

Isabelle/HOL-NSA — Non-Standard Analysis

May 22, 2012

Contents

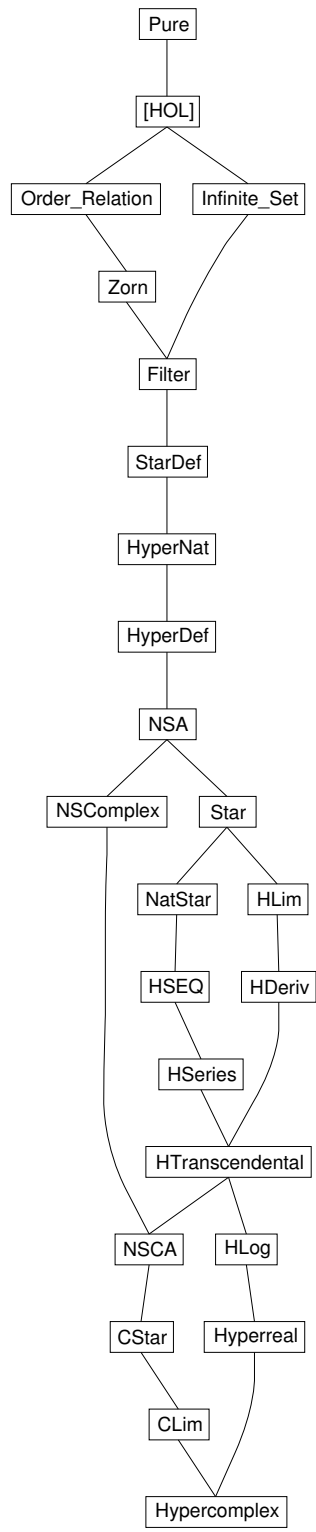
| | | |
|----------|--|-----------|
| 1 | Order-Relation: Orders as Relations | 7 |
| 1.1 | Orders on a set | 7 |
| 1.2 | Orders on the field | 8 |
| 1.3 | Orders on a type | 8 |
| 2 | Zorn: Zorn’s Lemma | 9 |
| 2.1 | Mathematical Preamble | 9 |
| 2.2 | Hausdorff’s Theorem: Every Set Contains a Maximal Chain. | 11 |
| 2.3 | Zorn’s Lemma: If All Chains Have Upper Bounds Then There Is a Maximal Element | 12 |
| 2.4 | Alternative version of Zorn’s Lemma | 13 |
| 3 | Infinite-Set: Infinite Sets and Related Concepts | 19 |
| 3.1 | Infinite Sets | 19 |
| 3.2 | Infinitely Many and Almost All | 26 |
| 3.3 | Enumeration of an Infinite Set | 29 |
| 3.4 | Miscellaneous | 30 |
| 4 | Filter: Filters and Ultrafilters | 30 |
| 4.1 | Definitions and basic properties | 30 |
| 4.1.1 | Filters | 30 |
| 4.1.2 | Ultrafilters | 31 |
| 4.1.3 | Free Ultrafilters | 32 |
| 4.2 | Collect properties | 32 |
| 4.3 | Maximal filter = Ultrafilter | 33 |
| 4.4 | Ultrafilter Theorem | 34 |
| 4.4.1 | Unions of chains of superfrechets | 35 |
| 4.4.2 | Existence of free ultrafilter | 37 |

| | | |
|----------|--|-----------|
| 5 | StarDef: Construction of Star Types Using Ultrafilters | 38 |
| 5.1 | A Free Ultrafilter over the Naturals | 39 |
| 5.2 | Definition of <i>star</i> type constructor | 39 |
| 5.3 | Transfer principle | 40 |
| 5.4 | Standard elements | 42 |
| 5.5 | Internal functions | 42 |
| 5.6 | Internal predicates | 44 |
| 5.7 | Internal sets | 45 |
| 5.8 | Syntactic classes | 47 |
| 5.9 | Ordering and lattice classes | 52 |
| 5.10 | Ordered group classes | 54 |
| 5.11 | Ring and field classes | 55 |
| 5.12 | Power | 57 |
| 5.13 | Number classes | 57 |
| 5.14 | Finite class | 59 |
| 6 | HyperNat: Hypernatural numbers | 59 |
| 6.1 | Properties Transferred from Naturals | 59 |
| 6.2 | Properties of the set of embedded natural numbers | 62 |
| 6.3 | Infinite Hypernatural Numbers – <i>HNatInfinite</i> | 62 |
| 6.3.1 | Closure Rules | 63 |
| 6.4 | Existence of an infinite hypernatural number | 64 |
| 6.4.1 | Alternative characterization of the set of infinite hypernaturals | 65 |
| 6.4.2 | Alternative Characterization of <i>HNatInfinite</i> using Free Ultrafilter | 66 |
| 6.5 | Embedding of the Hypernaturals into other types | 66 |
| 7 | HyperDef: Construction of Hyperreals Using Ultrafilters | 67 |
| 7.1 | Real vector class instances | 68 |
| 7.2 | Injection from <i>hypreal</i> | 70 |
| 7.3 | Properties of <i>starrel</i> | 71 |
| 7.4 | <i>hypreal-of-real</i> : the Injection from <i>real</i> to <i>hypreal</i> | 71 |
| 7.5 | Properties of <i>star-n</i> | 71 |
| 7.6 | Misc Others | 72 |
| 7.7 | Existence of Infinite Hyperreal Number | 72 |
| 7.8 | Absolute Value Function for the Hyperreals | 73 |
| 7.9 | Embedding the Naturals into the Hyperreals | 74 |
| 7.10 | Exponentials on the Hyperreals | 74 |
| 7.11 | Powers with Hypernatural Exponents | 76 |

| | |
|---|------------|
| 8 NSA: Infinite Numbers, Infinitesimals, Infinitely Close Relation | 79 |
| 8.1 Nonstandard Extension of the Norm Function | 80 |
| 8.2 Closure Laws for the Standard Reals | 82 |
| 8.3 Set of Finite Elements is a Subring of the Extended Reals . . | 84 |
| 8.4 Set of Infinitesimals is a Subring of the Hyperreals | 85 |
| 8.5 The Infinitely Close Relation | 91 |
| 8.6 Zero is the Only Infinitesimal that is also a Real | 97 |
| 8.7 Uniqueness: Two Infinitely Close Reals are Equal | 99 |
| 8.8 Existence of Unique Real Infinitely Close | 100 |
| 8.8.1 Lifting of the Ub and Lub Properties | 100 |
| 8.9 Finite, Infinite and Infinitesimal | 105 |
| 8.10 Theorems about Monads | 109 |
| 8.11 Proof that $x \approx y$ implies $ x \approx y $ | 110 |
| 8.12 More <i>HFinite</i> and <i>Infinitesimal</i> Theorems | 112 |
| 8.13 Theorems about Standard Part | 115 |
| 8.14 Alternative Definitions using Free Ultrafilter | 118 |
| 8.14.1 <i>HFinite</i> | 118 |
| 8.14.2 <i>HInfinite</i> | 118 |
| 8.14.3 <i>Infinitesimal</i> | 120 |
| 8.15 Proof that ω is an infinite number | 121 |
| 9 NSComplex: Nonstandard Complex Numbers | 124 |
| 9.1 Properties of Nonstandard Real and Imaginary Parts | 127 |
| 9.2 Addition for Nonstandard Complex Numbers | 127 |
| 9.3 More Minus Laws | 127 |
| 9.4 More Multiplication Laws | 128 |
| 9.5 Subtraction and Division | 128 |
| 9.6 Embedding Properties for <i>hcomplex-of-hypreal</i> Map | 128 |
| 9.7 HComplex theorems | 128 |
| 9.8 Modulus (Absolute Value) of Nonstandard Complex Number | 129 |
| 9.9 Conjugation | 130 |
| 9.10 More Theorems about the Function <i>hcmmod</i> | 131 |
| 9.11 Exponentiation | 131 |
| 9.12 The Function <i>hsgn</i> | 132 |
| 9.13 Polar Form for Nonstandard Complex Numbers | 134 |
| 9.14 <i>hcomplex-of-complex</i> : the Injection from type <i>complex</i> to to <i>hcomplex</i> | 136 |
| 9.15 Numerals and Arithmetic | 137 |
| 10 Star: Star-Transforms in Non-Standard Analysis | 138 |
| 10.1 Properties of the Star-transform Applied to Sets of Reals . . | 138 |

| | |
|---|------------|
| 11 NatStar: Star-transforms for the Hypernaturals | 144 |
| 11.1 Nonstandard Extensions of Functions | 145 |
| 11.2 Nonstandard Characterization of Induction | 147 |
| 12 HSEQ: Sequences and Convergence (Nonstandard) | 149 |
| 12.1 Limits of Sequences | 149 |
| 12.1.1 Equivalence of <i>LIMSEQ</i> and <i>NSLIMSEQ</i> | 152 |
| 12.1.2 Derived theorems about <i>NSLIMSEQ</i> | 153 |
| 12.2 Convergence | 154 |
| 12.3 Bounded Monotonic Sequences | 154 |
| 12.3.1 Upper Bounds and Lubs of Bounded Sequences | 156 |
| 12.3.2 A Bounded and Monotonic Sequence Converges | 156 |
| 12.4 Cauchy Sequences | 156 |
| 12.4.1 Equivalence Between NS and Standard | 157 |
| 12.4.2 Cauchy Sequences are Bounded | 158 |
| 12.4.3 Cauchy Sequences are Convergent | 158 |
| 12.5 Power Sequences | 159 |
| 13 HSeries: Finite Summation and Infinite Series for Hyperreals | 159 |
| 13.1 Nonstandard Sums | 161 |
| 14 HLim: Limits and Continuity (Nonstandard) | 163 |
| 14.1 Limits of Functions | 164 |
| 14.1.1 Equivalence of <i>LIM</i> and <i>NSLIM</i> | 166 |
| 14.2 Continuity | 167 |
| 14.3 Uniform Continuity | 169 |
| 15 HDeriv: Differentiation (Nonstandard) | 170 |
| 15.1 Derivatives | 171 |
| 15.1.1 Equivalence of NS and Standard definitions | 177 |
| 15.1.2 Differentiability predicate | 178 |
| 15.2 (NS) Increment | 178 |
| 16 HTranscendental: Nonstandard Extensions of Transcendental Functions | 179 |
| 16.1 Nonstandard Extension of Square Root Function | 180 |
| 17 NSCA: Non-Standard Complex Analysis | 191 |
| 17.1 Closure Laws for SComplex, the Standard Complex Numbers | 192 |
| 17.2 The Finite Elements form a Subring | 193 |
| 17.3 The Complex Infinitesimals form a Subring | 193 |
| 17.4 The “Infinitely Close” Relation | 194 |
| 17.5 Zero is the Only Infinitesimal Complex Number | 195 |
| 17.6 Properties of <i>hRe</i> , <i>hIm</i> and <i>HComplex</i> | 196 |

| | |
|--|------------|
| 17.7 Theorems About Monads | 199 |
| 17.8 Theorems About Standard Part | 199 |
| 18 CStar: Star-transforms in NSA, Extending Sets of Complex Numbers and Complex Functions | 202 |
| 18.1 Properties of the *-Transform Applied to Sets of Reals | 202 |
| 18.2 Theorems about Nonstandard Extensions of Functions | 202 |
| 18.3 Internal Functions - Some Redundancy With *f* Now | 202 |
| 19 CLim: Limits, Continuity and Differentiation for Complex Functions | 203 |
| 19.1 Limit of Complex to Complex Function | 203 |
| 19.2 Continuity | 205 |
| 19.3 Functions from Complex to Reals | 205 |
| 19.4 Differentiation of Natural Number Powers | 205 |
| 19.5 Derivative of Reciprocals (Function <i>inverse</i>) | 206 |
| 19.6 Derivative of Quotient | 206 |
| 19.7 Caratheodory Formulation of Derivative at a Point: Standard Proof | 206 |
| 20 HLog: Logarithms: Non-Standard Version | 206 |



1 Order-Relation: Orders as Relations

```
theory Order-Relation
imports Main
begin
```

1.1 Orders on a set

definition *preorder-on* $A\ r \equiv \text{refl-on } A\ r \wedge \text{trans } r$

definition *partial-order-on* $A\ r \equiv \text{preorder-on } A\ r \wedge \text{antisym } r$

definition *linear-order-on* $A\ r \equiv \text{partial-order-on } A\ r \wedge \text{total-on } A\ r$

definition *strict-linear-order-on* $A\ r \equiv \text{trans } r \wedge \text{irrefl } r \wedge \text{total-on } A\ r$

definition *well-order-on* $A\ r \equiv \text{linear-order-on } A\ r \wedge \text{wf}(r - \text{Id})$

lemmas *order-on-defs* =
preorder-on-def partial-order-on-def linear-order-on-def
strict-linear-order-on-def well-order-on-def

lemma *preorder-on-empty[simp]*: *preorder-on* $\{\}\ \{\}$
by (*simp add:preorder-on-def trans-def*)

lemma *partial-order-on-empty[simp]*: *partial-order-on* $\{\}\ \{\}$
by (*simp add:partial-order-on-def*)

lemma *linear-order-on-empty[simp]*: *linear-order-on* $\{\}\ \{\}$
by (*simp add:linear-order-on-def*)

lemma *well-order-on-empty[simp]*: *well-order-on* $\{\}\ \{\}$
by (*simp add:well-order-on-def*)

lemma *preorder-on-converse[simp]*: *preorder-on* $A\ (r^{-1}) = \text{preorder-on } A\ r$
by (*simp add:preorder-on-def*)

lemma *partial-order-on-converse[simp]*:
partial-order-on $A\ (r^{-1}) = \text{partial-order-on } A\ r$
by (*simp add: partial-order-on-def*)

lemma *linear-order-on-converse[simp]*:
linear-order-on $A\ (r^{-1}) = \text{linear-order-on } A\ r$
by (*simp add: linear-order-on-def*)

lemma *strict-linear-order-on-diff-Id*:
linear-order-on $A\ r \implies \text{strict-linear-order-on } A\ (r - \text{Id})$

by(*simp add: order-on-defs trans-diff-Id*)

1.2 Orders on the field

abbreviation *Refl* $r \equiv \text{refl-on (Field } r) r$

abbreviation *Preorder* $r \equiv \text{preorder-on (Field } r) r$

abbreviation *Partial-order* $r \equiv \text{partial-order-on (Field } r) r$

abbreviation *Total* $r \equiv \text{total-on (Field } r) r$

abbreviation *Linear-order* $r \equiv \text{linear-order-on (Field } r) r$

abbreviation *Well-order* $r \equiv \text{well-order-on (Field } r) r$

lemma *subset-Image-Image-iff*:

$\llbracket \text{Preorder } r; A \subseteq \text{Field } r; B \subseteq \text{Field } r \rrbracket \implies$
 $r \text{ “ } A \subseteq r \text{ “ } B \iff (\forall a \in A. \exists b \in B. (b, a):r)$

apply(*auto simp add: subset-eq preorder-on-def refl-on-def Image-def*)

apply *metis*

by(*metis trans-def*)

lemma *subset-Image1-Image1-iff*:

$\llbracket \text{Preorder } r; a : \text{Field } r; b : \text{Field } r \rrbracket \implies r \text{ “ } \{a\} \subseteq r \text{ “ } \{b\} \iff (b, a):r$
by(*simp add: subset-Image-Image-iff*)

lemma *Refl-antisym-eq-Image1-Image1-iff*:

$\llbracket \text{Refl } r; \text{antisym } r; a : \text{Field } r; b : \text{Field } r \rrbracket \implies r \text{ “ } \{a\} = r \text{ “ } \{b\} \iff a=b$
by(*simp add: set-eq-iff antisym-def refl-on-def*) *metis*

lemma *Partial-order-eq-Image1-Image1-iff*:

$\llbracket \text{Partial-order } r; a : \text{Field } r; b : \text{Field } r \rrbracket \implies r \text{ “ } \{a\} = r \text{ “ } \{b\} \iff a=b$
by(*auto simp: order-on-defs Refl-antisym-eq-Image1-Image1-iff*)

1.3 Orders on a type

abbreviation *strict-linear-order* $\equiv \text{strict-linear-order-on UNIV}$

abbreviation *linear-order* $\equiv \text{linear-order-on UNIV}$

abbreviation *well-order* $r \equiv \text{well-order-on UNIV}$

end

2 Zorn: Zorn’s Lemma

```
theory Zorn
imports Order-Relation Main
begin
```

```
definition chain-subset :: 'a set set  $\Rightarrow$  bool (chain $\subseteq$ )
where
  chain $\subseteq$  C  $\equiv$   $\forall A \in C. \forall B \in C. A \subseteq B \vee B \subseteq A$ 
```

The lemma and section numbers refer to an unpublished article [?].

```
definition chain :: 'a set set  $\Rightarrow$  'a set set set
where
  chain S = {F. F  $\subseteq$  S  $\wedge$  chain $\subseteq$  F}
```

```
definition super :: 'a set set  $\Rightarrow$  'a set set  $\Rightarrow$  'a set set set
where
  super S c = {d. d  $\in$  chain S  $\wedge$  c  $\subset$  d}
```

```
definition maxchain :: 'a set set  $\Rightarrow$  'a set set set
where
  maxchain S = {c. c  $\in$  chain S  $\wedge$  super S c = {}}
```

```
definition succ :: 'a set set  $\Rightarrow$  'a set set  $\Rightarrow$  'a set set
where
  succ S c = (if c  $\notin$  chain S  $\vee$  c  $\in$  maxchain S then c else SOME c'. c'  $\in$  super S c)
```

```
inductive-set TFin :: 'a set set  $\Rightarrow$  'a set set set
for S :: 'a set set
where
  succI: x  $\in$  TFin S  $\implies$  succ S x  $\in$  TFin S
  | Pow-UnionI: Y  $\in$  Pow (TFin S)  $\implies$   $\bigcup$  Y  $\in$  TFin S
```

2.1 Mathematical Preamble

```
lemma Union-lemma0:
  ( $\forall x \in C. x \subseteq A \mid B \subseteq x$ )  $\implies$  Union(C)  $\subseteq$  A  $\mid$  B  $\subseteq$  Union(C)
by blast
```

This is theorem *increasingD2* of ZF/Zorn.thy

```
lemma Abrial-axiom1: x  $\subseteq$  succ S x
apply (auto simp add: succ-def super-def maxchain-def)
apply (rule contrapos-np, assumption)
apply (rule someI2)
apply blast+
done
```

lemmas *TFin-UnionI* = *TFin.Pow-UnionI* [*OF PowI*]

lemma *TFin-induct*:

assumes *H*: $n \in TFin\ S$ **and**

I: $\forall x. x \in TFin\ S \implies P\ x \implies P\ (succ\ S\ x)$

$\forall Y. Y \subseteq TFin\ S \implies Ball\ Y\ P \implies P\ (Union\ Y)$

shows $P\ n$

using *H* **by** *induct* (*blast intro: I*)+

lemma *succ-trans*: $x \subseteq y \implies x \subseteq succ\ S\ y$

apply (*erule subset-trans*)

apply (*rule Abrial-axiom1*)

done

Lemma 1 of section 3.1

lemma *TFin-linear-lemma1*:

$[[\ n \in TFin\ S; m \in TFin\ S;$

$\forall x \in TFin\ S. x \subseteq m \longrightarrow x = m \mid succ\ S\ x \subseteq m$

$]] \implies n \subseteq m \mid succ\ S\ m \subseteq n$

apply (*erule TFin-induct*)

apply (*erule-tac* [2] *Union-lemma0*)

apply (*blast del: subsetI intro: succ-trans*)

done

Lemma 2 of section 3.2

lemma *TFin-linear-lemma2*:

$m \in TFin\ S \implies \forall n \in TFin\ S. n \subseteq m \longrightarrow n = m \mid succ\ S\ n \subseteq m$

apply (*erule TFin-induct*)

apply (*rule impI* [*THEN ballI*])

case split using *TFin-linear-lemma1*

apply (*rule-tac* $n1 = n$ **and** $m1 = x$ **in** *TFin-linear-lemma1* [*THEN disjE*],
assumption+)

apply (*erule-tac* $x = n$ **in** *bspec*, *assumption*)

apply (*blast del: subsetI intro: succ-trans, blast*)

second induction step

apply (*rule impI* [*THEN ballI*])

apply (*rule Union-lemma0* [*THEN disjE*])

apply (*rule-tac* [3] *disjI2*)

prefer 2 **apply** *blast*

apply (*rule ballI*)

apply (*rule-tac* $n1 = n$ **and** $m1 = x$ **in** *TFin-linear-lemma1* [*THEN disjE*],
assumption+, *auto*)

apply (*blast intro!: Abrial-axiom1* [*THEN subsetD*])

done

Re-ordering the premises of Lemma 2

lemma *TFin-subsetD*:

$[[n \subseteq m; m \in TFin\ S; n \in TFin\ S]] ==> n=m \mid succ\ S\ n \subseteq m$
by (rule *TFin-linear-lemma2* [rule-format])

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*: $[[m \in TFin\ S; n \in TFin\ S]] ==> n \subseteq m \mid m \subseteq n$
apply (rule *disjE*)
apply (rule *TFin-linear-lemma1* [*OF - -TFin-linear-lemma2*])
apply (assumption+, erule *disjI2*)
apply (blast del: *subsetI*
intro: *subsetI* *Abrial-axiom1* [*THEN subset-trans*])
done

Lemma 3 of section 3.3

lemma *eq-succ-upper*: $[[n \in TFin\ S; m \in TFin\ S; m = succ\ S\ m]] ==> n \subseteq m$
apply (erule *TFin-induct*)
apply (drule *TFin-subsetD*)
apply (assumption+, force, blast)
done

Property 3.3 of section 3.3

lemma *equal-succ-Union*: $m \in TFin\ S ==> (m = succ\ S\ m) = (m = Union(TFin\ S))$
apply (rule *iffI*)
apply (rule *Union-upper* [*THEN equalityI*])
apply assumption
apply (rule *eq-succ-upper* [*THEN Union-least*], assumption+)
apply (erule *ssubst*)
apply (rule *Abrial-axiom1* [*THEN equalityI*])
apply (blast del: *subsetI* intro: *subsetI* *TFin-UnionI* *TFin.succI*)
done

2.2 Hausdorff’s Theorem: Every Set Contains a Maximal Chain.

NB: We assume the partial ordering is \subseteq , the subset relation!

lemma *empty-set-mem-chain*: $(\{\} :: 'a\ set\ set) \in chain\ S$
by (unfold *chain-def* *chain-subset-def*) auto

lemma *super-subset-chain*: $super\ S\ c \subseteq chain\ S$
by (unfold *super-def*) blast

lemma *maxchain-subset-chain*: $maxchain\ S \subseteq chain\ S$
by (unfold *maxchain-def*) blast

lemma *mem-super-Ex*: $c \in chain\ S - maxchain\ S ==> EX\ d. d \in super\ S\ c$
by (unfold *super-def* *maxchain-def*) auto

lemma *select-super*:

```

  c ∈ chain S – maxchain S ==> (∃ c'. c': super S c): super S c
  apply (erule mem-super-Ex [THEN exE])
  apply (rule someI2)
  apply auto
  done

```

lemma *select-not-equals*:

```

  c ∈ chain S – maxchain S ==> (∃ c'. c': super S c) ≠ c
  apply (rule notI)
  apply (drule select-super)
  apply (simp add: super-def less-le)
  done

```

lemma *succI3*: $c \in \text{chain } S - \text{maxchain } S \implies \text{succ } S c = (\exists c'. c': \text{super } S c)$
 by (unfold succ-def) (blast intro!: if-not-P)

lemma *succ-not-equals*: $c \in \text{chain } S - \text{maxchain } S \implies \text{succ } S c \neq c$

```

  apply (frule succI3)
  apply (simp (no-asm-simp))
  apply (rule select-not-equals, assumption)
  done

```

lemma *TFin-chain-lemma4*: $c \in \text{TFin } S \implies (c :: 'a \text{ set set}): \text{chain } S$

```

  apply (erule TFin-induct)
  apply (simp add: succ-def select-super [THEN super-subset-chain[THEN subsetD]])
  apply (unfold chain-def chain-subset-def)
  apply (rule CollectI, safe)
  apply (drule bspec, assumption)
  apply (rule-tac [2] m1 = Xa and n1 = X in TFin-subset-linear [THEN disjE])
  apply blast+
  done

```

theorem *Hausdorff*: $\exists c. (c :: 'a \text{ set set}): \text{maxchain } S$

```

  apply (rule-tac x = Union (TFin S) in exI)
  apply (rule classical)
  apply (subgoal-tac succ S (Union (TFin S)) = Union (TFin S) )
  prefer 2
  apply (blast intro!: TFin-UnionI equal-succ-Union [THEN iffD2, symmetric])
  apply (cut-tac subset-refl [THEN TFin-UnionI, THEN TFin-chain-lemma4])
  apply (drule DiffI [THEN succ-not-equals], blast+)
  done

```

2.3 Zorn’s Lemma: If All Chains Have Upper Bounds Then There Is a Maximal Element

lemma *chain-extend*:

```

  [| c ∈ chain S; z ∈ S; ∀ x ∈ c. x ⊆ (z :: 'a set) |] ==> {z} Un c ∈ chain S

```

by (unfold chain-def chain-subset-def) blast

lemma chain-Union-upper: $[[c \in \text{chain } S; x \in c]] \implies x \subseteq \text{Union}(c)$
by auto

lemma chain-ball-Union-upper: $c \in \text{chain } S \implies \forall x \in c. x \subseteq \text{Union}(c)$
by auto

lemma maxchain-Zorn:
 $[[c \in \text{maxchain } S; u \in S; \text{Union}(c) \subseteq u]] \implies \text{Union}(c) = u$
 apply (rule ccontr)
 apply (simp add: maxchain-def)
 apply (erule conjE)
 apply (subgoal-tac $(\{u\} \cup c) \in \text{super } S c$)
 apply simp
 apply (unfold super-def less-le)
 apply (blast intro: chain-extend dest: chain-Union-upper)
 done

theorem Zorn-Lemma:
 $\forall c \in \text{chain } S. \text{Union}(c): S \implies \exists y \in S. \forall z \in S. y \subseteq z \longrightarrow y = z$
 apply (cut-tac Hausdorff maxchain-subset-chain)
 apply (erule exE)
 apply (drule subsetD, assumption)
 apply (drule bspec, assumption)
 apply (rule-tac $x = \text{Union}(c)$ in bexI)
 apply (rule ballI, rule impI)
 apply (blast dest!: maxchain-Zorn, assumption)
 done

2.4 Alternative version of Zorn’s Lemma

lemma Zorn-Lemma2:
 $\forall c \in \text{chain } S. \exists y \in S. \forall x \in c. x \subseteq y$
 $\implies \exists y \in S. \forall x \in S. (y :: 'a \text{ set}) \subseteq x \longrightarrow y = x$
 apply (cut-tac Hausdorff maxchain-subset-chain)
 apply (erule exE)
 apply (drule subsetD, assumption)
 apply (drule bspec, assumption, erule bexE)
 apply (rule-tac $x = y$ in bexI)
 prefer 2 apply assumption
 apply clarify
 apply (rule ccontr)
 apply (frule-tac $z = x$ in chain-extend)
 apply (assumption, blast)
 apply (unfold maxchain-def super-def less-le)
 apply (blast elim!: equalityCE)
 done

Various other lemmas

lemma chainD: $[[c \in \text{chain } S; x \in c; y \in c]] \implies x \subseteq y \mid y \subseteq x$
by (unfold chain-def chain-subset-def) blast

lemma chainD2: $!!(c :: 'a \text{ set set}). c \in \text{chain } S \implies c \subseteq S$
by (unfold chain-def) blast

definition Chain $:: ('a * 'a) \text{set} \Rightarrow 'a \text{ set set}$ **where**
 $\text{Chain } r \equiv \{A. \forall a \in A. \forall b \in A. (a, b) : r \vee (b, a) \in r\}$

lemma mono-Chain: $r \subseteq s \implies \text{Chain } r \subseteq \text{Chain } s$
unfolding Chain-def **by** blast

Zorn’s lemma for partial orders:

lemma Zorns-po-lemma:

assumes po: *Partial-order* r **and** u: $\forall C \in \text{Chain } r. \exists u \in \text{Field } r. \forall a \in C. (a, u) : r$
shows $\exists m \in \text{Field } r. \forall a \in \text{Field } r. (m, a) : r \longrightarrow a = m$

proof –

have *Preorder* r **using** po **by** (simp add: partial-order-on-def)

— Mirror r in the set of subsets below (wrt r) elements of A

let $?B = \%x. r^{-1} \{x\}$ **let** $?S = ?B \text{ ‘ Field } r$

have $\forall C \in \text{chain } ?S. \exists U : ?S. \text{ALL } A : C. A \subseteq U$

proof (auto simp: chain-def chain-subset-def)

fix C **assume** 1: $C \subseteq ?S$ **and** 2: $\forall A \in C. \forall B \in C. A \subseteq B \mid B \subseteq A$

let $?A = \{x \in \text{Field } r. \exists M \in C. M = ?B x\}$

have $C = ?B \text{ ‘ } ?A$ **using** 1 **by** (auto simp: image-def)

have $?A \in \text{Chain } r$

proof (simp add: Chain-def, intro allI impI, elim conjE)

fix $a b$

assume $a \in \text{Field } r \ ?B a \in C \ b \in \text{Field } r \ ?B b \in C$

hence $?B a \subseteq ?B b \vee ?B b \subseteq ?B a$ **using** 2 **by** auto

thus $(a, b) \in r \vee (b, a) \in r$ **using** $\langle \text{Preorder } r \rangle \langle a : \text{Field } r \rangle \langle b : \text{Field } r \rangle$

by (simp add: subset-Image1-Image1-iff)

qed

then obtain u **where** $uA: u : \text{Field } r \ \forall a \in ?A. (a, u) : r$ **using** u **by** auto

have $\forall A \in C. A \subseteq r^{-1} \{u\}$ (**is** $?P u$)

proof auto

fix $a B$ **assume** $aB: B : C \ a : B$

with 1 **obtain** x **where** $x : \text{Field } r \ B = r^{-1} \{x\}$ **by** auto

thus $(a, u) : r$ **using** $uA \ aB \ \langle \text{Preorder } r \rangle$

by (auto simp add: preorder-on-def refl-on-def) (metis transD)

qed

thus $\exists u : \text{Field } r. ?P u$ **using** $\langle u : \text{Field } r \rangle$ **by** blast

qed

from Zorn-Lemma2 [OF this]

obtain $m B$ **where** $m : \text{Field } r \ B = r^{-1} \{m\}$

$\forall x \in \text{Field } r. B \subseteq r^{-1} \{x\} \longrightarrow B = r^{-1} \{x\}$

by auto

hence $\forall a \in \text{Field } r. (m, a) \in r \longrightarrow a = m$ using *po* $\langle \text{Preorder } r \rangle$ $\langle m:\text{Field } r \rangle$
 by $(\text{auto simp}:\text{subset-Image1-Image1-iff } \text{Partial-order-eq-Image1-Image1-iff})$
 thus $?thesis$ using $\langle m:\text{Field } r \rangle$ by *blast*
qed

definition *init-seg-of* :: $((a*'a)\text{set} * (a*'a)\text{set})\text{set}$ **where**
init-seg-of == $\{(r,s). r \subseteq s \wedge (\forall a b c. (a,b):s \wedge (b,c):r \longrightarrow (a,b):r)\}$

abbreviation *initialSegmentOf* :: $(a*'a)\text{set} \Rightarrow (a*'a)\text{set} \Rightarrow \text{bool}$
 (**infix** *initial'-segment'-of* 55) **where**
r initial-segment-of s == $(r,s):\text{init-seg-of}$

lemma *refl-on-init-seg-of* [*simp*]: *r initial-segment-of r*
 by $(\text{simp add}:\text{init-seg-of-def})$

lemma *trans-init-seg-of*:
r initial-segment-of s \Longrightarrow *s initial-segment-of t* \Longrightarrow *r initial-segment-of t*
 by $(\text{simp (no-asm-use) add}:\text{init-seg-of-def})$
 (*metis Domain-iff UnCI Un-absorb2 subset-trans*)

lemma *antisym-init-seg-of*:
r initial-segment-of s \Longrightarrow *s initial-segment-of r* \Longrightarrow $r=s$
 unfolding *init-seg-of-def* by *safe*

lemma *Chain-init-seg-of-Union*:
 $R \in \text{Chain } \text{init-seg-of} \Longrightarrow r \in R \Longrightarrow r \text{ initial-segment-of } \bigcup R$
 by $(\text{auto simp add}:\text{init-seg-of-def } \text{Chain-def } \text{Ball-def})$ *blast*

lemma *chain-subset-trans-Union*:
 $\text{chain}_{\subseteq} R \Longrightarrow \forall r \in R. \text{trans } r \Longrightarrow \text{trans}(\bigcup R)$
 apply $(\text{auto simp add}:\text{chain-subset-def})$
 apply $(\text{simp (no-asm-use) add}:\text{trans-def})$
 apply (metis subsetD)
 done

lemma *chain-subset-antisym-Union*:
 $\text{chain}_{\subseteq} R \Longrightarrow \forall r \in R. \text{antisym } r \Longrightarrow \text{antisym}(\bigcup R)$
 apply $(\text{auto simp add}:\text{chain-subset-def } \text{antisym-def})$
 apply (metis subsetD)
 done

lemma *chain-subset-Total-Union*:
 assumes $\text{chain}_{\subseteq} R \forall r \in R. \text{Total } r$
 shows $\text{Total}(\bigcup R)$
 proof $(\text{simp add}:\text{total-on-def } \text{Ball-def}, \text{auto del}:\text{disjCI})$
 fix $r s a b$ assume $A: r:R s:R a:\text{Field } r b:\text{Field } s a \neq b$
 from $\langle \text{chain}_{\subseteq} R \rangle \langle r:R \rangle \langle s:R \rangle$ have $r \subseteq s \vee s \subseteq r$
 by $(\text{simp add}:\text{chain-subset-def})$

```

thus  $(\exists r \in R. (a, b) \in r) \vee (\exists r \in R. (b, a) \in r)$ 
proof
  assume  $r \subseteq s$  hence  $(a, b):s \vee (b, a):s$  using assms(2) A
  by(simp add:total-on-def)(metis mono-Field subsetD)
  thus ?thesis using  $\langle s:R \rangle$  by blast
next
  assume  $s \subseteq r$  hence  $(a, b):r \vee (b, a):r$  using assms(2) A
  by(simp add:total-on-def)(metis mono-Field subsetD)
  thus ?thesis using  $\langle r:R \rangle$  by blast
qed
qed

```

```

lemma wf-Union-wf-init-segs:
assumes  $R \in \text{Chain init-seg-of}$  and  $\forall r \in R. \text{wf } r$  shows  $\text{wf}(\bigcup R)$ 
proof(simp add:wf-iff-no-infinite-down-chain, rule ccontr, auto)
  fix  $f$  assume  $1: \forall i. \exists r \in R. (f(\text{Suc } i), f i) \in r$ 
  then obtain  $r$  where  $r:R$  and  $(f(\text{Suc } 0), f 0) : r$  by auto
  { fix  $i$  have  $(f(\text{Suc } i), f i) \in r$ 
    proof(induct i)
      case  $0$  show ?case by fact
    next
      case  $(\text{Suc } i)$ 
      moreover obtain  $s$  where  $s \in R$  and  $(f(\text{Suc}(\text{Suc } i)), f(\text{Suc } i)) \in s$ 
      using  $1$  by auto
      moreover hence  $s$  initial-segment-of  $r \vee r$  initial-segment-of  $s$ 
      using assms(1) \langle r:R \rangle by(simp add:Chain-def)
      ultimately show ?case by(simp add:init-seg-of-def) blast
    qed
  }
  thus False using assms(2) \langle r:R \rangle
  by(simp add:wf-iff-no-infinite-down-chain) blast
qed

```

```

lemma initial-segment-of-Diff:
   $p$  initial-segment-of  $q \implies p - s$  initial-segment-of  $q - s$ 
unfolding init-seg-of-def by blast

```

```

lemma Chain-inits-DiffI:
   $R \in \text{Chain init-seg-of} \implies \{r - s \mid r. r \in R\} \in \text{Chain init-seg-of}$ 
unfolding Chain-def by (blast intro: initial-segment-of-Diff)

```

```

theorem well-ordering:  $\exists r::('a*'a)\text{set}. \text{Well-order } r \wedge \text{Field } r = \text{UNIV}$ 
proof–

```

```

— The initial segment relation on well-orders:
let ?WO =  $\{r::('a*'a)\text{set}. \text{Well-order } r\}$ 
def  $I \equiv \text{init-seg-of} \cap ?WO \times ?WO$ 
have  $I\text{-init}$ :  $I \subseteq \text{init-seg-of}$  by(auto simp:I-def)
hence subch:  $!!R. R : \text{Chain } I \implies \text{chain}_{\subseteq} R$ 
  by(auto simp:init-seg-of-def chain-subset-def Chain-def)

```

have *Chain-wo*: $!!R r. R \in \text{Chain } I \implies r \in R \implies \text{Well-order } r$
by(*simp add:Chain-def I-def*) *blast*
have *FI*: $\text{Field } I = ?\text{WO}$ **by**(*auto simp add:I-def init-seg-of-def Field-def*)
hence *0*: *Partial-order* *I*
by(*auto simp: partial-order-on-def preorder-on-def antisym-def antisym-init-seg-of-refl-on-def trans-def I-def elim!: trans-init-seg-of*)
— I-chains have upper bounds in ?WO wrt I: their Union
{ **fix** *R* **assume** $R \in \text{Chain } I$
hence *Ris*: $R \in \text{Chain init-seg-of}$ **using** *mono-Chain*[*OF I-init*] **by** *blast*
have *subch*: $\text{chain}_{\subseteq} R$ **using** $\langle R : \text{Chain } I \rangle$ *I-init*
by(*auto simp:init-seg-of-def chain-subset-def Chain-def*)
have $\forall r \in R. \text{Refl } r \ \forall r \in R. \text{trans } r \ \forall r \in R. \text{antisym } r \ \forall r \in R. \text{Total } r$
 $\forall r \in R. \text{wf}(r - \text{Id})$
using *Chain-wo*[*OF* $\langle R \in \text{Chain } I \rangle$] **by**(*simp-all add:order-on-defs*)
have $\text{Refl } (\bigcup R)$ **using** $\langle \forall r \in R. \text{Refl } r \rangle$ **by**(*auto simp:refl-on-def*)
moreover **have** $\text{trans } (\bigcup R)$
by(*rule chain-subset-trans-Union*[*OF subch* $\langle \forall r \in R. \text{trans } r \rangle$])
moreover **have** $\text{antisym}(\bigcup R)$
by(*rule chain-subset-antisym-Union*[*OF subch* $\langle \forall r \in R. \text{antisym } r \rangle$])
moreover **have** $\text{Total } (\bigcup R)$
by(*rule chain-subset-Total-Union*[*OF subch* $\langle \forall r \in R. \text{Total } r \rangle$])
moreover **have** $\text{wf}((\bigcup R) - \text{Id})$
proof—
have $(\bigcup R) - \text{Id} = \bigcup \{r - \text{Id} \mid r. r \in R\}$ **by** *blast*
with $\langle \forall r \in R. \text{wf}(r - \text{Id}) \rangle$ *wf-Union-wf-init-segs*[*OF Chain-inits-DiffI*[*OF Ris*]]
show *?thesis* **by** (*simp (no-asm-simp)*) *blast*
qed
ultimately **have** $\text{Well-order } (\bigcup R)$ **by**(*simp add:order-on-defs*)
moreover **have** $\forall r \in R. r \text{ initial-segment-of } \bigcup R$ **using** *Ris*
by(*simp add: Chain-init-seg-of-Union*)
ultimately **have** $\bigcup R : ?\text{WO} \wedge (\forall r \in R. (r, \bigcup R) : I)$
using *mono-Chain*[*OF I-init*] $\langle R \in \text{Chain } I \rangle$
by(*simp (no-asm) add:I-def del:Field-Union*)(*metis Chain-wo subsetD*)
}

hence *1*: $\forall R \in \text{Chain } I. \exists u \in \text{Field } I. \forall r \in R. (r, u) : I$ **by** (*subst FI*) *blast*
— Zorn’s Lemma yields a maximal well-order *m*:
then **obtain** *m*:: $(‘a * ‘a)\text{set}$ **where** $\text{Well-order } m$ **and**
 $\text{max}: \forall r. \text{Well-order } r \wedge (m, r) : I \longrightarrow r = m$
using *Zorns-po-lemma*[*OF 0 1*] **by** (*auto simp:FI*)
— Now show by contradiction that *m* covers the whole type:
{ **fix** *x*:: $‘a$ **assume** $x \notin \text{Field } m$

— We assume that *x* is not covered and extend *m* at the top with *x*
have $m \neq \{\}$
proof
assume $m = \{\}$
moreover **have** $\text{Well-order } \{(x, x)\}$
by(*simp add:order-on-defs refl-on-def trans-def antisym-def total-on-def Field-def Domain-unfold Domain-converse* [*symmetric*])
ultimately **show** *False* **using** *max*

by (auto simp:I-def init-seg-of-def simp del:Field-insert)
 qed
 hence $Field\ m \neq \{\}$ by(auto simp:Field-def)
 moreover have $wf(m-Id)$ using $\langle Well-order\ m \rangle$
 by(simp add:well-order-on-def)
 — The extension of m by x :
 let $?s = \{(a,x) \mid a. a : Field\ m\}$ let $?m = insert\ (x,x)\ m\ Un\ ?s$
 have $Fm: Field\ ?m = insert\ x\ (Field\ m)$
 apply(simp add:Field-insert Field-Un)
 unfolding Field-def by auto
 have $Refl\ m\ trans\ m\ antisym\ m\ Total\ m\ wf(m-Id)$
 using $\langle Well-order\ m \rangle$ by(simp-all add:order-on-defs)
 — We show that the extension is a well-order
 have $Refl\ ?m$ using $\langle Refl\ m \rangle\ Fm$ by(auto simp:refl-on-def)
 moreover have $trans\ ?m$ using $\langle trans\ m \rangle\ \langle x \notin Field\ m \rangle$
 unfolding trans-def Field-def Domain-unfold Domain-converse [symmetric]
 by blast
 moreover have $antisym\ ?m$ using $\langle antisym\ m \rangle\ \langle x \notin Field\ m \rangle$
 unfolding antisym-def Field-def Domain-unfold Domain-converse [symmetric]
 by blast
 moreover have $Total\ ?m$ using $\langle Total\ m \rangle\ Fm$ by(auto simp: total-on-def)
 moreover have $wf(?m-Id)$
 proof—
 have $wf\ ?s$ using $\langle x \notin Field\ m \rangle$
 by(auto simp add:wf-eq-minimal Field-def Domain-unfold Domain-converse
 [symmetric]) metis
 thus $?thesis$ using $\langle wf(m-Id) \rangle\ \langle x \notin Field\ m \rangle$
 wf-subset[OF $\langle wf\ ?s \rangle\ Diff-subset$]
 by (fastforce intro!: wf-Un simp add: Un-Diff Field-def)
 qed
 ultimately have $Well-order\ ?m$ by(simp add:order-on-defs)
 — We show that the extension is above m
 moreover hence $(m, ?m) : I$ using $\langle Well-order\ m \rangle\ \langle x \notin Field\ m \rangle$
 by(fastforce simp:I-def init-seg-of-def Field-def Domain-unfold Domain-converse
 [symmetric])
 ultimately
 — This contradicts maximality of m :
 have $False$ using $max\ \langle x \notin Field\ m \rangle$ unfolding Field-def by blast
 }
 hence $Field\ m = UNIV$ by auto
 moreover with $\langle Well-order\ m \rangle$ have $Well-order\ m$ by simp
 ultimately show $?thesis$ by blast
 qed

corollary well-order-on: $\exists r::('a*'a)set. well-order-on\ A\ r$

proof —

obtain $r::('a*'a)set$ where $wo: Well-order\ r$ and $univ: Field\ r = UNIV$
 using well-ordering[where $'a = 'a$] by blast
 let $?r = \{(x,y). x:A \ \& \ y:A \ \& \ (x,y):r\}$

```

have 1: Field ?r = A using wo univ
by(fastforce simp: Field-def Domain-unfold Domain-converse [symmetric] order-on-defs
refl-on-def)
have Refl r trans r antisym r Total r wf(r-Id)
using ⟨Well-order r⟩ by(simp-all add:order-on-defs)
have Refl ?r using ⟨Refl r⟩ by(auto simp:refl-on-def 1 univ)
moreover have trans ?r using ⟨trans r⟩
unfolding trans-def by blast
moreover have antisym ?r using ⟨antisym r⟩
unfolding antisym-def by blast
moreover have Total ?r using ⟨Total r⟩ by(simp add:total-on-def 1 univ)
moreover have wf(?r - Id) by(rule wf-subset[OF ⟨wf(r-Id)⟩]) blast
ultimately have Well-order ?r by(simp add:order-on-defs)
with 1 show ?thesis by metis
qed

end

