*=‹*prefix starset*››*∗s∗* -)› [*80*] *80*)
  where *starset A* = *Iset* (*star-of A*)

declare *starset-def* [*transfer-unfold*]

lemma *starset-mem*: *star-of* $x \in *s*\ A \longleftrightarrow x \in A$
by *transfer* (*rule refl*)

lemma *starset-UNIV*: $*s*\ (UNIV::'a\ set) = (UNIV::'a\ star\ set)$
by (*transfer UNIV-def*) (*rule refl*)

lemma *starset-empty*: $*s*\ \{\} = \{\}$
by (*transfer empty-def*) (*rule refl*)

lemma *starset-insert*: $*s*\ (insert\ x\ A) = insert\ (star\text{-}of\ x)\ (\ *s*\ A)$
by (*transfer insert-def Un-def*) (*rule refl*)

lemma *starset-Un*: $*s*\ (A \cup B) = *s*\ A \cup *s*\ B$
by (*transfer Un-def*) (*rule refl*)

lemma *starset-Int*: $*s*\ (A \cap B) = *s*\ A \cap *s*\ B$
by (*transfer Int-def*) (*rule refl*)

lemma *starset-Compl*: $*s*\ -A = -(\ *s*\ A)$
by (*transfer Compl-eq*) (*rule refl*)

lemma *starset-diff*: $*s*\ (A - B) = *s*\ A - *s*\ B$
by (*transfer set-diff-eq*) (*rule refl*)

lemma *starset-image*: $*s*\ (f\ `\ A) = (\ *f*\ f)\ `\ (\ *s*\ A)$
by (*transfer image-def*) (*rule refl*)

lemma *starset-vimage*: $*s*\ (f\ -`\ A) = (\ *f*\ f)\ -`\ (\ *s*\ A)$
by (*transfer vimage-def*) (*rule refl*)

lemma *starset-subset*: $(\ *s*\ A \subseteq *s*\ B) \longleftrightarrow A \subseteq B$
by (*transfer subset-eq*) (*rule refl*)

lemma *starset-eq*: $(\ *s*\ A = *s*\ B) \longleftrightarrow A = B$
by *transfer* (*rule refl*)

lemmas *starset-simps* [*simp*] =
starset-mem *starset-UNIV*
starset-empty *starset-insert*
starset-Un *starset-Int*
starset-Compl *starset-diff*
starset-image *starset-vimage*
starset-subset *starset-eq*

2.8 Syntactic classes

instantiation *star* :: (*zero*) *zero*

begin
  definition *star-zero-def*: *0* ≡ *star-of 0*
  instance ..
end

instantiation *star* :: (*one*) *one*
begin
  definition *star-one-def*: *1* ≡ *star-of 1*
  instance ..
end

instantiation *star* :: (*plus*) *plus*
begin
  definition *star-add-def*: (+) ≡ *f2* (+)
  instance ..
end

instantiation *star* :: (*times*) *times*
begin
  definition *star-mult-def*: ((*)) ≡ *f2* ((*))
  instance ..
end

instantiation *star* :: (*uminus*) *uminus*
begin
  definition *star-minus-def*: *uminus* ≡ *f* *uminus*
  instance ..
end

instantiation *star* :: (*minus*) *minus*
begin
  definition *star-diff-def*: (−) ≡ *f2* (−)
  instance ..
end

instantiation *star* :: (*abs*) *abs*
begin
  definition *star-abs-def*: *abs* ≡ *f* *abs*
  instance ..
end

instantiation *star* :: (*sgn*) *sgn*
begin
  definition *star-sgn-def*: *sgn* ≡ *f* *sgn*
  instance ..
end

instantiation *star* :: (*divide*) *divide*
begin

definition *star-divide-def*: *divide* ≡ ∗*f2*∗ *divide*
instance ..
end

instantiation *star* :: (*inverse*) *inverse*
begin
definition *star-inverse-def*: *inverse* ≡ ∗*f*∗ *inverse*
instance ..
end

instance *star* :: (*Rings.dvd*) *Rings.dvd* ..

instantiation *star* :: (*modulo*) *modulo*
begin
definition *star-mod-def*: (*mod*) ≡ ∗*f2*∗ (*mod*)
instance ..
end

instantiation *star* :: (*ord*) *ord*
begin
definition *star-le-def*: (≤) ≡ ∗*p2*∗ (≤)
definition *star-less-def*: (<) ≡ ∗*p2*∗ (<)
instance ..
end

lemmas *star-class-defs* [*transfer-unfold*] =
star-zero-def *star-one-def*
star-add-def *star-diff-def* *star-minus-def*
star-mult-def *star-divide-def* *star-inverse-def*
star-le-def *star-less-def* *star-abs-def* *star-sgn-def*
star-mod-def

Class operations preserve standard elements.

lemma *Standard-zero*: *0* ∈ *Standard*
by (*simp add*: *star-zero-def*)

lemma *Standard-one*: *1* ∈ *Standard*
by (*simp add*: *star-one-def*)

lemma *Standard-add*: *x* ∈ *Standard* ⟹ *y* ∈ *Standard* ⟹ *x* + *y* ∈ *Standard*
by (*simp add*: *star-add-def*)

lemma *Standard-diff*: *x* ∈ *Standard* ⟹ *y* ∈ *Standard* ⟹ *x* − *y* ∈ *Standard*
by (*simp add*: *star-diff-def*)

lemma *Standard-minus*: *x* ∈ *Standard* ⟹ − *x* ∈ *Standard*
by (*simp add*: *star-minus-def*)

lemma *Standard-mult*: *x* ∈ *Standard* ⟹ *y* ∈ *Standard* ⟹ *x* ∗ *y* ∈ *Standard*

by (*simp add*: *star-mult-def*)

lemma *Standard-divide*: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow x\ /\ y \in Standard$
by (*simp add*: *star-divide-def*)

lemma *Standard-inverse*: $x \in Standard \Longrightarrow inverse\ x \in Standard$
by (*simp add*: *star-inverse-def*)

lemma *Standard-abs*: $x \in Standard \Longrightarrow |x| \in Standard$
by (*simp add*: *star-abs-def*)

lemma *Standard-mod*: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow x\ mod\ y \in Standard$
by (*simp add*: *star-mod-def*)

lemmas *Standard-simps* [*simp*] =
Standard-zero *Standard-one*
Standard-add *Standard-diff* *Standard-minus*
Standard-mult *Standard-divide* *Standard-inverse*
Standard-abs *Standard-mod*

star-of preserves class operations.

lemma *star-of-add*: *star-of* $(x + y)$ = *star-of* x + *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-diff*: *star-of* $(x - y)$ = *star-of* x − *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-minus*: *star-of* $(-x)$ = − *star-of* x
by *transfer* (*rule refl*)

lemma *star-of-mult*: *star-of* $(x * y)$ = *star-of* x ∗ *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-divide*: *star-of* $(x\ /\ y)$ = *star-of* x / *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-inverse*: *star-of* (*inverse* x) = *inverse* (*star-of* x)
by *transfer* (*rule refl*)

lemma *star-of-mod*: *star-of* (x *mod* y) = *star-of* x *mod* *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-abs*: *star-of* $|x|$ = |*star-of* x|
by *transfer* (*rule refl*)

star-of preserves numerals.

lemma *star-of-zero*: *star-of* *0* = *0*
by *transfer* (*rule refl*)

lemma *star-of-one*: *star-of 1* = *1*
by *transfer* (*rule refl*)

star-of preserves orderings.

lemma *star-of-less*: (*star-of x* < *star-of y*) = (*x* < *y*)
by *transfer* (*rule refl*)

lemma *star-of-le*: (*star-of x* ≤ *star-of y*) = (*x* ≤ *y*)
by *transfer* (*rule refl*)

lemma *star-of-eq*: (*star-of x* = *star-of y*) = (*x* = *y*)
by *transfer* (*rule refl*)

As above, for *0*.

lemmas *star-of-0-less* = *star-of-less* [*of 0*, *simplified star-of-zero*]
lemmas *star-of-0-le* = *star-of-le* [*of 0*, *simplified star-of-zero*]
lemmas *star-of-0-eq* = *star-of-eq* [*of 0*, *simplified star-of-zero*]

lemmas *star-of-less-0* = *star-of-less* [*of - 0*, *simplified star-of-zero*]
lemmas *star-of-le-0* = *star-of-le* [*of - 0*, *simplified star-of-zero*]
lemmas *star-of-eq-0* = *star-of-eq* [*of - 0*, *simplified star-of-zero*]

As above, for *1*.

lemmas *star-of-1-less* = *star-of-less* [*of 1*, *simplified star-of-one*]
lemmas *star-of-1-le* = *star-of-le* [*of 1*, *simplified star-of-one*]
lemmas *star-of-1-eq* = *star-of-eq* [*of 1*, *simplified star-of-one*]

lemmas *star-of-less-1* = *star-of-less* [*of - 1*, *simplified star-of-one*]
lemmas *star-of-le-1* = *star-of-le* [*of - 1*, *simplified star-of-one*]
lemmas *star-of-eq-1* = *star-of-eq* [*of - 1*, *simplified star-of-one*]

lemmas *star-of-simps* [*simp*] =
star-of-add *star-of-diff* *star-of-minus*
star-of-mult *star-of-divide* *star-of-inverse*
star-of-mod *star-of-abs*
star-of-zero *star-of-one*
star-of-less *star-of-le* *star-of-eq*
star-of-0-less *star-of-0-le* *star-of-0-eq*
star-of-less-0 *star-of-le-0* *star-of-eq-0*
star-of-1-less *star-of-1-le* *star-of-1-eq*
star-of-less-1 *star-of-le-1* *star-of-eq-1*

2.9 Ordering and lattice classes

instance *star* :: (*order*) *order*
proof
show $\bigwedge x\ y{::}'a\ star.\ (x < y) = (x \le y \wedge \neg\ y \le x)$
by *transfer* (*rule less-le-not-le*)
show $\bigwedge x{::}'a\ star.\ x \le x$

by *transfer* (*rule order-refl*)
show $\bigwedge x\ y\ z{::}'a\ star.\ [\![x \le y;\ y \le z]\!] \Longrightarrow x \le z$
by *transfer* (*rule order-trans*)
show $\bigwedge x\ y{::}'a\ star.\ [\![x \le y;\ y \le x]\!] \Longrightarrow x = y$
by *transfer* (*rule order-antisym*)
qed

instantiation *star* :: (*semilattice-inf*) *semilattice-inf*
begin
definition *star-inf-def* [*transfer-unfold*]: *inf* ≡ *∗f2∗ inf*
instance by (*standard*; *transfer*) *auto*
end

instantiation *star* :: (*semilattice-sup*) *semilattice-sup*
begin
definition *star-sup-def* [*transfer-unfold*]: *sup* ≡ *∗f2∗ sup*
instance by (*standard*; *transfer*) *auto*
end

instance *star* :: (*lattice*) *lattice* **..**

instance *star* :: (*distrib-lattice*) *distrib-lattice*
by (*standard*; *transfer*) (*auto simp add*: *sup-inf-distrib1*)

lemma *Standard-inf* [*simp*]: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow inf\ x\ y \in Standard$
by (*simp add*: *star-inf-def*)

lemma *Standard-sup* [*simp*]: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow sup\ x\ y \in Standard$
by (*simp add*: *star-sup-def*)

lemma *star-of-inf* [*simp*]: *star-of* (*inf x y*) = *inf* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

lemma *star-of-sup* [*simp*]: *star-of* (*sup x y*) = *sup* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

instance *star* :: (*linorder*) *linorder*
by (*intro-classes*, *transfer*, *rule linorder-linear*)

lemma *star-max-def* [*transfer-unfold*]: *max* = *∗f2∗ max*
unfolding *max-def*
by (*intro ext*, *transfer*, *simp*)

lemma *star-min-def* [*transfer-unfold*]: *min* = *∗f2∗ min*
unfolding *min-def*
by (*intro ext*, *transfer*, *simp*)

lemma *Standard-max* [*simp*]: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow max\ x\ y \in$

Standard
by (*simp add*: *star-max-def*)

lemma *Standard-min* [*simp*]: $x \in$ *Standard* $\Longrightarrow$ $y \in$ *Standard* $\Longrightarrow$ *min x y* $\in$ *Standard*
by (*simp add*: *star-min-def*)

lemma *star-of-max* [*simp*]: *star-of* (*max x y*) = *max* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

lemma *star-of-min* [*simp*]: *star-of* (*min x y*) = *min* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

2.10 Ordered group classes

instance *star* :: (*semigroup-add*) *semigroup-add*
by (*intro-classes*, *transfer*, *rule add.assoc*)

instance *star* :: (*ab-semigroup-add*) *ab-semigroup-add*
by (*intro-classes*, *transfer*, *rule add.commute*)

instance *star* :: (*semigroup-mult*) *semigroup-mult*
by (*intro-classes*, *transfer*, *rule mult.assoc*)

instance *star* :: (*ab-semigroup-mult*) *ab-semigroup-mult*
by (*intro-classes*, *transfer*, *rule mult.commute*)

instance *star* :: (*comm-monoid-add*) *comm-monoid-add*
by (*intro-classes*, *transfer*, *rule comm-monoid-add-class.add-0*)

instance *star* :: (*monoid-mult*) *monoid-mult*
apply *intro-classes*
apply (*transfer*, *rule mult-1-left*)
apply (*transfer*, *rule mult-1-right*)
done

instance *star* :: (*power*) *power* **..**

instance *star* :: (*comm-monoid-mult*) *comm-monoid-mult*
by (*intro-classes*, *transfer*, *rule mult-1*)

instance *star* :: (*cancel-semigroup-add*) *cancel-semigroup-add*
apply *intro-classes*
apply (*transfer*, *erule add-left-imp-eq*)
apply (*transfer*, *erule add-right-imp-eq*)
done

instance *star* :: (*cancel-ab-semigroup-add*) *cancel-ab-semigroup-add*
by *intro-classes* (*transfer*, *simp add*: *diff-diff-eq*)+

instance *star* :: (*cancel-comm-monoid-add*) *cancel-comm-monoid-add* **..**

instance *star* :: (*ab-group-add*) *ab-group-add*
 apply *intro-classes*
  apply (*transfer*, *rule left-minus*)
 apply (*transfer*, *rule diff-conv-add-uminus*)
 done

instance *star* :: (*ordered-ab-semigroup-add*) *ordered-ab-semigroup-add*
 by (*intro-classes*, *transfer*, *rule add-left-mono*)

instance *star* :: (*ordered-cancel-ab-semigroup-add*) *ordered-cancel-ab-semigroup-add*
..

instance *star* :: (*ordered-ab-semigroup-add-imp-le*) *ordered-ab-semigroup-add-imp-le*
 by (*intro-classes*, *transfer*, *rule add-le-imp-le-left*)

instance *star* :: (*ordered-comm-monoid-add*) *ordered-comm-monoid-add* **..**
instance *star* :: (*ordered-ab-semigroup-monoid-add-imp-le*) *ordered-ab-semigroup-monoid-add-imp-le*
..
instance *star* :: (*ordered-cancel-comm-monoid-add*) *ordered-cancel-comm-monoid-add*
..
instance *star* :: (*ordered-ab-group-add*) *ordered-ab-group-add* **..**

instance *star* :: (*ordered-ab-group-add-abs*) *ordered-ab-group-add-abs*
 by *intro-classes* (*transfer*, *simp add*: *abs-ge-self abs-leI abs-triangle-ineq*)+

instance *star* :: (*linordered-cancel-ab-semigroup-add*) *linordered-cancel-ab-semigroup-add*
..

2.11 Ring and field classes

instance *star* :: (*semiring*) *semiring*
 by (*intro-classes*; *transfer*) (*fact distrib-right distrib-left*)+

instance *star* :: (*semiring-0*) *semiring-0*
 by (*intro-classes*; *transfer*) *simp-all*

instance *star* :: (*semiring-0-cancel*) *semiring-0-cancel* **..**

instance *star* :: (*comm-semiring*) *comm-semiring*
 by (*intro-classes*; *transfer*) (*fact distrib-right*)

instance *star* :: (*comm-semiring-0*) *comm-semiring-0* **..**
instance *star* :: (*comm-semiring-0-cancel*) *comm-semiring-0-cancel* **..**

instance *star* :: (*zero-neq-one*) *zero-neq-one*
 by (*intro-classes*; *transfer*) (*fact zero-neq-one*)

instance *star* :: (*semiring-1*) *semiring-1* **..**
instance *star* :: (*comm-semiring-1*) *comm-semiring-1* **..**

declare *dvd-def* [*transfer-refold*]

instance *star* :: (*comm-semiring-1-cancel*) *comm-semiring-1-cancel*
by (*intro-classes*; *transfer*) (*fact right-diff-distrib′*)

instance *star* :: (*semiring-no-zero-divisors*) *semiring-no-zero-divisors*
by (*intro-classes*; *transfer*) (*fact no-zero-divisors*)

instance *star* :: (*semiring-1-no-zero-divisors*) *semiring-1-no-zero-divisors* **..**

instance *star* :: (*semiring-no-zero-divisors-cancel*) *semiring-no-zero-divisors-cancel*
by (*intro-classes*; *transfer*) *simp-all*

instance *star* :: (*semiring-1-cancel*) *semiring-1-cancel* **..**
instance *star* :: (*ring*) *ring* **..**
instance *star* :: (*comm-ring*) *comm-ring* **..**
instance *star* :: (*ring-1*) *ring-1* **..**
instance *star* :: (*comm-ring-1*) *comm-ring-1* **..**
instance *star* :: (*semidom*) *semidom* **..**

instance *star* :: (*semidom-divide*) *semidom-divide*
by (*intro-classes*; *transfer*) *simp-all*

instance *star* :: (*ring-no-zero-divisors*) *ring-no-zero-divisors* **..**
instance *star* :: (*ring-1-no-zero-divisors*) *ring-1-no-zero-divisors* **..**
instance *star* :: (*idom*) *idom* **..**
instance *star* :: (*idom-divide*) *idom-divide* **..**

instance *star* :: (*divide-trivial*) *divide-trivial*
by (*intro-classes*; *transfer*) *simp-all*

instance *star* :: (*division-ring*) *division-ring*
by (*intro-classes*; *transfer*) (*simp-all add*: *divide-inverse*)

instance *star* :: (*field*) *field*
by (*intro-classes*; *transfer*) (*simp-all add*: *divide-inverse*)

instance *star* :: (*ordered-semiring*) *ordered-semiring*
by (*intro-classes*; *transfer*) (*fact mult-left-mono mult-right-mono*)+

instance *star* :: (*ordered-cancel-semiring*) *ordered-cancel-semiring* **..**

instance *star* :: (*linordered-semiring-strict*) *linordered-semiring-strict*
by (*intro-classes*; *transfer*) (*fact mult-strict-left-mono mult-strict-right-mono*)+

instance *star* :: (*ordered-comm-semiring*) *ordered-comm-semiring*
 by (*intro-classes*; *transfer*) (*fact mult-left-mono*)

instance *star* :: (*ordered-cancel-comm-semiring*) *ordered-cancel-comm-semiring* **..**

instance *star* :: (*linordered-comm-semiring-strict*) *linordered-comm-semiring-strict*
 by (*intro-classes*; *transfer*) (*fact mult-strict-left-mono*)

instance *star* :: (*ordered-ring*) *ordered-ring* **..**

instance *star* :: (*ordered-ring-abs*) *ordered-ring-abs*
 by (*intro-classes*; *transfer*) (*fact abs-eq-mult*)

instance *star* :: (*abs-if*) *abs-if*
 by (*intro-classes*; *transfer*) (*fact abs-if*)

instance *star* :: (*linordered-ring-strict*) *linordered-ring-strict* **..**
instance *star* :: (*ordered-comm-ring*) *ordered-comm-ring* **..**

instance *star* :: (*linordered-semidom*) *linordered-semidom*
 by (*intro-classes*; *transfer*) (*fact zero-less-one le-add-diff-inverse2*)+

instance *star* :: (*linordered-idom*) *linordered-idom*
 by (*intro-classes*; *transfer*) (*fact sgn-if*)

instance *star* :: (*linordered-field*) *linordered-field* **..**

instance *star* :: (*algebraic-semidom*) *algebraic-semidom* **..**

instantiation *star* :: (*normalization-semidom*) *normalization-semidom*
begin

definition *unit-factor-star* :: $'a$ *star* $\Rightarrow$ $'a$ *star*
 where [*transfer-unfold*]: *unit-factor-star* = ∗*f*∗ *unit-factor*

definition *normalize-star* :: $'a$ *star* $\Rightarrow$ $'a$ *star*
 where [*transfer-unfold*]: *normalize-star* = ∗*f*∗ *normalize*

instance
 by *standard* (*transfer*; *simp add*: *is-unit-unit-factor unit-factor-mult*)+

end

instance *star* :: (*semidom-modulo*) *semidom-modulo*
 by *standard* (*transfer*; *simp*)

2.12 Power

lemma *star-power-def* [*transfer-unfold*]: $(\hat{~}) \equiv \lambda x\ n.\ (\ *f*\ (\lambda x.\ x\ \hat{~}\ n))\ x$

proof (*rule eq-reflection*, *rule ext*, *rule ext*)
  show $x \mathbin{\hat{}} n = (\ *f*\ (\lambda x.\ x \mathbin{\hat{}} n))\ x$ **for** *n* :: *nat* **and** *x* :: *'a star*
  proof (*induct n arbitrary*: *x*)
    case *0*
    have $\bigwedge x{::}'a\ star.\ (\ *f*\ (\lambda x.\ 1))\ x = 1$
      by *transfer simp*
    then show *?case* **by** *simp*
  next
    case (*Suc n*)
    have $\bigwedge x{::}'a\ star.\ x * (\ *f*\ (\lambda x{::}'a.\ x \mathbin{\hat{}} n))\ x = (\ *f*\ (\lambda x{::}'a.\ x * x \mathbin{\hat{}} n))\ x$
      by *transfer simp*
    with *Suc* **show** *?case* **by** *simp*
  qed
qed

lemma *Standard-power* [*simp*]: $x \in Standard \Longrightarrow x \mathbin{\hat{}} n \in Standard$
  by (*simp add*: *star-power-def*)

lemma *star-of-power* [*simp*]: *star-of* $(x \mathbin{\hat{}} n) =$ *star-of* $x \mathbin{\hat{}} n$
  by *transfer* (*rule refl*)

2.13 Number classes

instance *star* :: (*numeral*) *numeral* ..

lemma *star-numeral-def* [*transfer-unfold*]: *numeral k* = *star-of* (*numeral k*)
  by (*induct k*) (*simp-all only*: *numeral.simps star-of-one star-of-add*)

lemma *Standard-numeral* [*simp*]: *numeral k* ∈ *Standard*
  by (*simp add*: *star-numeral-def*)

lemma *star-of-numeral* [*simp*]: *star-of* (*numeral k*) = *numeral k*
  by *transfer* (*rule refl*)

lemma *star-of-nat-def* [*transfer-unfold*]: *of-nat n* = *star-of* (*of-nat n*)
  by (*induct n*) *simp-all*

lemmas *star-of-compare-numeral* [*simp*] =
  star-of-less [*of numeral k*, *simplified star-of-numeral*]
  star-of-le [*of numeral k*, *simplified star-of-numeral*]
  star-of-eq [*of numeral k*, *simplified star-of-numeral*]
  star-of-less [*of - numeral k*, *simplified star-of-numeral*]
  star-of-le [*of - numeral k*, *simplified star-of-numeral*]
  star-of-eq [*of - numeral k*, *simplified star-of-numeral*]
  star-of-less [*of − numeral k*, *simplified star-of-numeral*]
  star-of-le [*of − numeral k*, *simplified star-of-numeral*]
  star-of-eq [*of − numeral k*, *simplified star-of-numeral*]
  star-of-less [*of - − numeral k*, *simplified star-of-numeral*]
  star-of-le [*of - − numeral k*, *simplified star-of-numeral*]

star-of-eq [*of - − numeral k*, *simplified star-of-numeral*] **for** *k*

lemma *Standard-of-nat* [*simp*]: *of-nat n* ∈ *Standard*
by (*simp add*: *star-of-nat-def*)

lemma *star-of-of-nat* [*simp*]: *star-of* (*of-nat n*) = *of-nat n*
by *transfer* (*rule refl*)

lemma *star-of-int-def* [*transfer-unfold*]: *of-int z* = *star-of* (*of-int z*)
by (*rule int-diff-cases* [*of z*]) *simp*

lemma *Standard-of-int* [*simp*]: *of-int z* ∈ *Standard*
by (*simp add*: *star-of-int-def*)

lemma *star-of-of-int* [*simp*]: *star-of* (*of-int z*) = *of-int z*
by *transfer* (*rule refl*)

instance *star* :: (*semiring-char-0*) *semiring-char-0*
proof
have *inj* (*star-of* :: *'a* ⇒ *'a star*)
by (*rule injI*) *simp*
then have *inj* (*star-of* ∘ *of-nat* :: *nat* ⇒ *'a star*)
using *inj-of-nat* **by** (*rule inj-compose*)
then show *inj* (*of-nat* :: *nat* ⇒ *'a star*)
by (*simp add*: *comp-def*)
qed

instance *star* :: (*ring-char-0*) *ring-char-0* **..**

2.14 Finite class

lemma *starset-finite*: *finite A* ⟹ *∗s∗ A* = *star-of ‘ A*
by (*erule finite-induct*) *simp-all*

instance *star* :: (*finite*) *finite*
proof *intro-classes*
show *finite* (*UNIV*::*'a star set*)
by (*metis starset-UNIV finite finite-imageI starset-finite*)
qed

end

3 Hypernatural numbers

theory *HyperNat*
imports *StarDef*
begin

type-synonym *hypnat* = *nat star*

abbreviation *hypnat-of-nat* :: *nat* ⇒ *nat star*
  where *hypnat-of-nat* ≡ *star-of*

definition *hSuc* :: *hypnat* ⇒ *hypnat*
  where *hSuc-def* [*transfer-unfold*]: *hSuc* = *∗f∗ Suc*

3.1 Properties Transferred from Naturals

lemma *hSuc-not-zero* [*iff*]: $\bigwedge m.\ hSuc\ m \neq 0$
  by *transfer* (*rule Suc-not-Zero*)

lemma *zero-not-hSuc* [*iff*]: $\bigwedge m.\ 0 \neq hSuc\ m$
  by *transfer* (*rule Zero-not-Suc*)

lemma *hSuc-hSuc-eq* [*iff*]: $\bigwedge m\ n.\ hSuc\ m = hSuc\ n \longleftrightarrow m = n$
  by *transfer* (*rule nat.inject*)

lemma *zero-less-hSuc* [*iff*]: $\bigwedge n.\ 0 < hSuc\ n$
  by *transfer* (*rule zero-less-Suc*)

lemma *hypnat-minus-zero* [*simp*]: $\bigwedge z{::}hypnat.\ z - z = 0$
  by *transfer* (*rule diff-self-eq-0*)

lemma *hypnat-diff-0-eq-0* [*simp*]: $\bigwedge n{::}hypnat.\ 0 - n = 0$
  by *transfer* (*rule diff-0-eq-0*)

lemma *hypnat-add-is-0* [*iff*]: $\bigwedge m\ n{::}hypnat.\ m + n = 0 \longleftrightarrow m = 0 \land n = 0$
  by *transfer* (*rule add-is-0*)

lemma *hypnat-diff-diff-left*: $\bigwedge i\ j\ k{::}hypnat.\ i - j - k = i - (j + k)$
  by *transfer* (*rule diff-diff-left*)

lemma *hypnat-diff-commute*: $\bigwedge i\ j\ k{::}hypnat.\ i - j - k = i - k - j$
  by *transfer* (*rule diff-commute*)

lemma *hypnat-diff-add-inverse* [*simp*]: $\bigwedge m\ n{::}hypnat.\ n + m - n = m$
  by *transfer* (*rule diff-add-inverse*)

lemma *hypnat-diff-add-inverse2* [*simp*]: $\bigwedge m\ n{::}hypnat.\ m + n - n = m$
  by *transfer* (*rule diff-add-inverse2*)

lemma *hypnat-diff-cancel* [*simp*]: $\bigwedge k\ m\ n{::}hypnat.\ (k + m) - (k + n) = m - n$
  by *transfer* (*rule diff-cancel*)

lemma *hypnat-diff-cancel2* [*simp*]: $\bigwedge k\ m\ n{::}hypnat.\ (m + k) - (n + k) = m - n$
  by *transfer* (*rule diff-cancel2*)

lemma *hypnat-diff-add-0* [*simp*]: $\bigwedge m\ n{::}hypnat.\ n - (n + m) = 0$

by *transfer* (*rule diff-add-0*)

lemma *hypnat-diff-mult-distrib*: $\bigwedge k\ m\ n::hypnat.\ (m - n) * k = (m * k) - (n * k)$
by *transfer* (*rule diff-mult-distrib*)

lemma *hypnat-diff-mult-distrib2*: $\bigwedge k\ m\ n::hypnat.\ k * (m - n) = (k * m) - (k * n)$
by *transfer* (*rule diff-mult-distrib2*)

lemma *hypnat-le-zero-cancel* [*iff*]: $\bigwedge n::hypnat.\ n \le 0 \longleftrightarrow n = 0$
by *transfer* (*rule le-0-eq*)

lemma *hypnat-mult-is-0* [*simp*]: $\bigwedge m\ n::hypnat.\ m * n = 0 \longleftrightarrow m = 0 \vee n = 0$
by *transfer* (*rule mult-is-0*)

lemma *hypnat-diff-is-0-eq* [*simp*]: $\bigwedge m\ n::hypnat.\ m - n = 0 \longleftrightarrow m \le n$
by *transfer* (*rule diff-is-0-eq*)

lemma *hypnat-not-less0* [*iff*]: $\bigwedge n::hypnat.\ \neg\ n < 0$
by *transfer* (*rule not-less0*)

lemma *hypnat-less-one* [*iff*]: $\bigwedge n::hypnat.\ n < 1 \longleftrightarrow n = 0$
by *transfer* (*rule less-one*)

lemma *hypnat-add-diff-inverse*: $\bigwedge m\ n::hypnat.\ \neg\ m < n \Longrightarrow n + (m - n) = m$
by *transfer* (*rule add-diff-inverse*)

lemma *hypnat-le-add-diff-inverse* [*simp*]: $\bigwedge m\ n::hypnat.\ n \le m \Longrightarrow n + (m - n) = m$
by *transfer* (*rule le-add-diff-inverse*)

lemma *hypnat-le-add-diff-inverse2* [*simp*]: $\bigwedge m\ n::hypnat.\ n \le m \Longrightarrow (m - n) + n = m$
by *transfer* (*rule le-add-diff-inverse2*)

declare *hypnat-le-add-diff-inverse2* [*OF order-less-imp-le*]

lemma *hypnat-le0* [*iff*]: $\bigwedge n::hypnat.\ 0 \le n$
by *transfer* (*rule le0*)

lemma *hypnat-le-add1* [*simp*]: $\bigwedge x\ n::hypnat.\ x \le x + n$
by *transfer* (*rule le-add1*)

lemma *hypnat-add-self-le* [*simp*]: $\bigwedge x\ n::hypnat.\ x \le n + x$
by *transfer* (*rule le-add2*)

lemma *hypnat-add-one-self-less* [*simp*]: $x < x + 1$ **for** $x :: hypnat$
by (*fact less-add-one*)

lemma *hypnat-neq0-conv* [*iff*]: $\bigwedge n$::*hypnat*. $n \neq 0 \longleftrightarrow 0 < n$
by *transfer* (*rule neq0-conv*)

lemma *hypnat-gt-zero-iff*: $0 < n \longleftrightarrow 1 \leq n$ **for** n :: *hypnat*
by (*auto simp add*: *linorder-not-less* [*symmetric*])

lemma *hypnat-gt-zero-iff2*: $0 < n \longleftrightarrow (\exists m.\ n = m + 1)$ **for** n :: *hypnat*
by (*auto intro*!: *add-nonneg-pos exI*[*of - n − 1*] *simp*: *hypnat-gt-zero-iff*)

lemma *hypnat-add-self-not-less*: $\neg\ x + y < x$ **for** $x\ y$ :: *hypnat*
by (*simp add*: *linorder-not-le* [*symmetric*] *add.commute* [*of x*])

lemma *hypnat-diff-split*: $P\ (a - b) \longleftrightarrow (a < b \longrightarrow P\ 0) \wedge (\forall d.\ a = b + d \longrightarrow P\ d)$
for $a\ b$ :: *hypnat*
— elimination of − on *hypnat*
proof (*cases* $a < b$ *rule*: *case-split*)
case *True*
then show *?thesis*
by (*auto simp add*: *hypnat-add-self-not-less order-less-imp-le hypnat-diff-is-0-eq* [*THEN iffD2*])
next
case *False*
then show *?thesis*
by (*auto simp add*: *linorder-not-less dest*: *order-le-less-trans*)
qed

3.2 Properties of the set of embedded natural numbers

lemma *of-nat-eq-star-of* [*simp*]: *of-nat* = *star-of*
proof
show *of-nat n* = *star-of n* **for** *n*
by *transfer simp*
qed

lemma *Nats-eq-Standard*: (*Nats* :: *nat star set*) = *Standard*
by (*auto simp*: *Nats-def Standard-def*)

lemma *hypnat-of-nat-mem-Nats* [*simp*]: *hypnat-of-nat* $n \in$ *Nats*
by (*simp add*: *Nats-eq-Standard*)

lemma *hypnat-of-nat-one* [*simp*]: *hypnat-of-nat* (*Suc* 0) = 1
by *transfer simp*

lemma *hypnat-of-nat-Suc* [*simp*]: *hypnat-of-nat* (*Suc n*) = *hypnat-of-nat* $n + 1$
by *transfer simp*

lemma *of-nat-eq-add*:

fixes *d*::*hypnat*
shows *of-nat m* = *of-nat n* + *d* $\Longrightarrow$ *d* $\in$ *range of-nat*
proof (*induct n arbitrary*: *d*)
case (*Suc n*)
then show *?case*
by (*metis Nats-def Nats-eq-Standard Standard-simps*(*4*) *hypnat-diff-add-inverse of-nat-in-Nats*)
qed *auto*

lemma *Nats-diff* [*simp*]: *a* $\in$ *Nats* $\Longrightarrow$ *b* $\in$ *Nats* $\Longrightarrow$ *a* − *b* $\in$ *Nats* **for** *a b* :: *hypnat*
by (*simp add*: *Nats-eq-Standard*)

3.3 Infinite Hypernatural Numbers – *HNatInfinite*

The set of infinite hypernatural numbers.

definition *HNatInfinite* :: *hypnat set*
where *HNatInfinite* = {*n*. *n* $\notin$ *Nats*}

lemma *Nats-not-HNatInfinite-iff*: *x* $\in$ *Nats* $\longleftrightarrow$ *x* $\notin$ *HNatInfinite*
by (*simp add*: *HNatInfinite-def*)

lemma *HNatInfinite-not-Nats-iff*: *x* $\in$ *HNatInfinite* $\longleftrightarrow$ *x* $\notin$ *Nats*
by (*simp add*: *HNatInfinite-def*)

lemma *star-of-neq-HNatInfinite*: *N* $\in$ *HNatInfinite* $\Longrightarrow$ *star-of n* $\neq$ *N*
by (*auto simp add*: *HNatInfinite-def Nats-eq-Standard*)

lemma *star-of-Suc-lessI*: $\bigwedge$*N*. *star-of n* < *N* $\Longrightarrow$ *star-of* (*Suc n*) $\neq$ *N* $\Longrightarrow$ *star-of* (*Suc n*) < *N*
by *transfer* (*rule Suc-lessI*)

lemma *star-of-less-HNatInfinite*:
assumes *N*: *N* $\in$ *HNatInfinite*
shows *star-of n* < *N*
proof (*induct n*)
case *0*
from *N* **have** *star-of 0* $\neq$ *N*
by (*rule star-of-neq-HNatInfinite*)
then show *?case* **by** *simp*
next
case (*Suc n*)
from *N* **have** *star-of* (*Suc n*) $\neq$ *N*
by (*rule star-of-neq-HNatInfinite*)
with *Suc* **show** *?case*
by (*rule star-of-Suc-lessI*)
qed

lemma *star-of-le-HNatInfinite*: *N* $\in$ *HNatInfinite* $\Longrightarrow$ *star-of n* $\leq$ *N*

by (*rule star-of-less-HNatInfinite* [*THEN order-less-imp-le*])

3.3.1 Closure Rules

lemma *Nats-less-HNatInfinite*: $x \in Nats \Longrightarrow y \in HNatInfinite \Longrightarrow x < y$
by (*auto simp add*: *Nats-def star-of-less-HNatInfinite*)

lemma *Nats-le-HNatInfinite*: $x \in Nats \Longrightarrow y \in HNatInfinite \Longrightarrow x \le y$
by (*rule Nats-less-HNatInfinite* [*THEN order-less-imp-le*])

lemma *zero-less-HNatInfinite*: $x \in HNatInfinite \Longrightarrow 0 < x$
by (*simp add*: *Nats-less-HNatInfinite*)

lemma *one-less-HNatInfinite*: $x \in HNatInfinite \Longrightarrow 1 < x$
by (*simp add*: *Nats-less-HNatInfinite*)

lemma *one-le-HNatInfinite*: $x \in HNatInfinite \Longrightarrow 1 \le x$
by (*simp add*: *Nats-le-HNatInfinite*)

lemma *zero-not-mem-HNatInfinite* [*simp*]: $0 \notin HNatInfinite$
by (*simp add*: *HNatInfinite-def*)

lemma *Nats-downward-closed*: $x \in Nats \Longrightarrow y \le x \Longrightarrow y \in Nats$ **for** $x\ y$ :: *hypnat*
using *HNatInfinite-not-Nats-iff Nats-le-HNatInfinite* **by** *fastforce*

lemma *HNatInfinite-upward-closed*: $x \in HNatInfinite \Longrightarrow x \le y \Longrightarrow y \in HNatInfinite$
using *HNatInfinite-not-Nats-iff Nats-downward-closed* **by** *blast*

lemma *HNatInfinite-add*: $x \in HNatInfinite \Longrightarrow x + y \in HNatInfinite$
using *HNatInfinite-upward-closed hypnat-le-add1* **by** *blast*

lemma *HNatInfinite-diff*: $[\![x \in HNatInfinite;\ y \in Nats]\!] \Longrightarrow x - y \in HNatInfinite$
by (*metis HNatInfinite-not-Nats-iff Nats-add Nats-le-HNatInfinite le-add-diff-inverse*)

lemma *HNatInfinite-is-Suc*: $x \in HNatInfinite \Longrightarrow \exists\, y.\ x = y + 1$ **for** x :: *hypnat*
using *hypnat-gt-zero-iff2 zero-less-HNatInfinite* **by** *blast*

3.4 Existence of an infinite hypernatural number

ω is in fact an infinite hypernatural number = $[<1,\ 2,\ 3,\ \ldots>]$

definition *whn* :: *hypnat*
where *hypnat-omega-def*: $whn = star\text{-}n\ (\lambda n{::}nat.\ n)$

lemma *hypnat-of-nat-neq-whn*: $hypnat\text{-}of\text{-}nat\ n \ne whn$
by (*simp add*: *FreeUltrafilterNat.singleton′ hypnat-omega-def star-of-def star-n-eq-iff*)

lemma *whn-neq-hypnat-of-nat*: $whn \ne hypnat\text{-}of\text{-}nat\ n$
by (*simp add*: *FreeUltrafilterNat.singleton hypnat-omega-def star-of-def star-n-eq-iff*)

lemma *whn-not-Nats* [*simp*]: *whn* $\notin$ *Nats*
by (*simp add*: *Nats-def image-def whn-neq-hypnat-of-nat*)

lemma *HNatInfinite-whn* [*simp*]: *whn* $\in$ *HNatInfinite*
by (*simp add*: *HNatInfinite-def*)

lemma *lemma-unbounded-set* [*simp*]: *eventually* (λn::*nat*. $m < n$) $\mathcal{U}$
by (*rule filter-leD*[*OF FreeUltrafilterNat.le-cofinite*])
(*auto simp add*: *cofinite-eq-sequentially eventually-at-top-dense*)

lemma *hypnat-of-nat-eq*: *hypnat-of-nat* m = *star-n* (λn::*nat*. m)
by (*simp add*: *star-of-def*)

lemma *SHNat-eq*: *Nats* = {n. $\exists N$. n = *hypnat-of-nat* N}
by (*simp add*: *Nats-def image-def*)

lemma *Nats-less-whn*: $n \in$ *Nats* $\Longrightarrow$ $n <$ *whn*
by (*simp add*: *Nats-less-HNatInfinite*)

lemma *Nats-le-whn*: $n \in$ *Nats* $\Longrightarrow$ $n \leq$ *whn*
by (*simp add*: *Nats-le-HNatInfinite*)

lemma *hypnat-of-nat-less-whn* [*simp*]: *hypnat-of-nat* $n <$ *whn*
by (*simp add*: *Nats-less-whn*)

lemma *hypnat-of-nat-le-whn* [*simp*]: *hypnat-of-nat* $n \leq$ *whn*
by (*simp add*: *Nats-le-whn*)

lemma *hypnat-zero-less-hypnat-omega* [*simp*]: *0* $<$ *whn*
by (*simp add*: *Nats-less-whn*)

lemma *hypnat-one-less-hypnat-omega* [*simp*]: *1* $<$ *whn*
by (*simp add*: *Nats-less-whn*)

3.4.1 Alternative characterization of the set of infinite hypernaturals

HNatInfinite = {N. $\forall n \in \mathbb{N}$. $n < N$}

unused, but possibly interesting

lemma *HNatInfinite-FreeUltrafilterNat-eventually*:
assumes $\bigwedge k$::*nat*. *eventually* (λn. $f\ n \neq k$) $\mathcal{U}$
shows *eventually* (λn. $m < f\ n$) $\mathcal{U}$
proof (*induct m*)
case *0*
then show *?case*
using *assms eventually-mono* **by** *fastforce*
next

case (*Suc m*)
then show *?case*
using *assms* [*of Suc m*] *eventually-elim2* **by** *fastforce*
qed

lemma *HNatInfinite-iff*: *HNatInfinite* = {*N*. ∀ *n* ∈ *Nats*. *n* < *N*}
using *HNatInfinite-def Nats-less-HNatInfinite* **by** *auto*

3.4.2 Alternative Characterization of *HNatInfinite* using Free Ultrafilter

lemma *HNatInfinite-FreeUltrafilterNat*:
star-n X ∈ *HNatInfinite* ⟹ ∀ *u*. *eventually* (λ*n*. *u* < *X n*) 𝒰
by (*metis* (*full-types*) *starP2-star-of starP-star-n star-less-def star-of-less-HNatInfinite*)

lemma *FreeUltrafilterNat-HNatInfinite*:
∀ *u*. *eventually* (λ*n*. *u* < *X n*) 𝒰 ⟹ *star-n X* ∈ *HNatInfinite*
by (*auto simp add*: *star-less-def starP2-star-n HNatInfinite-iff SHNat-eq hypnat-of-nat-eq*)

lemma *HNatInfinite-FreeUltrafilterNat-iff*:
(*star-n X* ∈ *HNatInfinite*) = (∀ *u*. *eventually* (λ*n*. *u* < *X n*) 𝒰)
by (*rule iffI* [*OF HNatInfinite-FreeUltrafilterNat FreeUltrafilterNat-HNatInfinite*])

3.5 Embedding of the Hypernaturals into other types

definition *of-hypnat* :: *hypnat* ⇒ *'a*::*semiring-1-cancel star*
where *of-hypnat-def* [*transfer-unfold*]: *of-hypnat* = **f* of-nat*

lemma *of-hypnat-0* [*simp*]: *of-hypnat 0* = *0*
by *transfer* (*rule of-nat-0*)

lemma *of-hypnat-1* [*simp*]: *of-hypnat 1* = *1*
by *transfer* (*rule of-nat-1*)

lemma *of-hypnat-hSuc*: ⋀*m*. *of-hypnat* (*hSuc m*) = *1* + *of-hypnat m*
by *transfer* (*rule of-nat-Suc*)

lemma *of-hypnat-add* [*simp*]: ⋀*m n*. *of-hypnat* (*m* + *n*) = *of-hypnat m* + *of-hypnat n*
by *transfer* (*rule of-nat-add*)

lemma *of-hypnat-mult* [*simp*]: ⋀*m n*. *of-hypnat* (*m* ∗ *n*) = *of-hypnat m* ∗ *of-hypnat n*
by *transfer* (*rule of-nat-mult*)

lemma *of-hypnat-less-iff* [*simp*]:
⋀*m n*. *of-hypnat m* < (*of-hypnat n*::*'a*::*linordered-semidom star*) ⟷ *m* < *n*
by *transfer* (*rule of-nat-less-iff*)

lemma *of-hypnat-0-less-iff* [*simp*]:
$\bigwedge n.\ 0 < (\textit{of-hypnat}\ n::'a::\textit{linordered-semidom star}) \longleftrightarrow 0 < n$
by *transfer* (*rule of-nat-0-less-iff*)

lemma *of-hypnat-less-0-iff* [*simp*]: $\bigwedge m.\ \neg\ (\textit{of-hypnat}\ m::'a::\textit{linordered-semidom star}) < 0$
by *transfer* (*rule of-nat-less-0-iff*)

lemma *of-hypnat-le-iff* [*simp*]:
$\bigwedge m\ n.\ \textit{of-hypnat}\ m \le (\textit{of-hypnat}\ n::'a::\textit{linordered-semidom star}) \longleftrightarrow m \le n$
by *transfer* (*rule of-nat-le-iff*)

lemma *of-hypnat-0-le-iff* [*simp*]: $\bigwedge n.\ 0 \le (\textit{of-hypnat}\ n::'a::\textit{linordered-semidom star})$
by *transfer* (*rule of-nat-0-le-iff*)

lemma *of-hypnat-le-0-iff* [*simp*]: $\bigwedge m.\ (\textit{of-hypnat}\ m::'a::\textit{linordered-semidom star}) \le 0 \longleftrightarrow m = 0$
by *transfer* (*rule of-nat-le-0-iff*)

lemma *of-hypnat-eq-iff* [*simp*]:
$\bigwedge m\ n.\ \textit{of-hypnat}\ m = (\textit{of-hypnat}\ n::'a::\textit{linordered-semidom star}) \longleftrightarrow m = n$
by *transfer* (*rule of-nat-eq-iff*)

lemma *of-hypnat-eq-0-iff* [*simp*]: $\bigwedge m.\ (\textit{of-hypnat}\ m::'a::\textit{linordered-semidom star}) = 0 \longleftrightarrow m = 0$
by *transfer* (*rule of-nat-eq-0-iff*)

lemma *HNatInfinite-of-hypnat-gt-zero*:
$N \in \textit{HNatInfinite} \Longrightarrow (0::'a::\textit{linordered-semidom star}) < \textit{of-hypnat}\ N$
by (*rule ccontr*) (*simp add*: *linorder-not-less*)

end

4 Construction of Hyperreals Using Ultrafilters

theory *HyperDef*
imports *Complex-Main HyperNat*
begin

type-synonym *hypreal* = *real star*

abbreviation *hypreal-of-real* :: *real* ⇒ *real star*
where *hypreal-of-real* ≡ *star-of*

abbreviation *hypreal-of-hypnat* :: *hypnat* ⇒ *hypreal*
where *hypreal-of-hypnat* ≡ *of-hypnat*

definition *omega* :: *hypreal* (‹ω›)

where $\omega = star\text{-}n\ (\lambda n.\ real\ (Suc\ n))$
— an infinite number = [<*1*, *2*, *3*, ...>]

definition *epsilon* :: *hypreal* (‹ε›)
where $\varepsilon = star\text{-}n\ (\lambda n.\ inverse\ (real\ (Suc\ n)))$
— an infinitesimal number = [<*1*, *1/2*, *1/3*, ...>]

