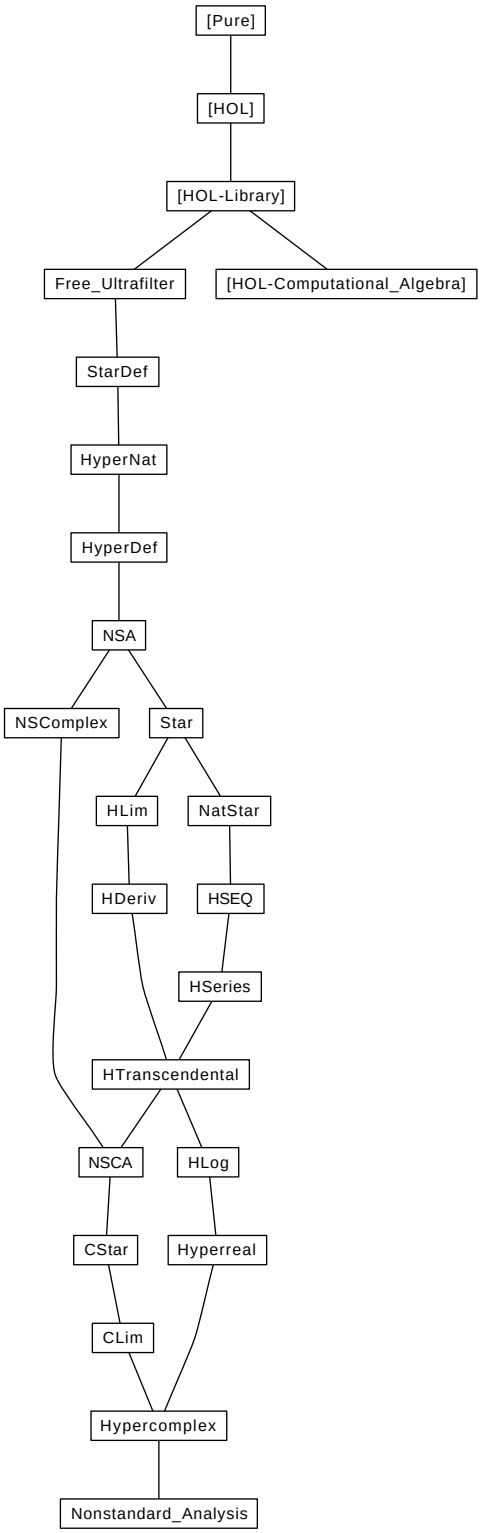
Isabelle/HOL-NSA — Non-Standard Analysis

October 8, 2017

Contents


[Pure]
[HOL]
[HOL-Library]
Free_Ultrafilter
[HOL-Computational_Algebra]
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 $F \vee$ *eventually* $(\lambda x.\ \neg\ P\ x)$ F
begin

lemma *eventually-imp-frequently*: *frequently* P $F \Longrightarrow$ *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* $(\lambda x.\ P\ x \vee Q\ x)$ $F \longleftrightarrow$ *eventually* P $F \vee$ *eventually* Q F
unfolding *frequently-eq-eventually*[*symmetric*] *frequently-disj-iff* **..**

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

lemma *eventually-imp-iff*: *eventually* $(\lambda x.\ P\ x \longrightarrow Q\ x)$ $F \longleftrightarrow$ (*eventually* P $F \longrightarrow$ *eventually* Q F)
using *frequently-imp-iff*[*of* P Q F] **by** (*simp add*: *frequently-eq-eventually*)

lemma *eventually-iff-iff*: *eventually* $(\lambda x.\ P\ x \longleftrightarrow Q\ x)$ $F \longleftrightarrow$ (*eventually* P $F \longleftrightarrow$ *eventually* Q F)
unfolding *iff-conv-conj-imp eventually-conj-iff eventually-imp-iff* **by** *simp*

lemma *eventually-not-iff*: *eventually* $(\lambda x.\ \neg\ P\ x)$ $F \longleftrightarrow \neg$ *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* $\Longrightarrow$ *inf F* (*principal* {*x*. *P x*}) $\neq$ *bot*
by (*simp add*: *trivial-limit-def eventually-inf-principal not-eventually*)

lemma *max-filter-ultrafilter*:
assumes $F \neq bot$
assumes *max*: $\bigwedge G.\ G \neq bot \Longrightarrow G \leq F \Longrightarrow F = G$
shows *ultrafilter F*
proof
show *eventually P F* $\vee$ ($\forall_F x$ *in F*. $\neg$ *P x*) **for** *P*
proof (*rule disjCI*)
assume $\neg$ ($\forall_F x$ *in F*. $\neg$ *P x*)
then have *inf F* (*principal* {*x*. *P x*}) $\neq$ *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* $\leq$ *G* $\longleftrightarrow$ ($\forall$ *P*. *frequently P F* $\longrightarrow$ *frequently P G*)
unfolding *frequently-def le-filter-def*
apply *auto*
apply (*erule-tac x*=$\lambda x.\ \neg$ *P x* **in** *allE*)
apply *auto*
done

lemma (**in** *ultrafilter*) *max-filter*:
assumes *G*: *G* $\neq$ *bot*
and *sub*: *G* $\leq$ *F*
shows *F* = *G*
proof (*rule antisym*)
show *F* $\leq$ *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* $\neq$ *bot*
shows $\exists\, U \leq F.$ *ultrafilter U*
proof −
let *?X* = {*G*. *G* $\neq$ *bot* $\wedge$ *G* $\leq$ *F*}
let *?R* = {(*b*, *a*). *a* $\neq$ *bot* $\wedge$ *a* $\leq$ *b* $\wedge$ *b* $\leq$ *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 ∀ C∈Chains ?R. ∃ u∈Field ?R. ∀ a∈C. (a, u) ∈ ?R
    proof (safe intro!: bexI del: notI)
      fix c x
      assume c: c ∈ Chains ?R

      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) ∈ Field ?R
        by (simp add: Chains-def Inf-c F)

      assume x ∈ c
      with c show inf F (Inf c) ≤ x x ≤ F
        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* (λ*x*. *x* = *a*) *F*
  by (*rule finite*) *simp*

lemma *singleton'*: ¬ *eventually* (*op* = *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* $\mathcal{U}$
apply (*unfold FreeUltrafilterNat-def*)
apply (*rule someI-ex*)
apply (*rule freeultrafilter-Ex*)
apply (*rule infinite-UNIV-nat*)
done

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

2.2 Definition of *star* type constructor

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

definition *star* = (*UNIV* :: (*nat* ⇒ 'a) *set*) // *starrel*

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

definition *star-n* :: (*nat* ⇒ 'a) ⇒ '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*: (∀ *x*. *P x*) ⟷ (∀ *X*. *P* (*star-n X*))
apply *auto*
apply (*rule-tac x* = *x* **in** *star-cases*)
apply *simp*
done

lemma *ex-star-eq*: (∃ *x*. *P x*) ⟷ (∃ *X*. *P* (*star-n X*))
apply *auto*

apply (*rule-tac x=x* **in** *star-cases*)
apply *auto*
done

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 *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``*$\{x\} \in star$
by (*simp add*: *star-def quotientI*)

lemma *star-n-eq-iff*: *star-n X* = *star-n Y* $\longleftrightarrow$ *eventually* $(\lambda n.\ X\ n = Y\ n)\ \mathcal{U}$
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*: $P \equiv eventually\ (\lambda n.\ Q)\ \mathcal{U} \Longrightarrow Trueprop\ P \equiv Trueprop\ Q$
by (*simp add*: *FreeUltrafilterNat.proper*)

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

definition *Ifun* :: $('a \Rightarrow 'b)\ star \Rightarrow 'a\ star \Rightarrow 'b\ star$ ((- ⋆/ -) [*300*, *301*] *300*)
where *Ifun f* ≡
$\lambda x.$ *Abs-star* $(\bigcup F \in$ *Rep-star f*$.\ \bigcup X \in$ *Rep-star x*$.$ *starrel``*$\{\lambda n.\ F\ n\ (X\ n)\})$

lemma *Ifun-congruent2*: *congruent2 starrel starrel* $(\lambda F\ X.$ *starrel``*$\{\lambda n.\ F\ n\ (X\ n)\})$
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* $(\lambda n.\ F\ n\ (X\ n))$
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*: $f \equiv$ *star-n F* $\Longrightarrow x \equiv$ *star-n X* $\Longrightarrow f \star x \equiv$ *star-n* $(\lambda n.\ F\ n\ (X\ n))$
by (*simp only*: *Ifun-star-n*)

definition *star-of* :: $'a \Rightarrow 'a\ star$
where *star-of x* ≡ *star-n* $(\lambda n.\ x)$

Initialize transfer tactic.

ML-file *transfer.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*]:
(⋀*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*: *all-star-eq FreeUltrafilterNat.eventually-all-iff*)

lemma *transfer-not* [*transfer-intro*]: *p* ≡ *eventually P* 𝒰 ⟹ ¬ *p* ≡ *eventually* (λ*n*. ¬ *P n*) 𝒰
by (*simp only*: *FreeUltrafilterNat.eventually-not-iff*)

lemma *transfer-conj* [*transfer-intro*]:
p ≡ *eventually P* 𝒰 ⟹ *q* ≡ *eventually Q* 𝒰 ⟹ *p* ∧ *q* ≡ *eventually* (λ*n*. *P n* ∧ *Q n*) 𝒰
by (*simp only*: *eventually-conj-iff*)

lemma *transfer-disj* [*transfer-intro*]:
p ≡ *eventually P* 𝒰 ⟹ *q* ≡ *eventually Q* 𝒰 ⟹ *p* ∨ *q* ≡ *eventually* (λ*n*. *P n* ∨ *Q n*) 𝒰
by (*simp only*: *FreeUltrafilterNat.eventually-disj-iff*)

lemma *transfer-imp* [*transfer-intro*]:
p ≡ *eventually P* 𝒰 ⟹ *q* ≡ *eventually Q* 𝒰 ⟹ *p* ⟶ *q* ≡ *eventually* (λ*n*. *P n* ⟶ *Q n*) 𝒰
by (*simp only*: *FreeUltrafilterNat.eventually-imp-iff*)

lemma *transfer-iff* [*transfer-intro*]:
p ≡ *eventually P* 𝒰 ⟹ *q* ≡ *eventually Q* 𝒰 ⟹ *p* = *q* ≡ *eventually* (λ*n*. *P n* = *Q n*) 𝒰
by (*simp only*: *FreeUltrafilterNat.eventually-iff-iff*)

lemma *transfer-if-bool* [*transfer-intro*]:
p ≡ *eventually P* 𝒰 ⟹ *x* ≡ *eventually X* 𝒰 ⟹ *y* ≡ *eventually Y* 𝒰 ⟹
(*if p then x else y*) ≡ *eventually* (λ*n*. *if P n then X n else Y n*) 𝒰
by (*simp only*: *if-bool-eq-conj transfer-conj transfer-imp transfer-not*)

lemma *transfer-eq* [*transfer-intro*]:

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

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

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

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

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

2.4 Standard elements

definition $\mathit{Standard} :: {'a}\ \mathit{star}\ \mathit{set}$
where $\mathit{Standard} = \mathit{range}\ \mathit{star\text{-}of}$

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

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

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

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

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

Nonstandard extensions of functions.

definition $\mathit{starfun} :: ({'a} \Rightarrow {'b}) \Rightarrow {'a}\ \mathit{star} \Rightarrow {'b}\ \mathit{star}\ \ (*f*\ \text{-}\ [80]\ 80)$
where $\mathit{starfun}\ f \equiv \lambda x.\ \mathit{star\text{-}of}\ f \star x$

definition *starfun2* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a\ star \Rightarrow 'b\ star \Rightarrow 'c\ star$ $(*f2*\ -\ [80]\ 80)$
  where *starfun2 f* $\equiv \lambda x\ y.$ *star-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-of c*
unfolding *Standard-def* **..**
then have $\exists\, x\ y.$ *starfun2 f x y* = *star-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-of a*) (*star-of b*) = *star-of c* **by** *transfer*
with *c* **have** *starfun2 f x y* = *starfun2 f* (*star-of a*) (*star-of b*) **by** *simp*
then have *x* = *star-of a* $\wedge$ *y* = *star-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* $\Rightarrow$ *bool*
where *unstar b* $\longleftrightarrow$ *b* = *star-of True*

lemma *unstar-star-n*: *unstar* (*star-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-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-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* (**p** - [*80*] *80*)
where **p* P* = ($\lambda x.$ *unstar* (*star-of P* $\star$ *x*))

definition *starP2* :: ('*a* $\Rightarrow$ '*b* $\Rightarrow$ *bool*) $\Rightarrow$ '*a star* $\Rightarrow$ '*b star* $\Rightarrow$ *bool* (**p2** - [*80*] *80*)
where **p2* P* = ($\lambda x\ y.$ *unstar* (*star-of P* $\star$ *x* $\star$ *y*))

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

lemma *starP-star-n*: ($*p*$ *P*) (*star-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-n X*) (*star-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-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*. ($*p2*$ *op* ∈) *x A*}

lemma *Iset-star-n*: (*star-n X* ∈ *Iset* (*star-n A*)) = (*eventually* ($\lambda n.$ *X n* ∈ *A n*) $\mathcal{U}$)
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* ($\lambda n.$ *X n* ∈ *A n*) $\mathcal{U}$
by (*simp only*: *Iset-star-n*)

lemma *transfer-Collect* [*transfer-intro*]:
(⋀*X*. *p* (*star-n X*) ≡ *eventually* ($\lambda n.$ *P n* (*X n*)) $\mathcal{U}$) ⟹
Collect p ≡ *Iset* (*star-n* ($\lambda n.$ *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* ($\lambda n.$ *A n* = *B n*) $\mathcal{U}$
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* ($\lambda n.$ *P n* (*X n*)) $\mathcal{U}$) ⟹
∀ *x*∈*a*. *p x* ≡ *eventually* ($\lambda n.$ ∀ *x*∈*A n*. *P n x*) $\mathcal{U}$
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* ($\lambda n.$ *P n* (*X n*)) $\mathcal{U}$) ⟹
∃ *x*∈*a*. *p x* ≡ *eventually* ($\lambda n.$ ∃ *x*∈*A n*. *P n x*) $\mathcal{U}$
by (*simp only*: *Bex-def transfer-ex transfer-conj transfer-mem*)

lemma *transfer-Iset* [*transfer-intro*]: $a \equiv \mathit{star\text{-}n}\ A \Longrightarrow \mathit{Iset}\ a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ (\lambda n.\ A\ n))$
by *simp*

Nonstandard extensions of sets.

definition *starset* :: *'a set* ⇒ *'a star set* ($*s*$ - [*80*] *80*)
where $\mathit{starset}\ A = \mathit{Iset}\ (\mathit{star\text{-}of}\ A)$

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

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

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

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

lemma *starset-insert*: $*s*\ (\mathit{insert}\ x\ A) = \mathit{insert}\ (\mathit{star\text{-}of}\ x)\ (*s*\ A)$
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*: (*op +*) ≡ ∗*f2*∗ (*op +*)
instance ..
end

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

instantiation *star* :: (*ord*) *ord*
begin
  definition *star-le-def*: (*op* ≤) ≡ ∗*p2*∗ (*op* ≤)
  definition *star-less-def*: (*op* <) ≡ ∗*p2*∗ (*op* <)
  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*: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow x - y \in Standard$
by (*simp add*: *star-diff-def*)

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

lemma *Standard-mult*: $x \in Standard \Longrightarrow y \in Standard \Longrightarrow x * y \in Standard$
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*: $star\text{-}of\ (x + y) = star\text{-}of\ x + star\text{-}of\ y$
by *transfer* (*rule refl*)

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

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

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

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

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

lemma *star-of-mod*: $star\text{-}of\ (x\ mod\ y) = star\text{-}of\ x\ mod\ star\text{-}of\ y$

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* $x \leq$ *star-of* $y) = (x \leq y)$
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*
apply *intro-classes*
apply (*transfer*, *rule less-le-not-le*)
apply (*transfer*, *rule order-refl*)
apply (*transfer*, *erule* (*1*) *order-trans*)
apply (*transfer*, *erule* (*1*) *order-antisym*)
done

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

lemma *Standard-sup* [*simp*]: *x* ∈ *Standard* ⟹ *y* ∈ *Standard* ⟹ *sup x y* ∈ *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*
apply (*rule ext*, *rule ext*)

apply (*unfold max-def*, *transfer*, *fold max-def*)
apply (*rule refl*)
done

lemma *star-min-def* [*transfer-unfold*]: *min* = *∗f2∗ min*
apply (*rule ext*, *rule ext*)
apply (*unfold min-def*, *transfer*, *fold min-def*)
apply (*rule refl*)
done

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

lemma *Standard-min* [*simp*]: *x* ∈ *Standard* ⟹ *y* ∈ *Standard* ⟹ *min x y* ∈ *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* :: (*semiring-div*) *semiring-div*
by (*intro-classes*; *transfer*) (*simp-all add*: *div-mult-mod-eq*)

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* :: (*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* **..**

2.12 Power

lemma *star-power-def* [*transfer-unfold*]: $(op\ \hat{}) \equiv \lambda x\ n.\ (\ *f*\ (\lambda x.\ x\ \hat{}\ n))\ x$
proof (*rule eq-reflection*, *rule ext*, *rule ext*)
show $x\ \hat{}\ n = (\ *f*\ (\lambda x.\ x\ \hat{}\ n))\ x$ **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 $\bigwedge x::'a\ star.\ x * (\ *f*\ (\lambda x::'a.\ x\ \hat{}\ n))\ x = (\ *f*\ (\lambda x::'a.\ x * x\ \hat{}\ n))\ x$
by *transfer simp*
with *Suc* **show** *?case* **by** *simp*
qed
qed

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

lemma *star-of-power* [*simp*]: *star-of* $(x\ \hat{}\ n) =$ *star-of* $x\ \hat{}\ n$
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* $k \in Standard$
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* $n \in Standard$
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-comp*)
then show *inj* (*of-nat* :: *nat ⇒ 'a star*)
by (*simp add*: *comp-def*)
qed

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

instance *star* :: (*semiring-parity*) *semiring-parity*
apply *intro-classes*
apply (*transfer*, *rule odd-one*)
apply (*transfer*, *erule* (*1*) *odd-even-add*)
apply (*transfer*, *erule even-multD*)
apply (*transfer*, *erule odd-ex-decrement*)
done

instance *star* :: (*semiring-div-parity*) *semiring-div-parity*
apply *intro-classes*
apply (*transfer*, *rule parity*)
apply (*transfer*, *rule one-mod-two-eq-one*)
apply (*transfer*, *rule zero-not-eq-two*)
done

instantiation *star* :: (*semiring-numeral-div*) *semiring-numeral-div*
begin

definition *divmod-star* :: *num ⇒ num ⇒ 'a star × 'a star*
where *divmod-star-def*: *divmod-star m n* = (*numeral m div numeral n*, *numeral m mod numeral n*)

definition *divmod-step-star* :: *num ⇒ 'a star × 'a star ⇒ 'a star × 'a star*
where *divmod-step-star l qr* =
(*let* (*q*, *r*) = *qr*
in if r ≥ numeral l then (*2 ∗ q + 1*, *r − numeral l*) *else* (*2 ∗ q*, *r*))

instance

proof
show *divmod m n* = (*numeral m div numeral n* :: *'a star*, *numeral m mod numeral n*) **for** *m n*
by (*fact divmod-star-def*)
show *divmod-step l qr* =
(*let* (*q*, *r*) = *qr*
in if r ≥ *numeral l then* (2 ∗ *q* + 1, *r* − *numeral l*) *else* (2 ∗ *q*, *r*))
for *l* **and** *qr* :: *'a star* × *'a star*
by (*fact divmod-step-star-def*)
qed (*transfer*,
fact
semiring-numeral-div-class.div-less
semiring-numeral-div-class.mod-less
semiring-numeral-div-class.div-positive
semiring-numeral-div-class.mod-less-eq-dividend
semiring-numeral-div-class.pos-mod-bound
semiring-numeral-div-class.pos-mod-sign
semiring-numeral-div-class.mod-mult2-eq
semiring-numeral-div-class.div-mult2-eq
semiring-numeral-div-class.discrete)+

end

declare *divmod-algorithm-code* [**where** *?'a* = *'a*::*semiring-numeral-div star*, *code*]

2.14 Finite class

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

instance *star* :: (*finite*) *finite*
apply *intro-classes*
apply (*subst starset-UNIV* [*symmetric*])
apply (*subst starset-finite* [*OF finite*])
apply (*rule finite-imageI* [*OF finite*])
done

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*]: ⋀*m*. *hSuc m* ≠ *0*
by *transfer* (*rule Suc-not-Zero*)

lemma *zero-not-hSuc* [*iff*]: ⋀*m*. *0* ≠ *hSuc m*
by *transfer* (*rule Zero-not-Suc*)

lemma *hSuc-hSuc-eq* [*iff*]: ⋀*m n*. *hSuc m* = *hSuc n* ⟷ *m* = *n*
by *transfer* (*rule nat.inject*)

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

lemma *hypnat-minus-zero* [*simp*]: ⋀*z*::*hypnat*. *z* − *z* = *0*
by *transfer* (*rule diff-self-eq-0*)

lemma *hypnat-diff-0-eq-0* [*simp*]: ⋀*n*::*hypnat*. *0* − *n* = *0*
by *transfer* (*rule diff-0-eq-0*)

lemma *hypnat-add-is-0* [*iff*]: ⋀*m n*::*hypnat*. *m* + *n* = *0* ⟷ *m* = *0* ∧ *n* = *0*
by *transfer* (*rule add-is-0*)

lemma *hypnat-diff-diff-left*: ⋀*i j k*::*hypnat*. *i* − *j* − *k* = *i* − (*j* + *k*)
by *transfer* (*rule diff-diff-left*)

lemma *hypnat-diff-commute*: ⋀*i j k*::*hypnat*. *i* − *j* − *k* = *i* − *k* − *j*
by *transfer* (*rule diff-commute*)

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

lemma *hypnat-diff-add-inverse2* [*simp*]: ⋀*m n*::*hypnat*. *m* + *n* − *n* = *m*
by *transfer* (*rule diff-add-inverse2*)

lemma *hypnat-diff-cancel* [*simp*]: ⋀*k m n*::*hypnat*. (*k* + *m*) − (*k* + *n*) = *m* − *n*
by *transfer* (*rule diff-cancel*)

lemma *hypnat-diff-cancel2* [*simp*]: ⋀*k m n*::*hypnat*. (*m* + *k*) − (*n* + *k*) = *m* − *n*
by *transfer* (*rule diff-cancel2*)

lemma *hypnat-diff-add-0* [*simp*]: ⋀*m n*::*hypnat*. *n* − (*n* + *m*) = *0*
by *transfer* (*rule diff-add-0*)

lemma *hypnat-diff-mult-distrib*: ⋀*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*]: $\bigwedge n{::}hypnat.\ n \leq 0 \longleftrightarrow n = 0$
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*]: $\bigwedge m\ n{::}hypnat.\ m - n = 0 \longleftrightarrow m \leq n$
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*]: $\bigwedge m\ n{::}hypnat.\ n \leq m \Longrightarrow n + (m - n) = m$
by *transfer* (*rule le-add-diff-inverse*)

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

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

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

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

lemma *hypnat-add-self-le* [*simp*]: $\bigwedge x\ n{::}hypnat.\ x \leq n + x$
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* ∈ *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* [*rule-format*]: ∀ *d*::*hypnat*. *of-nat m = of-nat n + d* −−> *d* ∈ *range of-nat*
apply (*induct n*)
apply (*auto simp add*: *add.assoc*)

apply (*case-tac x*)
apply (*auto simp add*: *add.commute* [*of 1*])
done

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

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

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

lemma *star-of-le-HNatInfinite*: $N \in HNatInfinite \Longrightarrow star\text{-}of\ n \leq N$
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*: $x \in Nats \Longrightarrow y \in HNatInfinite \Longrightarrow x \leq y$
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*: $x \in HNatInfinite \Longrightarrow 1 \leq x$
by (*simp add*: *Nats-le-HNatInfinite*)

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

lemma *Nats-downward-closed*: $x \in Nats \Longrightarrow y \leq x \Longrightarrow y \in Nats$ **for** $x\ y :: hypnat$
apply (*simp only*: *linorder-not-less* [*symmetric*])
apply (*erule contrapos-np*)
apply (*drule HNatInfinite-not-Nats-iff* [*THEN iffD2*])
apply (*erule* (*1*) *Nats-less-HNatInfinite*)
done

lemma *HNatInfinite-upward-closed*: $x \in HNatInfinite \Longrightarrow x \leq y \Longrightarrow y \in HNatInfinite$
apply (*simp only*: *HNatInfinite-not-Nats-iff*)
apply (*erule contrapos-nn*)
apply (*erule* (*1*) *Nats-downward-closed*)
done

lemma *HNatInfinite-add*: $x \in HNatInfinite \Longrightarrow x + y \in HNatInfinite$
apply (*erule HNatInfinite-upward-closed*)
apply (*rule hypnat-le-add1*)
done

lemma *HNatInfinite-add-one*: $x \in HNatInfinite \Longrightarrow x + 1 \in HNatInfinite$
by (*rule HNatInfinite-add*)

lemma *HNatInfinite-diff*: $x \in HNatInfinite \Longrightarrow y \in Nats \Longrightarrow x - y \in HNatInfinite$
apply (*frule* (*1*) *Nats-le-HNatInfinite*)
apply (*simp only*: *HNatInfinite-not-Nats-iff*)
apply (*erule contrapos-nn*)
apply (*drule* (*1*) *Nats-add*, *simp*)
done

lemma *HNatInfinite-is-Suc*: $x \in HNatInfinite \Longrightarrow \exists y.\ x = y + 1$ **for** $x :: hypnat$
apply (*rule-tac* $x = x - (1::hypnat)$ **in** *exI*)
apply (*simp add*: *Nats-le-HNatInfinite*)
done

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

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

lemma *hypnat-of-nat-eq*: $hypnat\text{-}of\text{-}nat\ m = star\text{-}n\ (\lambda n{::}nat.\ m)$
by (*simp add*: *star-of-def*)

lemma *SHNat-eq*: $Nats = \{n.\ \exists N.\ n = hypnat\text{-}of\text{-}nat\ N\}$
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*]: $hypnat\text{-}of\text{-}nat\ n < whn$
by (*simp add*: *Nats-less-whn*)

lemma *hypnat-of-nat-le-whn* [*simp*]: $hypnat\text{-}of\text{-}nat\ n \leq whn$
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

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

lemma *HNatInfinite-FreeUltrafilterNat-lemma*:
assumes $\forall N{::}nat.\ eventually\ (\lambda n.\ f\ n \neq N)\ \mathcal{U}$
shows $eventually\ (\lambda n.\ N < f\ n)\ \mathcal{U}$
apply (*induct N*)
using *assms*
apply (*drule-tac x = 0* **in** *spec*, *simp*)
using *assms*
apply (*drule-tac x = Suc N* **in** *spec*)
apply (*auto elim*: *eventually-elim2*)
done

lemma *HNatInfinite-iff*: $HNatInfinite = \{N.\ \forall n \in Nats.\ n < N\}$
apply (*safe intro*!: *Nats-less-HNatInfinite*)
apply (*auto simp add*: *HNatInfinite-def*)
done

3.4.2 Alternative Characterization of *HNatInfinite* using Free Ultrafilter

lemma *HNatInfinite-FreeUltrafilterNat*:
$star\text{-}n\ X \in HNatInfinite \Longrightarrow \forall u.\ eventually\ (\lambda n.\ u < X\ n)\ \mathcal{U}$
apply (*auto simp add*: *HNatInfinite-iff SHNat-eq*)
apply (*drule-tac x=star-of u* **in** *spec*, *simp*)
apply (*simp add*: *star-of-def star-less-def starP2-star-n*)
done

lemma *FreeUltrafilterNat-HNatInfinite*:
$\forall u.\ eventually\ (\lambda n.\ u < X\ n)\ \mathcal{U} \Longrightarrow star\text{-}n\ X \in HNatInfinite$
by (*auto simp add*: *star-less-def starP2-star-n HNatInfinite-iff SHNat-eq hypnat-of-nat-eq*)

lemma *HNatInfinite-FreeUltrafilterNat-iff*:
$(star\text{-}n\ X \in HNatInfinite) = (\forall u.\ eventually\ (\lambda n.\ u < X\ n)\ \mathcal{U})$
by (*rule iffI* [*OF HNatInfinite-FreeUltrafilterNat FreeUltrafilterNat-HNatInfinite*])

3.5 Embedding of the Hypernaturals into other types

definition *of-hypnat* :: $hypnat \Rightarrow {'a}{::}semiring\text{-}1\text{-}cancel\ star$
where *of-hypnat-def* [*transfer-unfold*]: $of\text{-}hypnat = *f* of\text{-}nat$

lemma *of-hypnat-0* [*simp*]: $of\text{-}hypnat\ 0 = 0$
by *transfer* (*rule of-nat-0*)

lemma *of-hypnat-1* [*simp*]: $of\text{-}hypnat\ 1 = 1$
by *transfer* (*rule of-nat-1*)

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

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

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

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

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

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

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

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

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

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

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

lemma *HNatInfinite-of-hypnat-gt-zero*:
$N \in$ *HNatInfinite* $\Longrightarrow$ (*0*::'a::*linordered-semidom star*) $<$ *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 ω = *star-n* (λ*n*. *real* (*Suc n*))
— an infinite number = [<*1*, *2*, *3*, ...>]

definition *epsilon* :: *hypreal* (ε)
where ε = *star-n* (λ*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* ≡ ∗*f*∗ (*scaleR r*)
instance ..
end

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

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

instance *star* :: (*real-vector*) *real-vector*
proof
fix *a b* :: *real*
show ⋀*x y*::'*a star*. *scaleR a* (*x* + *y*) = *scaleR a x* + *scaleR a y*
by *transfer* (*rule scaleR-right-distrib*)
show ⋀*x*::'*a star*. *scaleR* (*a* + *b*) *x* = *scaleR a x* + *scaleR b x*
by *transfer* (*rule scaleR-left-distrib*)
show ⋀*x*::'*a star*. *scaleR a* (*scaleR b x*) = *scaleR* (*a* ∗ *b*) *x*
by *transfer* (*rule scaleR-scaleR*)
show ⋀*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*]: $\bigwedge x\ y.$ *of-hypreal* (*x* + *y*) = *of-hypreal x* + *of-hypreal*

y
by *transfer* (*rule of-real-add*)

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

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

lemma *of-hypreal-mult* [*simp*]: $\bigwedge x\ y.$ *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*]: $x \in$ *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*: $X \in$ *Rep-star* $x \longleftrightarrow x =$ *star-n* X
by (*cases* x) (*simp add*: *star-n-def*)

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

lemma *Rep-star-star-n*: $X \in$ *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* ($\lambda n.$ $X\ n + Y\ n$)
by (*simp only*: *star-add-def starfun2-star-n*)

lemma *star-n-minus*: $-$ *star-n* X = *star-n* ($\lambda n.$ $-(X\ n)$)
by (*simp only*: *star-minus-def starfun-star-n*)

lemma *star-n-diff*: *star-n* X $-$ *star-n* Y = *star-n* ($\lambda n.$ $X\ n - Y\ n$)
by (*simp only*: *star-diff-def starfun2-star-n*)

lemma *star-n-mult*: *star-n* X $*$ *star-n* Y = *star-n* ($\lambda n.$ $X\ n * Y\ n$)
by (*simp only*: *star-mult-def starfun2-star-n*)

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

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

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

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

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

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

lemma *hypreal-omega-gt-zero* [*simp*]: $0 < \omega$
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.