```

3 Infinite-Set: Infinite Sets and Related Concepts

```

theory Infinite-Set
imports Main
begin

```

3.1 Infinite Sets

Some elementary facts about infinite sets, mostly by Stefan Merz. Beware! Because “infinite” merely abbreviates a negation, these lemmas may not work well with *blast*.

abbreviation

```

infinite :: 'a set ⇒ bool where
infinite S == ¬ finite S

```

Infinite sets are non-empty, and if we remove some elements from an infinite set, the result is still infinite.

```

lemma infinite-imp-nonempty: infinite S ==> S ≠ {}
by auto

```

lemma infinite-remove:

```

infinite S ⇒ infinite (S - {a})
by simp

```

lemma Diff-infinite-finite:

```

assumes T: finite T and S: infinite S
shows infinite (S - T)
using T
proof induct

```

```

from  $S$ 
show  $\text{infinite } (S - \{\})$  by auto
next
fix  $T x$ 
assume  $ih: \text{infinite } (S - T)$ 
have  $S - (\text{insert } x T) = (S - T) - \{x\}$ 
by (rule Diff-insert)
with  $ih$ 
show  $\text{infinite } (S - (\text{insert } x T))$ 
by (simp add: infinite-remove)
qed

```

```

lemma Un-infinite:  $\text{infinite } S \implies \text{infinite } (S \cup T)$ 
by simp

```

```

lemma infinite-Un:  $\text{infinite } (S \cup T) \longleftrightarrow \text{infinite } S \vee \text{infinite } T$ 
by simp

```

```

lemma infinite-super:
assumes  $T: S \subseteq T$  and  $S: \text{infinite } S$ 
shows  $\text{infinite } T$ 
proof
assume  $\text{finite } T$ 
with  $T$  have  $\text{finite } S$  by (simp add: finite-subset)
with  $S$  show False by simp
qed

```

As a concrete example, we prove that the set of natural numbers is infinite.

```

lemma finite-nat-bounded:
assumes  $S: \text{finite } (S::\text{nat set})$ 
shows  $\exists k. S \subseteq \{..<k\}$  (is  $\exists k. ?\text{bounded } S k$ )
using  $S$ 
proof induct
have  $?\text{bounded } \{\} 0$  by simp
then show  $\exists k. ?\text{bounded } \{\} k$  ..
next
fix  $S x$ 
assume  $\exists k. ?\text{bounded } S k$ 
then obtain  $k$  where  $k: ?\text{bounded } S k$  ..
show  $\exists k. ?\text{bounded } (\text{insert } x S) k$ 
proof (cases  $x < k$ )
case True
with  $k$  show ?thesis by auto
next
case False
with  $k$  have  $?\text{bounded } S (\text{Suc } x)$  by auto
then show ?thesis by auto
qed
qed

```

lemma *finite-nat-iff-bounded*:
 $finite (S::nat\ set) = (\exists k. S \subseteq \{..<k\})$ (is ?lhs = ?rhs)
proof
 assume ?lhs
 then show ?rhs by (rule *finite-nat-bounded*)
next
 assume ?rhs
 then obtain k where $S \subseteq \{..<k\}$..
 then show *finite* S
 by (rule *finite-subset*) simp
qed

lemma *finite-nat-iff-bounded-le*:
 $finite (S::nat\ set) = (\exists k. S \subseteq \{..k\})$ (is ?lhs = ?rhs)
proof
 assume ?lhs
 then obtain k where $S \subseteq \{..<k\}$
 by (*blast dest: finite-nat-bounded*)
 then have $S \subseteq \{..k\}$ by *auto*
 then show ?rhs ..
next
 assume ?rhs
 then obtain k where $S \subseteq \{..k\}$..
 then show *finite* S
 by (rule *finite-subset*) simp
qed

lemma *infinite-nat-iff-unbounded*:
 $infinite (S::nat\ set) = (\forall m. \exists n. m < n \wedge n \in S)$
 (is ?lhs = ?rhs)
proof
 assume ?lhs
 show ?rhs
proof (rule *ccontr*)
 assume $\neg ?rhs$
 then obtain m where $m: \forall n. m < n \longrightarrow n \notin S$ by *blast*
 then have $S \subseteq \{..m\}$
 by (*auto simp add: sym [OF linorder-not-less]*)
 with $\langle ?lhs \rangle$ show *False*
 by (*simp add: finite-nat-iff-bounded-le*)
qed
next
 assume ?rhs
 show ?lhs
proof
 assume *finite* S
 then obtain m where $S \subseteq \{..m\}$
 by (*auto simp add: finite-nat-iff-bounded-le*)

```

    then have  $\forall n. m < n \longrightarrow n \notin S$  by auto
    with  $\langle ?rhs \rangle$  show False by blast
  qed
qed

```

```

lemma infinite-nat-iff-unbounded-le:
  infinite (S::nat set) = ( $\forall m. \exists n. m \leq n \wedge n \in S$ )
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  show ?rhs
  proof
    fix m
    from  $\langle ?lhs \rangle$  obtain n where  $m < n \wedge n \in S$ 
    by (auto simp add: infinite-nat-iff-unbounded)
    then have  $m \leq n \wedge n \in S$  by simp
    then show  $\exists n. m \leq n \wedge n \in S$  ..
  qed
next
  assume ?rhs
  show ?lhs
  proof (auto simp add: infinite-nat-iff-unbounded)
    fix m
    from  $\langle ?rhs \rangle$  obtain n where  $Suc\ m \leq n \wedge n \in S$ 
    by blast
    then have  $m < n \wedge n \in S$  by simp
    then show  $\exists n. m < n \wedge n \in S$  ..
  qed
qed

```

For a set of natural numbers to be infinite, it is enough to know that for any number larger than some k , there is some larger number that is an element of the set.

```

lemma unbounded-k-infinite:
  assumes  $k: \forall m. k < m \longrightarrow (\exists n. m < n \wedge n \in S)$ 
  shows infinite (S::nat set)
proof -
  {
    fix m have  $\exists n. m < n \wedge n \in S$ 
    proof (cases  $k < m$ )
      case True
        with k show ?thesis by blast
    next
      case False
        from k obtain n where  $Suc\ k < n \wedge n \in S$  by auto
        with False have  $m < n \wedge n \in S$  by auto
        then show ?thesis ..
    qed
  }

```

```

then show ?thesis
  by (auto simp add: infinite-nat-iff-unbounded)
qed

```

```

lemma nat-infinite: infinite (UNIV :: nat set)
  by (auto simp add: infinite-nat-iff-unbounded)

```

```

lemma nat-not-finite: finite (UNIV :: nat set)  $\implies$  R
  by simp

```

Every infinite set contains a countable subset. More precisely we show that a set S is infinite if and only if there exists an injective function from the naturals into S .

```

lemma range-inj-infinite:
  inj (f :: nat  $\Rightarrow$  'a)  $\implies$  infinite (range f)

```

```

proof
  assume finite (range f) and inj f
  then have finite (UNIV :: nat set)
    by (rule finite-imageD)
  then show False by simp
qed

```

```

lemma int-infinite [simp]:
  shows infinite (UNIV :: int set)

```

```

proof –
  from inj-int have infinite (range int) by (rule range-inj-infinite)
  moreover
  have range int  $\subseteq$  (UNIV :: int set) by simp
  ultimately show infinite (UNIV :: int set) by (simp add: infinite-super)
qed

```

The “only if” direction is harder because it requires the construction of a sequence of pairwise different elements of an infinite set S . The idea is to construct a sequence of non-empty and infinite subsets of S obtained by successively removing elements of S .

```

lemma linorder-injI:
  assumes hyp:  $\forall x y. x < (y :: 'a :: linorder) \implies f x \neq f y$ 
  shows inj f

```

```

proof (rule inj-onI)
  fix x y
  assume f-eq:  $f x = f y$ 
  show  $x = y$ 
  proof (rule linorder-cases)
    assume  $x < y$ 
    with hyp have  $f x \neq f y$  by blast
    with f-eq show ?thesis by simp
  next

```

```

    assume  $x = y$ 
    then show ?thesis .
next
    assume  $y < x$ 
    with hyp have  $f y \neq f x$  by blast
    with f-eq show ?thesis by simp
qed
qed

lemma infinite-countable-subset:
  assumes inf: infinite (S::'a set)
  shows  $\exists f. inj (f::nat \Rightarrow 'a) \wedge range f \subseteq S$ 
proof -
  def Sseq  $\equiv nat-rec S (\lambda n T. T - \{SOME e. e \in T\})$ 
  def pick  $\equiv \lambda n. (SOME e. e \in Sseq n)$ 
  have Sseq-inf:  $\bigwedge n. infinite (Sseq n)$ 
  proof -
    fix n
    show infinite (Sseq n)
    proof (induct n)
      from inf show infinite (Sseq 0)
      by (simp add: Sseq-def)
    next
      fix n
      assume infinite (Sseq n) then show infinite (Sseq (Suc n))
      by (simp add: Sseq-def infinite-remove)
    qed
  qed
  have Sseq-S:  $\bigwedge n. Sseq n \subseteq S$ 
  proof -
    fix n
    show  $Sseq n \subseteq S$ 
    by (induct n) (auto simp add: Sseq-def)
  qed
  have Sseq-pick:  $\bigwedge n. pick n \in Sseq n$ 
  proof -
    fix n
    show  $pick n \in Sseq n$ 
    proof (unfold pick-def, rule someI-ex)
      from Sseq-inf have infinite (Sseq n) .
      then have  $Sseq n \neq \{\}$  by auto
      then show  $\exists x. x \in Sseq n$  by auto
    qed
  qed
  with Sseq-S have rng:  $range pick \subseteq S$ 
  by auto
  have pick-Sseq-gt:  $\bigwedge n m. pick n \notin Sseq (n + Suc m)$ 
  proof -
    fix n m

```

```

  show pick n ∉ Sseq (n + Suc m)
  by (induct m) (auto simp add: Sseq-def pick-def)
qed
have pick-pick:  $\bigwedge n m. \text{pick } n \neq \text{pick } (n + \text{Suc } m)$ 
proof -
  fix n m
  from Sseq-pick have pick (n + Suc m) ∈ Sseq (n + Suc m) .
  moreover from pick-Sseq-gt
  have pick n ∉ Sseq (n + Suc m) .
  ultimately show pick n ≠ pick (n + Suc m)
  by auto
qed
have inj: inj pick
proof (rule linorder-injI)
  fix i j :: nat
  assume i < j
  show pick i ≠ pick j
  proof
    assume eq: pick i = pick j
    from (i < j) obtain k where j = i + Suc k
    by (auto simp add: less-iff-Suc-add)
    with pick-pick have pick i ≠ pick j by simp
    with eq show False by simp
  qed
qed
from rng inj show ?thesis by auto
qed

```

lemma *infinite-iff-countable-subset*:

infinite $S = (\exists f. \text{inj } (f :: \text{nat} \Rightarrow 'a) \wedge \text{range } f \subseteq S)$

by (auto simp add: infinite-countable-subset range-inj-infinite infinite-super)

For any function with infinite domain and finite range there is some element that is the image of infinitely many domain elements. In particular, any infinite sequence of elements from a finite set contains some element that occurs infinitely often.

lemma *inf-img-fin-dom*:

assumes *img*: finite ($f'A$) and *dom*: infinite A

shows $\exists y \in f'A. \text{infinite } (f -' \{y\})$

proof (rule ccontr)

assume $\neg ?thesis$

with *img* have finite $(\bigcup y:f'A. f -' \{y\})$ by blast

moreover have $A \subseteq (\bigcup y:f'A. f -' \{y\})$ by auto

moreover note *dom*

ultimately show False by (simp add: infinite-super)

qed

lemma *inf-img-fin-domE*:

assumes finite ($f'A$) and infinite A

obtains y **where** $y \in f^c A$ **and** *infinite* $(f -^c \{y\})$
using *assms* **by** (*blast dest: inf-img-fin-dom*)

3.2 Infinitely Many and Almost All

We often need to reason about the existence of infinitely many (resp., all but finitely many) objects satisfying some predicate, so we introduce corresponding binders and their proof rules.

definition

Inf-many $:: ('a \Rightarrow bool) \Rightarrow bool$ (**binder** *INFM* 10) **where**
Inf-many $P = \text{infinite } \{x. P x\}$

definition

Alm-all $:: ('a \Rightarrow bool) \Rightarrow bool$ (**binder** *MOST* 10) **where**
Alm-all $P = (\neg (\text{INFM } x. \neg P x))$

notation (*xsymbols*)

Inf-many (**binder** \exists_∞ 10) **and**
Alm-all (**binder** \forall_∞ 10)

notation (*HTML output*)

Inf-many (**binder** \exists_∞ 10) **and**
Alm-all (**binder** \forall_∞ 10)

lemma *INFM-iff-infinite*: $(\text{INFM } x. P x) \longleftrightarrow \text{infinite } \{x. P x\}$
unfolding *Inf-many-def* ..

lemma *MOST-iff-cofinite*: $(\text{MOST } x. P x) \longleftrightarrow \text{finite } \{x. \neg P x\}$
unfolding *Alm-all-def Inf-many-def* **by** *simp*

lemmas *MOST-iff-finiteNeg = MOST-iff-cofinite*

lemma *not-INFM* [*simp*]: $\neg (\text{INFM } x. P x) \longleftrightarrow (\text{MOST } x. \neg P x)$
unfolding *Alm-all-def not-not* ..

lemma *not-MOST* [*simp*]: $\neg (\text{MOST } x. P x) \longleftrightarrow (\text{INFM } x. \neg P x)$
unfolding *Alm-all-def not-not* ..

lemma *INFM-const* [*simp*]: $(\text{INFM } x::'a. P) \longleftrightarrow P \wedge \text{infinite } (\text{UNIV}::'a \text{ set})$
unfolding *Inf-many-def* **by** *simp*

lemma *MOST-const* [*simp*]: $(\text{MOST } x::'a. P) \longleftrightarrow P \vee \text{finite } (\text{UNIV}::'a \text{ set})$
unfolding *Alm-all-def* **by** *simp*

lemma *INFM-EX*: $(\exists_\infty x. P x) \Longrightarrow (\exists x. P x)$
by (*erule contrapos-pp, simp*)

lemma *ALL-MOST*: $\forall x. P x \Longrightarrow \forall_\infty x. P x$

by *simp*

lemma *INFM-E*: **assumes** *INFM* x . P x **obtains** x **where** P x
using *INFM-EX* [*OF assms*] **by** (*rule exE*)

lemma *MOST-I*: **assumes** $\bigwedge x$. P x **shows** *MOST* x . P x
using *assms* **by** *simp*

lemma *INFM-mono*:

assumes *inf*: $\exists_{\infty} x$. P x **and** q : $\bigwedge x$. P $x \implies Q$ x
shows $\exists_{\infty} x$. Q x

proof –

from *inf* **have** *infinite* $\{x$. P $x\}$ **unfolding** *Inf-many-def* .
moreover from q **have** $\{x$. P $x\} \subseteq \{x$. Q $x\}$ **by** *auto*
ultimately show *?thesis*
by (*simp add: Inf-many-def infinite-super*)

qed

lemma *MOST-mono*: $\forall_{\infty} x$. P $x \implies (\bigwedge x$. P $x \implies Q$ $x) \implies \forall_{\infty} x$. Q x
unfolding *Alm-all-def* **by** (*blast intro: INFM-mono*)

lemma *INFM-disj-distrib*:

$(\exists_{\infty} x$. P $x \vee Q$ $x) \longleftrightarrow (\exists_{\infty} x$. P $x) \vee (\exists_{\infty} x$. Q $x)$
unfolding *Inf-many-def* **by** (*simp add: Collect-disj-eq*)

lemma *INFM-imp-distrib*:

$(\text{INFM } x$. P $x \longrightarrow Q$ $x) \longleftrightarrow ((\text{MOST } x$. P $x) \longrightarrow (\text{INFM } x$. Q $x))$
by (*simp only: imp-conv-disj INFM-disj-distrib not-MOST*)

lemma *MOST-conj-distrib*:

$(\forall_{\infty} x$. P $x \wedge Q$ $x) \longleftrightarrow (\forall_{\infty} x$. P $x) \wedge (\forall_{\infty} x$. Q $x)$
unfolding *Alm-all-def* **by** (*simp add: INFM-disj-distrib del: disj-not1*)

lemma *MOST-conjI*:

$\text{MOST } x$. P $x \implies \text{MOST } x$. Q $x \implies \text{MOST } x$. P $x \wedge Q$ x
by (*simp add: MOST-conj-distrib*)

lemma *INFM-conjI*:

$\text{INFM } x$. P $x \implies \text{MOST } x$. Q $x \implies \text{INFM } x$. P $x \wedge Q$ x
unfolding *MOST-iff-cofinite INFM-iff-infinite*
apply (*drule (1) Diff-infinite-finite*)
apply (*simp add: Collect-conj-eq Collect-neg-eq*)
done

lemma *MOST-rev-mp*:

assumes $\forall_{\infty} x$. P x **and** $\forall_{\infty} x$. P $x \longrightarrow Q$ x
shows $\forall_{\infty} x$. Q x

proof –

have $\forall_{\infty} x$. P $x \wedge (P$ $x \longrightarrow Q$ $x)$

using *assms* **by** (rule *MOST-conjI*)
thus *?thesis* **by** (rule *MOST-mono*) *simp*
qed

lemma *MOST-imp-iff*:
assumes $MOST\ x. P\ x$
shows $(MOST\ x. P\ x \longrightarrow Q\ x) \longleftrightarrow (MOST\ x. Q\ x)$
proof
assume $MOST\ x. P\ x \longrightarrow Q\ x$
with *assms* **show** $MOST\ x. Q\ x$ **by** (rule *MOST-rev-mp*)
next
assume $MOST\ x. Q\ x$
then show $MOST\ x. P\ x \longrightarrow Q\ x$ **by** (rule *MOST-mono*) *simp*
qed

lemma *INFM-MOST-simps* [*simp*]:
 $\bigwedge P\ Q. (INFM\ x. P\ x \wedge Q) \longleftrightarrow (INFM\ x. P\ x) \wedge Q$
 $\bigwedge P\ Q. (INFM\ x. P \wedge Q\ x) \longleftrightarrow P \wedge (INFM\ x. Q\ x)$
 $\bigwedge P\ Q. (MOST\ x. P\ x \vee Q) \longleftrightarrow (MOST\ x. P\ x) \vee Q$
 $\bigwedge P\ Q. (MOST\ x. P \vee Q\ x) \longleftrightarrow P \vee (MOST\ x. Q\ x)$
 $\bigwedge P\ Q. (MOST\ x. P\ x \longrightarrow Q) \longleftrightarrow ((INFM\ x. P\ x) \longrightarrow Q)$
 $\bigwedge P\ Q. (MOST\ x. P \longrightarrow Q\ x) \longleftrightarrow (P \longrightarrow (MOST\ x. Q\ x))$
unfolding *Alm-all-def Inf-many-def*
by (*simp-all add: Collect-conj-eq*)

Properties of quantifiers with injective functions.

lemma *INFM-inj*:
 $INFM\ x. P\ (f\ x) \Longrightarrow inj\ f \Longrightarrow INFM\ x. P\ x$
unfolding *INFM-iff-infinite*
by (*clarify, drule (1) finite-vimageI, simp*)

lemma *MOST-inj*:
 $MOST\ x. P\ x \Longrightarrow inj\ f \Longrightarrow MOST\ x. P\ (f\ x)$
unfolding *MOST-iff-cofinite*
by (*drule (1) finite-vimageI, simp*)

Properties of quantifiers with singletons.

lemma *not-INFM-eq* [*simp*]:
 $\neg (INFM\ x. x = a)$
 $\neg (INFM\ x. a = x)$
unfolding *INFM-iff-infinite* **by** *simp-all*

lemma *MOST-neq* [*simp*]:
 $MOST\ x. x \neq a$
 $MOST\ x. a \neq x$
unfolding *MOST-iff-cofinite* **by** *simp-all*

lemma *INFM-neq* [*simp*]:
 $(INFM\ x::'a. x \neq a) \longleftrightarrow infinite\ (UNIV::'a\ set)$

(*INFM* $x::'a. a \neq x$) \longleftrightarrow *infinite* (*UNIV*::'a set)
unfolding *INFM-iff-infinite* **by** *simp-all*

lemma *MOST-eq* [*simp*]:
(*MOST* $x::'a. x = a$) \longleftrightarrow *finite* (*UNIV*::'a set)
(*MOST* $x::'a. a = x$) \longleftrightarrow *finite* (*UNIV*::'a set)
unfolding *MOST-iff-cofinite* **by** *simp-all*

lemma *MOST-eq-imp*:
MOST $x. x = a \longrightarrow P x$
MOST $x. a = x \longrightarrow P x$
unfolding *MOST-iff-cofinite* **by** *simp-all*

Properties of quantifiers over the naturals.

lemma *INFM-nat*: ($\exists_{\infty} n. P (n::nat)$) = ($\forall m. \exists n. m < n \wedge P n$)
by (*simp add: Inf-many-def infinite-nat-iff-unbounded*)

lemma *INFM-nat-le*: ($\exists_{\infty} n. P (n::nat)$) = ($\forall m. \exists n. m \leq n \wedge P n$)
by (*simp add: Inf-many-def infinite-nat-iff-unbounded-le*)

lemma *MOST-nat*: ($\forall_{\infty} n. P (n::nat)$) = ($\exists m. \forall n. m < n \longrightarrow P n$)
by (*simp add: Alm-all-def INFM-nat*)

lemma *MOST-nat-le*: ($\forall_{\infty} n. P (n::nat)$) = ($\exists m. \forall n. m \leq n \longrightarrow P n$)
by (*simp add: Alm-all-def INFM-nat-le*)

3.3 Enumeration of an Infinite Set

The set’s element type must be wellordered (e.g. the natural numbers).

primrec (*in wellorder*) *enumerate* :: 'a set \Rightarrow nat \Rightarrow 'a **where**
enumerate-0: *enumerate* S 0 = (*LEAST* $n. n \in S$)
| *enumerate-Suc*: *enumerate* S (*Suc* n) = *enumerate* ($S - \{\text{LEAST } n. n \in S\}$)
 n

lemma *enumerate-Suc'*:
enumerate S (*Suc* n) = *enumerate* ($S - \{\text{enumerate } S\ 0\}$) n
by *simp*

lemma *enumerate-in-set*: *infinite* $S \Longrightarrow$ *enumerate* S $n : S$
apply (*induct n arbitrary: S*)
apply (*fastforce intro: LeastI dest!: infinite-imp-nonempty*)
apply *simp*
apply (*metis DiffE infinite-remove*)
done

declare *enumerate-0* [*simp del*] *enumerate-Suc* [*simp del*]

lemma *enumerate-step*: *infinite* $S \Longrightarrow$ *enumerate* S $n <$ *enumerate* S (*Suc* n)
apply (*induct n arbitrary: S*)

```

apply (rule order-le-neq-trans)
apply (simp add: enumerate-0 Least-le enumerate-in-set)
apply (simp only: enumerate-Suc')
apply (subgoal-tac enumerate (S - {enumerate S 0}) 0 : S - {enumerate S
0})
apply (blast intro: sym)
apply (simp add: enumerate-in-set del: Diff-iff)
apply (simp add: enumerate-Suc')
done

```

```

lemma enumerate-mono:  $m < n \implies \text{infinite } S \implies \text{enumerate } S \ m < \text{enumerate } S \ n$ 
apply (erule less-Suc-induct)
apply (auto intro: enumerate-step)
done

```

3.4 Miscellaneous

A few trivial lemmas about sets that contain at most one element. These simplify the reasoning about deterministic automata.

definition

```

atmost-one :: 'a set  $\Rightarrow$  bool where
atmost-one S = ( $\forall x \ y. x \in S \wedge y \in S \longrightarrow x = y$ )

```

```

lemma atmost-one-empty:  $S = \{\} \implies \text{atmost-one } S$ 
by (simp add: atmost-one-def)

```

```

lemma atmost-one-singleton:  $S = \{x\} \implies \text{atmost-one } S$ 
by (simp add: atmost-one-def)

```

```

lemma atmost-one-unique [elim]:  $\text{atmost-one } S \implies x \in S \implies y \in S \implies y = x$ 
by (simp add: atmost-one-def)

```

end

4 Filter: Filters and Ultrafilters

theory Filter

```

imports ~/src/HOL/Library/Zorn ~/src/HOL/Library/Infinite-Set
begin

```

4.1 Definitions and basic properties

4.1.1 Filters

```

locale filter =

```

```

fixes  $F :: 'a \text{ set set}$ 
assumes  $UNIV$  [iff]:  $UNIV \in F$ 
assumes  $empty$  [iff]:  $\{\} \notin F$ 
assumes  $Int$ :  $\llbracket u \in F; v \in F \rrbracket \implies u \cap v \in F$ 
assumes  $subset$ :  $\llbracket u \in F; u \subseteq v \rrbracket \implies v \in F$ 
begin

```

```

lemma  $memD$ :  $A \in F \implies - A \notin F$ 

```

```

proof

```

```

  assume  $A \in F$  and  $- A \in F$ 
  hence  $A \cap (- A) \in F$  by (rule  $Int$ )
  thus  $False$  by  $simp$ 

```

```

qed

```

```

lemma  $not-memI$ :  $- A \in F \implies A \notin F$ 
by (drule  $memD$ ,  $simp$ )

```

```

lemma  $Int-iff$ :  $(x \cap y \in F) = (x \in F \wedge y \in F)$ 
by (auto elim:  $subset$  intro:  $Int$ )

```

```

end

```

4.1.2 Ultrafilters

```

locale  $ultrafilter = filter +$ 
  assumes  $ultra$ :  $A \in F \vee - A \in F$ 
begin

```

```

lemma  $memI$ :  $- A \notin F \implies A \in F$ 
using  $ultra$  [of  $A$ ] by  $simp$ 

```

```

lemma  $not-memD$ :  $A \notin F \implies - A \in F$ 
by (rule  $memI$ ,  $simp$ )

```

```

lemma  $not-mem-iff$ :  $(A \notin F) = (- A \in F)$ 
by (rule  $iffI$  [ $OF$   $not-memD$   $not-memI$ ])

```

```

lemma  $Compl-iff$ :  $(- A \in F) = (A \notin F)$ 
by (rule  $iffI$  [ $OF$   $not-memI$   $not-memD$ ])

```

```

lemma  $Un-iff$ :  $(x \cup y \in F) = (x \in F \vee y \in F)$ 
apply (rule  $iffI$ )
  apply (erule  $contrapos-pp$ )
  apply ( $simp$  add:  $Int-iff$   $not-mem-iff$ )
  apply (auto elim:  $subset$ )
done

```

```

end

```

4.1.3 Free Ultrafilters

locale *freeultrafilter* = *ultrafilter* +
assumes *infinite*: $A \in F \implies \text{infinite } A$
begin

lemma *finite*: $\text{finite } A \implies A \notin F$
by (*erule contrapos-pn*, *erule infinite*)

lemma *singleton*: $\{x\} \notin F$
by (*rule finite*, *simp*)

lemma *insert-iff* [*simp*]: $(\text{insert } x \ A \in F) = (A \in F)$
apply (*subst insert-is-Un*)
apply (*subst Un-iff*)
apply (*simp add: singleton*)
done

lemma *filter*: *filter* F ..

lemma *ultrafilter*: *ultrafilter* F ..

end

4.2 Collect properties

lemma (**in** *filter*) *Collect-ex*:
 $(\{n. \exists x. P \ n \ x\} \in F) = (\exists X. \{n. P \ n \ (X \ n)\} \in F)$
proof
assume $\{n. \exists x. P \ n \ x\} \in F$
hence $\{n. P \ n \ (\text{SOME } x. P \ n \ x)\} \in F$
by (*auto elim: someI subset*)
thus $\exists X. \{n. P \ n \ (X \ n)\} \in F$ **by** *fast*
next
show $\exists X. \{n. P \ n \ (X \ n)\} \in F \implies \{n. \exists x. P \ n \ x\} \in F$
by (*auto elim: subset*)
qed

lemma (**in** *filter*) *Collect-conj*:
 $(\{n. P \ n \ \wedge \ Q \ n\} \in F) = (\{n. P \ n\} \in F \ \wedge \ \{n. Q \ n\} \in F)$
by (*subst Collect-conj-eq*, *rule Int-iff*)

lemma (**in** *ultrafilter*) *Collect-not*:
 $(\{n. \neg P \ n\} \in F) = (\{n. P \ n\} \notin F)$
by (*subst Collect-neg-eq*, *rule Compl-iff*)

lemma (**in** *ultrafilter*) *Collect-disj*:
 $(\{n. P \ n \ \vee \ Q \ n\} \in F) = (\{n. P \ n\} \in F \ \vee \ \{n. Q \ n\} \in F)$
by (*subst Collect-disj-eq*, *rule Un-iff*)

```

lemma (in ultrafilter) Collect-all:
  ( $\{n. \forall x. P n x\} \in F$ ) = ( $\forall X. \{n. P n (X n)\} \in F$ )
apply (rule Not-eq-iff [THEN iffD1])
apply (simp add: Collect-not [symmetric])
apply (rule Collect-ex)
done

```

4.3 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 $F = G$

Lemmas 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-lemma1:  $UNIV \in F \implies A \in \{X. \exists f \in F. A \cap f \subseteq X\}$ 
by blast

```

```

lemma extend-lemma2:  $F \subseteq \{X. \exists f \in F. A \cap f \subseteq X\}$ 
by blast

```

```

lemma (in filter) extend-filter:
assumes  $A: - A \notin F$ 
shows filter  $\{X. \exists f \in F. A \cap f \subseteq X\}$  (is filter ?X)
proof (rule filter.intro)
  show  $UNIV \in ?X$  by blast
next
  show  $\{\} \notin ?X$ 
  proof (clarify)
    fix  $f$  assume  $f: f \in F$  and  $Af: A \cap f \subseteq \{\}$ 
    from  $Af$  have  $fA: f \subseteq - A$  by blast
    from  $fA$  have  $- A \in F$  by (rule subset)
    with  $A$  show  $False$  by simp
  qed
next
  fix  $u$  and  $v$ 
  assume  $u: u \in ?X$  and  $v: v \in ?X$ 
  from  $u$  obtain  $f$  where  $f: f \in F$  and  $Af: A \cap f \subseteq u$  by blast
  from  $v$  obtain  $g$  where  $g: g \in F$  and  $Ag: A \cap g \subseteq v$  by blast
  from  $f g$  have  $fg: f \cap g \in F$  by (rule Int)
  from  $Af Ag$  have  $Afg: A \cap (f \cap g) \subseteq u \cap v$  by blast
  from  $fg Afg$  show  $u \cap v \in ?X$  by blast
next
  fix  $u$  and  $v$ 
  assume  $uv: u \subseteq v$  and  $u: u \in ?X$ 
  from  $u$  obtain  $f$  where  $f: f \in F$  and  $Afu: A \cap f \subseteq u$  by blast
  from  $Afu uv$  have  $Afv: A \cap f \subseteq v$  by blast
  from  $f Afv$  have  $\exists f \in F. A \cap f \subseteq v$  by blast
  thus  $v \in ?X$  by simp

```

qed

lemma (in filter) max-filter-ultrafilter:
assumes max: $\bigwedge G. \llbracket \text{filter } G; F \subseteq G \rrbracket \implies F = G$
shows ultrafilter-axioms F
proof (rule ultrafilter-axioms.intro)
 fix A **show** $A \in F \vee \neg A \in F$
proof (rule disjCI)
 let ?X = $\{X. \exists f \in F. A \cap f \subseteq X\}$
assume AF: $\neg A \notin F$
from AF **have** X: filter ?X **by** (rule extend-filter)
from UNIV **have** AX: $A \in ?X$ **by** (rule extend-lemma1)
have FX: $F \subseteq ?X$ **by** (rule extend-lemma2)
from X FX **have** $F = ?X$ **by** (rule max)
with AX **show** $A \in F$ **by** simp
 qed
 qed

lemma (in ultrafilter) max-filter:
assumes G: filter G **and** sub: $F \subseteq G$ **shows** $F = G$
proof
show $F \subseteq G$ **using** sub .
show $G \subseteq F$
proof
 fix A **assume** A: $A \in G$
from G A **have** $\neg A \notin G$ **by** (rule filter.memD)
with sub **have** B: $\neg A \notin F$ **by** blast
thus $A \in F$ **by** (rule memI)
 qed
 qed

4.4 Ultrafilter Theorem

A local context makes proof of ultrafilter Theorem more modular

context

fixes frechet :: 'a set set
and superfrechet :: 'a set set set

assumes infinite-UNIV: infinite (UNIV :: 'a set)

defines frechet-def: $\text{frechet} \equiv \{A. \text{finite } (\neg A)\}$
and superfrechet-def: $\text{superfrechet} \equiv \{G. \text{filter } G \wedge \text{frechet} \subseteq G\}$

begin

lemma superfrechetI:

$\llbracket \text{filter } G; \text{frechet} \subseteq G \rrbracket \implies G \in \text{superfrechet}$
by (simp add: superfrechet-def)

lemma superfrechetD1:

$G \in \text{superfrechet} \implies \text{filter } G$
by (*simp add: superfrechet-def*)

lemma *superfrechetD2*:

$G \in \text{superfrechet} \implies \text{frechet} \subseteq G$
by (*simp add: superfrechet-def*)

A few properties of free filters

lemma *filter-cofinite*:

assumes *inf*: *infinite* (*UNIV* :: 'a set)
shows *filter* {*A*:: 'a set. *finite* ($- A$)} (**is** *filter* ?*F*)

proof (*rule filter.intro*)

show *UNIV* \in ?*F* **by** *simp*

next

show {} \notin ?*F* **using** *inf* **by** *simp*

next

fix *u v* **assume** *u* \in ?*F* **and** *v* \in ?*F*

thus $u \cap v \in$?*F* **by** *simp*

next

fix *u v* **assume** *uv*: $u \subseteq v$ **and** *u*: $u \in$?*F*

from *uv* **have** *vu*: $- v \subseteq - u$ **by** *simp*

from *u* **show** $v \in$?*F*

by (*simp add: finite-subset* [*OF vu*])

qed

We prove: 1. Existence of maximal filter i.e. ultrafilter; 2. Freeness property i.e ultrafilter is free. Use a locale to prove various lemmas and then export main result: The ultrafilter Theorem

lemma *filter-frechet*: *filter frechet*

by (*unfold frechet-def, rule filter-cofinite* [*OF infinite-UNIV*])

lemma *frechet-in-superfrechet*: *frechet* \in *superfrechet*

by (*rule superfrechetI* [*OF filter-frechet subset-refl*])

lemma *lemma-mem-chain-filter*:

$\llbracket c \in \text{chain superfrechet}; x \in c \rrbracket \implies \text{filter } x$

by (*unfold chain-def superfrechet-def, blast*)

4.4.1 Unions of chains of superfrechets

In this section we prove that superfrechet is closed with respect to unions of non-empty chains. We must show 1) Union of a chain is a filter, 2) Union of a chain contains frechet.

Number 2 is trivial, but 1 requires us to prove all the filter rules.

lemma *Union-chain-UNIV*:

$\llbracket c \in \text{chain superfrechet}; c \neq \{\} \rrbracket \implies \text{UNIV} \in \bigcup c$

proof –

assume 1: $c \in \text{chain superfrechet}$ **and** 2: $c \neq \{\}$
from 2 **obtain** x **where** 3: $x \in c$ **by** *blast*
from 1 3 **have** *filter* x **by** (*rule lemma-mem-chain-filter*)
hence $UNIV \in x$ **by** (*rule filter.UNIV*)
with 3 **show** $UNIV \in \bigcup c$ **by** *blast*
qed

lemma *Union-chain-empty*:

$c \in \text{chain superfrechet} \implies \{\} \notin \bigcup c$

proof

assume 1: $c \in \text{chain superfrechet}$ **and** 2: $\{\} \in \bigcup c$
from 2 **obtain** x **where** 3: $x \in c$ **and** 4: $\{\} \in x$..
from 1 3 **have** *filter* x **by** (*rule lemma-mem-chain-filter*)
hence $\{\} \notin x$ **by** (*rule filter.empty*)
with 4 **show** *False* **by** *simp*
qed

lemma *Union-chain-Int*:

$\llbracket c \in \text{chain superfrechet}; u \in \bigcup c; v \in \bigcup c \rrbracket \implies u \cap v \in \bigcup c$

proof –

assume $c: c \in \text{chain superfrechet}$
assume $u \in \bigcup c$
then obtain x **where** $ux: u \in x$ **and** $xc: x \in c$..
assume $v \in \bigcup c$
then obtain y **where** $vy: v \in y$ **and** $yc: y \in c$..
from $c xc yc$ **have** $x \subseteq y \vee y \subseteq x$ **by** (*rule chainD*)
with $xc yc$ **have** $xyz: x \cup y \in c$
by (*auto simp add: Un-absorb1 Un-absorb2*)
with c **have** $fx: \text{filter } (x \cup y)$ **by** (*rule lemma-mem-chain-filter*)
from ux **have** $uxy: u \in x \cup y$ **by** *simp*
from vy **have** $vxy: v \in x \cup y$ **by** *simp*
from $fxy uxy vxy$ **have** $u \cap v \in x \cup y$ **by** (*rule filter.Int*)
with xyz **show** $u \cap v \in \bigcup c$..
qed

lemma *Union-chain-subset*:

$\llbracket c \in \text{chain superfrechet}; u \in \bigcup c; u \subseteq v \rrbracket \implies v \in \bigcup c$

proof –

assume $c: c \in \text{chain superfrechet}$
and $u: u \in \bigcup c$ **and** $uv: u \subseteq v$
from u **obtain** x **where** $ux: u \in x$ **and** $xc: x \in c$..
from $c xc$ **have** $fx: \text{filter } x$ **by** (*rule lemma-mem-chain-filter*)
from $fx ux uv$ **have** $vx: v \in x$ **by** (*rule filter.subset*)
with xc **show** $v \in \bigcup c$..
qed

lemma *Union-chain-filter*:

assumes *chain*: $c \in \text{chain superfrechet}$ **and** *nonempty*: $c \neq \{\}$
shows *filter* $(\bigcup c)$

proof (*rule filter.intro*)
 show $UNIV \in \bigcup c$ **using** *chain nonempty* **by** (*rule Union-chain-UNIV*)
next
 show $\{\} \notin \bigcup c$ **using** *chain* **by** (*rule Union-chain-empty*)
next
 fix $u v$ **assume** $u \in \bigcup c$ **and** $v \in \bigcup c$
 with *chain* **show** $u \cap v \in \bigcup c$ **by** (*rule Union-chain-Int*)
next
 fix $u v$ **assume** $u \in \bigcup c$ **and** $u \subseteq v$
 with *chain* **show** $v \in \bigcup c$ **by** (*rule Union-chain-subset*)
qed

lemma *lemma-mem-chain-frechet-subset*:
 $\llbracket c \in \text{chain superfrechet}; x \in c \rrbracket \implies \text{frechet} \subseteq x$
by (*unfold superfrechet-def chain-def, blast*)

lemma *Union-chain-superfrechet*:
 $\llbracket c \neq \{\}; c \in \text{chain superfrechet} \rrbracket \implies \bigcup c \in \text{superfrechet}$
proof (*rule superfrechetI*)
 assume 1: $c \in \text{chain superfrechet}$ **and** 2: $c \neq \{\}$
 thus *filter* $(\bigcup c)$ **by** (*rule Union-chain-filter*)
 from 2 **obtain** x **where** 3: $x \in c$ **by** *blast*
 from 1 3 **have** $\text{frechet} \subseteq x$ **by** (*rule lemma-mem-chain-frechet-subset*)
 also from 3 **have** $x \subseteq \bigcup c$ **by** *blast*
 finally **show** $\text{frechet} \subseteq \bigcup c$.
qed

4.4.2 Existence of free ultrafilter

lemma *max-cofinite-filter-Ex*:
 $\exists U \in \text{superfrechet}. \forall G \in \text{superfrechet}. U \subseteq G \longrightarrow U = G$
proof (*rule Zorn-Lemma2 [rule-format]*)
 fix c **assume** $c: c \in \text{chain superfrechet}$
 show $\exists U \in \text{superfrechet}. \forall G \in c. G \subseteq U$ (**is** ? U)
proof (*cases*)
 assume $c = \{\}$
 with *frechet-in-superfrechet* **show** ? U **by** *blast*
next
 assume $A: c \neq \{\}$
 from A c **have** $\bigcup c \in \text{superfrechet}$
 by (*rule Union-chain-superfrechet*)
 thus ? U **by** *blast*
qed
qed

lemma *mem-superfrechet-all-infinite*:
 $\llbracket U \in \text{superfrechet}; A \in U \rrbracket \implies \text{infinite } A$
proof
 assume $U: U \in \text{superfrechet}$ **and** $A: A \in U$ **and** *fin*: *finite* A