4.1 Real vector class instances

instantiation *star* :: (*scaleR*) *scaleR*
begin
definition *star-scaleR-def* [*transfer-unfold*]: $scaleR\ r \equiv *f*\ (scaleR\ r)$
instance **..**
end

lemma *Standard-scaleR* [*simp*]: $x \in Standard \Longrightarrow scaleR\ r\ x \in Standard$
by (*simp add*: *star-scaleR-def*)

lemma *star-of-scaleR* [*simp*]: $star\text{-}of\ (scaleR\ r\ x) = scaleR\ r\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

instance *star* :: (*real-vector*) *real-vector*
proof
fix *a b* :: *real*
show $\bigwedge x\ y::'a\ star.\ scaleR\ a\ (x + y) = scaleR\ a\ x + scaleR\ a\ y$
by *transfer* (*rule scaleR-right-distrib*)
show $\bigwedge x::'a\ star.\ scaleR\ (a + b)\ x = scaleR\ a\ x + scaleR\ b\ x$
by *transfer* (*rule scaleR-left-distrib*)
show $\bigwedge x::'a\ star.\ scaleR\ a\ (scaleR\ b\ x) = scaleR\ (a * b)\ x$
by *transfer* (*rule scaleR-scaleR*)
show $\bigwedge x::'a\ star.\ scaleR\ 1\ x = x$
by *transfer* (*rule scaleR-one*)
qed

instance *star* :: (*real-algebra*) *real-algebra*
proof
fix *a* :: *real*
show $\bigwedge x\ y::'a\ star.\ scaleR\ a\ x * y = scaleR\ a\ (x * y)$
by *transfer* (*rule mult-scaleR-left*)
show $\bigwedge x\ y::'a\ star.\ x * scaleR\ a\ y = scaleR\ a\ (x * y)$
by *transfer* (*rule mult-scaleR-right*)
qed

instance *star* :: (*real-algebra-1*) *real-algebra-1* **..**

instance *star* :: (*real-div-algebra*) *real-div-algebra* **..**

instance *star* :: (*field-char-0*) *field-char-0* **..**

instance *star* :: (*real-field*) *real-field* **..**

lemma *star-of-real-def* [*transfer-unfold*]: *of-real r* = *star-of* (*of-real r*)
 by (*unfold of-real-def*, *transfer*, *rule refl*)

lemma *Standard-of-real* [*simp*]: *of-real r* ∈ *Standard*
 by (*simp add*: *star-of-real-def*)

lemma *star-of-of-real* [*simp*]: *star-of* (*of-real r*) = *of-real r*
 by *transfer* (*rule refl*)

lemma *of-real-eq-star-of* [*simp*]: *of-real* = *star-of*
proof
 show *of-real r* = *star-of r* **for** *r* :: *real*
  by *transfer simp*
qed

lemma *Reals-eq-Standard*: (ℝ :: *hypreal set*) = *Standard*
 by (*simp add*: *Reals-def Standard-def*)

4.2 Injection from *hypreal*

definition *of-hypreal* :: *hypreal* ⇒ *'a*::*real-algebra-1 star*
 where [*transfer-unfold*]: *of-hypreal* = ∗*f*∗ *of-real*

lemma *Standard-of-hypreal* [*simp*]: *r* ∈ *Standard* ⟹ *of-hypreal r* ∈ *Standard*
 by (*simp add*: *of-hypreal-def*)

lemma *of-hypreal-0* [*simp*]: *of-hypreal 0* = *0*
 by *transfer* (*rule of-real-0*)

lemma *of-hypreal-1* [*simp*]: *of-hypreal 1* = *1*
 by *transfer* (*rule of-real-1*)

lemma *of-hypreal-add* [*simp*]: ⋀*x y*. *of-hypreal* (*x* + *y*) = *of-hypreal x* + *of-hypreal y*
 by *transfer* (*rule of-real-add*)

lemma *of-hypreal-minus* [*simp*]: ⋀*x*. *of-hypreal* (− *x*) = − *of-hypreal x*
 by *transfer* (*rule of-real-minus*)

lemma *of-hypreal-diff* [*simp*]: ⋀*x y*. *of-hypreal* (*x* − *y*) = *of-hypreal x* − *of-hypreal y*
 by *transfer* (*rule of-real-diff*)

lemma *of-hypreal-mult* [*simp*]: ⋀*x y*. *of-hypreal* (*x* ∗ *y*) = *of-hypreal x* ∗ *of-hypreal y*
 by *transfer* (*rule of-real-mult*)

lemma *of-hypreal-inverse* [*simp*]:
$\bigwedge x.$ *of-hypreal* (*inverse x*) =
inverse (*of-hypreal x* :: *'a*::{*real-div-algebra*, *division-ring*} *star*)
by *transfer* (*rule of-real-inverse*)

lemma *of-hypreal-divide* [*simp*]:
$\bigwedge x\ y.$ *of-hypreal* (*x* / *y*) =
(*of-hypreal x* / *of-hypreal y* :: *'a*::{*real-field*, *field*} *star*)
by *transfer* (*rule of-real-divide*)

lemma *of-hypreal-eq-iff* [*simp*]: $\bigwedge x\ y.$ (*of-hypreal x* = *of-hypreal y*) = (*x* = *y*)
by *transfer* (*rule of-real-eq-iff*)

lemma *of-hypreal-eq-0-iff* [*simp*]: $\bigwedge x.$ (*of-hypreal x* = *0*) = (*x* = *0*)
by *transfer* (*rule of-real-eq-0-iff*)

4.3 Properties of *starrel*

lemma *lemma-starrel-refl* [*simp*]: *x* ∈ *starrel* `` {*x*}
by (*simp add*: *starrel-def*)

lemma *starrel-in-hypreal* [*simp*]: *starrel*``{*x*}∈*star*
by (*simp add*: *star-def starrel-def quotient-def*, *blast*)

declare *Abs-star-inject* [*simp*] *Abs-star-inverse* [*simp*]
declare *equiv-starrel* [*THEN eq-equiv-class-iff*, *simp*]

4.4 *hypreal-of-real*: the Injection from *real* to *hypreal*

lemma *inj-star-of*: *inj star-of*
by (*rule inj-onI*) *simp*

lemma *mem-Rep-star-iff*: *X* ∈ *Rep-star x* ⟷ *x* = *star-n X*
by (*cases x*) (*simp add*: *star-n-def*)

lemma *Rep-star-star-n-iff* [*simp*]: *X* ∈ *Rep-star* (*star-n Y*) ⟷ *eventually* (*λn.* *Y n* = *X n*) $\mathcal{U}$
by (*simp add*: *star-n-def*)

lemma *Rep-star-star-n*: *X* ∈ *Rep-star* (*star-n X*)
by *simp*

4.5 Properties of *star-n*

lemma *star-n-add*: *star-n X* + *star-n Y* = *star-n* (*λn.* *X n* + *Y n*)
by (*simp only*: *star-add-def starfun2-star-n*)

lemma *star-n-minus*: − *star-n X* = *star-n* (*λn.* −(*X n*))
by (*simp only*: *star-minus-def starfun-star-n*)

lemma *star-n-diff*: *star-n X* − *star-n Y* = *star-n* (λ*n*. *X n* − *Y n*)
  by (*simp only*: *star-diff-def starfun2-star-n*)

lemma *star-n-mult*: *star-n X* ∗ *star-n Y* = *star-n* (λ*n*. *X n* ∗ *Y n*)
  by (*simp only*: *star-mult-def starfun2-star-n*)

lemma *star-n-inverse*: *inverse* (*star-n X*) = *star-n* (λ*n*. *inverse* (*X n*))
  by (*simp only*: *star-inverse-def starfun-star-n*)

lemma *star-n-le*: *star-n X* ≤ *star-n Y* = *eventually* (λ*n*. *X n* ≤ *Y n*) $\mathcal{U}$
  by (*simp only*: *star-le-def starP2-star-n*)

lemma *star-n-less*: *star-n X* < *star-n Y* = *eventually* (λ*n*. *X n* < *Y n*) $\mathcal{U}$
  by (*simp only*: *star-less-def starP2-star-n*)

lemma *star-n-zero-num*: *0* = *star-n* (λ*n*. *0*)
  by (*simp only*: *star-zero-def star-of-def*)

lemma *star-n-one-num*: *1* = *star-n* (λ*n*. *1*)
  by (*simp only*: *star-one-def star-of-def*)

lemma *star-n-abs*: |*star-n X*| = *star-n* (λ*n*. |*X n*|)
  by (*simp only*: *star-abs-def starfun-star-n*)

lemma *hypreal-omega-gt-zero* [*simp*]: *0* < ω
  by (*simp add*: *omega-def star-n-zero-num star-n-less*)

4.6 Existence of Infinite Hyperreal Number

Existence of infinite number not corresponding to any real number. Use assumption that member $\mathcal{U}$ is not finite.

lemma *hypreal-of-real-not-eq-omega*: *hypreal-of-real x* ≠ ω
proof −
  have *False* **if** $\forall_F$ *n in* $\mathcal{U}$. *x* = *1* + *real n* **for** *x*
  proof −
    have *finite* {*n*::*nat*. *x* = *1* + *real n*}
      by (*simp add*: *finite-nat-set-iff-bounded-le*) (*metis add.commute nat-le-linear nat-le-real-less*)
    then show *False*
      using *FreeUltrafilterNat.finite that* **by** *blast*
  qed
  then show *?thesis*
    by (*auto simp add*: *omega-def star-of-def star-n-eq-iff*)
qed

Existence of infinitesimal number also not corresponding to any real number.

lemma *hypreal-of-real-not-eq-epsilon*: *hypreal-of-real x* ≠ ε
proof −

have *False* **if** $\forall_F$ *n in* $\mathcal{U}$. *x* = *inverse* (*1* + *real n*) **for** *x*
proof −
have *finite* {*n*::*nat*. *x* = *inverse* (*1* + *real n*)}
by (*simp add*: *finite-nat-set-iff-bounded-le*) (*metis add.commute inverse-inverse-eq linear nat-le-real-less of-nat-Suc*)
then show *False*
using *FreeUltrafilterNat.finite that* **by** *blast*
qed
then show *?thesis*
by (*auto simp*: *epsilon-def star-of-def star-n-eq-iff*)
qed

lemma *epsilon-ge-zero* [*simp*]: *0* ≤ ε
by (*simp add*: *epsilon-def star-n-zero-num star-n-le*)

lemma *epsilon-not-zero*: ε ≠ *0*
using *hypreal-of-real-not-eq-epsilon* **by** *force*

lemma *epsilon-inverse-omega*: ε = *inverse* ω
by (*simp add*: *epsilon-def omega-def star-n-inverse*)

lemma *epsilon-gt-zero*: *0* < ε
by (*simp add*: *epsilon-inverse-omega*)

4.7 Embedding the Naturals into the Hyperreals

abbreviation *hypreal-of-nat* :: *nat* ⇒ *hypreal*
where *hypreal-of-nat* ≡ *of-nat*

lemma *SNat-eq*: *Nats* = {*n*. ∃ *N*. *n* = *hypreal-of-nat N*}
by (*simp add*: *Nats-def image-def*)

Naturals embedded in hyperreals: is a hyperreal c.f. NS extension.

lemma *hypreal-of-nat*: *hypreal-of-nat m* = *star-n* (λ*n*. *real m*)
by (*simp add*: *star-of-def* [*symmetric*])

declaration ‹
K (*Lin-Arith.add-simps* @{*thms star-of-zero star-of-one*
star-of-numeral star-of-add
star-of-minus star-of-diff star-of-mult}
#> *Lin-Arith.add-inj-thms* @{*thms star-of-le* [*THEN iffD2*]
star-of-less [*THEN iffD2*] *star-of-eq* [*THEN iffD2*]}
#> *Lin-Arith.add-inj-const* (***const-name*** ‹*StarDef.star-of*›, ***typ*** ‹*real* ⇒ *hypreal*›))
›

simproc-setup *fast-arith-hypreal* ((*m*::*hypreal*) < *n* | (*m*::*hypreal*) ≤ *n* | (*m*::*hypreal*) = *n*) =
‹*K Lin-Arith.simproc*›

4.8 Exponentials on the Hyperreals

lemma *hpowr-0* [*simp*]: $r \mathbin{\widehat{}} 0 = (1::hypreal)$
  for $r :: hypreal$
  by (*rule power-0*)

lemma *hpowr-Suc* [*simp*]: $r \mathbin{\widehat{}} (Suc\ n) = r * (r \mathbin{\widehat{}} n)$
  for $r :: hypreal$
  by (*rule power-Suc*)

lemma *hrealpow*: $star\text{-}n\ X \mathbin{\widehat{}} m = star\text{-}n\ (\lambda n.\ (X\ n::real) \mathbin{\widehat{}} m)$
  by (*induct m*) (*auto simp*: *star-n-one-num star-n-mult*)

lemma *hrealpow-sum-square-expand*:
  $(x + y) \mathbin{\widehat{}} Suc\ (Suc\ 0) =$
    $x \mathbin{\widehat{}} Suc\ (Suc\ 0) + y \mathbin{\widehat{}} Suc\ (Suc\ 0) + (hypreal\text{-}of\text{-}nat\ (Suc\ (Suc\ 0))) * x * y$
  for $x\ y :: hypreal$
  by (*simp add*: *distrib-left distrib-right*)

lemma *power-hypreal-of-real-numeral*:
  $(numeral\ v :: hypreal) \mathbin{\widehat{}} n = hypreal\text{-}of\text{-}real\ ((numeral\ v) \mathbin{\widehat{}} n)$
  by *simp*
declare *power-hypreal-of-real-numeral* [*of - numeral w*, *simp*] **for** *w*

lemma *power-hypreal-of-real-neg-numeral*:
  $(-\ numeral\ v :: hypreal) \mathbin{\widehat{}} n = hypreal\text{-}of\text{-}real\ ((-\ numeral\ v) \mathbin{\widehat{}} n)$
  by *simp*
declare *power-hypreal-of-real-neg-numeral* [*of - numeral w*, *simp*] **for** *w*

4.9 Powers with Hypernatural Exponents

Hypernatural powers of hyperreals.

definition $pow :: 'a::power\ star \Rightarrow nat\ star \Rightarrow 'a\ star$ (**infixr** ‹*pow*› *80*)
  where *hyperpow-def* [*transfer-unfold*]: $R\ pow\ N = (\ *f2*\ (\widehat{}))\ R\ N$

lemma *Standard-hyperpow* [*simp*]: $r \in Standard \Longrightarrow n \in Standard \Longrightarrow r\ pow\ n \in Standard$
  by (*simp add*: *hyperpow-def*)

lemma *hyperpow*: $star\text{-}n\ X\ pow\ star\text{-}n\ Y = star\text{-}n\ (\lambda n.\ X\ n \mathbin{\widehat{}} Y\ n)$
  by (*simp add*: *hyperpow-def starfun2-star-n*)

lemma *hyperpow-zero* [*simp*]: $\bigwedge n.\ (0::'a::\{power,semiring\text{-}0\}\ star)\ pow\ (n + (1::hypnat)) = 0$
  by *transfer simp*

lemma *hyperpow-not-zero*: $\bigwedge r\ n.\ r \neq (0::'a::\{field\}\ star) \Longrightarrow r\ pow\ n \neq 0$
  by *transfer* (*rule power-not-zero*)

lemma *hyperpow-inverse*: $\bigwedge r\ n.\ r \neq (0::'a::field\ star) \Longrightarrow inverse\ (r\ pow\ n) = (inverse\ r)\ pow\ n$
by *transfer* (*rule power-inverse* [*symmetric*])

lemma *hyperpow-hrabs*: $\bigwedge r\ n.\ |r::'a::\{linordered\text{-}idom\}\ star|\ pow\ n = |r\ pow\ n|$
by *transfer* (*rule power-abs* [*symmetric*])

lemma *hyperpow-add*: $\bigwedge r\ n\ m.\ (r::'a::monoid\text{-}mult\ star)\ pow\ (n + m) = (r\ pow\ n) * (r\ pow\ m)$
by *transfer* (*rule power-add*)

lemma *hyperpow-one* [*simp*]: $\bigwedge r.\ (r::'a::monoid\text{-}mult\ star)\ pow\ (1::hypnat) = r$
by *transfer* (*rule power-one-right*)

lemma *hyperpow-two*: $\bigwedge r.\ (r::'a::monoid\text{-}mult\ star)\ pow\ (2::hypnat) = r * r$
by *transfer* (*rule power2-eq-square*)

lemma *hyperpow-gt-zero*: $\bigwedge r\ n.\ (0::'a::\{linordered\text{-}semidom\}\ star) < r \Longrightarrow 0 < r\ pow\ n$
by *transfer* (*rule zero-less-power*)

lemma *hyperpow-ge-zero*: $\bigwedge r\ n.\ (0::'a::\{linordered\text{-}semidom\}\ star) \leq r \Longrightarrow 0 \leq r\ pow\ n$
by *transfer* (*rule zero-le-power*)

lemma *hyperpow-le*: $\bigwedge x\ y\ n.\ (0::'a::\{linordered\text{-}semidom\}\ star) < x \Longrightarrow x \leq y \Longrightarrow x\ pow\ n \leq y\ pow\ n$
by *transfer* (*rule power-mono* [*OF - order-less-imp-le*])

lemma *hyperpow-eq-one* [*simp*]: $\bigwedge n.\ 1\ pow\ n = (1::'a::monoid\text{-}mult\ star)$
by *transfer* (*rule power-one*)

lemma *hrabs-hyperpow-minus* [*simp*]: $\bigwedge(a::'a::linordered\text{-}idom\ star)\ n.\ |(-a)\ pow\ n| = |a\ pow\ n|$
by *transfer* (*rule abs-power-minus*)

lemma *hyperpow-mult*: $\bigwedge r\ s\ n.\ (r * s::'a::comm\text{-}monoid\text{-}mult\ star)\ pow\ n = (r\ pow\ n) * (s\ pow\ n)$
by *transfer* (*rule power-mult-distrib*)

lemma *hyperpow-two-le* [*simp*]: $\bigwedge r.\ (0::'a::\{monoid\text{-}mult,linordered\text{-}ring\text{-}strict\}\ star) \leq r\ pow\ 2$
by (*auto simp add*: *hyperpow-two zero-le-mult-iff*)

lemma *hyperpow-two-hrabs* [*simp*]: $|x::'a::linordered\text{-}idom\ star|\ pow\ 2 = x\ pow\ 2$
by (*simp add*: *hyperpow-hrabs*)

lemma *hyperpow-two-gt-one*: $\bigwedge r::'a::linordered\text{-}semidom\ star.\ 1 < r \Longrightarrow 1 < r\ pow\ 2$

by *transfer simp*

lemma *hyperpow-two-ge-one*: $\bigwedge r::'a::linordered\text{-}semidom\ star.\ 1 \le r \Longrightarrow 1 \le r$ *pow* 2
by *transfer* (*rule one-le-power*)

lemma *two-hyperpow-ge-one* [*simp*]: $(1::hypreal) \le 2$ *pow* n
by (*metis hyperpow-eq-one hyperpow-le one-le-numeral zero-less-one*)

lemma *hyperpow-minus-one2* [*simp*]: $\bigwedge n.\ (-\ 1)$ *pow* $(2 * n) = (1::hypreal)$
by *transfer* (*rule power-minus1-even*)

lemma *hyperpow-less-le*: $\bigwedge r\ n\ N.\ (0::hypreal) \le r \Longrightarrow r \le 1 \Longrightarrow n < N \Longrightarrow r$ *pow* $N \le r$ *pow* n
by *transfer* (*rule power-decreasing* [*OF order-less-imp-le*])

lemma *hyperpow-SHNat-le*:
$0 \le r \Longrightarrow r \le (1::hypreal) \Longrightarrow N \in HNatInfinite \Longrightarrow \forall n \in Nats.\ r$ *pow* $N \le r$ *pow* n
by (*auto intro*!: *hyperpow-less-le simp*: *HNatInfinite-iff*)

lemma *hyperpow-realpow*: (*hypreal-of-real* r) *pow* (*hypnat-of-nat* n) = *hypreal-of-real* $(r \mathbin{\char`\^} n)$
by *transfer* (*rule refl*)

lemma *hyperpow-SReal* [*simp*]: (*hypreal-of-real* r) *pow* (*hypnat-of-nat* n) $\in \mathbb{R}$
by (*simp add*: *Reals-eq-Standard*)

lemma *hyperpow-zero-HNatInfinite* [*simp*]: $N \in HNatInfinite \Longrightarrow (0::hypreal)$ *pow* $N = 0$
by (*drule HNatInfinite-is-Suc*, *auto*)

lemma *hyperpow-le-le*: $(0::hypreal) \le r \Longrightarrow r \le 1 \Longrightarrow n \le N \Longrightarrow r$ *pow* $N \le r$ *pow* n
by (*metis hyperpow-less-le le-less*)

lemma *hyperpow-Suc-le-self2*: $(0::hypreal) \le r \Longrightarrow r < 1 \Longrightarrow r$ *pow* $(n + (1::hypnat)) \le r$
by (*metis hyperpow-less-le hyperpow-one hypnat-add-self-le le-less*)

lemma *hyperpow-hypnat-of-nat*: $\bigwedge x.\ x$ *pow hypnat-of-nat* $n = x \mathbin{\char`\^} n$
by *transfer* (*rule refl*)

lemma *of-hypreal-hyperpow*:
$\bigwedge x\ n.$ *of-hypreal* (x *pow* n) = (*of-hypreal* $x::'a::\{real\text{-}algebra\text{-}1\}\ star$) *pow* n
by *transfer* (*rule of-real-power*)

end

5 Infinite Numbers, Infinitesimals, Infinitely Close Relation

theory *NSA*
imports *HyperDef HOL−Library.Lub-Glb*
begin

definition *hnorm* :: *'a::real-normed-vector star* ⇒ *real star*
where [*transfer-unfold*]: *hnorm* = *∗f∗ norm*

definition *Infinitesimal* :: (*'a::real-normed-vector*) *star set*
where *Infinitesimal* = {*x*. ∀ *r* ∈ *Reals*. *0* < *r* ⟶ *hnorm x* < *r*}

definition *HFinite* :: (*'a::real-normed-vector*) *star set*
where *HFinite* = {*x*. ∃ *r* ∈ *Reals*. *hnorm x* < *r*}

definition *HInfinite* :: (*'a::real-normed-vector*) *star set*
where *HInfinite* = {*x*. ∀ *r* ∈ *Reals*. *r* < *hnorm x*}

definition *approx* :: *'a::real-normed-vector star* ⇒ *'a star* ⇒ *bool* (**infixl** ‹≈› *50*)
where *x* ≈ *y* ⟷ *x* − *y* ∈ *Infinitesimal*
— the "infinitely close" relation

definition *st* :: *hypreal* ⇒ *hypreal*
where *st* = (*λx*. *SOME r*. *x* ∈ *HFinite* ∧ *r* ∈ ℝ ∧ *r* ≈ *x*)
— the standard part of a hyperreal

definition *monad* :: *'a::real-normed-vector star* ⇒ *'a star set*
where *monad x* = {*y*. *x* ≈ *y*}

definition *galaxy* :: *'a::real-normed-vector star* ⇒ *'a star set*
where *galaxy x* = {*y*. (*x* + −*y*) ∈ *HFinite*}

lemma *SReal-def*: ℝ ≡ {*x*. ∃ *r*. *x* = *hypreal-of-real r*}
by (*simp add*: *Reals-def image-def*)

5.1 Nonstandard Extension of the Norm Function

definition *scaleHR* :: *real star* ⇒ *'a star* ⇒ *'a::real-normed-vector star*
where [*transfer-unfold*]: *scaleHR* = *starfun2 scaleR*

lemma *Standard-hnorm* [*simp*]: *x* ∈ *Standard* ⟹ *hnorm x* ∈ *Standard*
by (*simp add*: *hnorm-def*)

lemma *star-of-norm* [*simp*]: *star-of* (*norm x*) = *hnorm* (*star-of x*)
by *transfer* (*rule refl*)

lemma *hnorm-ge-zero* [*simp*]: ⋀*x*::*'a::real-normed-vector star*. *0* ≤ *hnorm x*
by *transfer* (*rule norm-ge-zero*)

lemma *hnorm-eq-zero* [*simp*]: $\bigwedge x::'a::real\text{-}normed\text{-}vector\ star.\ hnorm\ x = 0 \longleftrightarrow x = 0$
by *transfer* (*rule norm-eq-zero*)

lemma *hnorm-triangle-ineq*: $\bigwedge x\ y::'a::real\text{-}normed\text{-}vector\ star.\ hnorm\ (x + y) \leq hnorm\ x + hnorm\ y$
by *transfer* (*rule norm-triangle-ineq*)

lemma *hnorm-triangle-ineq3*: $\bigwedge x\ y::'a::real\text{-}normed\text{-}vector\ star.\ |hnorm\ x - hnorm\ y| \leq hnorm\ (x - y)$
by *transfer* (*rule norm-triangle-ineq3*)

lemma *hnorm-scaleR*: $\bigwedge x::'a::real\text{-}normed\text{-}vector\ star.\ hnorm\ (a *_R x) = |star\text{-}of\ a| * hnorm\ x$
by *transfer* (*rule norm-scaleR*)

lemma *hnorm-scaleHR*: $\bigwedge a\ (x::'a::real\text{-}normed\text{-}vector\ star).\ hnorm\ (scaleHR\ a\ x) = |a| * hnorm\ x$
by *transfer* (*rule norm-scaleR*)

lemma *hnorm-mult-ineq*: $\bigwedge x\ y::'a::real\text{-}normed\text{-}algebra\ star.\ hnorm\ (x * y) \leq hnorm\ x * hnorm\ y$
by *transfer* (*rule norm-mult-ineq*)

lemma *hnorm-mult*: $\bigwedge x\ y::'a::real\text{-}normed\text{-}div\text{-}algebra\ star.\ hnorm\ (x * y) = hnorm\ x * hnorm\ y$
by *transfer* (*rule norm-mult*)

lemma *hnorm-hyperpow*: $\bigwedge(x::'a::\{real\text{-}normed\text{-}div\text{-}algebra\}\ star)\ n.\ hnorm\ (x\ pow\ n) = hnorm\ x\ pow\ n$
by *transfer* (*rule norm-power*)

lemma *hnorm-one* [*simp*]: $hnorm\ (1::'a::real\text{-}normed\text{-}div\text{-}algebra\ star) = 1$
by *transfer* (*rule norm-one*)

lemma *hnorm-zero* [*simp*]: $hnorm\ (0::'a::real\text{-}normed\text{-}vector\ star) = 0$
by *transfer* (*rule norm-zero*)

lemma *zero-less-hnorm-iff* [*simp*]: $\bigwedge x::'a::real\text{-}normed\text{-}vector\ star.\ 0 < hnorm\ x \longleftrightarrow x \neq 0$
by *transfer* (*rule zero-less-norm-iff*)

lemma *hnorm-minus-cancel* [*simp*]: $\bigwedge x::'a::real\text{-}normed\text{-}vector\ star.\ hnorm\ (- x) = hnorm\ x$
by *transfer* (*rule norm-minus-cancel*)

lemma *hnorm-minus-commute*: $\bigwedge a\ b::'a::real\text{-}normed\text{-}vector\ star.\ hnorm\ (a - b) = hnorm\ (b - a)$

by *transfer* (*rule norm-minus-commute*)

lemma *hnorm-triangle-ineq2*: $\bigwedge a\ b{::}'a{::}real\text{-}normed\text{-}vector\ star.\ hnorm\ a - hnorm\ b \le hnorm\ (a - b)$
by *transfer* (*rule norm-triangle-ineq2*)

lemma *hnorm-triangle-ineq4*: $\bigwedge a\ b{::}'a{::}real\text{-}normed\text{-}vector\ star.\ hnorm\ (a - b) \le hnorm\ a + hnorm\ b$
by *transfer* (*rule norm-triangle-ineq4*)

lemma *abs-hnorm-cancel* [*simp*]: $\bigwedge a{::}'a{::}real\text{-}normed\text{-}vector\ star.\ |hnorm\ a| = hnorm\ a$
by *transfer* (*rule abs-norm-cancel*)

lemma *hnorm-of-hypreal* [*simp*]: $\bigwedge r.\ hnorm\ (of\text{-}hypreal\ r{::}'a{::}real\text{-}normed\text{-}algebra\text{-}1\ star) = |r|$
by *transfer* (*rule norm-of-real*)

lemma *nonzero-hnorm-inverse*:
$\bigwedge a{::}'a{::}real\text{-}normed\text{-}div\text{-}algebra\ star.\ a \ne 0 \Longrightarrow hnorm\ (inverse\ a) = inverse\ (hnorm\ a)$
by *transfer* (*rule nonzero-norm-inverse*)

lemma *hnorm-inverse*:
$\bigwedge a{::}'a{::}\{real\text{-}normed\text{-}div\text{-}algebra,\ division\text{-}ring\}\ star.\ hnorm\ (inverse\ a) = inverse\ (hnorm\ a)$
by *transfer* (*rule norm-inverse*)

lemma *hnorm-divide*: $\bigwedge a\ b{::}'a{::}\{real\text{-}normed\text{-}field,\ field\}\ star.\ hnorm\ (a\ /\ b) = hnorm\ a\ /\ hnorm\ b$
by *transfer* (*rule norm-divide*)

lemma *hypreal-hnorm-def* [*simp*]: $\bigwedge r{::}hypreal.\ hnorm\ r = |r|$
by *transfer* (*rule real-norm-def*)

lemma *hnorm-add-less*:
$\bigwedge(x{::}'a{::}real\text{-}normed\text{-}vector\ star)\ y\ r\ s.\ hnorm\ x < r \Longrightarrow hnorm\ y < s \Longrightarrow hnorm\ (x + y) < r + s$
by *transfer* (*rule norm-add-less*)

lemma *hnorm-mult-less*:
$\bigwedge(x{::}'a{::}real\text{-}normed\text{-}algebra\ star)\ y\ r\ s.\ hnorm\ x < r \Longrightarrow hnorm\ y < s \Longrightarrow hnorm\ (x * y) < r * s$
by *transfer* (*rule norm-mult-less*)

lemma *hnorm-scaleHR-less*: $|x| < r \Longrightarrow hnorm\ y < s \Longrightarrow hnorm\ (scaleHR\ x\ y) < r * s$
by (*simp only*: *hnorm-scaleHR*) (*simp add*: *mult-strict-mono'*)

5.2 Closure Laws for the Standard Reals

lemma *Reals-add-cancel*: $x + y \in \mathbb{R} \Longrightarrow y \in \mathbb{R} \Longrightarrow x \in \mathbb{R}$
by (*drule* (*1*) *Reals-diff*) *simp*

lemma *SReal-hrabs*: $x \in \mathbb{R} \Longrightarrow |x| \in \mathbb{R}$
for x :: *hypreal*
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-hypreal-of-real* [*simp*]: *hypreal-of-real* $x \in \mathbb{R}$
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-divide-numeral*: $r \in \mathbb{R} \Longrightarrow r$ / (*numeral* w::*hypreal*) $\in \mathbb{R}$
by *simp*

ε is not in Reals because it is an infinitesimal

lemma *SReal-epsilon-not-mem*: $\varepsilon \notin \mathbb{R}$
by (*auto simp*: *SReal-def hypreal-of-real-not-eq-epsilon* [*symmetric*])

lemma *SReal-omega-not-mem*: $\omega \notin \mathbb{R}$
by (*auto simp*: *SReal-def hypreal-of-real-not-eq-omega* [*symmetric*])

lemma *SReal-UNIV-real*: {x. *hypreal-of-real* $x \in \mathbb{R}$} = (*UNIV*::*real set*)
by *simp*

lemma *SReal-iff*: $x \in \mathbb{R} \longleftrightarrow (\exists\, y.\ x =$ *hypreal-of-real* $y)$
by (*simp add*: *SReal-def*)

lemma *hypreal-of-real-image*: *hypreal-of-real* ‘(*UNIV*::*real set*) = $\mathbb{R}$
by (*simp add*: *Reals-eq-Standard Standard-def*)

lemma *inv-hypreal-of-real-image*: *inv hypreal-of-real* ‘ $\mathbb{R}$ = *UNIV*
by (*simp add*: *Reals-eq-Standard Standard-def inj-star-of*)

lemma *SReal-dense*: $x \in \mathbb{R} \Longrightarrow y \in \mathbb{R} \Longrightarrow x < y \Longrightarrow \exists\, r \in$ *Reals*. $x < r \wedge r < y$
for $x\ y$ :: *hypreal*
using *dense* **by** (*fastforce simp add*: *SReal-def*)

5.3 Set of Finite Elements is a Subring of the Extended Reals

lemma *HFinite-add*: $x \in$ *HFinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow x + y \in$ *HFinite*
unfolding *HFinite-def* **by** (*blast intro*!: *Reals-add hnorm-add-less*)

lemma *HFinite-mult*: $x \in$ *HFinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow x * y \in$ *HFinite*
for $x\ y$ :: *'a*::*real-normed-algebra star*
unfolding *HFinite-def* **by** (*blast intro*!: *Reals-mult hnorm-mult-less*)

lemma *HFinite-scaleHR*: $x \in$ *HFinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow$ *scaleHR* $x\ y \in$ *HFinite*
by (*auto simp*: *HFinite-def intro*!: *Reals-mult hnorm-scaleHR-less*)

```
lemma HFinite-minus-iff: − x ∈ HFinite ⟷ x ∈ HFinite
  by (simp add: HFinite-def)

lemma HFinite-star-of [simp]: star-of x ∈ HFinite
 by (simp add: HFinite-def) (metis SReal-hypreal-of-real gt-ex star-of-less star-of-norm)

lemma SReal-subset-HFinite: (ℝ::hypreal set) ⊆ HFinite
  by (auto simp add: SReal-def)

lemma HFiniteD: x ∈ HFinite ⟹ ∃ t ∈ Reals. hnorm x < t
  by (simp add: HFinite-def)

lemma HFinite-hrabs-iff [iff]: |x| ∈ HFinite ⟷ x ∈ HFinite
  for x :: hypreal
  by (simp add: HFinite-def)

lemma HFinite-hnorm-iff [iff]: hnorm x ∈ HFinite ⟷ x ∈ HFinite
  for x :: hypreal
  by (simp add: HFinite-def)

lemma HFinite-numeral [simp]: numeral w ∈ HFinite
  unfolding star-numeral-def by (rule HFinite-star-of)
```

As always with numerals, *0* and *1* are special cases.

```
lemma HFinite-0 [simp]: 0 ∈ HFinite
  unfolding star-zero-def by (rule HFinite-star-of)

lemma HFinite-1 [simp]: 1 ∈ HFinite
  unfolding star-one-def by (rule HFinite-star-of)

lemma hrealpow-HFinite: x ∈ HFinite ⟹ x ^ n ∈ HFinite
  for x :: 'a::{real-normed-algebra,monoid-mult} star
  by (induct n) (auto intro: HFinite-mult)

lemma HFinite-bounded:
  fixes x y :: hypreal
  assumes x ∈ HFinite and y: y ≤ x 0 ≤ y shows y ∈ HFinite
proof (cases x ≤ 0)
  case True
  then have y = 0
    using y by auto
  then show ?thesis
    by simp
next
  case False
  then show ?thesis
    using assms le-less-trans by (auto simp: HFinite-def)
qed
```

5.4 Set of Infinitesimals is a Subring of the Hyperreals

lemma *InfinitesimalI*: ($\bigwedge r.\ r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow hnorm\ x < r$) $\Longrightarrow x \in$ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *InfinitesimalD*: $x \in$ *Infinitesimal* $\Longrightarrow \forall\, r \in$ *Reals*. $0 < r \longrightarrow hnorm\ x < r$
by (*simp add*: *Infinitesimal-def*)

lemma *InfinitesimalI2*: ($\bigwedge r.\ 0 < r \Longrightarrow hnorm\ x <$ *star-of* r) $\Longrightarrow x \in$ *Infinitesimal*
by (*auto simp add*: *Infinitesimal-def SReal-def*)

lemma *InfinitesimalD2*: $x \in$ *Infinitesimal* $\Longrightarrow 0 < r \Longrightarrow hnorm\ x <$ *star-of* r
by (*auto simp add*: *Infinitesimal-def SReal-def*)

lemma *Infinitesimal-zero* [*iff*]: $0 \in$ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-add*:
assumes $x \in$ *Infinitesimal* $y \in$ *Infinitesimal*
shows $x + y \in$ *Infinitesimal*
proof (*rule InfinitesimalI*)
show *hnorm* $(x + y) < r$
if $r \in \mathbb{R}$ **and** $0 < r$ **for** r :: *real star*
proof −
have *hnorm* $x < r/2$ *hnorm* $y < r/2$
using *InfinitesimalD SReal-divide-numeral assms half-gt-zero that* **by** *blast+*
then show *?thesis*
using *hnorm-add-less* **by** *fastforce*
qed
qed

lemma *Infinitesimal-minus-iff* [*simp*]: $-\ x \in$ *Infinitesimal* $\longleftrightarrow x \in$ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-hnorm-iff*: *hnorm* $x \in$ *Infinitesimal* $\longleftrightarrow x \in$ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-hrabs-iff* [*iff*]: $|x| \in$ *Infinitesimal* $\longleftrightarrow x \in$ *Infinitesimal*
for x :: *hypreal*
by (*simp add*: *abs-if*)

lemma *Infinitesimal-of-hypreal-iff* [*simp*]:
(*of-hypreal* x::$'a$::*real-normed-algebra-1 star*) $\in$ *Infinitesimal* $\longleftrightarrow x \in$ *Infinitesimal*
by (*subst Infinitesimal-hnorm-iff* [*symmetric*]) *simp*

lemma *Infinitesimal-diff*: $x \in$ *Infinitesimal* $\Longrightarrow y \in$ *Infinitesimal* $\Longrightarrow x - y \in$ *Infinitesimal*
using *Infinitesimal-add* [*of* x $-\ y$] **by** *simp*

lemma *Infinitesimal-mult*:
  fixes *x y* :: *'a::real-normed-algebra star*
  assumes *x* ∈ *Infinitesimal y* ∈ *Infinitesimal*
  shows *x* ∗ *y* ∈ *Infinitesimal*
  proof (*rule InfinitesimalI*)
  show *hnorm* (*x* ∗ *y*) < *r*
    if *r* ∈ ℝ **and** *0* < *r* **for** *r* :: *real star*
  proof −
    have *hnorm x* < *1 hnorm y* < *r*
      using *assms that* **by** (*auto simp add*: *InfinitesimalD*)
    then show *?thesis*
      using *hnorm-mult-less* **by** *fastforce*
  qed
qed

lemma *Infinitesimal-HFinite-mult*:
  fixes *x y* :: *'a::real-normed-algebra star*
  assumes *x* ∈ *Infinitesimal y* ∈ *HFinite*
  shows *x* ∗ *y* ∈ *Infinitesimal*
proof (*rule InfinitesimalI*)
  obtain *t* **where** *hnorm y* < *t t* ∈ *Reals*
    using *HFiniteD* ‹*y* ∈ *HFinite*› **by** *blast*
  then have *t* > *0*
    using *hnorm-ge-zero le-less-trans* **by** *blast*
  show *hnorm* (*x* ∗ *y*) < *r*
    if *r* ∈ ℝ **and** *0* < *r* **for** *r* :: *real star*
  proof −
    have *hnorm x* < *r*/*t*
        by (*meson InfinitesimalD Reals-divide* ‹*hnorm y* < *t*› ‹*t* ∈ ℝ› *assms*(*1*)
divide-pos-pos hnorm-ge-zero le-less-trans that)
    then have *hnorm* (*x* ∗ *y*) < (*r* / *t*) ∗ *t*
      using ‹*hnorm y* < *t*› *hnorm-mult-less* **by** *blast*
    then show *?thesis*
      using ‹*0* < *t*› **by** *auto*
  qed
qed

lemma *Infinitesimal-HFinite-scaleHR*:
  assumes *x* ∈ *Infinitesimal y* ∈ *HFinite*
  shows *scaleHR x y* ∈ *Infinitesimal*
proof (*rule InfinitesimalI*)
  obtain *t* **where** *hnorm y* < *t t* ∈ *Reals*
    using *HFiniteD* ‹*y* ∈ *HFinite*› **by** *blast*
  then have *t* > *0*
    using *hnorm-ge-zero le-less-trans* **by** *blast*
  show *hnorm* (*scaleHR x y*) < *r*
    if *r* ∈ ℝ **and** *0* < *r* **for** *r* :: *real star*
  proof −
    have |*x*| ∗ *hnorm y* < (*r* / *t*) ∗ *t*

by (*metis InfinitesimalD Reals-divide* ‹$0 < t$› ‹*hnorm* $y < t$› ‹$t \in \mathbb{R}$› *assms*(*1*) *divide-pos-pos hnorm-ge-zero hypreal-hnorm-def mult-strict-mono′ that*)
then show *?thesis*
by (*simp add*: ‹$0 < t$› *hnorm-scaleHR less-imp-not-eq2*)
qed
qed

lemma *Infinitesimal-HFinite-mult2*:
fixes x y :: *′a::real-normed-algebra star*
assumes $x \in$ *Infinitesimal* $y \in$ *HFinite*
shows $y * x \in$ *Infinitesimal*
proof (*rule InfinitesimalI*)
obtain t **where** *hnorm* $y < t$ $t \in$ *Reals*
using *HFiniteD* ‹$y \in$ *HFinite*› **by** *blast*
then have $t > 0$
using *hnorm-ge-zero le-less-trans* **by** *blast*
show *hnorm* $(y * x) < r$
if $r \in \mathbb{R}$ **and** $0 < r$ **for** r :: *real star*
proof −
have *hnorm* $x < r/t$
by (*meson InfinitesimalD Reals-divide* ‹*hnorm* $y < t$› ‹$t \in \mathbb{R}$› *assms*(*1*) *divide-pos-pos hnorm-ge-zero le-less-trans that*)
then have *hnorm* $(y * x) < t * (r / t)$
using ‹*hnorm* $y < t$› *hnorm-mult-less* **by** *blast*
then show *?thesis*
using ‹$0 < t$› **by** *auto*
qed
qed

lemma *Infinitesimal-scaleR2*:
assumes $x \in$ *Infinitesimal* **shows** $a *_R x \in$ *Infinitesimal*
by (*metis HFinite-star-of Infinitesimal-HFinite-mult2 Infinitesimal-hnorm-iff assms hnorm-scaleR hypreal-hnorm-def star-of-norm*)

lemma *Compl-HFinite*: − *HFinite* = *HInfinite*
proof −
have $r <$ *hnorm* x **if** $*$: $\bigwedge s.\ s \in \mathbb{R} \Longrightarrow s \le$ *hnorm* x **and** $r \in \mathbb{R}$
for x :: *′a star* **and** r :: *hypreal*
using $*$ [*of* $r+1$] ‹$r \in \mathbb{R}$› **by** *auto*
then show *?thesis*
by (*auto simp add*: *HInfinite-def HFinite-def linorder-not-less*)
qed

lemma *HInfinite-inverse-Infinitesimal*:
$x \in$ *HInfinite* $\Longrightarrow$ *inverse* $x \in$ *Infinitesimal*
for x :: *′a::real-normed-div-algebra star*
by (*simp add*: *HInfinite-def InfinitesimalI hnorm-inverse inverse-less-imp-less*)

lemma *inverse-Infinitesimal-iff-HInfinite*:

$x \neq 0 \Longrightarrow inverse\ x \in Infinitesimal \longleftrightarrow x \in HInfinite$
for x :: $'a$::*real-normed-div-algebra star*
by (*metis Compl-HFinite Compl-iff HInfinite-inverse-Infinitesimal InfinitesimalD Infinitesimal-HFinite-mult Reals-1 hnorm-one left-inverse less-irrefl zero-less-one*)

lemma *HInfiniteI*: $(\bigwedge r.\ r \in \mathbb{R} \Longrightarrow r < hnorm\ x) \Longrightarrow x \in HInfinite$
by (*simp add*: *HInfinite-def*)

lemma *HInfiniteD*: $x \in HInfinite \Longrightarrow r \in \mathbb{R} \Longrightarrow r < hnorm\ x$
by (*simp add*: *HInfinite-def*)

lemma *HInfinite-mult*:
fixes $x\ y$ :: $'a$::*real-normed-div-algebra star*
assumes $x \in HInfinite$ $y \in HInfinite$ **shows** $x * y \in HInfinite$
proof (*rule HInfiniteI*, *simp only*: *hnorm-mult*)
have $x \neq 0$
using *Compl-HFinite HFinite-0 assms* **by** *blast*
show $r < hnorm\ x * hnorm\ y$
if $r \in \mathbb{R}$ **for** r :: *real star*
proof −
have $r = r * 1$
by *simp*
also have $\ldots < hnorm\ x * hnorm\ y$
by (*meson HInfiniteD Reals-1* ‹$x \neq 0$› *assms le-numeral-extra*(*1*) *mult-strict-mono that zero-less-hnorm-iff*)
finally show *?thesis* **.**
qed
qed

lemma *hypreal-add-zero-less-le-mono*: $r < x \Longrightarrow 0 \leq y \Longrightarrow r < x + y$
for $r\ x\ y$ :: *hypreal*
by *simp*

lemma *HInfinite-add-ge-zero*: $x \in HInfinite \Longrightarrow 0 \leq y \Longrightarrow 0 \leq x \Longrightarrow x + y \in HInfinite$
for $x\ y$ :: *hypreal*
by (*auto simp*: *abs-if add.commute HInfinite-def*)

lemma *HInfinite-add-ge-zero2*: $x \in HInfinite \Longrightarrow 0 \leq y \Longrightarrow 0 \leq x \Longrightarrow y + x \in HInfinite$
for $x\ y$ :: *hypreal*
by (*auto intro*!: *HInfinite-add-ge-zero simp add*: *add.commute*)

lemma *HInfinite-add-gt-zero*: $x \in HInfinite \Longrightarrow 0 < y \Longrightarrow 0 < x \Longrightarrow x + y \in HInfinite$
for $x\ y$ :: *hypreal*
by (*blast intro*: *HInfinite-add-ge-zero order-less-imp-le*)

lemma *HInfinite-minus-iff*: $-\ x \in HInfinite \longleftrightarrow x \in HInfinite$

by (*simp add*: *HInfinite-def*)

lemma *HInfinite-add-le-zero*: $x \in HInfinite \Longrightarrow y \leq 0 \Longrightarrow x \leq 0 \Longrightarrow x + y \in HInfinite$
for $x\ y$ :: *hypreal*
by (*metis* (*no-types*, *lifting*) *HInfinite-add-ge-zero2 HInfinite-minus-iff add.inverse-distrib-swap neg-0-le-iff-le*)

lemma *HInfinite-add-lt-zero*: $x \in HInfinite \Longrightarrow y < 0 \Longrightarrow x < 0 \Longrightarrow x + y \in HInfinite$
for $x\ y$ :: *hypreal*
by (*blast intro*: *HInfinite-add-le-zero order-less-imp-le*)

lemma *not-Infinitesimal-not-zero*: $x \notin Infinitesimal \Longrightarrow x \neq 0$
by *auto*

lemma *HFinite-diff-Infinitesimal-hrabs*:
$x \in HFinite - Infinitesimal \Longrightarrow |x| \in HFinite - Infinitesimal$
for x :: *hypreal*
by *blast*

lemma *hnorm-le-Infinitesimal*: $e \in Infinitesimal \Longrightarrow hnorm\ x \leq e \Longrightarrow x \in Infinitesimal$
by (*auto simp*: *Infinitesimal-def abs-less-iff*)

lemma *hnorm-less-Infinitesimal*: $e \in Infinitesimal \Longrightarrow hnorm\ x < e \Longrightarrow x \in Infinitesimal$
by (*erule hnorm-le-Infinitesimal*, *erule order-less-imp-le*)

lemma *hrabs-le-Infinitesimal*: $e \in Infinitesimal \Longrightarrow |x| \leq e \Longrightarrow x \in Infinitesimal$
for x :: *hypreal*
by (*erule hnorm-le-Infinitesimal*) *simp*

lemma *hrabs-less-Infinitesimal*: $e \in Infinitesimal \Longrightarrow |x| < e \Longrightarrow x \in Infinitesimal$
for x :: *hypreal*
by (*erule hnorm-less-Infinitesimal*) *simp*

lemma *Infinitesimal-interval*:
$e \in Infinitesimal \Longrightarrow e' \in Infinitesimal \Longrightarrow e' < x \Longrightarrow x < e \Longrightarrow x \in Infinitesimal$
for x :: *hypreal*
by (*auto simp add*: *Infinitesimal-def abs-less-iff*)

lemma *Infinitesimal-interval2*:
$e \in Infinitesimal \Longrightarrow e' \in Infinitesimal \Longrightarrow e' \leq x \Longrightarrow x \leq e \Longrightarrow x \in Infinitesimal$
for x :: *hypreal*
by (*auto intro*: *Infinitesimal-interval simp add*: *order-le-less*)

lemma *lemma-Infinitesimal-hyperpow*: $x \in Infinitesimal \Longrightarrow 0 < N \Longrightarrow |x\ pow\ N| \leq |x|$

for *x* :: *hypreal*
apply (*clarsimp simp*: *Infinitesimal-def*)
by (*metis Reals-1 abs-ge-zero hyperpow-Suc-le-self2 hyperpow-hrabs hypnat-gt-zero-iff2 zero-less-one*)

lemma *Infinitesimal-hyperpow*: *x* ∈ *Infinitesimal* ⟹ *0* < *N* ⟹ *x pow N* ∈ *Infinitesimal*
for *x* :: *hypreal*
using *hrabs-le-Infinitesimal lemma-Infinitesimal-hyperpow* **by** *blast*

lemma *hrealpow-hyperpow-Infinitesimal-iff*:
(*x* ^ *n* ∈ *Infinitesimal*) ⟷ *x pow* (*hypnat-of-nat n*) ∈ *Infinitesimal*
by (*simp only*: *hyperpow-hypnat-of-nat*)

lemma *Infinitesimal-hrealpow*: *x* ∈ *Infinitesimal* ⟹ *0* < *n* ⟹ *x* ^ *n* ∈ *Infinitesimal*
for *x* :: *hypreal*
by (*simp add*: *hrealpow-hyperpow-Infinitesimal-iff Infinitesimal-hyperpow*)

lemma *not-Infinitesimal-mult*:
x ∉ *Infinitesimal* ⟹ *y* ∉ *Infinitesimal* ⟹ *x* ∗ *y* ∉ *Infinitesimal*
for *x y* :: *'a::real-normed-div-algebra star*
by (*metis* (*no-types*, *lifting*) *inverse-Infinitesimal-iff-HInfinite ComplI Compl-HFinite Infinitesimal-HFinite-mult divide-inverse eq-divide-imp inverse-inverse-eq mult-zero-right*)

lemma *Infinitesimal-mult-disj*: *x* ∗ *y* ∈ *Infinitesimal* ⟹ *x* ∈ *Infinitesimal* ∨ *y* ∈ *Infinitesimal*
for *x y* :: *'a::real-normed-div-algebra star*
using *not-Infinitesimal-mult* **by** *blast*

lemma *HFinite-Infinitesimal-not-zero*: *x* ∈ *HFinite*−*Infinitesimal* ⟹ *x* ≠ *0*
by *blast*

lemma *HFinite-Infinitesimal-diff-mult*:
x ∈ *HFinite* − *Infinitesimal* ⟹ *y* ∈ *HFinite* − *Infinitesimal* ⟹ *x* ∗ *y* ∈ *HFinite* − *Infinitesimal*
for *x y* :: *'a::real-normed-div-algebra star*
by (*simp add*: *HFinite-mult not-Infinitesimal-mult*)

lemma *Infinitesimal-subset-HFinite*: *Infinitesimal* ⊆ *HFinite*
using *HFinite-def InfinitesimalD Reals-1 zero-less-one* **by** *blast*

lemma *Infinitesimal-star-of-mult*: *x* ∈ *Infinitesimal* ⟹ *x* ∗ *star-of r* ∈ *Infinitesimal*
for *x* :: *'a::real-normed-algebra star*
by (*erule HFinite-star-of* [*THEN* [*2*] *Infinitesimal-HFinite-mult*])

lemma *Infinitesimal-star-of-mult2*: *x* ∈ *Infinitesimal* ⟹ *star-of r* ∗ *x* ∈ *Infinitesimal*

for x :: *'a::real-normed-algebra star*
by (*erule HFinite-star-of* [*THEN* [*2*] *Infinitesimal-HFinite-mult2*])