A few lemmas first.

lemma *lemma-omega-empty-singleton-disj*:

$\{n::nat.\ x = real\ n\} = \{\} \vee (\exists\, y.\ \{n::nat.\ x = real\ n\} = \{y\})$
by *force*

lemma *lemma-finite-omega-set*: *finite* $\{n::nat.\ x = real\ n\}$
using *lemma-omega-empty-singleton-disj* [*of x*] **by** *auto*

lemma *not-ex-hypreal-of-real-eq-omega*: $\nexists x.$ *hypreal-of-real* $x = \omega$
apply (*simp add*: *omega-def star-of-def star-n-eq-iff*)
apply *clarify*
apply (*rule-tac x2=x−1* **in** *lemma-finite-omega-set* [*THEN FreeUltrafilterNat.finite*, *THEN notE*])
apply (*erule eventually-mono*)
apply *auto*
done

lemma *hypreal-of-real-not-eq-omega*: *hypreal-of-real* $x \neq \omega$
using *not-ex-hypreal-of-real-eq-omega* **by** *auto*

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

lemma *lemma-epsilon-empty-singleton-disj*:
$\{n::nat.\ x = inverse(real(Suc\ n))\} = \{\} \vee (\exists\, y.\ \{n::nat.\ x = inverse(real(Suc\ n))\} = \{y\})$
by *auto*

lemma *lemma-finite-epsilon-set*: *finite* $\{n.\ x = inverse\ (real\ (Suc\ n))\}$
using *lemma-epsilon-empty-singleton-disj* [*of x*] **by** *auto*

lemma *not-ex-hypreal-of-real-eq-epsilon*: $\nexists x.$ *hypreal-of-real* $x = \varepsilon$
by (*auto simp*: *epsilon-def star-of-def star-n-eq-iff*
lemma-finite-epsilon-set [*THEN FreeUltrafilterNat.finite*] *simp del*: *of-nat-Suc*)

lemma *hypreal-of-real-not-eq-epsilon*: *hypreal-of-real* $x \neq \varepsilon$
using *not-ex-hypreal-of-real-eq-epsilon* **by** *auto*

lemma *hypreal-epsilon-not-zero*: $\varepsilon \neq 0$
by (*simp add*: *epsilon-def star-zero-def star-of-def star-n-eq-iff FreeUltrafilterNat.proper*
del: *star-of-zero*)

lemma *hypreal-epsilon-inverse-omega*: $\varepsilon = inverse\ \omega$
by (*simp add*: *epsilon-def omega-def star-n-inverse*)

lemma *hypreal-epsilon-gt-zero*: $0 < \varepsilon$
by (*simp add*: *hypreal-epsilon-inverse-omega*)

4.7 Absolute Value Function for the Hyperreals

lemma *hrabs-add-less*: $|x| < r \Longrightarrow |y| < s \Longrightarrow |x + y| < r + s$
for *x y r s* :: *hypreal*

by (*simp add*: *abs-if split*: *if-split-asm*)

lemma *hrabs-less-gt-zero*: $|x| < r \Longrightarrow 0 < r$
for *x r* :: *hypreal*
by (*blast intro*!: *order-le-less-trans abs-ge-zero*)

lemma *hrabs-disj*: $|x| = x \vee |x| = -x$
for *x* :: *'a*::*abs-if*
by (*simp add*: *abs-if*)

lemma *hrabs-add-lemma-disj*: $y + - x + (y + - z) = |x + - z| \Longrightarrow y = z \vee x = y$
for *x y z* :: *hypreal*
by (*simp add*: *abs-if split*: *if-split-asm*)

4.8 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-inj-thms* [@{*thm star-of-le*} *RS iffD2*,
@{*thm star-of-less*} *RS iffD2*, @{*thm star-of-eq*} *RS iffD2*]
#> *Lin-Arith.add-simps* [@{*thm star-of-zero*}, @{*thm star-of-one*},
@{*thm star-of-numeral*}, @{*thm star-of-add*},
@{*thm star-of-minus*}, @{*thm star-of-diff*}, @{*thm star-of-mult*}]
#> *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*›

4.9 Exponentials on the Hyperreals

lemma *hpowr-0* [*simp*]: *r* ^ *0* = (*1*::*hypreal*)
for *r* :: *hypreal*
by (*rule power-0*)

lemma *hpowr-Suc* [*simp*]: *r* ^ (*Suc n*) = *r* ∗ (*r* ^ *n*)
for *r* :: *hypreal*
by (*rule power-Suc*)

lemma *hrealpow-two*: $r \hat{} Suc\ (Suc\ 0) = r * r$
for r :: *hypreal*
by *simp*

lemma *hrealpow-two-le* [*simp*]: $0 \leq r \hat{} Suc\ (Suc\ 0)$
for r :: *hypreal*
by (*auto simp add*: *zero-le-mult-iff*)

lemma *hrealpow-two-le-add-order* [*simp*]: $0 \leq u \hat{} Suc\ (Suc\ 0) + v \hat{} Suc\ (Suc\ 0)$
for $u\ v$ :: *hypreal*
by (*simp only*: *hrealpow-two-le add-nonneg-nonneg*)

lemma *hrealpow-two-le-add-order2* [*simp*]: $0 \leq u \hat{} Suc\ (Suc\ 0) + v \hat{} Suc\ (Suc\ 0) + w \hat{} Suc\ (Suc\ 0)$
for $u\ v\ w$ :: *hypreal*
by (*simp only*: *hrealpow-two-le add-nonneg-nonneg*)

lemma *hypreal-add-nonneg-eq-0-iff*: $0 \leq x \Longrightarrow 0 \leq y \Longrightarrow x + y = 0 \longleftrightarrow x = 0 \wedge y = 0$
for $x\ y$ :: *hypreal*
by *arith*

lemma *hypreal-three-squares-add-zero-iff*: $x * x + y * y + z * z = 0 \longleftrightarrow x = 0 \wedge y = 0 \wedge z = 0$
for $x\ y\ z$ :: *hypreal*
by (*simp only*: *zero-le-square add-nonneg-nonneg hypreal-add-nonneg-eq-0-iff*) *auto*

lemma *hrealpow-three-squares-add-zero-iff* [*simp*]:
$x \hat{} Suc\ (Suc\ 0) + y \hat{} Suc\ (Suc\ 0) + z \hat{} Suc\ (Suc\ 0) = 0 \longleftrightarrow x = 0 \wedge y = 0 \wedge z = 0$
for $x\ y\ z$ :: *hypreal*
by (*simp only*: *hypreal-three-squares-add-zero-iff hrealpow-two*)

lemma *hrabs-hrealpow-two* [*simp*]: $|x \hat{} Suc\ (Suc\ 0)| = x \hat{} Suc\ (Suc\ 0)$
for x :: *hypreal*
by (*simp add*: *abs-mult*)

lemma *two-hrealpow-ge-one* [*simp*]: $(1{::}hypreal) \leq 2 \hat{} n$
using *power-increasing* [*of 0 n 2*::*hypreal*] **by** *simp*

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

lemma *hrealpow-sum-square-expand*:

$(x + y) \hat{} Suc\ (Suc\ 0) =$
$x \hat{} Suc\ (Suc\ 0) + y \hat{} Suc\ (Suc\ 0) + (hypreal\text{-}of\text{-}nat\ (Suc\ (Suc\ 0))) * x * y$
for $x\ y :: hypreal$
by (*simp add*: *distrib-left distrib-right*)

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

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

4.10 Powers with Hypernatural Exponents

Hypernatural powers of hyperreals.

definition *pow* :: *'a*::*power star* ⇒ *nat star* ⇒ *'a star* (**infixr** *pow* 80)
where *hyperpow-def* [*transfer-unfold*]: *R pow N* = (∗*f2*∗ *op* ^) *R N*

lemma *Standard-hyperpow* [*simp*]: *r* ∈ *Standard* ⟹ *n* ∈ *Standard* ⟹ *r pow n* ∈ *Standard*
by (*simp add*: *hyperpow-def*)

lemma *hyperpow*: *star-n X pow star-n Y* = *star-n* (λ*n*. *X n* ^ *Y n*)
by (*simp add*: *hyperpow-def starfun2-star-n*)

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

lemma *hyperpow-not-zero*: ⋀*r n*. *r* ≠ (*0*::*'a*::{*field*} *star*) ⟹ *r pow n* ≠ *0*
by *transfer* (*rule power-not-zero*)

lemma *hyperpow-inverse*: ⋀*r n*. *r* ≠ (*0*::*'a*::*field star*) ⟹ *inverse* (*r pow n*) = (*inverse r*) *pow n*
by *transfer* (*rule power-inverse* [*symmetric*])

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

lemma *hyperpow-add*: ⋀*r n m*. (*r*::*'a*::*monoid-mult star*) *pow* (*n* + *m*) = (*r pow n*) ∗ (*r pow m*)
by *transfer* (*rule power-add*)

lemma *hyperpow-one* [*simp*]: ⋀*r*. (*r*::*'a*::*monoid-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 *hrabs-hyperpow-two* [*simp*]:
$|x\ pow\ 2| = (x::'a::\{monoid\text{-}mult,linordered\text{-}ring\text{-}strict\}\ star)\ pow\ 2$
by (*simp only*: *abs-of-nonneg hyperpow-two-le*)

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

The precondition could be weakened to $(0::'a) \leq x$.

lemma *hypreal-mult-less-mono*: $u < v \Longrightarrow x < y \Longrightarrow 0 < v \Longrightarrow 0 < x \Longrightarrow u * x < v * y$
for *u v x y* :: *hypreal*
by (*simp add*: *mult-strict-mono order-less-imp-le*)

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*: $\bigwedge r::'a::linordered\text{-}semidom\ star.\ 1 \leq r \Longrightarrow 1 \leq r\ pow\ 2$

by *transfer* (*rule one-le-power*)

lemma *two-hyperpow-ge-one* [*simp*]: (*1*::*hypreal*) ≤ *2 pow n*
apply (*rule-tac y = 1 pow n* **in** *order-trans*)
apply (*rule-tac* [*2*] *hyperpow-le*)
apply *auto*
done

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

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

lemma *hyperpow-SHNat-le*:
0 ≤ *r* ⟹ *r* ≤ (*1*::*hypreal*) ⟹ *N* ∈ *HNatInfinite* ⟹ ∀ *n*∈*Nats*. *r pow N* ≤ *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* ^ *n*)
by *transfer* (*rule refl*)

lemma *hyperpow-SReal* [*simp*]: (*hypreal-of-real r*) *pow* (*hypnat-of-nat n*) ∈ ℝ
by (*simp add*: *Reals-eq-Standard*)

lemma *hyperpow-zero-HNatInfinite* [*simp*]: *N* ∈ *HNatInfinite* ⟹ (*0*::*hypreal*) *pow N* = *0*
by (*drule HNatInfinite-is-Suc*, *auto*)

lemma *hyperpow-le-le*: (*0*::*hypreal*) ≤ *r* ⟹ *r* ≤ *1* ⟹ *n* ≤ *N* ⟹ *r pow N* ≤ *r pow n*
apply (*drule order-le-less* [*of n*, *THEN iffD1*])
apply (*auto intro*: *hyperpow-less-le*)
done

lemma *hyperpow-Suc-le-self2*: (*0*::*hypreal*) ≤ *r* ⟹ *r* < *1* ⟹ *r pow* (*n* + (*1*::*hypnat*)) ≤ *r*
apply (*drule-tac n* = (*1*::*hypnat*) **in** *hyperpow-le-le*)
apply *auto*
done

lemma *hyperpow-hypnat-of-nat*: ⋀*x*. *x pow hypnat-of-nat n* = *x* ^ *n*
by *transfer* (*rule refl*)

lemma *of-hypreal-hyperpow*:
⋀*x n*. *of-hypreal* (*x pow n*) = (*of-hypreal x*::′*a*::{*real-algebra-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* = (λ*x*. *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*]: $\bigwedge x::'a::$*real-normed-vector star*. $0 \leq$ *hnorm* x
by *transfer* (*rule norm-ge-zero*)

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

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

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

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

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

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

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

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

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

lemma *zero-less-hnorm-iff* [*simp*]: $\bigwedge x::'a::$*real-normed-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-normed-vector star*. *hnorm* $(-x) =$ *hnorm* x

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

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

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

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

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

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

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

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

lemma *hnorm-divide*: $\bigwedge a\ b{::}'a{::}\{$*real-normed-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-normed-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-normed-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-minus-iff* [*simp*]: $-\ x \in \mathbb{R} \longleftrightarrow x \in \mathbb{R}$
apply *auto*
apply (*drule Reals-minus*)
apply *auto*
done

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

lemma *SReal-iff*: $x \in \mathbb{R} \longleftrightarrow (\exists\, y.\ x = hypreal\text{-}of\text{-}real\ y)$
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* ‘ $\mathbb{R} = UNIV$
apply (*auto simp add*: *SReal-def*)
apply (*rule inj-star-of* [*THEN inv-f-f*, *THEN subst*], *blast*)
done

lemma *SReal-hypreal-of-real-image*: $\exists\, x.\ x \in P \Longrightarrow P \subseteq \mathbb{R} \Longrightarrow \exists\, Q.\ P = hypreal\text{-}of\text{-}real$ ‘ Q
unfolding *SReal-def image-def* **by** *blast*

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*
apply (*auto simp*: *SReal-def*)
apply (*drule dense*)
apply *auto*
done

Completeness of Reals, but both lemmas are unused.

lemma *SReal-sup-lemma*:
$P \subseteq \mathbb{R} \Longrightarrow (\exists\, x \in P.\ y < x) = (\exists\, X.\ hypreal\text{-}of\text{-}real\ X \in P \wedge y < hypreal\text{-}of\text{-}real\ X)$
by (*blast dest*!: *SReal-iff* [*THEN iffD1*])

lemma *SReal-sup-lemma2*:
$P \subseteq \mathbb{R} \Longrightarrow \exists\, x.\ x \in P \Longrightarrow \exists\, y \in Reals.\ \forall\, x \in P.\ x < y \Longrightarrow$
$(\exists\, X.\ X \in \{w.\ hypreal\text{-}of\text{-}real\ w \in P\}) \wedge$
$(\exists\, Y.\ \forall\, X \in \{w.\ hypreal\text{-}of\text{-}real\ w \in P\}.\ X < Y)$
apply (*rule conjI*)
apply (*fast dest*!: *SReal-iff* [*THEN iffD1*])
apply (*auto*, *frule subsetD*, *assumption*)
apply (*drule SReal-iff* [*THEN iffD1*])
apply (*auto*, *rule-tac x = ya* **in** *exI*, *auto*)
done

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*: $-\ x \in HFinite \longleftrightarrow x \in HFinite$
by (*simp add*: *HFinite-def*)

lemma *HFinite-star-of* [*simp*]: *star-of x* $\in$ *HFinite*
apply (*simp add*: *HFinite-def*)
apply (*rule-tac x*=*star-of* (*norm x*) + *1* **in** *bexI*)
apply (*transfer*, *simp*)
apply (*blast intro*: *Reals-add SReal-hypreal-of-real Reals-1*)
done

lemma *SReal-subset-HFinite*: (ℝ::*hypreal set*) $\subseteq$ *HFinite*

by (*auto simp add*: *SReal-def*)

lemma *HFiniteD*: $x \in HFinite \Longrightarrow \exists t \in Reals.\ hnorm\ x < t$
by (*simp add*: *HFinite-def*)

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

lemma *HFinite-hnorm-iff* [*iff*]: $hnorm\ x \in HFinite \longleftrightarrow x \in HFinite$
for x :: *hypreal*
by (*simp add*: *HFinite-def*)

lemma *HFinite-numeral* [*simp*]: $numeral\ w \in HFinite$
unfolding *star-numeral-def* **by** (*rule HFinite-star-of*)

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

lemma *HFinite-0* [*simp*]: $0 \in HFinite$
unfolding *star-zero-def* **by** (*rule HFinite-star-of*)

lemma *HFinite-1* [*simp*]: $1 \in HFinite$
unfolding *star-one-def* **by** (*rule HFinite-star-of*)

lemma *hrealpow-HFinite*: $x \in HFinite \Longrightarrow x \hat{}\ n \in HFinite$
for x :: $'a$::{*real-normed-algebra*,*monoid-mult*} *star*
by (*induct n*) (*auto simp add*: *power-Suc intro*: *HFinite-mult*)

lemma *HFinite-bounded*: $x \in HFinite \Longrightarrow y \leq x \Longrightarrow 0 \leq y \Longrightarrow y \in HFinite$
for $x\ y$ :: *hypreal*
apply (*cases* $x \leq 0$)
apply (*drule-tac* $y = x$ **in** *order-trans*)
apply (*drule-tac* [2] *order-antisym*)
apply (*auto simp add*: *linorder-not-le*)
apply (*auto intro*: *order-le-less-trans simp add*: *abs-if HFinite-def*)
done

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

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

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

lemma *hypreal-sum-of-halves*: $x\ /\ 2 + x\ /\ 2 = x$
for x :: *hypreal*
by *auto*

lemma *Infinitesimal-add*: $x \in \mathit{Infinitesimal} \Longrightarrow y \in \mathit{Infinitesimal} \Longrightarrow x + y \in \mathit{Infinitesimal}$
apply (*rule InfinitesimalI*)
apply (*rule hypreal-sum-of-halves* [*THEN subst*])
apply (*drule half-gt-zero*)
apply (*blast intro*: *hnorm-add-less SReal-divide-numeral dest*: *InfinitesimalD*)
done

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

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

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

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

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

lemma *Infinitesimal-mult*: $x \in \mathit{Infinitesimal} \Longrightarrow y \in \mathit{Infinitesimal} \Longrightarrow x * y \in \mathit{Infinitesimal}$
for $x\ y$ :: $'a{::}\mathit{real\text{-}normed\text{-}algebra\ star}$
apply (*rule InfinitesimalI*)
apply (*subgoal-tac hnorm* $(x * y) < 1 * r$)
apply (*simp only*: *mult-1*)
apply (*rule hnorm-mult-less*)
apply (*simp-all add*: *InfinitesimalD*)
done

lemma *Infinitesimal-HFinite-mult*: $x \in \mathit{Infinitesimal} \Longrightarrow y \in \mathit{HFinite} \Longrightarrow x * y \in \mathit{Infinitesimal}$
for $x\ y$ :: $'a{::}\mathit{real\text{-}normed\text{-}algebra\ star}$

apply (*rule InfinitesimalI*)
apply (*drule HFiniteD*, *clarify*)
apply (*subgoal-tac* $0 < t$)
apply (*subgoal-tac hnorm* $(x * y) < (r \;/\; t) * t$, *simp*)
apply (*subgoal-tac* $0 < r \;/\; t$)
apply (*rule hnorm-mult-less*)
apply (*simp add*: *InfinitesimalD*)
apply *assumption*
apply *simp*
apply (*erule order-le-less-trans* [*OF hnorm-ge-zero*])
done

lemma *Infinitesimal-HFinite-scaleHR*:
$x \in Infinitesimal \Longrightarrow y \in HFinite \Longrightarrow scaleHR\ x\ y \in Infinitesimal$
apply (*rule InfinitesimalI*)
apply (*drule HFiniteD*, *clarify*)
apply (*drule InfinitesimalD*)
apply (*simp add*: *hnorm-scaleHR*)
apply (*subgoal-tac* $0 < t$)
apply (*subgoal-tac* $|x| * hnorm\ y < (r \;/\; t) * t$, *simp*)
apply (*subgoal-tac* $0 < r \;/\; t$)
apply (*rule mult-strict-mono'*, *simp-all*)
apply (*erule order-le-less-trans* [*OF hnorm-ge-zero*])
done

lemma *Infinitesimal-HFinite-mult2*:
$x \in Infinitesimal \Longrightarrow y \in HFinite \Longrightarrow y * x \in Infinitesimal$
for $x\ y :: 'a$::*real-normed-algebra star*
apply (*rule InfinitesimalI*)
apply (*drule HFiniteD*, *clarify*)
apply (*subgoal-tac* $0 < t$)
apply (*subgoal-tac hnorm* $(y * x) < t * (r \;/\; t)$, *simp*)
apply (*subgoal-tac* $0 < r \;/\; t$)
apply (*rule hnorm-mult-less*)
apply *assumption*
apply (*simp add*: *InfinitesimalD*)
apply *simp*
apply (*erule order-le-less-trans* [*OF hnorm-ge-zero*])
done

lemma *Infinitesimal-scaleR2*: $x \in Infinitesimal \Longrightarrow a *_R x \in Infinitesimal$
apply (*case-tac* $a = 0$, *simp*)
apply (*rule InfinitesimalI*)
apply (*drule InfinitesimalD*)
apply (*drule-tac* $x{=}r \;/\; |star\text{-}of\ a|$ **in** *bspec*)
apply (*simp add*: *Reals-eq-Standard*)
apply *simp*
apply (*simp add*: *hnorm-scaleR pos-less-divide-eq mult.commute*)
done

lemma *Compl-HFinite*: $-$ *HFinite* = *HInfinite*
apply (*auto simp add*: *HInfinite-def HFinite-def linorder-not-less*)
apply (*rule-tac y*=$r + 1$ **in** *order-less-le-trans*, *simp*)
apply *simp*
done

lemma *HInfinite-inverse-Infinitesimal*: $x \in$ *HInfinite* $\Longrightarrow$ *inverse* $x \in$ *Infinitesimal*
for x :: $'a$::*real-normed-div-algebra star*
apply (*rule InfinitesimalI*)
apply (*subgoal-tac* $x \neq 0$)
apply (*rule inverse-less-imp-less*)
apply (*simp add*: *nonzero-hnorm-inverse*)
apply (*simp add*: *HInfinite-def Reals-inverse*)
apply *assumption*
apply (*clarify*, *simp add*: *Compl-HFinite* [*symmetric*])
done

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*: $x \in$ *HInfinite* $\Longrightarrow y \in$ *HInfinite* $\Longrightarrow x * y \in$ *HInfinite*
for $x\ y$ :: $'a$::*real-normed-div-algebra star*
apply (*rule HInfiniteI*, *simp only*: *hnorm-mult*)
apply (*subgoal-tac* $r * 1 <$ *hnorm* $x *$ *hnorm* y, *simp only*: *mult-1*)
apply (*case-tac* $x = 0$, *simp add*: *HInfinite-def*)
apply (*rule mult-strict-mono*)
apply (*simp-all add*: *HInfiniteD*)
done

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

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*
apply (*drule HInfinite-minus-iff* [*THEN iffD2*])
apply (*rule HInfinite-minus-iff* [*THEN iffD1*])
apply (*simp only*: *minus-add add.commute*)
apply (*rule HInfinite-add-ge-zero*)
apply *simp-all*
done

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 *HFinite-sum-squares*:
$a \in HFinite \Longrightarrow b \in HFinite \Longrightarrow c \in HFinite \Longrightarrow a * a + b * b + c * c \in HFinite$
for *a b c* :: *'a*::*real-normed-algebra star*
by (*auto intro*: *HFinite-mult HFinite-add*)

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

lemma *not-Infinitesimal-not-zero2*: $x \in HFinite - Infinitesimal \Longrightarrow x \neq 0$
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*: $e \in \mathit{Infinitesimal} \Longrightarrow |x| < e \Longrightarrow x \in \mathit{Infinitesimal}$
for x :: *hypreal*
by (*erule hnorm-less-Infinitesimal*) *simp*

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

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

lemma *lemma-Infinitesimal-hyperpow*: $x \in \mathit{Infinitesimal} \Longrightarrow 0 < N \Longrightarrow |x\ \mathit{pow}\ N| \leq |x|$
for x :: *hypreal*
apply (*unfold Infinitesimal-def*)
apply (*auto intro*!: *hyperpow-Suc-le-self2*
simp: *hyperpow-hrabs* [*symmetric*] *hypnat-gt-zero-iff2 abs-ge-zero*)
done

lemma *Infinitesimal-hyperpow*: $x \in \mathit{Infinitesimal} \Longrightarrow 0 < N \Longrightarrow x\ \mathit{pow}\ N \in \mathit{Infinitesimal}$
for x :: *hypreal*
apply (*rule hrabs-le-Infinitesimal*)
apply (*rule-tac* [2] *lemma-Infinitesimal-hyperpow*)
apply *auto*
done

lemma *hrealpow-hyperpow-Infinitesimal-iff*:
$(x \hat{}\ n \in \mathit{Infinitesimal}) \longleftrightarrow x\ \mathit{pow}\ (\mathit{hypnat\text{-}of\text{-}nat}\ n) \in \mathit{Infinitesimal}$
by (*simp only*: *hyperpow-hypnat-of-nat*)

lemma *Infinitesimal-hrealpow*: $x \in \mathit{Infinitesimal} \Longrightarrow 0 < n \Longrightarrow x \hat{}\ n \in \mathit{Infinitesimal}$
for x :: *hypreal*
by (*simp add*: *hrealpow-hyperpow-Infinitesimal-iff Infinitesimal-hyperpow*)

lemma *not-Infinitesimal-mult*:
$x \notin \mathit{Infinitesimal} \Longrightarrow y \notin \mathit{Infinitesimal} \Longrightarrow x * y \notin \mathit{Infinitesimal}$
for $x\ y$ :: *'a*::*real-normed-div-algebra star*
apply (*unfold Infinitesimal-def*, *clarify*, *rename-tac r s*)
apply (*simp only*: *linorder-not-less hnorm-mult*)
apply (*drule-tac* $x = r * s$ **in** *bspec*)
apply (*fast intro*: *Reals-mult*)

apply *simp*
apply (*drule-tac* $c = s$ **and** $d = hnorm\ y$ **and** $a = r$ **and** $b = hnorm\ x$ **in** *mult-mono*)
apply *simp-all*
done

lemma *Infinitesimal-mult-disj*: $x * y \in Infinitesimal \Longrightarrow x \in Infinitesimal \vee y \in Infinitesimal$
for $x\ y :: 'a$::*real-normed-div-algebra star*
apply (*rule ccontr*)
apply (*drule de-Morgan-disj* [*THEN iffD1*])
apply (*fast dest*: *not-Infinitesimal-mult*)
done

lemma *HFinite-Infinitesimal-not-zero*: $x \in HFinite - Infinitesimal \Longrightarrow x \neq 0$
by *blast*

lemma *HFinite-Infinitesimal-diff-mult*:
$x \in HFinite - Infinitesimal \Longrightarrow y \in HFinite - Infinitesimal \Longrightarrow x * y \in HFinite - Infinitesimal$
for $x\ y :: 'a$::*real-normed-div-algebra star*
apply *clarify*
apply (*blast dest*: *HFinite-mult not-Infinitesimal-mult*)
done

lemma *Infinitesimal-subset-HFinite*: $Infinitesimal \subseteq HFinite$
apply (*simp add*: *Infinitesimal-def HFinite-def*)
apply *auto*
apply (*rule-tac* $x = 1$ **in** *bexI*)
apply *auto*
done

lemma *Infinitesimal-star-of-mult*: $x \in Infinitesimal \Longrightarrow x * star\text{-}of\ r \in Infinitesimal$
for $x :: 'a$::*real-normed-algebra star*
by (*erule HFinite-star-of* [*THEN* [*2*] *Infinitesimal-HFinite-mult*])

lemma *Infinitesimal-star-of-mult2*: $x \in Infinitesimal \Longrightarrow star\text{-}of\ r * x \in Infinitesimal$
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 *hypreal-minus-distrib1*: $-(y + -x) = x + -y$
for $x\ y :: 'a::ab\text{-}group\text{-}add$
by (*simp add*: *add.commute*)

lemma *approx-sym*: $x \approx y \Longrightarrow y \approx x$
apply (*simp add*: *approx-def*)
apply (*drule Infinitesimal-minus-iff* [*THEN iffD2*])
apply *simp*
done

lemma *approx-trans*: $x \approx y \Longrightarrow y \approx z \Longrightarrow x \approx z$
apply (*simp add*: *approx-def*)
apply (*drule* (*1*) *Infinitesimal-add*)
apply *simp*
done

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 phi ss 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 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*: $x \approx y \longleftrightarrow monad\ x = monad\ y$
by (*auto simp add*: *monad-def dest*: *approx-sym elim*!: *approx-trans equalityCE*)

lemma *Infinitesimal-approx*: $x \in \mathit{Infinitesimal} \Longrightarrow y \in \mathit{Infinitesimal} \Longrightarrow x \approx y$
apply (*simp add*: *mem-infmal-iff*)
apply (*blast intro*: *approx-trans approx-sym*)
done

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

lemma *approx-minus*: $a \approx b \Longrightarrow -a \approx -b$
apply (*rule approx-minus-iff* [*THEN iffD2*, *THEN approx-sym*])
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*simp add*: *add.commute*)
done

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

lemma *approx-mult2*: $a \approx b \Longrightarrow c \in \mathit{HFinite} \Longrightarrow c * a \approx c * b$
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*: $u \approx v * x \Longrightarrow x \approx y \Longrightarrow v \in \mathit{HFinite} \Longrightarrow u \approx v * y$
for *u v x y* :: *'a*::*real-normed-algebra star*
by (*blast intro*: *approx-mult2 approx-trans*)

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

lemma *approx-mult-subst-star-of*: $u \approx x * \mathit{star\text{-}of}\ v \Longrightarrow x \approx y \Longrightarrow u \approx y * \mathit{star\text{-}of}\ v$
for $u\ x\ y :: 'a{::}\mathit{real\text{-}normed\text{-}algebra\ star}$
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*: $x \in \mathit{Infinitesimal} \Longrightarrow -x \approx x$
by (*blast intro*: *Infinitesimal-minus-iff* [*THEN iffD2*] *mem-infmal-iff* [*THEN iffD1*] *approx-trans2*)

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

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

lemma *Infinitesimal-add-approx*: $y \in \mathit{Infinitesimal} \Longrightarrow x + y = z \Longrightarrow x \approx z$
apply (*rule bex-Infinitesimal-iff* [*THEN iffD1*])
apply (*drule Infinitesimal-minus-iff* [*THEN iffD2*])
apply (*auto simp add*: *add.assoc* [*symmetric*])
done

lemma *Infinitesimal-add-approx-self*: $y \in \mathit{Infinitesimal} \Longrightarrow x \approx x + y$
apply (*rule bex-Infinitesimal-iff* [*THEN iffD1*])
apply (*drule Infinitesimal-minus-iff* [*THEN iffD2*])
apply (*auto simp add*: *add.assoc* [*symmetric*])
done