```

from  $U$  have  $fil$ : filter  $U$  and  $fre$ : frechet  $\subseteq U$ 
  by (simp-all add: superfrechet-def)
from  $fin$  have  $- A \in frechet$  by (simp add: frechet-def)
with  $fre$  have  $cA$ :  $- A \in U$  by (rule subsetD)
from  $fil$   $A$   $cA$  have  $A \cap - A \in U$  by (rule filter.Int)
with  $fil$  show  $False$  by (simp add: filter.empty)
qed

```

There exists a free ultrafilter on any infinite set

lemma *freeultrafilter-Ex*:

$\exists U::'a$ set set. *freeultrafilter* U

proof –

from *max-cofinite-filter-Ex* **obtain** U

where U : $U \in superfrechet$

and max [rule-format]: $\forall G \in superfrechet. U \subseteq G \longrightarrow U = G ..$

from U **have** fil : filter U **by** (rule superfrechetD1)

from U **have** fre : frechet $\subseteq U$ **by** (rule superfrechetD2)

have $ultra$: *ultrafilter-axioms* U

proof (rule *filter.max-filter-ultrafilter* [OF fil])

fix G **assume** G : filter G **and** UG : $U \subseteq G$

from fre UG **have** frechet $\subseteq G$ **by** *simp*

with G **have** $G \in superfrechet$ **by** (rule *superfrechetI*)

from *this* UG **show** $U = G$ **by** (rule *max*)

qed

have $free$: *freeultrafilter-axioms* U

proof (rule *freeultrafilter-axioms.intro*)

fix A **assume** $A \in U$

with U **show** *infinite* A **by** (rule *mem-superfrechet-all-infinite*)

qed

from fil $ultra$ $free$ **have** *freeultrafilter* U

by (rule *freeultrafilter.intro* [OF *ultrafilter.intro*])

then **show** *?thesis* ..

qed

end

hide-const (open) *filter*

end

5 StarDef: Construction of Star Types Using Ultrafilters

theory *StarDef*

imports *Filter*

uses (*transfer.ML*)

begin

5.1 A Free Ultrafilter over the Naturals

definition

FreeUltrafilterNat :: *nat set set* (*U*) **where**
U = (*SOME U*. *freeultrafilter U*)

lemma *freeultrafilter-FreeUltrafilterNat*: *freeultrafilter U*

apply (*unfold FreeUltrafilterNat-def*)

apply (*rule someI-ex*)

apply (*rule freeultrafilter-Ex*)

apply (*rule nat-infinite*)

done

interpretation *FreeUltrafilterNat*: *freeultrafilter FreeUltrafilterNat*

by (*rule freeultrafilter-FreeUltrafilterNat*)

This rule takes the place of the old ultra tactic

lemma *ultra*:

$\llbracket \{n. P n\} \in \mathcal{U}; \{n. P n \longrightarrow Q n\} \in \mathcal{U} \rrbracket \implies \{n. Q n\} \in \mathcal{U}$

by (*simp add: Collect-imp-eq*)

FreeUltrafilterNat.Un-iff FreeUltrafilterNat.Compl-iff)

5.2 Definition of *star* type constructor

definition

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

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

typedef (**open**) *'a star* = *star* :: $(nat \Rightarrow 'a)$ *set set*

unfolding *star-def* **by** (*auto intro: quotientI*)

definition

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

theorem *star-cases* [*case-names star-n*, *cases type: star*]:

$(\bigwedge X. x = \text{star-n } X \implies P) \implies P$

by (*cases x*, *unfold star-n-def star-def*, *erule quotientE*, *fast*)

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

by (*auto*, *rule-tac x=x* **in** *star-cases*, *simp*)

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

by (*auto*, *rule-tac x=x* **in** *star-cases*, *auto*)

Proving that *starrel* is an equivalence relation

lemma *starrel-iff* [*iff*]: $((X, Y) \in \text{starrel}) = (\{n. X\ n = Y\ n\} \in \mathcal{U})$
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** (*auto intro: transI elim!: ultra*)
qed

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

lemma *starrel-in-star*: $\text{starrel} \llbracket \{x\} \in \text{star} \rrbracket$
by (*simp add: star-def quotientI*)

lemma *star-n-eq-iff*: $(\text{star-n } X = \text{star-n } Y) = (\{n. X\ n = Y\ n\} \in \mathcal{U})$
by (*simp add: star-n-def Abs-star-inject starrel-in-star equiv-starrel-iff*)

5.3 Transfer principle

This introduction rule starts each transfer proof.

lemma *transfer-start*:
 $P \equiv \{n. Q\} \in \mathcal{U} \implies \text{Trueprop } P \equiv \text{Trueprop } Q$
by (*subgoal-tac P \equiv Q, simp, simp add: atomize-eq*)

Initialize transfer tactic.

use *transfer.ML*
setup *Transfer-Principle.setup*

method-setup *transfer* = \llbracket
Attrib.thms \gg (*fn ths => fn ctxt =>*
SIMPLE-METHOD' (*Transfer-Principle.transfer-tac ctxt ths*))
 \gg *transfer principle*

Transfer introduction rules.

lemma *transfer-ex* [*transfer-intro*]:
 $\llbracket \bigwedge X. p (\text{star-n } X) \equiv \{n. P\ n (X\ n)\} \in \mathcal{U} \rrbracket$
 $\implies \exists x::'a\ \text{star}. p\ x \equiv \{n. \exists x. P\ n\ x\} \in \mathcal{U}$
by (*simp only: ex-star-eq FreeUltrafilterNat.Collect-ex*)

lemma *transfer-all* [*transfer-intro*]:
 $\llbracket \bigwedge X. p (\text{star-n } X) \equiv \{n. P\ n (X\ n)\} \in \mathcal{U} \rrbracket$
 $\implies \forall x::'a\ \text{star}. p\ x \equiv \{n. \forall x. P\ n\ x\} \in \mathcal{U}$
by (*simp only: all-star-eq FreeUltrafilterNat.Collect-all*)

lemma *transfer-not* [*transfer-intro*]:
 $\llbracket p \equiv \{n. P\ n\} \in \mathcal{U} \rrbracket \implies \neg p \equiv \{n. \neg P\ n\} \in \mathcal{U}$

by (simp only: FreeUltrafilterNat.Collect-not)

lemma transfer-conj [transfer-intro]:

$$\begin{aligned} & \llbracket p \equiv \{n. P\ n\} \in \mathcal{U}; q \equiv \{n. Q\ n\} \in \mathcal{U} \rrbracket \\ & \implies p \wedge q \equiv \{n. P\ n \wedge Q\ n\} \in \mathcal{U} \end{aligned}$$

by (simp only: FreeUltrafilterNat.Collect-conj)

lemma transfer-disj [transfer-intro]:

$$\begin{aligned} & \llbracket p \equiv \{n. P\ n\} \in \mathcal{U}; q \equiv \{n. Q\ n\} \in \mathcal{U} \rrbracket \\ & \implies p \vee q \equiv \{n. P\ n \vee Q\ n\} \in \mathcal{U} \end{aligned}$$

by (simp only: FreeUltrafilterNat.Collect-disj)

lemma transfer-imp [transfer-intro]:

$$\begin{aligned} & \llbracket p \equiv \{n. P\ n\} \in \mathcal{U}; q \equiv \{n. Q\ n\} \in \mathcal{U} \rrbracket \\ & \implies p \longrightarrow q \equiv \{n. P\ n \longrightarrow Q\ n\} \in \mathcal{U} \end{aligned}$$

by (simp only: imp-conv-disj transfer-disj transfer-not)

lemma transfer-iff [transfer-intro]:

$$\begin{aligned} & \llbracket p \equiv \{n. P\ n\} \in \mathcal{U}; q \equiv \{n. Q\ n\} \in \mathcal{U} \rrbracket \\ & \implies p = q \equiv \{n. P\ n = Q\ n\} \in \mathcal{U} \end{aligned}$$

by (simp only: iff-conv-conj-imp transfer-conj transfer-imp)

lemma transfer-if-bool [transfer-intro]:

$$\begin{aligned} & \llbracket p \equiv \{n. P\ n\} \in \mathcal{U}; x \equiv \{n. X\ n\} \in \mathcal{U}; y \equiv \{n. Y\ n\} \in \mathcal{U} \rrbracket \\ & \implies (\text{if } p \text{ then } x \text{ else } y) \equiv \{n. \text{if } P\ n \text{ then } X\ n \text{ else } Y\ n\} \in \mathcal{U} \end{aligned}$$

by (simp only: if-bool-eq-conj transfer-conj transfer-imp transfer-not)

lemma transfer-eq [transfer-intro]:

$$\llbracket x \equiv \text{star-n } X; y \equiv \text{star-n } Y \rrbracket \implies x = y \equiv \{n. X\ n = Y\ n\} \in \mathcal{U}$$

by (simp only: star-n-eq-iff)

lemma transfer-if [transfer-intro]:

$$\begin{aligned} & \llbracket p \equiv \{n. P\ n\} \in \mathcal{U}; x \equiv \text{star-n } X; y \equiv \text{star-n } Y \rrbracket \\ & \implies (\text{if } p \text{ then } x \text{ else } y) \equiv \text{star-n } (\lambda n. \text{if } P\ n \text{ then } X\ n \text{ else } Y\ n) \end{aligned}$$

apply (rule eq-reflection)

apply (auto simp add: star-n-eq-iff transfer-not elim!: ultra)

done

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

$$\begin{aligned} & \llbracket \bigwedge X. f (\text{star-n } X) = g (\text{star-n } X) \rrbracket \\ & \equiv \{n. F\ n (X\ n) = G\ n (X\ n)\} \in \mathcal{U} \\ & \implies f = g \equiv \{n. F\ n = G\ n\} \in \mathcal{U} \end{aligned}$$

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

lemma transfer-star-n [transfer-intro]: star-n X ≡ star-n (λn. X n)

by (rule reflexive)

lemma transfer-bool [transfer-intro]: p ≡ {n. p} ∈ U

by (simp add: atomize-eq)

5.4 Standard elements

definition

$star-of :: 'a \Rightarrow 'a\ star\ \mathbf{where}$
 $star-of\ x == star-n\ (\lambda n.\ x)$

definition

$Standard :: 'a\ star\ set\ \mathbf{where}$
 $Standard = range\ star-of$

Transfer tactic should remove occurrences of $star-of$

setup $\ll Transfer-Principle.add-const\ StarDef.star-of \gg$

declare $star-of-def\ [transfer-intro]$

lemma $star-of-inject$: $(star-of\ x = star-of\ y) = (x = y)$
by $(transfer, rule\ refl)$

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

5.5 Internal functions

definition

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

lemma $Ifun-congruent2$:

$congruent2\ starrel\ starrel\ (\lambda F\ X.\ starrel\ \{\lambda n.\ F\ n\ (X\ n)\})$
by $(auto\ simp\ add: congruent2-def\ equiv-starrel-iff\ elim!: ultra)$

lemma $Ifun-star-n$: $star-n\ F\ \star\ star-n\ X = star-n\ (\lambda n.\ F\ n\ (X\ n))$

by $(simp\ add: Ifun-def\ star-n-def\ Abs-star-inverse\ starrel-in-star$
 $UN-equiv-class2\ [OF\ equiv-starrel\ equiv-starrel\ Ifun-congruent2])$

Transfer tactic should remove occurrences of $Ifun$

setup $\ll Transfer-Principle.add-const\ StarDef.Ifun \gg$

lemma $transfer-Ifun\ [transfer-intro]$:

$\llbracket f \equiv star-n\ F; x \equiv star-n\ X \rrbracket \Longrightarrow f\ \star\ x \equiv star-n\ (\lambda n.\ F\ n\ (X\ n))$
by $(simp\ only: Ifun-star-n)$

lemma $Ifun-star-of\ [simp]$: $star-of\ f\ \star\ star-of\ x = star-of\ (f\ x)$

by $(transfer, rule\ refl)$

lemma $Standard-Ifun\ [simp]$:

$\llbracket f \in Standard; x \in Standard \rrbracket \Longrightarrow f\ \star\ x \in Standard$
by $(auto\ simp\ add: Standard-def)$

Nonstandard extensions of functions

definition

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

definition

$starfun2 :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a \ star \Rightarrow 'b \ star \Rightarrow 'c \ star)$
 $(*f2* - [80] \ 80)$ **where**
 $starfun2 \ f == \lambda x \ y. \ star\text{-of} \ f \ \star \ x \ \star \ y$

declare $starfun\text{-def}$ [*transfer-unfold*]

declare $starfun2\text{-def}$ [*transfer-unfold*]

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

lemma $starfun2\text{-star-n}$:

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

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

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

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

lemma $Standard\text{-starfun2}$ [*simp*]:

$\llbracket x \in Standard; y \in Standard \rrbracket \Longrightarrow starfun2 \ f \ x \ y \in Standard$
by (*simp add: starfun2-def*)

lemma $Standard\text{-starfun-iff}$:

assumes $inj: \bigwedge x \ y. \ f \ x = f \ y \Longrightarrow x = y$
shows $(starfun \ f \ x \in Standard) = (x \in Standard)$

proof

assume $x \in Standard$

thus $starfun \ f \ x \in Standard$ **by** *simp*

next

have inj' : $\bigwedge x \ y. \ starfun \ f \ x = starfun \ f \ y \Longrightarrow x = y$

using inj **by** *transfer*

assume $starfun \ f \ x \in Standard$

then obtain b **where** $b: starfun \ f \ x = star\text{-of} \ b$

unfolding $Standard\text{-def}$ **..**

hence $\exists x. \ starfun \ f \ x = star\text{-of} \ b$ **..**

hence $\exists a. \ f \ a = b$ **by** *transfer*

then obtain a **where** $f \ a = b$ **..**

hence $starfun \ f \ (star\text{-of} \ a) = star\text{-of} \ b$ **by** *transfer*

with b **have** $\text{starfun } f \ x = \text{starfun } f \ (\text{star-of } a)$ **by** *simp*
hence $x = \text{star-of } a$ **by** (rule *inj'*)
thus $x \in \text{Standard}$
unfolding *Standard-def* **by** *auto*
qed

lemma *Standard-starfun2-iff*:

assumes *inj*: $\bigwedge a \ b \ a' \ b'. f \ a \ b = f \ a' \ b' \implies a = a' \wedge b = b'$

shows $(\text{starfun2 } f \ x \ y \in \text{Standard}) = (x \in \text{Standard} \wedge y \in \text{Standard})$

proof

assume $x \in \text{Standard} \wedge y \in \text{Standard}$

thus $\text{starfun2 } f \ x \ y \in \text{Standard}$ **by** *simp*

next

have *inj'*: $\bigwedge x \ y \ z \ w. \text{starfun2 } f \ x \ y = \text{starfun2 } f \ z \ w \implies x = z \wedge y = w$

using *inj* **by** *transfer*

assume $\text{starfun2 } f \ x \ y \in \text{Standard}$

then obtain c **where** $c: \text{starfun2 } f \ x \ y = \text{star-of } c$

unfolding *Standard-def* **..**

hence $\exists x \ y. \text{starfun2 } f \ x \ y = \text{star-of } c$ **by** *auto*

hence $\exists a \ b. f \ a \ b = c$ **by** *transfer*

then obtain $a \ b$ **where** $f \ a \ b = c$ **by** *auto*

hence $\text{starfun2 } f \ (\text{star-of } a) \ (\text{star-of } b) = \text{star-of } c$

by *transfer*

with c **have** $\text{starfun2 } f \ x \ y = \text{starfun2 } f \ (\text{star-of } a) \ (\text{star-of } b)$

by *simp*

hence $x = \text{star-of } a \wedge y = \text{star-of } b$

by (rule *inj'*)

thus $x \in \text{Standard} \wedge y \in \text{Standard}$

unfolding *Standard-def* **by** *auto*

qed

5.6 Internal predicates

definition *unstar* :: $\text{bool } \text{star} \Rightarrow \text{bool}$ **where**

$\text{unstar } b \longleftrightarrow b = \text{star-of } \text{True}$

lemma *unstar-star-n*: $\text{unstar } (\text{star-n } P) = (\{n. P \ n\} \in \mathcal{U})$

by (*simp add: unstar-def star-of-def star-n-eq-iff*)

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

by (*simp add: unstar-def star-of-inject*)

Transfer tactic should remove occurrences of *unstar*

setup $\ll \text{Transfer-Principle.add-const StarDef.unstar} \gg$

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

$p \equiv \text{star-n } P \implies \text{unstar } p \equiv \{n. P \ n\} \in \mathcal{U}$

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) ($star-n X$) = ($\{n. P (X n)\} \in \mathcal{U}$)
by ($simp$ only: $starP-def$ $star-of-def$ $Ifun-star-n$ $unstar-star-n$)

lemma $starP2-star-n$:

($*p2*$ P) ($star-n X$) ($star-n Y$) = ($\{n. P (X n) (Y n)\} \in \mathcal{U}$)
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 $refl$)

5.7 Internal sets

definition

$Iset :: 'a \ set \ star \Rightarrow 'a \ star \ set$ **where**
 $Iset A = \{x. (*p2* op \in) x A\}$

lemma $Iset-star-n$:

($star-n X \in Iset (star-n A)$) = ($\{n. X n \in A n\} \in \mathcal{U}$)
by ($simp$ add: $Iset-def$ $starP2-star-n$)

Transfer tactic should remove occurrences of $Iset$

setup \ll $Transfer-Principle.add-const StarDef.Iset$ \gg

lemma $transfer-mem$ [$transfer-intro$]:

$\llbracket x \equiv star-n X; a \equiv Iset (star-n A) \rrbracket$
 $\implies x \in a \equiv \{n. X n \in A n\} \in \mathcal{U}$

by ($simp$ only: $Iset-star-n$)

lemma $transfer-Collect$ [$transfer-intro$]:

$\llbracket \bigwedge X. p (star-n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket$
 $\implies Collect p \equiv Iset (star-n (\lambda n. Collect (P n)))$

by ($simp$ add: $atomize-eq$ $set-eq-iff$ $all-star-eq$ $Iset-star-n$)

lemma $transfer-set-eq$ [$transfer-intro$]:

$\llbracket a \equiv Iset (star-n A); b \equiv Iset (star-n B) \rrbracket$

$$\implies a = b \equiv \{n. A n = B n\} \in \mathcal{U}$$

by (simp only: set-eq-iff transfer-all transfer-iff transfer-mem)

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

$$\llbracket a \equiv Iset (star-n A); \bigwedge X. p (star-n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket$$

$$\implies \forall x \in a. p x \equiv \{n. \forall x \in A n. P n x\} \in \mathcal{U}$$

by (simp only: Ball-def transfer-all transfer-imp transfer-mem)

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

$$\llbracket a \equiv Iset (star-n A); \bigwedge X. p (star-n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket$$

$$\implies \exists x \in a. p x \equiv \{n. \exists x \in A n. P n x\} \in \mathcal{U}$$

by (simp only: Bex-def transfer-ex transfer-conj transfer-mem)

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

$$\llbracket a \equiv star-n A \rrbracket \implies Iset a \equiv Iset (star-n (\lambda n. A n))$$

by simp

Nonstandard extensions of sets.

definition

starset :: 'a set \Rightarrow 'a star set (*s* - [80] 80) **where**

starset A = Iset (star-of A)

declare *starset-def* [transfer-unfold]

lemma *starset-mem*: (star-of x \in *s* A) = (x \in A)

by (transfer, rule refl)

lemma *starset-UNIV*: *s* (UNIV::'a set) = (UNIV::'a star set)

by (transfer UNIV-def, rule refl)

lemma *starset-empty*: *s* {} = {}

by (transfer empty-def, rule refl)

lemma *starset-insert*: *s* (insert x A) = insert (star-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*: $(*s* A \subseteq *s* B) = (A \subseteq B)$
by (*transfer subset-eq*, *rule refl*)

lemma *starset-eq*: $(*s* A = *s* B) = (A = B)$
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*

5.8 Syntactic classes

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

definition
star-zero-def: $0 \equiv \text{star-of } 0$

instance ..

end

instantiation *star* :: (*one*) *one*
begin

definition
star-one-def: $1 \equiv \text{star-of } 1$

instance ..

end

instantiation *star* :: (*plus*) *plus*
begin

definition
star-add-def: $(op +) \equiv *f2* (op +)$

instance ..

end

instantiation *star* :: (*times*) *times*
begin

definition

star-mult-def: $(op \ *) \equiv *f2* (op \ *)$

instance ..

end

instantiation *star* :: (*uminus*) *uminus*
begin

definition

star-minus-def: $uminus \equiv *f* \ minus$

instance ..

end

instantiation *star* :: (*minus*) *minus*
begin

definition

star-diff-def: $(op \ -) \equiv *f2* (op \ -)$

instance ..

end

instantiation *star* :: (*abs*) *abs*
begin

definition

star-abs-def: $abs \equiv *f* \ abs$

instance ..

end

instantiation *star* :: (*sgn*) *sgn*
begin

definition

star-sgn-def: $sgn \equiv *f* \ sgn$

instance ..

end

instantiation *star* :: (*inverse*) *inverse*
begin

definition

star-divide-def: $(op \ /) \equiv *f2* (op \ /)$

definition

star-inverse-def: $inverse \equiv *f* inverse$

instance ..

end

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

instantiation *star* :: (*Divides.div*) *Divides.div*
begin

definition

star-div-def: $(op \ div) \equiv *f2* (op \ div)$

definition

star-mod-def: $(op \ mod) \equiv *f2* (op \ mod)$

instance ..

end

instantiation *star* :: (*ord*) *ord*
begin

definition

star-le-def: $(op \ \leq) \equiv *p2* (op \ \leq)$

definition

star-less-def: $(op \ <) \equiv *p2* (op \ <)$

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-div-def *star-mod-def*

Class operations preserve standard elements

lemma *Standard-zero*: $0 \in \text{Standard}$
by (*simp add: star-zero-def*)

lemma *Standard-one*: $1 \in \text{Standard}$
by (*simp add: star-one-def*)

lemma *Standard-add*: $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies x + y \in \text{Standard}$
by (*simp add: star-add-def*)

lemma *Standard-diff*: $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies x - y \in \text{Standard}$
by (*simp add: star-diff-def*)

lemma *Standard-minus*: $x \in \text{Standard} \implies -x \in \text{Standard}$
by (*simp add: star-minus-def*)

lemma *Standard-mult*: $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies x * y \in \text{Standard}$
by (*simp add: star-mult-def*)

lemma *Standard-divide*: $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies x / y \in \text{Standard}$
by (*simp add: star-divide-def*)

lemma *Standard-inverse*: $x \in \text{Standard} \implies \text{inverse } x \in \text{Standard}$
by (*simp add: star-inverse-def*)

lemma *Standard-abs*: $x \in \text{Standard} \implies \text{abs } x \in \text{Standard}$
by (*simp add: star-abs-def*)

lemma *Standard-div*: $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies x \text{ div } y \in \text{Standard}$
by (*simp add: star-div-def*)

lemma *Standard-mod*: $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies x \text{ mod } y \in \text{Standard}$
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-div* *Standard-mod*

star-of preserves class operations

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

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

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

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

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

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

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

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

lemma *star-of-abs*: $star-of (abs x) = abs (star-of x)$
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-div  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

```

5.9 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  $\equiv$  *f2* inf

instance
  by default (transfer star-inf-def, auto)+

end

instantiation star :: (semilattice-sup) semilattice-sup
begin

definition
  star-sup-def [transfer-unfold]: sup  $\equiv$  *f2* sup

instance
  by default (transfer star-sup-def, auto)+

end

```

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

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

lemma *Standard-inf [simp]*:
 $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies \text{inf } x \ y \in \text{Standard}$
by (*simp add: star-inf-def*)

lemma *Standard-sup [simp]*:
 $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies \text{sup } x \ y \in \text{Standard}$
by (*simp add: star-sup-def*)

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

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

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

lemma *star-max-def [transfer-unfold]*: $\text{max} = *f2*$ *max*
apply (*rule ext*, *rule ext*)
apply (*unfold max-def*, *transfer*, *fold max-def*)
apply (*rule refl*)
done

lemma *star-min-def [transfer-unfold]*: $\text{min} = *f2*$ *min*
apply (*rule ext*, *rule ext*)
apply (*unfold min-def*, *transfer*, *fold min-def*)
apply (*rule refl*)
done

lemma *Standard-max [simp]*:
 $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies \text{max } x \ y \in \text{Standard}$
by (*simp add: star-max-def*)

lemma *Standard-min [simp]*:
 $\llbracket x \in \text{Standard}; y \in \text{Standard} \rrbracket \implies \text{min } x \ y \in \text{Standard}$
by (*simp add: star-min-def*)

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

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

5.10 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* :: (*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*, *rule add-imp-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-minus*)
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-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*

..

5.11 Ring and field classes

instance *star* :: (*semiring*) *semiring*

apply (*intro-classes*)

apply (*transfer*, rule *left-distrib*)

apply (*transfer*, rule *right-distrib*)

done

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

by *intro-classes* (*transfer*, *simp*)+

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

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

by (*intro-classes*, *transfer*, rule *left-distrib*)

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*, rule *zero-neq-one*)

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

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

instance *star* :: (*no-zero-divisors*) *no-zero-divisors*

by (*intro-classes*, *transfer*, rule *no-zero-divisors*)

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

instance *star* :: (*comm-semiring-1-cancel*) *comm-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* :: (*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 :: (division-ring) division-ring
apply (intro-classes)
apply (transfer, erule left-inverse)
apply (transfer, erule right-inverse)
apply (transfer, fact divide-inverse)
done

```

```

instance star :: (division-ring-inverse-zero) division-ring-inverse-zero
by (intro-classes, transfer, rule inverse-zero)

```

```

instance star :: (field) field
apply (intro-classes)
apply (transfer, erule left-inverse)
apply (transfer, rule divide-inverse)
done

```

```

instance star :: (field-inverse-zero) field-inverse-zero
apply intro-classes
apply (rule inverse-zero)
done

```

```

instance star :: (ordered-semiring) ordered-semiring
apply (intro-classes)
apply (transfer, erule (1) mult-left-mono)
apply (transfer, erule (1) mult-right-mono)
done

```

```

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

```

```

instance star :: (linordered-semiring-strict) linordered-semiring-strict
apply (intro-classes)
apply (transfer, erule (1) mult-strict-left-mono)
apply (transfer, erule (1) mult-strict-right-mono)
done

```

```

instance star :: (ordered-comm-semiring) ordered-comm-semiring
by (intro-classes, transfer, rule 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, rule mult-strict-left-mono)

```

```

instance star :: (ordered-ring) ordered-ring ..
instance star :: (ordered-ring-abs) ordered-ring-abs
  by intro-classes (transfer, rule abs-eq-mult)

```

```

instance star :: (abs-if) abs-if

```

```

by (intro-classes, transfer, rule abs-if)

instance star :: (sgn-if) sgn-if
by (intro-classes, transfer, rule sgn-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, rule zero-less-one)

instance star :: (linordered-idom) linordered-idom ..
instance star :: (linordered-field) linordered-field ..
instance star :: (linordered-field-inverse-zero) linordered-field-inverse-zero ..

```

5.12 Power

```

lemma star-power-def [transfer-unfold]:
  (op ^) ≡ λx n. ( *f* (λx. x ^ n) ) x
proof (rule eq-reflection, rule ext, rule ext)
  fix n :: nat
  show ∧x::'a star. x ^ n = ( *f* (λx. x ^ n) ) x
  proof (induct n)
    case 0
    have ∧x::'a star. ( *f* (λx. 1) ) x = 1
    by transfer simp
    then show ?case by simp
  next
  case (Suc n)
  have ∧x::'a star. x * ( *f* (λx::'a. x ^ n) ) x = ( *f* (λx::'a. x * x ^ n) ) x
  by transfer simp
  with Suc show ?case by simp
qed
qed

```

```

lemma Standard-power [simp]: x ∈ Standard ⇒ x ^ n ∈ Standard
  by (simp add: star-power-def)

```

```

lemma star-of-power [simp]: star-of (x ^ n) = star-of x ^ n
  by transfer (rule refl)

```

5.13 Number classes

```

instance star :: (numeral) numeral ..

```

```

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

```

```

lemma Standard-numeral [simp]: numeral k ∈ Standard

```

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

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

lemma *star-neg-numeral-def* [*transfer-unfold*]:
neg-numeral k = *star-of* (*neg-numeral k*)
by (*simp only: neg-numeral-def star-of-minus star-of-numeral*)

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

lemma *star-of-neg-numeral* [*simp*]: *star-of* (*neg-numeral k*) = *neg-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 neg-numeral k, simplified star-of-numeral*]
star-of-le [*of neg-numeral k, simplified star-of-numeral*]
star-of-eq [*of neg-numeral k, simplified star-of-numeral*]
star-of-less [*of - neg-numeral k, simplified star-of-numeral*]
star-of-le [*of - neg-numeral k, simplified star-of-numeral*]
star-of-eq [*of - neg-numeral k, simplified star-of-numeral*] **for** *k*

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

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

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

```

5.14 Finite class

```

lemma starset-finite: finite A ⇒ ** 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

```

6 HyperNat: 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 = ** Suc

```

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

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

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

lemma *hypnat-add-is-0* [iff]: $!!m\ n. (m+n = (0::hypnat)) = (m=0 \ \&\ n=0)$
by *transfer* (rule *add-is-0*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

lemma *hypnat-add-diff-inverse*: $!!m n. \sim m < n ==> n+(m-n) = (m::hypnat)$
by *transfer* (*rule add-diff-inverse*)

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

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

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

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

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

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

lemma *hypnat-add-one-self-less* [*simp*]: $(x::hypnat) < x + (1::hypnat)$
by (*insert add-strict-left-mono* [*OF zero-less-one*], *auto*)

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

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

lemma *hypnat-gt-zero-iff2*: $(0 < n) = (\exists m. n = m + (1::hypnat))$
apply *safe*
apply (*rule-tac* $x = n - (1::hypnat)$ **in** *exI*)
apply (*simp add: hypnat-gt-zero-iff*)
apply (*insert add-le-less-mono* [*OF - zero-less-one, of 0*], *auto*)
done

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

lemma *hypnat-diff-split*:
 $P(a - b::hypnat) = ((a < b \leftrightarrow P 0) \& (ALL d. a = b + d \leftrightarrow P d))$

— elimination of $-$ on *hypnat*
proof (*cases a < b rule: case-split*)
case *True*
 thus *?thesis*
 by (*auto simp add: hypnat-add-self-not-less order-less-imp-le hypnat-diff-is-0-eq [THEN iffD2]*)
next
case *False*
 thus *?thesis*
 by (*auto simp add: linorder-not-less dest: order-le-less-trans*)
qed

6.2 Properties of the set of embedded natural numbers

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

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

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

lemma *of-nat-eq-add [rule-format]:*
 $\forall d::hypnat. of-nat m = of-nat n + d \implies d \in range\ of-nat$
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 ∈ Nats; b ∈ Nats|] ==> (a-b :: hypnat) ∈ Nats*
by (*simp add: Nats-eq-Standard*)

6.3 Infinite Hypernatural Numbers – *HNatInfinite*

definition

HNatInfinite :: *hypnat set* **where**
HNatInfinite = {*n. n ∉ Nats*}

lemma *Nats-not-HNatInfinite-iff*: $(x \in \text{Nats}) = (x \notin \text{HNatInfinite})$
by (*simp add: HNatInfinite-def*)

lemma *HNatInfinite-not-Nats-iff*: $(x \in \text{HNatInfinite}) = (x \notin \text{Nats})$
by (*simp add: HNatInfinite-def*)

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

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

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

lemma *star-of-le-HNatInfinite*: $N \in \text{HNatInfinite} \implies \text{star-of } n \leq N$
by (*rule star-of-less-HNatInfinite [THEN order-less-imp-le]*)

6.3.1 Closure Rules

lemma *Nats-less-HNatInfinite*: $\llbracket x \in \text{Nats}; y \in \text{HNatInfinite} \rrbracket \implies x < y$
by (*auto simp add: Nats-def star-of-less-HNatInfinite*)

lemma *Nats-le-HNatInfinite*: $\llbracket x \in \text{Nats}; y \in \text{HNatInfinite} \rrbracket \implies x \leq y$
by (*rule Nats-less-HNatInfinite [THEN order-less-imp-le]*)

lemma *zero-less-HNatInfinite*: $x \in \text{HNatInfinite} \implies 0 < x$
by (*simp add: Nats-less-HNatInfinite*)

lemma *one-less-HNatInfinite*: $x \in \text{HNatInfinite} \implies 1 < x$
by (*simp add: Nats-less-HNatInfinite*)

lemma *one-le-HNatInfinite*: $x \in \text{HNatInfinite} \implies 1 \leq x$
by (*simp add: Nats-le-HNatInfinite*)

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

lemma *Nats-downward-closed*:

```

  [[x ∈ Nats; (y::hypnat) ≤ x]] ⇒ y ∈ Nats
  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 ∈ HNatInfinite; x ≤ y]] ⇒ y ∈ HNatInfinite
  apply (simp only: HNatInfinite-not-Nats-iff)
  apply (erule contrapos-nn)
  apply (erule (1) Nats-downward-closed)
  done

```

lemma *HNatInfinite-add*: $x \in \text{HNatInfinite} \implies x + y \in \text{HNatInfinite}$

```

  apply (erule HNatInfinite-upward-closed)
  apply (rule hypnat-le-add1)
  done

```

lemma *HNatInfinite-add-one*: $x \in \text{HNatInfinite} \implies x + 1 \in \text{HNatInfinite}$

```

  by (rule HNatInfinite-add)

```

lemma *HNatInfinite-diff*:

```

  [[x ∈ HNatInfinite; y ∈ Nats]] ⇒ x - y ∈ 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*: $x \in \text{HNatInfinite} \implies \exists y. x = y + (1::\text{hypnat})$

```

  apply (rule-tac x = x - (1::hypnat) in exI)
  apply (simp add: Nats-le-HNatInfinite)
  done

```

6.4 Existence of an infinite hypernatural number

definition

```

  whn :: hypnat where
  hypnat-omega-def: whn = star-n (%n::nat. n)

```

lemma *hypnat-of-nat-neq-whn*: $\text{hypnat-of-nat } n \neq \text{whn}$

```

  by (simp add: hypnat-omega-def star-of-def star-n-eq-iff)

```

lemma *whn-neq-hypnat-of-nat*: $\text{whn} \neq \text{hypnat-of-nat } n$

```

  by (simp add: 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*]: $\{n::nat. m < n\} \in FreeUltrafilterNat$
apply (*insert finite-atMost [of m]*)
apply (*drule FreeUltrafilterNat.finite*)
apply (*drule FreeUltrafilterNat.not-memD*)
apply (*simp add: Collect-neg-eq [symmetric] linorder-not-le atMost-def*)
done

lemma *Compl-Collect-le*: $-\{n::nat. N \leq n\} = \{n. n < N\}$
by (*simp add: Collect-neg-eq [symmetric] linorder-not-le*)

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

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

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

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

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

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

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

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

6.4.1 Alternative characterization of the set of infinite hypernaturals

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

lemma *HNatInfinite-FreeUltrafilterNat-lemma*:
assumes $\forall N::nat. \{n. f\ n \neq N\} \in FreeUltrafilterNat$
shows $\{n. N < f\ n\} \in FreeUltrafilterNat$

```

apply (induct N)
using assms
apply (drule-tac x = 0 in spec, simp)
using assms
apply (drule-tac x = Suc N in spec)
apply (elim ultra, auto)
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

```

6.4.2 Alternative Characterization of *HNatInfinite* using Free Ultrafilter

```

lemma HNatInfinite-FreeUltrafilterNat:
   $star-n X \in HNatInfinite ==> \forall u. \{n. u < X n\}: FreeUltrafilterNat$ 
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:
   $\forall u. \{n. u < X n\}: FreeUltrafilterNat ==> star-n X \in HNatInfinite$ 
by (auto simp add: star-less-def starP2-star-n HNatInfinite-iff SHNat-eq hypnat-of-nat-eq)

```

```

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

```

6.5 Embedding of the Hypernaturals into other types

definition

```

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

```

```

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

```

```

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

```

```

lemma of-hypnat-hSuc:  $\bigwedge m. of-hypnat (hSuc m) = 1 + of-hypnat m$ 
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]:

$\bigwedge m n. \text{of-hypnat } (m * n) = \text{of-hypnat } m * \text{of-hypnat } n$
by *transfer* (rule *of-nat-mult*)

lemma *of-hypnat-less-iff* [simp]:

$\bigwedge m n. (\text{of-hypnat } m < (\text{of-hypnat } n :: 'a :: \text{linordered-semidom star})) = (m < n)$
by *transfer* (rule *of-nat-less-iff*)

lemma *of-hypnat-0-less-iff* [simp]:

$\bigwedge n. (0 < (\text{of-hypnat } n :: 'a :: \text{linordered-semidom star})) = (0 < n)$
by *transfer* (rule *of-nat-0-less-iff*)

lemma *of-hypnat-less-0-iff* [simp]:

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

lemma *of-hypnat-le-iff* [simp]:

$\bigwedge m n. (\text{of-hypnat } m \leq (\text{of-hypnat } n :: 'a :: \text{linordered-semidom star})) = (m \leq n)$
by *transfer* (rule *of-nat-le-iff*)

lemma *of-hypnat-0-le-iff* [simp]:

$\bigwedge n. 0 \leq (\text{of-hypnat } n :: 'a :: \text{linordered-semidom star})$
by *transfer* (rule *of-nat-0-le-iff*)

lemma *of-hypnat-le-0-iff* [simp]:

$\bigwedge m. ((\text{of-hypnat } m :: 'a :: \text{linordered-semidom star}) \leq 0) = (m = 0)$
by *transfer* (rule *of-nat-le-0-iff*)

lemma *of-hypnat-eq-iff* [simp]:

$\bigwedge m n. (\text{of-hypnat } m = (\text{of-hypnat } n :: 'a :: \text{linordered-semidom star})) = (m = n)$
by *transfer* (rule *of-nat-eq-iff*)

lemma *of-hypnat-eq-0-iff* [simp]:

$\bigwedge m. ((\text{of-hypnat } m :: 'a :: \text{linordered-semidom star}) = 0) = (m = 0)$
by *transfer* (rule *of-nat-eq-0-iff*)

lemma *HNatInfinite-of-hypnat-gt-zero*:

$N \in \text{HNatInfinite} \implies (0 :: 'a :: \text{linordered-semidom star}) < \text{of-hypnat } N$
by (rule *ccontr*, *simp* add: *linorder-not-less*)

end

7 HyperDef: Construction of Hyperreals Using Ultrafilters

theory *HyperDef*

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

definition

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

notation (*xsymbols*)

omega (ω) **and**
epsilon (ε)

notation (*HTML output*)

omega (ω) **and**
epsilon (ε)

7.1 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*]: $x \in \text{Standard} \implies \text{scaleR } r \ x \in \text{Standard}$
by (*simp add: star-scaleR-def*)

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

```

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

instance star :: (real-algebra) real-algebra
proof
  fix a :: real
  show  $\bigwedge x y :: 'a \text{ star}. \text{scaleR } a x * y = \text{scaleR } a (x * y)$ 
    by transfer (rule mult-scaleR-left)
  show  $\bigwedge x y :: 'a \text{ star}. x * \text{scaleR } a y = \text{scaleR } a (x * y)$ 
    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  $\in$  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
  fix r :: real
  show of-real r = star-of r
    by transfer simp
qed

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

```

7.2 Injection from *hypreal*

definition

of-hypreal :: *hypreal* \Rightarrow 'a::real-algebra-1 star **where**
 [transfer-unfold]: *of-hypreal* = *f* *of-real*

lemma *Standard-of-hypreal* [simp]:

$r \in \text{Standard} \implies \text{of-hypreal } r \in \text{Standard}$

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]:

$\bigwedge x y. \text{of-hypreal } (x + y) = \text{of-hypreal } x + \text{of-hypreal } y$

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

lemma *of-hypreal-minus* [simp]: $\bigwedge x. \text{of-hypreal } (- x) = - \text{of-hypreal } x$

by *transfer* (*rule of-real-minus*)

lemma *of-hypreal-diff* [simp]:

$\bigwedge x y. \text{of-hypreal } (x - y) = \text{of-hypreal } x - \text{of-hypreal } y$

by *transfer* (*rule of-real-diff*)

lemma *of-hypreal-mult* [simp]:

$\bigwedge x y. \text{of-hypreal } (x * y) = \text{of-hypreal } x * \text{of-hypreal } y$

by *transfer* (*rule of-real-mult*)

lemma *of-hypreal-inverse* [simp]:

$\bigwedge x. \text{of-hypreal } (\text{inverse } x) =$

inverse (*of-hypreal* $x :: 'a::\{\text{real-div-algebra, division-ring-inverse-zero}\}$ star)

by *transfer* (*rule of-real-inverse*)

lemma *of-hypreal-divide* [simp]:

$\bigwedge x y. \text{of-hypreal } (x / y) =$

(*of-hypreal* $x / \text{of-hypreal } y :: 'a::\{\text{real-field, field-inverse-zero}\}$ star)

by *transfer* (*rule of-real-divide*)

lemma *of-hypreal-eq-iff* [simp]:

$\bigwedge x y. (\text{of-hypreal } x = \text{of-hypreal } y) = (x = y)$

by *transfer* (*rule of-real-eq-iff*)

lemma *of-hypreal-eq-0-iff* [simp]:

$\bigwedge x. (\text{of-hypreal } x = 0) = (x = 0)$

by *transfer* (*rule of-real-eq-0-iff*)

7.3 Properties of *starrel*

lemma *lemma-starrel-refl* [*simp*]: $x \in \text{starrel} \{x\}$
by (*simp add: starrel-def*)

lemma *starrel-in-hypreal* [*simp*]: $\text{starrel}\{x\}:\text{star}$
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*]

7.4 *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*: $(X \in \text{Rep-star } x) = (x = \text{star-}n \ X)$
by (*cases x, simp add: star-n-def*)

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

lemma *Rep-star-star-n*: $X \in \text{Rep-star } (\text{star-}n \ X)$
by *simp*

7.5 Properties of *star-n*

lemma *star-n-add*:
 $\text{star-}n \ X + \text{star-}n \ Y = \text{star-}n \ (\%n. X \ n + Y \ n)$
by (*simp only: star-add-def starfun2-star-n*)

lemma *star-n-minus*:
 $-\ \text{star-}n \ X = \text{star-}n \ (\%n. -(X \ n))$
by (*simp only: star-minus-def starfun-star-n*)

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

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

lemma *star-n-inverse*:
 $\text{inverse } (\text{star-}n \ X) = \text{star-}n \ (\%n. \text{inverse}(X \ n))$
by (*simp only: star-inverse-def starfun-star-n*)

lemma *star-n-le*:
 $\text{star-}n \ X \leq \text{star-}n \ Y =$

$(\{n. X n \leq Y n\} \in \text{FreeUltrafilterNat})$
by (*simp only: star-le-def starP2-star-n*)

lemma *star-n-less*:

$\text{star-n } X < \text{star-n } Y = (\{n. X n < Y n\} \in \text{FreeUltrafilterNat})$
by (*simp only: star-less-def starP2-star-n*)

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

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

lemma *star-n-abs*:

$\text{abs } (\text{star-n } X) = \text{star-n } (\%n. \text{abs } (X n))$
by (*simp only: star-abs-def starfun-star-n*)

7.6 Misc Others

lemma *hypreal-not-refl2*: $!!(x::\text{hypreal}). x < y \implies x \neq y$
by (*auto*)

lemma *hypreal-eq-minus-iff*: $((x::\text{hypreal}) = y) = (x + - y = 0)$
by *auto*

lemma *hypreal-mult-left-cancel*: $(c::\text{hypreal}) \neq 0 \implies (c*a=c*b) = (a=b)$
by *auto*

lemma *hypreal-mult-right-cancel*: $(c::\text{hypreal}) \neq 0 \implies (a*c=b*c) = (a=b)$
by *auto*

lemma *hypreal-omega-gt-zero* [*simp*]: $0 < \text{omega}$
by (*simp add: omega-def star-n-zero-num star-n-less*)

7.7 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*: $\{n::\text{nat}. x = \text{real } n\} = \{\} \mid (\exists y. \{n::\text{nat}. x = \text{real } n\} = \{y\})$
by *force*

lemma *lemma-finite-omega-set*: *finite* $\{n::\text{nat}. x = \text{real } n\}$
by (*cut-tac x = x in lemma-omega-empty-singleton-disj, auto*)

lemma *not-ex-hypreal-of-real-eq-omega*:
 $\sim (\exists x. \text{hypreal-of-real } x = \text{omega})$

```

apply (simp add: omega-def)
apply (simp add: star-of-def star-n-eq-iff)
apply (auto simp add: real-of-nat-Suc diff-eq-eq [symmetric]
        lemma-finite-omega-set [THEN FreeUltrafilterNat.finite])
done

```

```

lemma hypreal-of-real-not-eq-omega: hypreal-of-real  $x \neq \text{omega}$ 
by (insert not-ex-hypreal-of-real-eq-omega, auto)

```

Existence of infinitesimal number also not corresponding to any real number

```

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

```

```

lemma lemma-finite-epsilon-set: finite { $n. x = \text{inverse}(\text{real}(\text{Suc } n))\}$ 
by (cut-tac  $x = x$  in lemma-epsilon-empty-singleton-disj, auto)

```

```

lemma not-ex-hypreal-of-real-eq-epsilon:  $\sim (\exists x. \text{hypreal-of-real } x = \text{epsilon})$ 
by (auto simp add: epsilon-def star-of-def star-n-eq-iff
        lemma-finite-epsilon-set [THEN FreeUltrafilterNat.finite])

```

```

lemma hypreal-of-real-not-eq-epsilon: hypreal-of-real  $x \neq \text{epsilon}$ 
by (insert not-ex-hypreal-of-real-eq-epsilon, auto)

```

```

lemma hypreal-epsilon-not-zero:  $\text{epsilon} \neq 0$ 
by (simp add: epsilon-def star-zero-def star-of-def star-n-eq-iff
        del: star-of-zero)

```

```

lemma hypreal-epsilon-inverse-omega:  $\text{epsilon} = \text{inverse}(\text{omega})$ 
by (simp add: epsilon-def omega-def star-n-inverse)

```

```

lemma hypreal-epsilon-gt-zero:  $0 < \text{epsilon}$ 
by (simp add: hypreal-epsilon-inverse-omega)

```

7.8 Absolute Value Function for the Hyperreals

```

lemma hrabs-add-less:
  [|  $\text{abs } x < r; \text{abs } y < s$  |] ==>  $\text{abs}(x+y) < r + (s::\text{hypreal})$ 
by (simp add: abs-if split: split-if-asm)

```

```

lemma hrabs-less-gt-zero:  $\text{abs } x < r ==> (0::\text{hypreal}) < r$ 
by (blast intro!: order-le-less-trans abs-ge-zero)

```