5.5 The Infinitely Close Relation

lemma *mem-infmal-iff*: $x \in$ *Infinitesimal* $\longleftrightarrow x \approx 0$
by (*simp add*: *Infinitesimal-def approx-def*)

lemma *approx-minus-iff*: $x \approx y \longleftrightarrow x - y \approx 0$
by (*simp add*: *approx-def*)

lemma *approx-minus-iff2*: $x \approx y \longleftrightarrow - y + x \approx 0$
by (*simp add*: *approx-def add.commute*)

lemma *approx-refl* [*iff*]: $x \approx x$
by (*simp add*: *approx-def Infinitesimal-def*)

lemma *approx-sym*: $x \approx y \Longrightarrow y \approx x$
by (*metis Infinitesimal-minus-iff approx-def minus-diff-eq*)

lemma *approx-trans*:
assumes $x \approx y$ $y \approx z$ **shows** $x \approx z$
proof −
have $x - y \in$ *Infinitesimal* $z - y \in$ *Infinitesimal*
using *assms approx-def approx-sym* **by** *auto*
then have $x - z \in$ *Infinitesimal*
using *Infinitesimal-diff* **by** *force*
then show *?thesis*
by (*simp add*: *approx-def*)
qed

lemma *approx-trans2*: $r \approx x \Longrightarrow s \approx x \Longrightarrow r \approx s$
by (*blast intro*: *approx-sym approx-trans*)

lemma *approx-trans3*: $x \approx r \Longrightarrow x \approx s \Longrightarrow r \approx s$
by (*blast intro*: *approx-sym approx-trans*)

lemma *approx-reorient*: $x \approx y \longleftrightarrow y \approx x$
by (*blast intro*: *approx-sym*)

Reorientation simplification procedure: reorients (polymorphic) $0 = x$, $1 = x$, $nnn = x$ provided x isn't 0, 1 or a numeral.

simproc-setup *approx-reorient-simproc*
($0 \approx x$ | $1 \approx y$ | *numeral* $w \approx z$ | $- 1 \approx y$ | $-$ *numeral* $w \approx r$) =
‹
let val rule = @{*thm approx-reorient*} *RS eq-reflection*
fun proc ct =
case Thm.term-of ct of
\- \$ *t* \$ *u* => *if can HOLogic.dest-number u then NONE*

```
          else if can HOLogic.dest-number t then SOME rule else NONE
      | - => NONE
  in K (K proc) end
›
```

lemma *Infinitesimal-approx-minus*: $x - y \in Infinitesimal \longleftrightarrow x \approx y$
by (*simp add*: *approx-minus-iff* [*symmetric*] *mem-infmal-iff*)

lemma *approx-monad-iff*: $x \approx y \longleftrightarrow monad\ x = monad\ y$
apply (*simp add*: *monad-def set-eq-iff*)
using *approx-reorient approx-trans* **by** *blast*

lemma *Infinitesimal-approx*: $x \in Infinitesimal \Longrightarrow y \in Infinitesimal \Longrightarrow x \approx y$
by (*simp add*: *Infinitesimal-diff approx-def*)

lemma *approx-add*: $a \approx b \Longrightarrow c \approx d \Longrightarrow a + c \approx b + d$
proof (*unfold approx-def*)
assume *inf*: $a - b \in Infinitesimal\ c - d \in Infinitesimal$
have $a + c - (b + d) = (a - b) + (c - d)$ **by** *simp*
also have $... \in Infinitesimal$
using *inf* **by** (*rule Infinitesimal-add*)
finally show $a + c - (b + d) \in Infinitesimal$ **.**
qed

lemma *approx-minus*: $a \approx b \Longrightarrow - a \approx - b$
by (*metis approx-def approx-sym minus-diff-eq minus-diff-minus*)

lemma *approx-minus2*: $- a \approx - b \Longrightarrow a \approx b$
by (*auto dest*: *approx-minus*)

lemma *approx-minus-cancel* [*simp*]: $- a \approx - b \longleftrightarrow a \approx b$
by (*blast intro*: *approx-minus approx-minus2*)

lemma *approx-add-minus*: $a \approx b \Longrightarrow c \approx d \Longrightarrow a + - c \approx b + - d$
by (*blast intro*!: *approx-add approx-minus*)

lemma *approx-diff*: $a \approx b \Longrightarrow c \approx d \Longrightarrow a - c \approx b - d$
using *approx-add* [*of* $a\ b - c - d$] **by** *simp*

lemma *approx-mult1*: $a \approx b \Longrightarrow c \in HFinite \Longrightarrow a * c \approx b * c$
for $a\ b\ c$ :: *'a::real-normed-algebra star*
by (*simp add*: *approx-def Infinitesimal-HFinite-mult left-diff-distrib* [*symmetric*])

lemma *approx-mult2*: $a \approx b \Longrightarrow c \in HFinite \Longrightarrow c * a \approx c * b$
for $a\ b\ c$ :: *'a::real-normed-algebra star*
by (*simp add*: *approx-def Infinitesimal-HFinite-mult2 right-diff-distrib* [*symmetric*])

lemma *approx-mult-subst*: $u \approx v * x \Longrightarrow x \approx y \Longrightarrow v \in HFinite \Longrightarrow u \approx v * y$
for $u\ v\ x\ y$ :: *'a::real-normed-algebra star*

by (*blast intro*: *approx-mult2 approx-trans*)

lemma *approx-mult-subst2*: $u \approx x * v \Longrightarrow x \approx y \Longrightarrow v \in HFinite \Longrightarrow u \approx y * v$
for $u\ v\ x\ y$ :: *'a::real-normed-algebra star*
by (*blast intro*: *approx-mult1 approx-trans*)

lemma *approx-mult-subst-star-of*: $u \approx x * star\text{-}of\ v \Longrightarrow x \approx y \Longrightarrow u \approx y * star\text{-}of\ v$
for $u\ x\ y$ :: *'a::real-normed-algebra star*
by (*auto intro*: *approx-mult-subst2*)

lemma *approx-eq-imp*: $a = b \Longrightarrow a \approx b$
by (*simp add*: *approx-def*)

lemma *Infinitesimal-minus-approx*: $x \in Infinitesimal \Longrightarrow - x \approx x$
by (*blast intro*: *Infinitesimal-minus-iff* [*THEN iffD2*] *mem-infmal-iff* [*THEN iffD1*] *approx-trans2*)

lemma *bex-Infinitesimal-iff*: $(\exists y \in Infinitesimal.\ x - z = y) \longleftrightarrow x \approx z$
by (*simp add*: *approx-def*)

lemma *bex-Infinitesimal-iff2*: $(\exists y \in Infinitesimal.\ x = z + y) \longleftrightarrow x \approx z$
by (*force simp add*: *bex-Infinitesimal-iff* [*symmetric*])

lemma *Infinitesimal-add-approx*: $y \in Infinitesimal \Longrightarrow x + y = z \Longrightarrow x \approx z$
using *approx-sym bex-Infinitesimal-iff2* **by** *blast*

lemma *Infinitesimal-add-approx-self*: $y \in Infinitesimal \Longrightarrow x \approx x + y$
by (*simp add*: *Infinitesimal-add-approx*)

lemma *Infinitesimal-add-approx-self2*: $y \in Infinitesimal \Longrightarrow x \approx y + x$
by (*auto dest*: *Infinitesimal-add-approx-self simp add*: *add.commute*)

lemma *Infinitesimal-add-minus-approx-self*: $y \in Infinitesimal \Longrightarrow x \approx x + - y$
by (*blast intro*!: *Infinitesimal-add-approx-self Infinitesimal-minus-iff* [*THEN iffD2*])

lemma *Infinitesimal-add-cancel*: $y \in Infinitesimal \Longrightarrow x + y \approx z \Longrightarrow x \approx z$
using *Infinitesimal-add-approx approx-trans* **by** *blast*

lemma *Infinitesimal-add-right-cancel*: $y \in Infinitesimal \Longrightarrow x \approx z + y \Longrightarrow x \approx z$
by (*metis Infinitesimal-add-approx-self approx-monad-iff*)

lemma *approx-add-left-cancel*: $d + b \approx d + c \Longrightarrow b \approx c$
by (*metis add-diff-cancel-left bex-Infinitesimal-iff*)

lemma *approx-add-right-cancel*: $b + d \approx c + d \Longrightarrow b \approx c$
by (*simp add*: *approx-def*)

lemma *approx-add-mono1*: $b \approx c \Longrightarrow d + b \approx d + c$

by (*simp add*: *approx-add*)

lemma *approx-add-mono2*: $b \approx c \Longrightarrow b + a \approx c + a$
by (*simp add*: *add.commute approx-add-mono1*)

lemma *approx-add-left-iff* [*simp*]: $a + b \approx a + c \longleftrightarrow b \approx c$
by (*fast elim*: *approx-add-left-cancel approx-add-mono1*)

lemma *approx-add-right-iff* [*simp*]: $b + a \approx c + a \longleftrightarrow b \approx c$
by (*simp add*: *add.commute*)

lemma *approx-HFinite*: $x \in HFinite \Longrightarrow x \approx y \Longrightarrow y \in HFinite$
by (*metis HFinite-add Infinitesimal-subset-HFinite approx-sym subsetD bex-Infinitesimal-iff2*)

lemma *approx-star-of-HFinite*: $x \approx$ *star-of* $D \Longrightarrow x \in HFinite$
by (*rule approx-sym* [*THEN* [*2*] *approx-HFinite*], *auto*)

lemma *approx-mult-HFinite*: $a \approx b \Longrightarrow c \approx d \Longrightarrow b \in HFinite \Longrightarrow d \in HFinite \Longrightarrow a * c \approx b * d$
for *a b c d* :: *'a::real-normed-algebra star*
by (*meson approx-HFinite approx-mult2 approx-mult-subst2 approx-sym*)

lemma *scaleHR-left-diff-distrib*: $\bigwedge a\ b\ x.$ *scaleHR* $(a - b)\ x =$ *scaleHR* $a\ x -$ *scaleHR* $b\ x$
by *transfer* (*rule scaleR-left-diff-distrib*)

lemma *approx-scaleR1*: $a \approx$ *star-of* $b \Longrightarrow c \in HFinite \Longrightarrow$ *scaleHR* $a\ c \approx b *_R c$
unfolding *approx-def*
by (*metis Infinitesimal-HFinite-scaleHR scaleHR-def scaleHR-left-diff-distrib star-scaleR-def starfun2-star-of*)

lemma *approx-scaleR2*: $a \approx b \Longrightarrow c *_R a \approx c *_R b$
by (*simp add*: *approx-def Infinitesimal-scaleR2 scaleR-right-diff-distrib* [*symmetric*])

lemma *approx-scaleR-HFinite*: $a \approx$ *star-of* $b \Longrightarrow c \approx d \Longrightarrow d \in HFinite \Longrightarrow$ *scaleHR* $a\ c \approx b *_R d$
by (*meson approx-HFinite approx-scaleR1 approx-scaleR2 approx-sym approx-trans*)

lemma *approx-mult-star-of*: $a \approx$ *star-of* $b \Longrightarrow c \approx$ *star-of* $d \Longrightarrow a * c \approx$ *star-of* $b *$ *star-of* d
for *a c* :: *'a::real-normed-algebra star*
by (*blast intro*!: *approx-mult-HFinite approx-star-of-HFinite HFinite-star-of*)

lemma *approx-SReal-mult-cancel-zero*:
fixes *a x* :: *hypreal*
assumes $a \in \mathbb{R}$ $a \neq 0$ $a * x \approx 0$ **shows** $x \approx 0$
proof −
have *inverse* $a \in HFinite$
using *Reals-inverse SReal-subset-HFinite assms*(*1*) **by** *blast*

then show *?thesis*
using *assms* **by** (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
qed

lemma *approx-mult-SReal1*: $a \in \mathbb{R} \Longrightarrow x \approx 0 \Longrightarrow x * a \approx 0$
for *a x* :: *hypreal*
by (*auto dest*: *SReal-subset-HFinite* [*THEN subsetD*] *approx-mult1*)

lemma *approx-mult-SReal2*: $a \in \mathbb{R} \Longrightarrow x \approx 0 \Longrightarrow a * x \approx 0$
for *a x* :: *hypreal*
by (*auto dest*: *SReal-subset-HFinite* [*THEN subsetD*] *approx-mult2*)

lemma *approx-mult-SReal-zero-cancel-iff* [*simp*]: $a \in \mathbb{R} \Longrightarrow a \neq 0 \Longrightarrow a * x \approx 0 \longleftrightarrow x \approx 0$
for *a x* :: *hypreal*
by (*blast intro*: *approx-SReal-mult-cancel-zero approx-mult-SReal2*)

lemma *approx-SReal-mult-cancel*:
fixes *a w z* :: *hypreal*
assumes $a \in \mathbb{R}$ $a \neq 0$ $a * w \approx a * z$ **shows** $w \approx z$
proof −
have *inverse a* ∈ *HFinite*
using *Reals-inverse SReal-subset-HFinite assms*(*1*) **by** *blast*
then show *?thesis*
using *assms* **by** (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
qed

lemma *approx-SReal-mult-cancel-iff1* [*simp*]: $a \in \mathbb{R} \Longrightarrow a \neq 0 \Longrightarrow a * w \approx a * z \longleftrightarrow w \approx z$
for *a w z* :: *hypreal*
by (*meson SReal-subset-HFinite approx-SReal-mult-cancel approx-mult2 subsetD*)

lemma *approx-le-bound*:
fixes *z* :: *hypreal*
assumes $z \le f$ $f \approx g$ $g \le z$ **shows** $f \approx z$
proof −
obtain *y* **where** $z \le g + y$ **and** $y \in$ *Infinitesimal* $f = g + y$
using *assms bex-Infinitesimal-iff2* **by** *auto*
then have $z - g \in$ *Infinitesimal*
using *assms*(*3*) *hrabs-le-Infinitesimal* **by** *auto*
then show *?thesis*
by (*metis approx-def approx-trans2 assms*(*2*))
qed

lemma *approx-hnorm*: $x \approx y \Longrightarrow$ *hnorm* $x \approx$ *hnorm* y
for *x y* :: *'a*::*real-normed-vector star*
proof (*unfold approx-def*)
assume $x - y \in$ *Infinitesimal*
then have *hnorm* $(x - y) \in$ *Infinitesimal*

by (*simp only*: *Infinitesimal-hnorm-iff*)
moreover have (*0*::*real star*) ∈ *Infinitesimal*
by (*rule Infinitesimal-zero*)
moreover have *0* ≤ |*hnorm x* − *hnorm y*|
by (*rule abs-ge-zero*)
moreover have |*hnorm x* − *hnorm y*| ≤ *hnorm* (*x* − *y*)
by (*rule hnorm-triangle-ineq3*)
ultimately have |*hnorm x* − *hnorm y*| ∈ *Infinitesimal*
by (*rule Infinitesimal-interval2*)
then show *hnorm x* − *hnorm y* ∈ *Infinitesimal*
by (*simp only*: *Infinitesimal-hrabs-iff*)
qed

5.6 Zero is the Only Infinitesimal that is also a Real

lemma *Infinitesimal-less-SReal*: *x* ∈ ℝ ⟹ *y* ∈ *Infinitesimal* ⟹ *0* < *x* ⟹ *y* < *x*
for *x y* :: *hypreal*
using *InfinitesimalD* **by** *fastforce*

lemma *Infinitesimal-less-SReal2*: *y* ∈ *Infinitesimal* ⟹ ∀ *r* ∈ *Reals*. *0* < *r* ⟶ *y* < *r*
for *y* :: *hypreal*
by (*blast intro*: *Infinitesimal-less-SReal*)

lemma *SReal-not-Infinitesimal*: *0* < *y* ⟹ *y* ∈ ℝ ==> *y* ∉ *Infinitesimal*
for *y* :: *hypreal*
by (*auto simp add*: *Infinitesimal-def abs-if*)

lemma *SReal-minus-not-Infinitesimal*: *y* < *0* ⟹ *y* ∈ ℝ ⟹ *y* ∉ *Infinitesimal*
for *y* :: *hypreal*
using *Infinitesimal-minus-iff Reals-minus SReal-not-Infinitesimal neg-0-less-iff-less*
by *blast*

lemma *SReal-Int-Infinitesimal-zero*: ℝ *Int Infinitesimal* = {*0*::*hypreal*}
proof −
have *x* = *0* **if** *x* ∈ ℝ *x* ∈ *Infinitesimal* **for** *x* :: *real star*
using *that SReal-minus-not-Infinitesimal SReal-not-Infinitesimal not-less-iff-gr-or-eq*
by *blast*
then show *?thesis*
by *auto*
qed

lemma *SReal-Infinitesimal-zero*: *x* ∈ ℝ ⟹ *x* ∈ *Infinitesimal* ⟹ *x* = *0*
for *x* :: *hypreal*
using *SReal-Int-Infinitesimal-zero* **by** *blast*

lemma *SReal-HFinite-diff-Infinitesimal*: *x* ∈ ℝ ⟹ *x* ≠ *0* ⟹ *x* ∈ *HFinite* − *Infinitesimal*

for *x* :: *hypreal*
by (*auto dest*: *SReal-Infinitesimal-zero SReal-subset-HFinite* [*THEN subsetD*])

lemma *hypreal-of-real-HFinite-diff-Infinitesimal*:
hypreal-of-real x ≠ *0* ⟹ *hypreal-of-real x* ∈ *HFinite* − *Infinitesimal*
by (*rule SReal-HFinite-diff-Infinitesimal*) *auto*

lemma *star-of-Infinitesimal-iff-0* [*iff*]: *star-of x* ∈ *Infinitesimal* ⟷ *x* = *0*
proof
show *x* = *0* **if** *star-of x* ∈ *Infinitesimal*
proof −
have *hnorm* (*star-n* (λ*n*. *x*)) ∈ *Standard*
by (*metis Reals-eq-Standard SReal-iff star-of-def star-of-norm*)
then show *?thesis*
by (*metis InfinitesimalD2 less-irrefl star-of-norm that zero-less-norm-iff*)
qed
qed *auto*

lemma *star-of-HFinite-diff-Infinitesimal*: *x* ≠ *0* ⟹ *star-of x* ∈ *HFinite* − *Infinitesimal*
by *simp*

lemma *numeral-not-Infinitesimal* [*simp*]:
numeral w ≠ (*0*::*hypreal*) ⟹ (*numeral w* :: *hypreal*) ∉ *Infinitesimal*
by (*fast dest*: *Reals-numeral* [*THEN SReal-Infinitesimal-zero*])

Again: *1* is a special case, but not *0* this time.

lemma *one-not-Infinitesimal* [*simp*]:
(*1*::′*a*::{*real-normed-vector*,*zero-neq-one*} *star*) ∉ *Infinitesimal*
by (*metis star-of-Infinitesimal-iff-0 star-one-def zero-neq-one*)

lemma *approx-SReal-not-zero*: *y* ∈ ℝ ⟹ *x* ≈ *y* ⟹ *y* ≠ *0* ⟹ *x* ≠ *0*
for *x y* :: *hypreal*
using *SReal-Infinitesimal-zero approx-sym mem-infmal-iff* **by** *auto*

lemma *HFinite-diff-Infinitesimal-approx*:
x ≈ *y* ⟹ *y* ∈ *HFinite* − *Infinitesimal* ⟹ *x* ∈ *HFinite* − *Infinitesimal*
by (*meson Diff-iff approx-HFinite approx-sym approx-trans3 mem-infmal-iff*)

The premise *y* ≠ *0* is essential; otherwise *x* / *y* = *0* and we lose the *HFinite* premise.

lemma *Infinitesimal-ratio*:
y ≠ *0* ⟹ *y* ∈ *Infinitesimal* ⟹ *x* / *y* ∈ *HFinite* ⟹ *x* ∈ *Infinitesimal*
for *x y* :: ′*a*::{*real-normed-div-algebra*,*field*} *star*
using *Infinitesimal-HFinite-mult* **by** *fastforce*

lemma *Infinitesimal-SReal-divide*: *x* ∈ *Infinitesimal* ⟹ *y* ∈ ℝ ⟹ *x* / *y* ∈ *Infinitesimal*
for *x y* :: *hypreal*

by (*metis HFinite-star-of Infinitesimal-HFinite-mult Reals-inverse SReal-iff divide-inverse*)

6 Standard Part Theorem

Every finite $x \in R*$ is infinitely close to a unique real number (i.e. a member of *Reals*).

6.1 Uniqueness: Two Infinitely Close Reals are Equal

lemma *star-of-approx-iff* [*simp*]: *star-of* $x \approx$ *star-of* $y \longleftrightarrow x = y$
by (*metis approx-def right-minus-eq star-of-Infinitesimal-iff-0 star-of-simps*(*2*))

lemma *SReal-approx-iff*: $x \in \mathbb{R} \Longrightarrow y \in \mathbb{R} \Longrightarrow x \approx y \longleftrightarrow x = y$
for x y :: *hypreal*
by (*meson Reals-diff SReal-Infinitesimal-zero approx-def approx-refl right-minus-eq*)

lemma *numeral-approx-iff* [*simp*]:
(*numeral* $v \approx$ (*numeral* w :: *'a*::{*numeral*,*real-normed-vector*} *star*)) = (*numeral* v = (*numeral* w :: *'a*))
by (*metis star-of-approx-iff star-of-numeral*)

And also for $0 \approx \#nn$ and $1 \approx \#nn$, $\#nn \approx 0$ and $\#nn \approx 1$.

lemma [*simp*]:
(*numeral* $w \approx$ (*0*::*'a*::{*numeral*,*real-normed-vector*} *star*)) = (*numeral* w = (*0*::*'a*))
((*0*::*'a*::{*numeral*,*real-normed-vector*} *star*) $\approx$ *numeral* w) = (*numeral* w = (*0*::*'a*))
(*numeral* $w \approx$ (*1*::*'b*::{*numeral*,*one*,*real-normed-vector*} *star*)) = (*numeral* w = (*1*::*'b*))
((*1*::*'b*::{*numeral*,*one*,*real-normed-vector*} *star*) $\approx$ *numeral* w) = (*numeral* w = (*1*::*'b*))
$\neg$ (*0* $\approx$ (*1*::*'c*::{*zero-neq-one*,*real-normed-vector*} *star*))
$\neg$ (*1* $\approx$ (*0*::*'c*::{*zero-neq-one*,*real-normed-vector*} *star*))
unfolding *star-numeral-def star-zero-def star-one-def star-of-approx-iff*
by (*auto intro*: *sym*)

lemma *star-of-approx-numeral-iff* [*simp*]: *star-of* $k \approx$ *numeral* $w \longleftrightarrow k$ = *numeral* w
by (*subst star-of-approx-iff* [*symmetric*]) *auto*

lemma *star-of-approx-zero-iff* [*simp*]: *star-of* $k \approx 0 \longleftrightarrow k = 0$
by (*simp-all add*: *star-of-approx-iff* [*symmetric*])

lemma *star-of-approx-one-iff* [*simp*]: *star-of* $k \approx 1 \longleftrightarrow k = 1$
by (*simp-all add*: *star-of-approx-iff* [*symmetric*])

lemma *approx-unique-real*: $r \in \mathbb{R} \Longrightarrow s \in \mathbb{R} \Longrightarrow r \approx x \Longrightarrow s \approx x \Longrightarrow r = s$
for r s :: *hypreal*
by (*blast intro*: *SReal-approx-iff* [*THEN iffD1*] *approx-trans2*)

6.2 Existence of Unique Real Infinitely Close

6.2.1 Lifting of the Ub and Lub Properties

lemma *hypreal-of-real-isUb-iff*: *isUb* ℝ (*hypreal-of-real* ' *Q*) (*hypreal-of-real* *Y*) = *isUb UNIV Q Y*
for *Q* :: *real set* **and** *Y* :: *real*
by (*simp add*: *isUb-def setle-def*)

lemma *hypreal-of-real-isLub-iff*:
isLub ℝ (*hypreal-of-real* ' *Q*) (*hypreal-of-real* *Y*) = *isLub* (*UNIV* :: *real set*) *Q Y*
(**is** *?lhs* = *?rhs*)
for *Q* :: *real set* **and** *Y* :: *real*
proof
assume *?lhs*
then show *?rhs*
by (*simp add*: *isLub-def leastP-def*) (*metis hypreal-of-real-isUb-iff mem-Collect-eq setge-def star-of-le*)
next
assume *?rhs*
then show *?lhs*
apply (*simp add*: *isLub-def leastP-def hypreal-of-real-isUb-iff setge-def*)
by (*metis SReal-iff hypreal-of-real-isUb-iff isUb-def star-of-le*)
qed

lemma *lemma-isUb-hypreal-of-real*: *isUb* ℝ *P Y* ⟹ ∃ *Yo*. *isUb* ℝ *P* (*hypreal-of-real Yo*)
by (*auto simp add*: *SReal-iff isUb-def*)

lemma *lemma-isLub-hypreal-of-real*: *isLub* ℝ *P Y* ⟹ ∃ *Yo*. *isLub* ℝ *P* (*hypreal-of-real Yo*)
by (*auto simp add*: *isLub-def leastP-def isUb-def SReal-iff*)

lemma *SReal-complete*:
fixes *P* :: *hypreal set*
assumes *isUb* ℝ *P Y* *P* ⊆ ℝ *P* ≠ {}
shows ∃ *t*. *isLub* ℝ *P t*
proof −
obtain *Q* **where** *P* = *hypreal-of-real* ' *Q*
by (*metis* ‹*P* ⊆ ℝ› *hypreal-of-real-image subset-imageE*)
then show *?thesis*
by (*metis assms*(*1*) ‹*P* ≠ {}› *equals0I hypreal-of-real-isLub-iff hypreal-of-real-isUb-iff image-empty lemma-isUb-hypreal-of-real reals-complete*)
qed

Lemmas about lubs.

lemma *lemma-st-part-lub*:
fixes *x* :: *hypreal*
assumes *x* ∈ *HFinite*
shows ∃ *t*. *isLub* ℝ {*s*. *s* ∈ ℝ ∧ *s* < *x*} *t*

proof −
  obtain t **where** t: $t \in \mathbb{R}$ *hnorm* $x < t$
    using *HFiniteD assms* **by** *blast*
  then have *isUb* $\mathbb{R}$ $\{s.\ s \in \mathbb{R} \wedge s < x\}$ t
    by (*simp add*: *abs-less-iff isUbI le-less-linear less-imp-not-less setleI*)
  moreover have $\exists\, y.\ y \in \mathbb{R} \wedge y < x$
    using t **by** (*rule-tac* $x = -t$ **in** *exI*) (*auto simp add*: *abs-less-iff*)
  ultimately show *?thesis*
    using *SReal-complete* **by** *fastforce*
qed

lemma *hypreal-setle-less-trans*: $S *\!<= x \Longrightarrow x < y \Longrightarrow S *\!<= y$
  for $x\ y$ :: *hypreal*
  by (*meson le-less-trans less-imp-le setle-def*)

lemma *hypreal-gt-isUb*: *isUb* $R\ S\ x \Longrightarrow x < y \Longrightarrow y \in R \Longrightarrow$ *isUb* $R\ S\ y$
  for $x\ y$ :: *hypreal*
  using *hypreal-setle-less-trans isUb-def* **by** *blast*

lemma *lemma-SReal-ub*: $x \in \mathbb{R} \Longrightarrow$ *isUb* $\mathbb{R}$ $\{s.\ s \in \mathbb{R} \wedge s < x\}$ x
  for x :: *hypreal*
  by (*auto intro*: *isUbI setleI order-less-imp-le*)

lemma *lemma-SReal-lub*:
  fixes x :: *hypreal*
  assumes $x \in \mathbb{R}$ **shows** *isLub* $\mathbb{R}$ $\{s.\ s \in \mathbb{R} \wedge s < x\}$ x
proof −
  have $x \le y$ **if** *isUb* $\mathbb{R}$ $\{s \in \mathbb{R}.\ s < x\}$ y **for** y
  proof −
    have $y \in \mathbb{R}$
      using *isUbD2a that* **by** *blast*
    show *?thesis*
    proof (*cases* $x\ y$ *rule*: *linorder-cases*)
      case *greater*
      then obtain r **where** $y < r\ r < x$
        using *dense* **by** *blast*
      then show *?thesis*
        using *isUbD* [*OF that*]
        by *simp* (*meson SReal-dense* ‹$y \in \mathbb{R}$› *assms greater not-le*)
    qed *auto*
  qed
  with *assms* **show** *?thesis*
    by (*simp add*: *isLubI2 isUbI setgeI setleI*)
qed

lemma *lemma-st-part-major*:
  fixes $x\ r\ t$ :: *hypreal*
  assumes x: $x \in$ *HFinite* **and** r: $r \in \mathbb{R}$ $0 < r$ **and** t: *isLub* $\mathbb{R}$ $\{s.\ s \in \mathbb{R} \wedge s < x\}$ t

shows $|x - t| < r$
proof $-$
have $t \in \mathbb{R}$
using *isLubD1a t* **by** *blast*
have *lemma-st-part-gt-ub*: $x < r \Longrightarrow r \in \mathbb{R} \Longrightarrow$ *isUb* $\mathbb{R}$ $\{s.\ s \in \mathbb{R} \wedge s < x\}$ r
for r :: *hypreal*
by (*auto dest*: *order-less-trans intro*: *order-less-imp-le intro*!: *isUbI setleI*)

have *isUb* $\mathbb{R}$ $\{s \in \mathbb{R}.\ s < x\}$ t
by (*simp add*: *t isLub-isUb*)
then have $\neg\ r + t < x$
by (*metis* (*mono-tags*, *lifting*) *Reals-add* ‹$t \in \mathbb{R}$› *add-le-same-cancel2 isUbD leD mem-Collect-eq r*)
then have $x - t \leq r$
by *simp*
moreover have $\neg\ x < t - r$
using *lemma-st-part-gt-ub isLub-le-isUb* ‹$t \in \mathbb{R}$› *r t x* **by** *fastforce*
then have $-\ (x - t) \leq r$
by *linarith*
moreover have *False* **if** $x - t = r \vee -\ (x - t) = r$
proof $-$
have $x \in \mathbb{R}$
by (*metis* ‹$t \in \mathbb{R}$› ‹$r \in \mathbb{R}$› *that Reals-add-cancel Reals-minus-iff add-uminus-conv-diff*)
then have *isLub* $\mathbb{R}$ $\{s \in \mathbb{R}.\ s < x\}$ x
by (*rule lemma-SReal-lub*)
then show *False*
using *r t that x isLub-unique* **by** *force*
qed
ultimately show *?thesis*
using *abs-less-iff dual-order.order-iff-strict* **by** *blast*
qed

lemma *lemma-st-part-major2*:
$x \in$ *HFinite* $\Longrightarrow$ *isLub* $\mathbb{R}$ $\{s.\ s \in \mathbb{R} \wedge s < x\}$ $t \Longrightarrow \forall r \in$ *Reals*. $0 < r \longrightarrow |x - t| < r$
for $x\ t$ :: *hypreal*
by (*blast dest*!: *lemma-st-part-major*)

Existence of real and Standard Part Theorem.

lemma *lemma-st-part-Ex*: $x \in$ *HFinite* $\Longrightarrow \exists\, t\in$*Reals*. $\forall r \in$ *Reals*. $0 < r \longrightarrow |x - t| < r$
for x :: *hypreal*
by (*meson isLubD1a lemma-st-part-lub lemma-st-part-major2*)

lemma *st-part-Ex*: $x \in$ *HFinite* $\Longrightarrow \exists\, t\in$*Reals*. $x \approx t$
for x :: *hypreal*
by (*metis InfinitesimalI approx-def hypreal-hnorm-def lemma-st-part-Ex*)

There is a unique real infinitely close.

lemma *st-part-Ex1*: $x \in \mathit{HFinite} \Longrightarrow \exists!t{::}\mathit{hypreal}.\ t \in \mathbb{R} \wedge x \approx t$
by (*meson SReal-approx-iff approx-trans2 st-part-Ex*)

6.3 Finite, Infinite and Infinitesimal

lemma *HFinite-Int-HInfinite-empty* [*simp*]: $\mathit{HFinite}\ \mathit{Int}\ \mathit{HInfinite} = \{\}$
using *Compl-HFinite* **by** *blast*

lemma *HFinite-not-HInfinite*:
assumes x: $x \in \mathit{HFinite}$ **shows** $x \notin \mathit{HInfinite}$
using *Compl-HFinite* x **by** *blast*

lemma *not-HFinite-HInfinite*: $x \notin \mathit{HFinite} \Longrightarrow x \in \mathit{HInfinite}$
using *Compl-HFinite* **by** *blast*

lemma *HInfinite-HFinite-disj*: $x \in \mathit{HInfinite} \vee x \in \mathit{HFinite}$
by (*blast intro*: *not-HFinite-HInfinite*)

lemma *HInfinite-HFinite-iff*: $x \in \mathit{HInfinite} \longleftrightarrow x \notin \mathit{HFinite}$
by (*blast dest*: *HFinite-not-HInfinite not-HFinite-HInfinite*)

lemma *HFinite-HInfinite-iff*: $x \in \mathit{HFinite} \longleftrightarrow x \notin \mathit{HInfinite}$
by (*simp add*: *HInfinite-HFinite-iff*)

lemma *HInfinite-diff-HFinite-Infinitesimal-disj*:
$x \notin \mathit{Infinitesimal} \Longrightarrow x \in \mathit{HInfinite} \vee x \in \mathit{HFinite} - \mathit{Infinitesimal}$
by (*fast intro*: *not-HFinite-HInfinite*)

lemma *HFinite-inverse*: $x \in \mathit{HFinite} \Longrightarrow x \notin \mathit{Infinitesimal} \Longrightarrow \mathit{inverse}\ x \in \mathit{HFinite}$
for x :: *'a::real-normed-div-algebra star*
using *HInfinite-inverse-Infinitesimal not-HFinite-HInfinite* **by** *force*

lemma *HFinite-inverse2*: $x \in \mathit{HFinite} - \mathit{Infinitesimal} \Longrightarrow \mathit{inverse}\ x \in \mathit{HFinite}$
for x :: *'a::real-normed-div-algebra star*
by (*blast intro*: *HFinite-inverse*)

Stronger statement possible in fact.

lemma *Infinitesimal-inverse-HFinite*: $x \notin \mathit{Infinitesimal} \Longrightarrow \mathit{inverse}\ x \in \mathit{HFinite}$
for x :: *'a::real-normed-div-algebra star*
using *HFinite-HInfinite-iff HInfinite-inverse-Infinitesimal* **by** *fastforce*

lemma *HFinite-not-Infinitesimal-inverse*:
$x \in \mathit{HFinite} - \mathit{Infinitesimal} \Longrightarrow \mathit{inverse}\ x \in \mathit{HFinite} - \mathit{Infinitesimal}$
for x :: *'a::real-normed-div-algebra star*
using *HFinite-Infinitesimal-not-zero HFinite-inverse2 Infinitesimal-HFinite-mult2* **by** *fastforce*

lemma *approx-inverse*:

fixes *x y* :: *'a::real-normed-div-algebra star*
assumes *x* ≈ *y* **and** *y*: *y* ∈ *HFinite* − *Infinitesimal* **shows** *inverse x* ≈ *inverse y*
proof −
have *x*: *x* ∈ *HFinite* − *Infinitesimal*
using *HFinite-diff-Infinitesimal-approx assms*(*1*) *y* **by** *blast*
with *y HFinite-inverse2* **have** *inverse x* ∈ *HFinite inverse y* ∈ *HFinite*
by *blast+*
then have *inverse y* ∗ *x* ≈ *1*
by (*metis Diff-iff approx-mult2 assms*(*1*) *left-inverse not-Infinitesimal-not-zero y*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *DiffD2 HFinite-Infinitesimal-not-zero Infinitesimal-mult-disj x approx-def approx-sym left-diff-distrib left-inverse*)
qed

lemmas *star-of-approx-inverse* = *star-of-HFinite-diff-Infinitesimal* [*THEN* [*2*] *approx-inverse*]
lemmas *hypreal-of-real-approx-inverse* = *hypreal-of-real-HFinite-diff-Infinitesimal* [*THEN* [*2*] *approx-inverse*]

lemma *inverse-add-Infinitesimal-approx*:
x ∈ *HFinite* − *Infinitesimal* ⟹ *h* ∈ *Infinitesimal* ⟹ *inverse* (*x* + *h*) ≈ *inverse x*
for *x h* :: *'a::real-normed-div-algebra star*
by (*auto intro*: *approx-inverse approx-sym Infinitesimal-add-approx-self*)

lemma *inverse-add-Infinitesimal-approx2*:
x ∈ *HFinite* − *Infinitesimal* ⟹ *h* ∈ *Infinitesimal* ⟹ *inverse* (*h* + *x*) ≈ *inverse x*
for *x h* :: *'a::real-normed-div-algebra star*
by (*metis add.commute inverse-add-Infinitesimal-approx*)

lemma *inverse-add-Infinitesimal-approx-Infinitesimal*:
x ∈ *HFinite* − *Infinitesimal* ⟹ *h* ∈ *Infinitesimal* ⟹ *inverse* (*x* + *h*) − *inverse x* ≈ *h*
for *x h* :: *'a::real-normed-div-algebra star*
by (*meson Infinitesimal-approx bex-Infinitesimal-iff inverse-add-Infinitesimal-approx*)

lemma *Infinitesimal-square-iff*: *x* ∈ *Infinitesimal* ⟷ *x* ∗ *x* ∈ *Infinitesimal*
for *x* :: *'a::real-normed-div-algebra star*
using *Infinitesimal-mult Infinitesimal-mult-disj* **by** *auto*
declare *Infinitesimal-square-iff* [*symmetric*, *simp*]

lemma *HFinite-square-iff* [*simp*]: *x* ∗ *x* ∈ *HFinite* ⟷ *x* ∈ *HFinite*
for *x* :: *'a::real-normed-div-algebra star*
using *HFinite-HInfinite-iff HFinite-mult HInfinite-mult* **by** *blast*

lemma *HInfinite-square-iff* [*simp*]: $x * x \in$ *HInfinite* $\longleftrightarrow x \in$ *HInfinite*
for x :: $'a$::*real-normed-div-algebra star*
by (*auto simp add*: *HInfinite-HFinite-iff*)

lemma *approx-HFinite-mult-cancel*: $a \in$ *HFinite* $-$ *Infinitesimal* $\Longrightarrow a * w \approx a * z \Longrightarrow w \approx z$
for $a\ w\ z$ :: $'a$::*real-normed-div-algebra star*
by (*metis DiffD2 Infinitesimal-mult-disj bex-Infinitesimal-iff right-diff-distrib*)

lemma *approx-HFinite-mult-cancel-iff1*: $a \in$ *HFinite* $-$ *Infinitesimal* $\Longrightarrow a * w \approx a * z \longleftrightarrow w \approx z$
for $a\ w\ z$ :: $'a$::*real-normed-div-algebra star*
by (*auto intro*: *approx-mult2 approx-HFinite-mult-cancel*)

lemma *HInfinite-HFinite-add-cancel*: $x + y \in$ *HInfinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow x \in$ *HInfinite*
using *HFinite-add HInfinite-HFinite-iff* **by** *blast*

lemma *HInfinite-HFinite-add*: $x \in$ *HInfinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow x + y \in$ *HInfinite*
by (*metis* (*no-types*, *opaque-lifting*) *HFinite-HInfinite-iff HFinite-add HFinite-minus-iff add.commute add-minus-cancel*)

lemma *HInfinite-ge-HInfinite*: $x \in$ *HInfinite* $\Longrightarrow x \le y \Longrightarrow 0 \le x \Longrightarrow y \in$ *HInfinite*
for $x\ y$ :: *hypreal*
by (*auto intro*: *HFinite-bounded simp add*: *HInfinite-HFinite-iff*)

lemma *Infinitesimal-inverse-HInfinite*: $x \in$ *Infinitesimal* $\Longrightarrow x \neq 0 \Longrightarrow$ *inverse* $x \in$ *HInfinite*
for x :: $'a$::*real-normed-div-algebra star*
by (*metis Infinitesimal-HFinite-mult not-HFinite-HInfinite one-not-Infinitesimal right-inverse*)

lemma *HInfinite-HFinite-not-Infinitesimal-mult*:
$x \in$ *HInfinite* $\Longrightarrow y \in$ *HFinite* $-$ *Infinitesimal* $\Longrightarrow x * y \in$ *HInfinite*
for $x\ y$ :: $'a$::*real-normed-div-algebra star*
by (*metis* (*no-types*, *opaque-lifting*) *HFinite-HInfinite-iff HFinite-Infinitesimal-not-zero HFinite-inverse2 HFinite-mult mult.assoc mult.right-neutral right-inverse*)

lemma *HInfinite-HFinite-not-Infinitesimal-mult2*:
$x \in$ *HInfinite* $\Longrightarrow y \in$ *HFinite* $-$ *Infinitesimal* $\Longrightarrow y * x \in$ *HInfinite*
for $x\ y$ :: $'a$::*real-normed-div-algebra star*
by (*metis Diff-iff HInfinite-HFinite-iff HInfinite-inverse-Infinitesimal Infinitesimal-HFinite-mult2 divide-inverse mult-zero-right nonzero-eq-divide-eq*)

lemma *HInfinite-gt-SReal*: $x \in$ *HInfinite* $\Longrightarrow 0 < x \Longrightarrow y \in \mathbb{R} \Longrightarrow y < x$
for $x\ y$ :: *hypreal*
by (*auto dest*!: *bspec simp add*: *HInfinite-def abs-if order-less-imp-le*)

lemma *HInfinite-gt-zero-gt-one*: $x \in HInfinite \Longrightarrow 0 < x \Longrightarrow 1 < x$
for x :: *hypreal*
by (*auto intro*: *HInfinite-gt-SReal*)

lemma *not-HInfinite-one* [*simp*]: $1 \notin HInfinite$
by (*simp add*: *HInfinite-HFinite-iff*)

lemma *approx-hrabs-disj*: $|x| \approx x \vee |x| \approx -x$
for x :: *hypreal*
by (*simp add*: *abs-if*)

6.4 Theorems about Monads

lemma *monad-hrabs-Un-subset*: *monad* $|x| \leq$ *monad* $x \cup$ *monad* $(- x)$
for x :: *hypreal*
by (*simp add*: *abs-if*)

lemma *Infinitesimal-monad-eq*: $e \in Infinitesimal \Longrightarrow$ *monad* $(x + e) =$ *monad* x
by (*fast intro*!: *Infinitesimal-add-approx-self* [*THEN approx-sym*] *approx-monad-iff* [*THEN iffD1*])

lemma *mem-monad-iff*: $u \in$ *monad* $x \longleftrightarrow - u \in$ *monad* $(- x)$
by (*simp add*: *monad-def*)

lemma *Infinitesimal-monad-zero-iff*: $x \in Infinitesimal \longleftrightarrow x \in$ *monad* 0
by (*auto intro*: *approx-sym simp add*: *monad-def mem-infmal-iff*)

lemma *monad-zero-minus-iff*: $x \in$ *monad* $0 \longleftrightarrow - x \in$ *monad* 0
by (*simp add*: *Infinitesimal-monad-zero-iff* [*symmetric*])

lemma *monad-zero-hrabs-iff*: $x \in$ *monad* $0 \longleftrightarrow |x| \in$ *monad* 0
for x :: *hypreal*
using *Infinitesimal-monad-zero-iff* **by** *blast*

lemma *mem-monad-self* [*simp*]: $x \in$ *monad* x
by (*simp add*: *monad-def*)

6.5 Proof that $x \approx y$ implies $|x| \approx |y|$

lemma *approx-subset-monad*: $x \approx y \Longrightarrow \{x, y\} \leq$ *monad* x
by (*simp* (*no-asm*)) (*simp add*: *approx-monad-iff*)

lemma *approx-subset-monad2*: $x \approx y \Longrightarrow \{x, y\} \leq$ *monad* y
using *approx-subset-monad approx-sym* **by** *auto*

lemma *mem-monad-approx*: $u \in$ *monad* $x \Longrightarrow x \approx u$
by (*simp add*: *monad-def*)

lemma *approx-mem-monad*: $x \approx u \Longrightarrow u \in$ *monad* x

by (*simp add*: *monad-def*)

lemma *approx-mem-monad2*: $x \approx u \Longrightarrow x \in monad\ u$
using *approx-mem-monad approx-sym* **by** *blast*

lemma *approx-mem-monad-zero*: $x \approx y \Longrightarrow x \in monad\ 0 \Longrightarrow y \in monad\ 0$
using *approx-trans monad-def* **by** *blast*

lemma *Infinitesimal-approx-hrabs*: $x \approx y \Longrightarrow x \in Infinitesimal \Longrightarrow |x| \approx |y|$
for $x\ y$ :: *hypreal*
using *approx-hnorm* **by** *fastforce*

lemma *less-Infinitesimal-less*: $0 < x \Longrightarrow x \notin Infinitesimal \Longrightarrow e \in Infinitesimal \Longrightarrow e < x$
for x :: *hypreal*
using *Infinitesimal-interval less-linear* **by** *blast*

lemma *Ball-mem-monad-gt-zero*: $0 < x \Longrightarrow x \notin Infinitesimal \Longrightarrow u \in monad\ x \Longrightarrow 0 < u$
for $u\ x$ :: *hypreal*
by (*metis bex-Infinitesimal-iff2 less-Infinitesimal-less less-add-same-cancel2 mem-monad-approx*)

lemma *Ball-mem-monad-less-zero*: $x < 0 \Longrightarrow x \notin Infinitesimal \Longrightarrow u \in monad\ x \Longrightarrow u < 0$
for $u\ x$ :: *hypreal*
by (*metis Ball-mem-monad-gt-zero approx-monad-iff less-asym linorder-neqE-linordered-idom mem-infmal-iff mem-monad-approx mem-monad-self*)

lemma *lemma-approx-gt-zero*: $0 < x \Longrightarrow x \notin Infinitesimal \Longrightarrow x \approx y \Longrightarrow 0 < y$
for $x\ y$ :: *hypreal*
by (*blast dest*: *Ball-mem-monad-gt-zero approx-subset-monad*)

lemma *lemma-approx-less-zero*: $x < 0 \Longrightarrow x \notin Infinitesimal \Longrightarrow x \approx y \Longrightarrow y < 0$
for $x\ y$ :: *hypreal*
by (*blast dest*: *Ball-mem-monad-less-zero approx-subset-monad*)

lemma *approx-hrabs*: $x \approx y \Longrightarrow |x| \approx |y|$
for $x\ y$ :: *hypreal*
by (*drule approx-hnorm*) *simp*

lemma *approx-hrabs-zero-cancel*: $|x| \approx 0 \Longrightarrow x \approx 0$
for x :: *hypreal*
using *mem-infmal-iff* **by** *blast*

lemma *approx-hrabs-add-Infinitesimal*: $e \in Infinitesimal \Longrightarrow |x| \approx |x + e|$
for $e\ x$ :: *hypreal*
by (*fast intro*: *approx-hrabs Infinitesimal-add-approx-self*)

lemma *approx-hrabs-add-minus-Infinitesimal*: $e \in$ *Infinitesimal* $==> |x| \approx |x + -e|$
for $e\ x$ :: *hypreal*
by (*fast intro*: *approx-hrabs Infinitesimal-add-minus-approx-self*)

lemma *hrabs-add-Infinitesimal-cancel*:
$e \in$ *Infinitesimal* $\Longrightarrow e' \in$ *Infinitesimal* $\Longrightarrow |x + e| = |y + e'| \Longrightarrow |x| \approx |y|$
for $e\ e'\ x\ y$ :: *hypreal*
by (*metis approx-hrabs-add-Infinitesimal approx-trans2*)

lemma *hrabs-add-minus-Infinitesimal-cancel*:
$e \in$ *Infinitesimal* $\Longrightarrow e' \in$ *Infinitesimal* $\Longrightarrow |x + -e| = |y + -e'| \Longrightarrow |x| \approx |y|$
for $e\ e'\ x\ y$ :: *hypreal*
by (*meson Infinitesimal-minus-iff hrabs-add-Infinitesimal-cancel*)