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

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

lemma *Infinitesimal-add-cancel*: $y \in \mathit{Infinitesimal} \Longrightarrow x + y \approx z \Longrightarrow x \approx z$
apply (*drule-tac* $x = x$ **in** *Infinitesimal-add-approx-self* [*THEN approx-sym*])
apply (*erule approx-trans3* [*THEN approx-sym*], *assumption*)
done

lemma *Infinitesimal-add-right-cancel*: $y \in \mathit{Infinitesimal} \Longrightarrow x \approx z + y \Longrightarrow x \approx z$
apply (*drule-tac* $x = z$ **in** *Infinitesimal-add-approx-self2* [*THEN approx-sym*])
apply (*erule approx-trans3* [*THEN approx-sym*])
apply (*simp add*: *add.commute*)
apply (*erule approx-sym*)
done

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

apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*simp add*: *approx-minus-iff* [*symmetric*] *ac-simps*)
done

lemma *approx-add-right-cancel*: $b + d \approx c + d \Longrightarrow b \approx c$
apply (*rule approx-add-left-cancel*)
apply (*simp add*: *add.commute*)
done

lemma *approx-add-mono1*: $b \approx c \Longrightarrow d + b \approx d + c$
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*simp add*: *approx-minus-iff* [*symmetric*] *ac-simps*)
done

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*: $x \in HFinite \Longrightarrow x \approx y \Longrightarrow y \in HFinite$
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*], *safe*)
apply (*drule Infinitesimal-subset-HFinite* [*THEN subsetD*, *THEN HFinite-minus-iff* [*THEN iffD2*]])
apply (*drule HFinite-add*)
apply (*auto simp add*: *add.assoc*)
done

lemma *approx-star-of-HFinite*: $x \approx star\text{-}of\ D \Longrightarrow x \in HFinite$
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*
apply (*rule approx-trans*)
apply (*rule-tac* [*2*] *approx-mult2*)
apply (*rule approx-mult1*)
prefer *2* **apply** (*blast intro*: *approx-HFinite approx-sym*, *auto*)
done

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*: $a \approx star\text{-}of\ b \Longrightarrow c \in HFinite \Longrightarrow scaleHR\ a\ c \approx b *_R c$
apply (*unfold approx-def*)

apply (*drule* (*1*) *Infinitesimal-HFinite-scaleHR*)
apply (*simp only*: *scaleHR-left-diff-distrib*)
apply (*simp add*: *scaleHR-def star-scaleR-def* [*symmetric*])
done

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* $a\ c \approx b *_R d$
apply (*rule approx-trans*)
apply (*rule-tac* [*2*] *approx-scaleR2*)
apply (*rule approx-scaleR1*)
prefer *2* **apply** (*blast intro*: *approx-HFinite approx-sym*, *auto*)
done

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*: $a \in \mathbb{R} \Longrightarrow a \neq 0 \Longrightarrow a * x \approx 0 \Longrightarrow x \approx 0$
for $a\ x ::$ *hypreal*
apply (*drule Reals-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
done

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*: $a \in \mathbb{R} \Longrightarrow a \neq 0 \Longrightarrow a * w \approx a * z \Longrightarrow w \approx z$
for $a\ w\ z ::$ *hypreal*
apply (*drule Reals-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
done

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 (*auto intro*!: *approx-mult2 SReal-subset-HFinite* [*THEN subsetD*]
intro: *approx-SReal-mult-cancel*)

lemma *approx-le-bound*: $z \leq f \Longrightarrow f \approx g \Longrightarrow g \leq z \Longrightarrow f \approx z$
for *z* :: *hypreal*
apply (*simp add*: *bex-Infinitesimal-iff2* [*symmetric*], *auto*)
apply (*rule-tac x* = *g* + *y* − *z* **in** *bexI*)
apply *simp*
apply (*rule Infinitesimal-interval2*)
apply (*rule-tac* [*2*] *Infinitesimal-zero*, *auto*)
done

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 $0 \leq |hnorm\ x - hnorm\ y|$
by (*rule abs-ge-zero*)
moreover have $|hnorm\ x - hnorm\ y| \leq hnorm\ (x - y)$
by (*rule hnorm-triangle-ineq3*)
ultimately have $|hnorm\ x - hnorm\ y| \in Infinitesimal$
by (*rule Infinitesimal-interval2*)
then show $hnorm\ x - hnorm\ y \in Infinitesimal$
by (*simp only*: *Infinitesimal-hrabs-iff*)
qed

5.6 Zero is the Only Infinitesimal that is also a Real

lemma *Infinitesimal-less-SReal*: $x \in \mathbb{R} \Longrightarrow y \in Infinitesimal \Longrightarrow 0 < x \Longrightarrow y < x$
for *x y* :: *hypreal*
apply (*simp add*: *Infinitesimal-def*)
apply (*rule abs-ge-self* [*THEN order-le-less-trans*], *auto*)
done

lemma *Infinitesimal-less-SReal2*: $y \in Infinitesimal \Longrightarrow \forall\, r \in Reals.\ 0 < r \longrightarrow y < r$
for *y* :: *hypreal*
by (*blast intro*: *Infinitesimal-less-SReal*)

lemma *SReal-not-Infinitesimal*: $0 < y \Longrightarrow y \in \mathbb{R} \Longrightarrow y \notin Infinitesimal$
for *y* :: *hypreal*
apply (*simp add*: *Infinitesimal-def*)

apply (*auto simp add*: *abs-if*)
done

lemma *SReal-minus-not-Infinitesimal*: $y < 0 \Longrightarrow y \in \mathbb{R} \Longrightarrow y \notin$ *Infinitesimal*
for y :: *hypreal*
apply (*subst Infinitesimal-minus-iff* [*symmetric*])
apply (*rule SReal-not-Infinitesimal*, *auto*)
done

lemma *SReal-Int-Infinitesimal-zero*: $\mathbb{R}$ *Int Infinitesimal* = {*0*::*hypreal*}
apply *auto*
apply (*cut-tac* $x = x$ **and** $y = 0$ **in** *linorder-less-linear*)
apply (*blast dest*: *SReal-not-Infinitesimal SReal-minus-not-Infinitesimal*)
done

lemma *SReal-Infinitesimal-zero*: $x \in \mathbb{R} \Longrightarrow x \in$ *Infinitesimal* $\Longrightarrow x = 0$
for x :: *hypreal*
using *SReal-Int-Infinitesimal-zero* **by** *blast*

lemma *SReal-HFinite-diff-Infinitesimal*: $x \in \mathbb{R} \Longrightarrow x \neq 0 \Longrightarrow x \in$ *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 $x \neq 0 \Longrightarrow$ *hypreal-of-real* $x \in$ *HFinite* − *Infinitesimal*
by (*rule SReal-HFinite-diff-Infinitesimal*) *auto*

lemma *star-of-Infinitesimal-iff-0* [*iff*]: *star-of* $x \in$ *Infinitesimal* $\longleftrightarrow x = 0$
apply (*auto simp add*: *Infinitesimal-def*)
apply (*drule-tac x*=*hnorm* (*star-of x*) **in** *bspec*)
apply (*simp add*: *SReal-def*)
apply (*rule-tac x*=*norm x* **in** *exI*, *simp*)
apply *simp*
done

lemma *star-of-HFinite-diff-Infinitesimal*: $x \neq 0 \Longrightarrow$ *star-of* $x \in$ *HFinite* − *Infinitesimal*
by *simp*

lemma *numeral-not-Infinitesimal* [*simp*]:
numeral $w \neq$ (*0*::*hypreal*) $\Longrightarrow$ (*numeral* w :: *hypreal*) $\notin$ *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*) $\notin$ *Infinitesimal*
apply (*simp only*: *star-one-def star-of-Infinitesimal-iff-0*)
apply *simp*
done

lemma *approx-SReal-not-zero*: $y \in \mathbb{R} \Longrightarrow x \approx y \Longrightarrow y \neq 0 \Longrightarrow x \neq 0$
for $x\ y$ :: *hypreal*
apply (*cut-tac* $x = 0$ **and** $y = y$ **in** *linorder-less-linear*, *simp*)
apply (*blast dest*: *approx-sym* [*THEN mem-infmal-iff* [*THEN iffD2*]]
SReal-not-Infinitesimal SReal-minus-not-Infinitesimal)
done

lemma *HFinite-diff-Infinitesimal-approx*:
$x \approx y \Longrightarrow y \in HFinite - Infinitesimal \Longrightarrow x \in HFinite - Infinitesimal$
apply (*auto intro*: *approx-sym* [*THEN* [2] *approx-HFinite*] *simp*: *mem-infmal-iff*)
apply (*drule approx-trans3*, *assumption*)
apply (*blast dest*: *approx-sym*)
done

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

lemma *Infinitesimal-ratio*:
$y \neq 0 \Longrightarrow y \in Infinitesimal \Longrightarrow x\ /\ y \in HFinite \Longrightarrow x \in Infinitesimal$
for $x\ y$ :: $'a$::{*real-normed-div-algebra*,*field*} *star*
apply (*drule Infinitesimal-HFinite-mult2*, *assumption*)
apply (*simp add*: *divide-inverse mult.assoc*)
done

lemma *Infinitesimal-SReal-divide*: $x \in Infinitesimal \Longrightarrow y \in \mathbb{R} \Longrightarrow x\ /\ y \in Infinitesimal$
for $x\ y$ :: *hypreal*
apply (*simp add*: *divide-inverse*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult*
dest!: *Reals-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
done

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* $y \longleftrightarrow x = y$
apply *safe*
apply (*simp add*: *approx-def*)
apply (*simp only*: *star-of-diff* [*symmetric*])
apply (*simp only*: *star-of-Infinitesimal-iff-0*)
apply *simp*
done

lemma *SReal-approx-iff*: $x \in \mathbb{R} \Longrightarrow y \in \mathbb{R} \Longrightarrow x \approx y \longleftrightarrow x = y$

for *x y* :: *hypreal*
apply *auto*
apply (*simp add*: *approx-def*)
apply (*drule* (*1*) *Reals-diff*)
apply (*drule* (*1*) *SReal-Infinitesimal-zero*)
apply *simp*
done

lemma *numeral-approx-iff* [*simp*]:
(*numeral v* ≈ (*numeral w* :: *'a*::{*numeral*,*real-normed-vector*} *star*)) =
(*numeral v* = (*numeral w* :: *'a*))
apply (*unfold star-numeral-def*)
apply (*rule star-of-approx-iff*)
done

And also for *0* ≈ #*nn* and *1* ≈ #*nn*, #*nn* ≈ *0* and #*nn* ≈ *1*.

lemma [*simp*]:
(*numeral w* ≈ (*0*::*'a*::{*numeral*,*real-normed-vector*} *star*)) = (*numeral w* = (*0*::*'a*))
((*0*::*'a*::{*numeral*,*real-normed-vector*} *star*) ≈ *numeral w*) = (*numeral w* = (*0*::*'a*))
(*numeral w* ≈ (*1*::*'b*::{*numeral*,*one*,*real-normed-vector*} *star*)) = (*numeral w* = (*1*::*'b*))
((*1*::*'b*::{*numeral*,*one*,*real-normed-vector*} *star*) ≈ *numeral w*) = (*numeral w* = (*1*::*'b*))
¬ (*0* ≈ (*1*::*'c*::{*zero-neq-one*,*real-normed-vector*} *star*))
¬ (*1* ≈ (*0*::*'c*::{*zero-neq-one*,*real-normed-vector*} *star*))
apply (*unfold star-numeral-def star-zero-def star-one-def*)
apply (*unfold star-of-approx-iff*)
apply (*auto intro*: *sym*)
done

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

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

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

lemma *approx-unique-real*: *r* ∈ ℝ ⟹ *s* ∈ ℝ ⟹ *r* ≈ *x* ⟹ *s* ≈ *x* ⟹ *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-isLub1*: *isLub* $\mathbb{R}$ (*hypreal-of-real* ‘ Q) (*hypreal-of-real* Y) $\Longrightarrow$ *isLub UNIV Q Y*
for Q :: *real set* **and** Y :: *real*
apply (*simp add*: *isLub-def leastP-def*)
apply (*auto intro*: *hypreal-of-real-isUb-iff* [*THEN iffD2*]
simp add: *hypreal-of-real-isUb-iff setge-def*)
done

lemma *hypreal-of-real-isLub2*: *isLub UNIV Q Y* $\Longrightarrow$ *isLub* $\mathbb{R}$ (*hypreal-of-real* ‘ Q) (*hypreal-of-real* Y)
for Q :: *real set* **and** Y :: *real*
apply (*auto simp add*: *isLub-def leastP-def hypreal-of-real-isUb-iff setge-def*)
apply (*metis SReal-iff hypreal-of-real-isUb-iff isUbD2a star-of-le*)
done

lemma *hypreal-of-real-isLub-iff*:
isLub $\mathbb{R}$ (*hypreal-of-real* ‘ Q) (*hypreal-of-real* Y) = *isLub* (*UNIV* :: *real set*) Q Y
for Q :: *real set* **and** Y :: *real*
by (*blast intro*: *hypreal-of-real-isLub1 hypreal-of-real-isLub2*)

lemma *lemma-isUb-hypreal-of-real*: *isUb* $\mathbb{R}$ P Y $\Longrightarrow$ $\exists$ *Yo*. *isUb* $\mathbb{R}$ P (*hypreal-of-real Yo*)
by (*auto simp add*: *SReal-iff isUb-def*)

lemma *lemma-isLub-hypreal-of-real*: *isLub* $\mathbb{R}$ P Y $\Longrightarrow$ $\exists$ *Yo*. *isLub* $\mathbb{R}$ P (*hypreal-of-real Yo*)
by (*auto simp add*: *isLub-def leastP-def isUb-def SReal-iff*)

lemma *lemma-isLub-hypreal-of-real2*: $\exists$ *Yo*. *isLub* $\mathbb{R}$ P (*hypreal-of-real Yo*) $\Longrightarrow$ $\exists$ Y. *isLub* $\mathbb{R}$ P Y
by (*auto simp add*: *isLub-def leastP-def isUb-def*)

lemma *SReal-complete*: $P \subseteq \mathbb{R} \Longrightarrow \exists x.\ x \in P \Longrightarrow \exists Y.\ isUb\ \mathbb{R}\ P\ Y \Longrightarrow \exists t{::}hypreal.\ isLub\ \mathbb{R}\ P\ t$
apply (*frule SReal-hypreal-of-real-image*)
apply (*auto*, *drule lemma-isUb-hypreal-of-real*)
apply (*auto intro*!: *reals-complete lemma-isLub-hypreal-of-real2*
simp add: *hypreal-of-real-isLub-iff hypreal-of-real-isUb-iff*)
done

Lemmas about lubs.

lemma *lemma-st-part-ub*: $x \in HFinite \Longrightarrow \exists u.\ isUb\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ u$
for x :: *hypreal*
apply (*drule HFiniteD*, *safe*)
apply (*rule exI*, *rule isUbI*)

apply (*auto intro*: *setleI isUbI simp add*: *abs-less-iff*)
done

lemma *lemma-st-part-nonempty*: $x \in HFinite \Longrightarrow \exists y.\ y \in \{s.\ s \in \mathbb{R} \wedge s < x\}$
for x :: *hypreal*
apply (*drule HFiniteD*, *safe*)
apply (*drule Reals-minus*)
apply (*rule-tac* $x = -t$ **in** *exI*)
apply (*auto simp add*: *abs-less-iff*)
done

lemma *lemma-st-part-lub*: $x \in HFinite \Longrightarrow \exists t.\ isLub\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t$
for x :: *hypreal*
by (*blast intro*!: *SReal-complete lemma-st-part-ub lemma-st-part-nonempty Collect-restrict*)

lemma *lemma-st-part-le1*:
$x \in HFinite \Longrightarrow isLub\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow x \leq t + r$
for $x\ r\ t$ :: *hypreal*
apply (*frule isLubD1a*)
apply (*rule ccontr*, *drule linorder-not-le* [*THEN iffD2*])
apply (*drule* (*1*) *Reals-add*)
apply (*drule-tac* $y = r + t$ **in** *isLubD1* [*THEN setleD*], *auto*)
done

lemma *hypreal-setle-less-trans*: $S *<= x \Longrightarrow x < y \Longrightarrow S *<= y$
for $x\ y$ :: *hypreal*
apply (*simp add*: *setle-def*)
apply (*auto dest*!: *bspec order-le-less-trans intro*: *order-less-imp-le*)
done

lemma *hypreal-gt-isUb*: $isUb\ R\ S\ x \Longrightarrow x < y \Longrightarrow y \in R \Longrightarrow isUb\ R\ S\ y$
for $x\ y$ :: *hypreal*
apply (*simp add*: *isUb-def*)
apply (*blast intro*: *hypreal-setle-less-trans*)
done

lemma *lemma-st-part-gt-ub*: $x \in HFinite \Longrightarrow x < y \Longrightarrow y \in \mathbb{R} \Longrightarrow isUb\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ y$
for $x\ y$ :: *hypreal*
by (*auto dest*: *order-less-trans intro*: *order-less-imp-le intro*!: *isUbI setleI*)

lemma *lemma-minus-le-zero*: $t \leq t + -r \Longrightarrow r \leq 0$
for $r\ t$ :: *hypreal*
apply (*drule-tac* $c = -t$ **in** *add-left-mono*)
apply (*auto simp add*: *add.assoc* [*symmetric*])
done

lemma *lemma-st-part-le2*:

$x \in \mathit{HFinite} \Longrightarrow \mathit{isLub}\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow t + -r \leq x$
for $x\ r\ t$ :: *hypreal*
apply (*frule isLubD1a*)
apply (*rule ccontr*, *drule linorder-not-le* [*THEN iffD1*])
apply (*drule Reals-minus*, *drule-tac* $a = t$ **in** *Reals-add*, *assumption*)
apply (*drule lemma-st-part-gt-ub*, *assumption*+)
apply (*drule isLub-le-isUb*, *assumption*)
apply (*drule lemma-minus-le-zero*)
apply (*auto dest*: *order-less-le-trans*)
done

lemma *lemma-st-part1a*:
$x \in \mathit{HFinite} \Longrightarrow \mathit{isLub}\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow x + -t \leq r$
for $x\ r\ t$ :: *hypreal*
apply (*subgoal-tac* $x \leq t + r$)
apply (*auto intro*: *lemma-st-part-le1*)
done

lemma *lemma-st-part2a*:
$x \in \mathit{HFinite} \Longrightarrow \mathit{isLub}\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow -(x + -t) \leq r$
for $x\ r\ t$ :: *hypreal*
apply (*subgoal-tac* $(t + -r \leq x)$)
apply *simp*
apply (*rule lemma-st-part-le2*)
apply *auto*
done

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

lemma *lemma-SReal-lub*: $x \in \mathbb{R} \Longrightarrow \mathit{isLub}\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ x$
for x :: *hypreal*
apply (*auto intro*!: *isLubI2 lemma-SReal-ub setgeI*)
apply (*frule isUbD2a*)
apply (*rule-tac* $x = x$ **and** $y = y$ **in** *linorder-cases*)
apply (*auto intro*!: *order-less-imp-le*)
apply (*drule SReal-dense*, *assumption*, *assumption*, *safe*)
apply (*drule-tac* $y = r$ **in** *isUbD*)
apply (*auto dest*: *order-less-le-trans*)
done

lemma *lemma-st-part-not-eq1*:
$x \in \mathit{HFinite} \Longrightarrow \mathit{isLub}\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow x + - t \neq r$
for $x\ r\ t$ :: *hypreal*

apply *auto*
apply (*frule isLubD1a* [*THEN Reals-minus*])
using *Reals-add-cancel* [*of* $x - t$] **apply** *simp*
apply (*drule-tac* $x = x$ **in** *lemma-SReal-lub*)
apply (*drule isLub-unique*, *assumption*, *auto*)
done

lemma *lemma-st-part-not-eq2*:
$x \in HFinite \Longrightarrow isLub\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow -(x + -t) \neq r$
for $x\ r\ t$ :: *hypreal*
apply (*auto*)
apply (*frule isLubD1a*)
using *Reals-add-cancel* [*of* $-\ x\ t$] **apply** *simp*
apply (*drule-tac* $x = x$ **in** *lemma-SReal-lub*)
apply (*drule isLub-unique*, *assumption*, *auto*)
done

lemma *lemma-st-part-major*:
$x \in HFinite \Longrightarrow isLub\ \mathbb{R}\ \{s.\ s \in \mathbb{R} \wedge s < x\}\ t \Longrightarrow r \in \mathbb{R} \Longrightarrow 0 < r \Longrightarrow |x - t| < r$
for $x\ r\ t$ :: *hypreal*
apply (*frule lemma-st-part1a*)
apply (*frule-tac* [*4*] *lemma-st-part2a*, *auto*)
apply (*drule order-le-imp-less-or-eq*)+
apply *auto*
using *lemma-st-part-not-eq2* **apply** *fastforce*
using *lemma-st-part-not-eq1* **apply** *fastforce*
done

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*
apply (*frule lemma-st-part-lub*, *safe*)
apply (*frule isLubD1a*)
apply (*blast dest*: *lemma-st-part-major2*)
done

lemma *st-part-Ex*: $x \in HFinite \Longrightarrow \exists\, t{\in}Reals.\ x \approx t$
for x :: *hypreal*
apply (*simp add*: *approx-def Infinitesimal-def*)
apply (*drule lemma-st-part-Ex*, *auto*)

done

There is a unique real infinitely close.

lemma *st-part-Ex1*: $x \in HFinite \Longrightarrow \exists!t::hypreal.\ t \in \mathbb{R} \wedge x \approx t$
apply (*drule st-part-Ex*, *safe*)
apply (*drule-tac* [*2*] *approx-sym*, *drule-tac* [*2*] *approx-sym*, *drule-tac* [*2*] *approx-sym*)
apply (*auto intro*!: *approx-unique-real*)
done

6.3 Finite, Infinite and Infinitesimal

lemma *HFinite-Int-HInfinite-empty* [*simp*]: *HFinite Int HInfinite* = {}
apply (*simp add*: *HFinite-def HInfinite-def*)
apply (*auto dest*: *order-less-trans*)
done

lemma *HFinite-not-HInfinite*:
assumes *x*: $x \in HFinite$
shows $x \notin HInfinite$
proof
assume *x'*: $x \in HInfinite$
with *x* **have** $x \in HFinite \cap HInfinite$ **by** *blast*
then show *False* **by** *auto*
qed

lemma *not-HFinite-HInfinite*: $x \notin HFinite \Longrightarrow x \in HInfinite$
apply (*simp add*: *HInfinite-def HFinite-def*, *auto*)
apply (*drule-tac* $x = r + 1$ **in** *bspec*)
apply (*auto*)
done

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

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

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

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

lemma *HFinite-inverse*: $x \in HFinite \Longrightarrow x \notin Infinitesimal \Longrightarrow inverse\ x \in HFinite$
for *x* :: *'a*::*real-normed-div-algebra star*
apply (*subgoal-tac* $x \neq 0$)

apply (*cut-tac* $x = inverse\ x$ **in** *HInfinite-HFinite-disj*)
apply (*auto dest*!: *HInfinite-inverse-Infinitesimal simp*: *nonzero-inverse-inverse-eq*)
done

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

Stronger statement possible in fact.

lemma *Infinitesimal-inverse-HFinite*: $x \notin Infinitesimal \Longrightarrow inverse\ x \in HFinite$
for $x :: 'a$::*real-normed-div-algebra star*
apply (*drule HInfinite-diff-HFinite-Infinitesimal-disj*)
apply (*blast intro*: *HFinite-inverse HInfinite-inverse-Infinitesimal Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

lemma *HFinite-not-Infinitesimal-inverse*:
$x \in HFinite - Infinitesimal \Longrightarrow inverse\ x \in HFinite - Infinitesimal$
for $x :: 'a$::*real-normed-div-algebra star*
apply (*auto intro*: *Infinitesimal-inverse-HFinite*)
apply (*drule Infinitesimal-HFinite-mult2*, *assumption*)
apply (*simp add*: *not-Infinitesimal-not-zero*)
done

lemma *approx-inverse*: $x \approx y \Longrightarrow y \in HFinite - Infinitesimal \Longrightarrow inverse\ x \approx inverse\ y$
for $x\ y :: 'a$::*real-normed-div-algebra star*
apply (*frule HFinite-diff-Infinitesimal-approx*, *assumption*)
apply (*frule not-Infinitesimal-not-zero2*)
apply (*frule-tac* $x = x$ **in** *not-Infinitesimal-not-zero2*)
apply (*drule HFinite-inverse2*)+
apply (*drule approx-mult2*, *assumption*, *auto*)
apply (*drule-tac* $c = inverse\ x$ **in** *approx-mult1*, *assumption*)
apply (*auto intro*: *approx-sym simp add*: *mult.assoc*)
done

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*:
$x \in HFinite - Infinitesimal \Longrightarrow h \in Infinitesimal \Longrightarrow inverse\ (x + h) \approx inverse\ x$
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*:
$x \in HFinite - Infinitesimal \Longrightarrow h \in Infinitesimal \Longrightarrow inverse\ (h + x) \approx inverse$

x
for *x h* :: *'a*::*real-normed-div-algebra star*
apply (*rule add.commute* [*THEN subst*])
apply (*blast intro*: *inverse-add-Infinitesimal-approx*)
done

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*
apply (*rule approx-trans2*)
apply (*auto intro*: *inverse-add-Infinitesimal-approx*
simp add: *mem-infmal-iff approx-minus-iff* [*symmetric*])
done

lemma *Infinitesimal-square-iff*: *x* ∈ *Infinitesimal* ⟷ *x* ∗ *x* ∈ *Infinitesimal*
for *x* :: *'a*::*real-normed-div-algebra star*
apply (*auto intro*: *Infinitesimal-mult*)
apply (*rule ccontr*, *frule Infinitesimal-inverse-HFinite*)
apply (*frule not-Infinitesimal-not-zero*)
apply (*auto dest*: *Infinitesimal-HFinite-mult simp add*: *mult.assoc*)
done
declare *Infinitesimal-square-iff* [*symmetric*, *simp*]

lemma *HFinite-square-iff* [*simp*]: *x* ∗ *x* ∈ *HFinite* ⟷ *x* ∈ *HFinite*
for *x* :: *'a*::*real-normed-div-algebra star*
apply (*auto intro*: *HFinite-mult*)
apply (*auto dest*: *HInfinite-mult simp add*: *HFinite-HInfinite-iff*)
done

lemma *HInfinite-square-iff* [*simp*]: *x* ∗ *x* ∈ *HInfinite* ⟷ *x* ∈ *HInfinite*
for *x* :: *'a*::*real-normed-div-algebra star*
by (*auto simp add*: *HInfinite-HFinite-iff*)

lemma *approx-HFinite-mult-cancel*: *a* ∈ *HFinite* − *Infinitesimal* ⟹ *a* ∗ *w* ≈ *a* ∗ *z* ⟹ *w* ≈ *z*
for *a w z* :: *'a*::*real-normed-div-algebra star*
apply *safe*
apply (*frule HFinite-inverse*, *assumption*)
apply (*drule not-Infinitesimal-not-zero*)
apply (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
done

lemma *approx-HFinite-mult-cancel-iff1*: *a* ∈ *HFinite* − *Infinitesimal* ⟹ *a* ∗ *w* ≈ *a* ∗ *z* ⟷ *w* ≈ *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* ∈ *HInfinite* ⟹ *y* ∈ *HFinite* ⟹ *x*

$\in$ *HInfinite*
apply (*rule ccontr*)
apply (*drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*auto dest*: *HFinite-add simp add*: *HInfinite-HFinite-iff*)
done

lemma *HInfinite-HFinite-add*: $x \in HInfinite \Longrightarrow y \in HFinite \Longrightarrow x + y \in HInfinite$
apply (*rule-tac* $y = -y$ **in** *HInfinite-HFinite-add-cancel*)
apply (*auto simp add*: *add.assoc HFinite-minus-iff*)
done

lemma *HInfinite-ge-HInfinite*: $x \in HInfinite \Longrightarrow x \leq y \Longrightarrow 0 \leq x \Longrightarrow y \in HInfinite$
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*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*auto dest*: *Infinitesimal-HFinite-mult2*)
done

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*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*frule HFinite-Infinitesimal-not-zero*)
apply (*drule HFinite-not-Infinitesimal-inverse*)
apply (*safe*, *drule HFinite-mult*)
apply (*auto simp add*: *mult.assoc HFinite-HInfinite-iff*)
done

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*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*frule HFinite-Infinitesimal-not-zero*)
apply (*drule HFinite-not-Infinitesimal-inverse*)
apply (*safe*, *drule-tac* x=*inverse* y **in** *HFinite-mult*)
apply *assumption*
apply (*auto simp add*: *mult.assoc* [*symmetric*] *HFinite-HInfinite-iff*)
done

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*
using *hrabs-disj* [*of x*] **by** *auto*

6.4 Theorems about Monads

lemma *monad-hrabs-Un-subset*: $monad\ |x| \leq monad\ x \cup monad\ (-x)$
for x :: *hypreal*
by (*rule hrabs-disj* [*of x*, *THEN disjE*]) *auto*

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*
by (*rule hrabs-disj* [*of x*, *THEN disjE*]) (*auto simp*: *monad-zero-minus-iff* [*symmetric*])

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*: $x \approx y \Longrightarrow \{x, y\} \leq monad\ y$
apply (*drule approx-sym*)
apply (*fast dest*: *approx-subset-monad*)
done

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

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

lemma *approx-mem-monad2*: $x \approx u \Longrightarrow x \in \mathit{monad}\ u$
apply (*simp add*: *monad-def*)
apply (*blast intro*!: *approx-sym*)
done

lemma *approx-mem-monad-zero*: $x \approx y \Longrightarrow x \in \mathit{monad}\ 0 \Longrightarrow y \in \mathit{monad}\ 0$
apply (*drule mem-monad-approx*)
apply (*fast intro*: *approx-mem-monad approx-trans*)
done

lemma *Infinitesimal-approx-hrabs*: $x \approx y \Longrightarrow x \in \mathit{Infinitesimal} \Longrightarrow |x| \approx |y|$
for *x y* :: *hypreal*
apply (*drule Infinitesimal-monad-zero-iff* [*THEN iffD1*])
apply (*blast intro*: *approx-mem-monad-zero monad-zero-hrabs-iff* [*THEN iffD1*]
mem-monad-approx approx-trans3)
done

lemma *less-Infinitesimal-less*: $0 < x \Longrightarrow x \notin \mathit{Infinitesimal} \Longrightarrow e \in \mathit{Infinitesimal} \Longrightarrow e < x$
for *x* :: *hypreal*
apply (*rule ccontr*)
apply (*auto intro*: *Infinitesimal-zero* [*THEN* [*2*] *Infinitesimal-interval*]
dest!: *order-le-imp-less-or-eq simp add*: *linorder-not-less*)
done

lemma *Ball-mem-monad-gt-zero*: $0 < x \Longrightarrow x \notin \mathit{Infinitesimal} \Longrightarrow u \in \mathit{monad}\ x \Longrightarrow 0 < u$
for *u x* :: *hypreal*
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*erule bex-Infinitesimal-iff2* [*THEN iffD2*, *THEN bexE*])
apply (*drule-tac* $e = -xa$ **in** *less-Infinitesimal-less*, *auto*)
done

lemma *Ball-mem-monad-less-zero*: $x < 0 \Longrightarrow x \notin \mathit{Infinitesimal} \Longrightarrow u \in \mathit{monad}\ x \Longrightarrow u < 0$
for *u x* :: *hypreal*
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*erule bex-Infinitesimal-iff* [*THEN iffD2*, *THEN bexE*])
apply (*cut-tac* $x = -x$ **and** $e = xa$ **in** *less-Infinitesimal-less*, *auto*)
done