```

lemma hrabs-disj:  $\text{abs } x = (x::'a::\text{abs-if}) \mid \text{abs } x = -x$ 
by (simp add: abs-if)

```

```

lemma hrabs-add-lemma-disj:  $(y::\text{hypreal}) + -x + (y + -z) = \text{abs}(x + -z)$ 
==>  $y = z \mid x = y$ 

```

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

7.9 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)

lemma *hypreal-of-nat-eq*:
 hypreal-of-nat (n::nat) = hypreal-of-real (real n)
 by (simp add: real-of-nat-def)

lemma *hypreal-of-nat*:
 hypreal-of-nat m = star-n (%n. real m)
apply (fold star-of-def)
apply (simp add: real-of-nat-def)
done

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-neg-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) =
 ⟨⟨ fn - => fn ss => fn ct => Lin-Arith.simproc ss (term-of ct) ⟩⟩

7.10 Exponentials on the Hyperreals

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

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

lemma *hrealpow-two*: (r::hypreal) ^ Suc (Suc 0) = r * r

by *simp*

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

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

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

lemma *hypreal-add-nonneg-eq-0-iff*:
 $[\![\ 0 \leq x; 0 \leq y \]\!] \implies (x+y = 0) = (x = 0 \ \& \ y = (0::\text{hypreal}))$
 by *arith*

FIXME: DELETE THESE

lemma *hypreal-three-squares-add-zero-iff*:
 $(x*x + y*y + z*z = 0) = (x = 0 \ \& \ y = 0 \ \& \ z = (0::\text{hypreal}))$
apply (*simp only: zero-le-square add-nonneg-nonneg hypreal-add-nonneg-eq-0-iff, auto*)
done

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

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

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

lemma *two-hrealpow-gt* [*simp*]: *hypreal-of-nat* $n < 2 \wedge n$
apply (*induct n*)
apply (*auto simp add: left-distrib*)
apply (*cut-tac n = n in two-hrealpow-ge-one, arith*)
done

lemma *hrealpow*:
 $\text{star-n } X \wedge m = \text{star-n } (\%n. (X \text{ n}::\text{real}) \wedge m)$
apply (*induct-tac m*)
apply (*auto simp add: star-n-one-num star-n-mult power-0*)
done

lemma *hrealpow-sum-square-expand*:

$$(x + (y::\text{hypreal})) \wedge \text{Suc} (\text{Suc } 0) =$$

$$x \wedge \text{Suc} (\text{Suc } 0) + y \wedge \text{Suc} (\text{Suc } 0) + (\text{hypreal-of-nat} (\text{Suc} (\text{Suc } 0))) * x * y$$

by (*simp add: right-distrib left-distrib*)

lemma *power-hypreal-of-real-numeral*:

$$(\text{numeral } v :: \text{hypreal}) \wedge n = \text{hypreal-of-real} ((\text{numeral } v) \wedge n)$$

by *simp*

declare *power-hypreal-of-real-numeral* [*of - numeral w, simp*] **for** *w*

lemma *power-hypreal-of-real-neg-numeral*:

$$(\text{neg-numeral } v :: \text{hypreal}) \wedge n = \text{hypreal-of-real} ((\text{neg-numeral } v) \wedge n)$$

by *simp*

declare *power-hypreal-of-real-neg-numeral* [*of - numeral w, simp*] **for** *w*

7.11 Powers with Hypernatural Exponents

definition *pow* :: [*'a::power star, nat star*] \Rightarrow *'a star* (**infixr** *pow* 80) **where**

$$\text{hyperpow-def} [\text{transfer-unfold}]: R \text{ pow } N = (*f2* \text{ op } \wedge) R N$$

lemma *Standard-hyperpow* [*simp*]:

$$[r \in \text{Standard}; n \in \text{Standard}] \Longrightarrow r \text{ pow } n \in \text{Standard}$$

unfolding *hyperpow-def* **by** *simp*

lemma *hyperpow*: *star-n X pow star-n Y = star-n* ($\%n. X n \wedge Y n$)

by (*simp add: hyperpow-def starfun2-star-n*)

lemma *hyperpow-zero* [*simp*]:

$$\bigwedge n. (0::'a::\{\text{power, semiring-0}\} \text{ star}) \text{ pow } (n + (1::\text{hypnat})) = 0$$

by *transfer simp*

lemma *hyperpow-not-zero*:

$$\bigwedge r n. r \neq (0::'a::\{\text{field}\} \text{ star}) \Longrightarrow r \text{ pow } n \neq 0$$

by *transfer (rule field-power-not-zero)*

lemma *hyperpow-inverse*:

$$\bigwedge r n. r \neq (0::'a::\{\text{field-inverse-zero}\} \text{ star})$$

$$\Longrightarrow \text{inverse} (r \text{ pow } n) = (\text{inverse } r) \text{ pow } n$$

by *transfer (rule power-inverse)*

lemma *hyperpow-hrabs*:

$$\bigwedge r n. \text{abs} (r::'a::\{\text{linordered-idom}\} \text{ star}) \text{ pow } n = \text{abs} (r \text{ pow } n)$$

by *transfer (rule power-abs [symmetric])*

lemma *hyperpow-add*:

$$\bigwedge r n m. (r::'a::\{\text{monoid-mult}\} \text{ star}) \text{ pow } (n + m) = (r \text{ pow } n) * (r \text{ pow } m)$$

by *transfer (rule power-add)*

lemma *hyperpow-one* [simp]:

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

lemma *hyperpow-two*:

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

lemma *hyperpow-gt-zero*:

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

lemma *hyperpow-ge-zero*:

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

lemma *hyperpow-le*:

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

lemma *hyperpow-eq-one* [simp]:

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

lemma *hrabs-hyperpow-minus-one* [simp]:

$\bigwedge n. \text{abs}(-1 \text{ pow } n) = (1::'a::\{\text{linordered-idom}\} \text{ star})$
by *transfer* (rule *abs-power-minus-one*)

lemma *hyperpow-mult*:

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

lemma *hyperpow-two-le* [simp]:

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

lemma *hrabs-hyperpow-two* [simp]:

$\text{abs}(x \text{ pow } 2) =$
 $(x::'a::\{\text{monoid-mult, linordered-ring-strict}\} \text{ star}) \text{ pow } 2$
by (*simp only: abs-of-nonneg hyperpow-two-le*)

lemma *hyperpow-two-hrabs* [simp]:

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

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

lemma *hypreal-mult-less-mono*:

$\llbracket u < v; x < y; (0::\text{hypreal}) < v; 0 < x \rrbracket \implies u * x < v * y$
by (*simp add: mult-strict-mono order-less-imp-le*)

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

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

lemma *two-hyperpow-ge-one* [*simp*]: $(1::\text{hypreal}) \leq 2 \text{ pow } n$
apply (*rule-tac y = 1 pow n in order-trans*)
apply (*rule-tac [2] hyperpow-le, auto*)
done

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

lemma *hyperpow-less-le*:
 $\llbracket r \ n \ N. \llbracket (0::\text{hypreal}) \leq r; r \leq 1; n < N \rrbracket \implies r \text{ pow } N \leq r \text{ pow } n$
by *transfer (rule power-decreasing [OF order-less-imp-le])*

lemma *hyperpow-SHNat-le*:
 $\llbracket 0 \leq r; r \leq (1::\text{hypreal}); N \in \text{HNatInfinite} \rrbracket$
 $\implies \text{ALL } n: \text{Nats. } r \text{ pow } N \leq r \text{ pow } n$
by (*auto intro!: hyperpow-less-le simp add: HNatInfinite-iff*)

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

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

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

lemma *hyperpow-le-le*:
 $\llbracket (0::\text{hypreal}) \leq r; r \leq 1; n \leq N \rrbracket \implies r \text{ pow } N \leq r \text{ pow } n$
apply (*drule order-le-less [of n, THEN iffD1]*)
apply (*auto intro: hyperpow-less-le*)
done

lemma *hyperpow-Suc-le-self2*:
 $\llbracket (0::\text{hypreal}) \leq r; r < 1 \rrbracket \implies r \text{ pow } (n + (1::\text{hypnat})) \leq r$

```

apply (drule-tac  $n = (1::\text{hypnat})$ ) in hyperpow-le-le
apply auto
done

```

```

lemma hyperpow-hypnat-of-nat:  $\bigwedge x. x \text{ pow hypnat-of-nat } n = x \wedge n$ 
by transfer (rule refl)

```

```

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

```

```

end

```

8 NSA: Infinite Numbers, Infinitesimals, Infinitely Close Relation

```

theory NSA
imports HyperDef RComplete
begin

```

```

definition
 $hnorm :: 'a::\text{real-normed-vector star} \Rightarrow \text{real star}$  where
 $[\text{transfer-unfold}]: hnorm = *f* norm$ 

```

```

definition
 $Infinitesimal :: ('a::\text{real-normed-vector}) \text{ star set}$  where
 $Infinitesimal = \{x. \forall r \in \text{Reals}. 0 < r \longrightarrow hnorm x < r\}$ 

```

```

definition
 $HFinite :: ('a::\text{real-normed-vector}) \text{ star set}$  where
 $HFinite = \{x. \exists r \in \text{Reals}. hnorm x < r\}$ 

```

```

definition
 $HInfinite :: ('a::\text{real-normed-vector}) \text{ star set}$  where
 $HInfinite = \{x. \forall r \in \text{Reals}. r < hnorm x\}$ 

```

```

definition
 $approx :: ['a::\text{real-normed-vector star}, 'a \text{ star}] \Rightarrow \text{bool}$  (infixl  $@=$  50) where
— the ‘infinitely close’ relation
 $(x @= y) = ((x - y) \in Infinitesimal)$ 

```

```

definition
 $st :: \text{hypreal} \Rightarrow \text{hypreal}$  where
— the standard part of a hyperreal
 $st = (\%x. @r. x \in HFinite \ \& \ r \in \text{Reals} \ \& \ r @= x)$ 

```

definition

monad $:: 'a::\text{real-normed-vector star} \Rightarrow 'a \text{ star set}$ **where**
monad $x = \{y. x @ = y\}$

definition

galaxy $:: 'a::\text{real-normed-vector star} \Rightarrow 'a \text{ star set}$ **where**
galaxy $x = \{y. (x + -y) \in \text{HFinite}\}$

notation (*xsymbols*)

approx (**infixl** ≈ 50)

notation (*HTML output*)

approx (**infixl** ≈ 50)

lemma *SReal-def*: $\text{Reals} == \{x. \exists r. x = \text{hyreal-of-real } r\}$
by (*simp add: Reals-def image-def*)

8.1 Nonstandard Extension of the Norm Function

definition

scaleHR $:: \text{real star} \Rightarrow 'a \text{ star} \Rightarrow 'a::\text{real-normed-vector star}$ **where**
 $[\text{transfer-unfold}]: \text{scaleHR} = \text{starfun2 } \text{scaleR}$

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

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

lemma *hnorm-ge-zero* [*simp*]:

$\bigwedge x::'a::\text{real-normed-vector star}. 0 \leq \text{hnorm } x$
by *transfer (rule norm-ge-zero)*

lemma *hnorm-eq-zero* [*simp*]:

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

lemma *hnorm-triangle-ineq*:

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

lemma *hnorm-triangle-ineq3*:

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

lemma *hnorm-scaleR*:

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

lemma *hnorm-scaleHR*:

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

lemma *hnorm-mult-ineq*:

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

lemma *hnorm-mult*:

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

lemma *hnorm-hyperpow*:

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

lemma *hnorm-one [simp]*:

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

lemma *hnorm-zero [simp]*:

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

lemma *zero-less-hnorm-iff [simp]*:

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

lemma *hnorm-minus-cancel [simp]*:

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

lemma *hnorm-minus-commute*:

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

lemma *hnorm-triangle-ineq2*:

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

lemma *hnorm-triangle-ineq4*:

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

lemma *abs-hnorm-cancel [simp]*:

$\bigwedge a :: 'a :: \text{real-normed-vector star}. |hnorm a| = hnorm a$

by transfer (rule abs-norm-cancel)

lemma *hnorm-of-hypreal* [simp]:

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

lemma *nonzero-hnorm-inverse*:

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

lemma *hnorm-inverse*:

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

lemma *hnorm-divide*:

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

lemma *hypreal-hnorm-def* [simp]:

$\bigwedge r::\text{hypreal}. \text{hnorm } r = |r|$
 by transfer (rule real-norm-def)

lemma *hnorm-add-less*:

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

lemma *hnorm-mult-less*:

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

lemma *hnorm-scaleHR-less*:

$\llbracket |x| < r; \text{hnorm } y < s \rrbracket \implies \text{hnorm (scaleHR } x y) < r * s$
 apply (simp only: hnorm-scaleHR)
 apply (simp add: mult-strict-mono')
 done

8.2 Closure Laws for the Standard Reals

lemma *Reals-minus-iff* [simp]: $(-x \in \text{Reals}) = (x \in \text{Reals})$

apply auto
 apply (drule Reals-minus, auto)
 done

lemma *Reals-add-cancel*: $\llbracket x + y \in \text{Reals}; y \in \text{Reals} \rrbracket \implies x \in \text{Reals}$

by (*drule* (1) *Reals-diff*, *simp*)

lemma *SReal-hrabs*: $(x::\text{hypreal}) \in \text{Reals} \implies \text{abs } x \in \text{Reals}$
by (*simp add: Reals-eq-Standard*)

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

lemma *SReal-divide-numeral*: $r \in \text{Reals} \implies r / (\text{numeral } w::\text{hypreal}) \in \text{Reals}$
by *simp*

epsilon is not in Reals because it is an infinitesimal

lemma *SReal-epsilon-not-mem*: $\text{epsilon} \notin \text{Reals}$
apply (*simp add: SReal-def*)
apply (*auto simp add: hypreal-of-real-not-eq-epsilon [THEN not-sym]*)
done

lemma *SReal-omega-not-mem*: $\text{omega} \notin \text{Reals}$
apply (*simp add: SReal-def*)
apply (*auto simp add: hypreal-of-real-not-eq-omega [THEN not-sym]*)
done

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

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

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

lemma *inv-hypreal-of-real-image*: $\text{inv hypreal-of-real } ` \text{Reals} = \text{UNIV}$
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*:
 $[\exists x. x \in P; P \subseteq \text{Reals}] \implies \exists Q. P = \text{hypreal-of-real } ` Q$
by (*simp add: SReal-def image-def, blast*)

lemma *SReal-dense*:
 $[(x::\text{hypreal}) \in \text{Reals}; y \in \text{Reals}; x < y] \implies \exists r \in \text{Reals}. x < r \ \& \ r < y$
apply (*auto simp add: SReal-def*)
apply (*drule dense, auto*)
done

Completeness of Reals, but both lemmas are unused.

lemma *SReal-sup-lemma*:
 $P \subseteq \text{Reals} \implies ((\exists x \in P. y < x) =$

($\exists X. \text{hypreal-of-real } X \in P \ \& \ y < \text{hypreal-of-real } X$)
by (*blast dest!*: *SReal-iff* [*THEN iffD1*])

lemma *SReal-sup-lemma2*:

$[[P \subseteq \text{Reals}; \exists x. x \in P; \exists y \in \text{Reals}. \forall x \in P. x < y]]$
 $==> (\exists X. X \in \{w. \text{hypreal-of-real } w \in P\}) \ \&$
 $(\exists Y. \forall X \in \{w. \text{hypreal-of-real } w \in P\}. X < Y)$

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

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

lemma *HFinite-add*: $[[x \in \text{HFinite}; y \in \text{HFinite}]] ==> (x+y) \in \text{HFinite}$
apply (*simp add: HFinite-def*)
apply (*blast intro!*: *Reals-add hnorm-add-less*)
done

lemma *HFinite-mult*:

fixes $x \ y :: 'a::\text{real-normed-algebra star}$
shows $[[x \in \text{HFinite}; y \in \text{HFinite}]] ==> x*y \in \text{HFinite}$
apply (*simp add: HFinite-def*)
apply (*blast intro!*: *Reals-mult hnorm-mult-less*)
done

lemma *HFinite-scaleHR*:

$[[x \in \text{HFinite}; y \in \text{HFinite}]] ==> \text{scaleHR } x \ y \in \text{HFinite}$
apply (*simp add: HFinite-def*)
apply (*blast intro!*: *Reals-mult hnorm-scaleHR-less*)
done

lemma *HFinite-minus-iff*: $(-x \in \text{HFinite}) = (x \in \text{HFinite})$
by (*simp add: HFinite-def*)

lemma *HFinite-star-of* [*simp*]: $\text{star-of } x \in \text{HFinite}$

apply (*simp add: HFinite-def*)
apply (*rule-tac* $x=\text{star-of } (\text{norm } x) + 1$ **in** $be xI$)
apply (*transfer*, *simp*)
apply (*blast intro: Reals-add SReal-hypreal-of-real Reals-1*)
done

lemma *SReal-subset-HFinite*: $(\text{Reals}::\text{hypreal set}) \subseteq \text{HFinite}$
by (*auto simp add: SReal-def*)

lemma *HFiniteD*: $x \in \text{HFinite} ==> \exists t \in \text{Reals}. \text{hnorm } x < t$
by (*simp add: HFinite-def*)

lemma *HFinite-hrabs-iff* [iff]: $(\text{abs } (x::\text{hypreal}) \in \text{HFinite}) = (x \in \text{HFinite})$
by (*simp add: HFinite-def*)

lemma *HFinite-hnorm-iff* [iff]:
 $(\text{hnorm } (x::\text{hypreal}) \in \text{HFinite}) = (x \in \text{HFinite})$
by (*simp add: HFinite-def*)

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

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

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

lemma *hrealpow-HFinite*:
fixes $x :: 'a::\{\text{real-normed-algebra, monoid-mult}\}$ *star*
shows $x \in \text{HFinite} \implies x \wedge n \in \text{HFinite}$
apply (*induct n*)
apply (*auto simp add: power-Suc intro: HFinite-mult*)
done

lemma *HFinite-bounded*:
 $[(x::\text{hypreal}) \in \text{HFinite}; y \leq x; 0 \leq y] \implies y \in \text{HFinite}$
apply (*cases x ≤ 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

8.4 Set of Infinitesimals is a Subring of the Hyperreals

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

lemma *InfinitesimalD*:
 $x \in \text{Infinitesimal} \implies \forall r \in \text{Reals}. 0 < r \dashrightarrow \text{hnorm } x < r$
by (*simp add: Infinitesimal-def*)

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

lemma *InfinesimalD2*:

$\llbracket x \in \text{Infinesimal}; 0 < r \rrbracket \implies \text{hnorm } x < \text{star-of } r$
by (*auto simp add: Infinesimal-def SReal-def*)

lemma *Infinesimal-zero [iff]*: $0 \in \text{Infinesimal}$

by (*simp add: Infinesimal-def*)

lemma *hypreal-sum-of-halves*: $x/(2::\text{hypreal}) + x/(2::\text{hypreal}) = x$

by *auto*

lemma *Infinesimal-add*:

$\llbracket x \in \text{Infinesimal}; y \in \text{Infinesimal} \rrbracket \implies (x+y) \in \text{Infinesimal}$
apply (*rule InfinesimalI*)
apply (*rule hypreal-sum-of-halves [THEN subst]*)
apply (*drule half-gt-zero*)
apply (*blast intro: hnorm-add-less SReal-divide-numeral dest: InfinesimalD*)
done

lemma *Infinesimal-minus-iff [simp]*: $(-x:\text{Infinesimal}) = (x:\text{Infinesimal})$

by (*simp add: Infinesimal-def*)

lemma *Infinesimal-hnorm-iff*:

$(\text{hnorm } x \in \text{Infinesimal}) = (x \in \text{Infinesimal})$
by (*simp add: Infinesimal-def*)

lemma *Infinesimal-hrabs-iff [iff]*:

$(\text{abs } (x::\text{hypreal}) \in \text{Infinesimal}) = (x \in \text{Infinesimal})$
by (*simp add: abs-if*)

lemma *Infinesimal-of-hypreal-iff [simp]*:

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

lemma *Infinesimal-diff*:

$\llbracket x \in \text{Infinesimal}; y \in \text{Infinesimal} \rrbracket \implies x-y \in \text{Infinesimal}$
by (*simp add: diff-minus Infinesimal-add*)

lemma *Infinesimal-mult*:

fixes $x y :: 'a::\text{real-normed-algebra star}$
shows $\llbracket x \in \text{Infinesimal}; y \in \text{Infinesimal} \rrbracket \implies (x * y) \in \text{Infinesimal}$
apply (*rule InfinesimalI*)
apply (*subgoal-tac hnorm (x * y) < 1 * r, simp only: mult-1*)
apply (*rule hnorm-mult-less*)
apply (*simp-all add: InfinesimalD*)
done

lemma *Infinesimal-HFinite-mult*:

fixes $x y :: 'a::\text{real-normed-algebra star}$

```

  shows [| x ∈ Infinitesimal; y ∈ HFinite |] ==> (x * y) ∈ Infinitesimal
  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 Reals-divide)
  apply assumption
  apply (simp only: divide-pos-pos)
  apply (erule order-le-less-trans [OF hnorm-ge-zero])
  done

```

lemma *Infinitesimal-HFinite-scaleHR*:

```

  [| x ∈ Infinitesimal; y ∈ HFinite |] ==> scaleHR x y ∈ 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 (simp only: divide-pos-pos)
  apply (erule order-le-less-trans [OF hnorm-ge-zero])
  done

```

lemma *Infinitesimal-HFinite-mult2*:

```

  fixes x y :: 'a::real-normed-algebra star
  shows [| x ∈ Infinitesimal; y ∈ HFinite |] ==> (y * x) ∈ Infinitesimal
  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 Reals-divide)
  apply (simp only: divide-pos-pos)
  apply (erule order-le-less-trans [OF hnorm-ge-zero])
  done

```

lemma *Infinitesimal-scaleR2*:

```

  x ∈ Infinitesimal ==> a *R x ∈ Infinitesimal
  apply (case-tac a = 0, simp)
  apply (rule InfinitesimalI)
  apply (drule InfinitesimalD)
  apply (drule-tac x=r / |star-of a| in bspec)
  apply (simp add: Reals-eq-Standard)

```

```

apply (simp add: divide-pos-pos)
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:
  fixes  $x :: 'a::\text{real-normed-div-algebra star}$ 
  shows  $x \in \text{HInfinite} \implies \text{inverse } x \in \text{Infinitesimal}$ 
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:  $(\bigwedge r. r \in \mathbb{R} \implies r < \text{hnorm } x) \implies x \in \text{HInfinite}$ 
by (simp add: HInfinite-def)

```

```

lemma HInfiniteD:  $\llbracket x \in \text{HInfinite}; r \in \mathbb{R} \rrbracket \implies r < \text{hnorm } x$ 
by (simp add: HInfinite-def)

```

```

lemma HInfinite-mult:
  fixes  $x y :: 'a::\text{real-normed-div-algebra star}$ 
  shows  $\llbracket x \in \text{HInfinite}; y \in \text{HInfinite} \rrbracket \implies (x*y) \in \text{HInfinite}$ 
apply (rule HInfiniteI, simp only: hnorm-mult)
apply (subgoal-tac  $r * 1 < \text{hnorm } x * \text{hnorm } y$ , 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:  $\llbracket r < x; (0::\text{hypreal}) \leq y \rrbracket \implies r < x+y$ 
by (auto dest: add-less-le-mono)

```

```

lemma HInfinite-add-ge-zero:
   $\llbracket (x::\text{hypreal}) \in \text{HInfinite}; 0 \leq y; 0 \leq x \rrbracket \implies (x + y): \text{HInfinite}$ 
by (auto intro!: hypreal-add-zero-less-le-mono
  simp add: abs-if add-commute add-nonneg-nonneg HInfinite-def)

```

```

lemma HInfinite-add-ge-zero2:
   $\llbracket (x::\text{hypreal}) \in \text{HInfinite}; 0 \leq y; 0 \leq x \rrbracket \implies (y + x): \text{HInfinite}$ 
by (auto intro!: HInfinite-add-ge-zero simp add: add-commute)

```

lemma *HInfinite-add-gt-zero*:

$\llbracket (x::\text{hypreal}) \in \text{HInfinite}; 0 < y; 0 < x \rrbracket \implies (x + y): \text{HInfinite}$
by (*blast intro: HInfinite-add-ge-zero order-less-imp-le*)

lemma *HInfinite-minus-iff*: $(-x \in \text{HInfinite}) = (x \in \text{HInfinite})$

by (*simp add: HInfinite-def*)

lemma *HInfinite-add-le-zero*:

$\llbracket (x::\text{hypreal}) \in \text{HInfinite}; y \leq 0; x \leq 0 \rrbracket \implies (x + y): \text{HInfinite}$
apply (*drule HInfinite-minus-iff [THEN iffD2]*)
apply (*rule HInfinite-minus-iff [THEN iffD1]*)
apply (*auto intro: HInfinite-add-ge-zero*)
done

lemma *HInfinite-add-lt-zero*:

$\llbracket (x::\text{hypreal}) \in \text{HInfinite}; y < 0; x < 0 \rrbracket \implies (x + y): \text{HInfinite}$
by (*blast intro: HInfinite-add-le-zero order-less-imp-le*)

lemma *HFinite-sum-squares*:

fixes $a b c :: 'a::\text{real-normed-algebra star}$
shows $\llbracket a: \text{HFinite}; b: \text{HFinite}; c: \text{HFinite} \rrbracket$
 $\implies a*a + b*b + c*c \in \text{HFinite}$
by (*auto intro: HFinite-mult HFinite-add*)

lemma *not-Infinitesimal-not-zero*: $x \notin \text{Infinitesimal} \implies x \neq 0$

by *auto*

lemma *not-Infinitesimal-not-zero2*: $x \in \text{HFinite} - \text{Infinitesimal} \implies x \neq 0$

by *auto*

lemma *HFinite-diff-Infinitesimal-hrabs*:

$(x::\text{hypreal}) \in \text{HFinite} - \text{Infinitesimal} \implies \text{abs } x \in \text{HFinite} - \text{Infinitesimal}$
by *blast*

lemma *hnorm-le-Infinitesimal*:

$\llbracket e \in \text{Infinitesimal}; \text{hnorm } x \leq e \rrbracket \implies x \in \text{Infinitesimal}$
by (*auto simp add: Infinitesimal-def abs-less-iff*)

lemma *hnorm-less-Infinitesimal*:

$\llbracket e \in \text{Infinitesimal}; \text{hnorm } x < e \rrbracket \implies x \in \text{Infinitesimal}$
by (*erule hnorm-le-Infinitesimal, erule order-less-imp-le*)

lemma *hrabs-le-Infinitesimal*:

$\llbracket e \in \text{Infinitesimal}; \text{abs } (x::\text{hypreal}) \leq e \rrbracket \implies x \in \text{Infinitesimal}$
by (*erule hnorm-le-Infinitesimal, simp*)

lemma *hrabs-less-Infinitesimal*:

$\llbracket e \in \text{Infinitesimal}; \text{abs } (x::\text{hypreal}) < e \rrbracket \implies x \in \text{Infinitesimal}$

by (erule hnorm-less-Infinitesimal, simp)

lemma *Infinitesimal-interval*:

$[[e \in \text{Infinitesimal}; e' \in \text{Infinitesimal}; e' < x ; x < e]]$
 $\implies (x::\text{hypreal}) \in \text{Infinitesimal}$

by (auto simp add: Infinitesimal-def abs-less-iff)

lemma *Infinitesimal-interval2*:

$[[e \in \text{Infinitesimal}; e' \in \text{Infinitesimal};$
 $e' \leq x ; x \leq e]]$ $\implies (x::\text{hypreal}) \in \text{Infinitesimal}$

by (auto intro: Infinitesimal-interval simp add: order-le-less)

lemma *lemma-Infinitesimal-hyperpow*:

$[[(x::\text{hypreal}) \in \text{Infinitesimal}; 0 < N]]$ $\implies \text{abs } (x \text{ pow } N) \leq \text{abs } x$

apply (unfold Infinitesimal-def)

apply (auto intro!: hyperpow-Suc-le-self2

simp add: hyperpow-hrabs [symmetric] hypnat-gt-zero-iff2 abs-ge-zero)

done

lemma *Infinitesimal-hyperpow*:

$[[(x::\text{hypreal}) \in \text{Infinitesimal}; 0 < N]]$ $\implies x \text{ pow } N \in \text{Infinitesimal}$

apply (rule hrabs-le-Infinitesimal)

apply (rule-tac [2] lemma-Infinitesimal-hyperpow, auto)

done

lemma *hrealpow-hyperpow-Infinitesimal-iff*:

$(x \wedge n \in \text{Infinitesimal}) = (x \text{ pow } (\text{hypnat-of-nat } n) \in \text{Infinitesimal})$

by (simp only: hyperpow-hypnat-of-nat)

lemma *Infinitesimal-hrealpow*:

$[[(x::\text{hypreal}) \in \text{Infinitesimal}; 0 < n]]$ $\implies x \wedge n \in \text{Infinitesimal}$

by (simp add: hrealpow-hyperpow-Infinitesimal-iff Infinitesimal-hyperpow)

lemma *not-Infinitesimal-mult*:

fixes $x y :: 'a::\text{real-normed-div-algebra star}$

shows $[[x \notin \text{Infinitesimal}; y \notin \text{Infinitesimal}]] \implies (x*y) \notin \text{Infinitesimal}$

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 (drule mp, blast intro: mult-pos-pos)

apply (drule-tac $c = s$ and $d = \text{hnorm } y$ and $a = r$ and $b = \text{hnorm } x$ in mult-mono)

apply (simp-all (no-asm-simp))

done

lemma *Infinitesimal-mult-disj*:

fixes $x y :: 'a::\text{real-normed-div-algebra star}$

shows $x*y \in \text{Infinitesimal} \implies x \in \text{Infinitesimal} \mid y \in \text{Infinitesimal}$
apply (rule ccontr)
apply (drule de-Morgan-disj [THEN iffD1])
apply (fast dest: not-Infinitesimal-mult)
done

lemma *HFinite-Infinitesimal-not-zero*: $x \in \text{HFinite} - \text{Infinitesimal} \implies x \neq 0$
by blast

lemma *HFinite-Infinitesimal-diff-mult*:
fixes $x\ y :: 'a::\text{real-normed-div-algebra star}$
shows $\llbracket x \in \text{HFinite} - \text{Infinitesimal};$
 $y \in \text{HFinite} - \text{Infinitesimal}$
 $\rrbracket \implies (x*y) \in \text{HFinite} - \text{Infinitesimal}$
apply clarify
apply (blast dest: HFinite-mult not-Infinitesimal-mult)
done

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

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

lemma *Infinitesimal-star-of-mult2*:
fixes $x :: 'a::\text{real-normed-algebra star}$
shows $x \in \text{Infinitesimal} \implies \text{star-of } r * x \in \text{Infinitesimal}$
by (erule HFinite-star-of [THEN [2] Infinitesimal-HFinite-mult2])

8.5 The Infinitely Close Relation

lemma *mem-infmal-iff*: $(x \in \text{Infinitesimal}) = (x @= 0)$
by (simp add: Infinitesimal-def approx-def)

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

lemma *approx-minus-iff2*: $(x @= y) = (-y + x @= 0)$
by (simp add: approx-def diff-minus add-commute)

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

lemma *hypreal-minus-distrib1*: $-(y + -(x::'a::\text{ab-group-add})) = x + -y$

by (simp add: add-commute)

lemma *approx-sym*: $x \text{ @} = y \implies y \text{ @} = x$
 apply (simp add: approx-def)
 apply (drule *Infinesimal-minus-iff* [THEN *iffD2*])
 apply simp
 done

lemma *approx-trans*: $[x \text{ @} = y; y \text{ @} = z] \implies x \text{ @} = z$
 apply (simp add: approx-def)
 apply (drule (1) *Infinesimal-add*)
 apply (simp add: diff-minus)
 done

lemma *approx-trans2*: $[r \text{ @} = x; s \text{ @} = x] \implies r \text{ @} = s$
 by (blast intro: approx-sym approx-trans)

lemma *approx-trans3*: $[x \text{ @} = r; x \text{ @} = s] \implies r \text{ @} = s$
 by (blast intro: approx-sym approx-trans)

lemma *approx-reorient*: $(x \text{ @} = y) = (y \text{ @} = x)$
 by (blast intro: approx-sym)

simproc-setup *approx-reorient-simproc*
 ($0 \text{ @} = x \mid 1 \text{ @} = y \mid \text{numeral } w \text{ @} = z \mid \text{neg-numeral } w \text{ @} = r$) =
 ⟨⟨
 let val rule = @{thm *approx-reorient*} RS eq-reflection
 fun proc phi ss ct = case 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 *Infinesimal-approx-minus*: $(x - y \in \text{Infinesimal}) = (x \text{ @} = y)$
 by (simp add: approx-minus-iff [symmetric] mem-infmal-iff)

lemma *approx-monad-iff*: $(x \text{ @} = y) = (\text{monad}(x) = \text{monad}(y))$
 apply (simp add: monad-def)
 apply (auto dest: approx-sym elim!: approx-trans equalityCE)
 done

lemma *Infinesimal-approx*:
 $[x \in \text{Infinesimal}; y \in \text{Infinesimal}] \implies x \text{ @} = y$
 apply (simp add: mem-infmal-iff)
 apply (blast intro: approx-trans approx-sym)
 done

lemma *approx-add*: $[| a \text{ @} = b; c \text{ @} = d |] \implies a+c \text{ @} = b+d$
proof (*unfold approx-def*)
assume *inf*: $a - b \in \text{Infinitesimal}$ $c - d \in \text{Infinitesimal}$
have $a + c - (b + d) = (a - b) + (c - d)$ **by** *simp*
also have $\dots \in \text{Infinitesimal}$ **using** *inf* **by** (*rule Infinitesimal-add*)
finally show $a + c - (b + d) \in \text{Infinitesimal}$.
qed

lemma *approx-minus*: $a \text{ @} = b \implies -a \text{ @} = -b$
apply (*rule approx-minus-iff [THEN iffD2, THEN approx-sym]*)
apply (*drule approx-minus-iff [THEN iffD1]*)
apply (*simp add: add-commute diff-minus*)
done

lemma *approx-minus2*: $-a \text{ @} = -b \implies a \text{ @} = b$
by (*auto dest: approx-minus*)

lemma *approx-minus-cancel* [*simp*]: $(-a \text{ @} = -b) = (a \text{ @} = b)$
by (*blast intro: approx-minus approx-minus2*)

lemma *approx-add-minus*: $[| a \text{ @} = b; c \text{ @} = d |] \implies a + -c \text{ @} = b + -d$
by (*blast intro!: approx-add approx-minus*)

lemma *approx-diff*: $[| a \text{ @} = b; c \text{ @} = d |] \implies a - c \text{ @} = b - d$
by (*simp only: diff-minus approx-add approx-minus*)

lemma *approx-mult1*:
fixes $a b c :: 'a::\text{real-normed-algebra star}$
shows $[| a \text{ @} = b; c \in \text{HFinite} |] \implies a*c \text{ @} = b*c$
by (*simp add: approx-def Infinitesimal-HFinite-mult*
left-diff-distrib [symmetric])

lemma *approx-mult2*:
fixes $a b c :: 'a::\text{real-normed-algebra star}$
shows $[| a \text{ @} = b; c \in \text{HFinite} |] \implies c*a \text{ @} = c*b$
by (*simp add: approx-def Infinitesimal-HFinite-mult2*
right-diff-distrib [symmetric])

lemma *approx-mult-subst*:
fixes $u v x y :: 'a::\text{real-normed-algebra star}$
shows $[| u \text{ @} = v*x; x \text{ @} = y; v \in \text{HFinite} |] \implies u \text{ @} = v*y$
by (*blast intro: approx-mult2 approx-trans*)

lemma *approx-mult-subst2*:
fixes $u v x y :: 'a::\text{real-normed-algebra star}$
shows $[| u \text{ @} = x*v; x \text{ @} = y; v \in \text{HFinite} |] \implies u \text{ @} = y*v$
by (*blast intro: approx-mult1 approx-trans*)

lemma *approx-mult-subst-star-of*:

fixes $u\ x\ y :: 'a::\text{real-normed-algebra star}$
shows $[\![\ u\ @= x*\text{star-of } v; x\ @= y\ \]\] ==> u\ @= y*\text{star-of } v$
by (*auto intro: approx-mult-subst2*)

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

lemma *Infinitesimal-minus-approx*: $x \in \text{Infinitesimal} ==> -x\ @= x$
by (*blast intro: Infinitesimal-minus-iff [THEN iffD2]*
mem-infmal-iff [THEN iffD1] approx-trans2)

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

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

lemma *Infinitesimal-add-approx*: $[\![\ y \in \text{Infinitesimal}; x + y = z\ \]\] ==> x\ @= 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*: $y \in \text{Infinitesimal} ==> x\ @= x + y$
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*: $y \in \text{Infinitesimal} ==> x\ @= y + x$
by (*auto dest: Infinitesimal-add-approx-self simp add: add-commute*)

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

lemma *Infinitesimal-add-cancel*: $[\![\ y \in \text{Infinitesimal}; x+y\ @= z\ \]\] ==> x\ @= 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 \text{Infinitesimal}; x\ @= z + y\ \]\] ==> x\ @= 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*: $d + b\ @= d + c ==> b\ @= c$

```

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

```

```

lemma approx-add-right-cancel:  $b + d @= c + d ==> b @= c$ 
apply (rule approx-add-left-cancel)
apply (simp add: add-commute)
done

```

```

lemma approx-add-mono1:  $b @= c ==> d + b @= d + c$ 
apply (rule approx-minus-iff [THEN iffD2])
apply (simp add: approx-minus-iff [symmetric] add-ac)
done

```

```

lemma approx-add-mono2:  $b @= c ==> b + a @= c + a$ 
by (simp add: add-commute approx-add-mono1)

```

```

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

```

```

lemma approx-add-right-iff [simp]:  $(b + a @= c + a) = (b @= c)$ 
by (simp add: add-commute)

```

```

lemma approx-HFinite:  $[| x \in HFinite; x @= y |] ==> y \in HFinite$ 
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:  $x @= \text{star-of } D ==> x \in HFinite$ 
by (rule approx-sym [THEN [2] approx-HFinite], auto)

```

```

lemma approx-mult-HFinite:
  fixes  $a b c d :: 'a::\text{real-normed-algebra star}$ 
  shows  $[| a @= b; c @= d; b: HFinite; d: HFinite |] ==> a * c @= b * d$ 
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:
 $\bigwedge a b x. \text{scaleHR } (a - b) x = \text{scaleHR } a x - \text{scaleHR } b x$ 
by transfer (rule scaleR-left-diff-distrib)

```

```

lemma approx-scaleR1:
 $[| a @= \text{star-of } b; c: HFinite |] ==> \text{scaleHR } a c @= b *_R c$ 

```

```

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 @= b ==> c *R a @= c *R b
by (simp add: approx-def Infinitesimal-scaleR2
      scaleR-right-diff-distrib [symmetric])

```

```

lemma approx-scaleR-HFinite:
  [| a @= star-of b; c @= d; d: HFinite |] ==> scaleHR a c @= 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:
  fixes a c :: 'a::real-normed-algebra star
  shows [| a @= star-of b; c @= star-of d |]
    ==> a*c @= star-of b*star-of d
by (blast intro!: approx-mult-HFinite approx-star-of-HFinite HFinite-star-of)

```

```

lemma approx-SReal-mult-cancel-zero:
  [| (a::hypreal) ∈ Reals; a ≠ 0; a*x @= 0 |] ==> x @= 0
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::hypreal) ∈ Reals; x @= 0 |] ==> x*a @= 0
by (auto dest: SReal-subset-HFinite [THEN subsetD] approx-mult1)

```

```

lemma approx-mult-SReal2: [| (a::hypreal) ∈ Reals; x @= 0 |] ==> a*x @= 0
by (auto dest: SReal-subset-HFinite [THEN subsetD] approx-mult2)

```

```

lemma approx-mult-SReal-zero-cancel-iff [simp]:
  [| (a::hypreal) ∈ Reals; a ≠ 0 |] ==> (a*x @= 0) = (x @= 0)
by (blast intro: approx-SReal-mult-cancel-zero approx-mult-SReal2)

```

```

lemma approx-SReal-mult-cancel:
  [| (a::hypreal) ∈ Reals; a ≠ 0; a* w @= a*z |] ==> w @= z
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::hypreal) ∈ Reals; a ≠ 0 |] ==> (a* w @= a*z) = (w @= z)

```

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

lemma approx-le-bound: [| (z::hypreal) \leq f; f @= g; g \leq z |] ==> f @= z
apply (simp add: beX-Infinitesimal-iff2 [symmetric], auto)
apply (rule-tac x = g+y-z in beXI)
apply (simp (no-asm))
apply (rule Infinitesimal-interval2)
apply (rule-tac [2] Infinitesimal-zero, auto)
done

lemma approx-hnorm:
fixes x y :: 'a::real-normed-vector star
shows x \approx y \implies hnorm x \approx hnorm y
proof (unfold approx-def)
assume x - y \in Infinitesimal
hence 1: hnorm (x - y) \in Infinitesimal
by (simp only: Infinitesimal-hnorm-iff)
moreover have 2: (0::real star) \in Infinitesimal
by (rule Infinitesimal-zero)
moreover have 3: 0 \leq |hnorm x - hnorm y|
by (rule abs-ge-zero)
moreover have 4: |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)
thus hnorm x - hnorm y \in Infinitesimal
by (simp only: Infinitesimal-hrabs-iff)
qed

8.6 Zero is the Only Infinitesimal that is also a Real

lemma Infinitesimal-less-SReal:
[| (x::hypreal) \in Reals; y \in Infinitesimal; 0 < x |] ==> y < x
apply (simp add: Infinitesimal-def)
apply (rule abs-ge-self [THEN order-le-less-trans], auto)
done

lemma Infinitesimal-less-SReal2:
(y::hypreal) \in Infinitesimal ==> $\forall r \in$ Reals. 0 < r \implies y < r
by (blast intro: Infinitesimal-less-SReal)

lemma SReal-not-Infinitesimal:
[| 0 < y; (y::hypreal) \in Reals |] ==> y \notin Infinitesimal
apply (simp add: Infinitesimal-def)
apply (auto simp add: abs-iff)
done

lemma SReal-minus-not-Infinitesimal:

```

  [|  $y < 0$ ;  $(y::\text{hypreal}) \in \text{Reals}$  |] ==>  $y \notin \text{Infinitesimal}$ 
apply (subst Infinitesimal-minus-iff [symmetric])
apply (rule SReal-not-Infinitesimal, auto)
done

```

```

lemma SReal-Int-Infinitesimal-zero:  $\text{Reals Int Infinitesimal} = \{0::\text{hypreal}\}$ 
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:
  [|  $(x::\text{hypreal}) \in \text{Reals}$ ;  $x \in \text{Infinitesimal}$  |] ==>  $x = 0$ 
by (cut-tac SReal-Int-Infinitesimal-zero, blast)

```

```

lemma SReal-HFfinite-diff-Infinitesimal:
  [|  $(x::\text{hypreal}) \in \text{Reals}$ ;  $x \neq 0$  |] ==>  $x \in \text{HFfinite} - \text{Infinitesimal}$ 
by (auto dest: SReal-Infinitesimal-zero SReal-subset-HFfinite [THEN subsetD])

```

```

lemma hypreal-of-real-HFfinite-diff-Infinitesimal:
   $\text{hypreal-of-real } x \neq 0 ==> \text{hypreal-of-real } x \in \text{HFfinite} - \text{Infinitesimal}$ 
by (rule SReal-HFfinite-diff-Infinitesimal, auto)

```

```

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

```

```

lemma star-of-HFfinite-diff-Infinitesimal:
   $x \neq 0 ==> \text{star-of } x \in \text{HFfinite} - \text{Infinitesimal}$ 
by simp

```

```

lemma numeral-not-Infinitesimal [simp]:
   $\text{numeral } w \neq (0::\text{hypreal}) ==> (\text{numeral } w :: \text{hypreal}) \notin \text{Infinitesimal}$ 
by (fast dest: Reals-numeral [THEN SReal-Infinitesimal-zero])

```