6.6 More *HFinite* and *Infinitesimal* Theorems

Interesting slightly counterintuitive theorem: necessary for proving that an open interval is an NS open set.

lemma *Infinitesimal-add-hypreal-of-real-less*:
assumes $x < y$ **and** u: $u \in$ *Infinitesimal*
shows *hypreal-of-real* $x + u <$ *hypreal-of-real* y
proof −
have $|u| <$ *hypreal-of-real* $y -$ *hypreal-of-real* x
using *InfinitesimalD* ‹$x < y$› u **by** *fastforce*
then show *?thesis*
by (*simp add*: *abs-less-iff*)
qed

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less*:
$x \in$ *Infinitesimal* $\Longrightarrow$ |*hypreal-of-real* r| $<$ *hypreal-of-real* $y \Longrightarrow$
|*hypreal-of-real* $r + x$| $<$ *hypreal-of-real* y
by (*metis Infinitesimal-add-hypreal-of-real-less approx-hrabs-add-Infinitesimal approx-sym bex-Infinitesimal-iff2 star-of-abs star-of-less*)

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less2*:
$x \in$ *Infinitesimal* $\Longrightarrow$ |*hypreal-of-real* r| $<$ *hypreal-of-real* $y \Longrightarrow$
|$x +$ *hypreal-of-real* r| $<$ *hypreal-of-real* y
using *Infinitesimal-add-hrabs-hypreal-of-real-less* **by** *fastforce*

lemma *hypreal-of-real-le-add-Infininitesimal-cancel*:
assumes *le*: *hypreal-of-real* $x + u \le$ *hypreal-of-real* $y + v$
and u: $u \in$ *Infinitesimal* **and** v: $v \in$ *Infinitesimal*
shows *hypreal-of-real* $x \le$ *hypreal-of-real* y
proof (*rule ccontr*)
assume ¬ *hypreal-of-real* $x \le$ *hypreal-of-real* y
then have *hypreal-of-real* $y + (v - u) <$ *hypreal-of-real* x
by (*simp add*: *Infinitesimal-add-hypreal-of-real-less Infinitesimal-diff* $u\ v$)
then show *False*

by (*simp add*: *add-diff-eq add-le-imp-le-diff le leD*)
qed

lemma *hypreal-of-real-le-add-Infininitesimal-cancel2*:
u ∈ *Infinitesimal* ⟹ *v* ∈ *Infinitesimal* ⟹
hypreal-of-real x + *u* ≤ *hypreal-of-real y* + *v* ⟹ *x* ≤ *y*
by (*blast intro*: *star-of-le* [*THEN iffD1*] *intro*!: *hypreal-of-real-le-add-Infininitesimal-cancel*)

lemma *hypreal-of-real-less-Infinitesimal-le-zero*:
hypreal-of-real x < *e* ⟹ *e* ∈ *Infinitesimal* ⟹ *hypreal-of-real x* ≤ *0*
by (*metis Infinitesimal-interval eq-iff le-less-linear star-of-Infinitesimal-iff-0 star-of-eq-0*)

lemma *Infinitesimal-add-not-zero*: *h* ∈ *Infinitesimal* ⟹ *x* ≠ *0* ⟹ *star-of x* + *h* ≠ *0*
by (*metis Infinitesimal-add-approx-self star-of-approx-zero-iff*)

lemma *monad-hrabs-less*: *y* ∈ *monad x* ⟹ *0* < *hypreal-of-real e* ⟹ |*y* − *x*| < *hypreal-of-real e*
by (*simp add*: *Infinitesimal-approx-minus approx-sym less-Infinitesimal-less mem-monad-approx*)

lemma *mem-monad-SReal-HFinite*: *x* ∈ *monad* (*hypreal-of-real a*) ⟹ *x* ∈ *HFinite*
using *HFinite-star-of approx-HFinite mem-monad-approx* **by** *blast*

6.7 Theorems about Standard Part

lemma *st-approx-self*: *x* ∈ *HFinite* ⟹ *st x* ≈ *x*
by (*metis* (*no-types*, *lifting*) *approx-refl approx-trans3 someI-ex st-def st-part-Ex st-part-Ex1*)

lemma *st-SReal*: *x* ∈ *HFinite* ⟹ *st x* ∈ ℝ
by (*metis* (*mono-tags*, *lifting*) *approx-sym someI-ex st-def st-part-Ex*)

lemma *st-HFinite*: *x* ∈ *HFinite* ⟹ *st x* ∈ *HFinite*
by (*erule st-SReal* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])

lemma *st-unique*: *r* ∈ ℝ ⟹ *r* ≈ *x* ⟹ *st x* = *r*
by (*meson SReal-subset-HFinite approx-HFinite approx-unique-real st-SReal st-approx-self subsetD*)

lemma *st-SReal-eq*: *x* ∈ ℝ ⟹ *st x* = *x*
by (*metis approx-refl st-unique*)

lemma *st-hypreal-of-real* [*simp*]: *st* (*hypreal-of-real x*) = *hypreal-of-real x*
by (*rule SReal-hypreal-of-real* [*THEN st-SReal-eq*])

lemma *st-eq-approx*: *x* ∈ *HFinite* ⟹ *y* ∈ *HFinite* ⟹ *st x* = *st y* ⟹ *x* ≈ *y*
by (*auto dest*!: *st-approx-self elim*!: *approx-trans3*)

lemma *approx-st-eq*:
assumes *x*: $x \in$ *HFinite* **and** *y*: $y \in$ *HFinite* **and** *xy*: $x \approx y$
shows *st* $x =$ *st* y
proof −
have *st* $x \approx x$ *st* $y \approx y$ *st* $x \in \mathbb{R}$ *st* $y \in \mathbb{R}$
by (*simp-all add*: *st-approx-self st-SReal x y*)
with *xy* **show** *?thesis*
by (*fast elim*: *approx-trans approx-trans2 SReal-approx-iff* [*THEN iffD1*])
qed

lemma *st-eq-approx-iff*: $x \in$ *HFinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow x \approx y \longleftrightarrow$ *st* $x =$ *st* y
by (*blast intro*: *approx-st-eq st-eq-approx*)

lemma *st-Infinitesimal-add-SReal*: $x \in \mathbb{R} \Longrightarrow e \in$ *Infinitesimal* $\Longrightarrow$ *st* $(x + e) = x$
by (*simp add*: *Infinitesimal-add-approx-self st-unique*)

lemma *st-Infinitesimal-add-SReal2*: $x \in \mathbb{R} \Longrightarrow e \in$ *Infinitesimal* $\Longrightarrow$ *st* $(e + x) = x$
by (*metis add.commute st-Infinitesimal-add-SReal*)

lemma *HFinite-st-Infinitesimal-add*: $x \in$ *HFinite* $\Longrightarrow \exists\, e \in$ *Infinitesimal*. $x =$ *st*$(x) + e$
by (*blast dest!*: *st-approx-self* [*THEN approx-sym*] *bex-Infinitesimal-iff2* [*THEN iffD2*])

lemma *st-add*: $x \in$ *HFinite* $\Longrightarrow y \in$ *HFinite* $\Longrightarrow$ *st* $(x + y) =$ *st* $x +$ *st* y
by (*simp add*: *st-unique st-SReal st-approx-self approx-add*)

lemma *st-numeral* [*simp*]: *st* (*numeral w*) = *numeral w*
by (*rule Reals-numeral* [*THEN st-SReal-eq*])

lemma *st-neg-numeral* [*simp*]: *st* (− *numeral w*) = − *numeral w*
using *st-unique* **by** *auto*

lemma *st-0* [*simp*]: *st* $0 = 0$
by (*simp add*: *st-SReal-eq*)

lemma *st-1* [*simp*]: *st* $1 = 1$
by (*simp add*: *st-SReal-eq*)

lemma *st-neg-1* [*simp*]: *st* $(-\ 1) = -\ 1$
by (*simp add*: *st-SReal-eq*)

lemma *st-minus*: $x \in$ *HFinite* $\Longrightarrow$ *st* $(-\ x) = -$ *st* x
by (*simp add*: *st-unique st-SReal st-approx-self approx-minus*)

lemma *st-diff*: $[\![x \in$ *HFinite*; $y \in$ *HFinite*$]\!] \Longrightarrow$ *st* $(x - y) =$ *st* $x -$ *st* y
by (*simp add*: *st-unique st-SReal st-approx-self approx-diff*)

lemma *st-mult*: $[\![x \in \mathit{HFinite};\ y \in \mathit{HFinite}]\!] \Longrightarrow st\ (x * y) = st\ x * st\ y$
by (*simp add*: *st-unique st-SReal st-approx-self approx-mult-HFinite*)

lemma *st-Infinitesimal*: $x \in \mathit{Infinitesimal} \Longrightarrow st\ x = 0$
by (*simp add*: *st-unique mem-infmal-iff*)

lemma *st-not-Infinitesimal*: $st(x) \neq 0 \Longrightarrow x \notin \mathit{Infinitesimal}$
by (*fast intro*: *st-Infinitesimal*)

lemma *st-inverse*: $x \in \mathit{HFinite} \Longrightarrow st\ x \neq 0 \Longrightarrow st\ (\mathit{inverse}\ x) = \mathit{inverse}\ (st\ x)$
by (*simp add*: *approx-inverse st-SReal st-approx-self st-not-Infinitesimal st-unique*)

lemma *st-divide* [*simp*]: $x \in \mathit{HFinite} \Longrightarrow y \in \mathit{HFinite} \Longrightarrow st\ y \neq 0 \Longrightarrow st\ (x\ /\ y) = st\ x\ /\ st\ y$
by (*simp add*: *divide-inverse st-mult st-not-Infinitesimal HFinite-inverse st-inverse*)

lemma *st-idempotent* [*simp*]: $x \in \mathit{HFinite} \Longrightarrow st\ (st\ x) = st\ x$
by (*blast intro*: *st-HFinite st-approx-self approx-st-eq*)

lemma *Infinitesimal-add-st-less*:
$x \in \mathit{HFinite} \Longrightarrow y \in \mathit{HFinite} \Longrightarrow u \in \mathit{Infinitesimal} \Longrightarrow st\ x < st\ y \Longrightarrow st\ x + u < st\ y$
by (*metis Infinitesimal-add-hypreal-of-real-less SReal-iff st-SReal star-of-less*)

lemma *Infinitesimal-add-st-le-cancel*:
$x \in \mathit{HFinite} \Longrightarrow y \in \mathit{HFinite} \Longrightarrow u \in \mathit{Infinitesimal} \Longrightarrow$
$st\ x \leq st\ y + u \Longrightarrow st\ x \leq st\ y$
by (*meson Infinitesimal-add-st-less leD le-less-linear*)

lemma *st-le*: $x \in \mathit{HFinite} \Longrightarrow y \in \mathit{HFinite} \Longrightarrow x \leq y \Longrightarrow st\ x \leq st\ y$
by (*metis approx-le-bound approx-sym linear st-SReal st-approx-self st-part-Ex1*)

lemma *st-zero-le*: $0 \leq x \Longrightarrow x \in \mathit{HFinite} \Longrightarrow 0 \leq st\ x$
by (*metis HFinite-0 st-0 st-le*)

lemma *st-zero-ge*: $x \leq 0 \Longrightarrow x \in \mathit{HFinite} \Longrightarrow st\ x \leq 0$
by (*metis HFinite-0 st-0 st-le*)

lemma *st-hrabs*: $x \in \mathit{HFinite} \Longrightarrow |st\ x| = st\ |x|$
by (*simp add*: *order-class.order.antisym st-zero-ge linorder-not-le st-zero-le abs-if st-minus linorder-not-less*)

6.8 Alternative Definitions using Free Ultrafilter

6.8.1 *HFinite*

lemma *HFinite-FreeUltrafilterNat*:
assumes $\mathit{star\text{-}n}\ X \in \mathit{HFinite}$
shows $\exists\, u.\ \mathit{eventually}\ (\lambda n.\ \mathit{norm}\ (X\ n) < u)\ \mathcal{U}$

proof –
  obtain *r* **where** *hnorm* (*star-n X*) < *hypreal-of-real r*
    using *HFiniteD SReal-iff assms* **by** *fastforce*
  then have $\forall_F$ *n in* $\mathcal{U}$. *norm* (*X n*) < *r*
    by (*simp add*: *hnorm-def star-n-less star-of-def starfun-star-n*)
  then show *?thesis* **..**
qed

lemma *FreeUltrafilterNat-HFinite*:
  assumes *eventually* (λ*n*. *norm* (*X n*) < *u*) $\mathcal{U}$
  shows *star-n X* ∈ *HFinite*
proof –
  have *hnorm* (*star-n X*) < *hypreal-of-real u*
    by (*simp add*: *assms hnorm-def star-n-less star-of-def starfun-star-n*)
  then show *?thesis*
    by (*meson HInfiniteD SReal-hypreal-of-real less-asym not-HFinite-HInfinite*)
qed

lemma *HFinite-FreeUltrafilterNat-iff*:
  star-n X ∈ *HFinite* ⟷ (∃ *u*. *eventually* (λ*n*. *norm* (*X n*) < *u*) $\mathcal{U}$)
  using *FreeUltrafilterNat-HFinite HFinite-FreeUltrafilterNat* **by** *blast*

6.8.2 *HInfinite*

Exclude this type of sets from free ultrafilter for Infinite numbers!

lemma *FreeUltrafilterNat-const-Finite*:
  eventually (λ*n*. *norm* (*X n*) = *u*) $\mathcal{U}$ ⟹ *star-n X* ∈ *HFinite*
  by (*simp add*: *FreeUltrafilterNat-HFinite* [**where** *u* = *u*+*1*] *eventually-mono*)

lemma *HInfinite-FreeUltrafilterNat*:
  assumes *star-n X* ∈ *HInfinite* **shows** $\forall_F$ *n in* $\mathcal{U}$. *u* < *norm* (*X n*)
proof –
have ¬ ($\forall_F$ *n in* $\mathcal{U}$. *norm* (*X n*) < *u* + *1*)
  using *FreeUltrafilterNat-HFinite HFinite-HInfinite-iff assms* **by** *auto*
  then show *?thesis*
    by (*auto simp flip*: *FreeUltrafilterNat.eventually-not-iff elim*: *eventually-mono*)
qed

lemma *FreeUltrafilterNat-HInfinite*:
  assumes ⋀*u*. *eventually* (λ*n*. *u* < *norm* (*X n*)) $\mathcal{U}$
  shows *star-n X* ∈ *HInfinite*
proof –
  { **fix** *u*
    assume $\forall_F$*n in* $\mathcal{U}$. *norm* (*X n*) < *u* $\forall_F$*n in* $\mathcal{U}$. *u* < *norm* (*X n*)
    then have $\forall_F$ *x in* $\mathcal{U}$. *False*
      by *eventually-elim auto*
    then have *False*
      by (*simp add*: *eventually-False FreeUltrafilterNat.proper*) }
  then show *?thesis*

using *HFinite-FreeUltrafilterNat HInfinite-HFinite-iff assms* **by** *blast*
qed

lemma *HInfinite-FreeUltrafilterNat-iff*:
star-n X ∈ *HInfinite* ⟷ (∀ *u. eventually* (λ*n. u* < *norm* (*X n*)) 𝒰)
using *HInfinite-FreeUltrafilterNat FreeUltrafilterNat-HInfinite* **by** *blast*

6.8.3 *Infinitesimal*

lemma *ball-SReal-eq*: (∀ *x*::*hypreal* ∈ *Reals. P x*) ⟷ (∀ *x*::*real. P* (*star-of x*))
by (*auto simp*: *SReal-def*)

lemma *Infinitesimal-FreeUltrafilterNat-iff*:
(*star-n X* ∈ *Infinitesimal*) = (∀ *u*>*0*. *eventually* (λ*n. norm* (*X n*) < *u*) 𝒰) (**is** *?lhs* = *?rhs*)
proof –
have *?lhs* ⟷ (∀ *r*>*0*. *hnorm* (*star-n X*) < *hypreal-of-real r*)
by (*simp add*: *Infinitesimal-def ball-SReal-eq*)
also have ... ⟷ *?rhs*
by (*simp add*: *hnorm-def starfun-star-n star-of-def star-less-def starP2-star-n*)
finally show *?thesis* **.**
qed

Infinitesimals as smaller than $1/n$ for all n::*nat* (> 0).

lemma *lemma-Infinitesimal*: (∀ *r*. *0* < *r* ⟶ *x* < *r*) ⟷ (∀ *n*. *x* < *inverse* (*real* (*Suc n*)))
by (*meson inverse-positive-iff-positive less-trans of-nat-0-less-iff reals-Archimedean zero-less-Suc*)

lemma *lemma-Infinitesimal2*:
(∀ *r* ∈ *Reals*. *0* < *r* ⟶ *x* < *r*) ⟷ (∀ *n*. *x* < *inverse*(*hypreal-of-nat* (*Suc n*))) (**is** - = *?rhs*)
proof (*intro iffI strip*)
assume *R*: *?rhs*
fix *r*::*hypreal*
assume *r* ∈ ℝ *0* < *r*
then obtain *n y* **where** *inverse* (*real* (*Suc n*)) < *y* **and** *r*: *r* = *hypreal-of-real y*
by (*metis SReal-iff reals-Archimedean star-of-0-less*)
then have *inverse* (*1* + *hypreal-of-nat n*) < *hypreal-of-real y*
by (*metis of-nat-Suc star-of-inverse star-of-less star-of-nat-def*)
then show *x* < *r*
by (*metis R r le-less-trans less-imp-le of-nat-Suc*)
qed (*meson Reals-inverse Reals-of-nat of-nat-0-less-iff positive-imp-inverse-positive zero-less-Suc*)

lemma *Infinitesimal-hypreal-of-nat-iff*:
Infinitesimal = {*x*. ∀ *n*. *hnorm x* < *inverse* (*hypreal-of-nat* (*Suc n*))}

using *Infinitesimal-def lemma-Infinitesimal2* **by** *auto*

6.9 Proof that ω is an infinite number

It will follow that ε is an infinitesimal number.

lemma *Suc-Un-eq*: $\{n.\ n < Suc\ m\} = \{n.\ n < m\}\ Un\ \{n.\ n = m\}$
by (*auto simp add*: *less-Suc-eq*)

Prove that any segment is finite and hence cannot belong to $\mathcal{U}$.

lemma *finite-real-of-nat-segment*: *finite* $\{n{::}nat.\ real\ n < real\ (m{::}nat)\}$
by *auto*

lemma *finite-real-of-nat-less-real*: *finite* $\{n{::}nat.\ real\ n < u\}$
proof −
obtain m **where** $u < real\ m$
using *reals-Archimedean2* **by** *blast*
then have $\{n.\ real\ n < u\} \subseteq \{..<m\}$
by *force*
then show *?thesis*
using *finite-nat-iff-bounded* **by** *force*
qed

lemma *finite-real-of-nat-le-real*: *finite* $\{n{::}nat.\ real\ n \leq u\}$
by (*metis infinite-nat-iff-unbounded leD le-nat-floor mem-Collect-eq*)

lemma *finite-rabs-real-of-nat-le-real*: *finite* $\{n{::}nat.\ |real\ n| \leq u\}$
by (*simp add*: *finite-real-of-nat-le-real*)

lemma *rabs-real-of-nat-le-real-FreeUltrafilterNat*:
$\neg$ *eventually* $(\lambda n.\ |real\ n| \leq u)\ \mathcal{U}$
by (*blast intro*!: *FreeUltrafilterNat.finite finite-rabs-real-of-nat-le-real*)

lemma *FreeUltrafilterNat-nat-gt-real*: *eventually* $(\lambda n.\ u < real\ n)\ \mathcal{U}$
proof −
have $\{n{::}nat.\ \neg\ u < real\ n\} = \{n.\ real\ n \leq u\}$
by *auto*
then show *?thesis*
by (*auto simp add*: *FreeUltrafilterNat.finite′ finite-real-of-nat-le-real*)
qed

The complement of $\{n.\ |real\ n| \leq u\} = \{n.\ u < |real\ n|\}$ is in $\mathcal{U}$ by property of (free) ultrafilters.

ω is a member of *HInfinite*.

theorem *HInfinite-omega* [*simp*]: $\omega \in HInfinite$
proof −
have $\forall_F\ n\ in\ \mathcal{U}.\ u < norm\ (1 + real\ n)$ **for** u
using *FreeUltrafilterNat-nat-gt-real* [*of* $u-1$] *eventually-mono* **by** *fastforce*

then show *?thesis*
by (*simp add*: *omega-def FreeUltrafilterNat-HInfinite*)
qed

Epsilon is a member of Infinitesimal.

lemma *Infinitesimal-epsilon* [*simp*]: $\varepsilon \in$ *Infinitesimal*
by (*auto intro*!: *HInfinite-inverse-Infinitesimal HInfinite-omega*
simp add: *epsilon-inverse-omega*)

lemma *HFinite-epsilon* [*simp*]: $\varepsilon \in$ *HFinite*
by (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*])

lemma *epsilon-approx-zero* [*simp*]: $\varepsilon \approx 0$
by (*simp add*: *mem-infmal-iff* [*symmetric*])

Needed for proof that we define a hyperreal [<$X(n)$] $\approx$ *hypreal-of-real a* given that $\forall\, n.\ |X\ n - a| < 1/n$. Used in proof of *NSLIM* $\Rightarrow$ *LIM*.

lemma *real-of-nat-less-inverse-iff*: $0 < u \Longrightarrow u <$ *inverse* (*real*(*Suc n*)) $\longleftrightarrow$ *real*(*Suc n*) $<$ *inverse u*
using *less-imp-inverse-less* **by** *force*

lemma *finite-inverse-real-of-posnat-gt-real*: $0 < u \Longrightarrow$ *finite* {*n*. $u <$ *inverse* (*real* (*Suc n*))}
proof (*simp only*: *real-of-nat-less-inverse-iff*)
have {*n*. $1 +$ *real n* $<$ *inverse u*} = {*n*. *real n* $<$ *inverse u* $- 1$}
by *fastforce*
then show *finite* {*n*. *real* (*Suc n*) $<$ *inverse u*}
using *finite-real-of-nat-less-real* [*of inverse u* $- 1$]
by *auto*
qed

lemma *finite-inverse-real-of-posnat-ge-real*:
assumes $0 < u$
shows *finite* {*n*. $u \le$ *inverse* (*real* (*Suc n*))}
proof $-$
have $\forall$ *na*. $u \le$ *inverse* ($1 +$ *real na*) $\longrightarrow$ *na* $\le$ *ceiling* (*inverse u*)
by (*smt* (*verit*, *best*) *assms ceiling-less-cancel ceiling-of-nat inverse-inverse-eq inverse-le-iff-le*)
then show *?thesis*
apply (*auto simp add*: *finite-nat-set-iff-bounded-le*)
by (*meson assms inverse-positive-iff-positive le-nat-iff less-imp-le zero-less-ceiling*)
qed

lemma *inverse-real-of-posnat-ge-real-FreeUltrafilterNat*:
$0 < u \Longrightarrow \neg$ *eventually* (λn. $u \le$ *inverse*(*real*(*Suc n*))) $\mathcal{U}$
by (*blast intro*!: *FreeUltrafilterNat.finite finite-inverse-real-of-posnat-ge-real*)

lemma *FreeUltrafilterNat-inverse-real-of-posnat*:
$0 < u \Longrightarrow$ *eventually* (λn. *inverse*(*real*(*Suc n*)) $< u$) $\mathcal{U}$

by (*drule inverse-real-of-posnat-ge-real-FreeUltrafilterNat*)
(*simp add*: *FreeUltrafilterNat.eventually-not-iff not-le*[*symmetric*])

Example of an hypersequence (i.e. an extended standard sequence) whose term with an hypernatural suffix is an infinitesimal i.e. the whn'nth term of the hypersequence is a member of Infinitesimal

lemma *SEQ-Infinitesimal*: ($*f*$ (λn::*nat*. *inverse*(*real*(*Suc n*)))) *whn* $\in$ *Infinitesimal*
by (*simp add*: *hypnat-omega-def starfun-star-n star-n-inverse Infinitesimal-FreeUltrafilterNat-iff FreeUltrafilterNat-inverse-real-of-posnat del*: *of-nat-Suc*)

Example where we get a hyperreal from a real sequence for which a particular property holds. The theorem is used in proofs about equivalence of nonstandard and standard neighbourhoods. Also used for equivalence of nonstandard ans standard definitions of pointwise limit.

$|X(n) - x| < 1/n \Longrightarrow [<X\ n>] - hypreal\text{-}of\text{-}real\ x| \in Infinitesimal$

lemma *real-seq-to-hypreal-Infinitesimal*:
$\forall n.\ norm\ (X\ n - x) < inverse\ (real\ (Suc\ n)) \Longrightarrow star\text{-}n\ X - star\text{-}of\ x \in Infinitesimal$
unfolding *star-n-diff star-of-def Infinitesimal-FreeUltrafilterNat-iff star-n-inverse*
by (*auto dest*!: *FreeUltrafilterNat-inverse-real-of-posnat*
intro: *order-less-trans elim*!: *eventually-mono*)

lemma *real-seq-to-hypreal-approx*:
$\forall n.\ norm\ (X\ n - x) < inverse\ (real\ (Suc\ n)) \Longrightarrow star\text{-}n\ X \approx star\text{-}of\ x$
by (*metis bex-Infinitesimal-iff real-seq-to-hypreal-Infinitesimal*)

lemma *real-seq-to-hypreal-approx2*:
$\forall n.\ norm\ (x - X\ n) < inverse(real(Suc\ n)) \Longrightarrow star\text{-}n\ X \approx star\text{-}of\ x$
by (*metis norm-minus-commute real-seq-to-hypreal-approx*)

lemma *real-seq-to-hypreal-Infinitesimal2*:
$\forall n.\ norm(X\ n - Y\ n) < inverse(real(Suc\ n)) \Longrightarrow star\text{-}n\ X - star\text{-}n\ Y \in Infinitesimal$
unfolding *Infinitesimal-FreeUltrafilterNat-iff star-n-diff*
by (*auto dest*!: *FreeUltrafilterNat-inverse-real-of-posnat*
intro: *order-less-trans elim*!: *eventually-mono*)

end

7 Nonstandard Complex Numbers

theory *NSComplex*
imports *NSA*
begin

type-synonym *hcomplex* = *complex star*

abbreviation *hcomplex-of-complex* :: *complex* ⇒ *complex star*
where *hcomplex-of-complex* ≡ *star-of*

abbreviation *hcmod* :: *complex star* ⇒ *real star*
where *hcmod* ≡ *hnorm*

7.0.1 Real and Imaginary parts

definition *hRe* :: *hcomplex* ⇒ *hypreal*
where *hRe* = ∗*f*∗ *Re*

definition *hIm* :: *hcomplex* ⇒ *hypreal*
where *hIm* = ∗*f*∗ *Im*

7.0.2 Imaginary unit

definition *iii* :: *hcomplex*
where *iii* = *star-of* i

7.0.3 Complex conjugate

definition *hcnj* :: *hcomplex* ⇒ *hcomplex*
where *hcnj* = ∗*f*∗ *cnj*

7.0.4 Argand

definition *hsgn* :: *hcomplex* ⇒ *hcomplex*
where *hsgn* = ∗*f*∗ *sgn*

definition *harg* :: *hcomplex* ⇒ *hypreal*
where *harg* = ∗*f*∗ *Arg*

definition — abbreviation for *cos a* + *i sin a*
hcis :: *hypreal* ⇒ *hcomplex*
where *hcis* = ∗*f*∗ *cis*

7.0.5 Injection from hyperreals

abbreviation *hcomplex-of-hypreal* :: *hypreal* ⇒ *hcomplex*
where *hcomplex-of-hypreal* ≡ *of-hypreal*

definition — abbreviation for *r* ∗ (*cos a* + *i sin a*)
hrcis :: *hypreal* ⇒ *hypreal* ⇒ *hcomplex*
where *hrcis* = ∗*f2*∗ *rcis*

7.0.6 e ^ $(x + iy)$

definition *hExp* :: *hcomplex* ⇒ *hcomplex*
where *hExp* = ∗*f*∗ *exp*

definition *HComplex* :: *hypreal* ⇒ *hypreal* ⇒ *hcomplex*
where *HComplex* = ∗*f2*∗ *Complex*

lemmas *hcomplex-defs* [*transfer-unfold*] =
hRe-def hIm-def iii-def hcnj-def hsgn-def harg-def hcis-def hrcis-def hExp-def HComplex-def

lemma *Standard-hRe* [*simp*]: *x* ∈ *Standard* ⟹ *hRe x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hIm* [*simp*]: *x* ∈ *Standard* ⟹ *hIm x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-iii* [*simp*]: *iii* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hcnj* [*simp*]: *x* ∈ *Standard* ⟹ *hcnj x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hsgn* [*simp*]: *x* ∈ *Standard* ⟹ *hsgn x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-harg* [*simp*]: *x* ∈ *Standard* ⟹ *harg x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hcis* [*simp*]: *r* ∈ *Standard* ⟹ *hcis r* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hExp* [*simp*]: *x* ∈ *Standard* ⟹ *hExp x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hrcis* [*simp*]: *r* ∈ *Standard* ⟹ *s* ∈ *Standard* ⟹ *hrcis r s* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-HComplex* [*simp*]: *r* ∈ *Standard* ⟹ *s* ∈ *Standard* ⟹ *HComplex r s* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *hcmod-def*: *hcmod* = ∗*f*∗ *cmod*
by (*rule hnorm-def*)

7.1 Properties of Nonstandard Real and Imaginary Parts

lemma *hcomplex-hRe-hIm-cancel-iff*: ⋀*w z*. *w* = *z* ⟷ *hRe w* = *hRe z* ∧ *hIm w* = *hIm z*
by *transfer* (*rule complex-eq-iff*)

lemma *hcomplex-equality* [*intro?*]: $\bigwedge z\ w.\ hRe\ z = hRe\ w \Longrightarrow hIm\ z = hIm\ w \Longrightarrow z = w$
by *transfer* (*rule complex-eqI*)

lemma *hcomplex-hRe-zero* [*simp*]: $hRe\ 0 = 0$
by *transfer simp*

lemma *hcomplex-hIm-zero* [*simp*]: $hIm\ 0 = 0$
by *transfer simp*

lemma *hcomplex-hRe-one* [*simp*]: $hRe\ 1 = 1$
by *transfer simp*

lemma *hcomplex-hIm-one* [*simp*]: $hIm\ 1 = 0$
by *transfer simp*

7.2 Addition for Nonstandard Complex Numbers

lemma *hRe-add*: $\bigwedge x\ y.\ hRe\ (x + y) = hRe\ x + hRe\ y$
by *transfer simp*

lemma *hIm-add*: $\bigwedge x\ y.\ hIm\ (x + y) = hIm\ x + hIm\ y$
by *transfer simp*

7.3 More Minus Laws

lemma *hRe-minus*: $\bigwedge z.\ hRe\ (-\ z) = -\ hRe\ z$
by *transfer* (*rule uminus-complex.sel*)

lemma *hIm-minus*: $\bigwedge z.\ hIm\ (-\ z) = -\ hIm\ z$
by *transfer* (*rule uminus-complex.sel*)

lemma *hcomplex-add-minus-eq-minus*: $x + y = 0 \Longrightarrow x = -\ y$
for *x y* :: *hcomplex*
apply (*drule minus-unique*)
apply (*simp add*: *minus-equation-iff* [*of x y*])
done

lemma *hcomplex-i-mult-eq* [*simp*]: $iii * iii = -\ 1$
by *transfer* (*rule i-squared*)

lemma *hcomplex-i-mult-left* [*simp*]: $\bigwedge z.\ iii * (iii * z) = -\ z$
by *transfer* (*rule complex-i-mult-minus*)

lemma *hcomplex-i-not-zero* [*simp*]: $iii \neq 0$
by *transfer* (*rule complex-i-not-zero*)

7.4 More Multiplication Laws

lemma *hcomplex-mult-minus-one*: $-\ 1 * z = -\ z$

for z :: *hcomplex*
by *simp*

lemma *hcomplex-mult-minus-one-right*: $z * - 1 = - z$
for z :: *hcomplex*
by *simp*

lemma *hcomplex-mult-left-cancel*: $c \neq 0 \Longrightarrow c * a = c * b \longleftrightarrow a = b$
for $a\ b\ c$ :: *hcomplex*
by *simp*

lemma *hcomplex-mult-right-cancel*: $c \neq 0 \Longrightarrow a * c = b * c \longleftrightarrow a = b$
for $a\ b\ c$ :: *hcomplex*
by *simp*

7.5 Subtraction and Division

lemma *hcomplex-diff-eq-eq* [*simp*]: $x - y = z \longleftrightarrow x = z + y$
for $x\ y\ z$ :: *hcomplex*
by (*rule diff-eq-eq*)

7.6 Embedding Properties for *hcomplex-of-hypreal* Map

lemma *hRe-hcomplex-of-hypreal* [*simp*]: $\bigwedge z.$ *hRe* (*hcomplex-of-hypreal* z) = z
by *transfer* (*rule Re-complex-of-real*)

lemma *hIm-hcomplex-of-hypreal* [*simp*]: $\bigwedge z.$ *hIm* (*hcomplex-of-hypreal* z) = *0*
by *transfer* (*rule Im-complex-of-real*)

lemma *hcomplex-of-epsilon-not-zero* [*simp*]: *hcomplex-of-hypreal* $\varepsilon \neq 0$
by (*simp add*: *epsilon-not-zero*)

7.7 *HComplex* theorems

lemma *hRe-HComplex* [*simp*]: $\bigwedge x\ y.$ *hRe* (*HComplex* $x\ y$) = x
by *transfer simp*

lemma *hIm-HComplex* [*simp*]: $\bigwedge x\ y.$ *hIm* (*HComplex* $x\ y$) = y
by *transfer simp*

lemma *hcomplex-surj* [*simp*]: $\bigwedge z.$ *HComplex* (*hRe* z) (*hIm* z) = z
by *transfer* (*rule complex-surj*)

lemma *hcomplex-induct* [*case-names rect*]:
($\bigwedge x\ y.$ P (*HComplex* $x\ y$)) $\Longrightarrow P\ z$
by (*rule hcomplex-surj* [*THEN subst*]) *blast*

7.8 Modulus (Absolute Value) of Nonstandard Complex Number

lemma *hcomplex-of-hypreal-abs*:
hcomplex-of-hypreal $|x|$ = *hcomplex-of-hypreal* (*hcmod* (*hcomplex-of-hypreal* *x*))
by *simp*

lemma *HComplex-inject* [*simp*]: $\bigwedge x\ y\ x'\ y'$. *HComplex x y* = *HComplex x′ y′* $\longleftrightarrow$ $x = x' \wedge y = y'$
by *transfer* (*rule complex.inject*)

lemma *HComplex-add* [*simp*]:
$\bigwedge$*x1 y1 x2 y2*. *HComplex x1 y1* + *HComplex x2 y2* = *HComplex* (*x1* + *x2*) (*y1* + *y2*)
by *transfer* (*rule complex-add*)

lemma *HComplex-minus* [*simp*]: $\bigwedge x\ y$. − *HComplex x y* = *HComplex* (− *x*) (− *y*)
by *transfer* (*rule complex-minus*)

lemma *HComplex-diff* [*simp*]:
$\bigwedge$*x1 y1 x2 y2*. *HComplex x1 y1* − *HComplex x2 y2* = *HComplex* (*x1* − *x2*) (*y1* − *y2*)
by *transfer* (*rule complex-diff*)

lemma *HComplex-mult* [*simp*]:
$\bigwedge$*x1 y1 x2 y2*. *HComplex x1 y1* ∗ *HComplex x2 y2* = *HComplex* (*x1*∗*x2* − *y1*∗*y2*) (*x1*∗*y2* + *y1*∗*x2*)
by *transfer* (*rule complex-mult*)

HComplex-inverse is proved below.

lemma *hcomplex-of-hypreal-eq*: $\bigwedge r$. *hcomplex-of-hypreal r* = *HComplex r 0*
by *transfer* (*rule complex-of-real-def*)

lemma *HComplex-add-hcomplex-of-hypreal* [*simp*]:
$\bigwedge x\ y\ r$. *HComplex x y* + *hcomplex-of-hypreal r* = *HComplex* (*x* + *r*) *y*
by *transfer* (*rule Complex-add-complex-of-real*)

lemma *hcomplex-of-hypreal-add-HComplex* [*simp*]:
$\bigwedge r\ x\ y$. *hcomplex-of-hypreal r* + *HComplex x y* = *HComplex* (*r* + *x*) *y*
by *transfer* (*rule complex-of-real-add-Complex*)

lemma *HComplex-mult-hcomplex-of-hypreal*:
$\bigwedge x\ y\ r$. *HComplex x y* ∗ *hcomplex-of-hypreal r* = *HComplex* (*x* ∗ *r*) (*y* ∗ *r*)
by *transfer* (*rule Complex-mult-complex-of-real*)

lemma *hcomplex-of-hypreal-mult-HComplex*:
$\bigwedge r\ x\ y$. *hcomplex-of-hypreal r* ∗ *HComplex x y* = *HComplex* (*r* ∗ *x*) (*r* ∗ *y*)
by *transfer* (*rule complex-of-real-mult-Complex*)

lemma *i-hcomplex-of-hypreal* [*simp*]: $\bigwedge r.\ iii * hcomplex\text{-}of\text{-}hypreal\ r = HComplex\ 0\ r$
by *transfer* (*rule i-complex-of-real*)

lemma *hcomplex-of-hypreal-i* [*simp*]: $\bigwedge r.\ hcomplex\text{-}of\text{-}hypreal\ r * iii = HComplex\ 0\ r$
by *transfer* (*rule complex-of-real-i*)

7.9 Conjugation

lemma *hcomplex-hcnj-cancel-iff* [*iff*]: $\bigwedge x\ y.\ hcnj\ x = hcnj\ y \longleftrightarrow x = y$
by *transfer* (*rule complex-cnj-cancel-iff*)

lemma *hcomplex-hcnj-hcnj* [*simp*]: $\bigwedge z.\ hcnj\ (hcnj\ z) = z$
by *transfer* (*rule complex-cnj-cnj*)

lemma *hcomplex-hcnj-hcomplex-of-hypreal* [*simp*]:
$\bigwedge x.\ hcnj\ (hcomplex\text{-}of\text{-}hypreal\ x) = hcomplex\text{-}of\text{-}hypreal\ x$
by *transfer* (*rule complex-cnj-complex-of-real*)

lemma *hcomplex-hmod-hcnj* [*simp*]: $\bigwedge z.\ hcmod\ (hcnj\ z) = hcmod\ z$
by *transfer* (*rule complex-mod-cnj*)

lemma *hcomplex-hcnj-minus*: $\bigwedge z.\ hcnj\ (-\ z) = -\ hcnj\ z$
by *transfer* (*rule complex-cnj-minus*)

lemma *hcomplex-hcnj-inverse*: $\bigwedge z.\ hcnj\ (inverse\ z) = inverse\ (hcnj\ z)$
by *transfer* (*rule complex-cnj-inverse*)

lemma *hcomplex-hcnj-add*: $\bigwedge w\ z.\ hcnj\ (w + z) = hcnj\ w + hcnj\ z$
by *transfer* (*rule complex-cnj-add*)

lemma *hcomplex-hcnj-diff*: $\bigwedge w\ z.\ hcnj\ (w - z) = hcnj\ w - hcnj\ z$
by *transfer* (*rule complex-cnj-diff*)

lemma *hcomplex-hcnj-mult*: $\bigwedge w\ z.\ hcnj\ (w * z) = hcnj\ w * hcnj\ z$
by *transfer* (*rule complex-cnj-mult*)

lemma *hcomplex-hcnj-divide*: $\bigwedge w\ z.\ hcnj\ (w\ /\ z) = hcnj\ w\ /\ hcnj\ z$
by *transfer* (*rule complex-cnj-divide*)

lemma *hcnj-one* [*simp*]: $hcnj\ 1 = 1$
by *transfer* (*rule complex-cnj-one*)

lemma *hcomplex-hcnj-zero* [*simp*]: $hcnj\ 0 = 0$
by *transfer* (*rule complex-cnj-zero*)

lemma *hcomplex-hcnj-zero-iff* [*iff*]: $\bigwedge z.\ hcnj\ z = 0 \longleftrightarrow z = 0$
by *transfer* (*rule complex-cnj-zero-iff*)

lemma *hcomplex-mult-hcnj*: $\bigwedge z.\ z * hcnj\ z = hcomplex\text{-}of\text{-}hypreal\ ((hRe\ z)^2 + (hIm\ z)^2)$
by *transfer* (*rule complex-mult-cnj*)

7.10 More Theorems about the Function *hcmod*

lemma *hcmod-hcomplex-of-hypreal-of-nat* [*simp*]:
$hcmod\ (hcomplex\text{-}of\text{-}hypreal\ (hypreal\text{-}of\text{-}nat\ n)) = hypreal\text{-}of\text{-}nat\ n$
by *simp*

lemma *hcmod-hcomplex-of-hypreal-of-hypnat* [*simp*]:
$hcmod\ (hcomplex\text{-}of\text{-}hypreal(hypreal\text{-}of\text{-}hypnat\ n)) = hypreal\text{-}of\text{-}hypnat\ n$
by *simp*

lemma *hcmod-mult-hcnj*: $\bigwedge z.\ hcmod\ (z * hcnj\ z) = (hcmod\ z)^2$
by *transfer* (*rule complex-mod-mult-cnj*)

lemma *hcmod-triangle-ineq2* [*simp*]: $\bigwedge a\ b.\ hcmod\ (b + a) - hcmod\ b \le hcmod\ a$
by *transfer* (*rule complex-mod-triangle-ineq2*)

lemma *hcmod-diff-ineq* [*simp*]: $\bigwedge a\ b.\ hcmod\ a - hcmod\ b \le hcmod\ (a + b)$
by *transfer* (*rule norm-diff-ineq*)

7.11 Exponentiation

lemma *hcomplexpow-0* [*simp*]: $z\ \widehat{}\ 0 = 1$
for z :: *hcomplex*
by (*rule power-0*)

lemma *hcomplexpow-Suc* [*simp*]: $z\ \widehat{}\ (Suc\ n) = z * (z\ \widehat{}\ n)$
for z :: *hcomplex*
by (*rule power-Suc*)

lemma *hcomplexpow-i-squared* [*simp*]: $iii^2 = -1$
by *transfer* (*rule power2-i*)

lemma *hcomplex-of-hypreal-pow*: $\bigwedge x.\ hcomplex\text{-}of\text{-}hypreal\ (x\ \widehat{}\ n) = hcomplex\text{-}of\text{-}hypreal\ x\ \widehat{}\ n$
by *transfer* (*rule of-real-power*)

lemma *hcomplex-hcnj-pow*: $\bigwedge z.\ hcnj\ (z\ \widehat{}\ n) = hcnj\ z\ \widehat{}\ n$
by *transfer* (*rule complex-cnj-power*)

lemma *hcmod-hcomplexpow*: $\bigwedge x.\ hcmod\ (x\ \widehat{}\ n) = hcmod\ x\ \widehat{}\ n$
by *transfer* (*rule norm-power*)

lemma *hcpow-minus*:
$\bigwedge x\ n.\ (- x :: hcomplex)\ pow\ n = (if\ (\ *p*\ even)\ n\ then\ (x\ pow\ n)\ else - (x\ pow\ n))$

by *transfer simp*

lemma *hcpow-mult*: $(r * s)$ *pow* $n = (r$ *pow* $n) * (s$ *pow* $n)$
for *r s* :: *hcomplex*
by (*fact hyperpow-mult*)

lemma *hcpow-zero2* [*simp*]: $\bigwedge n.$ *0 pow* (*hSuc n*) = (*0*::'*a*::*semiring-1 star*)
by *transfer* (*rule power-0-Suc*)

lemma *hcpow-not-zero* [*simp,intro*]: $\bigwedge r\ n.\ r \neq 0 \Longrightarrow r$ *pow* $n \neq$ (*0*::*hcomplex*)
by (*fact hyperpow-not-zero*)

lemma *hcpow-zero-zero*: *r pow n* = *0* $\Longrightarrow$ *r* = *0*
for *r* :: *hcomplex*
by (*blast intro*: *ccontr dest*: *hcpow-not-zero*)

7.12 The Function *hsgn*

lemma *hsgn-zero* [*simp*]: *hsgn 0* = *0*
by *transfer* (*rule sgn-zero*)

lemma *hsgn-one* [*simp*]: *hsgn 1* = *1*
by *transfer* (*rule sgn-one*)

lemma *hsgn-minus*: $\bigwedge z.$ *hsgn* (− *z*) = − *hsgn z*
by *transfer* (*rule sgn-minus*)

lemma *hsgn-eq*: $\bigwedge z.$ *hsgn z* = *z* / *hcomplex-of-hypreal* (*hcmod z*)
by *transfer* (*rule sgn-eq*)

lemma *hcmod-i*: $\bigwedge x\ y.$ *hcmod* (*HComplex x y*) = (**f* sqrt*) $(x^2 + y^2)$
by *transfer* (*rule complex-norm*)

lemma *hcomplex-eq-cancel-iff1* [*simp*]:
hcomplex-of-hypreal xa = *HComplex x y* ⟷ *xa* = *x* ∧ *y* = *0*
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *hcomplex-eq-cancel-iff2* [*simp*]:
HComplex x y = *hcomplex-of-hypreal xa* ⟷ *x* = *xa* ∧ *y* = *0*
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *HComplex-eq-0* [*simp*]: $\bigwedge x\ y.$ *HComplex x y* = *0* ⟷ *x* = *0* ∧ *y* = *0*
by *transfer* (*rule Complex-eq-0*)

lemma *HComplex-eq-1* [*simp*]: $\bigwedge x\ y.$ *HComplex x y* = *1* ⟷ *x* = *1* ∧ *y* = *0*
by *transfer* (*rule Complex-eq-1*)

lemma *i-eq-HComplex-0-1*: *iii* = *HComplex 0 1*
by *transfer* (*simp add*: *complex-eq-iff*)

lemma *HComplex-eq-i* [*simp*]: $\bigwedge x\ y.\ HComplex\ x\ y = iii \longleftrightarrow x = 0 \wedge y = 1$
by *transfer* (*rule Complex-eq-i*)

lemma *hRe-hsgn* [*simp*]: $\bigwedge z.\ hRe\ (hsgn\ z) = hRe\ z\ /\ hcmod\ z$
by *transfer* (*rule Re-sgn*)

lemma *hIm-hsgn* [*simp*]: $\bigwedge z.\ hIm\ (hsgn\ z) = hIm\ z\ /\ hcmod\ z$
by *transfer* (*rule Im-sgn*)

lemma *HComplex-inverse*: $\bigwedge x\ y.\ inverse\ (HComplex\ x\ y) = HComplex\ (x\ /\ (x^2 + y^2))\ (-\ y\ /\ (x^2 + y^2))$
by *transfer* (*rule complex-inverse*)

lemma *hRe-mult-i-eq*[*simp*]: $\bigwedge y.\ hRe\ (iii * hcomplex\text{-}of\text{-}hypreal\ y) = 0$
by *transfer simp*

lemma *hIm-mult-i-eq* [*simp*]: $\bigwedge y.\ hIm\ (iii * hcomplex\text{-}of\text{-}hypreal\ y) = y$
by *transfer simp*

lemma *hcmod-mult-i* [*simp*]: $\bigwedge y.\ hcmod\ (iii * hcomplex\text{-}of\text{-}hypreal\ y) = |y|$
by *transfer* (*simp add*: *norm-complex-def*)

lemma *hcmod-mult-i2* [*simp*]: $\bigwedge y.\ hcmod\ (hcomplex\text{-}of\text{-}hypreal\ y * iii) = |y|$
by *transfer* (*simp add*: *norm-complex-def*)

7.12.1 *harg*

lemma *cos-harg-i-mult-zero* [*simp*]: $\bigwedge y.\ y \neq 0 \Longrightarrow (\ *f*\ cos)\ (harg\ (HComplex\ 0\ y)) = 0$
by *transfer* (*simp add*: *Complex-eq*)