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

lemma *lemma-approx-less-zero*: $x < 0 \Longrightarrow x \notin \mathit{Infinitesimal} \Longrightarrow x \approx y \Longrightarrow y < 0$
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 *hrabs-disj* [*of x*] **by** (*auto dest*: *approx-minus*)

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

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

lemma *hrabs-add-Infinitesimal-cancel*:
$e \in \mathit{Infinitesimal} \Longrightarrow e' \in \mathit{Infinitesimal} \Longrightarrow |x + e| = |y + e'| \Longrightarrow |x| \approx |y|$
for $e\ e'\ x\ y$ :: *hypreal*
apply (*drule-tac* $x = x$ **in** *approx-hrabs-add-Infinitesimal*)
apply (*drule-tac* $x = y$ **in** *approx-hrabs-add-Infinitesimal*)
apply (*auto intro*: *approx-trans2*)
done

lemma *hrabs-add-minus-Infinitesimal-cancel*:
$e \in \mathit{Infinitesimal} \Longrightarrow e' \in \mathit{Infinitesimal} \Longrightarrow |x + -e| = |y + -e'| \Longrightarrow |x| \approx |y|$
for $e\ e'\ x\ y$ :: *hypreal*
apply (*drule-tac* $x = x$ **in** *approx-hrabs-add-minus-Infinitesimal*)
apply (*drule-tac* $x = y$ **in** *approx-hrabs-add-minus-Infinitesimal*)
apply (*auto intro*: *approx-trans2*)
done

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*:
$x < y \Longrightarrow u \in \mathit{Infinitesimal} \Longrightarrow \mathit{hypreal\text{-}of\text{-}real}\ x + u < \mathit{hypreal\text{-}of\text{-}real}\ y$
apply (*simp add*: *Infinitesimal-def*)
apply (*drule-tac* $x = \mathit{hypreal\text{-}of\text{-}real}\ y + -\mathit{hypreal\text{-}of\text{-}real}\ x$ **in** *bspec*, *simp*)
apply (*simp add*: *abs-less-iff*)
done

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less*:
x ∈ *Infinitesimal* ⟹ |*hypreal-of-real r*| < *hypreal-of-real y* ⟹
|*hypreal-of-real r* + *x*| < *hypreal-of-real y*
apply (*drule-tac x* = *hypreal-of-real r* **in** *approx-hrabs-add-Infinitesimal*)
apply (*drule approx-sym* [*THEN bex-Infinitesimal-iff2* [*THEN iffD2*]])
apply (*auto intro*!: *Infinitesimal-add-hypreal-of-real-less*
simp del: *star-of-abs simp add*: *star-of-abs* [*symmetric*])
done

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less2*:
x ∈ *Infinitesimal* ⟹ |*hypreal-of-real r*| < *hypreal-of-real y* ⟹
|*x* + *hypreal-of-real r*| < *hypreal-of-real y*
apply (*rule add.commute* [*THEN subst*])
apply (*erule Infinitesimal-add-hrabs-hypreal-of-real-less*, *assumption*)
done

lemma *hypreal-of-real-le-add-Infininitesimal-cancel*:
u ∈ *Infinitesimal* ⟹ *v* ∈ *Infinitesimal* ⟹
hypreal-of-real x + *u* ≤ *hypreal-of-real y* + *v* ⟹
hypreal-of-real x ≤ *hypreal-of-real y*
apply (*simp add*: *linorder-not-less* [*symmetric*], *auto*)
apply (*drule-tac u* = *v*−*u* **in** *Infinitesimal-add-hypreal-of-real-less*)
apply (*auto simp add*: *Infinitesimal-diff*)
done

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*
apply (*rule linorder-not-less* [*THEN iffD1*], *safe*)
apply (*drule Infinitesimal-interval*)
apply (*drule-tac* [*4*] *SReal-hypreal-of-real* [*THEN SReal-Infinitesimal-zero*],
auto)
done

lemma *Infinitesimal-add-not-zero*: *h* ∈ *Infinitesimal* ⟹ *x* ≠ *0* ⟹ *star-of x* + *h*
≠ *0*
apply *auto*
apply (*subgoal-tac h* = − *star-of x*)
apply (*auto intro*: *minus-unique* [*symmetric*])
done

lemma *Infinitesimal-square-cancel* [*simp*]: *x* ∗ *x* + *y* ∗ *y* ∈ *Infinitesimal* ⟹ *x* ∗
x ∈ *Infinitesimal*
for *x y* :: *hypreal*

apply (*rule Infinitesimal-interval2*)
apply (*rule-tac* [*3*] *zero-le-square*, *assumption*)
apply *auto*
done

lemma *HFinite-square-cancel* [*simp*]: $x * x + y * y \in HFinite \Longrightarrow x * x \in HFinite$
for *x y* :: *hypreal*
apply (*rule HFinite-bounded*, *assumption*)
apply *auto*
done

lemma *Infinitesimal-square-cancel2* [*simp*]: $x * x + y * y \in Infinitesimal \Longrightarrow y * y \in Infinitesimal$
for *x y* :: *hypreal*
apply (*rule Infinitesimal-square-cancel*)
apply (*rule add.commute* [*THEN subst*])
apply *simp*
done

lemma *HFinite-square-cancel2* [*simp*]: $x * x + y * y \in HFinite \Longrightarrow y * y \in HFinite$
for *x y* :: *hypreal*
apply (*rule HFinite-square-cancel*)
apply (*rule add.commute* [*THEN subst*])
apply *simp*
done

lemma *Infinitesimal-sum-square-cancel* [*simp*]:
$x * x + y * y + z * z \in Infinitesimal \Longrightarrow x * x \in Infinitesimal$
for *x y z* :: *hypreal*
apply (*rule Infinitesimal-interval2*, *assumption*)
apply (*rule-tac* [*2*] *zero-le-square*, *simp*)
apply (*insert zero-le-square* [*of y*])
apply (*insert zero-le-square* [*of z*], *simp del*:*zero-le-square*)
done

lemma *HFinite-sum-square-cancel* [*simp*]: $x * x + y * y + z * z \in HFinite \Longrightarrow x * x \in HFinite$
for *x y z* :: *hypreal*
apply (*rule HFinite-bounded*, *assumption*)
apply (*rule-tac* [*2*] *zero-le-square*)
apply (*insert zero-le-square* [*of y*])
apply (*insert zero-le-square* [*of z*], *simp del*:*zero-le-square*)
done

lemma *Infinitesimal-sum-square-cancel2* [*simp*]:
$y * y + x * x + z * z \in Infinitesimal \Longrightarrow x * x \in Infinitesimal$
for *x y z* :: *hypreal*
apply (*rule Infinitesimal-sum-square-cancel*)

apply (*simp add*: *ac-simps*)
done

lemma *HFinite-sum-square-cancel2* [*simp*]: $y * y + x * x + z * z \in HFinite \Longrightarrow x * x \in HFinite$
for *x y z* :: *hypreal*
apply (*rule HFinite-sum-square-cancel*)
apply (*simp add*: *ac-simps*)
done

lemma *Infinitesimal-sum-square-cancel3* [*simp*]:
$z * z + y * y + x * x \in Infinitesimal \Longrightarrow x * x \in Infinitesimal$
for *x y z* :: *hypreal*
apply (*rule Infinitesimal-sum-square-cancel*)
apply (*simp add*: *ac-simps*)
done

lemma *HFinite-sum-square-cancel3* [*simp*]: $z * z + y * y + x * x \in HFinite \Longrightarrow x * x \in HFinite$
for *x y z* :: *hypreal*
apply (*rule HFinite-sum-square-cancel*)
apply (*simp add*: *ac-simps*)
done

lemma *monad-hrabs-less*: $y \in monad\ x \Longrightarrow 0 < hypreal\text{-}of\text{-}real\ e \Longrightarrow |y - x| < hypreal\text{-}of\text{-}real\ e$
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*drule bex-Infinitesimal-iff* [*THEN iffD2*])
apply (*auto dest*!: *InfinitesimalD*)
done

lemma *mem-monad-SReal-HFinite*: $x \in monad\ (hypreal\text{-}of\text{-}real\ a) \Longrightarrow x \in HFinite$
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*])
apply (*safe dest*!: *Infinitesimal-subset-HFinite* [*THEN subsetD*])
apply (*erule SReal-hypreal-of-real* [*THEN SReal-subset-HFinite* [*THEN subsetD*], *THEN HFinite-add*])
done

6.7 Theorems about Standard Part

lemma *st-approx-self*: $x \in HFinite \Longrightarrow st\ x \approx x$
apply (*simp add*: *st-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *st-SReal*: $x \in HFinite \Longrightarrow st\ x \in \mathbb{R}$

apply (*simp add*: *st-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *st-HFinite*: $x \in HFinite \Longrightarrow st\ x \in HFinite$
by (*erule st-SReal* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])

lemma *st-unique*: $r \in \mathbb{R} \Longrightarrow r \approx x \Longrightarrow st\ x = r$
apply (*frule SReal-subset-HFinite* [*THEN subsetD*])
apply (*drule* (*1*) *approx-HFinite*)
apply (*unfold st-def*)
apply (*rule some-equality*)
apply (*auto intro*: *approx-unique-real*)
done

lemma *st-SReal-eq*: $x \in \mathbb{R} \Longrightarrow st\ x = x$
by (*metis approx-refl st-unique*)

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

lemma *st-eq-approx*: $x \in HFinite \Longrightarrow y \in HFinite \Longrightarrow st\ x = st\ y \Longrightarrow x \approx y$
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*: $x \in \mathbb{R} \Longrightarrow e \in Infinitesimal \Longrightarrow st\ (x + e) = x$
apply (*erule st-unique*)
apply (*erule Infinitesimal-add-approx-self*)
done

lemma *st-Infinitesimal-add-SReal2*: $x \in \mathbb{R} \Longrightarrow e \in Infinitesimal \Longrightarrow st\ (e + x) = x$
apply (*erule st-unique*)
apply (*erule Infinitesimal-add-approx-self2*)

done

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*]: $st\ (-\ numeral\ w) = -\ numeral\ w$
proof −
from *Reals-numeral* **have** $numeral\ w \in \mathbb{R}$ **.**
then have $-\ numeral\ w \in \mathbb{R}$ **by** *simp*
with *st-SReal-eq* **show** *?thesis* **.**
qed

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

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

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

lemma *st-inverse*: $x \in HFinite \Longrightarrow st\ x \neq 0 \Longrightarrow st\ (inverse\ x) = inverse\ (st\ x)$
apply (*rule-tac c1* = *st x* **in** *mult-left-cancel* [*THEN iffD1*])
apply (*auto simp add*: *st-mult* [*symmetric*] *st-not-Infinitesimal HFinite-inverse*)
apply (*subst right-inverse*, *auto*)
done

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

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

lemma *Infinitesimal-add-st-less*:
$x \in HFinite \Longrightarrow y \in HFinite \Longrightarrow u \in Infinitesimal \Longrightarrow st\ x < st\ y \Longrightarrow st\ x + u < st\ y$
apply (*drule st-SReal*)+
apply (*auto intro*!: *Infinitesimal-add-hypreal-of-real-less simp add*: *SReal-iff*)
done

lemma *Infinitesimal-add-st-le-cancel*:
$x \in HFinite \Longrightarrow y \in HFinite \Longrightarrow u \in Infinitesimal \Longrightarrow$
$st\ x \leq st\ y + u \Longrightarrow st\ x \leq st\ y$
apply (*simp add*: *linorder-not-less* [*symmetric*])
apply (*auto dest*: *Infinitesimal-add-st-less*)
done

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

lemma *st-zero-le*: $0 \leq x \Longrightarrow x \in HFinite \Longrightarrow 0 \leq st\ x$
apply (*subst st-0* [*symmetric*])
apply (*rule st-le*, *auto*)
done

lemma *st-zero-ge*: $x \leq 0 \Longrightarrow x \in HFinite \Longrightarrow st\ x \leq 0$
apply (*subst st-0* [*symmetric*])
apply (*rule st-le*, *auto*)
done

lemma *st-hrabs*: $x \in HFinite \Longrightarrow |st\ x| = st\ |x|$
apply (*simp add*: *linorder-not-le st-zero-le abs-if st-minus linorder-not-less*)
apply (*auto dest*!: *st-zero-ge* [*OF order-less-imp-le*])
done

6.8 Alternative Definitions using Free Ultrafilter

6.8.1 *HFinite*

lemma *HFinite-FreeUltrafilterNat*:
star-n X $\in HFinite \Longrightarrow \exists u.\ eventually\ (\lambda n.\ norm\ (X\ n) < u)\ \mathcal{U}$
apply (*auto simp add*: *HFinite-def SReal-def*)
apply (*rule-tac x*=*r* **in** *exI*)
apply (*simp add*: *hnorm-def star-of-def starfun-star-n*)
apply (*simp add*: *star-less-def starP2-star-n*)

done

lemma *FreeUltrafilterNat-HFinite*:
∃ *u*. *eventually* (λ*n*. *norm* (*X n*) < *u*) $\mathcal{U}$ ⟹ *star-n X* ∈ *HFinite*
apply (*auto simp add*: *HFinite-def mem-Rep-star-iff*)
apply (*rule-tac x*=*star-of u* **in** *bexI*)
apply (*simp add*: *hnorm-def starfun-star-n star-of-def*)
apply (*simp add*: *star-less-def starP2-star-n*)
apply (*simp add*: *SReal-def*)
done

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

6.8.2 *HInfinite*

lemma *lemma-Compl-eq*: − {*n*. *u* < *norm* (*f n*)} = {*n*. *norm* (*f n*) ≤ *u*}
by *auto*

lemma *lemma-Compl-eq2*: − {*n*. *norm* (*f n*) < *u*} = {*n*. *u* ≤ *norm* (*f n*)}
by *auto*

lemma *lemma-Int-eq1*: {*n*. *norm* (*f n*) ≤ *u*} *Int* {*n*. *u* ≤ *norm* (*f n*)} = {*n*. *norm*(*f n*) = *u*}
by *auto*

lemma *lemma-FreeUltrafilterNat-one*: {*n*. *norm* (*f n*) = *u*} ≤ {*n*. *norm* (*f n*) < *u* + (*1*::*real*)}
by *auto*

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*
apply (*rule FreeUltrafilterNat-HFinite*)
apply (*rule-tac x* = *u* + *1* **in** *exI*)
apply (*auto elim*: *eventually-mono*)
done

lemma *HInfinite-FreeUltrafilterNat*:
star-n X ∈ *HInfinite* ⟹ ∀ *u*. *eventually* (λ*n*. *u* < *norm* (*X n*)) $\mathcal{U}$
apply (*drule HInfinite-HFinite-iff* [*THEN iffD1*])
apply (*simp add*: *HFinite-FreeUltrafilterNat-iff*)
apply (*rule allI*, *drule-tac x*=*u* + *1* **in** *spec*)
apply (*simp add*: *FreeUltrafilterNat.eventually-not-iff* [*symmetric*])
apply (*auto elim*: *eventually-mono*)
done

lemma *lemma-Int-HI*: {*n*. *norm* (*Xa n*) < *u*} ∩ {*n*. *X n* = *Xa n*} ⊆ {*n*. *norm*

$(X\ n) < u\}$
for u :: *real*
by *auto*

lemma *lemma-Int-HIa*: $\{n.\ u < norm\ (X\ n)\} \cap \{n.\ norm\ (X\ n) < u\} = \{\}$
by (*auto intro*: *order-less-asym*)

lemma *FreeUltrafilterNat-HInfinite*:
$\forall u.$ *eventually* $(\lambda n.\ u < norm\ (X\ n))\ \mathcal{U} \Longrightarrow$ *star-n* $X \in$ *HInfinite*
apply (*rule HInfinite-HFinite-iff* [*THEN iffD2*])
apply (*safe*, *drule HFinite-FreeUltrafilterNat*, *safe*)
apply (*drule-tac* $x = u$ **in** *spec*)
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 show *False*
by (*simp add*: *eventually-False FreeUltrafilterNat.proper*)
qed

lemma *HInfinite-FreeUltrafilterNat-iff*:
star-n $X \in$ *HInfinite* $\longleftrightarrow$ $(\forall u.$ *eventually* $(\lambda n.\ u < norm\ (X\ n))\ \mathcal{U})$
by (*blast intro*!: *HInfinite-FreeUltrafilterNat FreeUltrafilterNat-HInfinite*)

6.8.3 *Infinitesimal*

lemma *ball-SReal-eq*: $(\forall x{::}hypreal \in Reals.\ P\ x) \longleftrightarrow (\forall x{::}real.\ P\ (star\text{-}of\ x))$
by (*auto simp*: *SReal-def*)

lemma *Infinitesimal-FreeUltrafilterNat*:
star-n $X \in$ *Infinitesimal* $\Longrightarrow \forall u{>}0.$ *eventually* $(\lambda n.\ norm\ (X\ n) < u)\ \mathcal{U}$
apply (*simp add*: *Infinitesimal-def ball-SReal-eq*)
apply (*simp add*: *hnorm-def starfun-star-n star-of-def*)
apply (*simp add*: *star-less-def starP2-star-n*)
done

lemma *FreeUltrafilterNat-Infinitesimal*:
$\forall u{>}0.$ *eventually* $(\lambda n.\ norm\ (X\ n) < u)\ \mathcal{U} \Longrightarrow$ *star-n* $X \in$ *Infinitesimal*
apply (*simp add*: *Infinitesimal-def ball-SReal-eq*)
apply (*simp add*: *hnorm-def starfun-star-n star-of-def*)
apply (*simp add*: *star-less-def starP2-star-n*)
done

lemma *Infinitesimal-FreeUltrafilterNat-iff*:
(*star-n* $X \in$ *Infinitesimal*) = $(\forall u{>}0.$ *eventually* $(\lambda n.\ norm\ (X\ n) < u)\ \mathcal{U})$
by (*blast intro*!: *Infinitesimal-FreeUltrafilterNat FreeUltrafilterNat-Infinitesimal*)

Infinitesimals as smaller than $1/n$ for all $n{::}nat\ (> 0)$.

lemma *lemma-Infinitesimal*: $(\forall r.\ 0 < r \longrightarrow x < r) \longleftrightarrow (\forall n.\ x < inverse\ (real\ (Suc\ n)))$
apply (*auto simp del*: *of-nat-Suc*)
apply (*blast dest*!: *reals-Archimedean intro*: *order-less-trans*)
done

lemma *lemma-Infinitesimal2*:
$(\forall r \in Reals.\ 0 < r \longrightarrow x < r) \longleftrightarrow (\forall n.\ x < inverse(hypreal\text{-}of\text{-}nat\ (Suc\ n)))$
apply *safe*
apply (*drule-tac* $x = inverse\ (hypreal\text{-}of\text{-}real\ (real\ (Suc\ n)))$ **in** *bspec*)
apply *simp-all*
using *less-imp-of-nat-less* **apply** *fastforce*
apply (*auto dest*!: *reals-Archimedean simp add*: *SReal-iff simp del*: *of-nat-Suc*)
apply (*drule star-of-less* [*THEN iffD2*])
apply *simp*
apply (*blast intro*: *order-less-trans*)
done

lemma *Infinitesimal-hypreal-of-nat-iff*:
$Infinitesimal = \{x.\ \forall n.\ hnorm\ x < inverse\ (hypreal\text{-}of\text{-}nat\ (Suc\ n))\}$
apply (*simp add*: *Infinitesimal-def*)
apply (*auto simp add*: *lemma-Infinitesimal2*)
done

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* $\{n::nat.\ real\ n < u\}$
apply (*cut-tac* $x = u$ **in** *reals-Archimedean2*, *safe*)
apply (*rule finite-real-of-nat-segment* [*THEN* [*2*] *finite-subset*])
apply (*auto dest*: *order-less-trans*)
done

lemma *lemma-real-le-Un-eq*: $\{n.\ f\ n \leq u\} = \{n.\ f\ n < u\} \cup \{n.\ u = (f\ n :: real)\}$
by (*auto dest*: *order-le-imp-less-or-eq simp add*: *order-less-imp-le*)

lemma *finite-real-of-nat-le-real*: *finite* $\{n::nat.\ real\ n \leq u\}$
by (*auto simp add*: *lemma-real-le-Un-eq lemma-finite-omega-set finite-real-of-nat-less-real*)

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* ($\lambda n.$ $u < real\ n$) $\mathcal{U}$
apply (*rule FreeUltrafilterNat.finite'*)
apply (*subgoal-tac* $\{n{::}nat.\ \neg\ u < real\ n\} = \{n.\ real\ n \leq u\}$)
apply (*auto simp add*: *finite-real-of-nat-le-real*)
done

The complement of $\{n.\ |real\ n| \leq u\} = \{n.\ u < |real\ n|\}$ is in $\mathcal{U}$ by property of (free) ultrafilters.

lemma *Compl-real-le-eq*: $-\ \{n{::}nat.\ real\ n \leq u\} = \{n.\ u < real\ n\}$
by (*auto dest*!: *order-le-less-trans simp add*: *linorder-not-le*)

ω is a member of *HInfinite*.

theorem *HInfinite-omega* [*simp*]: $\omega \in HInfinite$
apply (*simp add*: *omega-def*)
apply (*rule FreeUltrafilterNat-HInfinite*)
apply *clarify*
apply (*rule-tac u1* = *u*−*1* **in** *eventually-mono* [*OF FreeUltrafilterNat-nat-gt-real*])
apply *auto*
done

Epsilon is a member of Infinitesimal.

lemma *Infinitesimal-epsilon* [*simp*]: $\varepsilon \in Infinitesimal$
by (*auto intro*!: *HInfinite-inverse-Infinitesimal HInfinite-omega*
simp add: *hypreal-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*: $0 < u \Longrightarrow u < inverse\ (real(Suc\ n)) \longleftrightarrow real(Suc\ n) < inverse\ u$
apply (*simp add*: *inverse-eq-divide*)
apply (*subst pos-less-divide-eq*, *assumption*)
apply (*subst pos-less-divide-eq*)
apply *simp*
apply (*simp add*: *mult.commute*)
done

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 *lemma-real-le-Un-eq2*:
$\{n.\ u \leq inverse(real(Suc\ n))\} =$
$\{n.\ u < inverse(real(Suc\ n))\} \cup \{n.\ u = inverse(real(Suc\ n))\}$
by (*auto dest*: *order-le-imp-less-or-eq simp add*: *order-less-imp-le*)

lemma *finite-inverse-real-of-posnat-ge-real*: $0 < u \Longrightarrow$ *finite* $\{n.\ u \leq inverse\ (real\ (Suc\ n))\}$
by (*auto simp add*: *lemma-real-le-Un-eq2 lemma-finite-epsilon-set finite-inverse-real-of-posnat-gt-real simp del*: *of-nat-Suc*)

lemma *inverse-real-of-posnat-ge-real-FreeUltrafilterNat*:
$0 < u \Longrightarrow \neg$ *eventually* $(\lambda n.\ u \leq inverse(real(Suc\ n)))\ \mathcal{U}$
by (*blast intro*!: *FreeUltrafilterNat.finite finite-inverse-real-of-posnat-ge-real*)

The complement of $\{n.\ u \leq inverse(real(Suc\ n))\} = \{n.\ inverse\ (real\ (Suc\ n)) < u\}$ is in $\mathcal{U}$ by property of (free) ultrafilters.

lemma *Compl-le-inverse-eq*: $-\{n.\ u \leq inverse(real(Suc\ n))\} = \{n.\ inverse(real(Suc\ n)) < u\}$
by (*auto dest*!: *order-le-less-trans simp add*: *linorder-not-le*)

lemma *FreeUltrafilterNat-inverse-real-of-posnat*:
$0 < u \Longrightarrow$ *eventually* $(\lambda n.\ inverse(real(Suc\ n)) < u)\ \mathcal{U}$
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*: $(\ *f*\ (\lambda n{::}nat.\ inverse(real(Suc\ n))))\ whn \in Infinitesimal$
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*]: $x \in Standard \Longrightarrow hsgn\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-harg* [*simp*]: $x \in Standard \Longrightarrow harg\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hcis* [*simp*]: $r \in Standard \Longrightarrow hcis\ r \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hExp* [*simp*]: $x \in Standard \Longrightarrow hExp\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hrcis* [*simp*]: $r \in Standard \Longrightarrow s \in Standard \Longrightarrow hrcis\ r\ s \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-HComplex* [*simp*]: $r \in Standard \Longrightarrow s \in Standard \Longrightarrow HComplex\ r\ s \in Standard$
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*: $\bigwedge w\ z.\ w = z \longleftrightarrow hRe\ w = hRe\ z \wedge hIm\ w = hIm\ z$
by *transfer* (*rule complex-Re-Im-cancel-iff*)

lemma *hcomplex-equality* [*intro?*]: $\bigwedge z\ w.\ hRe\ z = hRe\ w \Longrightarrow hIm\ z = hIm\ w \Longrightarrow z = w$
by *transfer* (*rule complex-equality*)

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*]: $\bigwedge z.\ hRe\ (hcomplex\text{-}of\text{-}hypreal\ z) = z$
by *transfer* (*rule Re-complex-of-real*)

lemma *hIm-hcomplex-of-hypreal* [*simp*]: $\bigwedge z.\ hIm\ (hcomplex\text{-}of\text{-}hypreal\ z) = 0$
by *transfer* (*rule Im-complex-of-real*)

lemma *hcomplex-of-hypreal-epsilon-not-zero* [*simp*]: $hcomplex\text{-}of\text{-}hypreal\ \varepsilon \neq 0$
by (*simp add*: *hypreal-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*:
$hcomplex\text{-}of\text{-}hypreal\ |x| = hcomplex\text{-}of\text{-}hypreal\ (hcmod\ (hcomplex\text{-}of\text{-}hypreal\ x))$
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*]:
⋀*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*: ⋀*r*. *hcomplex-of-hypreal r* = *HComplex r 0*
by *transfer* (*rule complex-of-real-def*)

lemma *HComplex-add-hcomplex-of-hypreal* [*simp*]:
⋀*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*]:
⋀*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*:
⋀*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*:
⋀*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*]: ⋀*r*. *iii* ∗ *hcomplex-of-hypreal r* = *HComplex 0 r*
by *transfer* (*rule i-complex-of-real*)

lemma *hcomplex-of-hypreal-i* [*simp*]: ⋀*r*. *hcomplex-of-hypreal r* ∗ *iii* = *HComplex 0 r*
by *transfer* (*rule complex-of-real-i*)

7.9 Conjugation

lemma *hcomplex-hcnj-cancel-iff* [*iff*]: ⋀*x y*. *hcnj x* = *hcnj y* ⟷ *x* = *y*
by *transfer* (*rule complex-cnj-cancel-iff*)

lemma *hcomplex-hcnj-hcnj* [*simp*]: ⋀*z*. *hcnj* (*hcnj z*) = *z*
by *transfer* (*rule complex-cnj-cnj*)

lemma *hcomplex-hcnj-hcomplex-of-hypreal* [*simp*]:
⋀*x*. *hcnj* (*hcomplex-of-hypreal x*) = *hcomplex-of-hypreal x*
by *transfer* (*rule complex-cnj-complex-of-real*)

lemma *hcomplex-hmod-hcnj* [*simp*]: ⋀*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*]: $\bigwedge a\ b.\ hcmod\ (b + a) - hcmod\ b \leq hcmod\ a$
by *transfer* (*rule complex-mod-triangle-ineq2*)

lemma *hcmod-diff-ineq* [*simp*]: $\bigwedge a\ b.\ hcmod\ a - hcmod\ b \leq hcmod\ (a + b)$
by *transfer* (*rule norm-diff-ineq*)

7.11 Exponentiation

lemma *hcomplexpow-0* [*simp*]: $z \hat{} 0 = 1$
for z :: *hcomplex*
by (*rule power-0*)

lemma *hcomplexpow-Suc* [*simp*]: $z \hat{} (Suc\ n) = z * (z \hat{} n)$
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-of-hypreal* $(x \hat{} n) =$ *hcomplex-of-hypreal* $x \hat{} n$
by *transfer* (*rule of-real-power*)

lemma *hcomplex-hcnj-pow*: $\bigwedge z.$ *hcnj* $(z \hat{} n) =$ *hcnj* $z \hat{} n$
by *transfer* (*rule complex-cnj-power*)

lemma *hcmod-hcomplexpow*: $\bigwedge x.$ *hcmod* $(x \hat{} n) =$ *hcmod* $x \hat{} n$
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*]: $\bigwedge n.\ 0\ pow\ (hSuc\ n) = (0::'a::semiring\text{-}1\ star)$
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*: $\bigwedge z.\ hsgn\ z = z\ /\ hcomplex\text{-}of\text{-}hypreal\ (hcmod\ z)$
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*]:
$hcomplex\text{-}of\text{-}hypreal\ xa = HComplex\ x\ y \longleftrightarrow xa = x \wedge y = 0$
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *hcomplex-eq-cancel-iff2* [*simp*]:
$HComplex\ x\ y = hcomplex\text{-}of\text{-}hypreal\ xa \longleftrightarrow x = xa \wedge y = 0$
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *HComplex-eq-0* [*simp*]: $\bigwedge x\ y.\ HComplex\ x\ y = 0 \longleftrightarrow x = 0 \wedge y = 0$
by *transfer* (*rule Complex-eq-0*)

lemma *HComplex-eq-1* [*simp*]: $\bigwedge x\ y.\ HComplex\ x\ y = 1 \longleftrightarrow x = 1 \wedge y = 0$
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*]:
$\bigwedge r\ a.\ hRe\ (hcomplex\text{-}of\text{-}hypreal\ r * HComplex\ ((\ *f*\ cos)\ a)\ ((\ *f*\ sin)\ a)) = r * (\ *f*\ cos)\ a$
by *transfer simp*

lemma *hRe-hrcis* [*simp*]: $\bigwedge r\ a.\ hRe\ (hrcis\ r\ a) = r * (\ *f*\ cos)\ a$
by *transfer* (*rule Re-rcis*)

lemma *hIm-hcomplex-polar* [*simp*]:
$\bigwedge r\ a.\ hIm\ (hcomplex\text{-}of\text{-}hypreal\ r * HComplex\ ((\ *f*\ cos)\ a)\ ((\ *f*\ sin)\ a)) = r * (\ *f*\ sin)\ a$
by *transfer simp*

lemma *hIm-hrcis* [*simp*]: $\bigwedge r\ a.\ hIm\ (hrcis\ r\ a) = r * (\ *f*\ sin)\ a$
by *transfer* (*rule Im-rcis*)

lemma *hcmod-unit-one* [*simp*]: $\bigwedge a.\ hcmod\ (HComplex\ ((\ *f*\ cos)\ a)\ ((\ *f*\ sin)\ a)) = 1$
by *transfer* (*simp add*: *cmod-unit-one*)

lemma *hcmod-complex-polar* [*simp*]:
$\bigwedge r\ a.\ hcmod\ (hcomplex\text{-}of\text{-}hypreal\ r * HComplex\ ((\ *f*\ cos)\ a)\ ((\ *f*\ sin)\ a)) = |r|$
by *transfer* (*simp add*: *Complex-eq cmod-complex-polar*)

lemma *hcmod-hrcis* [*simp*]: $\bigwedge r\ a.\ hcmod(hrcis\ r\ a) = |r|$
by *transfer* (*rule complex-mod-rcis*)