```

lemma one-not-Infinitesimal [simp]:
  ( $1::'a::\{\text{real-normed-vector, zero-neq-one}\}$  star)  $\notin \text{Infinitesimal}$ 
apply (simp only: star-one-def star-of-Infinitesimal-iff-0)
apply simp
done

```

```

lemma approx-SReal-not-zero:
  [|  $(y::\text{hypreal}) \in \text{Reals}$ ;  $x @= y$ ;  $y \neq 0$  |] ==>  $x \neq 0$ 

```

```

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:
  [|  $x @= y$ ;  $y \in \text{HFinite} - \text{Infinitesimal}$  |]
  ==>  $x \in \text{HFinite} - \text{Infinitesimal}$ 
apply (auto intro: approx-sym [THEN [2] approx-HFinite]
simp add: mem-infmal-iff)
apply (drule approx-trans3, assumption)
apply (blast dest: approx-sym)
done

```

```

lemma Infinitesimal-ratio:
  fixes  $x\ y :: 'a::\{\text{real-normed-div-algebra,field}\}$  star
  shows [|  $y \neq 0$ ;  $y \in \text{Infinitesimal}$ ;  $x/y \in \text{HFinite}$  |]
  ==>  $x \in \text{Infinitesimal}$ 
apply (drule Infinitesimal-HFinite-mult2, assumption)
apply (simp add: divide-inverse mult-assoc)
done

```

```

lemma Infinitesimal-SReal-divide:
  [| ( $x::\text{hypreal}$ )  $\in \text{Infinitesimal}$ ;  $y \in \text{Reals}$  |] ==>  $x/y \in \text{Infinitesimal}$ 
apply (simp add: divide-inverse)
apply (auto intro!: Infinitesimal-HFinite-mult
dest!: Reals-inverse [THEN SReal-subset-HFinite [THEN subsetD]])
done

```

8.7 Uniqueness: Two Infinitely Close Reals are Equal

```

lemma star-of-approx-iff [simp]: (star-of  $x @=$  star-of  $y$ ) = ( $x = y$ )
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:
  [| ( $x::\text{hypreal}$ )  $\in \text{Reals}$ ;  $y \in \text{Reals}$  |] ==> ( $x @= y$ ) = ( $x = y$ )
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]:
 $(\text{numeral } v \text{ @=} (\text{numeral } w \text{ :: 'a}::\{\text{numeral,real-normed-vector}\} \text{ star})) =$
 $(\text{numeral } v = (\text{numeral } w \text{ :: 'a}))$
apply (*unfold star-numeral-def*)
apply (*rule star-of-approx-iff*)
done

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

lemma *star-of-approx-numeral-iff* [simp]:
 $(\text{star-of } k \text{ @=} \text{numeral } w) = (k = \text{numeral } w)$
by (*subst star-of-approx-iff [symmetric], auto*)

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

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

lemma *approx-unique-real*:
 $[| (r::\text{hypreal}) \in \text{Reals}; s \in \text{Reals}; r \text{ @=} x; s \text{ @=} x |] ==> r = s$
by (*blast intro: SReal-approx-iff [THEN iffD1] approx-trans2*)

8.8 Existence of Unique Real Infinitely Close

8.8.1 Lifting of the Ub and Lub Properties

lemma *hypreal-of-real-isUb-iff*:
 $(\text{isUb } (\text{Reals}) (\text{hypreal-of-real } 'Q) (\text{hypreal-of-real } Y)) =$
 $(\text{isUb } (\text{UNIV} \text{ :: real set}) Q Y)$
by (*simp add: isUb-def settle-def*)

lemma *hypreal-of-real-isLub1*:
 $\text{isLub } \text{Reals } (\text{hypreal-of-real } 'Q) (\text{hypreal-of-real } Y)$
 $==> \text{isLub } (\text{UNIV} \text{ :: real set}) Q Y$
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 :: real set) Q Y
 \implies *isLub Reals (hypreal-of-real ‘ Q) (hypreal-of-real Y)*

apply (*simp add: isLub-def leastP-def*)

apply (*auto simp add: hypreal-of-real-isUb-iff setge-def*)

apply (*frule-tac x2 = x in isUbD2a [THEN SReal-iff [THEN iffD1], THEN exE]*)

prefer 2 **apply** *assumption*

apply (*drule-tac x = xa in spec*)

apply (*auto simp add: hypreal-of-real-isUb-iff*)

done

lemma *hypreal-of-real-isLub-iff*:

(isLub Reals (hypreal-of-real ‘ Q) (hypreal-of-real Y)) =
(isLub (UNIV :: real set) Q Y)

by (*blast intro: hypreal-of-real-isLub1 hypreal-of-real-isLub2*)

lemma *lemma-isUb-hypreal-of-real*:

isUb Reals P Y \implies \exists Yo. isUb Reals P (hypreal-of-real Yo)

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

lemma *lemma-isLub-hypreal-of-real*:

isLub Reals P Y \implies \exists Yo. isLub Reals 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 Reals P (hypreal-of-real Yo) \implies \exists Y. isLub Reals P Y*

by (*auto simp add: isLub-def leastP-def isUb-def*)

lemma *SReal-complete*:

$[[P \subseteq \text{Reals}; \exists x. x \in P; \exists Y. \text{isUb Reals } P \ Y]]$

$\implies \exists t::\text{hypreal. isLub Reals } P \ t$

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

lemma *hypreal-isLub-unique*:

$[[\text{isLub } R \ S \ x; \text{isLub } R \ S \ y]] \implies x = (y::\text{hypreal})$

apply (*frule isLub-isUb*)

apply (*frule-tac x = y in isLub-isUb*)

apply (*blast intro!: order-antisym dest!: isLub-le-isUb*)

done

lemma *lemma-st-part-ub*:

```
(x::hypreal) ∈ HFinite ==> ∃ u. isUb Reals {s. s ∈ Reals & s < x} u
apply (drule HFiniteD, safe)
apply (rule exI, rule isUbI)
apply (auto intro: settleI isUbI simp add: abs-less-iff)
done
```

lemma *lemma-st-part-nonempty*:

```
(x::hypreal) ∈ HFinite ==> ∃ y. y ∈ {s. s ∈ Reals & s < x}
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-subset*: $\{s. s \in \text{Reals} \ \& \ s < x\} \subseteq \text{Reals}$

by *auto*

lemma *lemma-st-part-lub*:

```
(x::hypreal) ∈ HFinite ==> ∃ t. isLub Reals {s. s ∈ Reals & s < x} t
by (blast intro!: SReal-complete lemma-st-part-ub lemma-st-part-nonempty lemma-st-part-subset)
```

lemma *lemma-hypreal-le-left-cancel*: $((t::hypreal) + r \leq t) = (r \leq 0)$

```
apply safe
apply (drule-tac c = -t in add-left-mono)
apply (drule-tac [2] c = t in add-left-mono)
apply (auto simp add: add-assoc [symmetric])
done
```

lemma *lemma-st-part-le1*:

```
[| (x::hypreal) ∈ HFinite; isLub Reals {s. s ∈ Reals & s < x} t;
  r ∈ Reals; 0 < r |] ==> x ≤ t + r
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 settleD], auto)
done
```

lemma *hypreal-setle-less-trans*:

```
[| S *≤ (x::hypreal); x < y |] ==> S *≤ y
apply (simp add: settle-def)
apply (auto dest!: bspec order-le-less-trans intro: order-less-imp-le)
done
```

lemma *hypreal-gt-isUb*:

```
[| isUb R S (x::hypreal); x < y; y ∈ R |] ==> isUb R S y
apply (simp add: isUb-def)
apply (blast intro: hypreal-setle-less-trans)
done
```

lemma *lemma-st-part-gt-ub*:

$[[(x::hypreal) \in HFinite; x < y; y \in Reals]]$
 $\implies isUb\ Reals\ \{s. s \in Reals \ \&\ s < x\}\ y$

by (*auto dest: order-less-trans intro: order-less-imp-le intro!: isUbI settleI*)

lemma *lemma-minus-le-zero*: $t \leq t + -r \implies r \leq (0::hypreal)$

apply (*drule-tac c = -t in add-left-mono*)

apply (*auto simp add: add-assoc [symmetric]*)

done

lemma *lemma-st-part-le2*:

$[[(x::hypreal) \in HFinite;$
 $isLub\ Reals\ \{s. s \in Reals \ \&\ s < x\}\ t;$
 $r \in Reals; 0 < r]]$
 $\implies t + -r \leq x$

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*:

$[[(x::hypreal) \in HFinite;$
 $isLub\ Reals\ \{s. s \in Reals \ \&\ s < x\}\ t;$
 $r \in Reals; 0 < r]]$
 $\implies x + -t \leq r$

apply (*subgoal-tac x ≤ t+r*)

apply (*auto intro: lemma-st-part-le1*)

done

lemma *lemma-st-part2a*:

$[[(x::hypreal) \in HFinite;$
 $isLub\ Reals\ \{s. s \in Reals \ \&\ s < x\}\ t;$
 $r \in Reals; 0 < r]]$
 $\implies -(x + -t) \leq r$

apply (*subgoal-tac (t + -r ≤ x)*)

apply (*auto intro: lemma-st-part-le2*)

done

lemma *lemma-SReal-ub*:

$(x::hypreal) \in Reals \implies isUb\ Reals\ \{s. s \in Reals \ \&\ s < x\}\ x$

by (*auto intro: isUbI settleI order-less-imp-le*)

lemma *lemma-SReal-lub*:

$(x::hypreal) \in Reals \implies isLub\ Reals\ \{s. s \in Reals \ \&\ s < x\}\ x$

```

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::hypreal) ∈ HFinite;
    isLub Reals {s. s ∈ Reals & s < x} t;
    r ∈ Reals; 0 < r |]
  ==> x + -t ≠ r
apply auto
apply (frule isLubD1a [THEN Reals-minus])
apply (drule Reals-add-cancel, assumption)
apply (drule-tac x = x in lemma-SReal-lub)
apply (drule hypreal-isLub-unique, assumption, auto)
done

```

```

lemma lemma-st-part-not-eq2:
  [| (x::hypreal) ∈ HFinite;
    isLub Reals {s. s ∈ Reals & s < x} t;
    r ∈ Reals; 0 < r |]
  ==> -(x + -t) ≠ r
apply (auto)
apply (frule isLubD1a)
apply (drule Reals-add-cancel, assumption)
apply (drule-tac a = -x in Reals-minus, simp)
apply (drule-tac x = x in lemma-SReal-lub)
apply (drule hypreal-isLub-unique, assumption, auto)
done

```

```

lemma lemma-st-part-major:
  [| (x::hypreal) ∈ HFinite;
    isLub Reals {s. s ∈ Reals & s < x} t;
    r ∈ Reals; 0 < r |]
  ==> abs (x - t) < r
apply (frule lemma-st-part1a)
apply (frule-tac [4] lemma-st-part2a, auto)
apply (drule order-le-imp-less-or-eq)+
apply (auto dest: lemma-st-part-not-eq1 lemma-st-part-not-eq2 simp add: abs-less-iff)
done

```

```

lemma lemma-st-part-major2:
  [| (x::hypreal) ∈ HFinite; isLub Reals {s. s ∈ Reals & s < x} t |]
  ==> ∀ r ∈ Reals. 0 < r --> abs (x - t) < r
by (blast dest!: lemma-st-part-major)

```

Existence of real and Standard Part Theorem

lemma *lemma-st-part-Ex*:

$(x::\text{hypreal}) \in \text{HFinite}$

$\implies \exists t \in \text{Reals}. \forall r \in \text{Reals}. 0 < r \implies \text{abs } (x - t) < r$

apply (*frule lemma-st-part-lub, safe*)

apply (*frule isLubD1a*)

apply (*blast dest: lemma-st-part-major2*)

done

lemma *st-part-Ex*:

$(x::\text{hypreal}) \in \text{HFinite} \implies \exists t \in \text{Reals}. x @= t$

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*: $x \in \text{HFinite} \implies \text{EX! } t::\text{hypreal}. t \in \text{Reals} \ \& \ x @= t$

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

8.9 Finite, Infinite and Infinitesimal

lemma *HFinite-Int-HInfinite-empty* [*simp*]: $\text{HFinite Int HInfinite} = \{\}$

apply (*simp add: HFinite-def HInfinite-def*)

apply (*auto dest: order-less-trans*)

done

lemma *HFinite-not-HInfinite*:

assumes $x: x \in \text{HFinite}$ **shows** $x \notin \text{HInfinite}$

proof

assume $x': x \in \text{HInfinite}$

with x **have** $x \in \text{HFinite} \cap \text{HInfinite}$ **by** *blast*

thus *False* **by** *auto*

qed

lemma *not-HFinite-HInfinite*: $x \notin \text{HFinite} \implies x \in \text{HInfinite}$

apply (*simp add: HInfinite-def HFinite-def, auto*)

apply (*drule-tac x = r + 1 in bspec*)

apply (*auto*)

done

lemma *HInfinite-HFinite-disj*: $x \in \text{HInfinite} \mid x \in \text{HFinite}$

by (*blast intro: not-HFinite-HInfinite*)

lemma *HInfinite-HFinite-iff*: $(x \in \text{HInfinite}) = (x \notin \text{HFinite})$

by (*blast dest: HFinite-not-HInfinite not-HFinite-HInfinite*)

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

lemma *HInfinite-diff-HFinite-Infinitesimal-disj*:
 $x \notin \text{Infinitesimal} \implies x \in \text{HInfinite} \mid x \in \text{HFinite} - \text{Infinitesimal}$
by (*fast intro: not-HFinite-HInfinite*)

lemma *HFinite-inverse*:
fixes $x :: 'a::\text{real-normed-div-algebra star}$
shows $[\mid x \in \text{HFinite}; x \notin \text{Infinitesimal} \mid] \implies \text{inverse } x \in \text{HFinite}$
apply (*subgoal-tac x $\neq 0$*)
apply (*cut-tac x = inverse x in HInfinite-HFinite-disj*)
apply (*auto dest!: HInfinite-inverse-Infinitesimal*
simp add: nonzero-inverse-inverse-eq)
done

lemma *HFinite-inverse2*:
fixes $x :: 'a::\text{real-normed-div-algebra star}$
shows $x \in \text{HFinite} - \text{Infinitesimal} \implies \text{inverse } x \in \text{HFinite}$
by (*blast intro: HFinite-inverse*)

lemma *Infinitesimal-inverse-HFinite*:
fixes $x :: 'a::\text{real-normed-div-algebra star}$
shows $x \notin \text{Infinitesimal} \implies \text{inverse}(x) \in \text{HFinite}$
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*:
fixes $x :: 'a::\text{real-normed-div-algebra star}$
shows $x \in \text{HFinite} - \text{Infinitesimal} \implies \text{inverse } x \in \text{HFinite} - \text{Infinitesimal}$
apply (*auto intro: Infinitesimal-inverse-HFinite*)
apply (*drule Infinitesimal-HFinite-mult2, assumption*)
apply (*simp add: not-Infinitesimal-not-zero right-inverse*)
done

lemma *approx-inverse*:
fixes $x y :: 'a::\text{real-normed-div-algebra star}$
shows
 $[\mid x @= y; y \in \text{HFinite} - \text{Infinitesimal} \mid]$
 $\implies \text{inverse } x @= \text{inverse } y$
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  $c = \text{inverse } x$  in approx-mult1, assumption)
apply (auto intro: approx-sym simp add: mult-assoc)
done

```

```

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

```

```

lemma inverse-add-Infinitesimal-approx:
  fixes  $x h :: 'a::\text{real-normed-div-algebra star}$ 
  shows
    [|  $x \in \text{HFfinite} - \text{Infinitesimal}$ ;
       $h \in \text{Infinitesimal}$  |]  $\implies \text{inverse}(x + h) @= \text{inverse } x$ 
apply (auto intro: approx-inverse approx-sym Infinitesimal-add-approx-self)
done

```

```

lemma inverse-add-Infinitesimal-approx2:
  fixes  $x h :: 'a::\text{real-normed-div-algebra star}$ 
  shows
    [|  $x \in \text{HFfinite} - \text{Infinitesimal}$ ;
       $h \in \text{Infinitesimal}$  |]  $\implies \text{inverse}(h + x) @= \text{inverse } x$ 
apply (rule add-commute [THEN subst])
apply (blast intro: inverse-add-Infinitesimal-approx)
done

```

```

lemma inverse-add-Infinitesimal-approx-Infinitesimal:
  fixes  $x h :: 'a::\text{real-normed-div-algebra star}$ 
  shows
    [|  $x \in \text{HFfinite} - \text{Infinitesimal}$ ;
       $h \in \text{Infinitesimal}$  |]  $\implies \text{inverse}(x + h) - \text{inverse } x @= h$ 
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:
  fixes  $x :: 'a::\text{real-normed-div-algebra star}$ 
  shows  $(x \in \text{Infinitesimal}) = (x*x \in \text{Infinitesimal})$ 
apply (auto intro: Infinitesimal-mult)
apply (rule ccontr, frule Infinitesimal-inverse-HFfinite)
apply (frule not-Infinitesimal-not-zero)
apply (auto dest: Infinitesimal-HFfinite-mult simp add: mult-assoc)
done
declare Infinitesimal-square-iff [symmetric, simp]

```

```

lemma HFfinite-square-iff [simp]:
  fixes  $x :: 'a::\text{real-normed-div-algebra star}$ 
  shows  $(x*x \in \text{HFfinite}) = (x \in \text{HFfinite})$ 

```

```

apply (auto intro: HFinite-mult)
apply (auto dest: HInfinite-mult simp add: HFinite-HInfinite-iff)
done

```

```

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

```

```

lemma approx-HFinite-mult-cancel:
  fixes a w z :: 'a::real-normed-div-algebra star
  shows [| a: HFinite-Infinitesimal; a * w @= a * z |] ==> w @= z
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:
  fixes a w z :: 'a::real-normed-div-algebra star
  shows a: HFinite-Infinitesimal ==> (a * w @= a * z) = (w @= z)
by (auto intro: approx-mult2 approx-HFinite-mult-cancel)

```

```

lemma HInfinite-HFinite-add-cancel:
  [| x + y ∈ HInfinite; y ∈ HFinite |] ==> x ∈ 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 ∈ HInfinite; y ∈ HFinite |] ==> x + y ∈ 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::hypreal) ∈ HInfinite; x ≤ y; 0 ≤ x |] ==> y ∈ HInfinite
by (auto intro: HFinite-bounded simp add: HInfinite-HFinite-iff)

```

```

lemma Infinitesimal-inverse-HInfinite:
  fixes x :: 'a::real-normed-div-algebra star
  shows [| x ∈ Infinitesimal; x ≠ 0 |] ==> inverse x ∈ HInfinite
apply (rule ccontr, drule HFinite-HInfinite-iff [THEN iffD2])
apply (auto dest: Infinitesimal-HFinite-mult2)
done

```

```

lemma HInfinite-HFinite-not-Infinitesimal-mult:
  fixes x y :: 'a::real-normed-div-algebra star

```

```

  shows [| x ∈ HInfinite; y ∈ HFinite – Infinitesimal |]
    ==> x * y ∈ HInfinite
  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:
  fixes x y :: 'a::real-normed-div-algebra star
  shows [| x ∈ HInfinite; y ∈ HFinite – Infinitesimal |]
    ==> y * x ∈ HInfinite
  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::hypreal) ∈ HInfinite; 0 < x; y ∈ Reals |] ==> y < x
  by (auto dest!: bspec simp add: HInfinite-def abs-if order-less-imp-le)

```

```

lemma HInfinite-gt-zero-gt-one:
  [| (x::hypreal) ∈ HInfinite; 0 < x |] ==> 1 < x
  by (auto intro: HInfinite-gt-SReal)

```

```

lemma not-HInfinite-one [simp]: 1 ∉ HInfinite
  apply (simp (no-asm) add: HInfinite-HFinite-iff)
done

```

```

lemma approx-hrabs-disj: abs (x::hypreal) @= x | abs x @= -x
  by (cut-tac x = x in hrabs-disj, auto)

```

8.10 Theorems about Monads

```

lemma monad-hrabs-Un-subset: monad (abs x) ≤ monad(x::hypreal) Un monad(-x)
  by (rule-tac x1 = x in hrabs-disj [THEN disjE], auto)

```

```

lemma Infinitesimal-monad-eq: e ∈ Infinitesimal ==> 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 ∈ monad x) = (-u ∈ monad (-x))
  by (simp add: monad-def)

```

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

lemma *monad-zero-minus-iff*: $(x \in \text{monad } 0) = (-x \in \text{monad } 0)$
apply (*simp (no-asm) add: Infinitesimal-monad-zero-iff [symmetric]*)
done

lemma *monad-zero-hrabs-iff*: $((x::\text{hypreal}) \in \text{monad } 0) = (\text{abs } x \in \text{monad } 0)$
apply (*rule-tac x1 = x in hrabs-disj [THEN disjE]*)
apply (*auto simp add: monad-zero-minus-iff [symmetric]*)
done

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

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

lemma *approx-subset-monad*: $x @= y \implies \{x,y\} \leq \text{monad } x$
apply (*simp (no-asm)*)
apply (*simp add: approx-monad-iff*)
done

lemma *approx-subset-monad2*: $x @= y \implies \{x,y\} \leq \text{monad } y$
apply (*drule approx-sym*)
apply (*fast dest: approx-subset-monad*)
done

lemma *mem-monad-approx*: $u \in \text{monad } x \implies x @= u$
by (*simp add: monad-def*)

lemma *approx-mem-monad*: $x @= u \implies u \in \text{monad } x$
by (*simp add: monad-def*)

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

lemma *approx-mem-monad-zero*: $[[x @= y; x \in \text{monad } 0]] \implies y \in \text{monad } 0$
apply (*drule mem-monad-approx*)
apply (*fast intro: approx-mem-monad approx-trans*)
done

lemma *Infinitesimal-approx-hrabs*:
 $[[x @= y; (x::\text{hypreal}) \in \text{Infinitesimal}]] \implies \text{abs } x @= \text{abs } y$
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*:

[[$0 < x$; $(x::\text{hypreal}) \notin \text{Infinitesimal}$; $e : \text{Infinitesimal}$]] $\implies e < x$
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*:

[[$0 < (x::\text{hypreal})$; $x \notin \text{Infinitesimal}$; $u \in \text{monad } x$]] $\implies 0 < u$
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*:

[[$(x::\text{hypreal}) < 0$; $x \notin \text{Infinitesimal}$; $u \in \text{monad } x$]] $\implies u < 0$
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*:

[[$0 < (x::\text{hypreal})$; $x \notin \text{Infinitesimal}$; $x @= y$]] $\implies 0 < y$
by (blast dest: *Ball-mem-monad-gt-zero approx-subset-monad*)

lemma *lemma-approx-less-zero*:

[[$(x::\text{hypreal}) < 0$; $x \notin \text{Infinitesimal}$; $x @= y$]] $\implies y < 0$
by (blast dest: *Ball-mem-monad-less-zero approx-subset-monad*)

theorem *approx-hrabs*: $(x::\text{hypreal}) @= y \implies \text{abs } x @= \text{abs } y$

by (drule *approx-hnorm, simp*)

lemma *approx-hrabs-zero-cancel*: $\text{abs}(x::\text{hypreal}) @= 0 \implies x @= 0$

apply (cut-tac $x = x$ in *hrabs-disj*)

apply (auto dest: *approx-minus*)

done

lemma *approx-hrabs-add-Infinitesimal*:

$(e::\text{hypreal}) \in \text{Infinitesimal} \implies \text{abs } x @= \text{abs}(x+e)$

by (fast intro: *approx-hrabs Infinitesimal-add-approx-self*)

lemma *approx-hrabs-add-minus-Infinitesimal*:

$(e::\text{hypreal}) \in \text{Infinitesimal} \implies \text{abs } x @= \text{abs}(x - e)$

by (fast intro: *approx-hrabs Infinitesimal-add-minus-approx-self*)

lemma *hrabs-add-Infinitesimal-cancel*:

[[$(e::\text{hypreal}) \in \text{Infinitesimal}$; $e' \in \text{Infinitesimal}$;
 $\text{abs}(x+e) = \text{abs}(y+e')$]] $\implies \text{abs } x @= \text{abs } y$

```

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:
  [| ( $e::\text{hypreal}$ )  $\in$  Infinitesimal;  $e' \in$  Infinitesimal;
     $\text{abs}(x + -e) = \text{abs}(y + -e')$  |] ==>  $\text{abs } x @= \text{abs } y$ 
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

```

8.12 More *HFinite* and *Infinitesimal* Theorems

```

lemma Infinitesimal-add-hypreal-of-real-less:
  [|  $x < y$ ;  $u \in$  Infinitesimal |]
  ==>  $\text{hypreal-of-real } x + u < \text{hypreal-of-real } y$ 
apply (simp add: Infinitesimal-def)
apply (drule-tac  $x = \text{hypreal-of-real } y + -\text{hypreal-of-real } x$  in bspec, simp)
apply (simp add: abs-less-iff)
done

```

```

lemma Infinitesimal-add-hrabs-hypreal-of-real-less:
  [|  $x \in$  Infinitesimal;  $\text{abs}(\text{hypreal-of-real } r) < \text{hypreal-of-real } y$  |]
  ==>  $\text{abs}(\text{hypreal-of-real } r + x) < \text{hypreal-of-real } y$ 
apply (drule-tac  $x = \text{hypreal-of-real } r$  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:
  [|  $x \in$  Infinitesimal;  $\text{abs}(\text{hypreal-of-real } r) < \text{hypreal-of-real } y$  |]
  ==>  $\text{abs}(x + \text{hypreal-of-real } r) < \text{hypreal-of-real } y$ 
apply (rule add-commute [THEN subst])
apply (erule Infinitesimal-add-hrabs-hypreal-of-real-less, assumption)
done

```

```

lemma hypreal-of-real-le-add-Infinitesimal-cancel:
  [|  $u \in$  Infinitesimal;  $v \in$  Infinitesimal;
     $\text{hypreal-of-real } x + u \leq \text{hypreal-of-real } y + v$  |]
  ==>  $\text{hypreal-of-real } x \leq \text{hypreal-of-real } y$ 
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*:

$[| u \in \text{Infininitesimal}; v \in \text{Infininitesimal};$
 $\text{hypreal-of-real } x + u \leq \text{hypreal-of-real } y + v |]$
 $\implies x \leq y$

by (*blast intro: star-of-le [THEN iffD1]*
intro!: hypreal-of-real-le-add-Infininitesimal-cancel)

lemma *hypreal-of-real-less-Infininitesimal-le-zero*:

$[| \text{hypreal-of-real } x < e; e \in \text{Infininitesimal} |] \implies \text{hypreal-of-real } x \leq 0$

apply (*rule linorder-not-less [THEN iffD1], safe*)

apply (*drule Infininitesimal-interval*)

apply (*drule-tac [4] SReal-hypreal-of-real [THEN SReal-Infininitesimal-zero], auto*)

done

lemma *Infininitesimal-add-not-zero*:

$[| h \in \text{Infininitesimal}; x \neq 0 |] \implies \text{star-of } x + h \neq 0$

apply *auto*

apply (*subgoal-tac h = - star-of x, auto intro: minus-unique [symmetric]*)

done

lemma *Infininitesimal-square-cancel [simp]*:

$(x::\text{hypreal}) * x + y * y \in \text{Infininitesimal} \implies x * x \in \text{Infininitesimal}$

apply (*rule Infininitesimal-interval2*)

apply (*rule-tac [3] zero-le-square, assumption*)

apply (*auto*)

done

lemma *HFinite-square-cancel [simp]*:

$(x::\text{hypreal}) * x + y * y \in \text{HFinite} \implies x * x \in \text{HFinite}$

apply (*rule HFinite-bounded, assumption*)

apply (*auto*)

done

lemma *Infininitesimal-square-cancel2 [simp]*:

$(x::\text{hypreal}) * x + y * y \in \text{Infininitesimal} \implies y * y \in \text{Infininitesimal}$

apply (*rule Infininitesimal-square-cancel*)

apply (*rule add-commute [THEN subst]*)

apply (*simp (no-asm)*)

done

lemma *HFinite-square-cancel2 [simp]*:

$(x::\text{hypreal}) * x + y * y \in \text{HFinite} \implies y * y \in \text{HFinite}$

apply (*rule HFinite-square-cancel*)

apply (*rule add-commute [THEN subst]*)

apply (*simp (no-asm)*)

done

lemma *Infininitesimal-sum-square-cancel [simp]*:

```

    (x::hypreal)*x + y*y + z*z ∈ Infinitesimal ==> x*x ∈ Infinitesimal
  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::hypreal)*x + y*y + z*z ∈ HFinite ==> x*x ∈ HFinite
  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::hypreal)*y + x*x + z*z ∈ Infinitesimal ==> x*x ∈ Infinitesimal
  apply (rule Infinitesimal-sum-square-cancel)
  apply (simp add: add-ac)
  done

```

```

lemma HFinite-sum-square-cancel2 [simp]:
  (y::hypreal)*y + x*x + z*z ∈ HFinite ==> x*x ∈ HFinite
  apply (rule HFinite-sum-square-cancel)
  apply (simp add: add-ac)
  done

```

```

lemma Infinitesimal-sum-square-cancel3 [simp]:
  (z::hypreal)*z + y*y + x*x ∈ Infinitesimal ==> x*x ∈ Infinitesimal
  apply (rule Infinitesimal-sum-square-cancel)
  apply (simp add: add-ac)
  done

```

```

lemma HFinite-sum-square-cancel3 [simp]:
  (z::hypreal)*z + y*y + x*x ∈ HFinite ==> x*x ∈ HFinite
  apply (rule HFinite-sum-square-cancel)
  apply (simp add: add-ac)
  done

```

```

lemma monad-hrabs-less:
  [| y ∈ monad x; 0 < hypreal-of-real e |]
  ==> abs (y - x) < hypreal-of-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 ∈ monad (hypreal-of-real a) ==> x ∈ HFinite

```

```

apply (drule mem-monad-approx [THEN approx-sym])
apply (drule bex-Infinitesimal-iff2 [THEN iffD2])
apply (safe dest!: Infinitesimal-subset-HFfinite [THEN subsetD])
apply (erule SReal-hypreal-of-real [THEN SReal-subset-HFfinite [THEN subsetD],
  THEN HFfinite-add])
done

```

8.13 Theorems about Standard Part

```

lemma st-approx-self:  $x \in HFfinite \implies st\ x \ @ = x$ 
apply (simp add: st-def)
apply (frule st-part-Ex, safe)
apply (rule someI2)
apply (auto intro: approx-sym)
done

```

```

lemma st-SReal:  $x \in HFfinite \implies st\ x \in Reals$ 
apply (simp add: st-def)
apply (frule st-part-Ex, safe)
apply (rule someI2)
apply (auto intro: approx-sym)
done

```

```

lemma st-HFfinite:  $x \in HFfinite \implies st\ x \in HFfinite$ 
by (erule st-SReal [THEN SReal-subset-HFfinite [THEN subsetD]])

```

```

lemma st-unique:  $\llbracket r \in \mathbf{R}; r \approx x \rrbracket \implies st\ x = r$ 
apply (frule SReal-subset-HFfinite [THEN subsetD])
apply (drule (1) approx-HFfinite)
apply (unfold st-def)
apply (rule some-equality)
apply (auto intro: approx-unique-real)
done

```

```

lemma st-SReal-eq:  $x \in Reals \implies st\ x = x$ 
apply (erule st-unique)
apply (rule approx-refl)
done

```

```

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

```

```

lemma st-eq-approx:  $\llbracket x \in HFfinite; y \in HFfinite; st\ x = st\ y \rrbracket \implies x\ @ = y$ 
by (auto dest!: st-approx-self elim!: approx-trans3)

```

```

lemma approx-st-eq:
  assumes  $x: x \in HFfinite$  and  $y: y \in HFfinite$  and  $xy: x\ @ = y$ 
  shows  $st\ x = st\ y$ 
proof –

```

have $st\ x \ @ = x\ st\ y \ @ = y\ st\ x \in Reals\ st\ y \in Reals$
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*:
 $\llbracket x \in HFinite; y \in HFinite \rrbracket$
 $\implies (x \ @ = y) = (st\ x = st\ y)$
by (*blast intro: approx-st-eq st-eq-approx*)

lemma *st-Infinitesimal-add-SReal*:
 $\llbracket x \in Reals; e \in Infinitesimal \rrbracket \implies st(x + e) = x$
apply (*erule st-unique*)
apply (*erule Infinitesimal-add-approx-self*)
done

lemma *st-Infinitesimal-add-SReal2*:
 $\llbracket x \in Reals; e \in Infinitesimal \rrbracket \implies st(e + x) = x$
apply (*erule st-unique*)
apply (*erule Infinitesimal-add-approx-self2*)
done

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

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

lemma *st-numeral [simp]*: $st(\text{numeral } w) = \text{numeral } w$
by (*rule Reals-numeral [THEN st-SReal-eq]*)

lemma *st-neg-numeral [simp]*: $st(\text{neg-numeral } w) = \text{neg-numeral } w$
by (*rule Reals-neg-numeral [THEN st-SReal-eq]*)

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

lemma *st-diff*: $\llbracket x \in HFinite; y \in HFinite \rrbracket \implies st(x - y) = st\ x - st\ y$
by (*simp add: st-unique st-SReal st-approx-self approx-diff*)

lemma *st-mult*: $\llbracket x \in \mathit{HFinite}; y \in \mathit{HFinite} \rrbracket \implies \mathit{st}(x * y) = \mathit{st} x * \mathit{st} y$
by (*simp add: st-unique st-SReal st-approx-self approx-mult-HFinite*)

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

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

lemma *st-inverse*:
 $\llbracket x \in \mathit{HFinite}; \mathit{st} x \neq 0 \rrbracket$
 $\implies \mathit{st}(\mathit{inverse} x) = \mathit{inverse}(\mathit{st} x)$
apply (*rule-tac c1 = st x in hypreal-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]*:
 $\llbracket x \in \mathit{HFinite}; y \in \mathit{HFinite}; \mathit{st} y \neq 0 \rrbracket$
 $\implies \mathit{st}(x/y) = (\mathit{st} x) / (\mathit{st} y)$
by (*simp add: divide-inverse st-mult st-not-Infinitesimal HFinite-inverse st-inverse*)

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

lemma *Infinitesimal-add-st-less*:
 $\llbracket x \in \mathit{HFinite}; y \in \mathit{HFinite}; u \in \mathit{Infinitesimal}; \mathit{st} x < \mathit{st} y \rrbracket$
 $\implies \mathit{st} x + u < \mathit{st} y$
apply (*drule st-SReal+*)
apply (*auto intro!: Infinitesimal-add-hypreal-of-real-less simp add: SReal-iff*)
done

lemma *Infinitesimal-add-st-le-cancel*:
 $\llbracket x \in \mathit{HFinite}; y \in \mathit{HFinite};$
 $u \in \mathit{Infinitesimal}; \mathit{st} x \leq \mathit{st} y + u$
 $\rrbracket \implies \mathit{st} x \leq \mathit{st} y$
apply (*simp add: linorder-not-less [symmetric]*)
apply (*auto dest: Infinitesimal-add-st-less*)
done

lemma *st-le*: $\llbracket x \in \mathit{HFinite}; y \in \mathit{HFinite}; x \leq y \rrbracket \implies \mathit{st}(x) \leq \mathit{st}(y)$
apply (*frule HFinite-st-Infinitesimal-add*)
apply (*rotate-tac 1*)
apply (*frule HFinite-st-Infinitesimal-add, safe*)
apply (*rule Infinitesimal-add-st-le-cancel*)
apply (*rule-tac [3] x = ea and y = e in Infinitesimal-diff*)
apply (*auto simp add: add-assoc [symmetric]*)
done

lemma *st-zero-le*: $[| 0 \leq x; x \in \mathit{HFinite} |] \implies 0 \leq \mathit{st} x$
apply (*subst st-0 [symmetric]*)
apply (*rule st-le, auto*)
done

lemma *st-zero-ge*: $[| x \leq 0; x \in \mathit{HFinite} |] \implies \mathit{st} x \leq 0$
apply (*subst st-0 [symmetric]*)
apply (*rule st-le, auto*)
done

lemma *st-hrabs*: $x \in \mathit{HFinite} \implies \mathit{abs}(\mathit{st} x) = \mathit{st}(\mathit{abs} x)$
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

8.14 Alternative Definitions using Free Ultrafilter

8.14.1 *HFinite*

lemma *HFinite-FreeUltrafilterNat*:
 $\mathit{star-n} X \in \mathit{HFinite} \implies \exists u. \{n. \mathit{norm} (X n) < u\} \in \mathit{FreeUltrafilterNat}$
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*:
 $\exists u. \{n. \mathit{norm} (X n) < u\} \in \mathit{FreeUltrafilterNat} \implies \mathit{star-n} X \in \mathit{HFinite}$
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*:
 $(\mathit{star-n} X \in \mathit{HFinite}) = (\exists u. \{n. \mathit{norm} (X n) < u\} \in \mathit{FreeUltrafilterNat})$
by (*blast intro!: HFinite-FreeUltrafilterNat FreeUltrafilterNat-HFinite*)

8.14.2 *HInfinite*

lemma *lemma-Compl-eq*: $-\{n. u < \mathit{norm} (xa n)\} = \{n. \mathit{norm} (xa n) \leq u\}$
by *auto*

lemma *lemma-Compl-eq2*: $-\{n. \mathit{norm} (xa n) < u\} = \{n. u \leq \mathit{norm} (xa n)\}$
by *auto*

lemma *lemma-Int-eq1*:

$$\{n. \text{norm } (xa \ n) \leq u\} \text{ Int } \{n. u \leq \text{norm } (xa \ n)\} \\ = \{n. \text{norm}(xa \ n) = u\}$$

by *auto*

lemma *lemma-FreeUltrafilterNat-one*:

$$\{n. \text{norm } (xa \ n) = u\} \leq \{n. \text{norm } (xa \ n) < u + (1::\text{real})\}$$

by *auto*

lemma *FreeUltrafilterNat-const-Finite*:

$$\{n. \text{norm } (X \ n) = u\} \in \text{FreeUltrafilterNat} \implies \text{star-}n \ X \in \text{HFinite}$$

apply (*rule FreeUltrafilterNat-HFinite*)

apply (*rule-tac x = u + 1 in exI*)

apply (*erule ultra, simp*)

done

lemma *HInfinite-FreeUltrafilterNat*:

$$\text{star-}n \ X \in \text{HInfinite} \implies \forall u. \{n. u < \text{norm } (X \ n)\} \in \text{FreeUltrafilterNat}$$

apply (*drule HInfinite-HFinite-iff [THEN iffD1]*)

apply (*simp add: HFinite-FreeUltrafilterNat-iff*)

apply (*rule allI, drule-tac x=u + 1 in spec*)

apply (*drule FreeUltrafilterNat.not-memD*)

apply (*simp add: Collect-neg-eq [symmetric] linorder-not-less*)

apply (*erule ultra, simp*)

done

lemma *lemma-Int-HI*:

$$\{n. \text{norm } (Xa \ n) < u\} \text{ Int } \{n. X \ n = Xa \ n\} \subseteq \{n. \text{norm } (X \ n) < (u::\text{real})\}$$

by *auto*

lemma *lemma-Int-HIa*: $\{n. u < \text{norm } (X \ n)\} \text{ Int } \{n. \text{norm } (X \ n) < u\} = \{\}$

by (*auto intro: order-less-asy*)

lemma *FreeUltrafilterNat-HInfinite*:

$$\forall u. \{n. u < \text{norm } (X \ n)\} \in \text{FreeUltrafilterNat} \implies \text{star-}n \ X \in \text{HInfinite}$$

apply (*rule HInfinite-HFinite-iff [THEN iffD2]*)

apply (*safe, drule HFinite-FreeUltrafilterNat, safe*)

apply (*drule-tac x = u in spec*)

apply (*drule (1) FreeUltrafilterNat.Int*)

apply (*simp add: Collect-conj-eq [symmetric]*)

apply (*subgoal-tac $\forall n. \neg (\text{norm } (X \ n) < u \wedge u < \text{norm } (X \ n))$, auto*)

done

lemma *HInfinite-FreeUltrafilterNat-iff*:

$$(\text{star-}n \ X \in \text{HInfinite}) = (\forall u. \{n. u < \text{norm } (X \ n)\} \in \text{FreeUltrafilterNat})$$

by (*blast intro!: HInfinite-FreeUltrafilterNat FreeUltrafilterNat-HInfinite*)

8.14.3 *Infinitesimal*

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

lemma *Infinitesimal-FreeUltrafilterNat*:

$\text{star-}n\ X \in \text{Infinitesimal} \implies \forall u > 0. \{n. \text{norm}\ (X\ n) < u\} \in \mathcal{U}$

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*:

$\forall u > 0. \{n. \text{norm}\ (X\ n) < u\} \in \mathcal{U} \implies \text{star-}n\ X \in \text{Infinitesimal}$

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*:

$(\text{star-}n\ X \in \text{Infinitesimal}) = (\forall u > 0. \{n. \text{norm}\ (X\ n) < u\} \in \mathcal{U})$

by (*blast intro!: Infinitesimal-FreeUltrafilterNat FreeUltrafilterNat-Infinitesimal*)

lemma *lemma-Infinitesimal*:

$(\forall r. 0 < r \implies x < r) = (\forall n. x < \text{inverse}(\text{real}\ (\text{Suc}\ n)))$

apply (*auto simp add: real-of-nat-Suc-gt-zero*)

apply (*blast dest!: reals-Archimedean intro: order-less-trans*)

done

lemma *lemma-Infinitesimal2*:

$(\forall r \in \text{Reals}. 0 < r \implies x < r) =$

$(\forall n. x < \text{inverse}(\text{hypreal-of-nat}\ (\text{Suc}\ n)))$

apply *safe*

apply (*drule-tac x = inverse (hypreal-of-real (real (Suc n))) in bspec*)

apply (*simp (no-asm-use)*)

apply (*rule real-of-nat-Suc-gt-zero [THEN positive-imp-inverse-positive, THEN star-of-less [THEN iffD2], THEN [?] impE]*)

prefer 2 **apply** *assumption*

apply (*simp add: real-of-nat-def*)

apply (*auto dest!: reals-Archimedean simp add: SReal-iff*)

apply (*drule star-of-less [THEN iffD2]*)

apply (*simp add: real-of-nat-def*)

apply (*blast intro: order-less-trans*)

done

lemma *Infinitesimal-hypreal-of-nat-iff*:

$\text{Infinitesimal} = \{x. \forall n. \text{hnorm}\ x < \text{inverse}\ (\text{hypreal-of-nat}\ (\text{Suc}\ n))\}$

```

apply (simp add: Infinitesimal-def)
apply (auto simp add: lemma-Infinitesimal2)
done

```

8.15 Proof that ω is an infinite number

It will follow that epsilon is an infinitesimal number.