7.13 Polar Form for Nonstandard Complex Numbers

lemma *complex-split-polar2*: $\forall n.\ \exists r\ a.\ (z\ n) = complex\text{-}of\text{-}real\ r * Complex\ (cos\ a)\ (sin\ a)$
unfolding *Complex-eq* **by** (*auto intro*: *complex-split-polar*)

lemma *hcomplex-split-polar*:
$\bigwedge z.\ \exists r\ a.\ z = hcomplex\text{-}of\text{-}hypreal\ r * (HComplex\ ((\ *f*\ cos)\ a)\ ((\ *f*\ sin)\ a))$
by *transfer* (*simp add*: *Complex-eq complex-split-polar*)

lemma *hcis-eq*:
$\bigwedge a.\ hcis\ a = hcomplex\text{-}of\text{-}hypreal\ ((\ *f*\ cos)\ a) + iii * hcomplex\text{-}of\text{-}hypreal\ ((\ *f*\ sin)\ a)$
by *transfer* (*simp add*: *complex-eq-iff*)

lemma *hrcis-Ex*: $\bigwedge z.\ \exists r\ a.\ z = hrcis\ r\ a$
by *transfer* (*rule rcis-Ex*)

lemma *hRe-hcomplex-polar* [*simp*]:
⋀*r a*. *hRe* (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) = *r* ∗ (∗*f*∗ *cos*) *a*
by *transfer simp*

lemma *hRe-hrcis* [*simp*]: ⋀*r a*. *hRe* (*hrcis r a*) = *r* ∗ (∗*f*∗ *cos*) *a*
by *transfer* (*rule Re-rcis*)

lemma *hIm-hcomplex-polar* [*simp*]:
⋀*r a*. *hIm* (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) = *r* ∗ (∗*f*∗ *sin*) *a*
by *transfer simp*

lemma *hIm-hrcis* [*simp*]: ⋀*r a*. *hIm* (*hrcis r a*) = *r* ∗ (∗*f*∗ *sin*) *a*
by *transfer* (*rule Im-rcis*)

lemma *hcmod-unit-one* [*simp*]: ⋀*a*. *hcmod* (*HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) = *1*
by *transfer* (*simp add*: *cmod-unit-one*)

lemma *hcmod-complex-polar* [*simp*]:
⋀*r a*. *hcmod* (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) = |*r*|
by *transfer* (*simp add*: *Complex-eq cmod-complex-polar*)

lemma *hcmod-hrcis* [*simp*]: ⋀*r a*. *hcmod*(*hrcis r a*) = |*r*|
by *transfer* (*rule complex-mod-rcis*)

(*r1* ∗ *hrcis a*) ∗ (*r2* ∗ *hrcis b*) = *r1* ∗ *r2* ∗ *hrcis* (*a* + *b*)

lemma *hcis-hrcis-eq*: ⋀*a*. *hcis a* = *hrcis 1 a*
by *transfer* (*rule cis-rcis-eq*)
declare *hcis-hrcis-eq* [*symmetric*, *simp*]

lemma *hrcis-mult*: ⋀*a b r1 r2*. *hrcis r1 a* ∗ *hrcis r2 b* = *hrcis* (*r1* ∗ *r2*) (*a* + *b*)
by *transfer* (*rule rcis-mult*)

lemma *hcis-mult*: ⋀*a b*. *hcis a* ∗ *hcis b* = *hcis* (*a* + *b*)
by *transfer* (*rule cis-mult*)

lemma *hcis-zero* [*simp*]: *hcis 0* = *1*
by *transfer* (*rule cis-zero*)

lemma *hrcis-zero-mod* [*simp*]: ⋀*a*. *hrcis 0 a* = *0*
by *transfer* (*rule rcis-zero-mod*)

lemma *hrcis-zero-arg* [*simp*]: ⋀*r*. *hrcis r 0* = *hcomplex-of-hypreal r*
by *transfer* (*rule rcis-zero-arg*)

lemma *hcomplex-i-mult-minus* [*simp*]: ⋀*x*. *iii* ∗ (*iii* ∗ *x*) = − *x*

by *transfer* (*rule complex-i-mult-minus*)

lemma *hcomplex-i-mult-minus2* [*simp*]: $iii * iii * x = - x$
by *simp*

lemma *hcis-hypreal-of-nat-Suc-mult*:
$\bigwedge a.$ *hcis* (*hypreal-of-nat* (*Suc n*) $*$ *a*) $=$ *hcis a* $*$ *hcis* (*hypreal-of-nat n* $*$ *a*)
by *transfer* (*simp add*: *distrib-right cis-mult*)

lemma *NSDeMoivre*: $\bigwedge a.$ (*hcis a*) ^ *n* $=$ *hcis* (*hypreal-of-nat n* $*$ *a*)
by *transfer* (*rule DeMoivre*)

lemma *hcis-hypreal-of-hypnat-Suc-mult*:
$\bigwedge a\ n.$ *hcis* (*hypreal-of-hypnat* ($n + 1$) $*$ *a*) $=$ *hcis a* $*$ *hcis* (*hypreal-of-hypnat n* $*$ *a*)
by *transfer* (*simp add*: *distrib-right cis-mult*)

lemma *NSDeMoivre-ext*: $\bigwedge a\ n.$ (*hcis a*) *pow n* $=$ *hcis* (*hypreal-of-hypnat n* $*$ *a*)
by *transfer* (*rule DeMoivre*)

lemma *NSDeMoivre2*: $\bigwedge a\ r.$ (*hrcis r a*) ^ *n* $=$ *hrcis* (*r* ^ *n*) (*hypreal-of-nat n* $*$ *a*)
by *transfer* (*rule DeMoivre2*)

lemma *DeMoivre2-ext*: $\bigwedge a\ r\ n.$ (*hrcis r a*) *pow n* $=$ *hrcis* (*r pow n*) (*hypreal-of-hypnat n* $*$ *a*)
by *transfer* (*rule DeMoivre2*)

lemma *hcis-inverse* [*simp*]: $\bigwedge a.$ *inverse* (*hcis a*) $=$ *hcis* ($-$ *a*)
by *transfer* (*rule cis-inverse*)

lemma *hrcis-inverse*: $\bigwedge a\ r.$ *inverse* (*hrcis r a*) $=$ *hrcis* (*inverse r*) ($-$ *a*)
by *transfer* (*simp add*: *rcis-inverse inverse-eq-divide* [*symmetric*])

lemma *hRe-hcis* [*simp*]: $\bigwedge a.$ *hRe* (*hcis a*) $=$ (**f* cos*) *a*
by *transfer simp*

lemma *hIm-hcis* [*simp*]: $\bigwedge a.$ *hIm* (*hcis a*) $=$ (**f* sin*) *a*
by *transfer simp*

lemma *cos-n-hRe-hcis-pow-n*: (**f* cos*) (*hypreal-of-nat n* $*$ *a*) $=$ *hRe* (*hcis a* ^ *n*)
by (*simp add*: *NSDeMoivre*)

lemma *sin-n-hIm-hcis-pow-n*: (**f* sin*) (*hypreal-of-nat n* $*$ *a*) $=$ *hIm* (*hcis a* ^ *n*)
by (*simp add*: *NSDeMoivre*)

lemma *cos-n-hRe-hcis-hcpow-n*: (**f* cos*) (*hypreal-of-hypnat n* $*$ *a*) $=$ *hRe* (*hcis a pow n*)
by (*simp add*: *NSDeMoivre-ext*)

lemma *sin-n-hIm-hcis-hcpow-n*: (*∗f∗ sin*) (*hypreal-of-hypnat n ∗ a*) = *hIm* (*hcis a pow n*)
by (*simp add*: *NSDeMoivre-ext*)

lemma *hExp-add*: ⋀*a b*. *hExp* (*a* + *b*) = *hExp a ∗ hExp b*
by *transfer* (*rule exp-add*)

7.14 *hcomplex-of-complex*: the Injection from type *complex* to to *hcomplex*

lemma *hcomplex-of-complex-i*: *iii* = *hcomplex-of-complex* i
by (*rule iii-def*)

lemma *hRe-hcomplex-of-complex*: *hRe* (*hcomplex-of-complex z*) = *hypreal-of-real* (*Re z*)
by *transfer* (*rule refl*)

lemma *hIm-hcomplex-of-complex*: *hIm* (*hcomplex-of-complex z*) = *hypreal-of-real* (*Im z*)
by *transfer* (*rule refl*)

lemma *hcmod-hcomplex-of-complex*: *hcmod* (*hcomplex-of-complex x*) = *hypreal-of-real* (*cmod x*)
by *transfer* (*rule refl*)

7.15 Numerals and Arithmetic

lemma *hcomplex-of-hypreal-eq-hcomplex-of-complex*:
hcomplex-of-hypreal (*hypreal-of-real x*) = *hcomplex-of-complex* (*complex-of-real x*)
by *transfer* (*rule refl*)

lemma *hcomplex-hypreal-numeral*:
hcomplex-of-complex (*numeral w*) = *hcomplex-of-hypreal*(*numeral w*)
by *transfer* (*rule of-real-numeral* [*symmetric*])

lemma *hcomplex-hypreal-neg-numeral*:
hcomplex-of-complex (− *numeral w*) = *hcomplex-of-hypreal*(− *numeral w*)
by *transfer* (*rule of-real-neg-numeral* [*symmetric*])

lemma *hcomplex-numeral-hcnj* [*simp*]: *hcnj* (*numeral v* :: *hcomplex*) = *numeral v*
by *transfer* (*rule complex-cnj-numeral*)

lemma *hcomplex-numeral-hcmod* [*simp*]: *hcmod* (*numeral v* :: *hcomplex*) = (*numeral v* :: *hypreal*)
by *transfer* (*rule norm-numeral*)

lemma *hcomplex-neg-numeral-hcmod* [*simp*]: *hcmod* (− *numeral v* :: *hcomplex*) = (*numeral v* :: *hypreal*)
by *transfer* (*rule norm-neg-numeral*)

lemma *hcomplex-numeral-hRe* [*simp*]: *hRe* (*numeral v* :: *hcomplex*) = *numeral v*
by *transfer* (*rule complex-Re-numeral*)

lemma *hcomplex-numeral-hIm* [*simp*]: *hIm* (*numeral v* :: *hcomplex*) = *0*
by *transfer* (*rule complex-Im-numeral*)

end

8 Star-Transforms in Non-Standard Analysis

theory *Star*
imports *NSA*
begin

definition — internal sets
starset-n :: (*nat* ⇒ ′*a set*) ⇒ ′*a star set*
(‹(‹*open-block notation*=‹*prefix starset-n*››∗*sn*∗ -)› [*80*] *80*)
where ∗*sn*∗ *As* = *Iset* (*star-n As*)

definition *InternalSets* :: ′*a star set set*
where *InternalSets* = {*X*. ∃ *As*. *X* = ∗*sn*∗ *As*}

definition — nonstandard extension of function
is-starext :: (′*a star* ⇒ ′*a star*) ⇒ (′*a* ⇒ ′*a*) ⇒ *bool*
where *is-starext F f* ⟷
(∀ *x y*. ∃ *X* ∈ *Rep-star x*. ∃ *Y* ∈ *Rep-star y*. *y* = *F x* ⟷ *eventually* (λ*n*. *Y n* = *f*(*X n*)) 𝒰)

definition — internal functions
starfun-n :: (*nat* ⇒ ′*a* ⇒ ′*b*) ⇒ ′*a star* ⇒ ′*b star*
(‹(‹*open-block notation*=‹*prefix starfun-n*››∗*fn*∗ -)› [*80*] *80*)
where ∗*fn*∗ *F* = *Ifun* (*star-n F*)

definition *InternalFuns* :: (′*a star* => ′*b star*) *set*
where *InternalFuns* = {*X*. ∃ *F*. *X* = ∗*fn*∗ *F*}

8.1 Preamble - Pulling ∃ over ∀

This proof does not need AC and was suggested by the referee for the JCM Paper: let *f x* be least *y* such that *Q x y*.

lemma *no-choice*: ∀ *x*. ∃ *y*. *Q x y* ⟹ ∃ *f* :: ′*a* ⇒ *nat*. ∀ *x*. *Q x* (*f x*)
by (*rule exI* [**where** *x* = λ*x*. *LEAST y*. *Q x y*]) (*blast intro*: *LeastI*)

8.2 Properties of the Star-transform Applied to Sets of Reals

lemma *STAR-star-of-image-subset*: *star-of* ‘ *A* ⊆ ∗*s*∗ *A*
by *auto*

lemma *STAR-hypreal-of-real-Int*: $*s*\ X \cap \mathbb{R} = \textit{hypreal-of-real}\ `\ X$
by (*auto simp add*: *SReal-def*)

lemma *STAR-star-of-Int*: $*s*\ X \cap \textit{Standard} = \textit{star-of}\ `\ X$
by (*auto simp add*: *Standard-def*)

lemma *lemma-not-hyprealA*: $x \notin \textit{hypreal-of-real}\ `\ A \Longrightarrow \forall\, y \in A.\ x \neq \textit{hypreal-of-real}\ y$
by *auto*

lemma *lemma-not-starA*: $x \notin \textit{star-of}\ `\ A \Longrightarrow \forall\, y \in A.\ x \neq \textit{star-of}\ y$
by *auto*

lemma *STAR-real-seq-to-hypreal*: $\forall\, n.\ (X\ n) \notin M \Longrightarrow \textit{star-n}\ X \notin *s*\ M$
by (*simp add*: *starset-def star-of-def Iset-star-n FreeUltrafilterNat.proper*)

lemma *STAR-singleton*: $*s*\ \{x\} = \{\textit{star-of}\ x\}$
by *simp*

lemma *STAR-not-mem*: $x \notin F \Longrightarrow \textit{star-of}\ x \notin *s*\ F$
by *transfer*

lemma *STAR-subset-closed*: $x \in *s*\ A \Longrightarrow A \subseteq B \Longrightarrow x \in *s*\ B$
by (*erule rev-subsetD*) *simp*

Nonstandard extension of a set (defined using a constant sequence) as a special case of an internal set.

lemma *starset-n-starset*: $\forall\, n.\ As\ n = A \Longrightarrow *sn*\ As = *s*\ A$
by (*drule fun-eq-iff* [*THEN iffD2*]) (*simp add*: *starset-n-def starset-def star-of-def*)

8.3 Theorems about nonstandard extensions of functions

Nonstandard extension of a function (defined using a constant sequence) as a special case of an internal function.

lemma *starfun-n-starfun*: $F = (\lambda n.\ f) \Longrightarrow *fn*\ F = *f*\ f$
by (*simp add*: *starfun-n-def starfun-def star-of-def*)

Prove that *abs* for hypreal is a nonstandard extension of abs for real w/o use of congruence property (proved after this for general nonstandard extensions of real valued functions).

Proof now Uses the ultrafilter tactic!

lemma *hrabs-is-starext-rabs*: *is-starext abs abs*
proof −
have $\exists\, f{\in}\textit{Rep-star}\ (\textit{star-n}\ h).\ \exists\, g{\in}\textit{Rep-star}\ (\textit{star-n}\ k).\ (\textit{star-n}\ k = |\textit{star-n}\ h|) = (\forall_F\ n\ \textit{in}\ \mathcal{U}.\ (g\ n{::}'a) = |f\ n|)$
for $x\ y :: 'a\ \textit{star}$ **and** $h\ k$

by (*metis* (*full-types*) *Rep-star-star-n star-n-abs star-n-eq-iff*)
then show *?thesis*
unfolding *is-starext-def* **by** (*metis star-cases*)
qed

Nonstandard extension of functions.

lemma *starfun*: $(*f* \ f) \ (star\text{-}n \ X) = star\text{-}n \ (\lambda n. \ f \ (X \ n))$
by (*rule starfun-star-n*)

lemma *starfun-if-eq*: $\bigwedge w. \ w \neq star\text{-}of \ x \Longrightarrow (*f* \ (\lambda z. \ if \ z = x \ then \ a \ else \ g \ z)) \ w = (*f* \ g) \ w$
by *transfer simp*

Multiplication: $(*f) \ x \ (*g) = *(f \ x \ g)$

lemma *starfun-mult*: $\bigwedge x. \ (*f* \ f) \ x * (*f* \ g) \ x = (*f* \ (\lambda x. \ f \ x * g \ x)) \ x$
by *transfer* (*rule refl*)
declare *starfun-mult* [*symmetric*, *simp*]

Addition: $(*f) + (*g) = *(f + g)$

lemma *starfun-add*: $\bigwedge x. \ (*f* \ f) \ x + (*f* \ g) \ x = (*f* \ (\lambda x. \ f \ x + g \ x)) \ x$
by *transfer* (*rule refl*)
declare *starfun-add* [*symmetric*, *simp*]

Subtraction: $(*f) + -(*g) = *(f + -g)$

lemma *starfun-minus*: $\bigwedge x. \ - \ (*f* \ f) \ x = (*f* \ (\lambda x. \ - \ f \ x)) \ x$
by *transfer* (*rule refl*)
declare *starfun-minus* [*symmetric*, *simp*]

lemma *starfun-add-minus*: $\bigwedge x. \ (*f* \ f) \ x + -(*f* \ g) \ x = (*f* \ (\lambda x. \ f \ x + -g \ x)) \ x$
by *transfer* (*rule refl*)
declare *starfun-add-minus* [*symmetric*, *simp*]

lemma *starfun-diff*: $\bigwedge x. \ (*f* \ f) \ x \ - (*f* \ g) \ x = (*f* \ (\lambda x. \ f \ x - g \ x)) \ x$
by *transfer* (*rule refl*)
declare *starfun-diff* [*symmetric*, *simp*]

Composition: $(*f) \circ (*g) = *(f \circ g)$

lemma *starfun-o2*: $(\lambda x. \ (*f* \ f) \ ((*f* \ g) \ x)) = *f* \ (\lambda x. \ f \ (g \ x))$
by *transfer* (*rule refl*)

lemma *starfun-o*: $(*f* \ f) \circ (*f* \ g) = (*f* \ (f \circ g))$
by (*transfer o-def*) (*rule refl*)

NS extension of constant function.

lemma *starfun-const-fun* [*simp*]: $\bigwedge x. \ (*f* \ (\lambda x. \ k)) \ x = star\text{-}of \ k$
by *transfer* (*rule refl*)

The NS extension of the identity function.

lemma *starfun-Id* [*simp*]: $\bigwedge x.\ (\ *f*\ (\lambda x.\ x))\ x = x$
 by *transfer* (*rule refl*)

The Star-function is a (nonstandard) extension of the function.

lemma *is-starext-starfun*: *is-starext* (*∗f∗ f*) *f*
proof −
 have $\exists X \in Rep\text{-}star\ x.\ \exists Y \in Rep\text{-}star\ y.\ (y = (*f*\ f)\ x) = (\forall_F\ n\ in\ \mathcal{U}.\ Y\ n = f\ (X\ n))$
  for *x y*
  by (*metis* (*mono-tags*) *Rep-star-star-n star-cases star-n-eq-iff starfun-star-n*)
 then show *?thesis*
  by (*auto simp*: *is-starext-def*)
qed

Any nonstandard extension is in fact the Star-function.

lemma *is-starfun-starext*:
 assumes *is-starext F f*
 shows $F = *f*\ f$
 proof −
 have $F\ x = (*f*\ f)\ x$
  if $\forall x\ y.\ \exists X \in Rep\text{-}star\ x.\ \exists Y \in Rep\text{-}star\ y.\ (y = F\ x) = (\forall_F\ n\ in\ \mathcal{U}.\ Y\ n = f\ (X\ n))$ **for** *x*
  by (*metis that mem-Rep-star-iff star-n-eq-iff starfun-star-n*)
 with *assms* **show** *?thesis*
  by (*force simp add*: *is-starext-def*)
qed

lemma *is-starext-starfun-iff*: *is-starext F f* $\longleftrightarrow F = *f*\ f$
 by (*blast intro*: *is-starfun-starext is-starext-starfun*)

Extended function has same solution as its standard version for real arguments. i.e they are the same for all real arguments.

lemma *starfun-eq*: (*∗f∗ f*) (*star-of a*) = *star-of* (*f a*)
 by (*rule starfun-star-of*)

lemma *starfun-approx*: (*∗f∗ f*) (*star-of a*) ≈ *star-of* (*f a*)
 by *simp*

Useful for NS definition of derivatives.

lemma *starfun-lambda-cancel*: $\bigwedge x'.\ (\ *f*\ (\lambda h.\ f\ (x + h)))\ x' = (\ *f*\ f)\ (star\text{-}of\ x + x')$
 by *transfer* (*rule refl*)

lemma *starfun-lambda-cancel2*: $(\ *f*\ (\lambda h.\ f\ (g\ (x + h))))\ x' = (\ *f*\ (f \circ g))\ (star\text{-}of\ x + x')$
 unfolding *o-def* **by** (*rule starfun-lambda-cancel*)

lemma *starfun-mult-HFinite-approx*:
$(*f* \ f) \ x \approx l \Longrightarrow (*f* \ g) \ x \approx m \Longrightarrow l \in HFinite \Longrightarrow m \in HFinite \Longrightarrow$
$(*f* \ (\lambda x. \ f \ x * g \ x)) \ x \approx l * m$
for *l m* :: *'a::real-normed-algebra star*
using *approx-mult-HFinite* **by** *auto*

lemma *starfun-add-approx*: $(*f* \ f) \ x \approx l \Longrightarrow (*f* \ g) \ x \approx m \Longrightarrow (*f* \ (\%x. \ f \ x + g \ x)) \ x \approx l + m$
by (*auto intro*: *approx-add*)

Examples: *hrabs* is nonstandard extension of *rabs*, *inverse* is nonstandard extension of *inverse*.

Can be proved easily using theorem *starfun* and properties of ultrafilter as for inverse below we use the theorem we proved above instead.

lemma *starfun-rabs-hrabs*: $*f* \ abs = abs$
by (*simp only*: *star-abs-def*)

lemma *starfun-inverse-inverse* [*simp*]: $(*f* \ inverse) \ x = inverse \ x$
by (*simp only*: *star-inverse-def*)

lemma *starfun-inverse*: $\bigwedge x. \ inverse \ ((*f* \ f) \ x) = (*f* \ (\lambda x. \ inverse \ (f \ x))) \ x$
by *transfer* (*rule refl*)
declare *starfun-inverse* [*symmetric*, *simp*]

lemma *starfun-divide*: $\bigwedge x. \ (*f* \ f) \ x \ / \ (*f* \ g) \ x = (*f* \ (\lambda x. \ f \ x \ / \ g \ x)) \ x$
by *transfer* (*rule refl*)
declare *starfun-divide* [*symmetric*, *simp*]

lemma *starfun-inverse2*: $\bigwedge x. \ inverse \ ((*f* \ f) \ x) = (*f* \ (\lambda x. \ inverse \ (f \ x))) \ x$
by *transfer* (*rule refl*)

General lemma/theorem needed for proofs in elementary topology of the reals.

lemma *starfun-mem-starset*: $\bigwedge x. \ (*f* \ f) \ x \in *s* \ A \Longrightarrow x \in *s* \ \{x. \ f \ x \in A\}$
by *transfer simp*

Alternative definition for *hrabs* with *rabs* function applied entrywise to equivalence class representative. This is easily proved using *starfun* and ns extension thm.

lemma *hypreal-hrabs*: $|star\text{-}n \ X| = star\text{-}n \ (\lambda n. \ |X \ n|)$
by (*simp only*: *starfun-rabs-hrabs* [*symmetric*] *starfun*)

Nonstandard extension of set through nonstandard extension of *rabs* function i.e. *hrabs*. A more general result should be where we replace *rabs* by some arbitrary function *f* and *hrabs* by its NS extenson. See second NS set extension below.

lemma *STAR-rabs-add-minus*: *∗s∗ {x. |x + − y| < r} = {x. |x + −star-of y| < star-of r}*
by *transfer* (*rule refl*)

lemma *STAR-starfun-rabs-add-minus*:
∗s∗ {x. |f x + − y| < r} = {x. |(∗f∗ f) x + −star-of y| < star-of r}
by *transfer* (*rule refl*)

Another characterization of Infinitesimal and one of ≈ relation. In this theory since *hypreal-hrabs* proved here. Maybe move both theorems??

lemma *Infinitesimal-FreeUltrafilterNat-iff2*:
star-n X ∈ Infinitesimal ⟷ (∀ m. eventually (λn. norm (X n) < inverse (real (Suc m))) 𝒰)
by (*simp add*: *Infinitesimal-hypreal-of-nat-iff star-of-def hnorm-def star-of-nat-def starfun-star-n star-n-inverse star-n-less*)

lemma *HNatInfinite-inverse-Infinitesimal* [*simp*]:
assumes *n ∈ HNatInfinite*
shows *inverse* (*hypreal-of-hypnat n*) *∈ Infinitesimal*
proof (*cases n*)
case (*star-n X*)
then have ∗: ⋀*k*. ∀$_F$ *n in 𝒰. k < X n*
using *HNatInfinite-FreeUltrafilterNat assms* **by** *blast*
have ∀$_F$ *n in 𝒰. inverse* (*real* (*X n*)) *< inverse* (*1* + *real m*) **for** *m*
using ∗ [*of Suc m*] **by** (*auto elim*!: *eventually-mono*)
then show *?thesis*
using *star-n* **by** (*auto simp*: *of-hypnat-def starfun-star-n star-n-inverse Infinitesimal-FreeUltrafilterNat-iff2*)
qed

lemma *approx-FreeUltrafilterNat-iff*:
star-n X ≈ star-n Y ⟷ (∀ r>0. eventually (λn. norm (X n − Y n) < r) 𝒰)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (*star-n X − star-n Y ≈ 0*)
using *approx-minus-iff* **by** *blast*
also have ... = *?rhs*
by (*metis* (*full-types*) *Infinitesimal-FreeUltrafilterNat-iff mem-infmal-iff star-n-diff*)
finally show *?thesis* **.**
qed

lemma *approx-FreeUltrafilterNat-iff2*:
star-n X ≈ star-n Y ⟷ (∀ m. eventually (λn. norm (X n − Y n) < inverse (real (Suc m))) 𝒰)
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (*star-n X − star-n Y ≈ 0*)
using *approx-minus-iff* **by** *blast*
also have ... = *?rhs*

by (*metis* (*full-types*) *Infinitesimal-FreeUltrafilterNat-iff2 mem-infmal-iff star-n-diff*)
finally show *?thesis* **.**
qed

lemma *inj-starfun*: *inj starfun*
proof (*rule inj-onI*)
show $\varphi = \psi$ **if** *eq*: $*f*\ \varphi = *f*\ \psi$ **for** $\varphi\ \psi :: 'a \Rightarrow 'b$
proof (*rule ext*, *rule ccontr*)
show *False*
if $\varphi\ x \neq \psi\ x$ **for** x
by (*metis eq that star-of-inject starfun-eq*)
qed
qed

end

9 Star-transforms for the Hypernaturals

theory *NatStar*
imports *Star*
begin

lemma *star-n-eq-starfun-whn*: $star\text{-}n\ X = (\ *f*\ X)\ whn$
by (*simp add*: *hypnat-omega-def starfun-def star-of-def Ifun-star-n*)

lemma *starset-n-Un*: $*sn*\ (\lambda n.\ (A\ n) \cup (B\ n)) = *sn*\ A \cup *sn*\ B$
proof −
have $\bigwedge N.\ Iset\ ((*f*\ (\lambda n.\ \{x.\ x \in A\ n \lor x \in B\ n\}))\ N) =$
$\{x.\ x \in Iset\ ((*f*\ A)\ N) \lor x \in Iset\ ((*f*\ B)\ N)\}$
by *transfer simp*
then show *?thesis*
by (*simp add*: *starset-n-def star-n-eq-starfun-whn Un-def*)
qed

lemma *InternalSets-Un*: $X \in InternalSets \Longrightarrow Y \in InternalSets \Longrightarrow X \cup Y \in InternalSets$
by (*auto simp add*: *InternalSets-def starset-n-Un* [*symmetric*])

lemma *starset-n-Int*: $*sn*\ (\lambda n.\ A\ n \cap B\ n) = *sn*\ A \cap *sn*\ B$
proof −
have $\bigwedge N.\ Iset\ ((*f*\ (\lambda n.\ \{x.\ x \in A\ n \land x \in B\ n\}))\ N) =$
$\{x.\ x \in Iset\ ((*f*\ A)\ N) \land x \in Iset\ ((*f*\ B)\ N)\}$
by *transfer simp*
then show *?thesis*
by (*simp add*: *starset-n-def star-n-eq-starfun-whn Int-def*)
qed

lemma *InternalSets-Int*: $X \in InternalSets \Longrightarrow Y \in InternalSets \Longrightarrow X \cap Y \in InternalSets$

by (*auto simp add*: *InternalSets-def starset-n-Int* [*symmetric*])

lemma *starset-n-Compl*: $*sn*$ $((\lambda n. - A\ n)) = - (*sn*\ A)$
proof −
have $\bigwedge N.$ *Iset* $((*f*\ (\lambda n. \{x.\ x \notin A\ n\}))\ N) =$
$\{x.\ x \notin$ *Iset* $((*f*\ A)\ N)\}$
by *transfer simp*
then show *?thesis*
by (*simp add*: *starset-n-def star-n-eq-starfun-whn Compl-eq*)
qed

lemma *InternalSets-Compl*: $X \in$ *InternalSets* $\Longrightarrow - X \in$ *InternalSets*
by (*auto simp add*: *InternalSets-def starset-n-Compl* [*symmetric*])

lemma *starset-n-diff*: $*sn*\ (\lambda n.\ (A\ n) - (B\ n)) = *sn*\ A - *sn*\ B$
proof −
have $\bigwedge N.$ *Iset* $((*f*\ (\lambda n. \{x.\ x \in A\ n \wedge x \notin B\ n\}))\ N) =$
$\{x.\ x \in$ *Iset* $((*f*\ A)\ N) \wedge x \notin$ *Iset* $((*f*\ B)\ N)\}$
by *transfer simp*
then show *?thesis*
by (*simp add*: *starset-n-def star-n-eq-starfun-whn set-diff-eq*)
qed

lemma *InternalSets-diff*: $X \in$ *InternalSets* $\Longrightarrow Y \in$ *InternalSets* $\Longrightarrow X - Y \in$ *InternalSets*
by (*auto simp add*: *InternalSets-def starset-n-diff* [*symmetric*])

lemma *NatStar-SHNat-subset*: *Nats* $\leq *s*$ (*UNIV*:: *nat set*)
by *simp*

lemma *NatStar-hypreal-of-real-Int*: $*s*\ X$ *Int Nats* = *hypnat-of-nat* ‘ X
by (*auto simp add*: *SHNat-eq*)

lemma *starset-starset-n-eq*: $*s*\ X = *sn*\ (\lambda n.\ X)$
by (*simp add*: *starset-n-starset*)

lemma *InternalSets-starset-n* [*simp*]: $(*s*\ X) \in$ *InternalSets*
by (*auto simp add*: *InternalSets-def starset-starset-n-eq*)

lemma *InternalSets-UNIV-diff*: $X \in$ *InternalSets* $\Longrightarrow$ *UNIV* $- X \in$ *InternalSets*
by (*simp add*: *InternalSets-Compl diff-eq*)

9.1 Nonstandard Extensions of Functions

Example of transfer of a property from reals to hyperreals — used for limit comparison of sequences.

lemma *starfun-le-mono*: $\forall n.\ N \leq n \longrightarrow f\ n \leq g\ n \Longrightarrow$
$\forall n.$ *hypnat-of-nat* $N \leq n \longrightarrow (*f*\ f)\ n \leq (*f*\ g)\ n$
by *transfer*

And another:

lemma *starfun-less-mono*:
$\forall n.\ N \leq n \longrightarrow f\ n < g\ n \Longrightarrow \forall n.\ \textit{hypnat-of-nat}\ N \leq n \longrightarrow (\ {*f*}\ f)\ n < (\ {*f*}\ g)\ n$
by *transfer*

Nonstandard extension when we increment the argument by one.

lemma *starfun-shift-one*: $\bigwedge N.\ (\ {*f*}\ (\lambda n.\ f\ (\textit{Suc}\ n)))\ N = (\ {*f*}\ f)\ (N + (\textit{1}::\textit{hypnat}))$
by *transfer simp*

Nonstandard extension with absolute value.

lemma *starfun-abs*: $\bigwedge N.\ (\ {*f*}\ (\lambda n.\ |f\ n|))\ N = |(\ {*f*}\ f)\ N|$
by *transfer* (*rule refl*)

The *hyperpow* function as a nonstandard extension of *realpow*.

lemma *starfun-pow*: $\bigwedge N.\ (\ {*f*}\ (\lambda n.\ r\ \widehat{}\ n))\ N = \textit{hypreal-of-real}\ r\ \textit{pow}\ N$
by *transfer* (*rule refl*)

lemma *starfun-pow2*: $\bigwedge N.\ (\ {*f*}\ (\lambda n.\ X\ n\ \widehat{}\ m))\ N = (\ {*f*}\ X)\ N\ \textit{pow}\ \textit{hypnat-of-nat}\ m$
by *transfer* (*rule refl*)

lemma *starfun-pow3*: $\bigwedge R.\ (\ {*f*}\ (\lambda r.\ r\ \widehat{}\ n))\ R = R\ \textit{pow}\ \textit{hypnat-of-nat}\ n$
by *transfer* (*rule refl*)

The *hypreal-of-hypnat* function as a nonstandard extension of *real*.

lemma *starfunNat-real-of-nat*: $(\ {*f*}\ \textit{real}) = \textit{hypreal-of-hypnat}$
by *transfer* (*simp add*: *fun-eq-iff*)

lemma *starfun-inverse-real-of-nat-eq*:
$N \in \textit{HNatInfinite} \Longrightarrow (\ {*f*}\ (\lambda x::\textit{nat}.\ \textit{inverse}\ (\textit{real}\ x)))\ N = \textit{inverse}\ (\textit{hypreal-of-hypnat}\ N)$
by (*metis of-hypnat-def starfun-inverse2*)

Internal functions – some redundancy with $*f*$ now.

lemma *starfun-n*: $(\ {*fn*}\ f)\ (\textit{star-n}\ X) = \textit{star-n}\ (\lambda n.\ f\ n\ (X\ n))$
by (*simp add*: *starfun-n-def Ifun-star-n*)

Multiplication: $(\ *fn)\ x\ (\ *gn) = *(fn\ x\ gn)$

lemma *starfun-n-mult*: $(\ {*fn*}\ f)\ z * (\ {*fn*}\ g)\ z = (\ {*fn*}\ (\lambda i\ x.\ f\ i\ x * g\ i\ x))\ z$
by (*cases z*) (*simp add*: *starfun-n star-n-mult*)

Addition: $(\ *fn) + (\ *gn) = *(fn + gn)$

lemma *starfun-n-add*: $(\ {*fn*}\ f)\ z + (\ {*fn*}\ g)\ z = (\ {*fn*}\ (\lambda i\ x.\ f\ i\ x + g\ i\ x))\ z$
by (*cases z*) (*simp add*: *starfun-n star-n-add*)

Subtraction: $(\ *fn) - (\ *gn) = *(fn + -\ gn)$

lemma *starfun-n-add-minus*: $(*fn* \ f) \ z + -(*fn* \ g) \ z = (*fn* \ (\lambda i \ x. \ f \ i \ x + -g \ i \ x)) \ z$
by (*cases z*) (*simp add*: *starfun-n star-n-minus star-n-add*)

Composition: $(*fn) \circ (*gn) = *(fn \circ gn)$

lemma *starfun-n-const-fun* [*simp*]: $(*fn* \ (\lambda i \ x. \ k)) \ z = star\text{-}of \ k$
by (*cases z*) (*simp add*: *starfun-n star-of-def*)

lemma *starfun-n-minus*: $- (*fn* \ f) \ x = (*fn* \ (\lambda i \ x. - (f \ i) \ x)) \ x$
by (*cases x*) (*simp add*: *starfun-n star-n-minus*)

lemma *starfun-n-eq* [*simp*]: $(*fn* \ f) \ (star\text{-}of \ n) = star\text{-}n \ (\lambda i. \ f \ i \ n)$
by (*simp add*: *starfun-n star-of-def*)

lemma *starfun-eq-iff*: $((*f* \ f) = (*f* \ g)) \longleftrightarrow f = g$
by *transfer* (*rule refl*)

lemma *starfunNat-inverse-real-of-nat-Infinitesimal* [*simp*]:
$N \in HNatInfinite \Longrightarrow (*f* \ (\lambda x. \ inverse \ (real \ x))) \ N \in Infinitesimal$
using *starfun-inverse-real-of-nat-eq* **by** *auto*

9.2 Nonstandard Characterization of Induction

lemma *hypnat-induct-obj*:
$\bigwedge n. \ ((*p* \ P) \ (0::hypnat) \land (\forall n. \ (*p* \ P) \ n \longrightarrow (*p* \ P) \ (n + 1))) \longrightarrow (*p* \ P) \ n$
by *transfer* (*induct-tac n*, *auto*)

lemma *hypnat-induct*:
$\bigwedge n. \ (*p* \ P) \ (0::hypnat) \Longrightarrow (\bigwedge n. \ (*p* \ P) \ n \Longrightarrow (*p* \ P) \ (n + 1)) \Longrightarrow (*p* \ P) \ n$
by *transfer* (*induct-tac n*, *auto*)

lemma *starP2-eq-iff*: $(*p2* \ (=)) = (=)$
by *transfer* (*rule refl*)

lemma *starP2-eq-iff2*: $(*p2* \ (\lambda x \ y. \ x = y)) \ X \ Y \longleftrightarrow X = Y$
by (*simp add*: *starP2-eq-iff*)

lemma *nonempty-set-star-has-least-lemma*:
$\exists n \in S. \ \forall m \in S. \ n \leq m$ **if** $S \neq \{\}$ **for** $S :: nat \ set$
proof
show $\forall m \in S. \ (LEAST \ n. \ n \in S) \leq m$
by (*simp add*: *Least-le*)
show $(LEAST \ n. \ n \in S) \in S$
by (*meson that LeastI-ex equals0I*)
qed

lemma *nonempty-set-star-has-least*:

$\bigwedge S$::*nat set star*. *Iset S* $\neq$ {} $\Longrightarrow$ $\exists$ *n* $\in$ *Iset S*. $\forall$ *m* $\in$ *Iset S*. *n* $\leq$ *m*
using *nonempty-set-star-has-least-lemma* **by** (*transfer empty-def*)

lemma *nonempty-InternalNatSet-has-least*: *S* $\in$ *InternalSets* $\Longrightarrow$ *S* $\neq$ {} $\Longrightarrow$ $\exists$ *n* $\in$ *S*. $\forall$ *m* $\in$ *S*. *n* $\leq$ *m*
for *S* :: *hypnat set*
by (*force simp add*: *InternalSets-def starset-n-def dest*!: *nonempty-set-star-has-least*)

Goldblatt, page 129 Thm 11.3.2.

lemma *internal-induct-lemma*:
$\bigwedge X$::*nat set star*.
(*0*::*hypnat*) $\in$ *Iset X* $\Longrightarrow$ $\forall$ *n*. *n* $\in$ *Iset X* $\longrightarrow$ *n* + *1* $\in$ *Iset X* $\Longrightarrow$ *Iset X* = (*UNIV*:: *hypnat set*)
apply (*transfer UNIV-def*)
apply (*rule equalityI* [*OF subset-UNIV subsetI*])
apply (*induct-tac x*, *auto*)
done

lemma *internal-induct*:
X $\in$ *InternalSets* $\Longrightarrow$ (*0*::*hypnat*) $\in$ *X* $\Longrightarrow$ $\forall$ *n*. *n* $\in$ *X* $\longrightarrow$ *n* + *1* $\in$ *X* $\Longrightarrow$ *X* = (*UNIV*:: *hypnat set*)
apply (*clarsimp simp add*: *InternalSets-def starset-n-def*)
apply (*erule* (*1*) *internal-induct-lemma*)
done

end

10 Sequences and Convergence (Nonstandard)

theory *HSEQ*
imports *Complex-Main NatStar*
abbrevs −−−> = $\longrightarrow_{NS}$
begin

definition *NSLIMSEQ* :: (*nat* ⇒ *'a*::*real-normed-vector*) ⇒ *'a* ⇒ *bool*
(‹(‹*notation*=‹*mixfix NSLIMSEQ*››(-)/ $\longrightarrow_{NS}$ (-))› [*60*, *60*] *60*) **where**
— Nonstandard definition of convergence of sequence
X $\longrightarrow_{NS}$ *L* $\longleftrightarrow$ ($\forall$ *N* $\in$ *HNatInfinite*. (**f* X*) *N* $\approx$ *star-of L*)

definition *nslim* :: (*nat* ⇒ *'a*::*real-normed-vector*) ⇒ *'a*
where *nslim X* = (*THE L*. *X* $\longrightarrow_{NS}$ *L*)
— Nonstandard definition of limit using choice operator

definition *NSconvergent* :: (*nat* ⇒ *'a*::*real-normed-vector*) ⇒ *bool*
where *NSconvergent X* $\longleftrightarrow$ ($\exists$ *L*. *X* $\longrightarrow_{NS}$ *L*)
— Nonstandard definition of convergence

definition *NSBseq* :: (*nat* ⇒ *'a*::*real-normed-vector*) ⇒ *bool*

where *NSBseq X* ⟷ (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ∈ *HFinite*)
— Nonstandard definition for bounded sequence

definition *NSCauchy* :: (*nat* ⇒ ′*a*::*real-normed-vector*) ⇒ *bool*
where *NSCauchy X* ⟷ (∀ *M* ∈ *HNatInfinite*. ∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *M* ≈ (∗*f*∗ *X*) *N*)
— Nonstandard definition

10.1 Limits of Sequences

lemma *NSLIMSEQ-I*: (⋀*N*. *N* ∈ *HNatInfinite* ⟹ *starfun X N* ≈ *star-of L*) ⟹ $X \longrightarrow_{NS} L$
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-D*: $X \longrightarrow_{NS} L$ ⟹ *N* ∈ *HNatInfinite* ⟹ *starfun X N* ≈ *star-of L*
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-const*: $(\lambda n.\ k) \longrightarrow_{NS} k$
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-add*: $X \longrightarrow_{NS} a \Longrightarrow Y \longrightarrow_{NS} b \Longrightarrow (\lambda n.\ X\ n + Y\ n) \longrightarrow_{NS} a + b$
by (*auto intro*: *approx-add simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-add-const*: $f \longrightarrow_{NS} a \Longrightarrow (\lambda n.\ f\ n + b) \longrightarrow_{NS} a + b$
by (*simp only*: *NSLIMSEQ-add NSLIMSEQ-const*)

lemma *NSLIMSEQ-mult*: $X \longrightarrow_{NS} a \Longrightarrow Y \longrightarrow_{NS} b \Longrightarrow (\lambda n.\ X\ n * Y\ n) \longrightarrow_{NS} a * b$
for *a b* :: ′*a*::*real-normed-algebra*
by (*auto intro*!: *approx-mult-HFinite simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-minus*: $X \longrightarrow_{NS} a \Longrightarrow (\lambda n.\ -\ X\ n) \longrightarrow_{NS} -\ a$
by (*auto simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-minus-cancel*: $(\lambda n.\ -\ X\ n) \longrightarrow_{NS} -a \Longrightarrow X \longrightarrow_{NS} a$
by (*drule NSLIMSEQ-minus*) *simp*

lemma *NSLIMSEQ-diff*: $X \longrightarrow_{NS} a \Longrightarrow Y \longrightarrow_{NS} b \Longrightarrow (\lambda n.\ X\ n - Y\ n) \longrightarrow_{NS} a - b$
using *NSLIMSEQ-add* [*of X a* − *Y* − *b*] **by** (*simp add*: *NSLIMSEQ-minus fun-Compl-def*)

lemma *NSLIMSEQ-diff-const*: $f \longrightarrow_{NS} a \Longrightarrow (\lambda n.\ f\ n - b) \longrightarrow_{NS} a - b$
by (*simp add*: *NSLIMSEQ-diff NSLIMSEQ-const*)

lemma *NSLIMSEQ-inverse*: $X \longrightarrow_{NS} a \Longrightarrow a \neq 0 \Longrightarrow (\lambda n.\ inverse\ (X\ n))$

$\longrightarrow_{NS}$ *inverse a*
for *a* :: *'a::real-normed-div-algebra*
by (*simp add*: *NSLIMSEQ-def star-of-approx-inverse*)

lemma *NSLIMSEQ-mult-inverse*: $X \longrightarrow_{NS} a \Longrightarrow Y \longrightarrow_{NS} b \Longrightarrow b \neq 0 \Longrightarrow (\lambda n.\ X\ n\ /\ Y\ n) \longrightarrow_{NS} a\ /\ b$
for *a b* :: *'a::real-normed-field*
by (*simp add*: *NSLIMSEQ-mult NSLIMSEQ-inverse divide-inverse*)

lemma *starfun-hnorm*: $\bigwedge x.\ hnorm\ ((\ {*}f{*}\ f)\ x) = (\ {*}f{*}\ (\lambda x.\ norm\ (f\ x)))\ x$
by *transfer simp*

lemma *NSLIMSEQ-norm*: $X \longrightarrow_{NS} a \Longrightarrow (\lambda n.\ norm\ (X\ n)) \longrightarrow_{NS} norm\ a$
by (*simp add*: *NSLIMSEQ-def starfun-hnorm* [*symmetric*] *approx-hnorm*)

Uniqueness of limit.

lemma *NSLIMSEQ-unique*: $X \longrightarrow_{NS} a \Longrightarrow X \longrightarrow_{NS} b \Longrightarrow a = b$
unfolding *NSLIMSEQ-def*
using *HNatInfinite-whn approx-trans3 star-of-approx-iff* **by** *blast*

lemma *NSLIMSEQ-pow* [*rule-format*]: $(X \longrightarrow_{NS} a) \longrightarrow ((\lambda n.\ (X\ n)\ \hat{}\ m) \longrightarrow_{NS} a\ \hat{}\ m)$
for *a* :: *'a::{real-normed-algebra,power}*
by (*induct m*) (*auto intro*: *NSLIMSEQ-mult NSLIMSEQ-const*)

We can now try and derive a few properties of sequences, starting with the limit comparison property for sequences.

lemma *NSLIMSEQ-le*: $f \longrightarrow_{NS} l \Longrightarrow g \longrightarrow_{NS} m \Longrightarrow \exists N.\ \forall n \geq N.\ f\ n \leq g\ n \Longrightarrow l \leq m$
for *l m* :: *real*
unfolding *NSLIMSEQ-def*
by (*metis HNatInfinite-whn bex-Infinitesimal-iff2 hypnat-of-nat-le-whn hypreal-of-real-le-add-Infininitesimal-c starfun-le-mono*)

lemma *NSLIMSEQ-le-const*: $X \longrightarrow_{NS} r \Longrightarrow \forall n.\ a \leq X\ n \Longrightarrow a \leq r$
for *a r* :: *real*
by (*erule NSLIMSEQ-le* [*OF NSLIMSEQ-const*]) *auto*

lemma *NSLIMSEQ-le-const2*: $X \longrightarrow_{NS} r \Longrightarrow \forall n.\ X\ n \leq a \Longrightarrow r \leq a$
for *a r* :: *real*
by (*erule NSLIMSEQ-le* [*OF - NSLIMSEQ-const*]) *auto*

Shift a convergent series by 1: By the equivalence between Cauchiness and convergence and because the successor of an infinite hypernatural is also infinite.

lemma *NSLIMSEQ-Suc-iff*: $((\lambda n.\ f\ (Suc\ n)) \longrightarrow_{NS} l) \longleftrightarrow (f \longrightarrow_{NS} l)$
proof

assume $*$: $f \longrightarrow_{NS} l$
show $(\lambda n.\ f(Suc\ n)) \longrightarrow_{NS} l$
proof (*rule NSLIMSEQ-I*)
fix *N*
assume $N \in HNatInfinite$
then have $({*}f{*}\ f)\ (N + 1) \approx$ *star-of l*
by (*simp add*: *HNatInfinite-add NSLIMSEQ-D* $*$)
then show $({*}f{*}\ (\lambda n.\ f\ (Suc\ n)))\ N \approx$ *star-of l*
by (*simp add*: *starfun-shift-one*)
qed
next
assume $*$: $(\lambda n.\ f(Suc\ n)) \longrightarrow_{NS} l$
show $f \longrightarrow_{NS} l$
proof (*rule NSLIMSEQ-I*)
fix *N*
assume $N \in HNatInfinite$
then have $({*}f{*}\ (\lambda n.\ f\ (Suc\ n)))\ (N - 1) \approx$ *star-of l*
using $*$ **by** (*simp add*: *HNatInfinite-diff NSLIMSEQ-D*)
then show $({*}f{*}\ f)\ N \approx$ *star-of l*
by (*simp add*: ‹$N \in HNatInfinite$› *one-le-HNatInfinite starfun-shift-one*)
qed
qed