$(r1 * hrcis\ a) * (r2 * hrcis\ b) = r1 * r2 * hrcis\ (a + b)$

lemma *hcis-hrcis-eq*: $\bigwedge a.\ hcis\ a = hrcis\ 1\ a$
by *transfer* (*rule cis-rcis-eq*)
declare *hcis-hrcis-eq* [*symmetric*, *simp*]

lemma *hrcis-mult*: $\bigwedge a\ b\ r1\ r2.\ hrcis\ r1\ a * hrcis\ r2\ b = hrcis\ (r1 * r2)\ (a + b)$
by *transfer* (*rule rcis-mult*)

lemma *hcis-mult*: $\bigwedge a\ b.\ hcis\ a * hcis\ b = hcis\ (a + b)$
by *transfer* (*rule cis-mult*)

lemma *hcis-zero* [*simp*]: $hcis\ 0 = 1$
by *transfer* (*rule cis-zero*)

lemma *hrcis-zero-mod* [*simp*]: $\bigwedge a.\ hrcis\ 0\ a = 0$
by *transfer* (*rule rcis-zero-mod*)

lemma *hrcis-zero-arg* [*simp*]: $\bigwedge r.\ hrcis\ r\ 0 = hcomplex\text{-}of\text{-}hypreal\ r$
by *transfer* (*rule rcis-zero-arg*)

lemma *hcomplex-i-mult-minus* [*simp*]: $\bigwedge x.\ iii * (iii * x) = - x$
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*:
$\bigwedge a.\ hcis\ (hypreal\text{-}of\text{-}nat\ (Suc\ n) * a) = hcis\ a * hcis\ (hypreal\text{-}of\text{-}nat\ n * a)$
by *transfer* (*simp add*: *distrib-right cis-mult*)

lemma *NSDeMoivre*: $\bigwedge a.\ (hcis\ a)\ \hat{}\ n = hcis\ (hypreal\text{-}of\text{-}nat\ n * a)$
by *transfer* (*rule DeMoivre*)

lemma *hcis-hypreal-of-hypnat-Suc-mult*:
$\bigwedge a\ n.\ hcis\ (hypreal\text{-}of\text{-}hypnat\ (n + 1) * a) = hcis\ a * hcis\ (hypreal\text{-}of\text{-}hypnat\ n * a)$
by *transfer* (*simp add*: *distrib-right cis-mult*)

lemma *NSDeMoivre-ext*: $\bigwedge a\ n.\ (hcis\ a)\ pow\ n = hcis\ (hypreal\text{-}of\text{-}hypnat\ n * a)$
by *transfer* (*rule DeMoivre*)

lemma *NSDeMoivre2*: $\bigwedge a\ r.\ (hrcis\ r\ a)\ \hat{}\ n = hrcis\ (r\ \hat{}\ n)\ (hypreal\text{-}of\text{-}nat\ n * a)$
by *transfer* (*rule DeMoivre2*)

lemma *DeMoivre2-ext*: $\bigwedge a\ r\ n.\ (hrcis\ r\ a)\ pow\ n = hrcis\ (r\ pow\ n)\ (hypreal\text{-}of\text{-}hypnat\ n * a)$
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*: $(\ *f*\ cos)\ (hypreal\text{-}of\text{-}nat\ n * a) = hRe\ (hcis\ a\ \hat{}\ n)$
by (*simp add*: *NSDeMoivre*)

lemma *sin-n-hIm-hcis-pow-n*: $(\ *f*\ sin)\ (hypreal\text{-}of\text{-}nat\ n * a) = hIm\ (hcis\ a\ \hat{}\ n)$
by (*simp add*: *NSDeMoivre*)

lemma *cos-n-hRe-hcis-hcpow-n*: $(\ *f*\ cos)\ (hypreal\text{-}of\text{-}hypnat\ n * a) = hRe\ (hcis\ a\ pow\ n)$
by (*simp add*: *NSDeMoivre-ext*)

lemma *sin-n-hIm-hcis-hcpow-n*: $(\ *f*\ sin)\ (hypreal\text{-}of\text{-}hypnat\ n * a) = hIm\ (hcis\ a\ pow\ n)$
by (*simp add*: *NSDeMoivre-ext*)

lemma *hExp-add*: $\bigwedge a\ b.\ hExp\ (a + b) = hExp\ a * hExp\ b$
by *transfer* (*rule exp-add*)

7.14 *hcomplex-of-complex*: the Injection from type *complex* to to *hcomplex*

lemma *hcomplex-of-complex-i*: $iii = hcomplex\text{-}of\text{-}complex\ \mathrm{i}$
by (*rule iii-def*)

lemma *hRe-hcomplex-of-complex*: $hRe\ (hcomplex\text{-}of\text{-}complex\ z) = hypreal\text{-}of\text{-}real\ (Re\ z)$
by *transfer* (*rule refl*)

lemma *hIm-hcomplex-of-complex*: $hIm\ (hcomplex\text{-}of\text{-}complex\ z) = hypreal\text{-}of\text{-}real\ (Im\ z)$
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* (∗*sn*∗ - [*80*] *80*)
where ∗*sn*∗ *As* = *Iset* (*star-n As*)

definition *InternalSets* :: *'a star set set*

where *InternalSets* = $\{X.\ \exists As.\ X = *sn* As\}$

definition — nonstandard extension of function
is-starext :: $('a\ star \Rightarrow 'a\ star) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
where *is-starext F f* $\longleftrightarrow$
$(\forall x\ y.\ \exists X \in Rep\text{-}star\ x.\ \exists Y \in Rep\text{-}star\ y.\ y = F\ x \longleftrightarrow eventually\ (\lambda n.\ Y\ n = f(X\ n))\ \mathcal{U})$

definition — internal functions
starfun-n :: $(nat \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a\ star \Rightarrow 'b\ star$ $(*fn*\ \text{-}\ [80]\ 80)$
where $*fn*\ F = Ifun\ (star\text{-}n\ F)$

definition *InternalFuns* :: $('a\ star => 'b\ star)\ set$
where *InternalFuns* = $\{X.\ \exists F.\ X = *fn*\ F\}$

8.1 Preamble - Pulling $\exists$ over $\forall$

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*: $\forall x.\ \exists y.\ Q\ x\ y \Longrightarrow \exists f :: 'a \Rightarrow nat.\ \forall x.\ Q\ x\ (f\ x)$
by (*rule exI* [**where** $x = \lambda x.\ LEAST\ y.\ Q\ x\ y$]) (*blast intro*: *LeastI*)

8.2 Properties of the Star-transform Applied to Sets of Reals

lemma *STAR-star-of-image-subset*: *star-of* ' $A \subseteq *s*\ A$
by *auto*

lemma *STAR-hypreal-of-real-Int*: $*s*\ X \cap \mathbb{R} =$ *hypreal-of-real* ' X
by (*auto simp add*: *SReal-def*)

lemma *STAR-star-of-Int*: $*s*\ X \cap Standard =$ *star-of* ' X
by (*auto simp add*: *Standard-def*)

lemma *lemma-not-hyprealA*: $x \notin$ *hypreal-of-real* ' $A \Longrightarrow \forall y \in A.\ x \neq$ *hypreal-of-real* y
by *auto*

lemma *lemma-not-starA*: $x \notin$ *star-of* ' $A \Longrightarrow \forall y \in A.\ x \neq$ *star-of* y
by *auto*

lemma *lemma-Compl-eq*: $-\{n.\ X\ n = xa\} = \{n.\ X\ n \neq xa\}$
by *auto*

lemma *STAR-real-seq-to-hypreal*: $\forall n.\ (X\ n) \notin M \Longrightarrow$ *star-n* $X \notin *s*\ M$
by (*simp add*: *starset-def star-of-def Iset-star-n FreeUltrafilterNat.proper*)

lemma *STAR-singleton*: $*s*\ \{x\} = \{$*star-of* $x\}$
by *simp*

lemma *STAR-not-mem*: $x \notin F \Longrightarrow \mathit{star\text{-}of}\ x \notin {*s*}\ F$
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*: $\forall n.\ F\ n = f \Longrightarrow {*fn*}\ F = {*f*}\ f$
apply (*drule fun-eq-iff* [*THEN iffD2*])
apply (*simp add*: *starfun-n-def starfun-def star-of-def*)
done

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*
apply (*simp add*: *is-starext-def*, *safe*)
apply (*rule-tac x*=*x* **in** *star-cases*)
apply (*rule-tac x*=*y* **in** *star-cases*)
apply (*unfold star-n-def*, *auto*)
apply (*rule bexI*, *rule-tac* [*2*] *lemma-starrel-refl*)
apply (*rule bexI*, *rule-tac* [*2*] *lemma-starrel-refl*)
apply (*fold star-n-def*)
apply (*unfold star-abs-def starfun-def star-of-def*)
apply (*simp add*: *Ifun-star-n star-n-eq-iff*)
done

Nonstandard extension of functions.

lemma *starfun*: $({*f*}\ f)\ (\mathit{star\text{-}n}\ X) = \mathit{star\text{-}n}\ (\lambda n.\ f\ (X\ n))$
by (*rule starfun-star-n*)

lemma *starfun-if-eq*: $\bigwedge w.\ w \neq \mathit{star\text{-}of}\ x \Longrightarrow ({*f*}\ (\lambda z.\ \mathit{if}\ z = x\ \mathit{then}\ a\ \mathit{else}\ g\ z))\ w = ({*f*}\ g)\ w$
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*)

This is trivial, given *starfun-Id*.

lemma *starfun-Idfun-approx*: $x \approx star\text{-}of\ a \Longrightarrow (*f* (\lambda x.\ x))\ x \approx star\text{-}of\ a$
by (*simp only*: *starfun-Id*)

The Star-function is a (nonstandard) extension of the function.

lemma *is-starext-starfun*: *is-starext* $(*f* f)\ f$
apply (*auto simp*: *is-starext-def*)
apply (*rule-tac* $x = x$ **in** *star-cases*)
apply (*rule-tac* $x = y$ **in** *star-cases*)
apply (*auto intro*!: *bexI* [*OF - Rep-star-star-n*] *simp*: *starfun star-n-eq-iff*)
done

Any nonstandard extension is in fact the Star-function.

lemma *is-starfun-starext*: *is-starext F f* $\Longrightarrow$ *F* = **f* f*
apply (*simp add*: *is-starext-def*)
apply (*rule ext*)
apply (*rule-tac x* = *x* **in** *star-cases*)
apply (*drule-tac x* = *x* **in** *spec*)
apply (*drule-tac x* = (**f* f*) *x* **in** *spec*)
apply (*auto simp add*: *starfun-star-n*)
apply (*simp add*: *star-n-eq-iff* [*symmetric*])
apply (*simp add*: *starfun-star-n* [*of f*, *symmetric*])
done

lemma *is-starext-starfun-iff*: *is-starext F f* $\longleftrightarrow$ *F* = **f* f*
by (*blast intro*: *is-starfun-starext is-starext-starfun*)

Extented 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*) $\approx$ *star-of* (*f a*)
by *simp*

Useful for NS definition of derivatives.

lemma *starfun-lambda-cancel*: $\bigwedge x'$. (**f** (λh. *f* (*x* + *h*))) *x'* = (**f* f*) (*star-of x* + *x'*)
by *transfer* (*rule refl*)

lemma *starfun-lambda-cancel2*: (**f** (λh. *f* (*g* (*x* + *h*)))) *x'* = (**f** (*f* $\circ$ *g*)) (*star-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** (λx. *f x* * *g x*)) *x* $\approx$ *l* * *m*
for *l m* :: *'a*::*real-normed-algebra star*
apply (*drule* (*3*) *approx-mult-HFinite*)
apply (*auto intro*: *approx-HFinite* [*OF - approx-sym*])
done

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*: $\bigwedge x. \; (*f* \; f) \; x : *s* \; A \Longrightarrow x \in *s* \; \{x. \; f \; x \in A\}$
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*: $*s* \; \{x. \; |x + - \; y| < r\} = \{x. \; |x + -star\text{-}of \; y| < star\text{-}of \; r\}$
by *transfer* (*rule refl*)

lemma *STAR-starfun-rabs-add-minus*:
$*s* \; \{x. \; |f \; x + - \; y| < r\} = \{x. \; |(*f* \; f) \; x + -star\text{-}of \; y| < star\text{-}of \; r\}$
by *transfer* (*rule refl*)

Another characterization of Infinitesimal and one of $\approx$ relation. In this theory since *hypreal-hrabs* proved here. Maybe move both theorems??

lemma *Infinitesimal-FreeUltrafilterNat-iff2*:
$star\text{-}n \; X \in Infinitesimal \longleftrightarrow (\forall m. \; eventually \; (\lambda n. \; norm \; (X \; n) < inverse \; (real \; (Suc \; m))) \; \mathcal{U})$
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*]:
n ∈ *HNatInfinite* ⟹ *inverse* (*hypreal-of-hypnat n*) ∈ *Infinitesimal*
apply (*cases n*)
apply (*auto simp*: *of-hypnat-def starfun-star-n star-n-inverse*
HNatInfinite-FreeUltrafilterNat-iff Infinitesimal-FreeUltrafilterNat-iff2)
apply (*drule-tac x* = *Suc m* **in** *spec*)
apply (*auto elim*!: *eventually-mono*)
done

lemma *approx-FreeUltrafilterNat-iff*:
star-n X ≈ *star-n Y* ⟷ (∀ *r*>*0*. *eventually* (λ*n*. *norm* (*X n* − *Y n*) < *r*) *U*)
apply (*subst approx-minus-iff*)
apply (*rule mem-infmal-iff* [*THEN subst*])
apply (*simp add*: *star-n-diff*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*)
done

lemma *approx-FreeUltrafilterNat-iff2*:
star-n X ≈ *star-n Y* ⟷ (∀ *m*. *eventually* (λ*n*. *norm* (*X n* − *Y n*) < *inverse* (*real* (*Suc m*))) *U*)
apply (*subst approx-minus-iff*)
apply (*rule mem-infmal-iff* [*THEN subst*])
apply (*simp add*: *star-n-diff*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff2*)
done

lemma *inj-starfun*: *inj starfun*
apply (*rule inj-onI*)
apply (*rule ext*, *rule ccontr*)
apply (*drule-tac x* = *star-n* (λ*n*. *xa*) **in** *fun-cong*)
apply (*auto simp add*: *starfun star-n-eq-iff FreeUltrafilterNat.proper*)
done

end

9 Star-transforms for the Hypernaturals

theory *NatStar*
imports *Star*
begin

lemma *star-n-eq-starfun-whn*: *star-n X* = (∗*f*∗ *X*) *whn*
by (*simp add*: *hypnat-omega-def starfun-def star-of-def Ifun-star-n*)

lemma *starset-n-Un*: ∗*sn*∗ (λ*n*. (*A n*) ∪ (*B n*)) = ∗*sn*∗ *A* ∪ ∗*sn*∗ *B*
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn Un-def*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Un*: $X \in \mathit{InternalSets} \Longrightarrow Y \in \mathit{InternalSets} \Longrightarrow X \cup Y \in \mathit{InternalSets}$
by (*auto simp add*: *InternalSets-def starset-n-Un* [*symmetric*])

lemma *starset-n-Int*: $*sn* (\lambda n.\ A\ n \cap B\ n) = *sn* A \cap *sn* B$
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn Int-def*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Int*: $X \in \mathit{InternalSets} \Longrightarrow Y \in \mathit{InternalSets} \Longrightarrow X \cap Y \in \mathit{InternalSets}$
by (*auto simp add*: *InternalSets-def starset-n-Int* [*symmetric*])

lemma *starset-n-Compl*: $*sn* ((\lambda n.\ -\ A\ n)) = -\ (\ *sn* A)$
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn Compl-eq*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Compl*: $X \in \mathit{InternalSets} \Longrightarrow -\ X \in \mathit{InternalSets}$
by (*auto simp add*: *InternalSets-def starset-n-Compl* [*symmetric*])

lemma *starset-n-diff*: $*sn* (\lambda n.\ (A\ n) - (B\ n)) = *sn* A - *sn* B$
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn set-diff-eq*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-diff*: $X \in \mathit{InternalSets} \Longrightarrow Y \in \mathit{InternalSets} \Longrightarrow X - Y \in \mathit{InternalSets}$
by (*auto simp add*: *InternalSets-def starset-n-diff* [*symmetric*])

lemma *NatStar-SHNat-subset*: $\mathit{Nats} \leq *s* (\mathit{UNIV}::\ \mathit{nat\ set})$
by *simp*

lemma *NatStar-hypreal-of-real-Int*: $*s* X\ \mathit{Int}\ \mathit{Nats} = \mathit{hypnat\text{-}of\text{-}nat}\ `\ X$
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*]: $(\ *s* X) \in \mathit{InternalSets}$
by (*auto simp add*: *InternalSets-def starset-starset-n-eq*)

lemma *InternalSets-UNIV-diff*: $X \in \mathit{InternalSets} \Longrightarrow \mathit{UNIV} - X \in \mathit{InternalSets}$
apply (*subgoal-tac UNIV* $-$ *X* $=$ $-$ *X*)
apply (*auto intro*: *InternalSets-Compl*)
done

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*: $\forall n.\ N \leq n \longrightarrow f\ n \leq g\ n \Longrightarrow$
$\forall n.\ hypnat\text{-}of\text{-}nat\ N \leq n \longrightarrow (\ *f*\ f)\ n \leq (\ *f*\ g)\ n$
by *transfer*

And another:

lemma *starfun-less-mono*:
$\forall n.\ N \leq n \longrightarrow f\ n < g\ n \Longrightarrow \forall n.\ hypnat\text{-}of\text{-}nat\ N \leq n \longrightarrow (\ *f*\ f)\ n < (\ *f*\ g)\ n$
by *transfer*

Nonstandard extension when we increment the argument by one.

lemma *starfun-shift-one*: $\bigwedge N.\ (\ *f*\ (\lambda n.\ f\ (Suc\ n)))\ N = (\ *f*\ f)\ (N + (1::hypnat))$
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*: $\bigwedge N.\ (\ *f*\ (\lambda n.\ r\ \hat{}\ n))\ N = hypreal\text{-}of\text{-}real\ r\ pow\ N$
by *transfer* (*rule refl*)

lemma *starfun-pow2*: $\bigwedge N.\ (\ *f*\ (\lambda n.\ X\ n\ \hat{}\ m))\ N = (\ *f*\ X)\ N\ pow\ hypnat\text{-}of\text{-}nat\ m$
by *transfer* (*rule refl*)

lemma *starfun-pow3*: $\bigwedge R.\ (\ *f*\ (\lambda r.\ r\ \hat{}\ n))\ R = R\ pow\ hypnat\text{-}of\text{-}nat\ n$
by *transfer* (*rule refl*)

The *hypreal-of-hypnat* function as a nonstandard extension of *real*.

lemma *starfunNat-real-of-nat*: $(\ *f*\ real) = hypreal\text{-}of\text{-}hypnat$
by *transfer* (*simp add*: *fun-eq-iff*)

lemma *starfun-inverse-real-of-nat-eq*:
$N \in HNatInfinite \Longrightarrow (\ *f*\ (\lambda x::nat.\ inverse\ (real\ x)))\ N = inverse\ (hypreal\text{-}of\text{-}hypnat\ N)$
apply (*rule-tac f1* $=$ *inverse* **in** *starfun-o2* [*THEN subst*])
apply (*subgoal-tac hypreal-of-hypnat* $N \neq 0$)
apply (*simp-all add*: *zero-less-HNatInfinite starfunNat-real-of-nat*)
done

Internal functions – some redundancy with $*f*$ now.

lemma *starfun-n*: $(\ *fn*\ f)\ (star\text{-}n\ X) = star\text{-}n\ (\lambda n.\ f\ n\ (X\ n))$

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*]:
$N \in HNatInfinite \Longrightarrow (*f* \; (\%x. \; inverse \; (real \; x))) \; N \in Infinitesimal$
apply (*rule-tac f1 = inverse* **in** *starfun-o2* [*THEN subst*])
apply (*subgoal-tac hypreal-of-hypnat N ~= 0*)
apply (*simp-all add*: *zero-less-HNatInfinite starfunNat-real-of-nat*)
done

9.2 Nonstandard Characterization of Induction

lemma *hypnat-induct-obj*:
$\bigwedge n. \; ((*p* \; P) \; (0::hypnat) \wedge (\forall n. \; (*p* \; P) \; n \longrightarrow (*p* \; P) \; (n + 1))) \longrightarrow (*p* \; P) \; n$
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*: $(*p2* \; (op =)) = (op =)$
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-nat-set-Least-mem*: $c \in S \Longrightarrow (LEAST\ n.\ n \in S) \in S$
for S :: *nat set*
by (*erule LeastI*)

lemma *nonempty-set-star-has-least*:
$\bigwedge S$::*nat set star*. *Iset* $S \neq \{\} \Longrightarrow \exists\, n \in$ *Iset* $S.\ \forall\, m \in$ *Iset* $S.\ n \leq m$
apply (*transfer empty-def*)
apply (*rule-tac* x=*LEAST* $n.\ n \in S$ **in** *bexI*)
apply (*simp add*: *Least-le*)
apply (*rule LeastI-ex*, *auto*)
done

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*
apply (*clarsimp simp add*: *InternalSets-def starset-n-def*)
apply (*erule nonempty-set-star-has-least*)
done

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 *HOL.Limits NatStar*
abbrevs $-\!\!-\!\!-\!\!> = \longrightarrow_{NS}$
begin

definition *NSLIMSEQ* :: (*nat* ⇒ *'a*::*real-normed-vector*) ⇒ *'a* ⇒ *bool*
(((-)/ $\longrightarrow_{NS}$ (-)) [*60*, *60*] *60*) **where**
— Nonstandard definition of convergence of sequence
X $\longrightarrow_{NS}$ *L* ⟷ (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ≈ *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* ⟷ (∃ *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-iff*: (*X* $\longrightarrow_{NS}$ *L*) ⟷ (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ≈ *star-of L*)
by (*simp add*: *NSLIMSEQ-def*)

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*: (λ*n*. *k*) $\longrightarrow_{NS}$ *k*
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-add*: *X* $\longrightarrow_{NS}$ *a* ⟹ *Y* $\longrightarrow_{NS}$ *b* ⟹ (λ*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* ⟹ (λ*n*. *f n* + *b*) $\longrightarrow_{NS}$ *a* + *b*
by (*simp only*: *NSLIMSEQ-add NSLIMSEQ-const*)

lemma *NSLIMSEQ-mult*: *X* $\longrightarrow_{NS}$ *a* ⟹ *Y* $\longrightarrow_{NS}$ *b* ⟹ (λ*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-add-minus*: $X \longrightarrow_{NS} a \Longrightarrow Y \longrightarrow_{NS} b \Longrightarrow (\lambda n.\ X\ n\ +\ -\ Y\ n) \longrightarrow_{NS} a\ +\ -\ b$
by (*simp add*: *NSLIMSEQ-diff*)

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*: $X \longrightarrow_{NS} a \Longrightarrow X \longrightarrow_{NS} b \Longrightarrow a = b$
apply (*simp add*: *NSLIMSEQ-def*)
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])+
apply (*auto dest*: *approx-trans3*)
done

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 $l\ m :: real$
apply (*simp add*: *NSLIMSEQ-def*, *safe*)
apply (*drule starfun-le-mono*)
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])+
apply (*drule-tac* $x = whn$ **in** *spec*)
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*])+
apply *clarify*
apply (*auto intro*: *hypreal-of-real-le-add-Infininitesimal-cancel2*)
done

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*: $f \longrightarrow_{NS} l \Longrightarrow (\lambda n.\ f(Suc\ n)) \longrightarrow_{NS} l$
apply (*unfold NSLIMSEQ-def*)
apply *safe*
apply (*drule-tac* $x = N + 1$ **in** *bspec*)
apply (*erule HNatInfinite-add*)
apply (*simp add*: *starfun-shift-one*)
done

lemma *NSLIMSEQ-imp-Suc*: $(\lambda n.\ f(Suc\ n)) \longrightarrow_{NS} l \Longrightarrow f \longrightarrow_{NS} l$
apply (*unfold NSLIMSEQ-def*)
apply *safe*
apply (*drule-tac* $x = N - 1$ **in** *bspec*)
apply (*erule Nats-1* [*THEN* [*2*] *HNatInfinite-diff*])
apply (*simp add*: *starfun-shift-one one-le-HNatInfinite*)
done

lemma *NSLIMSEQ-Suc-iff*: $(\lambda n.\ f\ (Suc\ n)) \longrightarrow_{NS} l \longleftrightarrow f \longrightarrow_{NS} l$
by (*blast intro*: *NSLIMSEQ-imp-Suc NSLIMSEQ-Suc*)

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* ∈ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix r :: *real*
assume r: $0 < r$
from *LIMSEQ-D* [*OF X r*] **obtain** *no* **where** ∀ *n*≥*no*. *norm* (*X n − L*) < *r* **..**
then have ∀ *n*≥*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* ≈ *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*
apply (*cases* $N \in$ *HNatInfinite*)
apply (*erule* (*1*) *NSBseqD*)
apply (*rule subsetD* [*OF Standard-subset-HFinite*])
apply (*simp add*: *HNatInfinite-def Nats-eq-Standard*)
done

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*: $N \in$ *HNatInfinite*
from *BseqD* [*OF X*] **obtain** *K* **where** $\forall n.\ norm\ (X\ n) \leq K$
by *fast*
then have $\forall N.$ *hnorm* (*starfun X N*) $\leq$ *star-of K*
by *transfer*
then have *hnorm* (*starfun X N*) $\leq$ *star-of K*
by *simp*
also have *star-of K* $<$ *star-of* $(K + 1)$
by *simp*
finally have $\exists x \in$*Reals*. *hnorm* (*starfun X N*) $< x$
by (*rule bexI*) *simp*
then show *starfun X N* $\in$ *HFinite*
by (*simp add*: *HFinite-def*)
qed

The nonstandard definition implies the standard definition.