```

lemma Suc-Un-eq:  $\{n. n < \text{Suc } m\} = \{n. n < m\} \cup \{n. n = m\}$ 
by (auto simp add: less-Suc-eq)

```

```

lemma finite-nat-segment: finite  $\{n::\text{nat}. n < m\}$ 
apply (induct m)
apply (auto simp add: Suc-Un-eq)
done

```

```

lemma finite-real-of-nat-segment: finite  $\{n::\text{nat}. \text{real } n < \text{real } (m::\text{nat})\}$ 
by (auto intro: finite-nat-segment)

```

```

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

```

```

lemma finite-real-of-nat-le-real: finite  $\{n::\text{nat}. \text{real } n \leq u\}$ 
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  $\{n::\text{nat}. \text{abs}(\text{real } n) \leq u\}$ 
apply (simp (no-asm) add: real-of-nat-Suc-gt-zero finite-real-of-nat-le-real)
done

```

```

lemma rabs-real-of-nat-le-real-FreeUltrafilterNat:
   $\{n. \text{abs}(\text{real } n) \leq u\} \notin \text{FreeUltrafilterNat}$ 
by (blast intro!: FreeUltrafilterNat.finite finite-rabs-real-of-nat-le-real)

```

```

lemma FreeUltrafilterNat-nat-gt-real:  $\{n. u < \text{real } n\} \in \text{FreeUltrafilterNat}$ 
apply (rule ccontr, drule FreeUltrafilterNat.not-memD)
apply (subgoal-tac -  $\{n::\text{nat}. u < \text{real } n\} = \{n. \text{real } n \leq u\}$ )
prefer 2 apply force
apply (simp add: finite-real-of-nat-le-real [THEN FreeUltrafilterNat.finite])
done

```

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*

lemma *FreeUltrafilterNat-omega*: $\{n. u < real\ n\} \in FreeUltrafilterNat$
apply (*cut-tac u = u in rabs-real-of-nat-le-real-FreeUltrafilterNat*)
apply (*auto dest: FreeUltrafilterNat.not-memD simp add: Compl-real-le-eq*)
done

theorem *HInfinite-omega [simp]*: $\omega \in HInfinite$
apply (*simp add: omega-def*)
apply (*rule FreeUltrafilterNat-HInfinite*)
apply (*simp (no-asm) add: real-norm-def real-of-nat-Suc diff-less-eq [symmetric]*
FreeUltrafilterNat-omega)
done

lemma *Infinitesimal-epsilon [simp]*: $\epsilon \in Infinitesimal$
by (*auto intro!*: *HInfinite-inverse-Infinitesimal HInfinite-omega simp add: hypreal-epsilon-inverse-omega*)

lemma *HFinite-epsilon [simp]*: $\epsilon \in HFinite$
by (*auto intro: Infinitesimal-subset-HFinite [THEN subsetD]*)

lemma *epsilon-approx-zero [simp]*: $\epsilon \approx 0$
apply (*simp (no-asm) add: mem-infmal-iff [symmetric]*)
done

lemma *real-of-nat-less-inverse-iff*:
 $0 < u \implies (u < inverse\ (real(Suc\ n))) = (real(Suc\ n) < inverse\ u)$
apply (*simp add: inverse-eq-divide*)
apply (*subst pos-less-divide-eq, assumption*)
apply (*subst pos-less-divide-eq*)
apply (*simp add: real-of-nat-Suc-gt-zero*)
apply (*simp add: mult-commute*)
done

lemma *finite-inverse-real-of-posnat-gt-real*:
 $0 < u \implies finite\ \{n. u < inverse(real(Suc\ n))\}$
apply (*simp (no-asm-simp) add: real-of-nat-less-inverse-iff*)
apply (*simp (no-asm-simp) add: real-of-nat-Suc less-diff-eq [symmetric]*)
apply (*rule finite-real-of-nat-less-real*)
done

lemma *lemma-real-le-Un-eq2*:
 $\{n. u \leq inverse(real(Suc\ n))\} =$
 $\{n. u < inverse(real(Suc\ n))\} \cup \{n. u = inverse(real(Suc\ n))\}$

```

apply (auto dest: order-le-imp-less-or-eq simp add: order-less-imp-le)
done

```

```

lemma real-of-nat-inverse-eq-iff:

```

```

  (u = inverse (real(Suc n))) = (real(Suc n) = inverse u)

```

```

by (auto simp add: real-of-nat-Suc-gt-zero less-imp-neq [THEN not-sym])

```

```

lemma lemma-finite-omega-set2: finite {n::nat. u = inverse(real(Suc n))}

```

```

apply (simp (no-asm-simp) add: real-of-nat-inverse-eq-iff)

```

```

apply (cut-tac x = inverse u - 1 in lemma-finite-omega-set)

```

```

apply (simp add: real-of-nat-Suc diff-eq-eq [symmetric] eq-commute)

```

```

done

```

```

lemma finite-inverse-real-of-posnat-ge-real:

```

```

  0 < u ==> finite {n. u ≤ inverse(real(Suc n))}

```

```

by (auto simp add: lemma-real-le-Un-eq2 lemma-finite-omega-set2 finite-inverse-real-of-posnat-gt-real)

```

```

lemma inverse-real-of-posnat-ge-real-FreeUltrafilterNat:

```

```

  0 < u ==> {n. u ≤ inverse(real(Suc n))} ∉ FreeUltrafilterNat

```

```

by (blast intro!: FreeUltrafilterNat.finite finite-inverse-real-of-posnat-ge-real)

```

```

lemma Compl-le-inverse-eq:

```

```

  - {n. u ≤ inverse(real(Suc n))} =

```

```

  {n. inverse(real(Suc n)) < u}

```

```

apply (auto dest!: order-le-less-trans simp add: linorder-not-le)

```

```

done

```

```

lemma FreeUltrafilterNat-inverse-real-of-posnat:

```

```

  0 < u ==>

```

```

  {n. inverse(real(Suc n)) < u} ∈ FreeUltrafilterNat

```

```

apply (cut-tac u = u in inverse-real-of-posnat-ge-real-FreeUltrafilterNat)

```

```

apply (auto dest: FreeUltrafilterNat.not-memD simp add: Compl-le-inverse-eq)

```

```

done

```

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* (%n::nat. inverse(real(Suc n)))) whn : Infinitesimal

```

```

apply (simp add: hypnat-omega-def starfun-star-n star-n-inverse)

```

```

apply (simp add: Infinitesimal-FreeUltrafilterNat-iff)

```

```

apply (simp add: real-of-nat-Suc-gt-zero FreeUltrafilterNat-inverse-real-of-posnat)

```

```

done

```

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 and standard definitions of pointwise limit.

```

lemma real-seq-to-hypreal-Infinitesimal:
   $\forall n. \text{norm}(X\ n - x) < \text{inverse}(\text{real}(\text{Suc}\ n))$ 
   $\implies \text{star-n}\ X - \text{star-of}\ x \in \text{Infinitesimal}$ 
apply (auto intro!: beXI dest: FreeUltrafilterNat-inverse-real-of-posnat FreeUltrafilterNat.Int intro: order-less-trans FreeUltrafilterNat.subset simp add: star-n-diff star-of-def Infinitesimal-FreeUltrafilterNat-iff star-n-inverse)
done

lemma real-seq-to-hypreal-approx:
   $\forall n. \text{norm}(X\ n - x) < \text{inverse}(\text{real}(\text{Suc}\ n))$ 
   $\implies \text{star-n}\ X \text{ @} = \text{star-of}\ x$ 
apply (subst approx-minus-iff)
apply (rule mem-infmal-iff [THEN subst])
apply (erule real-seq-to-hypreal-Infinitesimal)
done

lemma real-seq-to-hypreal-approx2:
   $\forall n. \text{norm}(x - X\ n) < \text{inverse}(\text{real}(\text{Suc}\ n))$ 
   $\implies \text{star-n}\ X \text{ @} = \text{star-of}\ x$ 
apply (rule real-seq-to-hypreal-approx)
apply (subst norm-minus-cancel [symmetric])
apply (simp del: norm-minus-cancel)
done

lemma real-seq-to-hypreal-Infinitesimal2:
   $\forall n. \text{norm}(X\ n - Y\ n) < \text{inverse}(\text{real}(\text{Suc}\ n))$ 
   $\implies \text{star-n}\ X - \text{star-n}\ Y \in \text{Infinitesimal}$ 
by (auto intro!: beXI
  dest: FreeUltrafilterNat-inverse-real-of-posnat
  FreeUltrafilterNat.Int
  intro: order-less-trans FreeUltrafilterNat.subset
  simp add: Infinitesimal-FreeUltrafilterNat-iff star-n-diff
  star-n-inverse)

end

```

9 NSComplex: Nonstandard Complex Numbers

```

theory NSComplex
imports Complex NSA
begin

type-synonym hcomplex = complex star

abbreviation
  hcomplex-of-complex :: complex  $\implies$  complex star where
  hcomplex-of-complex == star-of

```

abbreviation

$hmod :: \text{complex star} \Rightarrow \text{real star}$ **where**
 $hmod == hnorm$

definition

$hRe :: hcomplex \Rightarrow hypreal$ **where**
 $hRe = *f* Re$

definition

$hIm :: hcomplex \Rightarrow hypreal$ **where**
 $hIm = *f* Im$

definition

$iii :: hcomplex$ **where**
 $iii = \text{star-of } ii$

definition

$hcnj :: hcomplex \Rightarrow hcomplex$ **where**
 $hcnj = *f* cnj$

definition

$hsgn :: hcomplex \Rightarrow hcomplex$ **where**
 $hsgn = *f* sgn$

definition

$harg :: hcomplex \Rightarrow hypreal$ **where**
 $harg = *f* arg$

definition

$hcis :: hypreal \Rightarrow hcomplex$ **where**
 $hcis = *f* cis$

abbreviation

$hcomplex\text{-of-hypreal} :: hypreal \Rightarrow hcomplex$ **where**
 $hcomplex\text{-of-hypreal} \equiv \text{of-hypreal}$

definition

$hrcis :: [hypreal, hypreal] \Rightarrow hcomplex$ **where**
 $hrcis = *f2* rcis$

definition

$hexpi :: hcomplex \Rightarrow hcomplex$ **where**
 $hexpi = *f* expi$

definition

$HComplex :: [hypreal, hypreal] \Rightarrow hcomplex$ **where**
 $HComplex = *f2* Complex$

lemmas $hcomplex-defs$ [transfer-unfold] =
 $hRe-def$ $hIm-def$ $iii-def$ $hcnj-def$ $hsgn-def$ $harg-def$ $hcis-def$
 $hrcis-def$ $hexpi-def$ $HComplex-def$

lemma $Standard-hRe$ [simp]: $x \in Standard \implies hRe x \in Standard$
by (simp add: $hcomplex-defs$)

lemma $Standard-hIm$ [simp]: $x \in Standard \implies hIm x \in Standard$
by (simp add: $hcomplex-defs$)

lemma $Standard-iii$ [simp]: $iii \in Standard$
by (simp add: $hcomplex-defs$)

lemma $Standard-hcnj$ [simp]: $x \in Standard \implies hcnj x \in Standard$
by (simp add: $hcomplex-defs$)

lemma $Standard-hsgn$ [simp]: $x \in Standard \implies hsgn x \in Standard$
by (simp add: $hcomplex-defs$)

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

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

lemma $Standard-hexpi$ [simp]: $x \in Standard \implies hexpi x \in Standard$
by (simp add: $hcomplex-defs$)

lemma $Standard-hrcis$ [simp]:
 $\llbracket r \in Standard; s \in Standard \rrbracket \implies hrcis r s \in Standard$
by (simp add: $hcomplex-defs$)

lemma $Standard-HComplex$ [simp]:
 $\llbracket r \in Standard; s \in Standard \rrbracket \implies HComplex r s \in Standard$
by (simp add: $hcomplex-defs$)

lemma *hcmmod-def*: $hcmmod = *f* cmod$
by (*rule hnorm-def*)

9.1 Properties of Nonstandard Real and Imaginary Parts

lemma *hcomplex-hRe-hIm-cancel-iff*:
 $!!w z. (w=z) = (hRe(w) = hRe(z) \ \& \ hIm(w) = hIm(z))$
by *transfer (rule complex-Re-Im-cancel-iff)*

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

lemma *hcomplex-hRe-zero* [*simp*]: $hRe 0 = 0$
by *transfer (rule complex-Re-zero)*

lemma *hcomplex-hIm-zero* [*simp*]: $hIm 0 = 0$
by *transfer (rule complex-Im-zero)*

lemma *hcomplex-hRe-one* [*simp*]: $hRe 1 = 1$
by *transfer (rule complex-Re-one)*

lemma *hcomplex-hIm-one* [*simp*]: $hIm 1 = 0$
by *transfer (rule complex-Im-one)*

9.2 Addition for Nonstandard Complex Numbers

lemma *hRe-add*: $!!x y. hRe(x + y) = hRe(x) + hRe(y)$
by *transfer (rule complex-Re-add)*

lemma *hIm-add*: $!!x y. hIm(x + y) = hIm(x) + hIm(y)$
by *transfer (rule complex-Im-add)*

9.3 More Minus Laws

lemma *hRe-minus*: $!!z. hRe(-z) = - hRe(z)$
by *transfer (rule complex-Re-minus)*

lemma *hIm-minus*: $!!z. hIm(-z) = - hIm(z)$
by *transfer (rule complex-Im-minus)*

lemma *hcomplex-add-minus-eq-minus*:
 $x + y = (0::hcomplex) ==> x = -y$
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]: $!!z. iii * (iii * z) = -z$
by *transfer* (rule *complex-i-mult-minus*)

lemma *hcomplex-i-not-zero* [simp]: $iii \neq 0$
by *transfer* (rule *complex-i-not-zero*)

9.4 More Multiplication Laws

lemma *hcomplex-mult-minus-one*: $-1 * (z::hcomplex) = -z$
by *simp*

lemma *hcomplex-mult-minus-one-right*: $(z::hcomplex) * -1 = -z$
by *simp*

lemma *hcomplex-mult-left-cancel*:
 $(c::hcomplex) \neq (0::hcomplex) \implies (c*a=c*b) = (a=b)$
by *simp*

lemma *hcomplex-mult-right-cancel*:
 $(c::hcomplex) \neq (0::hcomplex) \implies (a*c=b*c) = (a=b)$
by *simp*

9.5 Subraction and Division

lemma *hcomplex-diff-eq-eq* [simp]: $((x::hcomplex) - y = z) = (x = z + y)$
by (rule *diff-eq-eq*)

9.6 Embedding Properties for *hcomplex-of-hypreal* Map

lemma *hRe-hcomplex-of-hypreal* [simp]: $!!z. hRe(hcomplex-of-hypreal z) = z$
by *transfer* (rule *Re-complex-of-real*)

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

lemma *hcomplex-of-hypreal-epsilon-not-zero* [simp]:
 $hcomplex-of-hypreal\ epsilon \neq 0$
by (*simp add: hypreal-epsilon-not-zero*)

9.7 HComplex theorems

lemma *hRe-HComplex* [simp]: $!!x y. hRe (HComplex x y) = x$
by *transfer* (rule *Re*)

lemma *hIm-HComplex* [simp]: $!!x y. hIm (HComplex x y) = y$
by *transfer* (rule *Im*)

lemma *hcomplex-surj* [simp]: $!!z. HComplex (hRe z) (hIm z) = z$

by transfer (rule complex-surj)

lemma *hcomplex-induct* [case-names rect]:

$$(\wedge x y. P (HComplex x y)) ==> P z$$

by (rule *hcomplex-surj* [THEN subst], blast)

9.8 Modulus (Absolute Value) of Nonstandard Complex Number

lemma *hcomplex-of-hypreal-abs*:

$$hcomplex-of-hypreal (abs x) =$$

$$hcomplex-of-hypreal(hcmod(hcomplex-of-hypreal x))$$

by *simp*

lemma *HComplex-inject* [simp]:

$$!!x y x' y'. HComplex x y = HComplex x' y' = (x=x' \& y=y')$$

by transfer (rule *complex.inject*)

lemma *HComplex-add* [simp]:

$$!!x1 y1 x2 y2. HComplex x1 y1 + HComplex x2 y2 = HComplex (x1+x2) (y1+y2)$$

by transfer (rule *complex-add*)

lemma *HComplex-minus* [simp]: !!x y. $- HComplex x y = HComplex (-x) (-y)$

by transfer (rule *complex-minus*)

lemma *HComplex-diff* [simp]:

$$!!x1 y1 x2 y2. HComplex x1 y1 - HComplex x2 y2 = HComplex (x1-x2) (y1-y2)$$

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*)

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)

9.9 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 . *hcmmod* (*hcnj* z) = *hcmmod* z

by transfer (rule complex-mod-cnj)

lemma *hcomplex-hcnj-minus*: !! z . *hcnj* ($-z$) = $- hcnj z$

by transfer (rule complex-cnj-minus)

lemma *hcomplex-hcnj-inverse*: !! z . *hcnj*(*inverse* z) = *inverse*(*hcnj* z)

by transfer (rule complex-cnj-inverse)

lemma *hcomplex-hcnj-add*: !! $w z$. *hcnj*($w + z$) = *hcnj*(w) + *hcnj*(z)

by transfer (rule complex-cnj-add)

lemma *hcomplex-hcnj-diff*: !! $w z$. *hcnj*($w - z$) = *hcnj*(w) - *hcnj*(z)

by transfer (rule complex-cnj-diff)

lemma *hcomplex-hcnj-mult*: !! $w z$. *hcnj*($w * z$) = *hcnj*(w) * *hcnj*(z)

by transfer (rule complex-cnj-mult)

lemma *hcomplex-hcnj-divide*: !! $w z$. *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]: $!!z. (hcnj\ z = 0) = (z = 0)$
by *transfer* (rule *complex-cnj-zero-iff*)

lemma *hcomplex-mult-hcnj*:
 $!!z. z * hcnj\ z = hcomplex-of-hypreal\ (hRe(z) ^ 2 + hIm(z) ^ 2)$
by *transfer* (rule *complex-mult-cnj*)

9.10 More Theorems about the Function *hcmmod*

lemma *hcmmod-hcomplex-of-hypreal-of-nat* [simp]:
 $hcmmod\ (hcomplex-of-hypreal(hypreal-of-nat\ n)) = hypreal-of-nat\ n$
by *simp*

lemma *hcmmod-hcomplex-of-hypreal-of-hypnat* [simp]:
 $hcmmod\ (hcomplex-of-hypreal(hypreal-of-hypnat\ n)) = hypreal-of-hypnat\ n$
by *simp*

lemma *hcmmod-mult-hcnj*: $!!z. hcmmod(z * hcnj(z)) = hcmmod(z) ^ 2$
by *transfer* (rule *complex-mod-mult-cnj*)

lemma *hcmmod-triangle-ineq2* [simp]:
 $!!a\ b. hcmmod(b + a) - hcmmod\ b \leq hcmmod\ a$
by *transfer* (rule *complex-mod-triangle-ineq2*)

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

9.11 Exponentiation

lemma *hcomplexpow-0* [simp]: $z ^ 0 = (1::hcomplex)$
by (rule *power-0*)

lemma *hcomplexpow-Suc* [simp]: $z ^ (Suc\ n) = (z::hcomplex) * (z ^ n)$
by (rule *power-Suc*)

lemma *hcomplexpow-i-squared* [simp]: $iii ^ 2 = -1$
by *transfer* (rule *power2-i*)

lemma *hcomplex-of-hypreal-pow*:
 $!!x. hcomplex-of-hypreal\ (x ^ n) = (hcomplex-of-hypreal\ x) ^ n$
by *transfer* (rule *of-real-power*)

lemma *hcomplex-hcnj-pow*: $!!z. hcnj(z ^ n) = hcnj(z) ^ n$
by *transfer* (rule *complex-cnj-power*)

lemma *hcmmod-hcomplexpow*: $!!x. hcmmod(x ^ n) = hcmmod(x) ^ n$

by transfer (rule norm-power)

lemma hcpow-minus:

!!x n. (-x::hcomplex) pow n =
 (if (** even) n then (x pow n) else -(x pow n))

by transfer (rule neg-power-if)

lemma hcpow-mult:

!!r s n. ((r::hcomplex) * s) pow n = (r pow n) * (s pow n)

by transfer (rule power-mult-distrib)

lemma hcpow-zero2 [simp]:

$\bigwedge n. 0 \text{ pow } (hSuc\ n) = (0::'a::\{\text{power, semiring-0}\} \text{ star})$

by transfer (rule power-0-Suc)

lemma hcpow-not-zero [simp,intro]:

!!r n. r \neq 0 \implies r pow n \neq (0::hcomplex)

by (rule hyperpow-not-zero)

lemma hcpow-zero-zero: r pow n = (0::hcomplex) \implies r = 0

by (blast intro: ccontr dest: hcpow-not-zero)

9.12 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: !!z. hsgn (-z) = - hsgn(z)

by transfer (rule sgn-minus)

lemma hsgn-eq: !!z. hsgn z = z / hcomplex-of-hypreal (hcmmod z)

by transfer (rule sgn-eq)

lemma hcmmod-i: !!x y. hcmmod (HComplex x y) = (** sqrt) (x ^ 2 + y ^ 2)

by transfer (rule complex-norm)

lemma hcomplex-eq-cancel-iff1 [simp]:

(hcomplex-of-hypreal xa = HComplex x y) = (xa = x & y = 0)

by (simp add: hcomplex-of-hypreal-eq)

lemma hcomplex-eq-cancel-iff2 [simp]:

(HComplex x y = hcomplex-of-hypreal xa) = (x = xa & y = 0)

by (simp add: hcomplex-of-hypreal-eq)

lemma HComplex-eq-0 [simp]: !!x y. (HComplex x y = 0) = (x = 0 & y = 0)

by transfer (rule Complex-eq-0)

lemma *HComplex-eq-1* [simp]: $!!x y. (HComplex\ x\ y = 1) = (x = 1 \ \&\ y = 0)$
by *transfer* (rule *Complex-eq-1*)

lemma *i-eq-HComplex-0-1*: $iii = HComplex\ 0\ 1$
by *transfer* (rule *i-def* [THEN *meta-eq-to-obj-eq*])

lemma *HComplex-eq-i* [simp]: $!!x y. (HComplex\ x\ y = iii) = (x = 0 \ \&\ y = 1)$
by *transfer* (rule *Complex-eq-i*)

lemma *hRe-hsgn* [simp]: $!!z. hRe(hsgn\ z) = hRe(z)/hcm\ o\ d\ z$
by *transfer* (rule *Re-sgn*)

lemma *hIm-hsgn* [simp]: $!!z. hIm(hsgn\ z) = hIm(z)/hcm\ o\ d\ z$
by *transfer* (rule *Im-sgn*)

lemma *HComplex-inverse*:
 $!!x y. inverse\ (HComplex\ x\ y) =$
 $HComplex\ (x/(x^2 + y^2))\ (-y/(x^2 + y^2))$
by *transfer* (rule *complex-inverse*)

lemma *hRe-mult-i-eq*[simp]:
 $!!y. hRe\ (iii * hcomplex-of-hypreal\ y) = 0$
by *transfer simp*

lemma *hIm-mult-i-eq* [simp]:
 $!!y. hIm\ (iii * hcomplex-of-hypreal\ y) = y$
by *transfer simp*

lemma *hcm\ o\ d-mult-i* [simp]: $!!y. hcm\ o\ d\ (iii * hcomplex-of-hypreal\ y) = abs\ y$
by *transfer simp*

lemma *hcm\ o\ d-mult-i2* [simp]: $!!y. hcm\ o\ d\ (hcomplex-of-hypreal\ y * iii) = abs\ y$
by *transfer simp*

lemma *cos-harg-i-mult-zero* [simp]:
 $!!y. y \neq 0 ==> (*f* cos)\ (harg(HComplex\ 0\ y)) = 0$
by *transfer* (rule *cos-arg-i-mult-zero*)

lemma *hcomplex-of-hypreal-zero-iff* [simp]:
 $!!y. (hcomplex-of-hypreal\ y = 0) = (y = 0)$
by *transfer* (rule *of-real-eq-0-iff*)

9.13 Polar Form for Nonstandard Complex Numbers

lemma *complex-split-polar2*:

$$\forall n. \exists r a. (z\ n) = \text{complex-of-real } r * (\text{Complex } (\cos\ a) (\sin\ a))$$

by (*blast intro: complex-split-polar*)

lemma *hcomplex-split-polar*:

$$!!z. \exists r a. z = \text{hcomplex-of-hypreal } r * (\text{HComplex}((*f* \cos)\ a)((*f* \sin)\ a))$$

by *transfer (rule complex-split-polar)*

lemma *hcis-eq*:

$$!!a. \text{hcis } a =$$

$$(\text{hcomplex-of-hypreal}((*f* \cos)\ a) + \\ \text{iii} * \text{hcomplex-of-hypreal}((*f* \sin)\ a))$$

by *transfer (simp add: cis-def)*

lemma *hrcis-Ex*: $!!z. \exists r a. z = \text{hrcis } r\ a$

by *transfer (rule rcis-Ex)*

lemma *hRe-hcomplex-polar [simp]*:

$$!!r\ a. \text{hRe } (\text{hcomplex-of-hypreal } r * \text{HComplex} ((*f* \cos)\ a) ((*f* \sin)\ a)) = \\ r * (*f* \cos)\ a$$

by *transfer simp*

lemma *hRe-hrcis [simp]*: $!!r\ a. \text{hRe}(\text{hrcis } r\ a) = r * (*f* \cos)\ a$

by *transfer (rule Re-rcis)*

lemma *hIm-hcomplex-polar [simp]*:

$$!!r\ a. \text{hIm } (\text{hcomplex-of-hypreal } r * \text{HComplex} ((*f* \cos)\ a) ((*f* \sin)\ a)) = \\ r * (*f* \sin)\ a$$

by *transfer simp*

lemma *hIm-hrcis [simp]*: $!!r\ a. \text{hIm}(\text{hrcis } r\ a) = r * (*f* \sin)\ a$

by *transfer (rule Im-rcis)*

lemma *hcmmod-unit-one [simp]*:

$$!!a. \text{hcmmod } (\text{HComplex} ((*f* \cos)\ a) ((*f* \sin)\ a)) = 1$$

by *transfer (rule cmmod-unit-one)*

lemma *hcmmod-complex-polar [simp]*:

$$!!r\ a. \text{hcmmod } (\text{hcomplex-of-hypreal } r * \text{HComplex} ((*f* \cos)\ a) ((*f* \sin)\ a)) = \\ \text{abs } r$$

by *transfer (rule cmmod-complex-polar)*

lemma *hcmmod-hrcis [simp]*: $!!r\ a. \text{hcmmod}(\text{hrcis } r\ a) = \text{abs } r$

by *transfer (rule complex-mod-rcis)*

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

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

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

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

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

lemma *hrcis-zero-arg [simp]*: !!*r*. *hrcis r 0 = hcomplex-of-hypreal r*
by *transfer (rule rcis-zero-arg)*

lemma *hcomplex-i-mult-minus [simp]*: !!*x*. *iii * (iii * x) = - x*
by *transfer (rule complex-i-mult-minus)*

lemma *hcomplex-i-mult-minus2 [simp]*: *iii * iii * x = - x*
by *simp*

lemma *hcis-hypreal-of-nat-Suc-mult*:
 !!*a*. *hcis (hypreal-of-nat (Suc n) * a) =*
 *hcis a * hcis (hypreal-of-nat n * a)*
apply *transfer*
apply (*simp add: left-distrib cis-mult*)
done

lemma *NSDeMoivre*: !!*a*. *(hcis a) ^ n = hcis (hypreal-of-nat n * a)*
apply *transfer*
apply (*fold real-of-nat-def*)
apply (*rule DeMoivre*)
done

lemma *hcis-hypreal-of-hypnat-Suc-mult*:
 !! *a n*. *hcis (hypreal-of-hypnat (n + 1) * a) =*
 *hcis a * hcis (hypreal-of-hypnat n * a)*
by *transfer (simp add: left-distrib cis-mult)*

lemma *NSDeMoivre-ext*:
 !!*a n*. *(hcis a) pow n = hcis (hypreal-of-hypnat n * a)*
by *transfer (fold real-of-nat-def, rule DeMoivre)*

lemma *NSDeMoivre2*:

!! $a r. (\text{hrcis } r \ a) \ ^{\wedge} n = \text{hrcis } (r \ ^{\wedge} n) \ (\text{hypreal-of-nat } n \ * \ a)$

by *transfer* (*fold real-of-nat-def*, *rule DeMoivre2*)

lemma *DeMoivre2-ext*:

!! $a r n. (\text{hrcis } r \ a) \ \text{pow } n = \text{hrcis } (r \ \text{pow } n) \ (\text{hypreal-of-hypnat } n \ * \ a)$

by *transfer* (*fold real-of-nat-def*, *rule DeMoivre2*)

lemma *hcis-inverse* [*simp*]: !! $a. \text{inverse}(\text{hcis } a) = \text{hcis } (-a)$

by *transfer* (*rule cis-inverse*)

lemma *hrcis-inverse*: !! $a r. \text{inverse}(\text{hrcis } r \ a) = \text{hrcis } (\text{inverse } r) \ (-a)$

by *transfer* (*simp add: rcis-inverse inverse-eq-divide* [*symmetric*])

lemma *hRe-hcis* [*simp*]: !! $a. \text{hRe}(\text{hcis } a) = (*f* \ \text{cos}) \ a$

by *transfer* (*rule Re-cis*)

lemma *hIm-hcis* [*simp*]: !! $a. \text{hIm}(\text{hcis } a) = (*f* \ \text{sin}) \ a$

by *transfer* (*rule Im-cis*)

lemma *cos-n-hRe-hcis-pow-n*: $(*f* \ \text{cos}) \ (\text{hypreal-of-nat } n \ * \ a) = \text{hRe}(\text{hcis } a \ ^{\wedge} n)$

by (*simp add: NSDeMoivre*)

lemma *sin-n-hIm-hcis-pow-n*: $(*f* \ \text{sin}) \ (\text{hypreal-of-nat } n \ * \ a) = \text{hIm}(\text{hcis } a \ ^{\wedge} n)$

by (*simp add: NSDeMoivre*)

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

by (*simp add: NSDeMoivre-ext*)

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

by (*simp add: NSDeMoivre-ext*)

lemma *hexpi-add*: !! $a b. \text{hexpi}(a + b) = \text{hexpi}(a) \ * \ \text{hexpi}(b)$

by *transfer* (*rule exp-add*)

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

lemma *inj-hcomplex-of-complex*: *inj*(*hcomplex-of-complex*)

by (*rule inj-star-of*)

lemma *hcomplex-of-complex-i*: *iii* = *hcomplex-of-complex ii*

by (*rule iii-def*)

lemma *hRe-hcomplex-of-complex*:

$\text{hRe} \ (\text{hcomplex-of-complex } z) = \text{hypreal-of-real} \ (\text{Re } z)$

by *transfer* (rule *refl*)

lemma *hIm-hcomplex-of-complex*:

$$hIm (hcomplex-of-complex z) = hypreal-of-real (Im z)$$

by *transfer* (rule *refl*)

lemma *hcmmod-hcomplex-of-complex*:

$$hcmmod (hcomplex-of-complex x) = hypreal-of-real (cmmod x)$$

by *transfer* (rule *refl*)

9.15 Numerals and Arithmetic

lemma *hcomplex-of-hypreal-eq-hcomplex-of-complex*:

$$\begin{aligned} hcomplex-of-hypreal (hypreal-of-real x) = \\ hcomplex-of-complex (complex-of-real x) \end{aligned}$$

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*:

$$hcomplex-of-complex (neg-numeral w) = hcomplex-of-hypreal (neg-numeral w)$$

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-hcmmod* [*simp*]:

$$hcmmod (numeral v :: hcomplex) = (numeral v :: hypreal)$$

by *transfer* (rule *norm-numeral*)

lemma *hcomplex-neg-numeral-hcmmod* [*simp*]:

$$hcmmod (neg-numeral v :: hcomplex) = (numeral v :: hypreal)$$

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

10 Star: Star-Transforms in Non-Standard Analysis

```
theory Star
imports NSA
begin
```

definition

```
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. ∃ As. X = *sn* As}
```

definition

```
is-starext :: ['a star => 'a star, 'a => 'a] => bool where
is-starext F f = (∀ x y. ∃ X ∈ Rep-star(x). ∃ Y ∈ Rep-star(y).
  ((y = (F x)) = ({n. Y n = f(X n)} : FreeUltrafilterNat)))
```

definition

```
starfun-n :: (nat => ('a => 'b)) => 'a star => 'b star (*fn* - [80] 80) where
*fn* F = Ifun (star-n F)
```

definition

```
InternalFuns :: ('a star => 'b star) set where
InternalFuns = {X. ∃ F. X = *fn* F}
```

```
lemma no-choice: ∀ x. ∃ y. Q x y ==> ∃ (f :: 'a => nat). ∀ x. Q x (f x)
apply (rule-tac x = %x. LEAST y. Q x y in exI)
apply (blast intro: LeastI)
done
```

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

```
lemma STAR-star-of-image-subset: star-of 'A <= *s* A
by auto
```

```
lemma STAR-hypreal-of-real-Int: *s* X Int Reals = hypreal-of-real 'X
by (auto simp add: SReal-def)
```

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

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

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

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

lemma *STAR-real-seq-to-hypreal*:
 $\forall n. (X n) \notin M \implies \text{star-n } X \notin *s* M$
apply (*unfold starset-def star-of-def*)
apply (*simp add: Iset-star-n*)
done

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

lemma *STAR-not-mem*: $x \notin F \implies \text{star-of } x \notin *s* F$
by *transfer*

lemma *STAR-subset-closed*: $[\![x : *s* A; A \leq B]\!] \implies x : *s* B$
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*: $\forall n. (As n = A) \implies *sn* As = *s* A$
apply (*drule fun-eq-iff [THEN iffD2]*)
apply (*simp add: starset-n-def starset-def star-of-def*)
done

lemma *starfun-n-starfun*: $\forall n. (F n = f) \implies *fn* F = *f* f$
apply (*drule fun-eq-iff [THEN iffD2]*)
apply (*simp add: starfun-n-def starfun-def star-of-def*)
done

```

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 ) (star-n X) = star-n (%n. f (X n))
by (rule starfun-star-n)

```

```

lemma starfun-if-eq:
  !!w. w  $\neq$  star-of x
  ==> (  $*f*$  ( $\lambda z$ . if z = x then a else g z) ) w = (  $*f*$  g ) w
by (transfer, simp)

```

```

lemma starfun-mult: !!x. (  $*f*$  f ) x * (  $*f*$  g ) x = (  $*f*$  (%x. f x * g x) ) x
by (transfer, rule refl)
declare starfun-mult [symmetric, simp]

```

```

lemma starfun-add: !!x. (  $*f*$  f ) x + (  $*f*$  g ) x = (  $*f*$  (%x. f x + g x) ) x
by (transfer, rule refl)
declare starfun-add [symmetric, simp]

```

```

lemma starfun-minus: !!x. - (  $*f*$  f ) x = (  $*f*$  (%x. - f x) ) x
by (transfer, rule refl)
declare starfun-minus [symmetric, simp]

```

```

lemma starfun-add-minus: !!x. (  $*f*$  f ) x + - (  $*f*$  g ) x = (  $*f*$  (%x. f x + -g x) ) x
by (transfer, rule refl)
declare starfun-add-minus [symmetric, simp]

```

```

lemma starfun-diff: !!x. (  $*f*$  f ) x - (  $*f*$  g ) x = (  $*f*$  (%x. f x - g x) ) x
by (transfer, rule refl)

```

declare *starfun-diff* [*symmetric*, *simp*]

lemma *starfun-o2*: $(\%x. (*f* f) ((*f* g) x)) = *f* (\%x. f (g x))$
by (*transfer*, *rule refl*)

lemma *starfun-o*: $(*f* f) o (*f* g) = (*f* (f o g))$
by (*transfer o-def*, *rule refl*)

NS extension of constant function

lemma *starfun-const-fun* [*simp*]: $!!x. (*f* (\%x. k)) x = \text{star-of } k$
by (*transfer*, *rule refl*)

the NS extension of the identity function

lemma *starfun-Id* [*simp*]: $!!x. (*f* (\%x. x)) x = x$
by (*transfer*, *rule refl*)

lemma *starfun-Idfun-approx*:

$x @= \text{star-of } a ==> (*f* (\%x. x)) x @= \text{star-of } a$
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 (*simp add: is-starext-def*, *auto*)
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 add: starfun star-n-eq-iff)

done

Any nonstandard extension is in fact the Star-function

lemma *is-starfun-starext*: *is-starext* $F f ==> F = *f* f$
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*: $(\text{is-starext } F f) = (F = *f* f)$
by (*blast intro: is-starfun-starext is-starext-starfun*)

extended function has same solution as its standard version for real arguments. i.e they are the same for all real arguments

lemma *starfun-eq*: $(** f) (star-of a) = star-of (f a)$
by (*rule starfun-star-of*)

lemma *starfun-approx*: $(** f) (star-of a) @= star-of (f a)$
by *simp*

lemma *starfun-lambda-cancel*:
 $!!x'. (** (%h. f (x + h))) x' = (** f) (star-of x + x')$
by (*transfer, rule refl*)

lemma *starfun-lambda-cancel2*:
 $(** (%h. f(g(x + h)))) x' = (** (f o g)) (star-of x + x')$
by (*unfold o-def, rule starfun-lambda-cancel*)

lemma *starfun-mult-HFinite-approx*:
fixes $l m :: 'a::real-normed-algebra$ *star*
shows $[| (** f) x @= l; (** g) x @= m;$
 $l: HFinite; m: HFinite$
 $|] ==> (** (%x. f x * g x)) x @= l * m$
apply (*drule (3) approx-mult-HFinite*)
apply (*auto intro: approx-HFinite [OF - approx-sym]*)
done

lemma *starfun-add-approx*: $[| (** f) x @= l; (** g) x @= m$
 $|] ==> (** (%x. f x + g x)) x @= l + m$
by (*auto intro: approx-add*)

Examples: *hrabs* is nonstandard extension of *rabs* inverse is nonstandard extension of inverse

lemma *starfun-rabs-hrabs*: $** abs = abs$
by (*simp only: star-abs-def*)

lemma *starfun-inverse-inverse* [*simp*]: $(** inverse) x = inverse(x)$
by (*simp only: star-inverse-def*)

lemma *starfun-inverse*: $!!x. inverse ((** f) x) = (** (%x. inverse (f x))) x$
by (*transfer, rule refl*)
declare *starfun-inverse* [*symmetric, simp*]

lemma *starfun-divide*: $!!x. (** f) x / (** g) x = (** (%x. f x / g x)) x$
by (*transfer, rule refl*)
declare *starfun-divide* [*symmetric, simp*]

lemma *starfun-inverse2*: $!!x. inverse ((** f) x) = (** (%x. inverse (f x))) x$
by (*transfer, rule refl*)

General lemma/theorem needed for proofs in elementary topology of the reals

lemma *starfun-mem-starset*:

$!!x. (*f* f) x : *s* A ==> x : *s* \{x. f x \in A\}$

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*:

$abs (star-n X) = star-n (\%n. abs (X n))$

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 extension. See second NS set extension below.

lemma *STAR-rabs-add-minus*:

$*s* \{x. abs (x + - y) < r\} =$
 $\{x. abs(x + -star-of y) < star-of r\}$

by (*transfer, rule refl*)

lemma *STAR-starfun-rabs-add-minus*:

$*s* \{x. abs (f x + - y) < r\} =$
 $\{x. abs((*f* f) x + -star-of y) < star-of r\}$

by (*transfer, rule refl*)

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

lemma *Infinitesimal-FreeUltrafilterNat-iff2*:

$(star-n X \in Infinitesimal) =$
 $(\forall m. \{n. norm(X n) < inverse(real(Suc m))\}$
 $\in FreeUltrafilterNat)$

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 real-of-nat-def)

lemma *HNatInfinite-inverse-Infinitesimal [simp]*:

$n \in HNatInfinite ==> inverse (hypreal-of-hypnat n) \in Infinitesimal$

apply (*cases n*)

apply (*auto simp add: of-hypnat-def starfun-star-n real-of-nat-def [symmetric]*
star-n-inverse real-norm-def)

HNatInfinite-FreeUltrafilterNat-iff
Infinitesimal-FreeUltrafilterNat-iff2)

apply (*drule-tac x=Suc m in spec*)

apply (*erule ultra, simp*)

done

lemma *approx-FreeUltrafilterNat-iff*: $star-n X @= star-n Y =$

$(\forall r>0. \{n. norm (X n - Y n) < r\} : FreeUltrafilterNat)$

```

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. {n. norm (X n - Y n) <
    inverse(real(Suc m))}) : FreeUltrafilterNat)
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 (%n. xa) in fun-cong)
apply (auto simp add: starfun star-n-eq-iff)
done

end

```

11 NatStar: 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* (%n. (A n) Un (B n)) = *sn* A Un *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 ∈ InternalSets; Y ∈ InternalSets |]
  ==> (X Un Y) ∈ InternalSets
by (auto simp add: InternalSets-def starset-n-Un [symmetric])

lemma starset-n-Int:
  *sn* (%n. (A n) Int (B n)) = *sn* A Int *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*:
 $\llbracket X \in \text{InternalSets}; Y \in \text{InternalSets} \rrbracket$
 $\implies (X \text{ Int } Y) \in \text{InternalSets}$
by (*auto simp add: InternalSets-def starset-n-Int [symmetric]*)

lemma *starset-n-Compl*: $*sn* (\%n. - A \ n) = -(*sn* A)$
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*: $X \in \text{InternalSets} \implies -X \in \text{InternalSets}$
by (*auto simp add: InternalSets-def starset-n-Compl [symmetric]*)

lemma *starset-n-diff*: $*sn* (\%n. (A \ n) - (B \ n)) = *sn* A - *sn* B$
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*:
 $\llbracket X \in \text{InternalSets}; Y \in \text{InternalSets} \rrbracket$
 $\implies (X - Y) \in \text{InternalSets}$
by (*auto simp add: InternalSets-def starset-n-diff [symmetric]*)

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

lemma *NatStar-hypreal-of-real-Int*:
 $*s* X \text{ Int } \text{Nats} = \text{hypnat-of-nat } X$
by (*auto simp add: SHNat-eq*)

lemma *starset-starset-n-eq*: $*s* X = *sn* (\%n. X)$
by (*simp add: starset-n-starset*)

lemma *InternalSets-starset-n [simp]*: $(*s* X) \in \text{InternalSets}$
by (*auto simp add: InternalSets-def starset-starset-n-eq*)

lemma *InternalSets-UNIV-diff*:
 $X \in \text{InternalSets} \implies \text{UNIV} - X \in \text{InternalSets}$
apply (*subgoal-tac* $\text{UNIV} - X = - X$)
by (*auto intro: InternalSets-Compl*)

11.1 Nonstandard Extensions of Functions

Example of transfer of a property from reals to hyperreals — used for limit comparison of sequences

lemma *starfun-le-mono*:

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

lemma *starfun-less-mono*:

$\forall n. N \leq n \dashrightarrow f n < g n$
 $\implies \forall n. \text{hypnat-of-nat } N \leq n \dashrightarrow (*f* f) n < (*f* g) n$
by *transfer*

Nonstandard extension when we increment the argument by one

lemma *starfun-shift-one*:

$!!N. (*f* (\%n. f (Suc n))) N = (*f* f) (N + (1::\text{hypnat}))$
by (*transfer, simp*)

Nonstandard extension with absolute value

lemma *starfun-abs*: $!!N. (*f* (\%n. \text{abs } (f n))) N = \text{abs}((*f* f) N)$
by (*transfer, rule refl*)

The hyperpow function as a nonstandard extension of realpow

lemma *starfun-pow*: $!!N. (*f* (\%n. r ^ n)) N = (\text{hypreal-of-real } r) \text{ pow } N$
by (*transfer, rule refl*)

lemma *starfun-pow2*:

$!!N. (*f* (\%n. (X n) ^ m)) N = (*f* X) N \text{ pow hypnat-of-nat } m$
by (*transfer, rule refl*)

lemma *starfun-pow3*: $!!R. (*f* (\%r. r ^ n)) R = (R) \text{ pow hypnat-of-nat } n$
by (*transfer, rule refl*)

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

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

lemma *starfun-inverse-real-of-nat-eq*:

$N \in \text{HNatInfinite}$
 $\implies (*f* (\%x::\text{nat. inverse}(\text{real } x))) N = \text{inverse}(\text{hypreal-of-hypnat } N)$
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 starfun-inverse-inverse*)
done

Internal functions - some redundancy with *f* now

lemma *starfun-n*: $(*fn* f) (\text{star-n } X) = \text{star-n } (\%n. f n (X n))$
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* (%i x. f i x * g i x)) z
apply (cases z)
apply (simp add: starfun-n star-n-mult)
done

```

Addition: $(*fn) + (*gn) = *(fn + gn)$

```

lemma starfun-n-add:
    (*fn* f) z + (*fn* g) z = (*fn* (%i x. f i x + g i x)) z
apply (cases z)
apply (simp add: starfun-n star-n-add)
done

```

Subtraction: $(*fn) - (*gn) = *(fn + - gn)$

```

lemma starfun-n-add-minus:
    (*fn* f) z + -( *fn* g) z = (*fn* (%i x. f i x + -g i x)) z
apply (cases z)
apply (simp add: starfun-n star-n-minus star-n-add)
done

```

Composition: $(*fn) o (*gn) = *(fn o gn)$

```

lemma starfun-n-const-fun [simp]:
    (*fn* (%i x. k)) z = star-of k
apply (cases z)
apply (simp add: starfun-n star-of-def)
done

```

```

lemma starfun-n-minus: - (*fn* f) x = (*fn* (%i x. - (f i) x)) x
apply (cases x)
apply (simp add: starfun-n star-n-minus)
done

```