10.1.1 Equivalence of *LIMSEQ* and *NSLIMSEQ*

lemma *LIMSEQ-NSLIMSEQ*:
assumes *X*: $X \longrightarrow L$
shows $X \longrightarrow_{NS} L$
proof (*rule NSLIMSEQ-I*)
fix *N*
assume *N*: $N \in HNatInfinite$
have *starfun X N* $-$ *star-of L* $\in$ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix *r* :: *real*
assume *r*: $0 < r$
from *LIMSEQ-D* [*OF X r*] **obtain** *no* **where** $\forall n \geq no.\ norm\ (X\ n - L) < r$ **..**
then have $\forall n \geq$ *star-of no*. *hnorm* (*starfun X n* $-$ *star-of L*) $<$ *star-of r*
by *transfer*
then show *hnorm* (*starfun X N* $-$ *star-of L*) $<$ *star-of r*
using *N* **by** (*simp add*: *star-of-le-HNatInfinite*)
qed
then show *starfun X N* $\approx$ *star-of L*
by (*simp only*: *approx-def*)
qed

lemma *NSLIMSEQ-LIMSEQ*:
assumes *X*: $X \longrightarrow_{NS} L$
shows $X \longrightarrow L$
proof (*rule LIMSEQ-I*)

fix *r* :: *real*
assume *r*: *0* < *r*
have ∃ *no*. ∀ *n*≥*no*. *hnorm* (*starfun X n* − *star-of L*) < *star-of r*
proof (*intro exI allI impI*)
fix *n*
assume *whn* ≤ *n*
with *HNatInfinite-whn* **have** *n* ∈ *HNatInfinite*
by (*rule HNatInfinite-upward-closed*)
with *X* **have** *starfun X n* ≈ *star-of L*
by (*rule NSLIMSEQ-D*)
then have *starfun X n* − *star-of L* ∈ *Infinitesimal*
by (*simp only*: *approx-def*)
then show *hnorm* (*starfun X n* − *star-of L*) < *star-of r*
using *r* **by** (*rule InfinitesimalD2*)
qed
then show ∃ *no*. ∀ *n*≥*no*. *norm* (*X n* − *L*) < *r*
by *transfer*
qed

theorem *LIMSEQ-NSLIMSEQ-iff*: $f \longrightarrow L \longleftrightarrow f \longrightarrow_{NS} L$
by (*blast intro*: *LIMSEQ-NSLIMSEQ NSLIMSEQ-LIMSEQ*)

10.1.2 Derived theorems about *NSLIMSEQ*

We prove the NS version from the standard one, since the NS proof seems more complicated than the standard one above!

lemma *NSLIMSEQ-norm-zero*: $(\lambda n.\ norm\ (X\ n)) \longrightarrow_{NS} 0 \longleftrightarrow X \longrightarrow_{NS} 0$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *tendsto-norm-zero-iff*)

lemma *NSLIMSEQ-rabs-zero*: $(\lambda n.\ |f\ n|) \longrightarrow_{NS} 0 \longleftrightarrow f \longrightarrow_{NS} (0{::}real)$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *tendsto-rabs-zero-iff*)

Generalization to other limits.

lemma *NSLIMSEQ-imp-rabs*: $f \longrightarrow_{NS} l \Longrightarrow (\lambda n.\ |f\ n|) \longrightarrow_{NS} |l|$
for *l* :: *real*
by (*simp add*: *NSLIMSEQ-def*) (*auto intro*: *approx-hrabs simp add*: *starfun-abs*)

lemma *NSLIMSEQ-inverse-zero*: $\forall y{::}real.\ \exists N.\ \forall n \geq N.\ y < f\ n \Longrightarrow (\lambda n.\ inverse\ (f\ n)) \longrightarrow_{NS} 0$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-zero*)

lemma *NSLIMSEQ-inverse-real-of-nat*: $(\lambda n.\ inverse\ (real\ (Suc\ n))) \longrightarrow_{NS} 0$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-real-of-nat* *del*: *of-nat-Suc*)

lemma *NSLIMSEQ-inverse-real-of-nat-add*: $(\lambda n.\ r + inverse\ (real\ (Suc\ n))) \longrightarrow_{NS} r$

by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-real-of-nat-add* *del*: *of-nat-Suc*)

lemma *NSLIMSEQ-inverse-real-of-nat-add-minus*: $(\lambda n.\ r + -\ inverse\ (real\ (Suc\ n))) \longrightarrow_{NS} r$
using *LIMSEQ-inverse-real-of-nat-add-minus* **by** (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

lemma *NSLIMSEQ-inverse-real-of-nat-add-minus-mult*:
$(\lambda n.\ r * (1 + -\ inverse\ (real\ (Suc\ n)))) \longrightarrow_{NS} r$
using *LIMSEQ-inverse-real-of-nat-add-minus-mult*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

10.2 Convergence

lemma *nslimI*: $X \longrightarrow_{NS} L \Longrightarrow nslim\ X = L$
by (*simp add*: *nslim-def*) (*blast intro*: *NSLIMSEQ-unique*)

lemma *lim-nslim-iff*: $lim\ X = nslim\ X$
by (*simp add*: *lim-def nslim-def LIMSEQ-NSLIMSEQ-iff*)

lemma *NSconvergentD*: $NSconvergent\ X \Longrightarrow \exists L.\ X \longrightarrow_{NS} L$
by (*simp add*: *NSconvergent-def*)

lemma *NSconvergentI*: $X \longrightarrow_{NS} L \Longrightarrow NSconvergent\ X$
by (*auto simp add*: *NSconvergent-def*)

lemma *convergent-NSconvergent-iff*: $convergent\ X = NSconvergent\ X$
by (*simp add*: *convergent-def NSconvergent-def LIMSEQ-NSLIMSEQ-iff*)

lemma *NSconvergent-NSLIMSEQ-iff*: $NSconvergent\ X \longleftrightarrow X \longrightarrow_{NS} nslim\ X$
by (*auto intro*: *theI NSLIMSEQ-unique simp add*: *NSconvergent-def nslim-def*)

10.3 Bounded Monotonic Sequences

lemma *NSBseqD*: $NSBseq\ X \Longrightarrow N \in HNatInfinite \Longrightarrow (\ *f*\ X)\ N \in HFinite$
by (*simp add*: *NSBseq-def*)

lemma *Standard-subset-HFinite*: $Standard \subseteq HFinite$
by (*auto simp*: *Standard-def*)

lemma *NSBseqD2*: $NSBseq\ X \Longrightarrow (\ *f*\ X)\ N \in HFinite$
using *HNatInfinite-def NSBseq-def Nats-eq-Standard Standard-starfun Standard-subset-HFinite*
by *blast*

lemma *NSBseqI*: $\forall N \in HNatInfinite.\ (\ *f*\ X)\ N \in HFinite \Longrightarrow NSBseq\ X$
by (*simp add*: *NSBseq-def*)

The standard definition implies the nonstandard definition.

lemma *Bseq-NSBseq*: $Bseq\ X \Longrightarrow NSBseq\ X$

unfolding *NSBseq-def*
proof *safe*
assume *X*: *Bseq X*
fix *N*
assume *N*: *N ∈ HNatInfinite*
from *BseqD* [*OF X*] **obtain** *K* **where** ∀ *n*. *norm* (*X n*) ≤ *K*
by *fast*
then have ∀ *N*. *hnorm* (*starfun X N*) ≤ *star-of K*
by *transfer*
then have *hnorm* (*starfun X N*) ≤ *star-of K*
by *simp*
also have *star-of K* < *star-of* (*K* + *1*)
by *simp*
finally have ∃ *x*∈*Reals*. *hnorm* (*starfun X N*) < *x*
by (*rule bexI*) *simp*
then show *starfun X N* ∈ *HFinite*
by (*simp add*: *HFinite-def*)
qed

The nonstandard definition implies the standard definition.

lemma *SReal-less-omega*: *r* ∈ ℝ ⟹ *r* < ω
using *HInfinite-omega*
by (*simp add*: *HInfinite-def*) (*simp add*: *order-less-imp-le*)

lemma *NSBseq-Bseq*: *NSBseq X* ⟹ *Bseq X*
proof (*rule ccontr*)
let *?n* = λ*K*. *LEAST n*. *K* < *norm* (*X n*)
assume *NSBseq X*
then have *finite*: (*∗f∗ X*) ((*∗f∗ ?n*) ω) ∈ *HFinite*
by (*rule NSBseqD2*)
assume ¬ *Bseq X*
then have ∀ *K*>*0*. ∃ *n*. *K* < *norm* (*X n*)
by (*simp add*: *Bseq-def linorder-not-le*)
then have ∀ *K*>*0*. *K* < *norm* (*X* (*?n K*))
by (*auto intro*: *LeastI-ex*)
then have ∀ *K*>*0*. *K* < *hnorm* ((*∗f∗ X*) ((*∗f∗ ?n*) *K*))
by *transfer*
then have ω < *hnorm* ((*∗f∗ X*) ((*∗f∗ ?n*) ω))
by *simp*
then have ∀ *r*∈ℝ. *r* < *hnorm* ((*∗f∗ X*) ((*∗f∗ ?n*) ω))
by (*simp add*: *order-less-trans* [*OF SReal-less-omega*])
then have (*∗f∗ X*) ((*∗f∗ ?n*) ω) ∈ *HInfinite*
by (*simp add*: *HInfinite-def*)
with *finite* **show** *False*
by (*simp add*: *HFinite-HInfinite-iff*)
qed

Equivalence of nonstandard and standard definitions for a bounded sequence.

lemma *Bseq-NSBseq-iff*: *Bseq X* = *NSBseq X*
by (*blast intro*!: *NSBseq-Bseq Bseq-NSBseq*)

A convergent sequence is bounded: Boundedness as a necessary condition for convergence. The nonstandard version has no existential, as usual.

lemma *NSconvergent-NSBseq*: *NSconvergent X* $\Longrightarrow$ *NSBseq X*
by (*simp add*: *NSconvergent-def NSBseq-def NSLIMSEQ-def*)
(*blast intro*: *HFinite-star-of approx-sym approx-HFinite*)

Standard Version: easily now proved using equivalence of NS and standard definitions.

lemma *convergent-Bseq*: *convergent X* $\Longrightarrow$ *Bseq X*
for *X* :: *nat* $\Rightarrow$ *'b*::*real-normed-vector*
by (*simp add*: *NSconvergent-NSBseq convergent-NSconvergent-iff Bseq-NSBseq-iff*)

10.3.1 Upper Bounds and Lubs of Bounded Sequences

lemma *NSBseq-isUb*: *NSBseq X* $\Longrightarrow \exists$ *U*::*real*. *isUb UNIV* {*x*. $\exists$ *n*. *X n* = *x*} *U*
by (*simp add*: *Bseq-NSBseq-iff* [*symmetric*] *Bseq-isUb*)

lemma *NSBseq-isLub*: *NSBseq X* $\Longrightarrow \exists$ *U*::*real*. *isLub UNIV* {*x*. $\exists$ *n*. *X n* = *x*} *U*
by (*simp add*: *Bseq-NSBseq-iff* [*symmetric*] *Bseq-isLub*)

10.3.2 A Bounded and Monotonic Sequence Converges

The best of both worlds: Easier to prove this result as a standard theorem and then use equivalence to "transfer" it into the equivalent nonstandard form if needed!

lemma *Bmonoseq-NSLIMSEQ*: $\forall_F$ *k in sequentially*. *X k* = *X m* $\Longrightarrow$ *X* $\longrightarrow_{NS}$ *X m*
unfolding *LIMSEQ-NSLIMSEQ-iff*[*symmetric*]
by (*simp add*: *eventually-mono eventually-nhds-x-imp-x filterlim-iff*)

lemma *NSBseq-mono-NSconvergent*: *NSBseq X* $\Longrightarrow \forall$ *m*. $\forall$ *n* $\geq$ *m*. *X m* $\leq$ *X n* $\Longrightarrow$ *NSconvergent X*
for *X* :: *nat* $\Rightarrow$ *real*
by (*auto intro*: *Bseq-mono-convergent*
simp: *convergent-NSconvergent-iff* [*symmetric*] *Bseq-NSBseq-iff* [*symmetric*])

10.4 Cauchy Sequences

lemma *NSCauchyI*:
($\bigwedge$*M N*. *M* $\in$ *HNatInfinite* $\Longrightarrow$ *N* $\in$ *HNatInfinite* $\Longrightarrow$ *starfun X M* $\approx$ *starfun X N*) $\Longrightarrow$ *NSCauchy X*
by (*simp add*: *NSCauchy-def*)

lemma *NSCauchyD*:

NSCauchy X ⟹ *M* ∈ *HNatInfinite* ⟹ *N* ∈ *HNatInfinite* ⟹ *starfun X M* ≈ *starfun X N*
by (*simp add*: *NSCauchy-def*)

10.4.1 Equivalence Between NS and Standard

lemma *Cauchy-NSCauchy*:
assumes *X*: *Cauchy X*
shows *NSCauchy X*
proof (*rule NSCauchyI*)
fix *M*
assume *M*: *M* ∈ *HNatInfinite*
fix *N*
assume *N*: *N* ∈ *HNatInfinite*
have *starfun X M* − *starfun X N* ∈ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix *r* :: *real*
assume *r*: *0* < *r*
from *CauchyD* [*OF X r*] **obtain** *k* **where** ∀ *m*≥*k*. ∀ *n*≥*k*. *norm* (*X m* − *X n*) < *r* **..**
then have ∀ *m*≥*star-of k*. ∀ *n*≥*star-of k*. *hnorm* (*starfun X m* − *starfun X n*) < *star-of r*
by *transfer*
then show *hnorm* (*starfun X M* − *starfun X N*) < *star-of r*
using *M N* **by** (*simp add*: *star-of-le-HNatInfinite*)
qed
then show *starfun X M* ≈ *starfun X N*
by (*simp only*: *approx-def*)
qed

lemma *NSCauchy-Cauchy*:
assumes *X*: *NSCauchy X*
shows *Cauchy X*
proof (*rule CauchyI*)
fix *r* :: *real*
assume *r*: *0* < *r*
have ∃ *k*. ∀ *m*≥*k*. ∀ *n*≥*k*. *hnorm* (*starfun X m* − *starfun X n*) < *star-of r*
proof (*intro exI allI impI*)
fix *M*
assume *whn* ≤ *M*
with *HNatInfinite-whn* **have** *M*: *M* ∈ *HNatInfinite*
by (*rule HNatInfinite-upward-closed*)
fix *N*
assume *whn* ≤ *N*
with *HNatInfinite-whn* **have** *N*: *N* ∈ *HNatInfinite*
by (*rule HNatInfinite-upward-closed*)
from *X M N* **have** *starfun X M* ≈ *starfun X N*
by (*rule NSCauchyD*)
then have *starfun X M* − *starfun X N* ∈ *Infinitesimal*

```
      by (simp only: approx-def)
    then show hnorm (starfun X M − starfun X N) < star-of r
      using r by (rule InfinitesimalD2)
  qed
  then show ∃ k. ∀ m≥k. ∀ n≥k. norm (X m − X n) < r
    by transfer
qed

theorem NSCauchy-Cauchy-iff: NSCauchy X = Cauchy X
  by (blast intro!: NSCauchy-Cauchy Cauchy-NSCauchy)
```

10.4.2 Cauchy Sequences are Bounded

A Cauchy sequence is bounded – nonstandard version.

```
lemma NSCauchy-NSBseq: NSCauchy X ⟹ NSBseq X
  by (simp add: Cauchy-Bseq Bseq-NSBseq-iff [symmetric] NSCauchy-Cauchy-iff)
```

10.4.3 Cauchy Sequences are Convergent

Equivalence of Cauchy criterion and convergence: We will prove this using our NS formulation which provides a much easier proof than using the standard definition. We do not need to use properties of subsequences such as boundedness, monotonicity etc... Compare with Harrison's corresponding proof in HOL which is much longer and more complicated. Of course, we do not have problems which he encountered with guessing the right instantiations for his 'espsilon-delta' proof(s) in this case since the NS formulations do not involve existential quantifiers.

```
lemma NSconvergent-NSCauchy: NSconvergent X ⟹ NSCauchy X
  by (simp add: NSconvergent-def NSLIMSEQ-def NSCauchy-def) (auto intro: approx-trans2)

lemma real-NSCauchy-NSconvergent:
  fixes X :: nat ⇒ real
  assumes NSCauchy X shows NSconvergent X
  unfolding NSconvergent-def NSLIMSEQ-def
proof −
  have ( ∗f∗ X) whn ∈ HFinite
    by (simp add: NSBseqD2 NSCauchy-NSBseq assms)
  moreover have ∀ N∈HNatInfinite. ( ∗f∗ X) whn ≈ ( ∗f∗ X) N
    using HNatInfinite-whn NSCauchy-def assms by blast
  ultimately show ∃ L. ∀ N∈HNatInfinite. ( ∗f∗ X) N ≈ hypreal-of-real L
    by (force dest!: st-part-Ex simp add: SReal-iff intro: approx-trans3)
qed

lemma NSCauchy-NSconvergent: NSCauchy X ⟹ NSconvergent X
  for X :: nat ⇒ 'a::banach
  using Cauchy-convergent NSCauchy-Cauchy convergent-NSconvergent-iff by auto
```

lemma *NSCauchy-NSconvergent-iff*: *NSCauchy X* = *NSconvergent X*
for *X* :: *nat* ⇒ *'a*::*banach*
by (*fast intro*: *NSCauchy-NSconvergent NSconvergent-NSCauchy*)

10.5 Power Sequences

The sequence x^n tends to 0 if $0 \leq x$ and $x < 1$. Proof will use (NS) Cauchy equivalence for convergence and also fact that bounded and monotonic sequence converges.

We now use NS criterion to bring proof of theorem through.

lemma *NSLIMSEQ-realpow-zero*:
fixes *x* :: *real*
assumes $0 \leq x$ $x < 1$ **shows** ($\lambda n.\ x \mathbin{\hat{}} n$) $\longrightarrow_{NS}$ *0*
proof −
have (∗*f*∗ ((^) *x*) *N* ≈ *0*
if *N*: *N* ∈ *HNatInfinite* **and** *x*: *NSconvergent* ((^) *x*) **for** *N*
proof −
have *hypreal-of-real x pow N* ≈ *hypreal-of-real x pow* (*N* + *1*)
by (*metis HNatInfinite-add N NSCauchy-NSconvergent-iff NSCauchy-def starfun-pow x*)
moreover obtain *L* **where** *L*: *hypreal-of-real x pow N* ≈ *hypreal-of-real L*
using *NSconvergentD* [*OF x*] *N* **by** (*auto simp add*: *NSLIMSEQ-def starfun-pow*)
ultimately have *hypreal-of-real x pow N* ≈ *hypreal-of-real L* ∗ *hypreal-of-real x*
by (*simp add*: *approx-mult-subst-star-of hyperpow-add*)
then have *hypreal-of-real L* ≈ *hypreal-of-real L* ∗ *hypreal-of-real x*
using *L approx-trans3* **by** *blast*
then show *?thesis*
by (*metis L* ‹*x* < *1*› *hyperpow-def less-irrefl mult.right-neutral mult-left-cancel star-of-approx-iff star-of-mult star-of-simps*(*9*) *starfun2-star-of*)
qed
with *assms* **show** *?thesis*
by (*force dest*!: *convergent-realpow simp add*: *NSLIMSEQ-def convergent-NSconvergent-iff*)
qed

lemma *NSLIMSEQ-abs-realpow-zero*: $|c| < 1 \Longrightarrow (\lambda n.\ |c| \mathbin{\hat{}} n) \longrightarrow_{NS} 0$
for *c* :: *real*
by (*simp add*: *LIMSEQ-abs-realpow-zero LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

lemma *NSLIMSEQ-abs-realpow-zero2*: $|c| < 1 \Longrightarrow (\lambda n.\ c \mathbin{\hat{}} n) \longrightarrow_{NS} 0$
for *c* :: *real*
by (*simp add*: *LIMSEQ-abs-realpow-zero2 LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

end

11 Finite Summation and Infinite Series for Hyperreals

theory *HSeries*
imports *HSEQ*
begin

definition *sumhr* :: *hypnat* × *hypnat* × (*nat* ⇒ *real*) ⇒ *hypreal*
where *sumhr* = (λ(*M*,*N*,*f*). *starfun2* (λ*m n*. *sum f* {*m*..<*n*}) *M N*)

definition *NSsums* :: (*nat* ⇒ *real*) ⇒ *real* ⇒ *bool* (**infixr** ‹*NSsums*› *80*)
where *f NSsums s* = (λ*n*. *sum f* {..<*n*}) $\longrightarrow_{NS}$ *s*

definition *NSsummable* :: (*nat* ⇒ *real*) ⇒ *bool*
where *NSsummable f* ⟷ (∃ *s*. *f NSsums s*)

definition *NSsuminf* :: (*nat* ⇒ *real*) ⇒ *real*
where *NSsuminf f* = (*THE s*. *f NSsums s*)

lemma *sumhr-app*: *sumhr* (*M*, *N*, *f*) = (∗*f2*∗ (λ*m n*. *sum f* {*m*..<*n*})) *M N*
by (*simp add*: *sumhr-def*)

Base case in definition of *sumr*.

lemma *sumhr-zero* [*simp*]: ⋀*m*. *sumhr* (*m*, *0*, *f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

Recursive case in definition of *sumr*.

lemma *sumhr-if*:
⋀*m n*. *sumhr* (*m*, *n* + *1*, *f*) = (*if n* + *1* ≤ *m then 0 else sumhr* (*m*, *n*, *f*) + (∗*f*∗ *f*) *n*)
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-Suc-zero* [*simp*]: ⋀*n*. *sumhr* (*n* + *1*, *n*, *f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-eq-bounds* [*simp*]: ⋀*n*. *sumhr* (*n*, *n*, *f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-Suc* [*simp*]: ⋀*m*. *sumhr* (*m*, *m* + *1*, *f*) = (∗*f*∗ *f*) *m*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-add-lbound-zero* [*simp*]: ⋀*k m*. *sumhr* (*m* + *k*, *k*, *f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-add*: ⋀*m n*. *sumhr* (*m*, *n*, *f*) + *sumhr* (*m*, *n*, *g*) = *sumhr* (*m*, *n*, λ*i*. *f i* + *g i*)
unfolding *sumhr-app* **by** *transfer* (*rule sum.distrib* [*symmetric*])

lemma *sumhr-mult*: $\bigwedge m\ n.\ hypreal\text{-}of\text{-}real\ r * sumhr\ (m,\ n,\ f) = sumhr\ (m,\ n,\ \lambda n.\ r * f\ n)$
unfolding *sumhr-app* **by** *transfer* (*rule sum-distrib-left*)

lemma *sumhr-split-add*: $\bigwedge n\ p.\ n < p \Longrightarrow sumhr\ (0,\ n,\ f) + sumhr\ (n,\ p,\ f) = sumhr\ (0,\ p,\ f)$
unfolding *sumhr-app* **by** *transfer* (*simp add*: *sum.atLeastLessThan-concat*)

lemma *sumhr-split-diff*: $n < p \Longrightarrow sumhr\ (0,\ p,\ f) - sumhr\ (0,\ n,\ f) = sumhr\ (n,\ p,\ f)$
by (*drule sumhr-split-add* [*symmetric*, **where** $f = f$]) *simp*

lemma *sumhr-hrabs*: $\bigwedge m\ n.\ |sumhr\ (m,\ n,\ f)| \le sumhr\ (m,\ n,\ \lambda i.\ |f\ i|)$
unfolding *sumhr-app* **by** *transfer* (*rule sum-abs*)

Other general version also needed.

lemma *sumhr-fun-hypnat-eq*:
$(\forall r.\ m \le r \wedge r < n \longrightarrow f\ r = g\ r) \longrightarrow$
$sumhr\ (hypnat\text{-}of\text{-}nat\ m,\ hypnat\text{-}of\text{-}nat\ n,\ f) =$
$sumhr\ (hypnat\text{-}of\text{-}nat\ m,\ hypnat\text{-}of\text{-}nat\ n,\ g)$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-const*: $\bigwedge n.\ sumhr\ (0,\ n,\ \lambda i.\ r) = hypreal\text{-}of\text{-}hypnat\ n * hypreal\text{-}of\text{-}real\ r$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-less-bounds-zero* [*simp*]: $\bigwedge m\ n.\ n < m \Longrightarrow sumhr\ (m,\ n,\ f) = 0$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-minus*: $\bigwedge m\ n.\ sumhr\ (m,\ n,\ \lambda i.\ -\ f\ i) = -\ sumhr\ (m,\ n,\ f)$
unfolding *sumhr-app* **by** *transfer* (*rule sum-negf*)

lemma *sumhr-shift-bounds*:
$\bigwedge m\ n.\ sumhr\ (m + hypnat\text{-}of\text{-}nat\ k,\ n + hypnat\text{-}of\text{-}nat\ k,\ f) =$
$sumhr\ (m,\ n,\ \lambda i.\ f\ (i + k))$
unfolding *sumhr-app* **by** *transfer* (*rule sum.shift-bounds-nat-ivl*)

11.1 Nonstandard Sums

Infinite sums are obtained by summing to some infinite hypernatural (such as *whn*).

lemma *sumhr-hypreal-of-hypnat-omega*: $sumhr\ (0,\ whn,\ \lambda i.\ 1) = hypreal\text{-}of\text{-}hypnat\ whn$
by (*simp add*: *sumhr-const*)

lemma *whn-eq-ωm1*: $hypreal\text{-}of\text{-}hypnat\ whn = \omega - 1$
unfolding *star-class-defs omega-def hypnat-omega-def of-hypnat-def star-of-def*
by (*simp add*: *starfun-star-n starfun2-star-n*)

lemma *sumhr-hypreal-omega-minus-one*: $sumhr(0,\ whn,\ \lambda i.\ 1) = \omega - 1$
by (*simp add*: *sumhr-const whn-eq-ωm1*)

lemma *sumhr-minus-one-realpow-zero* [*simp*]: $\bigwedge N.\ sumhr\ (0,\ N + N,\ \lambda i.\ (-1)\ \hat{}\ (i + 1)) = 0$
unfolding *sumhr-app*
by *transfer* (*induct-tac N*, *auto*)

lemma *sumhr-interval-const*:
$(\forall n.\ m \leq Suc\ n \longrightarrow f\ n = r) \wedge m \leq na \Longrightarrow$
$sumhr\ (hypnat\text{-}of\text{-}nat\ m,\ hypnat\text{-}of\text{-}nat\ na,\ f) = hypreal\text{-}of\text{-}nat\ (na - m) * hypreal\text{-}of\text{-}real\ r$
unfolding *sumhr-app* **by** *transfer simp*

lemma *starfunNat-sumr*: $\bigwedge N.\ (\ *f*\ (\lambda n.\ sum\ f\ \{0..<n\}))\ N = sumhr\ (0,\ N,\ f)$
unfolding *sumhr-app* **by** *transfer* (*rule refl*)

lemma *sumhr-hrabs-approx* [*simp*]: $sumhr\ (0,\ M, f) \approx sumhr\ (0,\ N, f) \Longrightarrow |sumhr\ (M,\ N,\ f)| \approx 0$
using *linorder-less-linear* [**where** $x = M$ **and** $y = N$]
by (*metis* (*no-types*, *lifting*) *abs-zero approx-hrabs approx-minus-iff approx-refl approx-sym sumhr-eq-bounds sumhr-less-bounds-zero sumhr-split-diff*)

11.2 Infinite sums: Standard and NS theorems

lemma *sums-NSsums-iff*: $f\ sums\ l \longleftrightarrow f\ NSsums\ l$
by (*simp add*: *sums-def NSsums-def LIMSEQ-NSLIMSEQ-iff*)

lemma *summable-NSsummable-iff*: $summable\ f \longleftrightarrow NSsummable\ f$
by (*simp add*: *summable-def NSsummable-def sums-NSsums-iff*)

lemma *suminf-NSsuminf-iff*: $suminf\ f = NSsuminf\ f$
by (*simp add*: *suminf-def NSsuminf-def sums-NSsums-iff*)

lemma *NSsums-NSsummable*: $f\ NSsums\ l \Longrightarrow NSsummable\ f$
unfolding *NSsums-def NSsummable-def* **by** *blast*

lemma *NSsummable-NSsums*: $NSsummable\ f \Longrightarrow f\ NSsums\ (NSsuminf\ f)$
unfolding *NSsummable-def NSsuminf-def NSsums-def*
by (*blast intro*: *theI NSLIMSEQ-unique*)

lemma *NSsums-unique*: $f\ NSsums\ s \Longrightarrow s = NSsuminf\ f$
by (*simp add*: *suminf-NSsuminf-iff* [*symmetric*] *sums-NSsums-iff sums-unique*)

lemma *NSseries-zero*: $\forall m.\ n \leq Suc\ m \longrightarrow f\ m = 0 \Longrightarrow f\ NSsums\ (sum\ f\ \{..<n\})$
by (*auto simp add*: *sums-NSsums-iff* [*symmetric*] *not-le*[*symmetric*] *intro*!: *sums-finite*)

lemma *NSsummable-NSCauchy*:

NSsummable f $\longleftrightarrow$ ($\forall$ *M* $\in$ *HNatInfinite*. $\forall$ *N* $\in$ *HNatInfinite*. |*sumhr* (*M*, *N*, *f*)| $\approx$ *0*) (**is** *?L=?R*)
proof −
have *?L* = ($\forall$ *M*$\in$*HNatInfinite*. $\forall$ *N*$\in$*HNatInfinite*. *sumhr* (*0*, *M*, *f*) $\approx$ *sumhr* (*0*, *N*, *f*))
by (*auto simp add*: *summable-iff-convergent convergent-NSconvergent-iff NSCauchy-def starfunNat-sumr*
simp flip: *NSCauchy-NSconvergent-iff summable-NSsummable-iff atLeast0LessThan*)
also have ... $\longleftrightarrow$ *?R*
by (*metis approx-hrabs-zero-cancel approx-minus-iff approx-refl approx-sym linorder-less-linear sumhr-hrabs-approx sumhr-split-diff*)
finally show *?thesis* **.**
qed

Terms of a convergent series tend to zero.

lemma *NSsummable-NSLIMSEQ-zero*: *NSsummable f* $\Longrightarrow$ *f* $\longrightarrow_{NS}$ *0*
by (*metis HNatInfinite-add NSLIMSEQ-def NSsummable-NSCauchy approx-hrabs-zero-cancel star-of-zero sumhr-Suc*)

Nonstandard comparison test.

lemma *NSsummable-comparison-test*: $\exists$ *N*. $\forall$ *n*. *N* $\le$ *n* $\longrightarrow$ |*f n*| $\le$ *g n* $\Longrightarrow$ *NSsummable g* $\Longrightarrow$ *NSsummable f*
by (*metis real-norm-def summable-NSsummable-iff summable-comparison-test*)

lemma *NSsummable-rabs-comparison-test*:
$\exists$ *N*. $\forall$ *n*. *N* $\le$ *n* $\longrightarrow$ |*f n*| $\le$ *g n* $\Longrightarrow$ *NSsummable g* $\Longrightarrow$ *NSsummable* (λ*k*. |*f k*|)
by (*rule NSsummable-comparison-test*) *auto*

end

12 Limits and Continuity (Nonstandard)

theory *HLim*
imports *Star*
abbrevs −−−> = −□→$_{NS}$
begin

Nonstandard Definitions.

definition *NSLIM* :: ('*a*::*real-normed-vector* $\Rightarrow$ '*b*::*real-normed-vector*) $\Rightarrow$ '*a* $\Rightarrow$ '*b* $\Rightarrow$ *bool*
(‹(‹*notation*=‹*mixfix NSLIM*››(-)/ −(-)/→$_{NS}$ (-))› [*60*, *0*, *60*] *60*)
where *f* −*a*→$_{NS}$ *L* $\longleftrightarrow$ ($\forall$ *x*. *x* $\neq$ *star-of a* $\wedge$ *x* $\approx$ *star-of a* $\longrightarrow$ (**f* f*) *x* $\approx$ *star-of L*)

definition *isNSCont* :: ('*a*::*real-normed-vector* $\Rightarrow$ '*b*::*real-normed-vector*) $\Rightarrow$ '*a* $\Rightarrow$ *bool*
where — NS definition dispenses with limit notions
isNSCont f a $\longleftrightarrow$ ($\forall$ *y*. *y* $\approx$ *star-of a* $\longrightarrow$ (**f* f*) *y* $\approx$ *star-of* (*f a*))

definition *isNSUCont* :: $('a::real\text{-}normed\text{-}vector \Rightarrow 'b::real\text{-}normed\text{-}vector) \Rightarrow bool$
where $isNSUCont\ f \longleftrightarrow (\forall x\ y.\ x \approx y \longrightarrow (\ {*}f{*}\ f)\ x \approx (\ {*}f{*}\ f)\ y)$

12.1 Limits of Functions

lemma *NSLIM-I*: $(\bigwedge x.\ x \neq star\text{-}of\ a \Longrightarrow x \approx star\text{-}of\ a \Longrightarrow starfun\ f\ x \approx star\text{-}of\ L) \Longrightarrow f\ {-a}{\rightarrow_{NS}}\ L$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-D*: $f\ {-a}{\rightarrow_{NS}}\ L \Longrightarrow x \neq star\text{-}of\ a \Longrightarrow x \approx star\text{-}of\ a \Longrightarrow starfun\ f\ x \approx star\text{-}of\ L$
by (*simp add*: *NSLIM-def*)

Proving properties of limits using nonstandard definition. The properties hold for standard limits as well!

lemma *NSLIM-mult*: $f\ {-x}{\rightarrow_{NS}}\ l \Longrightarrow g\ {-x}{\rightarrow_{NS}}\ m \Longrightarrow (\lambda x.\ f\ x * g\ x)\ {-x}{\rightarrow_{NS}}\ (l * m)$
for $l\ m$ :: *'a::real-normed-algebra*
by (*auto simp add*: *NSLIM-def intro*!: *approx-mult-HFinite*)

lemma *starfun-scaleR* [*simp*]: $starfun\ (\lambda x.\ f\ x *_R g\ x) = (\lambda x.\ scaleHR\ (starfun\ f\ x)\ (starfun\ g\ x))$
by *transfer* (*rule refl*)

lemma *NSLIM-scaleR*: $f\ {-x}{\rightarrow_{NS}}\ l \Longrightarrow g\ {-x}{\rightarrow_{NS}}\ m \Longrightarrow (\lambda x.\ f\ x *_R g\ x)\ {-x}{\rightarrow_{NS}}\ (l *_R m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-scaleR-HFinite*)

lemma *NSLIM-add*: $f\ {-x}{\rightarrow_{NS}}\ l \Longrightarrow g\ {-x}{\rightarrow_{NS}}\ m \Longrightarrow (\lambda x.\ f\ x + g\ x)\ {-x}{\rightarrow_{NS}}\ (l + m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-add*)

lemma *NSLIM-const* [*simp*]: $(\lambda x.\ k)\ {-x}{\rightarrow_{NS}}\ k$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-minus*: $f\ {-a}{\rightarrow_{NS}}\ L \Longrightarrow (\lambda x.\ -\ f\ x)\ {-a}{\rightarrow_{NS}}\ -L$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-diff*: $f\ {-x}{\rightarrow_{NS}}\ l \Longrightarrow g\ {-x}{\rightarrow_{NS}}\ m \Longrightarrow (\lambda x.\ f\ x - g\ x)\ {-x}{\rightarrow_{NS}}\ (l - m)$
by (*simp only*: *NSLIM-add NSLIM-minus diff-conv-add-uminus*)

lemma *NSLIM-add-minus*: $f\ {-x}{\rightarrow_{NS}}\ l \Longrightarrow g\ {-x}{\rightarrow_{NS}}\ m \Longrightarrow (\lambda x.\ f\ x + -\ g\ x)\ {-x}{\rightarrow_{NS}}\ (l + -m)$
by (*simp only*: *NSLIM-add NSLIM-minus*)

lemma *NSLIM-inverse*: $f\ {-a}{\rightarrow_{NS}}\ L \Longrightarrow L \neq 0 \Longrightarrow (\lambda x.\ inverse\ (f\ x))\ {-a}{\rightarrow_{NS}}\ (inverse\ L)$

for *L* :: *'a::real-normed-div-algebra*
unfolding *NSLIM-def* **by** (*metis* (*no-types*) *star-of-approx-inverse star-of-simps*(*6*) *starfun-inverse*)

lemma *NSLIM-zero*:
assumes *f*: $f\ -a\rightarrow_{NS}\ l$
shows $(\lambda x.\ f(x) - l)\ -a\rightarrow_{NS}\ 0$
proof −
have $(\lambda x.\ f\ x - l)\ -a\rightarrow_{NS}\ l - l$
by (*rule NSLIM-diff* [*OF f NSLIM-const*])
then show *?thesis* **by** *simp*
qed

lemma *NSLIM-zero-cancel*:
assumes $(\lambda x.\ f\ x - l)\ -x\rightarrow_{NS}\ 0$
shows $f\ -x\rightarrow_{NS}\ l$
proof −
have $(\lambda x.\ f\ x - l + l)\ -x\rightarrow_{NS}\ 0 + l$
by (*fast intro*: *assms NSLIM-const NSLIM-add*)
then show *?thesis*
by *simp*
qed

lemma *NSLIM-const-eq*:
fixes *a* :: *'a::real-normed-algebra-1*
assumes $(\lambda x.\ k)\ -a\rightarrow_{NS}\ l$
shows $k = l$
proof −
have $\neg\ (\lambda x.\ k)\ -a\rightarrow_{NS}\ l$ **if** $k \neq l$
proof −
have *star-of a* + *of-hypreal* ε ≈ *star-of a*
by (*simp add*: *approx-def*)
then show *?thesis*
using *epsilon-not-zero that* **by** (*force simp add*: *NSLIM-def*)
qed
with *assms* **show** *?thesis* **by** *metis*
qed

lemma *NSLIM-unique*: $f\ -a\rightarrow_{NS}\ l \Longrightarrow f\ -a\rightarrow_{NS}\ M \Longrightarrow l = M$
for *a* :: *'a::real-normed-algebra-1*
by (*drule* (*1*) *NSLIM-diff*) (*auto dest*!: *NSLIM-const-eq*)

lemma *NSLIM-mult-zero*: $f\ -x\rightarrow_{NS}\ 0 \Longrightarrow g\ -x\rightarrow_{NS}\ 0 \Longrightarrow (\lambda x.\ f\ x * g\ x)\ -x\rightarrow_{NS}\ 0$
for *f g* :: *'a::real-normed-vector* ⇒ *'b::real-normed-algebra*
by (*drule NSLIM-mult*) *auto*

lemma *NSLIM-self*: $(\lambda x.\ x)\ -a\rightarrow_{NS}\ a$
by (*simp add*: *NSLIM-def*)

12.1.1 Equivalence of *filterlim* and *NSLIM*

lemma *LIM-NSLIM*:
  assumes f: $f\ -a\rightarrow\ L$
  shows $f\ -a\rightarrow_{NS}\ L$
proof (*rule NSLIM-I*)
  fix x
  assume *neq*: $x \neq$ *star-of a*
  assume *approx*: $x \approx$ *star-of a*
  have *starfun f x* $-$ *star-of L* $\in$ *Infinitesimal*
  proof (*rule InfinitesimalI2*)
    fix r :: *real*
    assume r: $0 < r$
    from *LIM-D* [*OF f r*] **obtain** s
      where s: $0 < s$ **and** *less-r*: $\bigwedge x.\ x \neq a \Longrightarrow$ *norm* $(x - a) < s \Longrightarrow$ *norm* $(f\ x$
$-\ L) < r$
      by *fast*
    from *less-r* **have** *less-r'*:
      $\bigwedge x.\ x \neq$ *star-of a* $\Longrightarrow$ *hnorm* $(x -$ *star-of a*$)$ $<$ *star-of s* $\Longrightarrow$
        hnorm (*starfun f x* $-$ *star-of L*) $<$ *star-of r*
      by *transfer*
    from *approx* **have** x $-$ *star-of a* $\in$ *Infinitesimal*
      by (*simp only*: *approx-def*)
    then have *hnorm* $(x -$ *star-of a*$)$ $<$ *star-of s*
      using s **by** (*rule InfinitesimalD2*)
    with *neq* **show** *hnorm* (*starfun f x* $-$ *star-of L*) $<$ *star-of r*
      by (*rule less-r'*)
  qed
  then show *starfun f x* $\approx$ *star-of L*
    by (*unfold approx-def*)
qed

lemma *NSLIM-LIM*:
  assumes f: $f\ -a\rightarrow_{NS}\ L$
  shows $f\ -a\rightarrow\ L$
proof (*rule LIM-I*)
  fix r :: *real*
  assume r: $0 < r$
  have $\exists\, s>0.\ \forall\, x.\ x \neq$ *star-of a* $\wedge$ *hnorm* $(x -$ *star-of a*$)$ $< s \longrightarrow$
    hnorm (*starfun f x* $-$ *star-of L*) $<$ *star-of r*
  proof (*rule exI*, *safe*)
    show $0 < \varepsilon$
      by (*rule epsilon-gt-zero*)
  next
    fix x
    assume *neq*: $x \neq$ *star-of a*
    assume *hnorm* $(x -$ *star-of a*$)$ $< \varepsilon$
    with *Infinitesimal-epsilon* **have** x $-$ *star-of a* $\in$ *Infinitesimal*
      by (*rule hnorm-less-Infinitesimal*)
    then have $x \approx$ *star-of a*

by (*unfold approx-def*)
with *f neq* **have** *starfun f x* $\approx$ *star-of L*
by (*rule NSLIM-D*)
then have *starfun f x* $-$ *star-of L* $\in$ *Infinitesimal*
by (*unfold approx-def*)
then show *hnorm* (*starfun f x* $-$ *star-of L*) $<$ *star-of r*
using *r* **by** (*rule InfinitesimalD2*)
qed
then show $\exists s>0.\ \forall x.\ x \neq a \wedge norm\ (x - a) < s \longrightarrow norm\ (f\ x - L) < r$
by *transfer*
qed

theorem *LIM-NSLIM-iff*: $f\ {-x\rightarrow}\ L \longleftrightarrow f\ {-x\rightarrow}_{NS}\ L$
by (*blast intro*: *LIM-NSLIM NSLIM-LIM*)

12.2 Continuity

lemma *isNSContD*: *isNSCont f a* $\Longrightarrow y \approx$ *star-of a* $\Longrightarrow$ ($*f*$ *f*) $y \approx$ *star-of* (*f a*)
by (*simp add*: *isNSCont-def*)

lemma *isNSCont-NSLIM*: *isNSCont f a* $\Longrightarrow f\ {-a\rightarrow}_{NS}\ (f\ a)$
by (*simp add*: *isNSCont-def NSLIM-def*)

lemma *NSLIM-isNSCont*: $f\ {-a\rightarrow}_{NS}\ (f\ a) \Longrightarrow$ *isNSCont f a*
by (*force simp add*: *isNSCont-def NSLIM-def*)

NS continuity can be defined using NS Limit in similar fashion to standard definition of continuity.

lemma *isNSCont-NSLIM-iff*: *isNSCont f a* $\longleftrightarrow f\ {-a\rightarrow}_{NS}\ (f\ a)$
by (*blast intro*: *isNSCont-NSLIM NSLIM-isNSCont*)

Hence, NS continuity can be given in terms of standard limit.

lemma *isNSCont-LIM-iff*: (*isNSCont f a*) $=$ ($f\ {-a\rightarrow}\ (f\ a)$)
by (*simp add*: *LIM-NSLIM-iff isNSCont-NSLIM-iff*)

Moreover, it's trivial now that NS continuity is equivalent to standard continuity.

lemma *isNSCont-isCont-iff*: *isNSCont f a* $\longleftrightarrow$ *isCont f a*
by (*simp add*: *isCont-def*) (*rule isNSCont-LIM-iff*)

Standard continuity $\Longrightarrow$ NS continuity.

lemma *isCont-isNSCont*: *isCont f a* $\Longrightarrow$ *isNSCont f a*
by (*erule isNSCont-isCont-iff* [*THEN iffD2*])

NS continuity $\Longrightarrow$ Standard continuity.

lemma *isNSCont-isCont*: *isNSCont f a* $\Longrightarrow$ *isCont f a*
by (*erule isNSCont-isCont-iff* [*THEN iffD1*])

Alternative definition of continuity.

Prove equivalence between NS limits – seems easier than using standard definition.

lemma *NSLIM-at0-iff*: $f\ {-a\rightarrow_{NS}}\ L \longleftrightarrow (\lambda h.\ f\ (a + h))\ {-0\rightarrow_{NS}}\ L$
proof
  assume $f\ {-a\rightarrow_{NS}}\ L$
  then show $(\lambda h.\ f\ (a + h))\ {-0\rightarrow_{NS}}\ L$
   by (*simp add*: *NSLIM-def*) (*metis* (*no-types*) *add-cancel-left-right approx-add-left-iff starfun-lambda-cancel*)
next
  assume ∗: $(\lambda h.\ f\ (a + h))\ {-0\rightarrow_{NS}}\ L$
  show $f\ {-a\rightarrow_{NS}}\ L$
  proof (*clarsimp simp*: *NSLIM-def*)
    fix *x*
    assume $x \neq$ *star-of a* $x \approx$ *star-of a*
    then have $(*f*\ (\lambda h.\ f\ (a + h)))\ (-$ *star-of a* $+ x) \approx$ *star-of L*
      by (*metis* (*no-types*, *lifting*) ∗ *NSLIM-D add.right-neutral add-minus-cancel approx-minus-iff2 star-zero-def*)
    then show $(*f*\ f)\ x \approx$ *star-of L*
      by (*simp add*: *starfun-lambda-cancel*)
  qed
qed

lemma *isNSCont-minus*: *isNSCont f a* $\Longrightarrow$ *isNSCont* $(\lambda x.\ -\ f\ x)\ a$
  by (*simp add*: *isNSCont-def*)

lemma *isNSCont-inverse*: *isNSCont f x* $\Longrightarrow f\ x \neq 0 \Longrightarrow$ *isNSCont* $(\lambda x.$ *inverse* $(f\ x))\ x$
  for *f* :: *'a::real-normed-vector* ⇒ *'b::real-normed-div-algebra*
  using *NSLIM-inverse NSLIM-isNSCont isNSCont-NSLIM* **by** *blast*

lemma *isNSCont-const* [*simp*]: *isNSCont* $(\lambda x.\ k)\ a$
  by (*simp add*: *isNSCont-def*)

lemma *isNSCont-abs* [*simp*]: *isNSCont abs a*
  for *a* :: *real*
  by (*auto simp*: *isNSCont-def intro*: *approx-hrabs simp*: *starfun-rabs-hrabs*)

12.3 Uniform Continuity

lemma *isNSUContD*: *isNSUCont f* $\Longrightarrow x \approx y \Longrightarrow (\ *f*\ f)\ x \approx (\ *f*\ f)\ y$
  by (*simp add*: *isNSUCont-def*)

lemma *isUCont-isNSUCont*:
  fixes *f* :: *'a::real-normed-vector* ⇒ *'b::real-normed-vector*
  assumes *f*: *isUCont f*
  shows *isNSUCont f*
  unfolding *isNSUCont-def*

proof *safe*
  fix *x y* :: *'a star*
  assume *approx*: $x \approx y$
  have *starfun f x* − *starfun f y* ∈ *Infinitesimal*
  proof (*rule InfinitesimalI2*)
    fix *r* :: *real*
    assume *r*: $0 < r$
    with *f* **obtain** *s* **where** *s*: $0 < s$
      and *less-r*: ⋀*x y*. *norm* (*x* − *y*) < *s* ⟹ *norm* (*f x* − *f y*) < *r*
      by (*auto simp add*: *isUCont-def dist-norm*)
    from *less-r* **have** *less-r'*:
       ⋀*x y*. *hnorm* (*x* − *y*) < *star-of s* ⟹ *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
      by *transfer*
    from *approx* **have** *x* − *y* ∈ *Infinitesimal*
      by (*unfold approx-def*)
    then have *hnorm* (*x* − *y*) < *star-of s*
      using *s* **by** (*rule InfinitesimalD2*)
    then show *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
      by (*rule less-r'*)
  qed
  then show *starfun f x* ≈ *starfun f y*
    by (*unfold approx-def*)
qed

lemma *isNSUCont-isUCont*:
  fixes *f* :: *'a::real-normed-vector* ⇒ *'b::real-normed-vector*
  assumes *f*: *isNSUCont f*
  shows *isUCont f*
  unfolding *isUCont-def dist-norm*
proof *safe*
  fix *r* :: *real*
  assume *r*: $0 < r$
  have ∃ *s*>*0*. ∀ *x y*. *hnorm* (*x* − *y*) < *s* ⟶ *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
  proof (*rule exI*, *safe*)
    show $0 < \varepsilon$
      by (*rule epsilon-gt-zero*)
  next
    fix *x y* :: *'a star*
    assume *hnorm* (*x* − *y*) < ε
    with *Infinitesimal-epsilon* **have** *x* − *y* ∈ *Infinitesimal*
      by (*rule hnorm-less-Infinitesimal*)
    then have $x \approx y$
      by (*unfold approx-def*)
    with *f* **have** *starfun f x* ≈ *starfun f y*
      by (*simp add*: *isNSUCont-def*)
    then have *starfun f x* − *starfun f y* ∈ *Infinitesimal*
      by (*unfold approx-def*)

then show *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
using *r* **by** (*rule InfinitesimalD2*)
qed
then show ∃ *s*>*0*. ∀ *x y*. *norm* (*x* − *y*) < *s* ⟶ *norm* (*f x* − *f y*) < *r*
by *transfer*
qed

end

13 Differentiation (Nonstandard)

theory *HDeriv*
imports *HLim*
begin

Nonstandard Definitions.

definition *nsderiv* :: ['*a*::*real-normed-field* ⇒ '*a*, '*a*, '*a*] ⇒ *bool*
(‹(‹*notation*=‹*mixfix NSDERIV*››*NSDERIV* (-)/ (-)/ :> (-))› [*1000*, *1000*, *60*] *60*)
where *NSDERIV f x* :> *D* ⟷
(∀ *h* ∈ *Infinitesimal* − {*0*}. ((∗*f*∗ *f*)(*star-of x* + *h*) − *star-of* (*f x*)) / *h* ≈ *star-of D*)

definition *NSdifferentiable* :: ['*a*::*real-normed-field* ⇒ '*a*, '*a*] ⇒ *bool*
(**infixl** ‹*NSdifferentiable*› *60*)
where *f NSdifferentiable x* ⟷ (∃ *D*. *NSDERIV f x* :> *D*)

definition *increment* :: (*real* ⇒ *real*) ⇒ *real* ⇒ *hypreal* ⇒ *hypreal*
where *increment f x h* =
(*SOME inc*. *f NSdifferentiable x* ∧ *inc* = (∗*f*∗ *f*) (*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*))

13.1 Derivatives

lemma *DERIV-NS-iff*: (*DERIV f x* :> *D*) ⟷ (λ*h*. (*f* (*x* + *h*) − *f x*) / *h*) −*0*→$_{NS}$ *D*
by (*simp add*: *DERIV-def LIM-NSLIM-iff*)

lemma *NS-DERIV-D*: *DERIV f x* :> *D* ⟹ (λ*h*. (*f* (*x* + *h*) − *f x*) / *h*) −*0*→$_{NS}$ *D*
by (*simp add*: *DERIV-def LIM-NSLIM-iff*)

lemma *Infinitesimal-of-hypreal*:
x ∈ *Infinitesimal* ⟹ ((∗*f*∗ *of-real*) *x*::'*a*::*real-normed-div-algebra star*) ∈ *Infinitesimal*
by (*metis Infinitesimal-of-hypreal-iff of-hypreal-def*)

lemma *of-hypreal-eq-0-iff*: ⋀*x*. ((∗*f*∗ *of-real*) *x* = (*0*::'*a*::*real-algebra-1 star*)) = (*x* = *0*)

by *transfer* (*rule of-real-eq-0-iff*)

lemma *NSDeriv-unique*:
assumes *NSDERIV f x :> D NSDERIV f x :> E*
shows *NSDERIV f x :> D* $\Longrightarrow$ *NSDERIV f x :> E* $\Longrightarrow$ *D* = *E*
proof −
have $\exists$ *s*. (*s*::′*a star*) $\in$ *Infinitesimal* − {*0*}
by (*metis Diff-iff HDeriv.of-hypreal-eq-0-iff Infinitesimal-epsilon Infinitesimal-of-hypreal epsilon-not-zero singletonD*)
with *assms* **show** *?thesis*
by (*meson approx-trans3 nsderiv-def star-of-approx-iff*)
qed

First *NSDERIV* in terms of *NSLIM*.