lemma *SReal-less-omega*: $r \in \mathbb{R} \Longrightarrow r < \omega$
using *HInfinite-omega*
by (*simp add*: *HInfinite-def*) (*simp add*: *order-less-imp-le*)

lemma *NSBseq-Bseq*: *NSBseq X* $\Longrightarrow$ *Bseq X*
proof (*rule ccontr*)
let *?n* = $\lambda K.$ *LEAST n. K* < *norm* (*X n*)
assume *NSBseq X*
then have *finite*: ($*f*$ *X*) (($*f*$ *?n*) ω) $\in$ *HFinite*
by (*rule NSBseqD2*)
assume $\neg$ *Bseq X*
then have $\forall K{>}0.\ \exists n.\ K <$ *norm* (*X n*)
by (*simp add*: *Bseq-def linorder-not-le*)
then have $\forall K{>}0.$ *K* < *norm* (*X* (*?n K*))
by (*auto intro*: *LeastI-ex*)
then have $\forall K{>}0.$ *K* < *hnorm* (($*f*$ *X*) (($*f*$ *?n*) *K*))
by *transfer*
then have $\omega <$ *hnorm* (($*f*$ *X*) (($*f*$ *?n*) ω))
by *simp*
then have $\forall r{\in}\mathbb{R}.\ r <$ *hnorm* (($*f*$ *X*) (($*f*$ *?n*) ω))
by (*simp add*: *order-less-trans* [*OF SReal-less-omega*])
then have ($*f*$ *X*) (($*f*$ *?n*) ω) $\in$ *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*: $\forall n \geq m.\ X\ n = X\ m \Longrightarrow \exists L.\ X \longrightarrow_{NS} L$
by (*auto dest*!: *Bmonoseq-LIMSEQ simp add*: *LIMSEQ-NSLIMSEQ-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 $\Longrightarrow M \in HNatInfinite \Longrightarrow N \in HNatInfinite \Longrightarrow starfun\ X\ M \approx starfun\ X\ N$
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*: $M \in HNatInfinite$
fix *N*
assume *N*: $N \in HNatInfinite$
have $starfun\ X\ M - starfun\ X\ N \in Infinitesimal$
proof (*rule InfinitesimalI2*)
fix *r* :: *real*
assume *r*: $0 < r$
from *CauchyD* [*OF X r*] **obtain** *k* **where** $\forall m \geq k.\ \forall n \geq k.\ norm\ (X\ m - X\ n) < r$ **..**
then have $\forall m \geq star\text{-}of\ k.\ \forall n \geq star\text{-}of\ k.\ hnorm\ (starfun\ X\ m - starfun\ X\ n) < star\text{-}of\ r$

```
    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 stan-

dard 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* $\Longrightarrow$ *NSCauchy X*
by (*simp add*: *NSconvergent-def NSLIMSEQ-def NSCauchy-def*) (*auto intro*: *approx-trans2*)

lemma *real-NSCauchy-NSconvergent*: *NSCauchy X* $\Longrightarrow$ *NSconvergent X*
for *X* :: *nat* $\Rightarrow$ *real*
apply (*simp add*: *NSconvergent-def NSLIMSEQ-def*)
apply (*frule NSCauchy-NSBseq*)
apply (*simp add*: *NSBseq-def NSCauchy-def*)
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])
apply (*auto dest*!: *st-part-Ex simp add*: *SReal-iff*)
apply (*blast intro*: *approx-trans3*)
done

lemma *NSCauchy-NSconvergent*: *NSCauchy X* $\Longrightarrow$ *NSconvergent X*
for *X* :: *nat* $\Rightarrow$ *'a*::*banach*
apply (*drule NSCauchy-Cauchy* [*THEN Cauchy-convergent*])
apply (*erule convergent-NSconvergent-iff* [*THEN iffD1*])
done

lemma *NSCauchy-NSconvergent-iff*: *NSCauchy X* = *NSconvergent X*
for *X* :: *nat* $\Rightarrow$ *'a*::*banach*
by (*fast intro*: *NSCauchy-NSconvergent NSconvergent-NSCauchy*)

10.5 Power Sequences

The sequence x^n tends to 0 if $(0::'a) \leq x$ and $x < (1::'a)$. 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*: *0* $\leq$ *x* $\Longrightarrow$ *x* $<$ *1* $\Longrightarrow$ $(\lambda n.\ x\ \hat{}\ n) \longrightarrow_{NS} 0$
for *x* :: *real*
apply (*simp add*: *NSLIMSEQ-def*)
apply (*auto dest*!: *convergent-realpow simp add*: *convergent-NSconvergent-iff*)
apply (*frule NSconvergentD*)
apply (*auto simp add*: *NSLIMSEQ-def NSCauchy-NSconvergent-iff* [*symmetric*] *NSCauchy-def starfun-pow*)
apply (*frule HNatInfinite-add-one*)
apply (*drule bspec*, *assumption*)
apply (*drule bspec*, *assumption*)

apply (*drule-tac* $x = N + 1$ **in** *bspec*, *assumption*)
apply (*simp add*: *hyperpow-add*)
apply (*drule approx-mult-subst-star-of*, *assumption*)
apply (*drule approx-trans3*, *assumption*)
apply (*auto simp del*: *star-of-mult simp add*: *star-of-mult* [*symmetric*])
done

lemma *NSLIMSEQ-rabs-realpow-zero*: $|c| < 1 \Longrightarrow (\lambda n.\ |c|\ \hat{}\ n) \longrightarrow_{NS} 0$
for c :: *real*
by (*simp add*: *LIMSEQ-rabs-realpow-zero LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

lemma *NSLIMSEQ-rabs-realpow-zero2*: $|c| < 1 \Longrightarrow (\lambda n.\ c\ \hat{}\ n) \longrightarrow_{NS} 0$
for c :: *real*
by (*simp add*: *LIMSEQ-rabs-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*:

$\bigwedge m\ n.\ sumhr\ (m,\ n\ +\ 1,\ f) = (if\ n\ +\ 1\ \leq\ m\ then\ 0\ else\ sumhr\ (m,\ n,\ f)\ +\ (\ *f*\ f)\ n)$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-Suc-zero* [*simp*]: $\bigwedge n.\ sumhr\ (n\ +\ 1,\ n,\ f) = 0$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-eq-bounds* [*simp*]: $\bigwedge n.\ sumhr\ (n,\ n,\ f) = 0$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-Suc* [*simp*]: $\bigwedge m.\ sumhr\ (m,\ m\ +\ 1,\ f) = (\ *f*\ f)\ m$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-add-lbound-zero* [*simp*]: $\bigwedge k\ m.\ sumhr\ (m\ +\ k,\ k,\ f) = 0$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-add*: $\bigwedge m\ n.\ sumhr\ (m,\ n,\ f)\ +\ sumhr\ (m,\ n,\ g) = sumhr\ (m,\ n,\ \lambda i.\ f\ i\ +\ g\ i)$
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-add-nat-ivl*)

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*: $\bigwedge m\ n.\ |sumhr\ (m,\ n,\ f)| \leq sumhr\ (m,\ n,\ \lambda i.\ |f\ i|)$
unfolding *sumhr-app* **by** *transfer* (*rule sum-abs*)

Other general version also needed.

lemma *sumhr-fun-hypnat-eq*:
$(\forall r.\ m \leq r \wedge r < n \longrightarrow f\ r = g\ r) \longrightarrow$
sumhr (*hypnat-of-nat m*, *hypnat-of-nat n*, *f*) =
sumhr (*hypnat-of-nat m*, *hypnat-of-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 *sumhr-hypreal-omega-minus-one*: $sumhr(0,\ whn,\ \lambda i.\ 1) = \omega\ -\ 1$
apply (*simp add*: *sumhr-const*)

apply (*unfold star-class-defs omega-def hypnat-omega-def of-hypnat-def star-of-def*)
apply (*simp add*: *starfun-star-n starfun2-star-n*)
done

lemma *sumhr-minus-one-realpow-zero* [*simp*]: $\bigwedge N.\ sumhr\ (0,\ N\ +\ N,\ \lambda i.\ (-1)\ \hat{}\ (i\ +\ 1)) = 0$
unfolding *sumhr-app*
apply *transfer*
apply (*simp del*: *power-Suc add*: *mult-2* [*symmetric*])
apply (*induct-tac N*)
apply *simp-all*
done

lemma *sumhr-interval-const*:
$(\forall n.\ m \le Suc\ n \longrightarrow f\ n = r) \wedge m \le na \Longrightarrow$
$sumhr\ (hypnat\text{-}of\text{-}nat\ m,\ hypnat\text{-}of\text{-}nat\ na,\ f) = hypreal\text{-}of\text{-}nat\ (na\ -\ m)\ *\ hypreal\text{-}of\text{-}real\ r$
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** $y = N$]
apply *auto*
apply (*drule approx-sym* [*THEN approx-minus-iff* [*THEN iffD1*]])
apply (*auto dest*: *approx-hrabs simp add*: *sumhr-split-diff*)
done

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)$| \approx 0$)
apply (*auto simp add*: *summable-NSsummable-iff* [*symmetric*]
summable-iff-convergent convergent-NSconvergent-iff atLeast0LessThan[*symmetric*]
NSCauchy-NSconvergent-iff [*symmetric*] *NSCauchy-def starfunNat-sumr*)
apply (*cut-tac* $x = M$ **and** $y = N$ **in** *linorder-less-linear*)
apply *auto*
apply (*rule approx-minus-iff* [*THEN iffD2*, *THEN approx-sym*])
apply (*rule-tac* [*2*] *approx-minus-iff* [*THEN iffD2*])
apply (*auto dest*: *approx-hrabs-zero-cancel simp*: *sumhr-split-diff atLeast0LessThan*[*symmetric*])
done

Terms of a convergent series tend to zero.

lemma *NSsummable-NSLIMSEQ-zero*: *NSsummable* $f \Longrightarrow f \longrightarrow_{NS} 0$
apply (*auto simp add*: *NSLIMSEQ-def NSsummable-NSCauchy*)
apply (*drule bspec*)
apply *auto*
apply (*drule-tac* $x = N + 1$ **in** *bspec*)
apply (*auto intro*: *HNatInfinite-add-one approx-hrabs-zero-cancel*)
done

Nonstandard comparison test.

lemma *NSsummable-comparison-test*: $\exists\, N.\ \forall\, n.\ N \leq n \longrightarrow |f\ n| \leq g\ n \Longrightarrow$ *NSsummable* $g \Longrightarrow$ *NSsummable* f
apply (*fold summable-NSsummable-iff*)

apply (*rule summable-comparison-test*, *simp*, *assumption*)
done

lemma *NSsummable-rabs-comparison-test*:
$\exists N.\ \forall n.\ N \leq n \longrightarrow |f\ n| \leq g\ n \Longrightarrow NSsummable\ g \Longrightarrow NSsummable\ (\lambda k.\ |f\ k|)$
by (*rule NSsummable-comparison-test*) *auto*

end

12 Limits and Continuity (Nonstandard)

theory *HLim*
imports *Star*
abbrevs $---> = -\square\!\!\rightarrow_{NS}$
begin

Nonstandard Definitions.

definition *NSLIM* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ '*a* ⇒ '*b* ⇒ *bool*
(((-)/ −(-)/→$_{NS}$ (-)) [*60*, *0*, *60*] *60*)
where $f\ -a\rightarrow_{NS}\ L \longleftrightarrow (\forall x.\ x \neq star\text{-}of\ a \wedge x \approx star\text{-}of\ a \longrightarrow (\ *f*\ f)\ x \approx star\text{-}of\ L)$

definition *isNSCont* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ '*a* ⇒ *bool*
where — NS definition dispenses with limit notions
$isNSCont\ f\ a \longleftrightarrow (\forall y.\ y \approx star\text{-}of\ a \longrightarrow (\ *f*\ f)\ y \approx star\text{-}of\ (f\ a))$

definition *isNSUCont* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ *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*: $f\ -x\to_{NS}\ l \Longrightarrow g\ -x\to_{NS}\ m \Longrightarrow (\lambda x.\ f\ x *_R g\ x)\ -x\to_{NS}\ (l *_R m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-scaleR-HFinite*)

lemma *NSLIM-add*: $f\ -x\to_{NS}\ l \Longrightarrow g\ -x\to_{NS}\ m \Longrightarrow (\lambda x.\ f\ x + g\ x)\ -x\to_{NS}\ (l + m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-add*)

lemma *NSLIM-const* [*simp*]: $(\lambda x.\ k)\ -x\to_{NS}\ k$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-minus*: $f\ -a\to_{NS}\ L \Longrightarrow (\lambda x.\ -f\ x)\ -a\to_{NS}\ -L$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-diff*: $f\ -x\to_{NS}\ l \Longrightarrow g\ -x\to_{NS}\ m \Longrightarrow (\lambda x.\ f\ x - g\ x)\ -x\to_{NS}\ (l - m)$
by (*simp only*: *NSLIM-add NSLIM-minus diff-conv-add-uminus*)

lemma *NSLIM-add-minus*: $f\ -x\to_{NS}\ l \Longrightarrow g\ -x\to_{NS}\ m \Longrightarrow (\lambda x.\ f\ x + -\ g\ x)\ -x\to_{NS}\ (l + -m)$
by (*simp only*: *NSLIM-add NSLIM-minus*)

lemma *NSLIM-inverse*: $f\ -a\to_{NS}\ L \Longrightarrow L \neq 0 \Longrightarrow (\lambda x.\ inverse\ (f\ x))\ -a\to_{NS}\ (inverse\ L)$
for $L :: 'a$::*real-normed-div-algebra*
apply (*simp add*: *NSLIM-def*, *clarify*)
apply (*drule spec*)
apply (*auto simp add*: *star-of-approx-inverse*)
done

lemma *NSLIM-zero*:
assumes *f*: $f\ -a\to_{NS}\ l$
shows $(\lambda x.\ f(x) - l)\ -a\to_{NS}\ 0$
proof −
have $(\lambda x.\ f\ x - l)\ -a\to_{NS}\ l - l$
by (*rule NSLIM-diff* [*OF f NSLIM-const*])
then show *?thesis* **by** *simp*
qed

lemma *NSLIM-zero-cancel*: $(\lambda x.\ f\ x - l)\ -x\to_{NS}\ 0 \Longrightarrow f\ -x\to_{NS}\ l$
apply (*drule-tac* $g = \lambda x.\ l$ **and** $m = l$ **in** *NSLIM-add*)
apply (*auto simp add*: *add.assoc*)
done

lemma *NSLIM-const-not-eq*: $k \neq L \Longrightarrow \neg\ (\lambda x.\ k)\ -a\to_{NS}\ L$

for $a :: 'a$::*real-normed-algebra-1*
apply (*simp add*: *NSLIM-def*)
apply (*rule-tac x*=*star-of a* + *of-hypreal* ε **in** *exI*)
apply (*simp add*: *hypreal-epsilon-not-zero approx-def*)
done

lemma *NSLIM-not-zero*: $k \neq 0 \Longrightarrow \neg\ (\lambda x.\ k)\ -a\rightarrow_{NS}\ 0$
for $a :: 'a$::*real-normed-algebra-1*
by (*rule NSLIM-const-not-eq*)

lemma *NSLIM-const-eq*: $(\lambda x.\ k)\ -a\rightarrow_{NS}\ L \Longrightarrow k = L$
for $a :: 'a$::*real-normed-algebra-1*
by (*rule ccontr*) (*blast dest*: *NSLIM-const-not-eq*)

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* $\Rightarrow 'b$::*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* ∈ *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* ∈ *Infinitesimal*
by (*simp only*: *approx-def*)

then have *hnorm* $(x - \text{star-of } a) < \text{star-of } s$
using s **by** (*rule InfinitesimalD2*)
with *neq* **show** *hnorm* $(\text{starfun } f\ x - \text{star-of } L) < \text{star-of } r$
by (*rule less-r'*)
qed
then show $\text{starfun } f\ x \approx \text{star-of } L$
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 $\exists\, s{>}0.\ \forall x.\ x \neq \text{star-of } a \wedge \text{hnorm } (x - \text{star-of } a) < s \longrightarrow$
hnorm $(\text{starfun } f\ x - \text{star-of } L) < \text{star-of } r$
proof (*rule exI*, *safe*)
show $0 < \varepsilon$
by (*rule hypreal-epsilon-gt-zero*)
next
fix x
assume *neq*: $x \neq \text{star-of } a$
assume *hnorm* $(x - \text{star-of } a) < \varepsilon$
with *Infinitesimal-epsilon* **have** $x - \text{star-of } a \in \text{Infinitesimal}$
by (*rule hnorm-less-Infinitesimal*)
then have $x \approx \text{star-of } a$
by (*unfold approx-def*)
with f *neq* **have** $\text{starfun } f\ x \approx \text{star-of } L$
by (*rule NSLIM-D*)
then have $\text{starfun } f\ x - \text{star-of } L \in \text{Infinitesimal}$
by (*unfold approx-def*)
then show *hnorm* $(\text{starfun } f\ x - \text{star-of } L) < \text{star-of } r$
using r **by** (*rule InfinitesimalD2*)
qed
then show $\exists\, s{>}0.\ \forall x.\ x \neq a \wedge \text{norm } (x - a) < s \longrightarrow \text{norm } (f\ x - L) < r$
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*: $\text{isNSCont } f\ a \Longrightarrow y \approx \text{star-of } a \Longrightarrow (\ {*f*}\ f)\ y \approx \text{star-of } (f\ a)$
by (*simp add*: *isNSCont-def*)

lemma *isNSCont-NSLIM*: $\text{isNSCont } f\ a \Longrightarrow f\ -a\rightarrow_{NS}\ (f\ a)$

by (*simp add*: *isNSCont-def NSLIM-def*)

lemma *NSLIM-isNSCont*: $f\ -a\rightarrow_{NS}\ (f\ a) \Longrightarrow isNSCont\ f\ a$
apply (*auto simp add*: *isNSCont-def NSLIM-def*)
apply (*case-tac* $y = star\text{-}of\ a$)
apply *auto*
done

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-h-iff*: $f\ -a\rightarrow_{NS}\ L \longleftrightarrow (\lambda h.\ f\ (a + h))\ -0\rightarrow_{NS}\ L$
apply (*simp add*: *NSLIM-def*, *auto*)
apply (*drule-tac* $x = star\text{-}of\ a + x$ **in** *spec*)
apply (*drule-tac* [2] $x = -\ star\text{-}of\ a + x$ **in** *spec*, *safe*, *simp*)
apply (*erule mem-infmal-iff* [*THEN iffD2*, *THEN Infinitesimal-add-approx-self* [*THEN approx-sym*]])
apply (*erule-tac* [3] *approx-minus-iff2* [*THEN iffD1*])
prefer 2 **apply** (*simp add*: *add.commute*)
apply (*rule-tac* $x = x$ **in** *star-cases*)
apply (*rule-tac* [2] $x = x$ **in** *star-cases*)
apply (*auto simp add*: *starfun star-of-def star-n-minus star-n-add add.assoc star-n-zero-num*)
done

lemma *NSLIM-isCont-iff*: $f\ -a\rightarrow_{NS}\ f\ a \longleftrightarrow (\lambda h.\ f\ (a + h))\ -0\rightarrow_{NS}\ f\ a$
by (*fact NSLIM-h-iff*)

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* $\Rightarrow$ *'b*::*real-normed-div-algebra*
by (*auto intro*: *isCont-inverse simp add*: *isNSCont-isCont-iff*)

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* $\Rightarrow$ *'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* $\in$ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix r :: *real*
assume *r*: $0 < r$
with *f* **obtain** s **where** *s*: $0 < s$
and *less-r*: $\bigwedge x\ y.$ *norm* $(x - y) < s \Longrightarrow$ *norm* $(f\ x - f\ y) < r$
by (*auto simp add*: *isUCont-def dist-norm*)
from *less-r* **have** *less-r'*:
$\bigwedge x\ y.$ *hnorm* $(x - y) <$ *star-of s* $\Longrightarrow$ *hnorm* $($*starfun f x − starfun f y*$) <$ *star-of r*
by *transfer*
from *approx* **have** $x - y \in$ *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* < *ε*
by (*rule hypreal-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* ≈ *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*
((*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 *hnorm-of-hypreal*: ⋀*r*. *hnorm* ((∗*f*∗ *of-real*) *r*::'*a*::*real-normed-div-algebra star*) = |*r*|
by *transfer* (*rule norm-of-real*)

lemma *Infinitesimal-of-hypreal*:
x ∈ *Infinitesimal* ⟹ ((∗*f*∗ *of-real*) *x*::'*a*::*real-normed-div-algebra star*) ∈ *Infinitesimal*
apply (*rule InfinitesimalI2*)
apply (*drule* (*1*) *InfinitesimalD2*)
apply (*simp add*: *hnorm-of-hypreal*)
done

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*: *NSDERIV f x* :> *D* ⟹ *NSDERIV f x* :> *E* ⟹ *D* = *E*
apply (*subgoal-tac* (∗*f*∗ *of-real*) ε ∈ *Infinitesimal* − {*0*::'*a star*})
apply (*simp only*: *nsderiv-def*)
apply (*drule* (*1*) *bspec*)+
apply (*drule* (*1*) *approx-trans3*)
apply *simp*
apply (*simp add*: *Infinitesimal-of-hypreal*)
apply (*simp add*: *of-hypreal-eq-0-iff hypreal-epsilon-not-zero*)
done

First *NSDERIV* in terms of *NSLIM*.

First equivalence.

lemma *NSDERIV-NSLIM-iff*: (*NSDERIV f x* :> *D*) ⟷ (λ*h*. (*f* (*x* + *h*) − *f x*)

/ *h*) $-0\rightarrow_{NS}$ *D*
apply (*auto simp add*: *nsderiv-def NSLIM-def*)
apply (*drule-tac x* = *xa* **in** *bspec*)
apply (*rule-tac* [*3*] *ccontr*)
apply (*drule-tac* [*3*] *x* = *h* **in** *spec*)
apply (*auto simp add*: *mem-infmal-iff starfun-lambda-cancel*)
done

Second equivalence.

lemma *NSDERIV-NSLIM-iff2*: (*NSDERIV f x* :> *D*) $\longleftrightarrow$ ($\lambda z.$ (*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* ≠ *star-of x* ∧ *w* ≈ *star-of x* $\longrightarrow$ (∗*f*∗ ($\lambda z.$ (*f z* − *f x*) / (*z* − *x*))) *w* ≈ *star-of D*)
by (*simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def*)

lemma *hypreal-not-eq-minus-iff*: *x* ≠ *a* $\longleftrightarrow$ *x* − *a* ≠ (*0*::'*a*::*ab-group-add*)
by *auto*

lemma *NSDERIVD5*:
(*NSDERIV f x* :> *D*) $\Longrightarrow$
($\forall u.$ *u* ≈ *hypreal-of-real x* $\longrightarrow$
(∗*f*∗ ($\lambda z.$ *f z* − *f x*)) *u* ≈ *hypreal-of-real D* ∗ (*u* − *hypreal-of-real x*))
apply (*auto simp add*: *NSDERIV-iff2*)
apply (*case-tac u* = *hypreal-of-real x*, *auto*)
apply (*drule-tac x* = *u* **in** *spec*, *auto*)
apply (*drule-tac c* = *u* − *hypreal-of-real x* **and** *b* = *hypreal-of-real D* **in** *approx-mult1*)
apply (*drule-tac* [!] *hypreal-not-eq-minus-iff* [*THEN iffD1*])
apply (*subgoal-tac* [*2*] (∗*f*∗ ($\lambda z.$ *z* − *x*)) *u* ≠ (*0*::*hypreal*))
apply (*auto simp*: *approx-minus-iff* [*THEN iffD1*, *THEN mem-infmal-iff* [*THEN iffD2*]]
Infinitesimal-subset-HFinite [*THEN subsetD*])
done

lemma *NSDERIVD4*:
(*NSDERIV f x* :> *D*) $\Longrightarrow$
($\forall h \in$ *Infinitesimal*.
(∗*f*∗ *f*)(*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*) ≈ *hypreal-of-real D* ∗ *h*)
apply (*auto simp add*: *nsderiv-def*)
apply (*case-tac h* = *0*)
apply *auto*
apply (*drule-tac x* = *h* **in** *bspec*)
apply (*drule-tac* [*2*] *c* = *h* **in** *approx-mult1*)

apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

lemma *NSDERIVD3*:
(*NSDERIV f x :> D*) ⟹
∀ *h* ∈ *Infinitesimal* − {*0*}.
((∗*f*∗ *f*) (*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*)) ≈ *hypreal-of-real D* ∗ *h*
apply (*auto simp add*: *nsderiv-def*)
apply (*rule ccontr*, *drule-tac x* = *h* **in** *bspec*)
apply (*drule-tac* [*2*] *c* = *h* **in** *approx-mult1*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*] *simp add*: *mult.assoc*)
done

Differentiability implies continuity nice and simple "algebraic" proof.

lemma *NSDERIV-isNSCont*: *NSDERIV f x :> D* ⟹ *isNSCont f x*
apply (*auto simp add*: *nsderiv-def isNSCont-NSLIM-iff NSLIM-def*)
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*drule hypreal-not-eq-minus-iff* [*THEN iffD1*])
apply (*drule-tac x* = *xa* − *star-of x* **in** *bspec*)
prefer *2* **apply** (*simp add*: *add.assoc* [*symmetric*])
apply (*auto simp add*: *mem-infmal-iff* [*symmetric*] *add.commute*)
apply (*drule-tac c* = *xa* − *star-of x* **in** *approx-mult1*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*] *simp add*: *mult.assoc*)
apply (*drule-tac x3*=*D* **in**
HFinite-star-of [*THEN* [*2*] *Infinitesimal-HFinite-mult*, *THEN mem-infmal-iff* [*THEN iffD1*]])
apply (*auto simp add*: *mult.commute intro*: *approx-trans approx-minus-iff* [*THEN iffD2*])
done

Differentiation rules for combinations of functions follow from clear, straightforard, algebraic manipulations.

Constant function.

lemma *NSDERIV-const* [*simp*]: *NSDERIV* (λ*x*. *k*) *x :> 0*
by (*simp add*: *NSDERIV-NSLIM-iff*)

Sum of functions- proved easily.

lemma *NSDERIV-add*:
NSDERIV f x :> Da ⟹ *NSDERIV g x :> Db* ⟹ *NSDERIV* (λ*x*. *f x* + *g x*) *x :> Da* + *Db*
apply (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def*)
apply (*auto simp add*: *add-divide-distrib diff-divide-distrib dest*!: *spec*)
apply (*drule-tac b* = *star-of Da* **and** *d* = *star-of Db* **in** *approx-add*)
apply (*auto simp add*: *ac-simps algebra-simps*)
done

Product of functions - Proof is trivial but tedious and long due to rearrangement of terms.

lemma *lemma-nsderiv1*: $(a * b) - (c * d) = (b * (a - c)) + (c * (b - d))$
for *a b c d* :: *'a::comm-ring star*
by (*simp add*: *right-diff-distrib ac-simps*)

lemma *lemma-nsderiv2*: $(x - y) / z = star\text{-}of\ D + yb \Longrightarrow z \neq 0 \Longrightarrow$
$z \in Infinitesimal \Longrightarrow yb \in Infinitesimal \Longrightarrow x - y \approx 0$
for *x y z* :: *'a::real-normed-field star*
apply (*simp add*: *nonzero-divide-eq-eq*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult2 HFinite-add*
simp add: *mult.assoc mem-infmal-iff* [*symmetric*])
apply (*erule Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

lemma *NSDERIV-mult*:
$NSDERIV\ f\ x :> Da \Longrightarrow NSDERIV\ g\ x :> Db \Longrightarrow$
$NSDERIV\ (\lambda x.\ f\ x * g\ x)\ x :> (Da * g\ x) + (Db * f\ x)$
apply (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def*)
apply (*auto dest*!: *spec simp add*: *starfun-lambda-cancel lemma-nsderiv1*)
apply (*simp* (*no-asm*) *add*: *add-divide-distrib diff-divide-distrib*)
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*])+
apply (*auto simp add*: *times-divide-eq-right* [*symmetric*]
simp del: *times-divide-eq-right times-divide-eq-left*)
apply (*drule-tac* $D = Db$ **in** *lemma-nsderiv2*, *assumption*+)
apply (*drule-tac approx-minus-iff* [*THEN iffD2*, *THEN bex-Infinitesimal-iff2*
[*THEN iffD2*]])
apply (*auto intro*!: *approx-add-mono1 simp*: *distrib-right distrib-left mult.commute*
add.assoc)
apply (*rule-tac b1* = *star-of Db* * *star-of* (*f x*) **in** *add.commute* [*THEN subst*])
apply (*auto intro*!: *Infinitesimal-add-approx-self2* [*THEN approx-sym*]
Infinitesimal-add Infinitesimal-mult Infinitesimal-star-of-mult Infinitesimal-star-of-mult2
simp add: *add.assoc* [*symmetric*])
done

Multiplying by a constant.

lemma *NSDERIV-cmult*: $NSDERIV\ f\ x :> D \Longrightarrow NSDERIV\ (\lambda x.\ c * f\ x)\ x :>$
$c * D$
apply (*simp only*: *times-divide-eq-right* [*symmetric*] *NSDERIV-NSLIM-iff*
minus-mult-right right-diff-distrib [*symmetric*])
apply (*erule NSLIM-const* [*THEN NSLIM-mult*])
done

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 $(\lambda h.\ (f\ (x + h) - f\ x) / h) -0\to_{NS} D$

then have *deriv*: $(\lambda h.\ -\ ((f(x+h)\ -\ f\ x)\ /\ h))\ -0\to_{NS}\ -\ D$
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** $(\lambda h.\ (-\ f\ (x\ +\ h)\ +\ f\ x)\ /\ h)\ -0\to_{NS}\ -\ D$
by *simp*
then show $(\lambda h.\ (f\ (x\ +\ h)\ -\ f\ x)\ /\ h)\ -0\to_{NS}\ D \Longrightarrow (\lambda h.\ (f\ x\ -\ f\ (x\ +\ h))\ /\ h)\ -0\to_{NS}\ -\ D$
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*:
$NSDERIV\ g\ x\ :>\ D \Longrightarrow (\ *f*\ g)\ (star\text{-}of\ x\ +\ xa) = star\text{-}of\ (g\ x) \Longrightarrow xa \in Infinitesimal \Longrightarrow xa \neq 0 \Longrightarrow D = 0$
apply (*simp add*: *nsderiv-def*)
apply (*drule bspec*)
apply *auto*
done

Can be proved differently using *NSLIM-isCont-iff*.

lemma *NSDERIV-approx*:
$NSDERIV\ f\ x\ :>\ D \Longrightarrow h \in Infinitesimal \Longrightarrow h \neq 0 \Longrightarrow (\ *f*\ f)\ (star\text{-}of\ x\ +\ h)\ -\ star\text{-}of\ (f\ x) \approx 0$
apply (*simp add*: *nsderiv-def*)
apply (*simp add*: *mem-infmal-iff* [*symmetric*])
apply (*rule Infinitesimal-ratio*)
apply (*rule-tac* [*3*] *approx-star-of-HFinite*, *auto*)
done

From one version of differentiability

$f\ x\ -\ f\ a\ ----------\ \approx\ Db\ x\ -\ a$

lemma *NSDERIVD1*: [| *NSDERIV f* (*g x*) :> *Da*;
(∗*f*∗ *g*) (*star-of*(*x*) + *xa*) ≠ *star-of* (*g x*);
(∗*f*∗ *g*) (*star-of*(*x*) + *xa*) ≈ *star-of* (*g x*)