```

lemma starfun-n-eq [simp]:
    (*fn* f) (star-of n) = star-n (%i. f i n)
by (simp add: starfun-n star-of-def)

```

```

lemma starfun-eq-iff: (( *f* f) = ( *f* g)) = (f = g)
by (transfer, rule refl)

```

```

lemma starfunNat-inverse-real-of-nat-Infinitesimal [simp]:
    N ∈ HNatInfinite ==> (*f* (%x. inverse (real x))) N ∈ 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

```

11.2 Nonstandard Characterization of Induction

```

lemma hypnat-induct-obj:
    !!n. (( *p* P) (0::hypnat) &

```

$$(\forall n. (*p* P)(n) \dashrightarrow (*p* P)(n + 1)))$$

$$\dashrightarrow (*p* P)(n)$$

by (*transfer, induct-tac n, auto*)

lemma *hypnat-induct*:

$$!!n. [| (*p* P) (0::hypnat);$$

$$!!n. (*p* P)(n) ==> (*p* P)(n + 1)|]$$

$$==> (*p* P)(n)$$

by (*transfer, induct-tac n, auto*)

lemma *starP2-eq-iff*: ($*p2*$ ($op =$)) = ($op =$)

by *transfer (rule refl)*

lemma *starP2-eq-iff2*: ($*p2*$ ($\%x y. x = y$)) $X Y = (X = Y)$

by (*simp add: starP2-eq-iff*)

lemma *nonempty-nat-set-Least-mem*:

$$c \in (S :: \text{nat set}) ==> (\text{LEAST } n. n \in S) \in S$$

by (*erule LeastI*)

lemma *nonempty-set-star-has-least*:

$$!!S::\text{nat set star. Iset } S \neq \{\} ==> \exists n \in \text{Iset } S. \forall m \in \text{Iset } S. n \leq m$$

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*:

$$[| (S::hypnat set) \in \text{InternalSets}; S \neq \{\} |] ==> \exists n \in S. \forall m \in S. n \leq m$$

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*:

$$!!X::\text{nat set star. [| (0::hypnat) \in \text{Iset } X; \forall n. n \in \text{Iset } X \dashrightarrow n + 1 \in \text{Iset } X |]$$

$$==> \text{Iset } X = (\text{UNIV}::\text{hypnat set})$$

apply (*transfer UNIV-def*)

apply (*rule equalityI [OF subset-UNIV subsetI]*)

apply (*induct-tac x, auto*)

done

lemma *internal-induct*:

$$[| X \in \text{InternalSets}; (0::hypnat) \in X; \forall n. n \in X \dashrightarrow n + 1 \in X |]$$

$$==> X = (\text{UNIV}::\text{hypnat set})$$

apply (*clarsimp simp add: InternalSets-def starset-n-def*)

apply (*erule (1) internal-induct-lemma*)

done

end

12 HSEQ: Sequences and Convergence (Nonstandard)

theory *HSEQ*
imports *SEQ NatStar*
begin

definition

NSLIMSEQ :: $[nat \Rightarrow 'a::real\text{-normed}\text{-vector}, 'a] \Rightarrow bool$
 (((-)/ -----NS> (-)) [60, 60] 60) **where**
 — Nonstandard definition of convergence of sequence
 $X \text{ -----NS> } L = (\forall N \in HNatInfinite. (*f* X) N \approx star\text{-of } L)$

definition

nslim :: $(nat \Rightarrow 'a::real\text{-normed}\text{-vector}) \Rightarrow 'a$ **where**
 — Nonstandard definition of limit using choice operator
 $nslim X = (THE L. X \text{ -----NS> } L)$

definition

NSconvergent :: $(nat \Rightarrow 'a::real\text{-normed}\text{-vector}) \Rightarrow bool$ **where**
 — Nonstandard definition of convergence
 $NSconvergent X = (\exists L. X \text{ -----NS> } L)$

definition

NSBseq :: $(nat \Rightarrow 'a::real\text{-normed}\text{-vector}) \Rightarrow bool$ **where**
 — Nonstandard definition for bounded sequence
 $NSBseq X = (\forall N \in HNatInfinite. (*f* X) N : HFinite)$

definition

NSCauchy :: $(nat \Rightarrow 'a::real\text{-normed}\text{-vector}) \Rightarrow bool$ **where**
 — Nonstandard definition
 $NSCauchy X = (\forall M \in HNatInfinite. \forall N \in HNatInfinite. (*f* X) M \approx (*f* X) N)$

12.1 Limits of Sequences

lemma *NSLIMSEQ-iff*:

$(X \text{ -----NS> } L) = (\forall N \in HNatInfinite. (*f* X) N \approx star\text{-of } L)$

by (*simp add: NSLIMSEQ-def*)

lemma *NSLIMSEQ-I*:

$(\bigwedge N. N \in HNatInfinite \implies starfun X N \approx star\text{-of } L) \implies X \text{ -----NS> } L$

by (simp add: NSLIMSEQ-def)

lemma NSLIMSEQ-D:

$\llbracket X \text{ ----NS} > L; N \in \text{HNatInfinite} \rrbracket \implies \text{starfun } X \ N \approx \text{star-of } L$
 by (simp add: NSLIMSEQ-def)

lemma NSLIMSEQ-const: $(\%n. k) \text{ ----NS} > k$

by (simp add: NSLIMSEQ-def)

lemma NSLIMSEQ-add:

$\llbracket X \text{ ----NS} > a; Y \text{ ----NS} > b \rrbracket \implies (\%n. X \ n + Y \ n) \text{ ----NS} > a + b$

by (auto intro: approx-add simp add: NSLIMSEQ-def starfun-add [symmetric])

lemma NSLIMSEQ-add-const: $f \text{ ----NS} > a \implies (\%n.(f \ n + b)) \text{ ----NS} > a + b$

by (simp only: NSLIMSEQ-add NSLIMSEQ-const)

lemma NSLIMSEQ-mult:

fixes $a \ b :: 'a::\text{real-normed-algebra}$

shows $\llbracket X \text{ ----NS} > a; Y \text{ ----NS} > b \rrbracket \implies (\%n. X \ n * Y \ n) \text{ ----NS} > a * b$

by (auto intro!: approx-mult-HFinite simp add: NSLIMSEQ-def)

lemma NSLIMSEQ-minus: $X \text{ ----NS} > a \implies (\%n. -(X \ n)) \text{ ----NS} > -a$

by (auto simp add: NSLIMSEQ-def)

lemma NSLIMSEQ-minus-cancel: $(\%n. -(X \ n)) \text{ ----NS} > -a \implies X \text{ ----NS} > a$

by (drule NSLIMSEQ-minus, simp)

lemma NSLIMSEQ-add-minus:

$\llbracket X \text{ ----NS} > a; Y \text{ ----NS} > b \rrbracket \implies (\%n. X \ n + -Y \ n) \text{ ----NS} > a + -b$

by (simp add: NSLIMSEQ-add NSLIMSEQ-minus)

lemma NSLIMSEQ-diff:

$\llbracket X \text{ ----NS} > a; Y \text{ ----NS} > b \rrbracket \implies (\%n. X \ n - Y \ n) \text{ ----NS} > a - b$

by (simp add: diff-minus NSLIMSEQ-add NSLIMSEQ-minus)

lemma NSLIMSEQ-diff-const: $f \text{ ----NS} > a \implies (\%n.(f \ n - b)) \text{ ----NS} > a - b$

by (simp add: NSLIMSEQ-diff NSLIMSEQ-const)

lemma NSLIMSEQ-inverse:

fixes $a :: 'a::\text{real-normed-div-algebra}$

shows $\llbracket X \text{ ----NS} > a; a \sim 0 \rrbracket \implies (\%n. \text{inverse}(X \ n)) \text{ ----NS} >$

inverse(*a*)

by (*simp add: NSLIMSEQ-def star-of-approx-inverse*)

lemma *NSLIMSEQ-mult-inverse*:

fixes *a b* :: 'a::real-normed-field

shows

$\llbracket X \text{ ----NS} > a; Y \text{ ----NS} > b; b \sim 0 \rrbracket \implies (\%n. X\ n / Y\ n) \text{ ----NS} > a/b$

by (*simp add: NSLIMSEQ-mult NSLIMSEQ-inverse divide-inverse*)

lemma *starfun-hnorm*: $\bigwedge x. \text{hnorm} ((\ast f \ast f) x) = (\ast f \ast (\lambda x. \text{norm} (f x))) x$

by *transfer simp*

lemma *NSLIMSEQ-norm*: $X \text{ ----NS} > a \implies (\lambda n. \text{norm} (X\ n)) \text{ ----NS} > \text{norm } a$

by (*simp add: NSLIMSEQ-def starfun-hnorm [symmetric] approx-hnorm*)

Uniqueness of limit

lemma *NSLIMSEQ-unique*: $\llbracket X \text{ ----NS} > a; X \text{ ----NS} > b \rrbracket \implies a = b$

apply (*simp add: NSLIMSEQ-def*)

apply (*drule HNatInfinite-whn [THEN [2] bspec]*)**+**

apply (*auto dest: approx-trans3*)

done

lemma *NSLIMSEQ-pow [rule-format]*:

fixes *a* :: 'a::{real-normed-algebra,power}

shows $(X \text{ ----NS} > a) \text{ -->} ((\%n. (X\ n) ^ m) \text{ ----NS} > a ^ m)$

apply (*induct m*)

apply (*auto simp add: power-Suc intro: NSLIMSEQ-mult NSLIMSEQ-const*)

done

We can now try and derive a few properties of sequences, starting with the limit comparison property for sequences.

lemma *NSLIMSEQ-le*:

$\llbracket f \text{ ----NS} > l; g \text{ ----NS} > m;$

$\exists N. \forall n \geq N. f(n) \leq g(n)$

$\rrbracket \implies l \leq (m::real)$

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-Infinitesimal-cancel2*)

done

lemma *NSLIMSEQ-le-const*: $\llbracket X \text{ ----NS} > (r::real); \forall n. a \leq X\ n \rrbracket \implies a \leq r$

by (*erule NSLIMSEQ-le [OF NSLIMSEQ-const], auto*)

lemma *NSLIMSEQ-le-const2*: $[| X \text{ ----NS} \rangle (r::\text{real}); \forall n. X n \leq a |] \implies r \leq a$
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*: $f \text{ ----NS} \rangle l \implies (\%n. f(\text{Suc } n)) \text{ ----NS} \rangle l$
apply (*unfold NSLIMSEQ-def, safe*)
apply (*drule-tac x=N + 1 in bspec*)
apply (*erule HNatInfinite-add*)
apply (*simp add: starfun-shift-one*)
done

lemma *NSLIMSEQ-imp-Suc*: $(\%n. f(\text{Suc } n)) \text{ ----NS} \rangle l \implies f \text{ ----NS} \rangle l$
apply (*unfold NSLIMSEQ-def, 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*: $((\%n. f(\text{Suc } n)) \text{ ----NS} \rangle l) = (f \text{ ----NS} \rangle l)$
by (*blast intro: NSLIMSEQ-imp-Suc NSLIMSEQ-Suc*)

12.1.1 Equivalence of LIMSEQ and NSLIMSEQ

lemma *LIMSEQ-NSLIMSEQ*:
assumes $X: X \text{ ----} \rangle L$ **shows** $X \text{ ----NS} \rangle L$
proof (*rule NSLIMSEQ-I*)
fix N **assume** $N: N \in \text{HNatInfinite}$
have $\text{starfun } X N - \text{star-of } L \in \text{Infinitesimal}$
proof (*rule InfinitesimalI2*)
fix $r::\text{real}$ **assume** $r: 0 < r$
from *LIMSEQ-D [OF X r]*
obtain no **where** $\forall n \geq no. \text{norm } (X n - L) < r ..$
hence $\forall n \geq \text{star-of } no. \text{hnorm } (\text{starfun } X n - \text{star-of } L) < \text{star-of } r$
by *transfer*
thus $\text{hnorm } (\text{starfun } X N - \text{star-of } L) < \text{star-of } r$
using N **by** (*simp add: star-of-le-HNatInfinite*)
qed
thus $\text{starfun } X N \approx \text{star-of } L$
by (*unfold approx-def*)
qed

lemma *NSLIMSEQ-LIMSEQ*:
assumes $X: X \text{ ----NS} \rangle L$ **shows** $X \text{ ----} \rangle L$
proof (*rule LIMSEQ-I*)

```

fix  $r::real$  assume  $r: 0 < r$ 
have  $\exists no. \forall n \geq no. hnorm (starfun X n - star-of L) < star-of r$ 
proof (intro exI allI impI)
  fix  $n$  assume  $whn \leq n$ 
  with  $HNatInfinite-whn$  have  $n \in HNatInfinite$ 
  by (rule  $HNatInfinite-upward-closed$ )
  with  $X$  have  $starfun X n \approx star-of L$ 
  by (rule  $NSLIMSEQ-D$ )
  hence  $starfun X n - star-of L \in Infinitesimal$ 
  by (unfold  $approx-def$ )
  thus  $hnorm (starfun X n - star-of L) < star-of r$ 
  using  $r$  by (rule  $InfinitesimalD2$ )
qed
thus  $\exists no. \forall n \geq no. norm (X n - L) < r$ 
by transfer
qed

```

```

theorem  $LIMSEQ-NSLIMSEQ-iff: (f \text{ ----> } L) = (f \text{ ----NS> } L)$ 
by (blast intro:  $LIMSEQ-NSLIMSEQ$   $NSLIMSEQ-LIMSEQ$ )

```

12.1.2 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  $NSLIMSEQ-norm-zero: ((\lambda n. norm (X n)) \text{ ----NS> } 0) = (X \text{ ----NS> } 0)$ 
by (simp add:  $LIMSEQ-NSLIMSEQ-iff$  [symmetric] tendsto-norm-zero-iff)

```

```

lemma  $NSLIMSEQ-rabs-zero: ((\%n. |f n|) \text{ ----NS> } 0) = (f \text{ ----NS> } (0::real))$ 
by (simp add:  $LIMSEQ-NSLIMSEQ-iff$  [symmetric] tendsto-rabs-zero-iff)

```

Generalization to other limits

```

lemma  $NSLIMSEQ-imp-rabs: f \text{ ----NS> } (l::real) ==> (\%n. |f n|) \text{ ----NS> } |l|$ 
apply (simp add:  $NSLIMSEQ-def$ )
apply (auto intro:  $approx-hrabs$ 
  simp add:  $starfun-abs$ )
done

```

```

lemma  $NSLIMSEQ-inverse-zero:$ 
   $\forall y::real. \exists N. \forall n \geq N. y < f(n)$ 
   $==> (\%n. inverse(f n)) \text{ ----NS> } 0$ 
by (simp add:  $LIMSEQ-NSLIMSEQ-iff$  [symmetric]  $LIMSEQ-inverse-zero$ )

```

```

lemma  $NSLIMSEQ-inverse-real-of-nat: (\%n. inverse(real(Suc n))) \text{ ----NS> } 0$ 
by (simp add:  $LIMSEQ-NSLIMSEQ-iff$  [symmetric]  $LIMSEQ-inverse-real-of-nat$ )

```

```

lemma  $NSLIMSEQ-inverse-real-of-nat-add:$ 

```

(% n . $r + \text{inverse}(\text{real}(\text{Suc } n))$) ----NS> r
by (*simp add: LIMSEQ-NSLIMSEQ-iff [symmetric] LIMSEQ-inverse-real-of-nat-add*)

lemma *NSLIMSEQ-inverse-real-of-nat-add-minus*:
 (% n . $r + -\text{inverse}(\text{real}(\text{Suc } n))$) ----NS> r
by (*simp add: LIMSEQ-NSLIMSEQ-iff [symmetric] LIMSEQ-inverse-real-of-nat-add-minus*)

lemma *NSLIMSEQ-inverse-real-of-nat-add-minus-mult*:
 (% n . $r * (1 + -\text{inverse}(\text{real}(\text{Suc } n)))$) ----NS> r
by (*simp add: LIMSEQ-NSLIMSEQ-iff [symmetric] LIMSEQ-inverse-real-of-nat-add-minus-mult*)

12.2 Convergence

lemma *nslimI*: X ----NS> $L \implies \text{nslim } X = L$
apply (*simp add: nslim-def*)
apply (*blast intro: NSLIMSEQ-unique*)
done

lemma *lim-nslim-iff*: $\text{lim } X = \text{nslim } X$
by (*simp add: lim-def nslim-def LIMSEQ-NSLIMSEQ-iff*)

lemma *NSconvergentD*: *NSconvergent* $X \implies \exists L. (X$ ----NS> $L)$
by (*simp add: NSconvergent-def*)

lemma *NSconvergentI*: $(X$ ----NS> $L) \implies \text{NSconvergent } X$
by (*auto simp add: NSconvergent-def*)

lemma *convergent-NSconvergent-iff*: *convergent* $X = \text{NSconvergent } X$
by (*simp add: convergent-def NSconvergent-def LIMSEQ-NSLIMSEQ-iff*)

lemma *NSconvergent-NSLIMSEQ-iff*: *NSconvergent* $X = (X$ ----NS> *nslim* $X)$
by (*auto intro: theI NSLIMSEQ-unique simp add: NSconvergent-def nslim-def*)

12.3 Bounded Monotonic Sequences

lemma *NSBseqD*: $[[\text{NSBseq } X; N : \text{HNatInfinite}]] \implies (*f* X) N : \text{HFinite}$
by (*simp add: NSBseq-def*)

lemma *Standard-subset-HFfinite*: *Standard* $\subseteq \text{HFfinite}$
unfolding *Standard-def* **by** *auto*

lemma *NSBseqD2*: *NSBseq* $X \implies (*f* X) N \in \text{HFfinite}$
apply (*cases* $N \in \text{HNatInfinite}$)
apply (*erule* (1) *NSBseqD*)
apply (*rule* *subsetD [OF Standard-subset-HFfinite]*)
apply (*simp add: HNatInfinite-def Nats-eq-Standard*)
done

lemma *NSBseqI*: $\forall N \in \text{HNatInfinite}. (*f* X) N : \text{HFfinite} \implies \text{NSBseq } X$

by (*simp add: NSBseq-def*)

The standard definition implies the nonstandard definition

lemma *Bseq-NSBseq*: $Bseq\ X \implies NSBseq\ X$

proof (*unfold NSBseq-def, 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*

hence $\forall N. hnorm\ (starfun\ X\ N) \leq star-of\ K$ **by** *transfer*

hence $hnorm\ (starfun\ X\ N) \leq star-of\ K$ **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*)

thus $starfun\ X\ N \in HFinite$ **by** (*simp add: HFinite-def*)

qed

The nonstandard definition implies the standard definition

lemma *SReal-less-omega*: $r \in \mathbb{R} \implies r < \omega$

apply (*insert HInfinite-omega*)

apply (*simp add: HInfinite-def*)

apply (*simp add: order-less-imp-le*)

done

lemma *NSBseq-Bseq*: $NSBseq\ X \implies Bseq\ X$

proof (*rule ccontr*)

let $?n = \lambda K. LEAST\ n. K < norm\ (X\ n)$

assume $NSBseq\ X$

hence $finite: (*f* X) ((*f* ?n) \omega) \in HFinite$

by (*rule NSBseqD2*)

assume $\neg Bseq\ X$

hence $\forall K > 0. \exists n. K < norm\ (X\ n)$

by (*simp add: Bseq-def linorder-not-le*)

hence $\forall K > 0. K < norm\ (X\ (?n\ K))$

by (*auto intro: LeastI-ex*)

hence $\forall K > 0. K < hnorm\ ((*f* X) ((*f* ?n) K))$

by *transfer*

hence $\omega < hnorm\ ((*f* X) ((*f* ?n) \omega))$

by *simp*

hence $\forall r \in \mathbb{R}. r < hnorm\ ((*f* X) ((*f* ?n) \omega))$

by (*simp add: order-less-trans [OF SReal-less-omega]*)

hence $(*f* X) ((*f* ?n) \omega) \in HInfinite$

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*: $NSconvergent\ X \implies NSBseq\ X$
apply (*simp add: NSconvergent-def NSBseq-def NSLIMSEQ-def*)
apply (*blast intro: HFinite-star-of approx-sym approx-HFinite*)
done

Standard Version: easily now proved using equivalence of NS and standard definitions

lemma *convergent-Bseq*: $convergent\ X \implies Bseq\ X$
by (*simp add: NSconvergent-NSBseq convergent-NSconvergent-iff Bseq-NSBseq-iff*)

12.3.1 Upper Bounds and Lubs of Bounded Sequences

lemma *NSBseq-isUb*: $NSBseq\ X \implies \exists U::real. isUb\ UNIV\ \{x. \exists n. X\ n = x\}$
 U
by (*simp add: Bseq-NSBseq-iff [symmetric] Bseq-isUb*)

lemma *NSBseq-isLub*: $NSBseq\ X \implies \exists U::real. isLub\ UNIV\ \{x. \exists n. X\ n = x\}$
 U
by (*simp add: Bseq-NSBseq-iff [symmetric] Bseq-isLub*)

12.3.2 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*: $\forall n \geq m. X\ n = X\ m \implies \exists L. (X\ \text{----} NS > L)$
by (*auto dest!: Bmonoseq-LIMSEQ simp add: LIMSEQ-NSLIMSEQ-iff*)

lemma *NSBseq-mono-NSconvergent*:
 $[\![\ NSBseq\ X; \forall m. \forall n \geq m. X\ m \leq X\ n \]\!] \implies NSconvergent\ (X::nat \implies real)$
by (*auto intro: Bseq-mono-convergent*
simp add: convergent-NSconvergent-iff [symmetric]
Bseq-NSBseq-iff [symmetric])

12.4 Cauchy Sequences

lemma *NSCauchyI*:
 $(\bigwedge M\ N. [\![\ M \in HNatInfinite; N \in HNatInfinite \]\!] \implies starfun\ X\ M \approx starfun\ X\ N)$
 $\implies NSCauchy\ X$
by (*simp add: NSCauchy-def*)

lemma *NSCauchyD*:
 $[\![\ NSCauchy\ X; M \in HNatInfinite; N \in HNatInfinite \]\!] \implies starfun\ X\ M \approx starfun\ X\ N$

by (*simp add: NSCauchy-def*)

12.4.1 Equivalence Between NS and Standard

lemma *Cauchy-NSCauchy*:

assumes X : *Cauchy* X **shows** *NSCauchy* X

proof (*rule NSCauchyI*)

fix M **assume** M : $M \in \text{HNatInfinite}$

fix N **assume** N : $N \in \text{HNatInfinite}$

have *starfun* X $M - \text{starfun } X$ $N \in \text{Infinitesimal}$

proof (*rule InfinitesimalI2*)

fix $r :: \text{real}$ **assume** r : $0 < r$

from *CauchyD* [*OF* X r]

obtain k **where** $\forall m \geq k. \forall n \geq k. \text{norm } (X\ m - X\ n) < r ..$

hence $\forall m \geq \text{star-of } k. \forall n \geq \text{star-of } k.$

$\text{hnorm } (\text{starfun } X\ m - \text{starfun } X\ n) < \text{star-of } r$

by *transfer*

thus $\text{hnorm } (\text{starfun } X\ M - \text{starfun } X\ N) < \text{star-of } r$

using M N **by** (*simp add: star-of-le-HNatInfinite*)

qed

thus *starfun* X $M \approx \text{starfun } X$ N

by (*unfold approx-def*)

qed

lemma *NSCauchy-Cauchy*:

assumes X : *NSCauchy* X **shows** *Cauchy* X

proof (*rule CauchyI*)

fix $r :: \text{real}$ **assume** r : $0 < r$

have $\exists k. \forall m \geq k. \forall n \geq k. \text{hnorm } (\text{starfun } X\ m - \text{starfun } X\ n) < \text{star-of } r$

proof (*intro exI allI impI*)

fix M **assume** $\text{whn} \leq M$

with *HNatInfinite-whn* **have** M : $M \in \text{HNatInfinite}$

by (*rule HNatInfinite-upward-closed*)

fix N **assume** $\text{whn} \leq N$

with *HNatInfinite-whn* **have** N : $N \in \text{HNatInfinite}$

by (*rule HNatInfinite-upward-closed*)

from X M N **have** *starfun* X $M \approx \text{starfun } X$ N

by (*rule NSCauchyD*)

hence *starfun* X $M - \text{starfun } X$ $N \in \text{Infinitesimal}$

by (*unfold approx-def*)

thus $\text{hnorm } (\text{starfun } X\ M - \text{starfun } X\ N) < \text{star-of } r$

using r **by** (*rule InfinitesimalD2*)

qed

thus $\exists k. \forall m \geq k. \forall n \geq k. \text{norm } (X\ m - X\ n) < r$

by *transfer*

qed

theorem *NSCauchy-Cauchy-iff*: *NSCauchy* $X = \text{Cauchy } X$

by (*blast intro!: NSCauchy-Cauchy Cauchy-NSCauchy*)

12.4.2 Cauchy Sequences are Bounded

A Cauchy sequence is bounded – nonstandard version

lemma *NSCauchy-NSBseq*: $NSCauchy\ X \implies NSBseq\ X$
by (*simp add: Cauchy-Bseq Bseq-NSBseq-iff [symmetric] NSCauchy-Cauchy-iff*)

12.4.3 Cauchy Sequences are Convergent

Equivalence of Cauchy criterion and convergence: We will prove this using our NS formulation which provides a much easier proof than using the standard definition. We do not need to use properties of subsequences such as boundedness, monotonicity etc... Compare with Harrison’s corresponding proof in HOL which is much longer and more complicated. Of course, we do not have problems which he encountered with guessing the right instantiations for his ‘epsilon-delta’ proof(s) in this case since the NS formulations do not involve existential quantifiers.

lemma *NSconvergent-NSCauchy*: $NSconvergent\ X \implies NSCauchy\ X$
apply (*simp add: NSconvergent-def NSLIMSEQ-def NSCauchy-def, safe*)
apply (*auto intro: approx-trans2*)
done

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

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

12.5 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 ≤ (x::real); x < 1 |] ==> (%n. x ^ n) -----NS> 0
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::hypnat) 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::real) ==> (%n. |c| ^ n) -----NS>
0
by (simp add: LIMSEQ-rabs-realpow-zero LIMSEQ-NSLIMSEQ-iff [symmetric])

lemma NSLIMSEQ-rabs-realpow-zero2: |c| < (1::real) ==> (%n. c ^ n) -----NS>
0
by (simp add: LIMSEQ-rabs-realpow-zero2 LIMSEQ-NSLIMSEQ-iff [symmetric])

end

```

13 HSeries: Finite Summation and Infinite Series for Hyperreals

```

theory HSeries
imports Series HSEQ
begin

```

definition

```

sumhr :: (hypnat * hypnat * (nat=>real)) => hypreal where
sumhr =
  (%(M,N,f). starfun2 (%m n. setsum f {m..<n}) M N)

```

definition

$NSsums :: [nat=>real,real] => bool$ (**infixr** $NSsums$ 80) **where**
 $f NSsums s = (\%n. setsum f \{0..<n\}) \text{ ---- } NS> s$

definition

$NSsummable :: (nat=>real) => bool$ **where**
 $NSsummable f = (\exists s. f NSsums s)$

definition

$NSsuminf :: (nat=>real) => real$ **where**
 $NSsuminf f = (THE s. f NSsums s)$

lemma $sumhr\text{-}app$: $sumhr(M,N,f) = (*f2* (\lambda m n. setsum f \{m..<n\})) M N$
by ($simp$ add : $sumhr\text{-}def$)

Base case in definition of $sumr$

lemma $sumhr\text{-}zero$ [$simp$]: $!!m. sumhr(m,0,f) = 0$
unfolding $sumhr\text{-}app$ **by** $transfer$ $simp$

Recursive case in definition of $sumr$

lemma $sumhr\text{-}if$:

$!!m n. sumhr(m,n+1,f) =$
 $(if\ n + 1 \leq m\ then\ 0\ else\ sumhr(m,n,f) + (*f* f)\ n)$

unfolding $sumhr\text{-}app$ **by** $transfer$ $simp$

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

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

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

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

lemma $sumhr\text{-}add$:

$!!m n. sumhr(m,n,f) + sumhr(m,n,g) = sumhr(m,n,\%i. f\ i + g\ i)$
unfolding $sumhr\text{-}app$ **by** $transfer$ ($rule$ $setsum\text{-}addf$ [$symmetric$])

lemma $sumhr\text{-}mult$:

$!!m n. hypreal\text{-}of\text{-}real\ r * sumhr(m,n,f) = sumhr(m,n,\%n. r * f\ n)$
unfolding $sumhr\text{-}app$ **by** $transfer$ ($rule$ $setsum\text{-}right\text{-}distrib$)

lemma $sumhr\text{-}split\text{-}add$:

$!!n p. n < p ==> sumhr(0,n,f) + sumhr(n,p,f) = sumhr(0,p,f)$
unfolding $sumhr\text{-}app$ **by** $transfer$ ($simp$ add : $setsum\text{-}add\text{-}nat\text{-}ivl$)

lemma *sumhr-split-diff*: $n < p \implies \text{sumhr}(0, p, f) - \text{sumhr}(0, n, f) = \text{sumhr}(n, p, f)$
by (*drule-tac* $f = f$ **in** *sumhr-split-add* [*symmetric*], *simp*)

lemma *sumhr-hrabs*: $\forall m n. \text{abs}(\text{sumhr}(m, n, f)) \leq \text{sumhr}(m, n, \%i. \text{abs}(f i))$
unfolding *sumhr-app* **by** *transfer* (*rule setsum-abs*)

other general version also needed

lemma *sumhr-fun-hypnat-eq*:
 $(\forall r. m \leq r \ \& \ r < n \implies f r = g r) \implies$
 $\text{sumhr}(\text{hypnat-of-nat } m, \text{hypnat-of-nat } n, f) =$
 $\text{sumhr}(\text{hypnat-of-nat } m, \text{hypnat-of-nat } n, g)$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-const*:
 $\forall n. \text{sumhr}(0, n, \%i. r) = \text{hypreal-of-hypnat } n * \text{hypreal-of-real } r$
unfolding *sumhr-app* **by** *transfer* (*simp add: real-of-nat-def*)

lemma *sumhr-less-bounds-zero* [*simp*]: $\forall m n. n < m \implies \text{sumhr}(m, n, f) = 0$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-minus*: $\forall m n. \text{sumhr}(m, n, \%i. - f i) = - \text{sumhr}(m, n, f)$
unfolding *sumhr-app* **by** *transfer* (*rule setsum-negf*)

lemma *sumhr-shift-bounds*:
 $\forall m n. \text{sumhr}(m + \text{hypnat-of-nat } k, n + \text{hypnat-of-nat } k, f) =$
 $\text{sumhr}(m, n, \%i. f(i + k))$
unfolding *sumhr-app* **by** *transfer* (*rule setsum-shift-bounds-nat-ivl*)

13.1 Nonstandard Sums

Infinite sums are obtained by summing to some infinite hypernatural (such as *whn*)

lemma *sumhr-hypreal-of-hypnat-omega*:
 $\text{sumhr}(0, \text{whn}, \%i. 1) = \text{hypreal-of-hypnat } \text{whn}$
by (*simp add: sumhr-const*)

lemma *sumhr-hypreal-omega-minus-one*: $\text{sumhr}(0, \text{whn}, \%i. 1) = \text{omega} - 1$
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 real-of-nat-def*)
done

lemma *sumhr-minus-one-realpow-zero* [*simp*]:
 $\forall N. \text{sumhr}(0, N + N, \%i. (-1) ^ (i+1)) = 0$

unfolding *sumhr-app*
by *transfer (simp del: power-Suc add: mult-2 [symmetric])*

lemma *sumhr-interval-const*:
 $(\forall n. m \leq \text{Suc } n \longrightarrow f\ n = r) \ \& \ m \leq na$
 $\implies \text{sumhr}(\text{hypnat-of-nat } m, \text{hypnat-of-nat } na, f) =$
 $(\text{hypreal-of-nat } (na - m) * \text{hypreal-of-real } r)$

unfolding *sumhr-app* **by** *transfer simp*

lemma *starfunNat-sumr*: $!!N. (*f* (\%n. \text{setsum } f \{0..<n\}))\ N = \text{sumhr}(0, N, f)$
unfolding *sumhr-app* **by** *transfer (rule refl)*

lemma *sumhr-hrabs-approx* [*simp*]: $\text{sumhr}(0, M, f) \ @ = \text{sumhr}(0, N, f)$
 $\implies \text{abs}(\text{sumhr}(M, N, f)) \ @ = 0$
apply (*cut-tac x = M and y = N in linorder-less-linear*)
apply (*auto simp add: approx-refl*)
apply (*drule approx-sym [THEN approx-minus-iff [THEN iffD1]]*)
apply (*auto dest: approx-hrabs*
simp add: sumhr-split-diff diff-minus [symmetric])

done

lemma *sums-NSsums-iff*: $(f \text{ sums } l) = (f \text{ NSsums } l)$
by (*simp add: sums-def NSsums-def LIMSEQ-NSLIMSEQ-iff*)

lemma *summable-NSsummable-iff*: $(\text{summable } f) = (\text{NSsummable } f)$
by (*simp add: summable-def NSsummable-def sums-NSsums-iff*)

lemma *suminf-NSsuminf-iff*: $(\text{suminf } f) = (\text{NSsuminf } f)$
by (*simp add: suminf-def NSsuminf-def sums-NSsums-iff*)

lemma *NSsums-NSsummable*: $f \text{ NSsums } l \implies \text{NSsummable } f$
by (*simp add: NSsums-def NSsummable-def, blast*)

lemma *NSsummable-NSsums*: $\text{NSsummable } f \implies f \text{ NSsums } (\text{NSsuminf } f)$
apply (*simp add: NSsummable-def NSsuminf-def NSsums-def*)
apply (*blast intro: theI NSLIMSEQ-unique*)
done

lemma *NSsums-unique*: $f \text{ NSsums } s \implies (s = \text{NSsuminf } f)$
by (*simp add: suminf-NSsuminf-iff [symmetric] sums-NSsums-iff sums-unique*)

lemma *NSseries-zero*:
 $\forall m. n \leq \text{Suc } m \longrightarrow f(m) = 0 \implies f \text{ NSsums } (\text{setsum } f \{0..<n\})$
by (*simp add: sums-NSsums-iff [symmetric] series-zero*)

lemma *NSsummable-NSCauchy*:
 $\text{NSsummable } f =$
 $(\forall M \in \text{HNatInfinite}. \forall N \in \text{HNatInfinite}. \text{abs}(\text{sumhr}(M, N, f)) \ @ = 0)$

```

apply (auto simp add: summable-NSsummable-iff [symmetric]
        summable-convergent-sumr-iff convergent-NSconvergent-iff
        NSCauchy-NSconvergent-iff [symmetric] NSCauchy-def starfunNat-sumr)
apply (cut-tac x = M and y = N in linorder-less-linear)
apply (auto simp add: approx-refl)
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 add: sumhr-split-diff diff-minus [symmetric])
done

```

Terms of a convergent series tend to zero

```

lemma NSsummable-NSLIMSEQ-zero: NSsummable f ==> f ----NS> 0
apply (auto simp add: NSLIMSEQ-def NSsummable-NSCauchy)
apply (drule bspec, 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:
  [|  $\exists N. \forall n. N \leq n \longrightarrow \text{abs}(f\ n) \leq g\ n; \text{NSsummable } g$  |] ==> NSsummable
  f
apply (fold summable-NSsummable-iff)
apply (rule summable-comparison-test, simp, assumption)
done

```

```

lemma NSsummable-rabs-comparison-test:
  [|  $\exists N. \forall n. N \leq n \longrightarrow \text{abs}(f\ n) \leq g\ n; \text{NSsummable } g$  |]
  ==> NSsummable (%k. abs (f k))
apply (rule NSsummable-comparison-test)
apply (auto)
done

```

end

14 HLim: Limits and Continuity (Nonstandard)

```

theory HLim
imports Star Lim
begin

```

Nonstandard Definitions

```

definition
  NSLIM :: [ $'a::\text{real-normed-vector} \Rightarrow 'b::\text{real-normed-vector}, 'a, 'b$ ] ==> bool
  (((-)/ -- (-)/ --NS> (-)) [60, 0, 60] 60) where
  f -- a --NS> L =

```

$$(\forall x. (x \neq \text{star-of } a \ \& \ x \ @ = \text{star-of } a \ \longrightarrow \ (\ *f* f \) \ x \ @ = \text{star-of } L))$$

definition

isNSCont :: [*a*::*real-normed-vector* => *b*::*real-normed-vector*, *a*] => *bool* **where**
 — NS definition dispenses with limit notions
isNSCont *f* *a* = ($\forall y. y \ @ = \text{star-of } a \ \longrightarrow$
 $(\ *f* f \) \ y \ @ = \text{star-of } (f \ a)$)

definition

isNSUCont :: [*a*::*real-normed-vector* => *b*::*real-normed-vector*] => *bool* **where**
isNSUCont *f* = ($\forall x y. x \ @ = y \ \longrightarrow \ (\ *f* f \) \ x \ @ = \ (\ *f* f \) \ y$)

14.1 Limits of Functions**lemma** *NSLIM-I*:

$$(\bigwedge x. \llbracket x \neq \text{star-of } a; x \approx \text{star-of } a \rrbracket \implies \text{starfun } f \ x \approx \text{star-of } L)$$

$$\implies f \ \text{-- } a \ \text{--NS} > L$$

by (*simp add: NSLIM-def*)

lemma *NSLIM-D*:

$$\llbracket f \ \text{-- } a \ \text{--NS} > L; x \neq \text{star-of } a; x \approx \text{star-of } a \rrbracket$$

$$\implies \text{starfun } f \ x \approx \text{star-of } L$$

by (*simp add: NSLIM-def*)

Proving properties of limits using nonstandard definition. The properties hold for standard limits as well!

lemma *NSLIM-mult*:

fixes *l m* :: *'a*::*real-normed-algebra*
shows $\llbracket f \ \text{-- } x \ \text{--NS} > l; g \ \text{-- } x \ \text{--NS} > m \rrbracket$
 $\implies (\%x. f(x) * g(x)) \ \text{-- } x \ \text{--NS} > (l * m)$

by (*auto simp add: NSLIM-def intro!: approx-mult-HFfinite*)

lemma *starfun-scaleR* [*simp*]:

$$\text{starfun } (\lambda x. f \ x \ *_R \ g \ x) = (\lambda x. \text{scaleHR } (\text{starfun } f \ x) (\text{starfun } g \ x))$$

by *transfer (rule refl)*

lemma *NSLIM-scaleR*:

$$\llbracket f \ \text{-- } x \ \text{--NS} > l; g \ \text{-- } x \ \text{--NS} > m \rrbracket$$

$$\implies (\%x. f(x) *_R g(x)) \ \text{-- } x \ \text{--NS} > (l *_R m)$$

by (*auto simp add: NSLIM-def intro!: approx-scaleR-HFfinite*)

lemma *NSLIM-add*:

$$\llbracket f \ \text{-- } x \ \text{--NS} > l; g \ \text{-- } x \ \text{--NS} > m \rrbracket$$

$$\implies (\%x. f(x) + g(x)) \ \text{-- } x \ \text{--NS} > (l + m)$$

by (*auto simp add: NSLIM-def intro!: approx-add*)

lemma *NSLIM-const* [*simp*]: ($\%x. k$) $\text{-- } x \ \text{--NS} > k$

by (*simp add: NSLIM-def*)

lemma *NSLIM-minus*: $f \text{ -- } a \text{ --NS} > L \implies (\%x. -f(x)) \text{ -- } a \text{ --NS} > -L$
by (*simp add: NSLIM-def*)

lemma *NSLIM-diff*:

$\llbracket f \text{ -- } x \text{ --NS} > l; g \text{ -- } x \text{ --NS} > m \rrbracket \implies (\lambda x. f x - g x) \text{ -- } x \text{ --NS} > (l - m)$

by (*simp only: diff-minus NSLIM-add NSLIM-minus*)

lemma *NSLIM-add-minus*: $\llbracket f \text{ -- } x \text{ --NS} > l; g \text{ -- } x \text{ --NS} > m \rrbracket \implies$
 $(\%x. f(x) + -g(x)) \text{ -- } x \text{ --NS} > (l + -m)$

by (*simp only: NSLIM-add NSLIM-minus*)

lemma *NSLIM-inverse*:

fixes $L :: 'a::\text{real-normed-div-algebra}$

shows $\llbracket f \text{ -- } a \text{ --NS} > L; L \neq 0 \rrbracket$

$\implies (\%x. \text{inverse}(f(x))) \text{ -- } a \text{ --NS} > (\text{inverse } L)$

apply (*simp add: NSLIM-def, clarify*)

apply (*drule spec*)

apply (*auto simp add: star-of-approx-inverse*)

done

lemma *NSLIM-zero*:

assumes $f: f \text{ -- } a \text{ --NS} > l$ **shows** $(\%x. f(x) - l) \text{ -- } a \text{ --NS} > 0$

proof –

have $(\lambda x. f x - l) \text{ -- } a \text{ --NS} > l - l$

by (*rule NSLIM-diff [OF f NSLIM-const]*)

thus *?thesis* **by** *simp*

qed

lemma *NSLIM-zero-cancel*: $(\%x. f(x) - l) \text{ -- } x \text{ --NS} > 0 \implies f \text{ -- } x \text{ --NS} > l$

apply (*drule-tac g = %x. l and m = l in NSLIM-add*)

apply (*auto simp add: diff-minus add-assoc*)

done

lemma *NSLIM-const-not-eq*:

fixes $a :: 'a::\text{real-normed-algebra-1}$

shows $k \neq L \implies \neg (\lambda x. k) \text{ -- } a \text{ --NS} > L$

apply (*simp add: NSLIM-def*)

apply (*rule-tac x=star-of a + of-hypreal epsilon in exI*)

apply (*simp add: hypreal-epsilon-not-zero approx-def*)

done

lemma *NSLIM-not-zero*:

fixes $a :: 'a::\text{real-normed-algebra-1}$

shows $k \neq 0 \implies \neg (\lambda x. k) \text{ -- } a \text{ --NS} > 0$

by (*rule NSLIM-const-not-eq*)

lemma *NSLIM-const-eq*:

```

fixes a :: 'a::real-normed-algebra-1
shows ( $\lambda x. k$ ) -- a --NS> L  $\implies$  k = L
apply (rule ccontr)
apply (blast dest: NSLIM-const-not-eq)
done

```

```

lemma NSLIM-unique:
  fixes a :: 'a::real-normed-algebra-1
  shows  $\llbracket f -- a --NS> L; f -- a --NS> M \rrbracket \implies L = M$ 
apply (drule (1) NSLIM-diff)
apply (auto dest!: NSLIM-const-eq)
done

```

```

lemma NSLIM-mult-zero:
  fixes f g :: 'a::real-normed-vector  $\Rightarrow$  'b::real-normed-algebra
  shows  $\llbracket f -- x --NS> 0; g -- x --NS> 0 \rrbracket \implies (\%x. f(x)*g(x)) --$ 
 $x --NS> 0$ 
by (drule NSLIM-mult, auto)

```

```

lemma NSLIM-self: ( $\%x. x$ ) -- a --NS> a
by (simp add: NSLIM-def)

```

14.1.1 Equivalence of LIM and NSLIM