First equivalence.

lemma *NSDERIV-NSLIM-iff*: (*NSDERIV f x :> D*) $\longleftrightarrow$ (λh. (*f* (*x* + *h*) − *f x*) / *h*) −*0*$\rightarrow_{NS}$ *D*
by (*auto simp add*: *nsderiv-def NSLIM-def starfun-lambda-cancel mem-infmal-iff*)

Second equivalence.

lemma *NSDERIV-NSLIM-iff2*: (*NSDERIV f x :> D*) $\longleftrightarrow$ (λz. (*f z* − *f x*) / (*z* − *x*)) −*x*$\rightarrow_{NS}$ *D*
by (*simp add*: *NSDERIV-NSLIM-iff DERIV-LIM-iff LIM-NSLIM-iff* [*symmetric*])

While we're at it!

lemma *NSDERIV-iff2*:
(*NSDERIV f x :> D*) $\longleftrightarrow$
($\forall$ *w*. *w* $\neq$ *star-of x* $\wedge$ *w* $\approx$ *star-of x* $\longrightarrow$ (**f** (λz. (*f z* − *f x*) / (*z* − *x*))) *w* $\approx$ *star-of D*)
by (*simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def*)

lemma *NSDERIVD5*:
⟦*NSDERIV f x :> D*; *u* $\approx$ *hypreal-of-real x*⟧ $\Longrightarrow$
(**f** (λz. *f z* − *f x*)) *u* $\approx$ *hypreal-of-real D* ∗ (*u* − *hypreal-of-real x*)
unfolding *NSDERIV-iff2*
apply (*case-tac u* = *hypreal-of-real x*, *auto*)
by (*metis* (*mono-tags*, *lifting*) *HFinite-star-of Infinitesimal-ratio approx-def approx-minus-iff approx-mult-subst approx-star-of-HFinite approx-sym mult-zero-right right-minus-eq*)

lemma *NSDERIVD4*:
⟦*NSDERIV f x :> D*; *h* $\in$ *Infinitesimal*⟧
$\Longrightarrow$ (**f* f*)(*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*) $\approx$ *hypreal-of-real D* ∗ *h*
apply (*clarsimp simp add*: *nsderiv-def*)
apply (*case-tac h* = *0*, *simp*)
by (*meson DiffI Infinitesimal-approx Infinitesimal-ratio Infinitesimal-star-of-mult2 approx-star-of-HFinite singletonD*)

Differentiability implies continuity nice and simple "algebraic" proof.

lemma *NSDERIV-isNSCont*:
assumes *NSDERIV f x :> D* **shows** *isNSCont f x*
unfolding *isNSCont-NSLIM-iff NSLIM-def*
proof *clarify*
fix x'
assume $x' \neq$ *star-of x* $x' \approx$ *star-of x*
then have *m0*: $x' -$ *star-of x* $\in$ *Infinitesimal* $- \{0\}$
using *bex-Infinitesimal-iff* **by** *auto*
then have $((*f* f)\ x' -$ *star-of* $(f\ x))\ /\ (x' -$ *star-of* $x) \approx$ *star-of D*
by (*metis* ‹$x' \approx$ *star-of x*› *add-diff-cancel-left′ assms bex-Infinitesimal-iff2 ns-deriv-def*)
then have $((*f* f)\ x' -$ *star-of* $(f\ x))\ /\ (x' -$ *star-of* $x) \in$ *HFinite*
by (*metis approx-star-of-HFinite*)
then show $(*f* f)\ x' \approx$ *star-of* $(f\ x)$
by (*metis* (*no-types*) *Diff-iff Infinitesimal-ratio m0 bex-Infinitesimal-iff insert-iff*)
qed

Differentiation rules for combinations of functions follow from clear, straightforward, algebraic manipulations.

Constant function.

lemma *NSDERIV-const* [*simp*]: *NSDERIV* $(\lambda x.\ k)$ x :> *0*
by (*simp add*: *NSDERIV-NSLIM-iff*)

Sum of functions- proved easily.

lemma *NSDERIV-add*:
assumes *NSDERIV f x :> Da NSDERIV g x :> Db*
shows *NSDERIV* $(\lambda x.\ f\ x + g\ x)\ x$:> $Da + Db$
proof −
have $((\lambda x.\ f\ x + g\ x)$ *has-field-derivative* $Da + Db)$ (*at x*)
using *assms DERIV-NS-iff NSDERIV-NSLIM-iff field-differentiable-add* **by** *blast*
then show *?thesis*
by (*simp add*: *DERIV-NS-iff NSDERIV-NSLIM-iff*)
qed

Product of functions - Proof is simple.

lemma *NSDERIV-mult*:
assumes *NSDERIV g x :> Db NSDERIV f x :> Da*
shows *NSDERIV* $(\lambda x.\ f\ x * g\ x)\ x$:> $(Da * g\ x) + (Db * f\ x)$
proof −
have (*f has-field-derivative Da*) (*at x*) (*g has-field-derivative Db*) (*at x*)
using *assms* **by** (*simp-all add*: *DERIV-NS-iff NSDERIV-NSLIM-iff*)
then have $((\lambda a.\ f\ a * g\ a)$ *has-field-derivative* $Da * g\ x + Db * f\ x)$ (*at x*)
using *DERIV-mult* **by** *blast*
then show *?thesis*

by (*simp add*: *DERIV-NS-iff NSDERIV-NSLIM-iff*)
qed

Multiplying by a constant.

lemma *NSDERIV-cmult*: *NSDERIV f x* :> *D* $\Longrightarrow$ *NSDERIV* ($\lambda x.\ c * f\ x$) *x* :> $c * D$
unfolding *times-divide-eq-right* [*symmetric*] *NSDERIV-NSLIM-iff*
minus-mult-right right-diff-distrib [*symmetric*]
by (*erule NSLIM-const* [*THEN NSLIM-mult*])

Negation of function.

lemma *NSDERIV-minus*: *NSDERIV f x* :> *D* $\Longrightarrow$ *NSDERIV* ($\lambda x.\ -\ f\ x$) *x* :> $-$ *D*
proof (*simp add*: *NSDERIV-NSLIM-iff*)
assume ($\lambda h.\ (f\ (x + h) - f\ x)\ /\ h$) $-0\rightarrow_{NS}$ *D*
then have *deriv*: ($\lambda h.\ -\ ((f(x+h) - f\ x)\ /\ h)$) $-0\rightarrow_{NS}$ $-$ *D*
by (*rule NSLIM-minus*)
have $\forall h.\ -\ ((f\ (x + h) - f\ x)\ /\ h) = (-\ f\ (x + h) + f\ x)\ /\ h$
by (*simp add*: *minus-divide-left*)
with *deriv* **have** ($\lambda h.\ (-\ f\ (x + h) + f\ x)\ /\ h$) $-0\rightarrow_{NS}$ $-$ *D*
by *simp*
then show ($\lambda h.\ (f\ (x + h) - f\ x)\ /\ h$) $-0\rightarrow_{NS}$ *D* $\Longrightarrow$ ($\lambda h.\ (f\ x - f\ (x + h))\ /\ h$) $-0\rightarrow_{NS}$ $-$ *D*
by *simp*
qed

Subtraction.

lemma *NSDERIV-add-minus*:
NSDERIV f x :> *Da* $\Longrightarrow$ *NSDERIV g x* :> *Db* $\Longrightarrow$ *NSDERIV* ($\lambda x.\ f\ x + -\ g\ x$) *x* :> *Da* + $-$ *Db*
by (*blast dest*: *NSDERIV-add NSDERIV-minus*)

lemma *NSDERIV-diff*:
NSDERIV f x :> *Da* $\Longrightarrow$ *NSDERIV g x* :> *Db* $\Longrightarrow$ *NSDERIV* ($\lambda x.\ f\ x - g\ x$) *x* :> *Da* $-$ *Db*
using *NSDERIV-add-minus* [*of f x Da g Db*] **by** *simp*

Similarly to the above, the chain rule admits an entirely straightforward derivation. Compare this with Harrison's HOL proof of the chain rule, which proved to be trickier and required an alternative characterisation of differentiability- the so-called Carathedory derivative. Our main problem is manipulation of terms.

13.2 Lemmas

lemma *NSDERIV-zero*:
$[\![$*NSDERIV g x* :> *D*; (**f** *g*) (*star-of x* + *y*) = *star-of* (*g x*); $y \in$ *Infinitesimal*; $y \neq 0]\!]$

$\Longrightarrow D = 0$
by (*force simp add*: *nsderiv-def*)

Can be proved differently using *NSLIM-isCont-iff*.

lemma *NSDERIV-approx*:
NSDERIV f x :> $D \Longrightarrow h \in$ *Infinitesimal* $\Longrightarrow h \neq 0 \Longrightarrow$
(*∗f∗ f*) (*star-of x* + *h*) − *star-of* (*f x*) ≈ *0*
by (*meson DiffI Infinitesimal-ratio approx-star-of-HFinite mem-infmal-iff nsderiv-def singletonD*)

From one version of differentiability

f x − *f a* −−−−−−−−−−−−−−− ≈ *Db x* − *a*

lemma *NSDERIVD1*:
⟦*NSDERIV f* (*g x*) :> *Da*;
(*∗f∗ g*) (*star-of x* + *y*) ≠ *star-of* (*g x*);
(*∗f∗ g*) (*star-of x* + *y*) ≈ *star-of* (*g x*)⟧
⟹ ((*∗f∗ f*) ((*∗f∗ g*) (*star-of x* + *y*)) −
star-of (*f* (*g x*))) / ((*∗f∗ g*) (*star-of x* + *y*) − *star-of* (*g x*)) ≈
star-of Da
by (*auto simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def*)

From other version of differentiability

f (*x* + *h*) − *f x* −−−−−−−−−−−−−−−−−−− ≈ *Db h*

lemma *NSDERIVD2*: [| *NSDERIV g x* :> *Db*; *y* ∈ *Infinitesimal*; *y* ≠ *0* |]
==> ((*∗f∗ g*) (*star-of*(*x*) + *y*) − *star-of*(*g x*)) / *y*
≈ *star-of*(*Db*)
by (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def mem-infmal-iff starfun-lambda-cancel*)

This proof uses both definitions of differentiability.

lemma *NSDERIV-chain*:
NSDERIV f (*g x*) :> *Da* ⟹ *NSDERIV g x* :> *Db* ⟹ *NSDERIV* (*f* ∘ *g*) *x* :>
Da ∗ *Db*
using *DERIV-NS-iff DERIV-chain NSDERIV-NSLIM-iff* **by** *blast*

Differentiation of natural number powers.

lemma *NSDERIV-Id* [*simp*]: *NSDERIV* (λ*x*. *x*) *x* :> *1*
by (*simp add*: *NSDERIV-NSLIM-iff NSLIM-def del*: *divide-self-if*)

lemma *NSDERIV-cmult-Id* [*simp*]: *NSDERIV* ((∗) *c*) *x* :> *c*
using *NSDERIV-Id* [*THEN NSDERIV-cmult*] **by** *simp*

lemma *NSDERIV-inverse*:
fixes *x* :: *'a*::*real-normed-field*
assumes *x* ≠ *0* — can't get rid of *x* ≠ *0* because it isn't continuous at zero
shows *NSDERIV* (λ*x*. *inverse x*) *x* :> − (*inverse x* ^ *Suc* (*Suc 0*))
proof −
{

fix *h* :: *'a star*
assume *h-Inf*: *h* ∈ *Infinitesimal*
from *this assms* **have** *not-0*: *star-of x* + *h* ≠ *0*
by (*rule Infinitesimal-add-not-zero*)
assume *h* ≠ *0*
from *h-Inf* **have** *h* ∗ *star-of x* ∈ *Infinitesimal*
by (*rule Infinitesimal-HFinite-mult*) *simp*
with *assms* **have** *inverse* (− (*h* ∗ *star-of x*) + − (*star-of x* ∗ *star-of x*)) ≈
inverse (− (*star-of x* ∗ *star-of x*))
proof −
have − (*h* ∗ *star-of x*) + − (*star-of x* ∗ *star-of x*) ≈ − (*star-of x* ∗ *star-of x*)
using ‹*h* ∗ *star-of x* ∈ *Infinitesimal*› *assms bex-Infinitesimal-iff* **by** *fastforce*
then show *?thesis*
by (*metis assms mult-eq-0-iff neg-equal-0-iff-equal star-of-approx-inverse star-of-minus star-of-mult*)
qed
moreover from *not-0* ‹*h* ≠ *0*› *assms*
have *inverse* (− (*h* ∗ *star-of x*) + − (*star-of x* ∗ *star-of x*))
= (*inverse* (*star-of x* + *h*) − *inverse* (*star-of x*)) / *h*
by (*simp add*: *division-ring-inverse-diff inverse-mult-distrib* [*symmetric*]
inverse-minus-eq [*symmetric*] *algebra-simps*)
ultimately have (*inverse* (*star-of x* + *h*) − *inverse* (*star-of x*)) / *h* ≈
− (*inverse* (*star-of x*) ∗ *inverse* (*star-of x*))
using *assms* **by** *simp*
}
then show *?thesis* **by** (*simp add*: *nsderiv-def*)
qed

13.2.1 Equivalence of NS and Standard definitions

lemma *divideR-eq-divide*: $x /_R y = x / y$
by (*simp add*: *divide-inverse mult.commute*)

Now equivalence between *NSDERIV* and *DERIV*.

lemma *NSDERIV-DERIV-iff*: *NSDERIV f x* :> *D* ⟷ *DERIV f x* :> *D*
by (*simp add*: *DERIV-def NSDERIV-NSLIM-iff LIM-NSLIM-iff*)

NS version.

lemma *NSDERIV-pow*: *NSDERIV* (λ*x*. *x* ^ *n*) *x* :> *real n* ∗ (*x* ^ (*n* − *Suc 0*))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-pow*)

Derivative of inverse.

lemma *NSDERIV-inverse-fun*:
NSDERIV f x :> *d* ⟹ *f x* ≠ *0* ⟹
NSDERIV (λ*x*. *inverse* (*f x*)) *x* :> (− (*d* ∗ *inverse* (*f x* ^ *Suc* (*Suc 0*))))
for *x* :: *'a*::{*real-normed-field*}
by (*simp add*: *NSDERIV-DERIV-iff DERIV-inverse-fun del*: *power-Suc*)

Derivative of quotient.

lemma *NSDERIV-quotient*:
fixes *x* :: *'a::real-normed-field*
shows *NSDERIV f x* :> *d* ⟹ *NSDERIV g x* :> *e* ⟹ *g x* ≠ *0* ⟹
NSDERIV (λ*y*. *f y* / *g y*) *x* :> (*d* ∗ *g x* − (*e* ∗ *f x*)) / (*g x* ^ *Suc* (*Suc 0*))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-quotient del*: *power-Suc*)

lemma *CARAT-NSDERIV*:
NSDERIV f x :> *l* ⟹ ∃ *g*. (∀ *z*. *f z* − *f x* = *g z* ∗ (*z* − *x*)) ∧ *isNSCont g x* ∧ *g x* = *l*
by (*simp add*: *CARAT-DERIV NSDERIV-DERIV-iff isNSCont-isCont-iff*)

lemma *hypreal-eq-minus-iff3*: *x* = *y* + *z* ⟷ *x* + − *z* = *y*
for *x y z* :: *hypreal*
by *auto*

lemma *CARAT-DERIVD*:
assumes *all*: ∀ *z*. *f z* − *f x* = *g z* ∗ (*z* − *x*)
and *nsc*: *isNSCont g x*
shows *NSDERIV f x* :> *g x*
proof −
from *nsc* **have** ∀ *w*. *w* ≠ *star-of x* ∧ *w* ≈ *star-of x* ⟶
(∗*f*∗ *g*) *w* ∗ (*w* − *star-of x*) / (*w* − *star-of x*) ≈ *star-of* (*g x*)
by (*simp add*: *isNSCont-def*)
with *all* **show** *?thesis*
by (*simp add*: *NSDERIV-iff2 starfun-if-eq cong*: *if-cong*)
qed

13.2.2 Differentiability predicate

lemma *NSdifferentiableD*: *f NSdifferentiable x* ⟹ ∃ *D*. *NSDERIV f x* :> *D*
by (*simp add*: *NSdifferentiable-def*)

lemma *NSdifferentiableI*: *NSDERIV f x* :> *D* ⟹ *f NSdifferentiable x*
by (*force simp add*: *NSdifferentiable-def*)

13.3 (NS) Increment

lemma *incrementI*:
f NSdifferentiable x ⟹
increment f x h = (∗*f*∗ *f*) (*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*)
by (*simp add*: *increment-def*)

lemma *incrementI2*:
NSDERIV f x :> *D* ⟹
increment f x h = (∗*f*∗ *f*) (*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*)
by (*erule NSdifferentiableI* [*THEN incrementI*])

The Increment theorem – Keisler p. 65.

lemma *increment-thm*:

assumes *NSDERIV f x :> D h* ∈ *Infinitesimal h* ≠ *0*
shows ∃ *e* ∈ *Infinitesimal. increment f x h = hypreal-of-real D* ∗ *h + e* ∗ *h*
proof −
have *inc*: *increment f x h* = (∗*f*∗ *f*) (*hypreal-of-real x + h*) − *hypreal-of-real* (*f x*)
using *assms*(*1*) *incrementI2* **by** *auto*
have ((∗*f*∗ *f*) (*hypreal-of-real x + h*) − *hypreal-of-real* (*f x*)) / *h* ≈ *hypreal-of-real D*
by (*simp add*: *NSDERIVD2 assms*)
then obtain *y* **where** *y* ∈ *Infinitesimal*
((∗*f*∗ *f*) (*hypreal-of-real x + h*) − *hypreal-of-real* (*f x*)) / *h* = *hypreal-of-real D + y*
by (*metis bex-Infinitesimal-iff2*)
then have *increment f x h* / *h* = *hypreal-of-real D + y*
by (*metis inc*)
then show *?thesis*
by (*metis* (*no-types*) ‹*y* ∈ *Infinitesimal*› ‹*h* ≠ *0*› *distrib-right mult.commute nonzero-mult-div-cancel-left times-divide-eq-right*)
qed

lemma *increment-approx-zero*: *NSDERIV f x :> D* ⟹ *h* ≈ *0* ⟹ *h* ≠ *0* ⟹ *increment f x h* ≈ *0*
by (*simp add*: *NSDERIV-approx incrementI2 mem-infmal-iff*)

end

14 Nonstandard Extensions of Transcendental Functions

theory *HTranscendental*
imports *Complex-Main HSeries HDeriv*
begin

definition
exphr :: *real* ⇒ *hypreal* **where**
— define exponential function using standard part
exphr x ≡ *st*(*sumhr* (*0*, *whn*, λ*n*. *inverse* (*fact n*) ∗ (*x* ^ *n*)))

definition
sinhr :: *real* ⇒ *hypreal* **where**
sinhr x ≡ *st*(*sumhr* (*0*, *whn*, λ*n*. *sin-coeff n* ∗ *x* ^ *n*))

definition
coshr :: *real* ⇒ *hypreal* **where**
coshr x ≡ *st*(*sumhr* (*0*, *whn*, λ*n*. *cos-coeff n* ∗ *x* ^ *n*))

14.1 Nonstandard Extension of Square Root Function

lemma *STAR-sqrt-zero* [*simp*]: (**f* sqrt*) *0* = *0*
by (*simp add*: *starfun star-n-zero-num*)

lemma *STAR-sqrt-one* [*simp*]: (**f* sqrt*) *1* = *1*
by (*simp add*: *starfun star-n-one-num*)

lemma *hypreal-sqrt-pow2-iff*: ((**f* sqrt*)(*x*) ^ *2* = *x*) = (*0* ≤ *x*)
proof (*cases x*)
case (*star-n X*)
then show *?thesis*
by (*simp add*: *star-n-le star-n-zero-num starfun hrealpow star-n-eq-iff del*: *hpowr-Suc power-Suc*)
qed

lemma *hypreal-sqrt-gt-zero-pow2*: ⋀*x*. *0* < *x* ⟹ (**f* sqrt*) (*x*) ^ *2* = *x*
by *transfer simp*

lemma *hypreal-sqrt-pow2-gt-zero*: *0* < *x* ⟹ *0* < (**f* sqrt*) (*x*) ^ *2*
by (*frule hypreal-sqrt-gt-zero-pow2*, *auto*)

lemma *hypreal-sqrt-not-zero*: *0* < *x* ⟹ (**f* sqrt*) (*x*) ≠ *0*
using *hypreal-sqrt-gt-zero-pow2* **by** *fastforce*

lemma *hypreal-inverse-sqrt-pow2*:
0 < *x* ⟹ *inverse* ((**f* sqrt*)(*x*)) ^ *2* = *inverse x*
by (*simp add*: *hypreal-sqrt-gt-zero-pow2 power-inverse*)

lemma *hypreal-sqrt-mult-distrib*:
⋀*x y*. ⟦*0* < *x*; *0* <*y*⟧ ⟹
(**f* sqrt*)(*x*y*) = (**f* sqrt*)(*x*) * (**f* sqrt*)(*y*)
by *transfer* (*auto intro*: *real-sqrt-mult*)

lemma *hypreal-sqrt-mult-distrib2*:
⟦*0*≤*x*; *0*≤*y*⟧ ⟹ (**f* sqrt*)(*x*y*) = (**f* sqrt*)(*x*) * (**f* sqrt*)(*y*)
by (*auto intro*: *hypreal-sqrt-mult-distrib simp add*: *order-le-less*)

lemma *hypreal-sqrt-approx-zero* [*simp*]:
assumes *0* < *x*
shows ((**f* sqrt*) *x* ≈ *0*) ⟷ (*x* ≈ *0*)
proof −
have (**f* sqrt*) *x* ∈ *Infinitesimal* ⟷ ((**f* sqrt*) *x*)^2 ∈ *Infinitesimal*
by (*metis Infinitesimal-hrealpow pos2 power2-eq-square Infinitesimal-square-iff*)
also have ... ⟷ *x* ∈ *Infinitesimal*
by (*simp add*: *assms hypreal-sqrt-gt-zero-pow2*)
finally show *?thesis*
using *mem-infmal-iff* **by** *blast*
qed

lemma *hypreal-sqrt-approx-zero2* [*simp*]:
$0 \le x \Longrightarrow ((\ {*}f{*}\ sqrt)(x) \approx 0) = (x \approx 0)$
by (*auto simp add*: *order-le-less*)

lemma *hypreal-sqrt-gt-zero*: $\bigwedge x.\ 0 < x \Longrightarrow 0 < (\ {*}f{*}\ sqrt)(x)$
by *transfer* (*simp add*: *real-sqrt-gt-zero*)

lemma *hypreal-sqrt-ge-zero*: $0 \le x \Longrightarrow 0 \le (\ {*}f{*}\ sqrt)(x)$
by (*auto intro*: *hypreal-sqrt-gt-zero simp add*: *order-le-less*)

lemma *hypreal-sqrt-lessI*:
$\bigwedge x\ u.\ [\![0 < u;\ x < u^2]\!] \Longrightarrow (\ {*}f{*}\ sqrt)\ x < u$
proof *transfer*
show $\bigwedge x\ u.\ [\![0 < u;\ x < u^2]\!] \Longrightarrow sqrt\ x < u$
by (*metis less-le real-sqrt-less-iff real-sqrt-pow2 real-sqrt-power*)
qed

lemma *hypreal-sqrt-hrabs* [*simp*]: $\bigwedge x.\ (\ {*}f{*}\ sqrt)(x^2) = |x|$
by *transfer simp*

lemma *hypreal-sqrt-hrabs2* [*simp*]: $\bigwedge x.\ (\ {*}f{*}\ sqrt)(x{*}x) = |x|$
by *transfer simp*

lemma *hypreal-sqrt-hyperpow-hrabs* [*simp*]:
$\bigwedge x.\ (\ {*}f{*}\ sqrt)(x\ pow\ (hypnat\text{-}of\text{-}nat\ 2)) = |x|$
by *transfer simp*

lemma *star-sqrt-HFinite*: $[\![x \in HFinite;\ 0 \le x]\!] \Longrightarrow (\ {*}f{*}\ sqrt)\ x \in HFinite$
by (*metis HFinite-square-iff hypreal-sqrt-pow2-iff power2-eq-square*)

lemma *st-hypreal-sqrt*:
assumes $x \in HFinite\ 0 \le x$
shows $st((\ {*}f{*}\ sqrt)\ x) = (\ {*}f{*}\ sqrt)(st\ x)$
proof (*rule power-inject-base*)
show $st\ (({*}f{*}\ sqrt)\ x)\ \hat{}\ Suc\ 1 = ({*}f{*}\ sqrt)\ (st\ x)\ \hat{}\ Suc\ 1$
using *assms hypreal-sqrt-pow2-iff* [*of x*]
by (*metis HFinite-square-iff hypreal-sqrt-hrabs2 power2-eq-square st-hrabs st-mult*)
show $0 \le st\ (({*}f{*}\ sqrt)\ x)$
by (*simp add*: *assms hypreal-sqrt-ge-zero st-zero-le star-sqrt-HFinite*)
show $0 \le ({*}f{*}\ sqrt)\ (st\ x)$
by (*simp add*: *assms hypreal-sqrt-ge-zero st-zero-le*)
qed

lemma *hypreal-sqrt-sum-squares-ge1* [*simp*]: $\bigwedge x\ y.\ x \le (\ {*}f{*}\ sqrt)(x^2 + y^2)$
by *transfer* (*rule real-sqrt-sum-squares-ge1*)

lemma *HFinite-hypreal-sqrt-imp-HFinite*:
$[\![0 \le x;\ (\ {*}f{*}\ sqrt)\ x \in HFinite]\!] \Longrightarrow x \in HFinite$
by (*metis HFinite-mult hypreal-sqrt-pow2-iff power2-eq-square*)

lemma *HFinite-hypreal-sqrt-iff* [*simp*]:
$0 \le x \Longrightarrow ((\ {*}f{*}\ sqrt)\ x \in HFinite) = (x \in HFinite)$
by (*blast intro*: *star-sqrt-HFinite HFinite-hypreal-sqrt-imp-HFinite*)

lemma *Infinitesimal-hypreal-sqrt*:
$[\![0 \le x;\ x \in Infinitesimal]\!] \Longrightarrow (\ {*}f{*}\ sqrt)\ x \in Infinitesimal$
by (*simp add*: *mem-infmal-iff*)

lemma *Infinitesimal-hypreal-sqrt-imp-Infinitesimal*:
$[\![0 \le x;\ (\ {*}f{*}\ sqrt)\ x \in Infinitesimal]\!] \Longrightarrow x \in Infinitesimal$
using *hypreal-sqrt-approx-zero2 mem-infmal-iff* **by** *blast*

lemma *Infinitesimal-hypreal-sqrt-iff* [*simp*]:
$0 \le x \Longrightarrow ((\ {*}f{*}\ sqrt)\ x \in Infinitesimal) = (x \in Infinitesimal)$
by (*blast intro*: *Infinitesimal-hypreal-sqrt-imp-Infinitesimal Infinitesimal-hypreal-sqrt*)

lemma *HInfinite-hypreal-sqrt*:
$[\![0 \le x;\ x \in HInfinite]\!] \Longrightarrow (\ {*}f{*}\ sqrt)\ x \in HInfinite$
by (*simp add*: *HInfinite-HFinite-iff*)

lemma *HInfinite-hypreal-sqrt-imp-HInfinite*:
$[\![0 \le x;\ (\ {*}f{*}\ sqrt)\ x \in HInfinite]\!] \Longrightarrow x \in HInfinite$
using *HFinite-hypreal-sqrt-iff HInfinite-HFinite-iff* **by** *blast*

lemma *HInfinite-hypreal-sqrt-iff* [*simp*]:
$0 \le x \Longrightarrow ((\ {*}f{*}\ sqrt)\ x \in HInfinite) = (x \in HInfinite)$
by (*blast intro*: *HInfinite-hypreal-sqrt HInfinite-hypreal-sqrt-imp-HInfinite*)

lemma *HFinite-exp* [*simp*]:
$sumhr\ (0,\ whn,\ \lambda n.\ inverse\ (fact\ n) * x \ \hat{}\ n) \in HFinite$
unfolding *sumhr-app star-zero-def starfun2-star-of atLeast0LessThan*
by (*metis NSBseqD2 NSconvergent-NSBseq convergent-NSconvergent-iff summable-iff-convergent summable-exp*)

lemma *exphr-zero* [*simp*]: $exphr\ 0 = 1$
proof −
have $\forall x{>}1.\ 1 = sumhr\ (0,\ 1,\ \lambda n.\ inverse\ (fact\ n) * 0 \ \hat{}\ n) + sumhr\ (1,\ x,\ \lambda n.\ inverse\ (fact\ n) * 0 \ \hat{}\ n)$
unfolding *sumhr-app* **by** *transfer* (*simp add*: *power-0-left*)
then have $sumhr\ (0,\ 1,\ \lambda n.\ inverse\ (fact\ n) * 0 \ \hat{}\ n) + sumhr\ (1,\ whn,\ \lambda n.\ inverse\ (fact\ n) * 0 \ \hat{}\ n) \approx 1$
by *auto*
then show *?thesis*
unfolding *exphr-def*
using *sumhr-split-add* [*OF hypnat-one-less-hypnat-omega*] *st-unique* **by** *auto*
qed

lemma *coshr-zero* [*simp*]: $coshr\ 0 = 1$

proof −
have $\forall x>1.\ 1 = sumhr\ (0,\ 1,\ \lambda n.\ cos\text{-}coeff\ n * 0 \mathbin{\hat{}} n) + sumhr\ (1,\ x,\ \lambda n.\ cos\text{-}coeff\ n * 0 \mathbin{\hat{}} n)$
unfolding *sumhr-app* **by** *transfer* (*simp add*: *power-0-left*)
then have $sumhr\ (0,\ 1,\ \lambda n.\ cos\text{-}coeff\ n * 0 \mathbin{\hat{}} n) + sumhr\ (1,\ whn,\ \lambda n.\ cos\text{-}coeff\ n * 0 \mathbin{\hat{}} n) \approx 1$
by *auto*
then show *?thesis*
unfolding *coshr-def*
using *sumhr-split-add* [*OF hypnat-one-less-hypnat-omega*] *st-unique* **by** *auto*
qed

lemma *STAR-exp-zero-approx-one* [*simp*]: $(*f*\ exp)\ (0::hypreal) \approx 1$
proof −
have $(*f*\ exp)\ (0::real\ star) = 1$
by *transfer simp*
then show *?thesis*
by *auto*
qed

lemma *STAR-exp-Infinitesimal*:
assumes $x \in Infinitesimal$ **shows** $(*f*\ exp)\ (x::hypreal) \approx 1$
proof ($cases\ x = 0$)
case *False*
have $NSDERIV\ exp\ 0 :> 1$
by (*metis DERIV-exp NSDERIV-DERIV-iff exp-zero*)
then have $((*f*\ exp)\ x - 1) / x \approx 1$
using *nsderiv-def False NSDERIVD2 assms* **by** *fastforce*
then have $(*f*\ exp)\ x - 1 \approx x$
using *NSDERIVD4* ‹$NSDERIV\ exp\ 0 :> 1$› *assms* **by** *fastforce*
then show *?thesis*
by (*meson Infinitesimal-approx approx-minus-iff approx-trans2 assms not-Infinitesimal-not-zero*)
qed *auto*

lemma *STAR-exp-epsilon* [*simp*]: $(*f*\ exp)\ \varepsilon \approx 1$
by (*auto intro*: *STAR-exp-Infinitesimal*)

lemma *STAR-exp-add*:
$\bigwedge(x::'a::\ \{banach,real\text{-}normed\text{-}field\}\ star)\ y.\ (*f*\ exp)(x + y) = (*f*\ exp)\ x * (*f*\ exp)\ y$
by *transfer* (*rule exp-add*)

lemma *exphr-hypreal-of-real-exp-eq*: $exphr\ x = hypreal\text{-}of\text{-}real\ (exp\ x)$
proof −
have $(\lambda n.\ inverse\ (fact\ n) * x \mathbin{\hat{}} n)\ sums\ exp\ x$
using *exp-converges* [*of x*] **by** *simp*
then have $(\lambda n.\ \sum n<n.\ inverse\ (fact\ n) * x \mathbin{\hat{}} n) \longrightarrow_{NS} exp\ x$
using *NSsums-def sums-NSsums-iff* **by** *blast*
then have $hypreal\text{-}of\text{-}real\ (exp\ x) \approx sumhr\ (0,\ whn,\ \lambda n.\ inverse\ (fact\ n) * x \mathbin{\hat{}}$

n)
  unfolding *starfunNat-sumr* [*symmetric*] *atLeast0LessThan*
  using *HNatInfinite-whn NSLIMSEQ-def approx-sym* **by** *blast*
 then show *?thesis*
  unfolding *exphr-def* **using** *st-eq-approx-iff* **by** *auto*
qed

lemma *starfun-exp-ge-add-one-self* [*simp*]: $\bigwedge x$::*hypreal*. $0 \leq x \Longrightarrow (1 + x) \leq ($ $*f*$ *exp*) *x*
 by *transfer* (*rule exp-ge-add-one-self-aux*)

exp maps infinities to infinities

lemma *starfun-exp-HInfinite*:
 fixes *x* :: *hypreal*
 assumes *x* ∈ *HInfinite* $0 \leq x$
 shows ($*f*$ *exp*) *x* ∈ *HInfinite*
proof −
 have $x \leq 1 + x$
  by *simp*
 also have ... ≤ ($*f*$ *exp*) *x*
  by (*simp add*: ‹$0 \leq x$›)
 finally show *?thesis*
  using *HInfinite-ge-HInfinite assms* **by** *blast*
qed

lemma *starfun-exp-minus*:
 $\bigwedge x$::*'a*:: {*banach*,*real-normed-field*} *star*. ($*f*$ *exp*) $(-x)$ = *inverse*(($*f*$ *exp*) *x*)
 by *transfer* (*rule exp-minus*)

exp maps infinitesimals to infinitesimals

lemma *starfun-exp-Infinitesimal*:
 fixes *x* :: *hypreal*
 assumes *x* ∈ *HInfinite* $x \leq 0$
 shows ($*f*$ *exp*) *x* ∈ *Infinitesimal*
proof −
 obtain *y* **where** $x = -y$ $y \geq 0$
  by (*metis abs-of-nonpos assms*(*2*) *eq-abs-iff'*)
 then have ($*f*$ *exp*) *y* ∈ *HInfinite*
  using *HInfinite-minus-iff assms*(*1*) *starfun-exp-HInfinite* **by** *blast*
 then show *?thesis*
  by (*simp add*: *HInfinite-inverse-Infinitesimal* ‹$x = - y$› *starfun-exp-minus*)
qed

lemma *starfun-exp-gt-one* [*simp*]: $\bigwedge x$::*hypreal*. $0 < x \Longrightarrow 1 < ($ $*f*$ *exp*) *x*
 by *transfer* (*rule exp-gt-one*)

abbreviation *real-ln* :: *real* ⇒ *real* **where**
 real-ln ≡ *ln*

lemma *starfun-ln-exp* [*simp*]: $\bigwedge x.\ (\ *f*\ real\text{-}ln)\ ((\ *f*\ exp)\ x) = x$
  by *transfer* (*rule ln-exp*)

lemma *starfun-exp-ln-iff* [*simp*]: $\bigwedge x.\ ((\ *f*\ exp)((\ *f*\ real\text{-}ln)\ x) = x) = (0 < x)$
  by *transfer* (*rule exp-ln-iff*)

lemma *starfun-exp-ln-eq*: $\bigwedge u\ x.\ (\ *f*\ exp)\ u = x \Longrightarrow (\ *f*\ real\text{-}ln)\ x = u$
  by *transfer* (*rule ln-unique*)

lemma *starfun-ln-less-self* [*simp*]: $\bigwedge x.\ 0 < x \Longrightarrow (\ *f*\ real\text{-}ln)\ x < x$
  by *transfer* (*rule ln-less-self*)

lemma *starfun-ln-ge-zero* [*simp*]: $\bigwedge x.\ 1 \le x \Longrightarrow 0 \le (\ *f*\ real\text{-}ln)\ x$
  by *transfer* (*rule ln-ge-zero*)

lemma *starfun-ln-gt-zero* [*simp*]: $\bigwedge x\ .1 < x \Longrightarrow 0 < (\ *f*\ real\text{-}ln)\ x$
  by *transfer* (*rule ln-gt-zero*)

lemma *starfun-ln-not-eq-zero* [*simp*]: $\bigwedge x.\ [\![0 < x;\ x \neq 1]\!] \Longrightarrow (\ *f*\ real\text{-}ln)\ x \neq 0$
  by *transfer simp*

lemma *starfun-ln-HFinite*: $[\![x \in HFinite;\ 1 \le x]\!] \Longrightarrow (\ *f*\ real\text{-}ln)\ x \in HFinite$
  by (*metis HFinite-HInfinite-iff less-le-trans starfun-exp-HInfinite starfun-exp-ln-iff starfun-ln-ge-zero zero-less-one*)

lemma *starfun-ln-inverse*: $\bigwedge x.\ 0 < x \Longrightarrow (\ *f*\ real\text{-}ln)\ (inverse\ x) = -(\ *f*\ ln)\ x$
  by *transfer* (*rule ln-inverse*)

lemma *starfun-abs-exp-cancel*: $\bigwedge x.\ |(\ *f*\ exp)\ (x{::}hypreal)| = (\ *f*\ exp)\ x$
  by *transfer* (*rule abs-exp-cancel*)

lemma *starfun-exp-less-mono*: $\bigwedge x\ y{::}hypreal.\ x < y \Longrightarrow (\ *f*\ exp)\ x < (\ *f*\ exp)\ y$
  by *transfer* (*rule exp-less-mono*)

lemma *starfun-exp-HFinite*:
  fixes $x :: hypreal$
  assumes $x \in HFinite$
  shows $(\ *f*\ exp)\ x \in HFinite$
proof −
  obtain u **where** $u \in \mathbb{R}\ |x| < u$
    using *HFiniteD assms* **by** *force*
  with *assms* **have** $|(*f*\ exp)\ x| < (*f*\ exp)\ u$
    using *starfun-abs-exp-cancel starfun-exp-less-mono* **by** *auto*
  with ‹$u \in \mathbb{R}$› **show** *?thesis*
    by (*force simp*: *HFinite-def Reals-eq-Standard*)
qed

lemma *starfun-exp-add-HFinite-Infinitesimal-approx*:
fixes *x* :: *hypreal*
shows ⟦*x* ∈ *Infinitesimal*; *z* ∈ *HFinite*⟧ ⟹ (*∗f∗ exp*) (*z* + *x*::*hypreal*) ≈ (*∗f∗ exp*) *z*
using *STAR-exp-Infinitesimal approx-mult2 starfun-exp-HFinite* **by** (*fastforce simp add*: *STAR-exp-add*)

lemma *starfun-ln-HInfinite*:
⟦*x* ∈ *HInfinite*; *0* < *x*⟧ ⟹ (*∗f∗ real-ln*) *x* ∈ *HInfinite*
by (*metis HInfinite-HFinite-iff starfun-exp-HFinite starfun-exp-ln-iff*)

lemma *starfun-exp-HInfinite-Infinitesimal-disj*:
fixes *x* :: *hypreal*
shows *x* ∈ *HInfinite* ⟹ (*∗f∗ exp*) *x* ∈ *HInfinite* ∨ (*∗f∗ exp*) (*x*::*hypreal*) ∈ *Infinitesimal*
by (*meson linear starfun-exp-HInfinite starfun-exp-Infinitesimal*)

lemma *starfun-ln-HFinite-not-Infinitesimal*:
⟦*x* ∈ *HFinite* − *Infinitesimal*; *0* < *x*⟧ ⟹ (*∗f∗ real-ln*) *x* ∈ *HFinite*
by (*metis DiffD1 DiffD2 HInfinite-HFinite-iff starfun-exp-HInfinite-Infinitesimal-disj starfun-exp-ln-iff*)

lemma *starfun-ln-Infinitesimal-HInfinite*:
assumes *x* ∈ *Infinitesimal 0* < *x*
shows (*∗f∗ real-ln*) *x* ∈ *HInfinite*
proof −
have *inverse x* ∈ *HInfinite*
using *Infinitesimal-inverse-HInfinite assms* **by** *blast*
then show *?thesis*
using *HInfinite-minus-iff assms*(*2*) *starfun-ln-HInfinite starfun-ln-inverse* **by** *fastforce*
qed

lemma *starfun-ln-less-zero*: ⋀*x*. ⟦*0* < *x*; *x* < *1*⟧ ⟹ (*∗f∗ real-ln*) *x* < *0*
by *transfer* (*rule ln-less-zero*)

lemma *starfun-ln-Infinitesimal-less-zero*:
⟦*x* ∈ *Infinitesimal*; *0* < *x*⟧ ⟹ (*∗f∗ real-ln*) *x* < *0*
by (*auto intro*!: *starfun-ln-less-zero simp add*: *Infinitesimal-def*)

lemma *starfun-ln-HInfinite-gt-zero*:
⟦*x* ∈ *HInfinite*; *0* < *x*⟧ ⟹ *0* < (*∗f∗ real-ln*) *x*
by (*auto intro*!: *starfun-ln-gt-zero simp add*: *HInfinite-def*)

lemma *HFinite-sin* [*simp*]: *sumhr* (*0*, *whn*, *λn*. *sin-coeff n* ∗ *x* ^ *n*) ∈ *HFinite*
proof −
have *summable* (*λi*. *sin-coeff i* ∗ *x* ^ *i*)

```
    using summable-norm-sin [of x] by (simp add: summable-rabs-cancel)
  then have (*f* (λn. ∑n<n. sin-coeff n * x ^ n)) whn ∈ HFinite
    unfolding summable-sums-iff sums-NSsums-iff NSsums-def NSLIMSEQ-def
    using HFinite-star-of HNatInfinite-whn approx-HFinite approx-sym by blast
  then show ?thesis
    unfolding sumhr-app
    by (simp only: star-zero-def starfun2-star-of atLeast0LessThan)
qed

lemma STAR-sin-zero [simp]: ( *f* sin) 0 = 0
  by transfer (rule sin-zero)

lemma STAR-sin-Infinitesimal [simp]:
  fixes x :: 'a::{real-normed-field,banach} star
  assumes x ∈ Infinitesimal
  shows ( *f* sin) x ≈ x
proof (cases x = 0)
  case False
  have NSDERIV sin 0 :> 1
    by (metis DERIV-sin NSDERIV-DERIV-iff cos-zero)
  then have (*f* sin) x / x ≈ 1
    using False NSDERIVD2 assms by fastforce
  with assms show ?thesis
    unfolding star-one-def
    by (metis False Infinitesimal-approx Infinitesimal-ratio approx-star-of-HFinite)
qed auto

lemma HFinite-cos [simp]: sumhr (0, whn, λn. cos-coeff n * x ^ n) ∈ HFinite
proof −
  have summable (λi. cos-coeff i * x ^ i)
    using summable-norm-cos [of x] by (simp add: summable-rabs-cancel)
  then have (*f* (λn. ∑n<n. cos-coeff n * x ^ n)) whn ∈ HFinite
    unfolding summable-sums-iff sums-NSsums-iff NSsums-def NSLIMSEQ-def
    using HFinite-star-of HNatInfinite-whn approx-HFinite approx-sym by blast
  then show ?thesis
    unfolding sumhr-app
    by (simp only: star-zero-def starfun2-star-of atLeast0LessThan)
qed

lemma STAR-cos-zero [simp]: ( *f* cos) 0 = 1
  by transfer (rule cos-zero)

lemma STAR-cos-Infinitesimal [simp]:
  fixes x :: 'a::{real-normed-field,banach} star
  assumes x ∈ Infinitesimal
  shows ( *f* cos) x ≈ 1
proof (cases x = 0)
  case False
  have NSDERIV cos 0 :> 0
```

by (*metis DERIV-cos NSDERIV-DERIV-iff add.inverse-neutral sin-zero*)
then have (**f* cos*) *x* − *1* ≈ *0*
using *NSDERIV-approx assms* **by** *fastforce*
with *assms* **show** *?thesis*
using *approx-minus-iff* **by** *blast*
qed *auto*

lemma *STAR-tan-zero* [*simp*]: (**f* tan*) *0* = *0*
by *transfer* (*rule tan-zero*)

lemma *STAR-tan-Infinitesimal* [*simp*]:
assumes *x* ∈ *Infinitesimal*
shows (**f* tan*) *x* ≈ *x*
proof (*cases x* = *0*)
case *False*
have *NSDERIV tan 0* :> *1*
using *DERIV-tan* [*of 0*] **by** (*simp add*: *NSDERIV-DERIV-iff*)
then have (**f* tan*) *x* / *x* ≈ *1*
using *False NSDERIVD2 assms* **by** *fastforce*
with *assms* **show** *?thesis*
unfolding *star-one-def*
by (*metis False Infinitesimal-approx Infinitesimal-ratio approx-star-of-HFinite*)
qed *auto*

lemma *STAR-sin-cos-Infinitesimal-mult*:
fixes *x* :: *'a*::{*real-normed-field*,*banach*} *star*
shows *x* ∈ *Infinitesimal* ⟹ (**f* sin*) *x* * (**f* cos*) *x* ≈ *x*
using *approx-mult-HFinite* [*of* (**f* sin*) *x* - (**f* cos*) *x 1*]
by (*simp add*: *Infinitesimal-subset-HFinite* [*THEN subsetD*])

lemma *HFinite-pi*: *hypreal-of-real pi* ∈ *HFinite*
by *simp*

lemma *STAR-sin-Infinitesimal-divide*:
fixes *x* :: *'a*::{*real-normed-field*,*banach*} *star*
shows ⟦*x* ∈ *Infinitesimal*; *x* ≠ *0*⟧ ⟹ (**f* sin*) *x*/*x* ≈ *1*
using *DERIV-sin* [*of 0*::*'a*]
by (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)