|] ==> ((∗*f*∗ *f*) ((∗*f*∗ *g*) (*star-of*(*x*) + *xa*))
− *star-of* (*f* (*g x*)))
/ ((∗*f*∗ *g*) (*star-of*(*x*) + *xa*) − *star-of* (*g x*))
≈ *star-of*(*Da*)
by (*auto simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def*)

From other version of differentiability

$f\ (x\ +\ h)\ -\ f\ x\ ----------\ \approx\ Db\ h$

lemma *NSDERIVD2*: [| *NSDERIV g x* :> *Db*; *xa* ∈ *Infinitesimal*; *xa* ≠ *0* |]
==> ((∗*f*∗ *g*) (*star-of*(*x*) + *xa*) − *star-of*(*g x*)) / *xa*
≈ *star-of*(*Db*)
by (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def mem-infmal-iff starfun-lambda-cancel*)

lemma *lemma-chain*: *z* ≠ *0* ⟹ *x* ∗ *y* = (*x* ∗ *inverse z*) ∗ (*z* ∗ *y*)
for *x y z* :: *'a*::*real-normed-field star*
proof −
assume *z*: *z* ≠ *0*
have *x* ∗ *y* = *x* ∗ (*inverse z* ∗ *z*) ∗ *y* **by** (*simp add*: *z*)
then show *?thesis* **by** (*simp add*: *mult.assoc*)
qed

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*
apply (*simp* (*no-asm-simp*) *add*: *NSDERIV-NSLIM-iff NSLIM-def mem-infmal-iff*
[*symmetric*])
apply *clarify*
apply (*frule-tac f* = *g* **in** *NSDERIV-approx*)
apply (*auto simp add*: *starfun-lambda-cancel2 starfun-o* [*symmetric*])
apply (*case-tac* (∗*f*∗ *g*) (*star-of* (*x*) + *xa*) = *star-of* (*g x*))
apply (*drule-tac g* = *g* **in** *NSDERIV-zero*)
apply (*auto simp add*: *divide-inverse*)
apply (*rule-tac z1* = (∗*f*∗ *g*) (*star-of* (*x*) + *xa*) − *star-of* (*g x*) **and** *y1* =
inverse xa
in *lemma-chain* [*THEN ssubst*])
apply (*erule hypreal-not-eq-minus-iff* [*THEN iffD1*])
apply (*rule approx-mult-star-of*)
apply (*simp-all add*: *divide-inverse* [*symmetric*])
apply (*blast intro*: *NSDERIVD1 approx-minus-iff* [*THEN iffD2*])
apply (*blast intro*: *NSDERIVD2*)
done

Differentiation of natural number powers.

lemma *NSDERIV-Id* [*simp*]: *NSDERIV* ($\lambda x.\ x$) *x* :> *1*
by (*simp add*: *NSDERIV-NSLIM-iff NSLIM-def del*: *divide-self-if*)

lemma *NSDERIV-cmult-Id* [*simp*]: *NSDERIV* (*op* ∗ *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*::*'a*) because it isn't continuous at zero

shows *NSDERIV* ($\lambda x.$ *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*))
apply −
apply (*rule inverse-add-Infinitesimal-approx2*)
apply (*auto dest*!: *hypreal-of-real-HFinite-diff-Infinitesimal*
simp add: *inverse-minus-eq* [*symmetric*] *HFinite-minus-iff*)
done
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*
apply (*simp add*: *division-ring-inverse-diff nonzero-inverse-mult-distrib* [*symmetric*]
nonzero-inverse-minus-eq [*symmetric*] *ac-simps ring-distribs*)
apply (*subst nonzero-inverse-minus-eq* [*symmetric*])
using *distrib-right* [*symmetric*, *of h star-of x star-of x*] **apply** *simp*
apply (*simp add*: *field-simps*)
done
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* $\longleftrightarrow$ *DERIV f x :> D*
by (*simp add*: *DERIV-def NSDERIV-NSLIM-iff LIM-NSLIM-iff*)

NS version.

lemma *NSDERIV-pow*: *NSDERIV* ($\lambda x.\ x\ \hat{}\ n$) *x :> real n* $*$ (*x* $\hat{}$ (*n* $-$ *Suc 0*))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-pow*)

Derivative of inverse.

lemma *NSDERIV-inverse-fun*:
NSDERIV f x :> d $\Longrightarrow$ *f x* $\neq$ *0* $\Longrightarrow$
NSDERIV ($\lambda x.$ *inverse* (*f x*)) *x :>* ($-$ (*d* $*$ *inverse* (*f x* $\hat{}$ *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* $\Longrightarrow$ *NSDERIV g x :> e* $\Longrightarrow$ *g x* $\neq$ *0* $\Longrightarrow$
NSDERIV ($\lambda y.$ *f y* / *g y*) *x :>* (*d* $*$ *g x* $-$ (*e* $*$ *f x*)) / (*g x* $\hat{}$ *Suc* (*Suc 0*))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-quotient del*: *power-Suc*)

lemma *CARAT-NSDERIV*:
NSDERIV f x :> l $\Longrightarrow$ $\exists$ *g*. ($\forall$ *z*. *f z* $-$ *f x* $=$ *g z* $*$ (*z* $-$ *x*)) $\wedge$ *isNSCont g x* $\wedge$ *g x* $=$ *l*
by (*auto simp add*: *NSDERIV-DERIV-iff isNSCont-isCont-iff CARAT-DERIV mult.commute*)

lemma *hypreal-eq-minus-iff3*: *x* $=$ *y* $+$ *z* $\longleftrightarrow$ *x* $+ -$ *z* $=$ *y*
for *x y z* :: *hypreal*
by *auto*

lemma *CARAT-DERIVD*:
assumes *all*: $\forall$ *z*. *f z* $-$ *f x* $=$ *g z* $*$ (*z* $-$ *x*)
and *nsc*: *isNSCont g x*
shows *NSDERIV f x :> g x*
proof $-$
from *nsc* **have** $\forall$ *w*. *w* $\neq$ *star-of x* $\wedge$ *w* $\approx$ *star-of x* $\longrightarrow$
($*$*f*$*$ *g*) *w* $*$ (*w* $-$ *star-of x*) / (*w* $-$ *star-of x*) $\approx$ *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* $\Longrightarrow$ $\exists$ *D*. *NSDERIV f x :> D*
by (*simp add*: *NSdifferentiable-def*)

lemma *NSdifferentiableI*: *NSDERIV f x :> D* $\Longrightarrow$ *f NSdifferentiable x*

by (*force simp add*: *NSdifferentiable-def*)

13.3 (NS) Increment

lemma *incrementI*:
f NSdifferentiable x $\Longrightarrow$
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* $\Longrightarrow$
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*:
NSDERIV f x :> *D* $\Longrightarrow$ *h* ∈ *Infinitesimal* $\Longrightarrow$ *h* ≠ *0* $\Longrightarrow$
∃ *e* ∈ *Infinitesimal*. *increment f x h* = *hypreal-of-real D* ∗ *h* + *e* ∗ *h*
apply (*frule-tac h* = *h* **in** *incrementI2*, *simp add*: *nsderiv-def*)
apply (*drule bspec*, *auto*)
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*], *clarify*)
apply (*frule-tac b1* = *hypreal-of-real D* + *y* **in** *mult-right-cancel* [*THEN iffD2*])
apply (*erule-tac* [*2*]
V = (($*f*$ *f*) (*hypreal-of-real x* + *h*) − *hypreal-of-real* (*f x*)) / *h* = *hypreal-of-real* (*D*) + *y*
in *thin-rl*)
apply *assumption*
apply (*simp add*: *times-divide-eq-right* [*symmetric*])
apply (*auto simp add*: *distrib-right*)
done

lemma *increment-thm2*:
NSDERIV f x :> *D* $\Longrightarrow$ *h* ≈ *0* $\Longrightarrow$ *h* ≠ *0* $\Longrightarrow$
∃ *e* ∈ *Infinitesimal*. *increment f x h* = *hypreal-of-real D* ∗ *h* + *e* ∗ *h*
by (*blast dest*!: *mem-infmal-iff* [*THEN iffD2*] *intro*!: *increment-thm*)

lemma *increment-approx-zero*: *NSDERIV f x* :> *D* $\Longrightarrow$ *h* ≈ *0* $\Longrightarrow$ *h* ≠ *0* $\Longrightarrow$ *increment f x h* ≈ *0*
apply (*drule increment-thm2*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult2 HFinite-add*
simp add: *distrib-right* [*symmetric*] *mem-infmal-iff* [*symmetric*])
apply (*erule Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

end

14 Nonstandard Extensions of Transcendental Functions

theory *HTranscendental*
imports *HOL.Transcendental HSeries HDeriv*
begin

definition
exphr :: *real* => *hypreal* **where**
— define exponential function using standard part
exphr x = *st*(*sumhr* (*0*, *whn*, %*n*. *inverse* (*fact n*) * (*x* ^ *n*)))

definition
sinhr :: *real* => *hypreal* **where**
sinhr x = *st*(*sumhr* (*0*, *whn*, %*n*. *sin-coeff n* * *x* ^ *n*))

definition
coshr :: *real* => *hypreal* **where**
coshr x = *st*(*sumhr* (*0*, *whn*, %*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*)
apply (*cases x*)
apply (*auto simp add*: *star-n-le star-n-zero-num starfun hrealpow star-n-eq-iff*
simp del: *hpowr-Suc power-Suc*)
done

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*
apply (*frule hypreal-sqrt-pow2-gt-zero*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *hypreal-inverse-sqrt-pow2*:
0 < *x* ==> *inverse* (((*f* *) *sqrt*)(*x*)) ^ *2* = *inverse x*
apply (*cut-tac n* = *2* **and** *a* = (*f* *) *sqrt x* **in** *power-inverse* [*symmetric*])
apply (*auto dest*: *hypreal-sqrt-gt-zero-pow2*)

done

lemma *hypreal-sqrt-mult-distrib*:
!!$x\ y.\ [|0 < x;\ 0 < y\ |] ==>$
$(*f* sqrt)(x*y) = (*f* sqrt)(x) * (*f* sqrt)(y)$
apply *transfer*
apply (*auto intro*: *real-sqrt-mult-distrib*)
done

lemma *hypreal-sqrt-mult-distrib2*:
$[|0 \le x;\ 0 \le y\ |] ==>$
$(*f* sqrt)(x*y) = (*f* sqrt)(x) * (*f* sqrt)(y)$
by (*auto intro*: *hypreal-sqrt-mult-distrib simp add*: *order-le-less*)

lemma *hypreal-sqrt-approx-zero* [*simp*]:
$0 < x ==> ((*f* sqrt)(x) \approx 0) = (x \approx 0)$
apply (*auto simp add*: *mem-infmal-iff* [*symmetric*])
apply (*rule hypreal-sqrt-gt-zero-pow2* [*THEN subst*])
apply (*auto intro*: *Infinitesimal-mult*
dest!: *hypreal-sqrt-gt-zero-pow2* [*THEN ssubst*]
simp add: *numeral-2-eq-2*)
done

lemma *hypreal-sqrt-approx-zero2* [*simp*]:
$0 \le x ==> ((*f* sqrt)(x) \approx 0) = (x \approx 0)$
by (*auto simp add*: *order-le-less*)

lemma *hypreal-sqrt-sum-squares* [*simp*]:
$((*f* sqrt)(x*x + y*y + z*z) \approx 0) = (x*x + y*y + z*z \approx 0)$
apply (*rule hypreal-sqrt-approx-zero2*)
apply (*rule add-nonneg-nonneg*)+
apply (*auto*)
done

lemma *hypreal-sqrt-sum-squares2* [*simp*]:
$((*f* sqrt)(x*x + y*y) \approx 0) = (x*x + y*y \approx 0)$
apply (*rule hypreal-sqrt-approx-zero2*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *hypreal-sqrt-gt-zero*: !!$x.\ 0 < x ==> 0 < (*f* sqrt)(x)$
apply *transfer*
apply (*auto intro*: *real-sqrt-gt-zero*)
done

lemma *hypreal-sqrt-ge-zero*: $0 \le x ==> 0 \le (*f* sqrt)(x)$
by (*auto intro*: *hypreal-sqrt-gt-zero simp add*: *order-le-less*)

lemma *hypreal-sqrt-hrabs* [*simp*]: $!!x.\ (\ *f*\ sqrt)(x^2) = |x|$
by (*transfer*, *simp*)

lemma *hypreal-sqrt-hrabs2* [*simp*]: $!!x.\ (\ *f*\ sqrt)(x*x) = |x|$
by (*transfer*, *simp*)

lemma *hypreal-sqrt-hyperpow-hrabs* [*simp*]:
$!!x.\ (\ *f*\ sqrt)(x\ pow\ (hypnat\text{-}of\text{-}nat\ 2)) = |x|$
by (*transfer*, *simp*)

lemma *star-sqrt-HFinite*: $[\![x \in HFinite;\ 0 \le x]\!] \Longrightarrow (\ *f*\ sqrt)\ x \in HFinite$
apply (*rule HFinite-square-iff* [*THEN iffD1*])
apply (*simp only*: *hypreal-sqrt-mult-distrib2* [*symmetric*], *simp*)
done

lemma *st-hypreal-sqrt*:
$[|\ x \in HFinite;\ 0 \le x\ |] ==> st((\ *f*\ sqrt)\ x) = (\ *f*\ sqrt)(st\ x)$
apply (*rule power-inject-base* [**where** *n=1*])
apply (*auto intro*!: *st-zero-le hypreal-sqrt-ge-zero*)
apply (*rule st-mult* [*THEN subst*])
apply (*rule-tac* [*3*] *hypreal-sqrt-mult-distrib2* [*THEN subst*])
apply (*rule-tac* [*5*] *hypreal-sqrt-mult-distrib2* [*THEN subst*])
apply (*auto simp add*: *st-hrabs st-zero-le star-sqrt-HFinite*)
done

lemma *hypreal-sqrt-sum-squares-ge1* [*simp*]: $!!x\ y.\ x \le (\ *f*\ sqrt)(x^2 + y^2)$
by *transfer* (*rule real-sqrt-sum-squares-ge1*)

lemma *HFinite-hypreal-sqrt*:
$[|\ 0 \le x;\ x \in HFinite\ |] ==> (\ *f*\ sqrt)\ x \in HFinite$
apply (*auto simp add*: *order-le-less*)
apply (*rule HFinite-square-iff* [*THEN iffD1*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2*)
done

lemma *HFinite-hypreal-sqrt-imp-HFinite*:
$[|\ 0 \le x;\ (\ *f*\ sqrt)\ x \in HFinite\ |] ==> x \in HFinite$
apply (*auto simp add*: *order-le-less*)
apply (*drule HFinite-square-iff* [*THEN iffD2*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2 del*: *HFinite-square-iff*)
done

lemma *HFinite-hypreal-sqrt-iff* [*simp*]:
$0 \le x ==> ((\ *f*\ sqrt)\ x \in HFinite) = (x \in HFinite)$
by (*blast intro*: *HFinite-hypreal-sqrt HFinite-hypreal-sqrt-imp-HFinite*)

lemma *HFinite-sqrt-sum-squares* [*simp*]:

(($*f*$ *sqrt*)(*x***x* + *y***y*) ∈ *HFinite*) = (*x***x* + *y***y* ∈ *HFinite*)
apply (*rule HFinite-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *Infinitesimal-hypreal-sqrt*:
[| 0 ≤ *x*; *x* ∈ *Infinitesimal* |] ==> ($*f*$ *sqrt*) *x* ∈ *Infinitesimal*
apply (*auto simp add*: *order-le-less*)
apply (*rule Infinitesimal-square-iff* [*THEN iffD2*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2*)
done

lemma *Infinitesimal-hypreal-sqrt-imp-Infinitesimal*:
[| 0 ≤ *x*; ($*f*$ *sqrt*) *x* ∈ *Infinitesimal* |] ==> *x* ∈ *Infinitesimal*
apply (*auto simp add*: *order-le-less*)
apply (*drule Infinitesimal-square-iff* [*THEN iffD1*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2 del*: *Infinitesimal-square-iff* [*symmetric*])
done

lemma *Infinitesimal-hypreal-sqrt-iff* [*simp*]:
0 ≤ *x* ==> (($*f*$ *sqrt*) *x* ∈ *Infinitesimal*) = (*x* ∈ *Infinitesimal*)
by (*blast intro*: *Infinitesimal-hypreal-sqrt-imp-Infinitesimal Infinitesimal-hypreal-sqrt*)

lemma *Infinitesimal-sqrt-sum-squares* [*simp*]:
(($*f*$ *sqrt*)(*x***x* + *y***y*) ∈ *Infinitesimal*) = (*x***x* + *y***y* ∈ *Infinitesimal*)
apply (*rule Infinitesimal-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *HInfinite-hypreal-sqrt*:
[| 0 ≤ *x*; *x* ∈ *HInfinite* |] ==> ($*f*$ *sqrt*) *x* ∈ *HInfinite*
apply (*auto simp add*: *order-le-less*)
apply (*rule HInfinite-square-iff* [*THEN iffD1*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2*)
done

lemma *HInfinite-hypreal-sqrt-imp-HInfinite*:
[| 0 ≤ *x*; ($*f*$ *sqrt*) *x* ∈ *HInfinite* |] ==> *x* ∈ *HInfinite*
apply (*auto simp add*: *order-le-less*)
apply (*drule HInfinite-square-iff* [*THEN iffD2*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2 del*: *HInfinite-square-iff*)
done

lemma *HInfinite-hypreal-sqrt-iff* [*simp*]:
0 ≤ *x* ==> ((*f* *sqrt*) *x* ∈ *HInfinite*) = (*x* ∈ *HInfinite*)
by (*blast intro*: *HInfinite-hypreal-sqrt HInfinite-hypreal-sqrt-imp-HInfinite*)

lemma *HInfinite-sqrt-sum-squares* [*simp*]:
((*f* *sqrt*)(*x**x + *y**y) ∈ *HInfinite*) = (*x**x + *y**y ∈ *HInfinite*)
apply (*rule HInfinite-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *HFinite-exp* [*simp*]:
sumhr (*0*, *whn*, %*n*. *inverse* (*fact n*) ∗ *x* ^ *n*) ∈ *HFinite*
unfolding *sumhr-app*
apply (*simp only*: *star-zero-def starfun2-star-of atLeast0LessThan*)
apply (*rule NSBseqD2*)
apply (*rule NSconvergent-NSBseq*)
apply (*rule convergent-NSconvergent-iff* [*THEN iffD1*])
apply (*rule summable-iff-convergent* [*THEN iffD1*])
apply (*rule summable-exp*)
done

lemma *exphr-zero* [*simp*]: *exphr 0 = 1*
apply (*simp add*: *exphr-def sumhr-split-add* [*OF hypnat-one-less-hypnat-omega*, *symmetric*])
apply (*rule st-unique*, *simp*)
apply (*rule subst* [**where** *P*=λ*x*. *1* ≈ *x*, *OF - approx-refl*])
apply (*rule rev-mp* [*OF hypnat-one-less-hypnat-omega*])
apply (*rule-tac x*=*whn* **in** *spec*)
apply (*unfold sumhr-app*, *transfer*, *simp add*: *power-0-left*)
done

lemma *coshr-zero* [*simp*]: *coshr 0 = 1*
apply (*simp add*: *coshr-def sumhr-split-add*
[*OF hypnat-one-less-hypnat-omega*, *symmetric*])
apply (*rule st-unique*, *simp*)
apply (*rule subst* [**where** *P*=λ*x*. *1* ≈ *x*, *OF - approx-refl*])
apply (*rule rev-mp* [*OF hypnat-one-less-hypnat-omega*])
apply (*rule-tac x*=*whn* **in** *spec*)
apply (*unfold sumhr-app*, *transfer*, *simp add*: *cos-coeff-def power-0-left*)
done

lemma *STAR-exp-zero-approx-one* [*simp*]: (*f* *exp*) (*0*::*hypreal*) ≈ *1*
apply (*subgoal-tac* (*f* *exp*) (*0*::*hypreal*) = *1*, *simp*)
apply (*transfer*, *simp*)
done

lemma *STAR-exp-Infinitesimal*: *x* ∈ *Infinitesimal* ==> (*f* *exp*) (*x*::*hypreal*) ≈ *1*

apply (*case-tac x = 0*)
apply (*cut-tac* [*2*] *x = 0* **in** *DERIV-exp*)
apply (*auto simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
apply (*drule-tac x = x* **in** *bspec*, *auto*)
apply (*drule-tac c = x* **in** *approx-mult1*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp *add*: *mult.assoc*)
apply (*rule approx-add-right-cancel* [**where** *d=−1*])
apply (*rule approx-sym* [*THEN* [*2*] *approx-trans2*])
apply (*auto simp add*: *mem-infmal-iff*)
done

lemma *STAR-exp-epsilon* [*simp*]: (∗*f*∗ *exp*) *ε* ≈ *1*
by (*auto intro*: *STAR-exp-Infinitesimal*)

lemma *STAR-exp-add*:
!!(*x*::′*a*:: {*banach*,*real-normed-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-of-real* (*exp x*)
apply (*simp add*: *exphr-def*)
apply (*rule st-unique*, *simp*)
apply (*subst starfunNat-sumr* [*symmetric*])
unfolding *atLeast0LessThan*
apply (*rule NSLIMSEQ-D* [*THEN approx-sym*])
apply (*rule LIMSEQ-NSLIMSEQ*)
apply (*subst sums-def* [*symmetric*])
apply (*cut-tac exp-converges* [**where** *x=x*], *simp*)
apply (*rule HNatInfinite-whn*)
done

lemma *starfun-exp-ge-add-one-self* [*simp*]: !!*x*::*hypreal*. *0* ≤ *x* ==> (*1* + *x*) ≤ (∗*f*∗ *exp*) *x*
by *transfer* (*rule exp-ge-add-one-self-aux*)

lemma *starfun-exp-HInfinite*:
[| *x* ∈ *HInfinite*; *0* ≤ *x* |] ==> (∗*f*∗ *exp*) (*x*::*hypreal*) ∈ *HInfinite*
apply (*frule starfun-exp-ge-add-one-self*)
apply (*rule HInfinite-ge-HInfinite*, *assumption*)
apply (*rule order-trans* [*of - 1+x*], *auto*)
done

lemma *starfun-exp-minus*:
!!*x*::′*a*:: {*banach*,*real-normed-field*} *star*. (∗*f*∗ *exp*) (−*x*) = *inverse*((∗*f*∗ *exp*) *x*)
by *transfer* (*rule exp-minus*)

lemma *starfun-exp-Infinitesimal*:
[| $x \in$ *HInfinite*; $x \leq 0$ |] ==> ($*f*$ *exp*) (*x*::*hypreal*) $\in$ *Infinitesimal*
apply (*subgoal-tac* $\exists y.\ x = -\ y$)
apply (*rule-tac* [*2*] $x = -\ x$ **in** *exI*)
apply (*auto intro*!: *HInfinite-inverse-Infinitesimal starfun-exp-HInfinite*
simp add: *starfun-exp-minus HInfinite-minus-iff*)
done

lemma *starfun-exp-gt-one* [*simp*]: !!*x*::*hypreal*. $0 < x$ ==> $1 <$ ($*f*$ *exp*) *x*
by *transfer* (*rule exp-gt-one*)

abbreviation *real-ln* :: *real* $\Rightarrow$ *real* **where**
real-ln $\equiv$ *ln*

lemma *starfun-ln-exp* [*simp*]: !!*x*. ($*f*$ *real-ln*) (($*f*$ *exp*) *x*) = *x*
by *transfer* (*rule ln-exp*)

lemma *starfun-exp-ln-iff* [*simp*]: !!*x*. (($*f*$ *exp*)(($*f*$ *real-ln*) *x*) = *x*) = (*0* $<$ *x*)
by *transfer* (*rule exp-ln-iff*)

lemma *starfun-exp-ln-eq*: !!*u x*. ($*f*$ *exp*) *u* = *x* ==> ($*f*$ *real-ln*) *x* = *u*
by *transfer* (*rule ln-unique*)

lemma *starfun-ln-less-self* [*simp*]: !!*x*. $0 < x$ ==> ($*f*$ *real-ln*) *x* $<$ *x*
by *transfer* (*rule ln-less-self*)

lemma *starfun-ln-ge-zero* [*simp*]: !!*x*. $1 \leq x$ ==> $0 \leq$ ($*f*$ *real-ln*) *x*
by *transfer* (*rule ln-ge-zero*)

lemma *starfun-ln-gt-zero* [*simp*]: !!*x* .*1* $< x$ ==> $0 <$ ($*f*$ *real-ln*) *x*
by *transfer* (*rule ln-gt-zero*)

lemma *starfun-ln-not-eq-zero* [*simp*]: !!*x*. [| $0 < x$; $x \neq 1$ |] ==> ($*f*$ *real-ln*) *x* $\neq$ *0*
by *transfer simp*

lemma *starfun-ln-HFinite*: [| $x \in$ *HFinite*; $1 \leq x$ |] ==> ($*f*$ *real-ln*) *x* $\in$ *HFinite*
apply (*rule HFinite-bounded*)
apply *assumption*
apply (*simp-all add*: *starfun-ln-less-self order-less-imp-le*)
done

lemma *starfun-ln-inverse*: !!*x*. $0 < x$ ==> ($*f*$ *real-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*: $x \in HFinite \Longrightarrow (\ {*f*}\ exp)\ (x{::}hypreal) \in HFinite$
apply (*auto simp add*: *HFinite-def*, *rename-tac u*)
apply (*rule-tac x*=$(\ {*f*}\ exp)$ *u* **in** *rev-bexI*)
apply (*simp add*: *Reals-eq-Standard*)
apply (*simp add*: *starfun-abs-exp-cancel*)
apply (*simp add*: *starfun-exp-less-mono*)
done

lemma *starfun-exp-add-HFinite-Infinitesimal-approx*:
$[|x \in Infinitesimal;\ z \in HFinite\ |] \Longrightarrow (\ {*f*}\ exp)\ (z + x{::}hypreal) \approx (\ {*f*}\ exp)\ z$
apply (*simp add*: *STAR-exp-add*)
apply (*frule STAR-exp-Infinitesimal*)
apply (*drule approx-mult2*)
apply (*auto intro*: *starfun-exp-HFinite*)
done

lemma *starfun-ln-HInfinite*:
$[|\ x \in HInfinite;\ 0 < x\ |] \Longrightarrow (\ {*f*}\ real\text{-}ln)\ x \in HInfinite$
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*drule starfun-exp-HFinite*)
apply (*simp add*: *starfun-exp-ln-iff* [*THEN iffD2*] *HFinite-HInfinite-iff*)
done

lemma *starfun-exp-HInfinite-Infinitesimal-disj*:
$x \in HInfinite \Longrightarrow (\ {*f*}\ exp)\ x \in HInfinite \mid (\ {*f*}\ exp)\ (x{::}hypreal) \in Infinitesimal$
apply (*insert linorder-linear* [*of x 0*])
apply (*auto intro*: *starfun-exp-HInfinite starfun-exp-Infinitesimal*)
done

lemma *starfun-ln-HFinite-not-Infinitesimal*:
$[|\ x \in HFinite - Infinitesimal;\ 0 < x\ |] \Longrightarrow (\ {*f*}\ real\text{-}ln)\ x \in HFinite$
apply (*rule ccontr*, *drule HInfinite-HFinite-iff* [*THEN iffD2*])
apply (*drule starfun-exp-HInfinite-Infinitesimal-disj*)
apply (*simp add*: *starfun-exp-ln-iff* [*symmetric*] *HInfinite-HFinite-iff*
del: *starfun-exp-ln-iff*)
done

lemma *starfun-ln-Infinitesimal-HInfinite*:
$[|\ x \in Infinitesimal;\ 0 < x\ |] \Longrightarrow (\ {*f*}\ real\text{-}ln)\ x \in HInfinite$
apply (*drule Infinitesimal-inverse-HInfinite*)
apply (*frule positive-imp-inverse-positive*)

apply (*drule-tac* [*2*] *starfun-ln-HInfinite*)
apply (*auto simp add*: *starfun-ln-inverse HInfinite-minus-iff*)
done

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*, %*n*. *sin-coeff n* * *x* ^ *n*) ∈ *HFinite*
unfolding *sumhr-app*
apply (*simp only*: *star-zero-def starfun2-star-of atLeast0LessThan*)
apply (*rule NSBseqD2*)
apply (*rule NSconvergent-NSBseq*)
apply (*rule convergent-NSconvergent-iff* [*THEN iffD1*])
apply (*rule summable-iff-convergent* [*THEN iffD1*])
using *summable-norm-sin* [*of x*]
apply (*simp add*: *summable-rabs-cancel*)
done

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*
  shows *x* ∈ *Infinitesimal* ==> (*f* *) *sin*) *x* ≈ *x*
apply (*case-tac x* = *0*)
apply (*cut-tac* [*2*] *x* = *0* **in** *DERIV-sin*)
apply (*auto simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
apply (*drule bspec* [**where** *x* = *x*], *auto*)
apply (*drule approx-mult1* [**where** *c* = *x*])
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
        simp add: *mult.assoc*)
done

lemma *HFinite-cos* [*simp*]: *sumhr* (*0*, *whn*, %*n*. *cos-coeff n* * *x* ^ *n*) ∈ *HFinite*
unfolding *sumhr-app*
apply (*simp only*: *star-zero-def starfun2-star-of atLeast0LessThan*)
apply (*rule NSBseqD2*)
apply (*rule NSconvergent-NSBseq*)

apply (*rule convergent-NSconvergent-iff* [*THEN iffD1*])
apply (*rule summable-iff-convergent* [*THEN iffD1*])
using *summable-norm-cos* [*of x*]
apply (*simp add*: *summable-rabs-cancel*)
done

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*
  shows *x* ∈ *Infinitesimal* ==> (**f** *cos*) *x* ≈ *1*
apply (*case-tac x* = *0*)
apply (*cut-tac* [*2*] *x* = *0* **in** *DERIV-cos*)
apply (*auto simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
apply (*drule bspec* [**where** *x* = *x*])
apply *auto*
apply (*drule approx-mult1* [**where** *c* = *x*])
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
        simp add: *mult.assoc*)
apply (*rule approx-add-right-cancel* [**where** *d* = −*1*])
apply *simp*
done

lemma *STAR-tan-zero* [*simp*]: (**f** *tan*) *0* = *0*
by *transfer* (*rule tan-zero*)

lemma *STAR-tan-Infinitesimal*: *x* ∈ *Infinitesimal* ==> (**f** *tan*) *x* ≈ *x*
apply (*case-tac x* = *0*)
apply (*cut-tac* [*2*] *x* = *0* **in** *DERIV-tan*)
apply (*auto simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
apply (*drule bspec* [**where** *x* = *x*], *auto*)
apply (*drule approx-mult1* [**where** *c* = *x*])
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
        simp add: *mult.assoc*)
done

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 *lemma-split-hypreal-of-real*:

$N \in HNatInfinite$
$\Longrightarrow hypreal\text{-}of\text{-}real\ a =$
$hypreal\text{-}of\text{-}hypnat\ N * (inverse(hypreal\text{-}of\text{-}hypnat\ N) * hypreal\text{-}of\text{-}real\ a)$
by (*simp add*: *mult.assoc* [*symmetric*] *zero-less-HNatInfinite*)

lemma *STAR-sin-Infinitesimal-divide*:
fixes $x :: 'a::\{real\text{-}normed\text{-}field,banach\}\ star$
shows $[|x \in Infinitesimal;\ x \neq 0\ |] \Longrightarrow (\ *f*\ sin)\ x/x \approx 1$
using *DERIV-sin* [*of* $0::'a$]
by (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)

lemma *lemma-sin-pi*:
$n \in HNatInfinite$
$\Longrightarrow (\ *f*\ sin)\ (inverse\ (hypreal\text{-}of\text{-}hypnat\ n))/(inverse\ (hypreal\text{-}of\text{-}hypnat\ n)) \approx 1$
apply (*rule STAR-sin-Infinitesimal-divide*)
apply (*auto simp add*: *zero-less-HNatInfinite*)
done

lemma *STAR-sin-inverse-HNatInfinite*:
$n \in HNatInfinite$
$\Longrightarrow (\ *f*\ sin)\ (inverse\ (hypreal\text{-}of\text{-}hypnat\ n)) * hypreal\text{-}of\text{-}hypnat\ n \approx 1$
apply (*frule lemma-sin-pi*)
apply (*simp add*: *divide-inverse*)
done

lemma *Infinitesimal-pi-divide-HNatInfinite*:
$N \in HNatInfinite$
$\Longrightarrow hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ N) \in Infinitesimal$
apply (*simp add*: *divide-inverse*)
apply (*auto intro*: *Infinitesimal-HFinite-mult2*)
done

lemma *pi-divide-HNatInfinite-not-zero* [*simp*]:
$N \in HNatInfinite \Longrightarrow hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ N) \neq 0$
by (*simp add*: *zero-less-HNatInfinite*)

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi*:
$n \in HNatInfinite$