14.2 Proving $\sin *(1/n) \times 1/(1/n) \approx 1$ for $n = \infty$

lemma *lemma-sin-pi*:
n ∈ *HNatInfinite*
⟹ (**f* sin*) (*inverse* (*hypreal-of-hypnat n*))/(*inverse* (*hypreal-of-hypnat n*))
≈ *1*
by (*simp add*: *STAR-sin-Infinitesimal-divide zero-less-HNatInfinite*)

lemma *STAR-sin-inverse-HNatInfinite*:

n ∈ *HNatInfinite*
⟹ (*∗f∗ sin*) (*inverse* (*hypreal-of-hypnat n*)) ∗ *hypreal-of-hypnat n* ≈ *1*
by (*metis field-class.field-divide-inverse inverse-inverse-eq lemma-sin-pi*)

lemma *Infinitesimal-pi-divide-HNatInfinite*:
N ∈ *HNatInfinite*
⟹ *hypreal-of-real pi*/(*hypreal-of-hypnat N*) ∈ *Infinitesimal*
by (*simp add*: *Infinitesimal-HFinite-mult2 field-class.field-divide-inverse*)

lemma *pi-divide-HNatInfinite-not-zero* [*simp*]:
N ∈ *HNatInfinite* ⟹ *hypreal-of-real pi*/(*hypreal-of-hypnat N*) ≠ *0*
by (*simp add*: *zero-less-HNatInfinite*)

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi*:
assumes *n* ∈ *HNatInfinite*
shows (*∗f∗ sin*) (*hypreal-of-real pi* / *hypreal-of-hypnat n*) ∗ *hypreal-of-hypnat n* ≈
hypreal-of-real pi
proof −
have (*∗f∗ sin*) (*hypreal-of-real pi* / *hypreal-of-hypnat n*) / (*hypreal-of-real pi* / *hypreal-of-hypnat n*) ≈ *1*
using *Infinitesimal-pi-divide-HNatInfinite STAR-sin-Infinitesimal-divide assms pi-divide-HNatInfinite-not-zero* **by** *blast*
then have *hypreal-of-hypnat n* ∗ *star-of sin* ⋆ (*hypreal-of-real pi* / *hypreal-of-hypnat n*) / *hypreal-of-real pi* ≈ *1*
by (*simp add*: *mult.commute starfun-def*)
then show *?thesis*
apply (*simp add*: *starfun-def field-simps*)
by (*metis* (*no-types*, *lifting*) *approx-mult-subst-star-of approx-refl mult-cancel-right1 nonzero-eq-divide-eq pi-neq-zero star-of-eq-0*)
qed

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi2*:
n ∈ *HNatInfinite*
⟹ *hypreal-of-hypnat n* ∗ (*∗f∗ sin*) (*hypreal-of-real pi*/(*hypreal-of-hypnat n*)) ≈ *hypreal-of-real pi*
by (*metis STAR-sin-pi-divide-HNatInfinite-approx-pi mult.commute*)

lemma *starfunNat-pi-divide-n-Infinitesimal*:
N ∈ *HNatInfinite* ⟹ (*∗f∗* (λ*x*. *pi* / *real x*)) *N* ∈ *Infinitesimal*
by (*simp add*: *Infinitesimal-HFinite-mult2 divide-inverse starfunNat-real-of-nat*)

lemma *STAR-sin-pi-divide-n-approx*:
assumes *N* ∈ *HNatInfinite*
shows (*∗f∗ sin*) ((*∗f∗* (λ*x*. *pi* / *real x*)) *N*) ≈ *hypreal-of-real pi*/(*hypreal-of-hypnat N*)
proof −
have ∃ *s*. (*∗f∗ sin*) ((*∗f∗* (λ*n*. *pi* / *real n*)) *N*) ≈ *s* ∧ *hypreal-of-real pi* / *hypreal-of-hypnat N* ≈ *s*

by (*metis* (*lifting*) *Infinitesimal-approx Infinitesimal-pi-divide-HNatInfinite STAR-sin-Infinitesimal assms starfunNat-pi-divide-n-Infinitesimal*)
then show *?thesis*
by (*meson approx-trans2*)
qed

lemma *NSLIMSEQ-sin-pi*: ($\lambda n.$ *real n* $*$ *sin* (*pi* / *real n*)) $\longrightarrow_{NS}$ *pi*
proof –
have $*$: *hypreal-of-hypnat N* $*$ ($*f*$ *sin*) (($*f*$ ($\lambda x.$ *pi* / *real x*)) *N*) $\approx$ *hypreal-of-real pi*
if $N \in$ *HNatInfinite*
for *N* :: *nat star*
using *that*
by *simp* (*metis STAR-sin-pi-divide-HNatInfinite-approx-pi2 starfunNat-real-of-nat*)
show *?thesis*
by (*simp add*: *NSLIMSEQ-def starfunNat-real-of-nat*) (*metis* $*$ *starfun-o2*)
qed

lemma *NSLIMSEQ-cos-one*: ($\lambda n.$ *cos* (*pi* / *real n*))$\longrightarrow_{NS}$ *1*
proof –
have ($*f*$ *cos*) (($*f*$ ($\lambda x.$ *pi* / *real x*)) *N*) $\approx$ *1*
if $N \in$ *HNatInfinite* **for** *N*
using *that STAR-cos-Infinitesimal starfunNat-pi-divide-n-Infinitesimal* **by** *blast*
then show *?thesis*
by (*simp add*: *NSLIMSEQ-def*) (*metis STAR-cos-Infinitesimal starfunNat-pi-divide-n-Infinitesimal starfun-o2*)
qed

lemma *NSLIMSEQ-sin-cos-pi*:
($\lambda n.$ *real n* $*$ *sin* (*pi* / *real n*) $*$ *cos* (*pi* / *real n*)) $\longrightarrow_{NS}$ *pi*
using *NSLIMSEQ-cos-one NSLIMSEQ-mult NSLIMSEQ-sin-pi* **by** *force*

A familiar approximation to *cos x* when *x* is small

lemma *STAR-cos-Infinitesimal-approx*:
fixes *x* :: *'a*::{*real-normed-field*,*banach*} *star*
shows $x \in$ *Infinitesimal* $\Longrightarrow$ ($*f*$ *cos*) $x \approx 1 - x^2$
by (*metis Infinitesimal-square-iff STAR-cos-Infinitesimal approx-diff approx-sym diff-zero mem-infmal-iff power2-eq-square*)

lemma *STAR-cos-Infinitesimal-approx2*:
fixes *x* :: *hypreal*
assumes $x \in$ *Infinitesimal*
shows ($*f*$ *cos*) $x \approx 1 - (x^2)/2$
proof –
have $1 \approx 1 - x^2 / 2$
using *assms*
by (*auto intro*: *Infinitesimal-SReal-divide simp add*: *Infinitesimal-approx-minus* [*symmetric*] *numeral-2-eq-2*)
then show *?thesis*

using *STAR-cos-Infinitesimal approx-trans assms* **by** *blast*
qed

end

15 Non-Standard Complex Analysis

theory *NSCA*
imports *NSComplex HTranscendental*
begin

abbreviation

SComplex :: *hcomplex set* **where**
SComplex ≡ *Standard*

definition — standard part map
stc :: *hcomplex* => *hcomplex* **where**
stc x = (*SOME r*. *x* ∈ *HFinite* ∧ *r*∈*SComplex* ∧ *r* ≈ *x*)

15.1 Closure Laws for SComplex, the Standard Complex Numbers

lemma *SComplex-minus-iff* [*simp*]: (−*x* ∈ *SComplex*) = (*x* ∈ *SComplex*)
using *Standard-minus* **by** *fastforce*

lemma *SComplex-add-cancel*:
⟦*x* + *y* ∈ *SComplex*; *y* ∈ *SComplex*⟧ ⟹ *x* ∈ *SComplex*
using *Standard-diff* **by** *fastforce*

lemma *SReal-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (*hcomplex-of-complex r*) ∈ ℝ
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-hcmod-numeral*: *hcmod* (*numeral w* ::*hcomplex*) ∈ ℝ
by *simp*

lemma *SReal-hcmod-SComplex*: *x* ∈ *SComplex* ⟹ *hcmod x* ∈ ℝ
by (*simp add*: *Reals-eq-Standard*)

lemma *SComplex-divide-numeral*:
r ∈ *SComplex* ⟹ *r*/(*numeral w*::*hcomplex*) ∈ *SComplex*
by *simp*

lemma *SComplex-UNIV-complex*:
{*x*. *hcomplex-of-complex x* ∈ *SComplex*} = (*UNIV*::*complex set*)
by *simp*

lemma *SComplex-iff*: (*x* ∈ *SComplex*) = (∃ *y*. *x* = *hcomplex-of-complex y*)

by (*simp add*: *Standard-def image-def*)

lemma *hcomplex-of-complex-image*:
range hcomplex-of-complex = *SComplex*
by (*simp add*: *Standard-def*)

lemma *inv-hcomplex-of-complex-image*: *inv hcomplex-of-complex ‘SComplex* = *UNIV*
by (*auto simp add*: *Standard-def image-def*) (*metis inj-star-of inv-f-f*)

lemma *SComplex-hcomplex-of-complex-image*:
$[\![\exists x.\ x \in P;\ P \le SComplex]\!] \Longrightarrow \exists Q.\ P = hcomplex\text{-}of\text{-}complex\ `\ Q$
by (*metis Standard-def subset-imageE*)

lemma *SComplex-SReal-dense*:
$[\![x \in SComplex;\ y \in SComplex;\ hcmod\ x < hcmod\ y$
$]\!] \Longrightarrow \exists r \in Reals.\ hcmod\ x< r \wedge r < hcmod\ y$
by (*simp add*: *SReal-dense SReal-hcmod-SComplex*)

15.2 The Finite Elements form a Subring

lemma *HFinite-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (*hcomplex-of-complex r*) ∈ *HFinite*
by (*auto intro*!: *SReal-subset-HFinite* [*THEN subsetD*])

lemma *HFinite-hcmod-iff* [*simp*]: *hcmod x* ∈ *HFinite* ⟷ *x* ∈ *HFinite*
by (*simp add*: *HFinite-def*)

lemma *HFinite-bounded-hcmod*:
$[\![x \in HFinite;\ y \le hcmod\ x;\ 0 \le y]\!] \Longrightarrow y \in HFinite$
using *HFinite-bounded HFinite-hcmod-iff* **by** *blast*

15.3 The Complex Infinitesimals form a Subring

lemma *Infinitesimal-hcmod-iff*:
(*z* ∈ *Infinitesimal*) = (*hcmod z* ∈ *Infinitesimal*)
by (*simp add*: *Infinitesimal-def*)

lemma *HInfinite-hcmod-iff*: (*z* ∈ *HInfinite*) = (*hcmod z* ∈ *HInfinite*)
by (*simp add*: *HInfinite-def*)

lemma *HFinite-diff-Infinitesimal-hcmod*:
x ∈ *HFinite* − *Infinitesimal* ⟹ *hcmod x* ∈ *HFinite* − *Infinitesimal*
by (*simp add*: *Infinitesimal-hcmod-iff*)

lemma *hcmod-less-Infinitesimal*:
$[\![e \in Infinitesimal;\ hcmod\ x < hcmod\ e]\!] \Longrightarrow x \in Infinitesimal$
by (*auto elim*: *hrabs-less-Infinitesimal simp add*: *Infinitesimal-hcmod-iff*)

lemma *hcmod-le-Infinitesimal*:
$[\![e \in Infinitesimal;\ hcmod\ x \le hcmod\ e]\!] \Longrightarrow x \in Infinitesimal$

by (*auto elim*: *hrabs-le-Infinitesimal simp add*: *Infinitesimal-hcmod-iff*)

15.4 The “Infinitely Close” Relation

lemma *approx-SComplex-mult-cancel-zero*:
$[\![a \in SComplex;\ a \neq 0;\ a*x \approx 0]\!] \Longrightarrow x \approx 0$
by (*metis Infinitesimal-mult-disj SComplex-iff mem-infmal-iff star-of-Infinitesimal-iff-0 star-zero-def*)

lemma *approx-mult-SComplex1*: $[\![a \in SComplex;\ x \approx 0]\!] \Longrightarrow x*a \approx 0$
using *SComplex-iff approx-mult-subst-star-of* **by** *fastforce*

lemma *approx-mult-SComplex2*: $[\![a \in SComplex;\ x \approx 0]\!] \Longrightarrow a*x \approx 0$
by (*metis approx-mult-SComplex1 mult.commute*)

lemma *approx-mult-SComplex-zero-cancel-iff* [*simp*]:
$[\![a \in SComplex;\ a \neq 0]\!] \Longrightarrow (a*x \approx 0) = (x \approx 0)$
using *approx-SComplex-mult-cancel-zero approx-mult-SComplex2* **by** *blast*

lemma *approx-SComplex-mult-cancel*:
$[\![a \in SComplex;\ a \neq 0;\ a*w \approx a*z]\!] \Longrightarrow w \approx z$
by (*metis approx-SComplex-mult-cancel-zero approx-minus-iff right-diff-distrib*)

lemma *approx-SComplex-mult-cancel-iff1* [*simp*]:
$[\![a \in SComplex;\ a \neq 0]\!] \Longrightarrow (a*w \approx a*z) = (w \approx z)$
by (*metis HFinite-star-of SComplex-iff approx-SComplex-mult-cancel approx-mult2*)

lemma *approx-hcmod-approx-zero*: $(x \approx y) = (hcmod\ (y - x) \approx 0)$
by (*simp add*: *Infinitesimal-hcmod-iff approx-def hnorm-minus-commute*)

lemma *approx-approx-zero-iff*: $(x \approx 0) = (hcmod\ x \approx 0)$
by (*simp add*: *approx-hcmod-approx-zero*)

lemma *approx-minus-zero-cancel-iff* [*simp*]: $(-x \approx 0) = (x \approx 0)$
by (*simp add*: *approx-def*)

lemma *Infinitesimal-hcmod-add-diff*:
$u \approx 0 \Longrightarrow hcmod(x + u) - hcmod\ x \in Infinitesimal$
by (*metis add.commute add.left-neutral approx-add-right-iff approx-def approx-hnorm*)

lemma *approx-hcmod-add-hcmod*: $u \approx 0 \Longrightarrow hcmod(x + u) \approx hcmod\ x$
using *Infinitesimal-hcmod-add-diff approx-def* **by** *blast*

15.5 Zero is the Only Infinitesimal Complex Number

lemma *Infinitesimal-less-SComplex*:
$[\![x \in SComplex;\ y \in Infinitesimal;\ 0 < hcmod\ x]\!] \Longrightarrow hcmod\ y < hcmod\ x$

by (*auto intro*: *Infinitesimal-less-SReal SReal-hcmod-SComplex simp add*: *Infinitesimal-hcmod-iff*)

lemma *SComplex-Int-Infinitesimal-zero*: *SComplex Int Infinitesimal* = {*0*}
by (*auto simp add*: *Standard-def Infinitesimal-hcmod-iff*)

lemma *SComplex-Infinitesimal-zero*:
$[\![x \in SComplex; x \in Infinitesimal]\!] \Longrightarrow x = 0$
using *SComplex-iff* **by** *auto*

lemma *SComplex-HFinite-diff-Infinitesimal*:
$[\![x \in SComplex; x \neq 0]\!] \Longrightarrow x \in HFinite - Infinitesimal$
using *SComplex-iff* **by** *auto*

lemma *numeral-not-Infinitesimal* [*simp*]:
$numeral\ w \neq (0::hcomplex) \Longrightarrow (numeral\ w::hcomplex) \notin Infinitesimal$
by (*fast dest*: *Standard-numeral* [*THEN SComplex-Infinitesimal-zero*])

lemma *approx-SComplex-not-zero*:
$[\![y \in SComplex; x \approx y; y\neq 0]\!] \Longrightarrow x \neq 0$
by (*auto dest*: *SComplex-Infinitesimal-zero approx-sym* [*THEN mem-infmal-iff* [*THEN iffD2*]])

lemma *SComplex-approx-iff*:
$[\![x \in SComplex; y \in SComplex]\!] \Longrightarrow (x \approx y) = (x = y)$
by (*auto simp add*: *Standard-def*)

lemma *approx-unique-complex*:
$[\![r \in SComplex; s \in SComplex; r \approx x; s \approx x]\!] \Longrightarrow r = s$
by (*blast intro*: *SComplex-approx-iff* [*THEN iffD1*] *approx-trans2*)

15.6 Properties of *hRe*, *hIm* and *HComplex*

lemma *abs-hRe-le-hcmod*: $\bigwedge x.\ |hRe\ x| \leq hcmod\ x$
by *transfer* (*rule abs-Re-le-cmod*)

lemma *abs-hIm-le-hcmod*: $\bigwedge x.\ |hIm\ x| \leq hcmod\ x$
by *transfer* (*rule abs-Im-le-cmod*)

lemma *Infinitesimal-hRe*: $x \in Infinitesimal \Longrightarrow hRe\ x \in Infinitesimal$
using *Infinitesimal-hcmod-iff abs-hRe-le-hcmod hrabs-le-Infinitesimal* **by** *blast*

lemma *Infinitesimal-hIm*: $x \in Infinitesimal \Longrightarrow hIm\ x \in Infinitesimal$
using *Infinitesimal-hcmod-iff abs-hIm-le-hcmod hrabs-le-Infinitesimal* **by** *blast*

lemma *Infinitesimal-HComplex*:
assumes *x*: $x \in Infinitesimal$ **and** *y*: $y \in Infinitesimal$
shows $HComplex\ x\ y \in Infinitesimal$
proof –

have *hcmod* (*HComplex 0 y*) ∈ *Infinitesimal*
by (*simp add*: *hcmod-i y*)
moreover have *hcmod* (*hcomplex-of-hypreal x*) ∈ *Infinitesimal*
using *Infinitesimal-hcmod-iff Infinitesimal-of-hypreal-iff x* **by** *blast*
ultimately have *hcmod* (*HComplex x y*) ∈ *Infinitesimal*
by (*metis Infinitesimal-add Infinitesimal-hcmod-iff add.right-neutral hcomplex-of-hypreal-add-HComplex*)
then show *?thesis*
by (*simp add*: *Infinitesimal-hnorm-iff*)
qed

lemma *hcomplex-Infinitesimal-iff*:
(*x* ∈ *Infinitesimal*) ⟷ (*hRe x* ∈ *Infinitesimal* ∧ *hIm x* ∈ *Infinitesimal*)
using *Infinitesimal-HComplex Infinitesimal-hIm Infinitesimal-hRe* **by** *fastforce*

lemma *hRe-diff* [*simp*]: ⋀*x y*. *hRe* (*x* − *y*) = *hRe x* − *hRe y*
by *transfer simp*

lemma *hIm-diff* [*simp*]: ⋀*x y*. *hIm* (*x* − *y*) = *hIm x* − *hIm y*
by *transfer simp*

lemma *approx-hRe*: *x* ≈ *y* ⟹ *hRe x* ≈ *hRe y*
unfolding *approx-def* **by** (*drule Infinitesimal-hRe*) *simp*

lemma *approx-hIm*: *x* ≈ *y* ⟹ *hIm x* ≈ *hIm y*
unfolding *approx-def* **by** (*drule Infinitesimal-hIm*) *simp*

lemma *approx-HComplex*:
⟦*a* ≈ *b*; *c* ≈ *d*⟧ ⟹ *HComplex a c* ≈ *HComplex b d*
unfolding *approx-def* **by** (*simp add*: *Infinitesimal-HComplex*)

lemma *hcomplex-approx-iff*:
(*x* ≈ *y*) = (*hRe x* ≈ *hRe y* ∧ *hIm x* ≈ *hIm y*)
unfolding *approx-def* **by** (*simp add*: *hcomplex-Infinitesimal-iff*)

lemma *HFinite-hRe*: *x* ∈ *HFinite* ⟹ *hRe x* ∈ *HFinite*
using *HFinite-bounded-hcmod abs-ge-zero abs-hRe-le-hcmod* **by** *blast*

lemma *HFinite-hIm*: *x* ∈ *HFinite* ⟹ *hIm x* ∈ *HFinite*
using *HFinite-bounded-hcmod abs-ge-zero abs-hIm-le-hcmod* **by** *blast*

lemma *HFinite-HComplex*:
assumes *x* ∈ *HFinite y* ∈ *HFinite*
shows *HComplex x y* ∈ *HFinite*
proof −
have *HComplex x 0* ∈ *HFinite HComplex 0 y* ∈ *HFinite*
using *HFinite-hcmod-iff assms hcmod-i* **by** *fastforce+*
then have *HComplex x 0* + *HComplex 0 y* ∈ *HFinite*
using *HFinite-add* **by** *blast*

then show *?thesis*
by *simp*
qed

lemma *hcomplex-HFinite-iff*:
($x \in HFinite$) = ($hRe\ x \in HFinite \wedge hIm\ x \in HFinite$)
using *HFinite-HComplex HFinite-hIm HFinite-hRe* **by** *fastforce*

lemma *hcomplex-HInfinite-iff*:
($x \in HInfinite$) = ($hRe\ x \in HInfinite \vee hIm\ x \in HInfinite$)
by (*simp add*: *HInfinite-HFinite-iff hcomplex-HFinite-iff*)

lemma *hcomplex-of-hypreal-approx-iff* [*simp*]:
(*hcomplex-of-hypreal* $x \approx$ *hcomplex-of-hypreal* z) = ($x \approx z$)
by (*simp add*: *hcomplex-approx-iff*)

lemma *stc-part-Ex*:
assumes $x \in HFinite$
shows $\exists\, t \in SComplex.\ x \approx t$
proof −
let $?t$ = *HComplex* (*st* (*hRe* x)) (*st* (*hIm* x))
have $?t \in SComplex$
using *HFinite-hIm HFinite-hRe Reals-eq-Standard assms st-SReal* **by** *auto*
moreover have $x \approx ?t$
by (*simp add*: *HFinite-hIm HFinite-hRe assms hcomplex-approx-iff st-HFinite st-eq-approx*)
ultimately show *?thesis* **..**
qed

lemma *stc-part-Ex1*: $x \in HFinite \Longrightarrow \exists!t.\ t \in SComplex \wedge x \approx t$
using *approx-sym approx-unique-complex stc-part-Ex* **by** *blast*

15.7 Theorems About Monads

lemma *monad-zero-hcmod-iff*: ($x \in$ *monad 0*) = (*hcmod* $x \in$ *monad 0*)
by (*simp add*: *Infinitesimal-monad-zero-iff* [*symmetric*] *Infinitesimal-hcmod-iff*)

15.8 Theorems About Standard Part

lemma *stc-approx-self*: $x \in HFinite \Longrightarrow stc\ x \approx x$
unfolding *stc-def*
by (*metis* (*no-types*, *lifting*) *approx-reorient someI-ex stc-part-Ex1*)

lemma *stc-SComplex*: $x \in HFinite \Longrightarrow stc\ x \in SComplex$
unfolding *stc-def*
by (*metis* (*no-types*, *lifting*) *SComplex-iff approx-sym someI-ex stc-part-Ex*)

lemma *stc-HFinite*: $x \in HFinite \Longrightarrow stc\ x \in HFinite$
by (*erule stc-SComplex* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])

lemma *stc-unique*: $[\![y \in SComplex;\ y \approx x]\!] \Longrightarrow stc\ x = y$
by (*metis SComplex-approx-iff SComplex-iff approx-monad-iff approx-star-of-HFinite stc-SComplex stc-approx-self*)

lemma *stc-SComplex-eq* [*simp*]: $x \in SComplex \Longrightarrow stc\ x = x$
by (*simp add*: *stc-unique*)

lemma *stc-eq-approx*:
$[\![x \in HFinite;\ y \in HFinite;\ stc\ x = stc\ y]\!] \Longrightarrow x \approx y$
by (*auto dest*!: *stc-approx-self elim*!: *approx-trans3*)

lemma *approx-stc-eq*:
$[\![x \in HFinite;\ y \in HFinite;\ x \approx y]\!] \Longrightarrow stc\ x = stc\ y$
by (*metis approx-sym approx-trans3 stc-part-Ex1 stc-unique*)

lemma *stc-eq-approx-iff*:
$[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow (x \approx y) = (stc\ x = stc\ y)$
by (*blast intro*: *approx-stc-eq stc-eq-approx*)

lemma *stc-Infinitesimal-add-SComplex*:
$[\![x \in SComplex;\ e \in Infinitesimal]\!] \Longrightarrow stc(x + e) = x$
using *Infinitesimal-add-approx-self stc-unique* **by** *blast*

lemma *stc-Infinitesimal-add-SComplex2*:
$[\![x \in SComplex;\ e \in Infinitesimal]\!] \Longrightarrow stc(e + x) = x$
using *Infinitesimal-add-approx-self2 stc-unique* **by** *blast*

lemma *HFinite-stc-Infinitesimal-add*:
$x \in HFinite \Longrightarrow \exists\, e \in Infinitesimal.\ x = stc(x) + e$
by (*blast dest*!: *stc-approx-self* [*THEN approx-sym*] *bex-Infinitesimal-iff2* [*THEN iffD2*])

lemma *stc-add*:
$[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow stc\ (x + y) = stc(x) + stc(y)$
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-add*)

lemma *stc-zero*: $stc\ 0 = 0$
by *simp*

lemma *stc-one*: $stc\ 1 = 1$
by *simp*

lemma *stc-minus*: $y \in HFinite \Longrightarrow stc(-y) = -stc(y)$
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-minus*)

lemma *stc-diff*:
$[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow stc\ (x-y) = stc(x) - stc(y)$
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-diff*)

lemma *stc-mult*:
$[\![x \in HFinite;\ y \in HFinite]\!]$
$\Longrightarrow stc\ (x * y) = stc(x) * stc(y)$
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-mult-HFinite*)

lemma *stc-Infinitesimal*: $x \in Infinitesimal \Longrightarrow stc\ x = 0$
by (*simp add*: *stc-unique mem-infmal-iff*)

lemma *stc-not-Infinitesimal*: $stc(x) \neq 0 \Longrightarrow x \notin Infinitesimal$
by (*fast intro*: *stc-Infinitesimal*)

lemma *stc-inverse*:
$[\![x \in HFinite;\ stc\ x \neq 0]\!] \Longrightarrow stc(inverse\ x) = inverse\ (stc\ x)$
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-inverse stc-not-Infinitesimal*)

lemma *stc-divide* [*simp*]:
$[\![x \in HFinite;\ y \in HFinite;\ stc\ y \neq 0]\!]$
$\Longrightarrow stc(x/y) = (stc\ x)\ /\ (stc\ y)$
by (*simp add*: *divide-inverse stc-mult stc-not-Infinitesimal HFinite-inverse stc-inverse*)

lemma *stc-idempotent* [*simp*]: $x \in HFinite \Longrightarrow stc(stc(x)) = stc(x)$
by (*blast intro*: *stc-HFinite stc-approx-self approx-stc-eq*)

lemma *HFinite-HFinite-hcomplex-of-hypreal*:
$z \in HFinite \Longrightarrow hcomplex\text{-}of\text{-}hypreal\ z \in HFinite$
by (*simp add*: *hcomplex-HFinite-iff*)

lemma *SComplex-SReal-hcomplex-of-hypreal*:
$x \in \mathbb{R} \Longrightarrow hcomplex\text{-}of\text{-}hypreal\ x \in SComplex$
by (*simp add*: *Reals-eq-Standard*)

lemma *stc-hcomplex-of-hypreal*:
$z \in HFinite \Longrightarrow stc(hcomplex\text{-}of\text{-}hypreal\ z) = hcomplex\text{-}of\text{-}hypreal\ (st\ z)$
by (*simp add*: *SComplex-SReal-hcomplex-of-hypreal st-SReal st-approx-self stc-unique*)

lemma *hmod-stc-eq*:
assumes $x \in HFinite$
shows $hcmod(stc\ x) = st(hcmod\ x)$
by (*metis SReal-hcmod-SComplex approx-HFinite approx-hnorm assms st-unique stc-SComplex-eq stc-eq-approx-iff stc-part-Ex*)

lemma *Infinitesimal-hcnj-iff* [*simp*]:
$(hcnj\ z \in Infinitesimal) \longleftrightarrow (z \in Infinitesimal)$
by (*simp add*: *Infinitesimal-hcmod-iff*)

end

16 Star-transforms in NSA, Extending Sets of Complex Numbers and Complex Functions

theory *CStar*
imports *NSCA*
begin

16.1 Properties of the $*$-Transform Applied to Sets of Reals

lemma *STARC-hcomplex-of-complex-Int*: $*s*\ X \cap SComplex = hcomplex\text{-}of\text{-}complex$ ‘ X
by (*auto simp*: *Standard-def*)

lemma *lemma-not-hcomplexA*: $x \notin hcomplex\text{-}of\text{-}complex$ ‘ $A \Longrightarrow \forall y \in A.\ x \neq hcomplex\text{-}of\text{-}complex\ y$
by *auto*

16.2 Theorems about Nonstandard Extensions of Functions

lemma *starfunC-hcpow*: $\bigwedge Z.\ (\ *f*\ (\lambda z.\ z \frown n))\ Z = Z\ pow\ hypnat\text{-}of\text{-}nat\ n$
by *transfer* (*rule refl*)

lemma *starfunCR-cmod*: $*f*\ cmod = hcmod$
by *transfer* (*rule refl*)

16.3 Internal Functions - Some Redundancy With $*f*$ Now

lemma *starfun-Re*: $(\ *f*\ (\lambda x.\ Re\ (f\ x))) = (\lambda x.\ hRe\ ((\ *f*\ f)\ x))$
by *transfer* (*rule refl*)

lemma *starfun-Im*: $(\ *f*\ (\lambda x.\ Im\ (f\ x))) = (\lambda x.\ hIm\ ((\ *f*\ f)\ x))$
by *transfer* (*rule refl*)

lemma *starfunC-eq-Re-Im-iff*:
$(\ *f*\ f)\ x = z \longleftrightarrow (\ *f*\ (\lambda x.\ Re\ (f\ x)))\ x = hRe\ z \wedge (\ *f*\ (\lambda x.\ Im\ (f\ x)))\ x = hIm\ z$
by (*simp add*: *hcomplex-hRe-hIm-cancel-iff starfun-Re starfun-Im*)

lemma *starfunC-approx-Re-Im-iff*:
$(\ *f*\ f)\ x \approx z \longleftrightarrow (\ *f*\ (\lambda x.\ Re\ (f\ x)))\ x \approx hRe\ z \wedge (\ *f*\ (\lambda x.\ Im\ (f\ x)))\ x \approx hIm\ z$
by (*simp add*: *hcomplex-approx-iff starfun-Re starfun-Im*)

end

17 Limits, Continuity and Differentiation for Complex Functions

theory *CLim*

imports *CStar*
begin

declare *epsilon-not-zero* [*simp*]

lemma *lemma-complex-mult-inverse-squared* [*simp*]: $x \neq 0 \Longrightarrow x * (inverse\ x)^2 = inverse\ x$
for x :: *complex*
by (*simp add*: *numeral-2-eq-2*)

Changing the quantified variable. Install earlier?

lemma *all-shift*: $(\forall x::'a::comm\text{-}ring\text{-}1.\ P\ x) \longleftrightarrow (\forall x.\ P\ (x - a))$
by (*metis add-diff-cancel*)

17.1 Limit of Complex to Complex Function

lemma *NSLIM-Re*: $f\ -a\rightarrow_{NS}\ L \Longrightarrow (\lambda x.\ Re\ (f\ x))\ -a\rightarrow_{NS}\ Re\ L$
by (*simp add*: *NSLIM-def starfunC-approx-Re-Im-iff hRe-hcomplex-of-complex*)

lemma *NSLIM-Im*: $f\ -a\rightarrow_{NS}\ L \Longrightarrow (\lambda x.\ Im\ (f\ x))\ -a\rightarrow_{NS}\ Im\ L$
by (*simp add*: *NSLIM-def starfunC-approx-Re-Im-iff hIm-hcomplex-of-complex*)

lemma *LIM-Re*: $f\ -a\rightarrow\ L \Longrightarrow (\lambda x.\ Re\ (f\ x))\ -a\rightarrow\ Re\ L$
for f :: *'a::real-normed-vector* ⇒ *complex*
by (*simp add*: *LIM-NSLIM-iff NSLIM-Re*)

lemma *LIM-Im*: $f\ -a\rightarrow\ L \Longrightarrow (\lambda x.\ Im\ (f\ x))\ -a\rightarrow\ Im\ L$
for f :: *'a::real-normed-vector* ⇒ *complex*
by (*simp add*: *LIM-NSLIM-iff NSLIM-Im*)

lemma *LIM-cnj*: $f\ -a\rightarrow\ L \Longrightarrow (\lambda x.\ cnj\ (f\ x))\ -a\rightarrow\ cnj\ L$
for f :: *'a::real-normed-vector* ⇒ *complex*
by (*simp add*: *LIM-eq complex-cnj-diff* [*symmetric*] *del*: *complex-cnj-diff*)

lemma *LIM-cnj-iff*: $((\lambda x.\ cnj\ (f\ x))\ -a\rightarrow\ cnj\ L) \longleftrightarrow f\ -a\rightarrow\ L$
for f :: *'a::real-normed-vector* ⇒ *complex*
by (*simp add*: *LIM-eq complex-cnj-diff* [*symmetric*] *del*: *complex-cnj-diff*)

lemma *starfun-norm*: $(\ *f*\ (\lambda x.\ norm\ (f\ x))) = (\lambda x.\ hnorm\ ((\ *f*\ f)\ x))$
by *transfer* (*rule refl*)

lemma *star-of-Re* [*simp*]: *star-of* (*Re x*) = *hRe* (*star-of x*)
by *transfer* (*rule refl*)

lemma *star-of-Im* [*simp*]: *star-of* (*Im x*) = *hIm* (*star-of x*)
by *transfer* (*rule refl*)

Another equivalence result.

lemma *NSCLIM-NSCRLIM-iff*: $f\ -x\rightarrow_{NS}\ L \longleftrightarrow (\lambda y.\ cmod\ (f\ y - L))\ -x\rightarrow_{NS}\ 0$
  by (*simp add*: *NSLIM-def starfun-norm*
    approx-approx-zero-iff [*symmetric*] *approx-minus-iff* [*symmetric*])

Much, much easier standard proof.

lemma *CLIM-CRLIM-iff*: $f\ -x\rightarrow\ L \longleftrightarrow (\lambda y.\ cmod\ (f\ y - L))\ -x\rightarrow\ 0$
  for f :: *'a::real-normed-vector* ⇒ *complex*
  by (*simp add*: *LIM-eq*)

So this is nicer nonstandard proof.

lemma *NSCLIM-NSCRLIM-iff2*: $f\ -x\rightarrow_{NS}\ L \longleftrightarrow (\lambda y.\ cmod\ (f\ y - L))\ -x\rightarrow_{NS}\ 0$
  by (*simp add*: *LIM-NSLIM-iff* [*symmetric*] *CLIM-CRLIM-iff*)

lemma *NSLIM-NSCRLIM-Re-Im-iff*:
  $f\ -a\rightarrow_{NS}\ L \longleftrightarrow (\lambda x.\ Re\ (f\ x))\ -a\rightarrow_{NS}\ Re\ L \wedge (\lambda x.\ Im\ (f\ x))\ -a\rightarrow_{NS}\ Im\ L$
  apply (*auto intro*: *NSLIM-Re NSLIM-Im*)
  apply (*auto simp add*: *NSLIM-def starfun-Re starfun-Im*)
  apply (*auto dest*!: *spec*)
  apply (*simp add*: *hcomplex-approx-iff*)
  done

lemma *LIM-CRLIM-Re-Im-iff*: $f\ -a\rightarrow\ L \longleftrightarrow (\lambda x.\ Re\ (f\ x))\ -a\rightarrow\ Re\ L \wedge (\lambda x.\ Im\ (f\ x))\ -a\rightarrow\ Im\ L$
  for f :: *'a::real-normed-vector* ⇒ *complex*
  by (*simp add*: *LIM-NSLIM-iff NSLIM-NSCRLIM-Re-Im-iff*)

17.2 Continuity

lemma *NSLIM-isContc-iff*: $f\ -a\rightarrow_{NS}\ f\ a \longleftrightarrow (\lambda h.\ f\ (a + h))\ -0\rightarrow_{NS}\ f\ a$
  by (*rule NSLIM-at0-iff*)

17.3 Functions from Complex to Reals

lemma *isNSContCR-cmod* [*simp*]: *isNSCont cmod a*
  by (*auto intro*: *approx-hnorm*
    simp: *starfunCR-cmod hcmod-hcomplex-of-complex* [*symmetric*] *isNSCont-def*)

lemma *isContCR-cmod* [*simp*]: *isCont cmod a*
  by (*simp add*: *isNSCont-isCont-iff* [*symmetric*])

lemma *isCont-Re*: *isCont f a* ⟹ *isCont* (λ*x*. *Re* (*f x*)) *a*
  for f :: *'a::real-normed-vector* ⇒ *complex*
  by (*simp add*: *isCont-def LIM-Re*)

lemma *isCont-Im*: *isCont f a* ⟹ *isCont* (λ*x*. *Im* (*f x*)) *a*
  for f :: *'a::real-normed-vector* ⇒ *complex*
  by (*simp add*: *isCont-def LIM-Im*)

17.4 Differentiation of Natural Number Powers

lemma *CDERIV-pow* [*simp*]: *DERIV* ($\lambda x.\ x$ ^ n) x :> *complex-of-real* (*real* n) $*$ (x ^ ($n - $ *Suc 0*))
apply (*induct n*)
apply (*drule-tac* [*2*] *DERIV-ident* [*THEN DERIV-mult*])
apply (*auto simp add*: *distrib-right of-nat-Suc*)
apply (*case-tac n*)
apply (*auto simp add*: *ac-simps*)
done

Nonstandard version.

lemma *NSCDERIV-pow*: *NSDERIV* ($\lambda x.\ x$ ^ n) x :> *complex-of-real* (*real* n) $*$ (x ^ ($n - 1$))
by (*metis CDERIV-pow NSDERIV-DERIV-iff One-nat-def*)

Can't relax the premise $x \neq 0$: it isn't continuous at zero.

lemma *NSCDERIV-inverse*: $x \neq 0 \Longrightarrow$ *NSDERIV* ($\lambda x.$ *inverse* x) x :> $-$ (*inverse* $x)^2$
for x :: *complex*
unfolding *numeral-2-eq-2* **by** (*rule NSDERIV-inverse*)

lemma *CDERIV-inverse*: $x \neq 0 \Longrightarrow$ *DERIV* ($\lambda x.$ *inverse* x) x :> $-$ (*inverse* $x)^2$
for x :: *complex*
unfolding *numeral-2-eq-2* **by** (*rule DERIV-inverse*)

17.5 Derivative of Reciprocals (Function *inverse*)

lemma *CDERIV-inverse-fun*:
DERIV $f\,x$:> $d \Longrightarrow f\,x \neq 0 \Longrightarrow$ *DERIV* ($\lambda x.$ *inverse* ($f\,x$)) x :> $-$ ($d\ *$ *inverse* $((f\,x)^2)$)
for x :: *complex*
unfolding *numeral-2-eq-2* **by** (*rule DERIV-inverse-fun*)

lemma *NSCDERIV-inverse-fun*:
NSDERIV $f\,x$:> $d \Longrightarrow f\,x \neq 0 \Longrightarrow$ *NSDERIV* ($\lambda x.$ *inverse* ($f\,x$)) x :> $-$ ($d\ *$ *inverse* $((f\,x)^2)$)
for x :: *complex*
unfolding *numeral-2-eq-2* **by** (*rule NSDERIV-inverse-fun*)

17.6 Derivative of Quotient

lemma *CDERIV-quotient*:
DERIV $f\,x$:> $d \Longrightarrow$ *DERIV* $g\ x$:> $e \Longrightarrow g(x) \neq 0 \Longrightarrow$
DERIV ($\lambda y.\ f\ y\ /\ g\ y$) x :> $(d * g\ x - (e * f\,x))\ /\ (g\ x)^2$
for x :: *complex*
unfolding *numeral-2-eq-2* **by** (*rule DERIV-quotient*)

lemma *NSCDERIV-quotient*:
NSDERIV $f\,x$:> $d \Longrightarrow$ *NSDERIV* $g\ x$:> $e \Longrightarrow g\ x \neq$ (*0::complex*) $\Longrightarrow$

NSDERIV $(\lambda y.\ f\ y\ /\ g\ y)\ x\ :>\ (d * g\ x - (e * f\ x))\ /\ (g\ x)^2$
unfolding *numeral-2-eq-2* **by** (*rule NSDERIV-quotient*)

17.7 Caratheodory Formulation of Derivative at a Point: Standard Proof

lemma *CARAT-CDERIVD*:
$(\forall z.\ f\ z - f\ x = g\ z * (z - x)) \wedge isNSCont\ g\ x \wedge g\ x = l \Longrightarrow NSDERIV\ f\ x :> l$
by *clarify* (*rule CARAT-DERIVD*)

end

18 Logarithms: Non-Standard Version

theory *HLog*
imports *HTranscendental*
begin

definition *powhr* :: *hypreal* ⇒ *hypreal* ⇒ *hypreal* (**infixr** ‹*powhr*› *80*)
where [*transfer-unfold*]: *x powhr a* = *starfun2* (*powr*) *x a*

definition *hlog* :: *hypreal* ⇒ *hypreal* ⇒ *hypreal*
where [*transfer-unfold*]: *hlog a x* = *starfun2 log a x*

lemma *powhr*: (*star-n X*) *powhr* (*star-n Y*) = *star-n* (λ*n*. (*X n*) *powr* (*Y n*))
by (*simp add*: *powhr-def starfun2-star-n*)

lemma *powhr-one-eq-one* [*simp*]: $\bigwedge a.\ 1\ powhr\ a = 1$
by *transfer simp*

lemma *powhr-mult*: $\bigwedge a\ x\ y.\ 0 < x \Longrightarrow 0 < y \Longrightarrow (x * y)\ powhr\ a = (x\ powhr\ a) * (y\ powhr\ a)$
by *transfer* (*simp add*: *powr-mult*)

lemma *powhr-gt-zero* [*simp*]: $\bigwedge a\ x.\ 0 < x\ powhr\ a \longleftrightarrow x \neq 0$
by *transfer simp*

lemma *powhr-not-zero* [*simp*]: $\bigwedge a\ x.\ x\ powhr\ a \neq 0 \longleftrightarrow x \neq 0$
by *transfer simp*

lemma *powhr-divide*: $\bigwedge a\ x\ y.\ 0 \leq x \Longrightarrow 0 \leq y \Longrightarrow (x\ /\ y)\ powhr\ a = (x\ powhr\ a)\ /\ (y\ powhr\ a)$
by *transfer* (*rule powr-divide*)

lemma *powhr-add*: $\bigwedge a\ b\ x.\ x\ powhr\ (a + b) = (x\ powhr\ a) * (x\ powhr\ b)$
by *transfer* (*rule powr-add*)

lemma *powhr-powhr*: $\bigwedge a\ b\ x.\ (x\ powhr\ a)\ powhr\ b = x\ powhr\ (a * b)$
by *transfer* (*rule powr-powr*)

lemma *powhr-powhr-swap*: $\bigwedge a\ b\ x.\ (x\ powhr\ a)\ powhr\ b = (x\ powhr\ b)\ powhr\ a$
by *transfer* (*rule powr-powr-swap*)

lemma *powhr-minus*: $\bigwedge a\ x.\ x\ powhr\ (-\ a) = inverse\ (x\ powhr\ a)$
by *transfer* (*rule powr-minus*)

lemma *powhr-minus-divide*: $x\ powhr\ (-\ a) = 1\ /\ (x\ powhr\ a)$
by (*simp add*: *divide-inverse powhr-minus*)

lemma *powhr-less-mono*: $\bigwedge a\ b\ x.\ a < b \Longrightarrow 1 < x \Longrightarrow x\ powhr\ a < x\ powhr\ b$
by *transfer simp*

lemma *powhr-less-cancel*: $\bigwedge a\ b\ x.\ x\ powhr\ a < x\ powhr\ b \Longrightarrow 1 < x \Longrightarrow a < b$
by *transfer simp*

lemma *powhr-less-cancel-iff* [*simp*]: $1 < x \Longrightarrow x\ powhr\ a < x\ powhr\ b \longleftrightarrow a < b$
by (*blast intro*: *powhr-less-cancel powhr-less-mono*)

lemma *powhr-le-cancel-iff* [*simp*]: $1 < x \Longrightarrow x\ powhr\ a \le x\ powhr\ b \longleftrightarrow a \le b$
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *hlog*: $hlog\ (star\text{-}n\ X)\ (star\text{-}n\ Y) = star\text{-}n\ (\lambda n.\ log\ (X\ n)\ (Y\ n))$
by (*simp add*: *hlog-def starfun2-star-n*)

lemma *hlog-starfun-ln*: $\bigwedge x.\ (\ *f*\ ln)\ x = hlog\ ((\ *f*\ exp)\ 1)\ x$
by *transfer* (*rule log-ln*)

lemma *powhr-hlog-cancel* [*simp*]: $\bigwedge a\ x.\ 0 < a \Longrightarrow a \ne 1 \Longrightarrow 0 < x \Longrightarrow a\ powhr\ (hlog\ a\ x) = x$
by *transfer simp*

lemma *hlog-powhr-cancel* [*simp*]: $\bigwedge a\ y.\ 0 < a \Longrightarrow a \ne 1 \Longrightarrow hlog\ a\ (a\ powhr\ y) = y$
by *transfer simp*

lemma *hlog-mult*:
$\bigwedge a\ x\ y.\ hlog\ a\ (x * y) = (if\ x{\ne}0 \wedge y{\ne}0\ then\ hlog\ a\ x + hlog\ a\ y\ else\ 0)$
by *transfer* (*rule log-mult*)

lemma *hlog-as-starfun*: $\bigwedge a\ x.\ 0 < a \Longrightarrow a \ne 1 \Longrightarrow hlog\ a\ x = (\ *f*\ ln)\ x\ /\ (\ *f*\ ln)\ a$
by *transfer* (*simp add*: *log-def*)

lemma *hlog-eq-div-starfun-ln-mult-hlog*:
$\bigwedge a\ b\ x.\ 0 < a \Longrightarrow a \ne 1 \Longrightarrow 0 < b \Longrightarrow b \ne 1 \Longrightarrow 0 < x \Longrightarrow$
$hlog\ a\ x = ((\ *f*\ ln)\ b\ /\ (\ *f*\ ln)\ a) * hlog\ b\ x$
by *transfer* (*rule log-eq-div-ln-mult-log*)

lemma *powhr-as-starfun*: $\bigwedge a\ x.\ x\ powhr\ a = (if\ x = 0\ then\ 0\ else\ (\ {*}f{*}\ exp)\ (a\ *\ (\ {*}f{*}\ real\text{-}ln)\ x))$
by *transfer* (*simp add*: *powr-def*)

lemma *HInfinite-powhr*:
$x \in HInfinite \Longrightarrow 0 < x \Longrightarrow a \in HFinite - Infinitesimal \Longrightarrow 0 < a \Longrightarrow x\ powhr\ a \in HInfinite$
by (*auto intro*!: *starfun-ln-ge-zero starfun-ln-HInfinite HInfinite-HFinite-not-Infinitesimal-mult2 starfun-exp-HInfinite simp add*: *order-less-imp-le HInfinite-gt-zero-gt-one powhr-as-starfun zero-le-mult-iff*)

lemma *hlog-hrabs-HInfinite-Infinitesimal*:
$x \in HFinite - Infinitesimal \Longrightarrow a \in HInfinite \Longrightarrow 0 < a \Longrightarrow hlog\ a\ |x| \in Infinitesimal$
apply (*frule HInfinite-gt-zero-gt-one*)
apply (*auto intro*!: *starfun-ln-HFinite-not-Infinitesimal HInfinite-inverse-Infinitesimal Infinitesimal-HFinite-mult2 simp add*: *starfun-ln-HInfinite not-Infinitesimal-not-zero hlog-as-starfun divide-inverse*)
done

lemma *hlog-HInfinite-as-starfun*: $a \in HInfinite \Longrightarrow 0 < a \Longrightarrow hlog\ a\ x = (\ {*}f{*}\ ln)\ x\ /\ (\ {*}f{*}\ ln)\ a$
by (*rule hlog-as-starfun*) *auto*

lemma *hlog-one* [*simp*]: $\bigwedge a.\ hlog\ a\ 1 = 0$
by *transfer simp*

lemma *hlog-eq-one* [*simp*]: $\bigwedge a.\ 0 < a \Longrightarrow a \neq 1 \Longrightarrow hlog\ a\ a = 1$
by *transfer* (*rule log-eq-one*)

lemma *hlog-inverse*: $\bigwedge a\ x.\ hlog\ a\ (inverse\ x) = -\ hlog\ a\ x$
by *transfer* (*simp add*: *log-inverse*)

lemma *hlog-divide*: $hlog\ a\ (x\ /\ y) = (if\ x{\neq}0 \land y{\neq}0\ then\ hlog\ a\ x - hlog\ a\ y\ else\ 0)$
by (*simp add*: *hlog-mult hlog-inverse divide-inverse*)

lemma *hlog-less-cancel-iff* [*simp*]:
$\bigwedge a\ x\ y.\ 1 < a \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow hlog\ a\ x < hlog\ a\ y \longleftrightarrow x < y$
by *transfer simp*

lemma *hlog-le-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow hlog\ a\ x \leq hlog\ a\ y \longleftrightarrow x \leq y$
by (*simp add*: *linorder-not-less* [*symmetric*])

end

theory *Hyperreal*
imports *HLog*
begin

end
theory *Hypercomplex*
imports *CLim Hyperreal*
begin

end

theory *Nonstandard-Analysis*
imports *Hypercomplex*
begin

end