$\Longrightarrow (\ *f*\ sin)\ (hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ n)) * hypreal\text{-}of\text{-}hypnat\ n$
$\approx hypreal\text{-}of\text{-}real\ pi$
apply (*frule STAR-sin-Infinitesimal-divide*
[*OF Infinitesimal-pi-divide-HNatInfinite*
pi-divide-HNatInfinite-not-zero])

apply (*auto*)
apply (*rule approx-SReal-mult-cancel* [*of inverse* (*hypreal-of-real pi*)])
apply (*auto intro*: *Reals-inverse simp add*: *divide-inverse ac-simps*)
done

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*
apply (*rule mult.commute* [*THEN subst*])
apply (*erule STAR-sin-pi-divide-HNatInfinite-approx-pi*)
done

lemma *starfunNat-pi-divide-n-Infinitesimal*:
N ∈ *HNatInfinite* ==> (∗*f*∗ (%*x*. *pi* / *real x*)) *N* ∈ *Infinitesimal*
by (*auto intro*!: *Infinitesimal-HFinite-mult2*
simp add: *starfun-mult* [*symmetric*] *divide-inverse*
starfun-inverse [*symmetric*] *starfunNat-real-of-nat*)

lemma *STAR-sin-pi-divide-n-approx*:
N ∈ *HNatInfinite* ==>
(∗*f*∗ *sin*) ((∗*f*∗ (%*x*. *pi* / *real x*)) *N*) ≈
hypreal-of-real pi/(*hypreal-of-hypnat N*)
apply (*simp add*: *starfunNat-real-of-nat* [*symmetric*])
apply (*rule STAR-sin-Infinitesimal*)
apply (*simp add*: *divide-inverse*)
apply (*rule Infinitesimal-HFinite-mult2*)
apply (*subst starfun-inverse*)
apply (*erule starfunNat-inverse-real-of-nat-Infinitesimal*)
apply *simp*
done

lemma *NSLIMSEQ-sin-pi*: (%*n*. *real n* ∗ *sin* (*pi* / *real n*)) ⟶$_{NS}$ *pi*
apply (*auto simp add*: *NSLIMSEQ-def starfun-mult* [*symmetric*] *starfunNat-real-of-nat*)
apply (*rule-tac f1* = *sin* **in** *starfun-o2* [*THEN subst*])
apply (*auto simp add*: *starfun-mult* [*symmetric*] *starfunNat-real-of-nat divide-inverse*)
apply (*rule-tac f1* = *inverse* **in** *starfun-o2* [*THEN subst*])
apply (*auto dest*: *STAR-sin-pi-divide-HNatInfinite-approx-pi*
simp add: *starfunNat-real-of-nat mult.commute divide-inverse*)
done

lemma *NSLIMSEQ-cos-one*: (%*n*. *cos* (*pi* / *real n*))⟶$_{NS}$ *1*
apply (*simp add*: *NSLIMSEQ-def*, *auto*)
apply (*rule-tac f1* = *cos* **in** *starfun-o2* [*THEN subst*])
apply (*rule STAR-cos-Infinitesimal*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult2*
simp add: *starfun-mult* [*symmetric*] *divide-inverse*
starfun-inverse [*symmetric*] *starfunNat-real-of-nat*)

done

lemma *NSLIMSEQ-sin-cos-pi*:
(%*n*. *real n* ∗ *sin* (*pi* / *real n*) ∗ *cos* (*pi* / *real n*)) ⟶$_{NS}$ *pi*
by (*insert NSLIMSEQ-mult* [*OF NSLIMSEQ-sin-pi NSLIMSEQ-cos-one*], *simp*)

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* ∈ *Infinitesimal* ==> (∗*f*∗ *cos*) *x* ≈ *1* − *x*^{2}
apply (*rule STAR-cos-Infinitesimal* [*THEN approx-trans*])
apply (*auto simp add*: *Infinitesimal-approx-minus* [*symmetric*]
add.assoc [*symmetric*] *numeral-2-eq-2*)
done

lemma *STAR-cos-Infinitesimal-approx2*:
fixes *x* :: *hypreal* — perhaps could be generalised, like many other hypreal results

shows *x* ∈ *Infinitesimal* ==> (∗*f*∗ *cos*) *x* ≈ *1* − (*x*^{2})/*2*
apply (*rule STAR-cos-Infinitesimal* [*THEN approx-trans*])
apply (*auto intro*: *Infinitesimal-SReal-divide Infinitesimal-mult*
simp add: *Infinitesimal-approx-minus* [*symmetric*] *numeral-2-eq-2*)
done

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*)
by (*auto*, *drule Standard-minus*, *auto*)

lemma *SComplex-add-cancel*:
[| *x* + *y* ∈ *SComplex*; *y* ∈ *SComplex* |] ==> *x* ∈ *SComplex*

by (*drule* (*1*) *Standard-diff*, *simp*)

lemma *SReal-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (*hcomplex-of-complex r*) ∈ ℝ
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-hcmod-numeral* [*simp*]: *hcmod* (*numeral w* ::*hcomplex*) ∈ ℝ
by (*simp add*: *Reals-eq-Standard*)

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*:
hcomplex-of-complex '(*UNIV*::*complex set*) = *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*:
[| ∃ *x*. *x*: *P*; *P* ≤ *SComplex* |] ==> ∃ *Q*. *P* = *hcomplex-of-complex* ' *Q*
apply (*simp add*: *Standard-def*, *blast*)
done

lemma *SComplex-SReal-dense*:
[| *x* ∈ *SComplex*; *y* ∈ *SComplex*; *hcmod x* < *hcmod y*
|] ==> ∃ *r* ∈ *Reals*. *hcmod x*< *r* & *r* < *hcmod y*
apply (*auto intro*: *SReal-dense simp add*: *SReal-hcmod-SComplex*)
done

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*: (*x* ∈ *HFinite*) = (*hcmod x* ∈ *HFinite*)
by (*simp add*: *HFinite-def*)

lemma *HFinite-bounded-hcmod*:
[|$x \in$ *HFinite*; $y \leq$ *hcmod* x; $0 \leq y$ |] ==> y: *HFinite*
by (*auto intro*: *HFinite-bounded simp add*: *HFinite-hcmod-iff*)

15.3 The Complex Infinitesimals form a Subring

lemma *hcomplex-sum-of-halves*: $x/(2::hcomplex) + x/(2::hcomplex) = x$
by *auto*

lemma *Infinitesimal-hcmod-iff*:
($z \in$ *Infinitesimal*) = (*hcmod* $z \in$ *Infinitesimal*)
by (*simp add*: *Infinitesimal-def*)

lemma *HInfinite-hcmod-iff*: ($z \in$ *HInfinite*) = (*hcmod* $z \in$ *HInfinite*)
by (*simp add*: *HInfinite-def*)

lemma *HFinite-diff-Infinitesimal-hcmod*:
$x \in$ *HFinite* − *Infinitesimal* ==> *hcmod* $x \in$ *HFinite* − *Infinitesimal*
by (*simp add*: *HFinite-hcmod-iff Infinitesimal-hcmod-iff*)

lemma *hcmod-less-Infinitesimal*:
[| $e \in$ *Infinitesimal*; *hcmod* $x <$ *hcmod* e |] ==> $x \in$ *Infinitesimal*
by (*auto elim*: *hrabs-less-Infinitesimal simp add*: *Infinitesimal-hcmod-iff*)

lemma *hcmod-le-Infinitesimal*:
[| $e \in$ *Infinitesimal*; *hcmod* $x \leq$ *hcmod* e |] ==> $x \in$ *Infinitesimal*
by (*auto elim*: *hrabs-le-Infinitesimal simp add*: *Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-interval-hcmod*:
[| $e \in$ *Infinitesimal*;
$e' \in$ *Infinitesimal*;
hcmod $e' <$ *hcmod* x ; *hcmod* $x <$ *hcmod* e
|] ==> $x \in$ *Infinitesimal*
by (*auto intro*: *Infinitesimal-interval simp add*: *Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-interval2-hcmod*:
[| $e \in$ *Infinitesimal*;
$e' \in$ *Infinitesimal*;
hcmod $e' \leq$ *hcmod* x ; *hcmod* $x \leq$ *hcmod* e
|] ==> $x \in$ *Infinitesimal*
by (*auto intro*: *Infinitesimal-interval2 simp add*: *Infinitesimal-hcmod-iff*)

15.4 The "Infinitely Close" Relation

lemma *approx-SComplex-mult-cancel-zero*:
[| $a \in$ *SComplex*; $a \neq 0$; $a*x \approx 0$ |] ==> $x \approx 0$
apply (*drule Standard-inverse* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
done

lemma *approx-mult-SComplex1*: [| $a \in SComplex$; $x \approx 0$ |] ==> $x*a \approx 0$
by (*auto dest*: *Standard-subset-HFinite* [*THEN subsetD*] *approx-mult1*)

lemma *approx-mult-SComplex2*: [| $a \in SComplex$; $x \approx 0$ |] ==> $a*x \approx 0$
by (*auto dest*: *Standard-subset-HFinite* [*THEN subsetD*] *approx-mult2*)

lemma *approx-mult-SComplex-zero-cancel-iff* [*simp*]:
[|$a \in SComplex$; $a \neq 0$ |] ==> $(a*x \approx 0) = (x \approx 0)$
by (*blast intro*: *approx-SComplex-mult-cancel-zero approx-mult-SComplex2*)

lemma *approx-SComplex-mult-cancel*:
[| $a \in SComplex$; $a \neq 0$; $a* w \approx a*z$ |] ==> $w \approx z$
apply (*drule Standard-inverse* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult.assoc* [*symmetric*])
done

lemma *approx-SComplex-mult-cancel-iff1* [*simp*]:
[| $a \in SComplex$; $a \neq 0$|] ==> $(a* w \approx a*z) = (w \approx z)$
by (*auto intro*!: *approx-mult2 Standard-subset-HFinite* [*THEN subsetD*]
intro: *approx-SComplex-mult-cancel*)

lemma *approx-hcmod-approx-zero*: $(x \approx y) = (hcmod\ (y - x) \approx 0)$
apply (*subst hnorm-minus-commute*)
apply (*simp add*: *approx-def Infinitesimal-hcmod-iff*)
done

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*:
$u \approx 0$ ==> $hcmod(x + u) - hcmod\ x \in Infinitesimal$
apply (*drule approx-approx-zero-iff* [*THEN iffD1*])
apply (*rule-tac e = hcmod u* **and** *e' = - hcmod u* **in** *Infinitesimal-interval2*)
apply (*auto simp add*: *mem-infmal-iff* [*symmetric*])
apply (*rule-tac c1 = hcmod x* **in** *add-le-cancel-left* [*THEN iffD1*])
apply *auto*
done

lemma *approx-hcmod-add-hcmod*: $u \approx 0$ ==> $hcmod(x + u) \approx hcmod\ x$
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*auto intro*: *Infinitesimal-hcmod-add-diff simp add*: *mem-infmal-iff* [*symmetric*])
done

15.5 Zero is the Only Infinitesimal Complex Number

lemma *Infinitesimal-less-SComplex*:
[| $x \in SComplex$; $y \in Infinitesimal$; $0 < hcmod\ x$ |] ==> $hcmod\ y < hcmod\ x$
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*:
[| $x \in SComplex$; $x \in Infinitesimal$|] ==> $x = 0$
by (*cut-tac SComplex-Int-Infinitesimal-zero*, *blast*)

lemma *SComplex-HFinite-diff-Infinitesimal*:
[| $x \in SComplex$; $x \neq 0$ |] ==> $x \in HFinite - Infinitesimal$
by (*auto dest*: *SComplex-Infinitesimal-zero Standard-subset-HFinite* [*THEN subsetD*])

lemma *hcomplex-of-complex-HFinite-diff-Infinitesimal*:
$hcomplex\text{-}of\text{-}complex\ x \neq 0$
==> $hcomplex\text{-}of\text{-}complex\ x \in HFinite - Infinitesimal$
by (*rule SComplex-HFinite-diff-Infinitesimal*, *auto*)

lemma *numeral-not-Infinitesimal* [*simp*]:
$numeral\ w \neq (0::hcomplex)$ ==> $(numeral\ w::hcomplex) \notin Infinitesimal$
by (*fast dest*: *Standard-numeral* [*THEN SComplex-Infinitesimal-zero*])

lemma *approx-SComplex-not-zero*:
[| $y \in SComplex$; $x \approx y$; $y \neq 0$ |] ==> $x \neq 0$
by (*auto dest*: *SComplex-Infinitesimal-zero approx-sym* [*THEN mem-infmal-iff* [*THEN iffD2*]])

lemma *SComplex-approx-iff*:
[|$x \in SComplex$; $y \in SComplex$|] ==> $(x \approx y) = (x = y)$
by (*auto simp add*: *Standard-def*)

lemma *numeral-Infinitesimal-iff* [*simp*]:
$((numeral\ w :: hcomplex) \in Infinitesimal) =$
$(numeral\ w = (0::hcomplex))$
apply (*rule iffI*)
apply (*fast dest*: *Standard-numeral* [*THEN SComplex-Infinitesimal-zero*])
apply (*simp* (*no-asm-simp*))
done

lemma *approx-unique-complex*:
[| $r \in SComplex$; $s \in SComplex$; $r \approx x$; $s \approx x$|] ==> $r = s$
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*: $x \in Infinitesimal \Longrightarrow hRe\ x \in Infinitesimal$
apply (*rule InfinitesimalI2*, *simp*)
apply (*rule order-le-less-trans* [*OF abs-hRe-le-hcmod*])
apply (*erule* (*1*) *InfinitesimalD2*)
done

lemma *Infinitesimal-hIm*: $x \in Infinitesimal \Longrightarrow hIm\ x \in Infinitesimal$
apply (*rule InfinitesimalI2*, *simp*)
apply (*rule order-le-less-trans* [*OF abs-hIm-le-hcmod*])
apply (*erule* (*1*) *InfinitesimalD2*)
done

lemma *real-sqrt-lessI*: $[\![0 < u;\ x < u^2]\!] \Longrightarrow sqrt\ x < u$

by (*frule real-sqrt-less-mono*) *simp*

lemma *hypreal-sqrt-lessI*:
$\bigwedge x\ u.\ [\![0 < u;\ x < u^2]\!] \Longrightarrow (\ *f*\ sqrt)\ x < u$
by *transfer* (*rule real-sqrt-lessI*)

lemma *hypreal-sqrt-ge-zero*: $\bigwedge x.\ 0 \leq x \Longrightarrow 0 \leq (\ *f*\ sqrt)\ x$
by *transfer* (*rule real-sqrt-ge-zero*)

lemma *Infinitesimal-sqrt*:
$[\![x \in Infinitesimal;\ 0 \leq x]\!] \Longrightarrow (\ *f*\ sqrt)\ x \in Infinitesimal$
apply (*rule InfinitesimalI2*)
apply (*drule-tac r*=r^2 **in** *InfinitesimalD2*, *simp*)
apply (*simp add*: *hypreal-sqrt-ge-zero*)
apply (*rule hypreal-sqrt-lessI*, *simp-all*)
done

lemma *Infinitesimal-HComplex*:
$[\![x \in Infinitesimal;\ y \in Infinitesimal]\!] \Longrightarrow HComplex\ x\ y \in Infinitesimal$
apply (*rule Infinitesimal-hcmod-iff* [*THEN iffD2*])
apply (*simp add*: *hcmod-i*)
apply (*rule Infinitesimal-add*)
apply (*erule Infinitesimal-hrealpow*, *simp*)
apply (*erule Infinitesimal-hrealpow*, *simp*)
done

lemma *hcomplex-Infinitesimal-iff*:
$(x \in Infinitesimal) = (hRe\ x \in Infinitesimal \wedge hIm\ x \in Infinitesimal)$

apply (*safe intro*!: *Infinitesimal-hRe Infinitesimal-hIm*)
apply (*drule* (*1*) *Infinitesimal-HComplex*, *simp*)
done

lemma *hRe-diff* [*simp*]: $\bigwedge x\ y.\ hRe\ (x - y) = hRe\ x - hRe\ y$
by *transfer simp*

lemma *hIm-diff* [*simp*]: $\bigwedge x\ y.\ hIm\ (x - y) = hIm\ x - hIm\ y$
by *transfer simp*

lemma *approx-hRe*: $x \approx y \Longrightarrow hRe\ x \approx hRe\ y$
unfolding *approx-def* **by** (*drule Infinitesimal-hRe*) *simp*

lemma *approx-hIm*: $x \approx y \Longrightarrow hIm\ x \approx hIm\ y$
unfolding *approx-def* **by** (*drule Infinitesimal-hIm*) *simp*

lemma *approx-HComplex*:
$[\![a \approx b;\ c \approx d]\!] \Longrightarrow HComplex\ a\ c \approx HComplex\ b\ d$
unfolding *approx-def* **by** (*simp add*: *Infinitesimal-HComplex*)

lemma *hcomplex-approx-iff*:
$(x \approx y) = (hRe\ x \approx hRe\ y \wedge hIm\ x \approx hIm\ y)$
unfolding *approx-def* **by** (*simp add*: *hcomplex-Infinitesimal-iff*)

lemma *HFinite-hRe*: $x \in HFinite \Longrightarrow hRe\ x \in HFinite$
apply (*auto simp add*: *HFinite-def SReal-def*)
apply (*rule-tac x*=*star-of r* **in** *exI*, *simp*)
apply (*erule order-le-less-trans* [*OF abs-hRe-le-hcmod*])
done

lemma *HFinite-hIm*: $x \in HFinite \Longrightarrow hIm\ x \in HFinite$
apply (*auto simp add*: *HFinite-def SReal-def*)
apply (*rule-tac x*=*star-of r* **in** *exI*, *simp*)
apply (*erule order-le-less-trans* [*OF abs-hIm-le-hcmod*])
done

lemma *HFinite-HComplex*:
$[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow HComplex\ x\ y \in HFinite$
apply (*subgoal-tac HComplex x 0 + HComplex 0 y* $\in$ *HFinite*, *simp*)
apply (*rule HFinite-add*)
apply (*simp add*: *HFinite-hcmod-iff hcmod-i*)
apply (*simp add*: *HFinite-hcmod-iff hcmod-i*)
done

lemma *hcomplex-HFinite-iff*:
$(x \in HFinite) = (hRe\ x \in HFinite \wedge hIm\ x \in HFinite)$
apply (*safe intro*!: *HFinite-hRe HFinite-hIm*)
apply (*drule* (*1*) *HFinite-HComplex*, *simp*)
done

lemma *hcomplex-HInfinite-iff*:
(*x* ∈ *HInfinite*) = (*hRe x* ∈ *HInfinite* ∨ *hIm x* ∈ *HInfinite*)
by (*simp add*: *HInfinite-HFinite-iff hcomplex-HFinite-iff*)

lemma *hcomplex-of-hypreal-approx-iff* [*simp*]:
(*hcomplex-of-hypreal x* ≈ *hcomplex-of-hypreal z*) = (*x* ≈ *z*)
by (*simp add*: *hcomplex-approx-iff*)

lemma *Standard-HComplex*:
⟦*x* ∈ *Standard*; *y* ∈ *Standard*⟧ ⟹ *HComplex x y* ∈ *Standard*
by (*simp add*: *HComplex-def*)

lemma *stc-part-Ex*: *x*:*HFinite* ==> ∃*t* ∈ *SComplex*. *x* ≈ *t*
apply (*simp add*: *hcomplex-HFinite-iff hcomplex-approx-iff*)
apply (*rule-tac x*=*HComplex* (*st* (*hRe x*)) (*st* (*hIm x*)) **in** *bexI*)
apply (*simp add*: *st-approx-self* [*THEN approx-sym*])
apply (*simp add*: *Standard-HComplex st-SReal* [*unfolded Reals-eq-Standard*])
done

lemma *stc-part-Ex1*: *x*:*HFinite* ==> ∃!*t*. *t* ∈ *SComplex* & *x* ≈ *t*
apply (*drule stc-part-Ex*, *safe*)
apply (*drule-tac* [*2*] *approx-sym*, *drule-tac* [*2*] *approx-sym*, *drule-tac* [*2*] *approx-sym*)
apply (*auto intro*!: *approx-unique-complex*)
done

lemmas *hcomplex-of-complex-approx-inverse* =
hcomplex-of-complex-HFinite-diff-Infinitesimal [*THEN* [*2*] *approx-inverse*]

15.7 Theorems About Monads

lemma *monad-zero-hcmod-iff*: (*x* ∈ *monad 0*) = (*hcmod x*:*monad 0*)
by (*simp add*: *Infinitesimal-monad-zero-iff* [*symmetric*] *Infinitesimal-hcmod-iff*)

15.8 Theorems About Standard Part

lemma *stc-approx-self*: *x* ∈ *HFinite* ==> *stc x* ≈ *x*
apply (*simp add*: *stc-def*)
apply (*frule stc-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *stc-SComplex*: *x* ∈ *HFinite* ==> *stc x* ∈ *SComplex*
apply (*simp add*: *stc-def*)
apply (*frule stc-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *stc-HFinite*: $x \in HFinite \Longrightarrow stc\ x \in HFinite$
by (*erule stc-SComplex* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])

lemma *stc-unique*: $[\![y \in SComplex;\ y \approx x]\!] \Longrightarrow stc\ x = y$
apply (*frule Standard-subset-HFinite* [*THEN subsetD*])
apply (*drule* (*1*) *approx-HFinite*)
apply (*unfold stc-def*)
apply (*rule some-equality*)
apply (*auto intro*: *approx-unique-complex*)
done

lemma *stc-SComplex-eq* [*simp*]: $x \in SComplex \Longrightarrow stc\ x = x$
apply (*erule stc-unique*)
apply (*rule approx-refl*)
done

lemma *stc-hcomplex-of-complex*:
$stc\ (hcomplex\text{-}of\text{-}complex\ x) = hcomplex\text{-}of\text{-}complex\ x$
by *auto*

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 (*blast intro*: *approx-trans approx-trans2 SComplex-approx-iff* [*THEN iffD1*]
dest: *stc-approx-self stc-SComplex*)

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*:
$[\!|\ x \in SComplex;\ e \in Infinitesimal\ |\!] \Longrightarrow stc(x + e) = x$
apply (*erule stc-unique*)
apply (*erule Infinitesimal-add-approx-self*)
done

lemma *stc-Infinitesimal-add-SComplex2*:
$[\!|\ x \in SComplex;\ e \in Infinitesimal\ |\!] \Longrightarrow stc(e + x) = x$
apply (*erule stc-unique*)
apply (*erule Infinitesimal-add-approx-self2*)
done

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* ∈ *HFinite*; *y* ∈ *HFinite* |] ==> *stc* (*x* + *y*) = *stc*(*x*) + *stc*(*y*)
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-add*)

lemma *stc-numeral* [*simp*]: *stc* (*numeral w*) = *numeral w*
by (*rule Standard-numeral* [*THEN stc-SComplex-eq*])

lemma *stc-zero* [*simp*]: *stc 0* = *0*
by *simp*

lemma *stc-one* [*simp*]: *stc 1* = *1*
by *simp*

lemma *stc-minus*: *y* ∈ *HFinite* ==> *stc*(−*y*) = −*stc*(*y*)
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-minus*)

lemma *stc-diff*:
[| *x* ∈ *HFinite*; *y* ∈ *HFinite* |] ==> *stc* (*x*−*y*) = *stc*(*x*) − *stc*(*y*)
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-diff*)

lemma *stc-mult*:
[| *x* ∈ *HFinite*; *y* ∈ *HFinite* |]
==> *stc* (*x* ∗ *y*) = *stc*(*x*) ∗ *stc*(*y*)
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-mult-HFinite*)

lemma *stc-Infinitesimal*: *x* ∈ *Infinitesimal* ==> *stc x* = *0*
by (*simp add*: *stc-unique mem-infmal-iff*)

lemma *stc-not-Infinitesimal*: *stc*(*x*) ≠ *0* ==> *x* ∉ *Infinitesimal*
by (*fast intro*: *stc-Infinitesimal*)

lemma *stc-inverse*:
[| *x* ∈ *HFinite*; *stc x* ≠ *0* |]
==> *stc*(*inverse x*) = *inverse* (*stc x*)
apply (*drule stc-not-Infinitesimal*)
apply (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-inverse*)
done

lemma *stc-divide* [*simp*]:
[| *x* ∈ *HFinite*; *y* ∈ *HFinite*; *stc y* ≠ *0* |]
==> *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* ∈ *HFinite* ==> *stc*(*stc*(*x*)) = *stc*(*x*)
by (*blast intro*: *stc-HFinite stc-approx-self approx-stc-eq*)

lemma *HFinite-HFinite-hcomplex-of-hypreal*:

z ∈ *HFinite* ==> *hcomplex-of-hypreal z* ∈ *HFinite*
by (*simp add*: *hcomplex-HFinite-iff*)

lemma *SComplex-SReal-hcomplex-of-hypreal*:
x ∈ ℝ ==> *hcomplex-of-hypreal x* ∈ *SComplex*
apply (*rule Standard-of-hypreal*)
apply (*simp add*: *Reals-eq-Standard*)
done

lemma *stc-hcomplex-of-hypreal*:
z ∈ *HFinite* ==> *stc*(*hcomplex-of-hypreal z*) = *hcomplex-of-hypreal* (*st z*)
apply (*rule stc-unique*)
apply (*rule SComplex-SReal-hcomplex-of-hypreal*)
apply (*erule st-SReal*)
apply (*simp add*: *hcomplex-of-hypreal-approx-iff st-approx-self*)
done

lemma *Infinitesimal-hcnj-iff* [*simp*]:
(*hcnj z* ∈ *Infinitesimal*) = (*z* ∈ *Infinitesimal*)
by (*simp add*: *Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-hcomplex-of-hypreal-epsilon* [*simp*]:
hcomplex-of-hypreal ε ∈ *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* ∩ *SComplex* = *hcomplex-of-complex* ‘ *X*
by (*auto simp*: *Standard-def*)

lemma *lemma-not-hcomplexA*: *x* ∉ *hcomplex-of-complex* ‘ *A* ⟹ ∀ *y* ∈ *A*. *x* ≠ *hcomplex-of-complex y*
by *auto*

16.2 Theorems about Nonstandard Extensions of Functions

lemma *starfunC-hcpow*: ⋀*Z*. (∗*f*∗ (λ*z*. *z* ^ *n*)) *Z* = *Z pow hypnat-of-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 *hypreal-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-ring-1*. *P x*) $\longleftrightarrow$ ($\forall$ *x*. *P* (*x* $-$ *a*))
apply *auto*
apply (*drule-tac x* = *x* + *a* **in** *spec*)
apply (*simp add*: *add.assoc*)
done

lemma *complex-add-minus-iff* [*simp*]: $x + - a = 0 \longleftrightarrow x = a$
for $x\ a$ :: *complex*
by (*simp add*: *diff-eq-eq*)

lemma *complex-add-eq-0-iff* [*iff*]: $x + y = 0 \longleftrightarrow y = - x$
for $x\ y$ :: *complex*
apply *auto*
apply (*drule sym* [*THEN diff-eq-eq* [*THEN iffD2*]])
apply *auto*
done

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* $\Rightarrow$ *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* $\Rightarrow$ *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* $\Rightarrow$ *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* $\Rightarrow$ *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* $(Re\ x) = hRe\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

lemma *star-of-Im* [*simp*]: *star-of* $(Im\ x) = hIm\ (star\text{-}of\ x)$
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-h-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* $(\lambda x.\ Re\ (f\ x))\ a$
for f :: *'a::real-normed-vector* ⇒ *complex*
by (*simp add*: *isCont-def LIM-Re*)

lemma *isCont-Im*: *isCont f a* ⟹ *isCont* $(\lambda x.\ Im\ (f\ x))\ a$
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* $(\lambda x.\ x\ \hat{}\ n)\ x\ :>\ complex\text{-}of\text{-}real\ (real\ n)\ *\ (x\ \hat{}\ (n\ -\ Suc\ 0))$

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* (λ*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 $x \neq (0::'a)$: it isn't continuous at zero.

lemma *NSCDERIV-inverse*: *x* ≠ *0* ⟹ *NSDERIV* (λ*x*. *inverse x*) *x* :> − (*inverse x*)^2
for *x* :: *complex*
unfolding *numeral-2-eq-2* **by** (*rule NSDERIV-inverse*)

lemma *CDERIV-inverse*: *x* ≠ *0* ⟹ *DERIV* (λ*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* ⟹ *f x* ≠ *0* ⟹ *DERIV* (λ*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* ⟹ *f x* ≠ *0* ⟹ *NSDERIV* (λ*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* ⟹ *DERIV g x* :> *e* ⟹ *g*(*x*) ≠ *0* ⟹
DERIV (λ*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* ⟹ *NSDERIV g x* :> *e* ⟹ *g x* ≠ (*0*::*complex*) ⟹
NSDERIV (λ*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*:
(∀ *z*. *f z* − *f x* = *g z* ∗ (*z* − *x*)) ∧ *isNSCont g x* ∧ *g x* = *l* ⟹ *NSDERIV f x* :> *l*
by *clarify* (*rule CARAT-DERIVD*)

end

18 Logarithms: Non-Standard Version

theory *HLog*
imports *HTranscendental*
begin

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

lemma *hpfinite-witness*: ε ∈ {*x*. 0 ≤ *x* ∧ *x* ∈ *HFinite*}
by *auto*

definition *powhr* :: *hypreal* ⇒ *hypreal* ⇒ *hypreal* (**infixr** *powhr* 80)
where [*transfer-unfold*]: *x powhr a* = *starfun2* (*op 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*]: ⋀*a*. 1 *powhr a* = 1
by *transfer simp*

lemma *powhr-mult*: ⋀*a x y*. 0 < *x* ⟹ 0 < *y* ⟹ (*x* ∗ *y*) *powhr a* = (*x powhr a*) ∗ (*y powhr a*)
by *transfer* (*simp add*: *powr-mult*)

lemma *powhr-gt-zero* [*simp*]: ⋀*a x*. 0 < *x powhr a* ⟷ *x* ≠ 0
by *transfer simp*

lemma *powhr-not-zero* [*simp*]: ⋀*a x*. *x powhr a* ≠ 0 ⟷ *x* ≠ 0
by *transfer simp*

lemma *powhr-divide*: ⋀*a x y*. 0 < *x* ⟹ 0 < *y* ⟹ (*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*]: $1 < x \Longrightarrow x\ powhr\ a \leq x\ powhr\ b \longleftrightarrow a \leq b$
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*]: $\bigwedge a\ x.\ 0 < a \Longrightarrow a \neq 1 \Longrightarrow 0 < x \Longrightarrow a\ powhr\ (hlog\ a\ x) = x$
by *transfer simp*

lemma *hlog-powhr-cancel* [*simp*]: $\bigwedge a\ y.\ 0 < a \Longrightarrow a \neq 1 \Longrightarrow hlog\ a\ (a\ powhr\ y) = y$
by *transfer simp*

lemma *hlog-mult*:
$\bigwedge a\ x\ y.\ 0 < a \Longrightarrow a \neq 1 \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow hlog\ a\ (x * y) = hlog\ a\ x + hlog\ a\ y$
by *transfer* (*rule log-mult*)

lemma *hlog-as-starfun*: $\bigwedge a\ x.\ 0 < a \Longrightarrow a \neq 1 \Longrightarrow hlog\ a\ x = (\ *f*\ ln)\ x\ /\ ($

$*f*\ ln)\ a$
by *transfer* (*simp add*: *log-def*)

lemma *hlog-eq-div-starfun-ln-mult-hlog*:
$\bigwedge a\ b\ x.\ 0 < a \Longrightarrow a \neq 1 \Longrightarrow 0 < b \Longrightarrow b \neq 1 \Longrightarrow 0 < x \Longrightarrow$
$hlog\ a\ x = ((\ *f*\ ln)\ b\ /\ (\ *f*\ ln)\ a) * hlog\ b\ x$
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*: $0 < a \Longrightarrow a \neq 1 \Longrightarrow 0 < x \Longrightarrow hlog\ a\ (inverse\ x) = -\ hlog\ a\ x$
by (*rule add-left-cancel* [*of hlog a x*, *THEN iffD1*]) (*simp add*: *hlog-mult* [*symmetric*])

lemma *hlog-divide*: $0 < a \Longrightarrow a \neq 1 \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow hlog\ a\ (x\ /\ y) = hlog\ a\ x - hlog\ a\ y$
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