```

lemma LIM-NSLIM:
  assumes f: f -- a --> L shows f -- a --NS> L
proof (rule NSLIM-I)
  fix x
  assume neq: x  $\neq$  star-of a
  assume approx: x  $\approx$  star-of a
  have starfun f x - star-of L  $\in$  Infinitesimal
  proof (rule InfinitesimalI2)
    fix r::real assume r: 0 < r
    from LIM-D [OF f r]
    obtain s where s: 0 < s and
      less-r:  $\bigwedge x. \llbracket x \neq a; \text{norm } (x - a) < s \rrbracket \implies \text{norm } (f x - L) < r$ 
    by fast
    from less-r have less-r':
       $\bigwedge x. \llbracket x \neq \text{star-of } a; \text{hnorm } (x - \text{star-of } a) < \text{star-of } s \rrbracket$ 
 $\implies \text{hnorm } (\text{starfun } f x - \text{star-of } L) < \text{star-of } r$ 
    by transfer
    from approx have x - star-of a  $\in$  Infinitesimal
    by (unfold approx-def)
    hence hnorm (x - star-of a) < star-of s
    using s by (rule InfinitesimalD2)
    with neq show hnorm (starfun f x - star-of L) < star-of r
    by (rule less-r')
  qed
thus starfun f x  $\approx$  star-of L

```

by (*unfold approx-def*)
qed

lemma *NSLIM-LIM*:

assumes $f: f \dashrightarrow a \dashrightarrow NS > L$ shows $f \dashrightarrow a \dashrightarrow L$

proof (*rule LIM-I*)

fix $r::real$ assume $r: 0 < r$

have $\exists s > 0. \forall x. x \neq \text{star-of } a \wedge \text{hnorm } (x - \text{star-of } a) < s$
 $\longrightarrow \text{hnorm } (\text{starfun } f \ x - \text{star-of } L) < \text{star-of } r$

proof (*rule exI, safe*)

show $0 < \text{epsilon}$ by (*rule hypreal-epsilon-gt-zero*)

next

fix x assume $\text{neg}: x \neq \text{star-of } a$

assume $\text{hnorm } (x - \text{star-of } a) < \text{epsilon}$

with *Infinitesimal-epsilon*

have $x - \text{star-of } a \in \text{Infinitesimal}$

by (*rule hnorm-less-Infinitesimal*)

hence $x \approx \text{star-of } a$

by (*unfold approx-def*)

with $f \ \text{neg}$ have $\text{starfun } f \ x \approx \text{star-of } L$

by (*rule NSLIM-D*)

hence $\text{starfun } f \ x - \text{star-of } L \in \text{Infinitesimal}$

by (*unfold approx-def*)

thus $\text{hnorm } (\text{starfun } f \ x - \text{star-of } L) < \text{star-of } r$

using r by (*rule InfinitesimalD2*)

qed

thus $\exists s > 0. \forall x. x \neq a \wedge \text{norm } (x - a) < s \longrightarrow \text{norm } (f \ x - L) < r$
 by *transfer*

qed

theorem *LIM-NSLIM-iff*: $(f \dashrightarrow x \dashrightarrow L) = (f \dashrightarrow x \dashrightarrow NS > L)$

by (*blast intro: LIM-NSLIM NSLIM-LIM*)

14.2 Continuity

lemma *isNSContD*:

$\llbracket \text{isNSCont } f \ a; y \approx \text{star-of } a \rrbracket \Longrightarrow (*f* \ f) \ y \approx \text{star-of } (f \ a)$

by (*simp add: isNSCont-def*)

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

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

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

apply (*simp add: isNSCont-def NSLIM-def, auto*)

apply (*case-tac y = star-of a, auto*)

done

NS continuity can be defined using NS Limit in similar fashion to standard def of continuity

lemma *isNSCont-NSLIM-iff*: $(isNSCont\ f\ a) = (f\ \text{--}\ a\ \text{--}\ NS > (f\ a))$
by (*blast intro: isNSCont-NSLIM NSLIM-isNSCont*)

Hence, NS continuity can be given in terms of standard limit

lemma *isNSCont-LIM-iff*: $(isNSCont\ f\ a) = (f\ \text{--}\ a\ \text{--}\ > (f\ a))$
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*: $(isNSCont\ f\ a) = (isCont\ f\ a)$
apply (*simp add: isCont-def*)
apply (*rule isNSCont-LIM-iff*)
done

Standard continuity \iff NS continuity

lemma *isCont-isNSCont*: $isCont\ f\ a \implies isNSCont\ f\ a$
by (*erule isNSCont-isCont-iff [THEN iffD2]*)

NS continuity \iff Standard continuity

lemma *isNSCont-isCont*: $isNSCont\ f\ a \implies isCont\ f\ a$
by (*erule isNSCont-isCont-iff [THEN iffD1]*)

Alternative definition of continuity

lemma *NSLIM-h-iff*: $(f\ \text{--}\ a\ \text{--}\ NS > L) = ((\%h. f(a + h))\ \text{--}\ 0\ \text{--}\ NS > L)$
apply (*simp add: NSLIM-def, auto*)
apply (*drule-tac x = star-of a + x in spec*)
apply (*drule-tac [2] x = - star-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 diff-minus [symmetric]*)
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 approx-refl star-n-zero-num*)
done

lemma *NSLIM-isCont-iff*: $(f\ \text{--}\ a\ \text{--}\ NS > f\ a) = ((\%h. f(a + h))\ \text{--}\ 0\ \text{--}\ NS > f\ a)$
by (*rule NSLIM-h-iff*)

lemma *isNSCont-minus*: $isNSCont\ f\ a \implies isNSCont\ (\%x. - f\ x)\ a$
by (*simp add: isNSCont-def*)

lemma *isNSCont-inverse*:

fixes $f :: 'a::real-normed-vector \Rightarrow 'b::real-normed-div-algebra$
shows $[[isNSCont\ f\ x; f\ x \neq 0]] \implies isNSCont\ (\%x. inverse\ (f\ x))\ x$
by (*auto intro: isCont-inverse simp add: isNSCont-isCont-iff*)

lemma *isNSCont-const* [*simp*]: *isNSCont* (%*x*. *k*) *a*
by (*simp add: isNSCont-def*)

lemma *isNSCont-abs* [*simp*]: *isNSCont* *abs* (*a*::*real*)
apply (*simp add: isNSCont-def*)
apply (*auto intro: approx-hrabs simp add: starfun-rabs-hrabs*)
done

14.3 Uniform Continuity

lemma *isNSUContD*: [| *isNSUCont* *f*; *x* \approx *y* |] ==> (**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*
proof (*unfold isNSUCont-def, safe*)
fix *x y* :: '*a* *star*
assume *approx*: *x* \approx *y*
have *starfun* *f* *x* - *starfun* *f* *y* \in *Infinesimal*
proof (*rule InfinitesimalI2*)
fix *r*::*real* **assume** *r*: $0 < r$
with *f* **obtain** *s* **where** $0 < s$ **and**
less-r: $\bigwedge x y. \text{norm } (x - y) < s \implies \text{norm } (f x - f y) < r$
by (*auto simp add: isUCont-def dist-norm*)
from *less-r* **have** *less-r'*:
 $\bigwedge x y. \text{hnorm } (x - y) < \text{star-of } s$
 $\implies \text{hnorm } (\text{starfun } f x - \text{starfun } f y) < \text{star-of } r$
by *transfer*
from *approx* **have** *x* - *y* \in *Infinesimal*
by (*unfold approx-def*)
hence *hnorm* (*x* - *y*) $<$ *star-of* *s*
using *s* **by** (*rule InfinitesimalD2*)
thus *hnorm* (*starfun* *f* *x* - *starfun* *f* *y*) $<$ *star-of* *r*
by (*rule less-r'*)
qed
thus *starfun* *f* *x* \approx *starfun* *f* *y*
by (*unfold approx-def*)
qed

lemma *isNSUCont-isUCont*:
fixes *f* :: '*a*::*real-normed-vector* \Rightarrow '*b*::*real-normed-vector*
assumes *f*: *isNSUCont* *f* **shows** *isUCont* *f*
proof (*unfold isUCont-def dist-norm, safe*)
fix *r*::*real* **assume** *r*: $0 < r$
have $\exists s > 0. \forall x y. \text{hnorm } (x - y) < s$
 $\implies \text{hnorm } (\text{starfun } f x - \text{starfun } f y) < \text{star-of } r$
proof (*rule exI, safe*)

```

  show  $0 < \epsilon$  by (rule hypreal-epsilon-gt-zero)
next
fix  $x y :: 'a \text{ star}$ 
assume  $\text{hnorm } (x - y) < \epsilon$ 
with Infinitesimal-epsilon
have  $x - y \in \text{Infinitesimal}$ 
  by (rule hnorm-less-Infinitesimal)
hence  $x \approx y$ 
  by (unfold approx-def)
with  $f$  have  $\text{starfun } f x \approx \text{starfun } f y$ 
  by (simp add: isNSUCont-def)
hence  $\text{starfun } f x - \text{starfun } f y \in \text{Infinitesimal}$ 
  by (unfold approx-def)
thus  $\text{hnorm } (\text{starfun } f x - \text{starfun } f y) < \text{star-of } r$ 
  using  $r$  by (rule InfinitesimalD2)
qed
thus  $\exists s > 0. \forall x y. \text{norm } (x - y) < s \longrightarrow \text{norm } (f x - f y) < r$ 
  by transfer
qed
end

```

15 HDeriv: Differentiation (Nonstandard)

```

theory HDeriv
imports Deriv HLim
begin

```

Nonstandard Definitions

definition

```

nsderiv :: [ $'a :: \text{real-normed-field} \Rightarrow 'a, 'a, 'a] \Rightarrow \text{bool}$ 
  ((NSDERIV (-) / (-) / :> (-)) [1000, 1000, 60] 60) where
NSDERIV  $f x :> D = (\forall h \in \text{Infinitesimal} - \{0\}.
  ((** f)(\text{star-of } x + h)
  - \text{star-of } (f x)) / h @= \text{star-of } D)$ 

```

definition

```

NSdifferentiable :: [ $'a :: \text{real-normed-field} \Rightarrow 'a, 'a] \Rightarrow \text{bool}$ 
  (infixl NSdifferentiable 60) where
 $f \text{ NSdifferentiable } x = (\exists D. \text{NSDERIV } f x :> D)$ 

```

definition

```

increment :: [ $\text{real} \Rightarrow \text{real}, \text{real}, \text{hypreal}] \Rightarrow \text{hypreal}$  where
increment  $f x h = (@\text{inc. } f \text{ NSdifferentiable } x \ \&
  \text{inc} = (** f)(\text{hypreal-of-real } x + h) - \text{hypreal-of-real } (f x))$ 

```

15.1 Derivatives

lemma *DERIV-NS-iff*:

$$(DERIV f x :> D) = ((\%h. (f(x + h) - f(x))/h) -- 0 -- NS > D)$$

by (*simp add: deriv-def LIM-NSLIM-iff*)

lemma *NS-DERIV-D*: $DERIV f x :> D \implies (\%h. (f(x + h) - f(x))/h) -- 0 -- NS > D$

by (*simp add: deriv-def LIM-NSLIM-iff*)

lemma *hnorm-of-hypreal*:

$$\bigwedge r. \text{hnorm } ((*f* \text{ of-real } r :: 'a :: \text{real-normed-div-algebra star}) = |r|$$

by *transfer (rule norm-of-real)*

lemma *Infinitesimal-of-hypreal*:

$$x \in \text{Infinitesimal} \implies$$

$$((*f* \text{ of-real } x :: 'a :: \text{real-normed-div-algebra star}) \in \text{Infinitesimal})$$

apply (*rule InfinitesimalI2*)

apply (*drule (1) InfinitesimalD2*)

apply (*simp add: hnorm-of-hypreal*)

done

lemma *of-hypreal-eq-0-iff*:

$$\bigwedge x. ((*f* \text{ of-real } x = (0 :: 'a :: \text{real-algebra-1 star})) = (x = 0))$$

by *transfer (rule of-real-eq-0-iff)*

lemma *NSDeriv-unique*:

$$[| \text{NSDERIV } f x :> D; \text{NSDERIV } f x :> E |] \implies D = E$$

apply (*subgoal-tac (*f* of-real epsilon \in 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 Infinitesimal-epsilon*)

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*:

$$(\text{NSDERIV } f x :> D) = ((\%h. (f(x + h) - f(x))/h) -- 0 -- NS > D)$$

apply (*simp add: nsderiv-def NSLIM-def, auto*)

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*:

$(NSDERIV f x :> D) = ((\%z. (f(z) - f(x)) / (z-x)) \text{---} x \text{---} NS > D)$
by (*simp add: NSDERIV-NSLIM-iff DERIV-LIM-iff diff-minus [symmetric]*
LIM-NSLIM-iff [symmetric])

lemma *NSDERIV-iff2*:

$(NSDERIV f x :> D) =$
 $(\forall w.$
 $w \neq \text{star-of } x \ \& \ w \approx \text{star-of } x \text{ ---} >$
 $(*f* (\%z. (f z - f x) / (z-x))) w \approx \text{star-of } D)$
by (*simp add: NSDERIV-NSLIM-iff2 NSLIM-def*)

lemma *hypreal-not-eq-minus-iff*:

$(x \neq a) = (x - a \neq (0::'a::\text{ab-group-add}))$
by *auto*

lemma *NSDERIVD5*:

$(NSDERIV f x :> D) ==>$
 $(\forall u. u \approx \text{hypreal-of-real } x \text{ ---} >$
 $(*f* (\%z. f z - f x)) u \approx \text{hypreal-of-real } D * (u - \text{hypreal-of-real } x))$
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* (\%z. z-x)) u \neq (0::hypreal))*)
apply (*auto simp add:*
approx-minus-iff [THEN iffD1, THEN mem-infmal-iff [THEN iffD2]]
Infinitesimal-subset-HFinite [THEN subsetD])
done

lemma *NSDERIVD4*:

$(NSDERIV f x :> D) ==>$
 $(\forall h \in \text{Infinitesimal.}$
 $((*f* f)(\text{hypreal-of-real } x + h) -$
 $\text{hypreal-of-real } (f x)) \approx (\text{hypreal-of-real } D) * h)$
apply (*auto simp add: nsderiv-def*)
apply (*case-tac h = (0::hypreal))*)
apply (*auto simp add: diff-minus*)
apply (*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: diff-minus)
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-HFfinite [THEN subsetD]
      simp add: mult-assoc diff-minus)
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-HFfinite [THEN subsetD]
      simp add: mult-assoc nonzero-mult-divide-cancel-right)
apply (drule-tac x3=D in
      HFfinite-star-of [THEN [2] Infinitesimal-HFfinite-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, straightforward, algebraic manipulations

Constant function

```

lemma NSDERIV-const [simp]: (NSDERIV (%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 (%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: diff-minus add-ac)
done

```

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

```

lemma lemma-nsderiv1:

```

```

fixes a b c d :: 'a::comm-ring star
shows (a*b) - (c*d) = (b*(a - c)) + (c*(b - d))
by (simp add: right-diff-distrib mult-ac)

```

lemma lemma-nsderiv2:

```

fixes x y z :: 'a::real-normed-field star
shows [| (x - y) / z = star-of D + yb; z ≠ 0;
          z ∈ Infinitesimal; yb ∈ Infinitesimal |]
      ==> x - y ≈ 0
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; NSDERIV g x :> Db |]
      ==> NSDERIV (%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 add: left-distrib right-distrib 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
      ==> NSDERIV (%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 ==> NSDERIV (%x. -(f x)) x
      :> -D

```

proof (*simp add: NSDERIV-NSLIM-iff*)
assume $(\lambda h. (f (x + h) - f x) / h) \dashv\dashv 0 \dashv\dashv NS > D$
hence *deriv*: $(\lambda h. - ((f(x+h) - f x) / h)) \dashv\dashv 0 \dashv\dashv NS > - D$
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 diff-minus*)
with *deriv*
show $(\lambda h. (- f (x + h) + f x) / h) \dashv\dashv 0 \dashv\dashv NS > - D$ **by** *simp*
qed

Subtraction

lemma *NSDERIV-add-minus*: $[[NSDERIV f x :> Da; NSDERIV g x :> Db]]$
 $\implies NSDERIV (\%x. f x + -g x) x :> Da + -Db$
by (*blast dest: NSDERIV-add NSDERIV-minus*)

lemma *NSDERIV-diff*:
 $[[NSDERIV f x :> Da; NSDERIV g x :> Db]]$
 $\implies NSDERIV (\%x. f x - g x) x :> Da - Db$
apply (*simp add: diff-minus*)
apply (*blast intro: NSDERIV-add-minus*)
done

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.

lemma *NSDERIV-zero*:
 $[[NSDERIV g x :> D;$
 $(*f* g) (star-of x + xa) = star-of (g x);$
 $xa \in Infinitesimal;$
 $xa \neq 0$
 $]] \implies D = 0$
apply (*simp add: nsderiv-def*)
apply (*drule bspec, auto*)
done

lemma *NSDERIV-approx*:
 $[[NSDERIV f x :> D; h \in Infinitesimal; h \neq 0]]$
 $\implies (*f* f) (star-of x + h) - star-of (f x) \approx 0$
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

lemma NSDERIVD1: $[[\text{NSDERIV } f (g x) \text{ :> } Da;$
 $(*f* g) (\text{star-of}(x) + xa) \neq \text{star-of}(g x);$
 $(*f* g) (\text{star-of}(x) + xa) \approx \text{star-of}(g x)$
 $]] \implies ((*f* f) ((*f* g) (\text{star-of}(x) + xa))$
 $\quad - \text{star-of}(f (g x)))$
 $\quad / ((*f* g) (\text{star-of}(x) + xa) - \text{star-of}(g x))$
 $\quad \approx \text{star-of}(Da)$
by (*auto simp add: NSDERIV-NSLIM-iff2 NSLIM-def diff-minus [symmetric]*)

lemma NSDERIVD2: $[[\text{NSDERIV } g x \text{ :> } Db; xa \in \text{Infinesimal}; xa \neq 0 \]]$
 $\implies ((*f* g) (\text{star-of}(x) + xa) - \text{star-of}(g x)) / xa$
 $\quad \approx \text{star-of}(Db)$
by (*auto simp add: NSDERIV-NSLIM-iff NSLIM-def mem-infmal-iff starfun-lambda-cancel*)

lemma lemma-chain: $(z : 'a :: \text{real-normed-field star}) \neq 0 \implies x * y = (x * \text{inverse}(z)) * (z * y)$
proof –
assume $z : z \neq 0$
have $x * y = x * (\text{inverse } z * z) * y$ **by** (*simp add: z*)
thus *?thesis* **by** (*simp add: mult-assoc*)
qed

This proof uses both definitions of differentiability.

lemma NSDERIV-chain: $[[\text{NSDERIV } f (g x) \text{ :> } Da; \text{NSDERIV } g x \text{ :> } Db \]]$
 $\implies \text{NSDERIV } (f \circ g) x \text{ :> } Da * Db$
apply (*simp (no-asm-simp) add: NSDERIV-NSLIM-iff NSLIM-def*
 $\quad \text{mem-infmal-iff [symmetric]}$)
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) (\text{star-of}(x) + xa) = \text{star-of}(g x))*)
apply (*drule-tac g = g in NSDERIV-zero*)
apply (*auto simp add: divide-inverse*)
apply (*rule-tac z1 = (*f* g) (\text{star-of}(x) + xa) - \text{star-of}(g x) and y1 =*
 $\text{inverse } xa$ **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]: $\text{NSDERIV } (\%x. x) x \text{ :> } 1$
by (*simp add: NSDERIV-NSLIM-iff NSLIM-def del: divide-self-if*)

lemma NSDERIV-cmult-Id [simp]: $\text{NSDERIV } (op * c) x \text{ :> } c$
by (*cut-tac c = c and x = x in NSDERIV-Id [THEN NSDERIV-cmult], simp*)

```

lemma NSDERIV-inverse:
  fixes x :: 'a::{real-normed-field}
  shows x ≠ 0 ==> NSDERIV (%x. inverse(x)) x :> (– (inverse x ^ Suc (Suc 0)))
apply (simp add: nsderiv-def)
apply (rule ballI, simp, clarify)
apply (frule (1) Infinitesimal-add-not-zero)
apply (simp add: add-commute)

apply (simp add: inverse-add nonzero-inverse-mult-distrib [symmetric] power-Suc
  nonzero-inverse-minus-eq [symmetric] add-ac mult-ac diff-minus
  del: inverse-mult-distrib inverse-minus-eq
  minus-mult-left [symmetric] minus-mult-right [symmetric])
apply (subst mult-commute, simp add: nonzero-mult-divide-cancel-right)
apply (simp (no-asm-simp) add: mult-assoc [symmetric] left-distrib
  del: minus-mult-left [symmetric] minus-mult-right [symmetric])
apply (rule-tac y = inverse (– (star-of x * star-of x)) in approx-trans)
apply (rule inverse-add-Infinitesimal-approx2)
apply (auto dest!: hypreal-of-real-HFinite-diff-Infinitesimal
  simp add: inverse-minus-eq [symmetric] HFinite-minus-iff)
apply (rule Infinitesimal-HFinite-mult, auto)
done

```

15.1.1 Equivalence of NS and Standard definitions

```

lemma divideR-eq-divide: x /R y = x / y
by (simp add: real-scaleR-def divide-inverse mult-commute)

```

Now equivalence between NSDERIV and DERIV

```

lemma NSDERIV-DERIV-iff: (NSDERIV f x :> D) = (DERIV f x :> D)
by (simp add: deriv-def NSDERIV-NSLIM-iff LIM-NSLIM-iff)

```

```

lemma NSDERIV-pow: NSDERIV (%x. x ^ n) x :> real n * (x ^ (n – Suc 0))
by (simp add: NSDERIV-DERIV-iff DERIV-pow)

```

Derivative of inverse

```

lemma NSDERIV-inverse-fun:
  fixes x :: 'a::{real-normed-field}
  shows [| NSDERIV f x :> d; f(x) ≠ 0 |]
    ==> NSDERIV (%x. inverse(f x)) x :> (– (d * inverse(f x) ^ Suc (Suc 0)))
by (simp add: NSDERIV-DERIV-iff DERIV-inverse-fun del: power-Suc)

```

Derivative of quotient

```

lemma NSDERIV-quotient:
  fixes x :: 'a::{real-normed-field}

```

shows [| NSDERIV $f x :=> d$; NSDERIV $g x :=> e$; $g(x) \neq 0$ |]
 $==>$ NSDERIV ($\%y. f(y) / (g y) x :=> (d * g(x) - (e * f(x))) / (g(x) \wedge \text{Suc} (\text{Suc } 0))$)
by (*simp add: NSDERIV-DERIV-iff DERIV-quotient del: power-Suc*)

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

lemma hypreal-eq-minus-iff3: $(x = y + z) = (x + -z = (y::\text{hypreal}))$
by *auto*

lemma CARAT-DERIVD:
assumes *all*: $\forall z. f z - f x = g z * (z - x)$
and *nsc*: $\text{isNSCont } g \ x$
shows NSDERIV $f x :=> g \ x$
proof –
from *nsc*
have $\forall w. w \neq \text{star-of } x \wedge w \approx \text{star-of } x \longrightarrow$
 $(*f* g) w * (w - \text{star-of } x) / (w - \text{star-of } x) \approx$
 $\text{star-of } (g \ x)$
by (*simp add: isNSCont-def nonzero-mult-divide-cancel-right*)
thus *?thesis using all*
by (*simp add: NSDERIV-iff2 starfun-if-eq cong: if-cong*)
qed

15.1.2 Differentiability predicate

lemma NSdifferentiableD: $f \ \text{NSdifferentiable } x ==> \exists D. \ \text{NSDERIV } f \ x :=> D$
by (*simp add: NSdifferentiable-def*)

lemma NSdifferentiableI: NSDERIV $f \ x :=> D ==> f \ \text{NSdifferentiable } x$
by (*force simp add: NSdifferentiable-def*)

15.2 (NS) Increment

lemma incrementI:
 $f \ \text{NSdifferentiable } x ==>$
 $\text{increment } f \ x \ h = (*f* f) (\text{hypreal-of-real}(x) + h) -$
 $\text{hypreal-of-real } (f \ x)$
by (*simp add: increment-def*)

lemma incrementI2: NSDERIV $f \ x :=> D ==>$
 $\text{increment } f \ x \ h = (*f* f) (\text{hypreal-of-real}(x) + h) -$
 $\text{hypreal-of-real } (f \ x)$
apply (*erule NSdifferentiableI [THEN incrementI]*)
done

```

lemma increment-thm: [| NSDERIV f x :> D; h ∈ Infinitesimal; h ≠ 0 |]
  ==> ∃ 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 hypreal-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: left-distrib)
done

```

```

lemma increment-thm2:
  [| NSDERIV f x :> D; h ≈ 0; h ≠ 0 |]
  ==> ∃ 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; h ≈ 0; h ≠ 0 |]
  ==> increment f x h ≈ 0
apply (drule increment-thm2,
  auto intro!: Infinitesimal-HFinite-mult2 HFinite-add simp add: left-distrib
  [symmetric] mem-infmal-iff [symmetric])
apply (erule Infinitesimal-subset-HFinite [THEN subsetD])
done

```

end

16 HTranscendental: Nonstandard Extensions of Transcendental Functions

```

theory HTranscendental
imports Transcendental HSeries HDeriv
begin

```

definition

```

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

```

definition

```

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

```

definition

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

16.1 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 ≤ *x*; 0 ≤ *y*]] ==>
(**f** *sqrt*)(*x*y*) = (**f** *sqrt*)(*x*) * (**f** *sqrt*)(*y*)
by (*auto intro: hypreal-sqrt-mult-distrib simp add: order-le-less*)

lemma *hypreal-sqrt-approx-zero* [*simp*]:
0 < *x* ==> ((**f** *sqrt*)(*x*) @= 0) = (*x* @= 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]:
   $0 \leq x \implies (( *f* \text{sqrt})(x) \text{@} = 0) = (x \text{@} = 0)$ 
by (auto simp add: order-le-less)

```

```

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

```

```

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

```

```

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

```

```

lemma hypreal-sqrt-ge-zero:  $0 \leq x \implies 0 \leq (*f* \text{sqrt})(x)$ 
by (auto intro: hypreal-sqrt-gt-zero simp add: order-le-less)

```

```

lemma hypreal-sqrt-hrabs [simp]:  $!!x. (*f* \text{sqrt})(x \wedge 2) = \text{abs}(x)$ 
by (transfer, simp)

```

```

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

```

```

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

```

```

lemma star-sqrt-HFinite:  $\llbracket x \in \text{HFinite}; 0 \leq x \rrbracket \implies (*f* \text{sqrt}) x \in \text{HFinite}$ 
apply (rule HFinite-square-iff [THEN iffD1])
apply (simp only: hypreal-sqrt-mult-distrib2 [symmetric], simp)
done

```

```

lemma st-hypreal-sqrt:

```

```

    [| x ∈ HFinite; 0 ≤ 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]: $\forall x y. x \leq (*f* \text{sqrt})(x^2 + y^2)$
 by transfer (rule real-sqrt-sum-squares-ge1)

lemma *HFinite-hypreal-sqrt*:
 [| 0 ≤ x; x ∈ HFinite |] ==> (*f* sqrt) x ∈ 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 ≤ x; (*f* sqrt) x ∈ HFinite |] ==> x ∈ 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]:
 $0 \leq x \implies ((*f* \text{sqrt}) x \in \text{HFinite}) = (x \in \text{HFinite})$
 by (blast intro: HFinite-hypreal-sqrt HFinite-hypreal-sqrt-imp-HFinite)

lemma *HFinite-sqrt-sum-squares* [simp]:
 $((*f* \text{sqrt})(x*x + y*y) \in \text{HFinite}) = (x*x + y*y \in \text{HFinite})$
 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]:
   $0 \leq x \implies ((\text{*f* sqrt}) x \in \text{Infinitesimal}) = (x \in \text{Infinitesimal})$ 
by (blast intro: Infinitesimal-hypreal-sqrt-imp-Infinitesimal Infinitesimal-hypreal-sqrt)

```

```

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

```

```

lemma HInfinite-hypreal-sqrt:
   $[| 0 \leq x; x \in \text{HInfinite} |] \implies (\text{*f* sqrt}) x \in \text{HInfinite}$ 
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:
   $[| 0 \leq x; (\text{*f* sqrt}) x \in \text{HInfinite} |] \implies x \in \text{HInfinite}$ 
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]:
   $0 \leq x \implies ((\text{*f* sqrt}) x \in \text{HInfinite}) = (x \in \text{HInfinite})$ 
by (blast intro: HInfinite-hypreal-sqrt HInfinite-hypreal-sqrt-imp-HInfinite)

```

```

lemma HInfinite-sqrt-sum-squares [simp]:
   $((\text{*f* sqrt})(x*x + y*y) \in \text{HInfinite}) = (x*x + y*y \in \text{HInfinite})$ 
apply (rule HInfinite-hypreal-sqrt-iff)
apply (rule add-nonneg-nonneg)
apply (auto)
done

```

```

lemma HFinite-exp [simp]:
   $\text{sumhr } (0, \text{whn}, \%n. \text{inverse } (\text{real } (\text{fact } n)) * x ^ n) \in \text{HFinite}$ 
unfolding sumhr-app
apply (simp only: star-zero-def starfun2-star-of)
apply (rule NSBseqD2)

```

```

apply (rule NSconvergent-NSBseq)
apply (rule convergent-NSconvergent-iff [THEN iffD1])
apply (rule summable-convergent-sumr-iff [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= $\lambda x. 1 \approx x$ , 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)
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= $\lambda x. 1 \approx x$ , 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)
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: diff-minus mem-infmal-iff)
done

```

```

lemma STAR-exp-epsilon [simp]: (*f* exp) epsilon @= 1
by (auto intro: STAR-exp-Infinitesimal)

```

```

lemma STAR-exp-add: !!x y. (*f* exp)(x + y) = (*f* exp) x * (*f* exp) y

```

by transfer (rule exp-add)

lemma exphr-hypreal-of-real-exp-eq: exphr $x = \text{hypreal-of-real } (\text{exp } x)$
apply (simp add: exphr-def)
apply (rule st-unique, simp)
apply (subst starfunNat-sumr [symmetric])
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]: $!!x::\text{hypreal}. 0 \leq x \implies (1 + x) \leq (*f* \text{exp}) x$
by transfer (rule exp-ge-add-one-self-aux)

lemma starfun-exp-HInfinite:
 $[[x \in \text{HInfinite}; 0 \leq x]] \implies (*f* \text{exp}) (x::\text{hypreal}) \in \text{HInfinite}$
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: $!!x. (*f* \text{exp}) (-x) = \text{inverse}((*f* \text{exp}) x)$
by transfer (rule exp-minus)

lemma starfun-exp-Infinitesimal:
 $[[x \in \text{HInfinite}; x \leq 0]] \implies (*f* \text{exp}) (x::\text{hypreal}) \in \text{Infinitesimal}$
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]: $!!x::\text{hypreal}. 0 < x \implies 1 < (*f* \text{exp}) x$
by transfer (rule exp-gt-one)

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

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

lemma starfun-exp-ln-eq: $!!u x. (*f* \text{exp}) u = x \implies (*f* \text{ln}) x = u$
by transfer (rule ln-unique)

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

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

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

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

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

lemma *starfun-ln-inverse*: !!x. $0 < x \implies (*f* \ln) (\text{inverse } x) = -(*f* \ln) x$
by *transfer* (rule *ln-inverse*)

lemma *starfun-abs-exp-cancel*: $\bigwedge x. |(*f* \exp) (x::\text{hypreal})| = (*f* \exp) x$
by *transfer* (rule *abs-exp-cancel*)

lemma *starfun-exp-less-mono*: $\bigwedge x y::\text{hypreal}. x < y \implies (*f* \exp) x < (*f* \exp) y$
by *transfer* (rule *exp-less-mono*)

lemma *starfun-exp-HFinite*: $x \in HFinite \implies (*f* \exp) (x::\text{hypreal}) \in HFinite$
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*:
 $[x \in \text{Infinitesimal}; z \in HFinite] \implies (*f* \exp) (z + x::\text{hypreal}) \textcircled{=} (*f* \exp) z$
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*:
 $[x \in HInfinite; 0 < x] \implies (*f* \ln) x \in HInfinite$
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:
   $x \in HInfinite \implies (*f* \exp) x \in HInfinite \mid (*f* \exp) (x::hypreal) \in Infinitesimal$ 
apply (insert linorder-linear [of x 0])
apply (auto intro: starfun-exp-HInfinite starfun-exp-Infinitesimal)
done

```

```

lemma starfun-ln-HFinite-not-Infinitesimal:
   $[\mid x \in HFinite - Infinitesimal; 0 < x \mid] \implies (*f* \ln) x \in HFinite$ 
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:
   $[\mid x \in Infinitesimal; 0 < x \mid] \implies (*f* \ln) x \in HInfinite$ 
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.  $[\mid 0 < x; x < 1 \mid] \implies (*f* \ln) x < 0$ 
by transfer (rule ln-less-zero)

```

```

lemma starfun-ln-Infinitesimal-less-zero:
   $[\mid x \in Infinitesimal; 0 < x \mid] \implies (*f* \ln) x < 0$ 
by (auto intro!: starfun-ln-less-zero simp add: Infinitesimal-def)

```

```

lemma starfun-ln-HInfinite-gt-zero:
   $[\mid x \in HInfinite; 0 < x \mid] \implies 0 < (*f* \ln) x$ 
by (auto intro!: starfun-ln-gt-zero simp add: HInfinite-def)

```

```

lemma HFinite-sin [simp]:  $\text{sumhr } (0, \text{whn}, \%n. \text{sin-coeff } n * x ^ n) \in HFinite$ 
unfolding sumhr-app
apply (simp only: star-zero-def starfun2-star-of)
apply (rule NSBseqD2)
apply (rule NSconvergent-NSBseq)
apply (rule convergent-NSconvergent-iff [THEN iffD1])
apply (rule summable-convergent-sumr-iff [THEN iffD1])

```

apply (rule summable-sin)
done

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

lemma STAR-sin-Infinitesimal [simp]: $x \in \text{Infinitesimal} \implies (*f* \text{ sin}) x @= x$
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]: $\text{sumhr } (0, \text{whn}, \%n. \text{cos-coeff } n * x ^ n) \in \text{HFinite}$
unfolding sumhr-app
apply (simp only: star-zero-def starfun2-star-of)
apply (rule NSBseqD2)
apply (rule NSconvergent-NSBseq)
apply (rule convergent-NSconvergent-iff [THEN iffD1])
apply (rule summable-convergent-sumr-iff [THEN iffD1])
apply (rule summable-cos)
done

lemma STAR-cos-zero [simp]: ($*f*$ cos) 0 = 1
by transfer (rule cos-zero)

lemma STAR-cos-Infinitesimal [simp]: $x \in \text{Infinitesimal} \implies (*f* \text{ cos}) x @= 1$
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 add: diff-minus)
done

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

lemma STAR-tan-Infinitesimal: $x \in \text{Infinitesimal} \implies (*f* \text{ tan}) x @= x$
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:
  x ∈ Infinitesimal ==> (*f* sin) x * (*f* cos) x @= x
apply (insert approx-mult-HFinite [of (*f* sin) x - (*f* cos) x 1])
apply (simp add: Infinitesimal-subset-HFinite [THEN subsetD])
done

```

```

lemma HFinite-pi: hypreal-of-real pi ∈ HFinite
by simp

```

```

lemma lemma-split-hypreal-of-real:
  N ∈ HNatInfinite
  ==> hypreal-of-real a =
    hypreal-of-hypnat N * (inverse(hypreal-of-hypnat N) * hypreal-of-real a)
by (simp add: mult-assoc [symmetric] zero-less-HNatInfinite)

```

```

lemma STAR-sin-Infinitesimal-divide:
  [|x ∈ Infinitesimal; x ≠ 0|] ==> (*f* sin) x/x @= 1
apply (cut-tac x = 0 in DERIV-sin)
apply (simp add: NSDERIV-DERIV-iff [symmetric] nsderiv-def)
done

```

```

lemma lemma-sin-pi:
  n ∈ HNatInfinite
  ==> (*f* sin) (inverse (hypreal-of-hypnat n))/(inverse (hypreal-of-hypnat
n)) @= 1
apply (rule STAR-sin-Infinitesimal-divide)
apply (auto simp add: zero-less-HNatInfinite)
done

```

```

lemma STAR-sin-inverse-HNatInfinite:
  n ∈ HNatInfinite
  ==> (*f* sin) (inverse (hypreal-of-hypnat n)) * hypreal-of-hypnat n @= 1
apply (frule lemma-sin-pi)
apply (simp add: divide-inverse)
done

```

```

lemma Infinitesimal-pi-divide-HNatInfinite:

```

```

    N ∈ HNatInfinite
    ==> hypreal-of-real pi / (hypreal-of-hypnat N) ∈ Infinitesimal
apply (simp add: divide-inverse)
apply (auto intro: Infinitesimal-HFinite-mult2)
done

lemma pi-divide-HNatInfinite-not-zero [simp]:
    N ∈ HNatInfinite ==> hypreal-of-real pi / (hypreal-of-hypnat N) ≠ 0
by (simp add: zero-less-HNatInfinite)

lemma STAR-sin-pi-divide-HNatInfinite-approx-pi:
    n ∈ HNatInfinite
    ==> ( *f* sin ) ( hypreal-of-real pi / ( hypreal-of-hypnat n ) ) * hypreal-of-hypnat
n
    @= hypreal-of-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 mult-ac)
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
```

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
```

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:
  x ∈ Infinitesimal ==> (*f* cos) x @= 1 - x ^ 2
apply (rule STAR-cos-Infinitesimal [THEN approx-trans])
apply (auto simp add: Infinitesimal-approx-minus [symmetric]
      diff-minus add-assoc [symmetric] numeral-2-eq-2)
done
```

done

```
lemma STAR-cos-Infinitesimal-approx2:
  x ∈ Infinitesimal ==> (*f* cos) x @= 1 - (x ^ 2)/2
apply (rule STAR-cos-Infinitesimal [THEN approx-trans])
apply (auto intro: Infinitesimal-SReal-divide
      simp add: Infinitesimal-approx-minus [symmetric] numeral-2-eq-2)
done
```

done

end

17 NSCA: 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*)

17.1 Closure Laws for SComplex, the Standard Complex Numbers

lemma *SComplex-minus-iff* [*simp*]: $(-x \in SComplex) = (x \in SComplex)$
by (*auto*, *drule* *Standard-minus*, *auto*)

lemma *SComplex-add-cancel*:

$[[x + y \in SComplex; y \in SComplex]] \implies x \in SComplex$
by (*drule* (1) *Standard-diff*, *simp*)

lemma *SReal-hcmod-hcomplex-of-complex* [*simp*]:

$hcmod (hcomplex-of-complex\ r) \in Reals$
by (*simp* *add*: *Reals-eq-Standard*)

lemma *SReal-hcmod-numeral* [*simp*]: $hcmod (numeral\ w :: hcomplex) \in Reals$

by (*simp* *add*: *Reals-eq-Standard*)

lemma *SReal-hcmod-SComplex*: $x \in SComplex \implies hcmod\ x \in Reals$

by (*simp* *add*: *Reals-eq-Standard*)

lemma *SComplex-divide-numeral*:

$r \in SComplex \implies r / (numeral\ w :: hcomplex) \in SComplex$
by *simp*

lemma *SComplex-UNIV-complex*:

$\{x. hcomplex-of-complex\ x \in SComplex\} = (UNIV :: complex\ set)$
by *simp*

lemma *SComplex-iff*: $(x \in SComplex) = (\exists y. x = hcomplex-of-complex\ y)$

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*: $inv\ hcomplex-of-complex\ 'SComplex = UNIV$

apply (*auto* *simp* *add*: *Standard-def image-def*)

apply (*rule* *inj-hcomplex-of-complex* [*THEN* *inv-f-f*, *THEN* *subst*], *blast*)

done

lemma *SComplex-hcomplex-of-complex-image*:

[[$\exists x. x: P; P \leq SComplex$]] ==> $\exists Q. P = hcomplex\text{-of-complex} \text{ ‘ } Q$
apply (*simp add: Standard-def, blast*)
done

lemma *SComplex-SReal-dense*:

[[$x \in SComplex; y \in SComplex; hmod\ x < hmod\ y$]]
 ==> $\exists r \in Reals. hmod\ x < r \ \& \ r < hmod\ y$

apply (*auto intro: SReal-dense simp add: SReal-hmod-SComplex*)
done

lemma *SComplex-hmod-SReal*:

$z \in SComplex ==> hmod\ z \in Reals$

by (*simp add: Reals-eq-Standard*)

17.2 The Finite Elements form a Subring

lemma *HFinite-hmod-hcomplex-of-complex* [*simp*]:

$hmod\ (hcomplex\text{-of-complex}\ r) \in HFinite$

by (*auto intro!: SReal-subset-HFinite [THEN subsetD]*)

lemma *HFinite-hmod-iff*: $(x \in HFinite) = (hmod\ x \in HFinite)$

by (*simp add: HFinite-def*)

lemma *HFinite-bounded-hmod*:

[[$x \in HFinite; y \leq hmod\ x; 0 \leq y$]] ==> $y: HFinite$

by (*auto intro: HFinite-bounded simp add: HFinite-hmod-iff*)

17.3 The Complex Infinitesimals form a Subring

lemma *hcomplex-sum-of-halves*: $x/(2::hcomplex) + x/(2::hcomplex) = x$

by *auto*

lemma *Infinitesimal-hmod-iff*:

$(z \in Infinitesimal) = (hmod\ z \in Infinitesimal)$

by (*simp add: Infinitesimal-def*)

lemma *HInfinite-hmod-iff*: $(z \in HInfinite) = (hmod\ z \in HInfinite)$

by (*simp add: HInfinite-def*)

lemma *HFinite-diff-Infinitesimal-hmod*:

$x \in HFinite - Infinitesimal ==> hmod\ x \in HFinite - Infinitesimal$

by (*simp add: HFinite-hmod-iff Infinitesimal-hmod-iff*)

lemma *hmod-less-Infinitesimal*:

[[$e \in Infinitesimal; hmod\ x < hmod\ e$]] ==> $x \in Infinitesimal$

by (*auto elim: hrabs-less-Infinitesimal simp add: Infinitesimal-hmod-iff*)

lemma *hmod-le-Infinitesimal*:

[[$e \in Infinitesimal; hmod\ x \leq hmod\ e$]] ==> $x \in Infinitesimal$

by (*auto elim: hrabs-le-Infinitesimal simp add: Infinitesimal-hmod-iff*)

lemma *Infinitesimal-interval-hcmod*:

[[$e \in \text{Infinitesimal}$;
 $e' \in \text{Infinitesimal}$;
 $\text{hcmod } e' < \text{hcmod } x$; $\text{hcmod } x < \text{hcmod } e$
]] ==> $x \in \text{Infinitesimal}$

by (*auto intro: Infinitesimal-interval simp add: Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-interval2-hcmod*:

[[$e \in \text{Infinitesimal}$;
 $e' \in \text{Infinitesimal}$;
 $\text{hcmod } e' \leq \text{hcmod } x$; $\text{hcmod } x \leq \text{hcmod } e$
]] ==> $x \in \text{Infinitesimal}$

by (*auto intro: Infinitesimal-interval2 simp add: Infinitesimal-hcmod-iff*)

17.4 The “Infinitely Close” Relation

lemma *approx-SComplex-mult-cancel-zero*:

[[$a \in \text{SComplex}$; $a \neq 0$; $a * x @= 0$]] ==> $x @= 0$

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*: [[$a \in \text{SComplex}$; $x @= 0$]] ==> $x * a @= 0$

by (*auto dest: Standard-subset-HFinite [THEN subsetD] approx-mult1*)

lemma *approx-mult-SComplex2*: [[$a \in \text{SComplex}$; $x @= 0$]] ==> $a * x @= 0$

by (*auto dest: Standard-subset-HFinite [THEN subsetD] approx-mult2*)

lemma *approx-mult-SComplex-zero-cancel-iff [simp]*:

[[$a \in \text{SComplex}$; $a \neq 0$]] ==> $(a * x @= 0) = (x @= 0)$

by (*blast intro: approx-SComplex-mult-cancel-zero approx-mult-SComplex2*)

lemma *approx-SComplex-mult-cancel*:

[[$a \in \text{SComplex}$; $a \neq 0$; $a * w @= a * z$]] ==> $w @= z$

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]*:

[[$a \in \text{SComplex}$; $a \neq 0$]] ==> $(a * w @= a * z) = (w @= z)$

by (*auto intro!: approx-mult2 Standard-subset-HFinite [THEN subsetD]*)

intro: approx-SComplex-mult-cancel)

lemma *approx-hcmod-approx-zero*: $(x @= y) = (\text{hcmod } (y - x) @= 0)$

apply (*subst hnorm-minus-commute*)

apply (*simp add: approx-def Infinitesimal-hcmod-iff diff-minus*)

done

lemma *approx-approx-zero-iff*: $(x \text{ @} = 0) = (\text{hcm}od\ x \text{ @} = 0)$
by (*simp add: approx-hcm-approx-zero*)

lemma *approx-minus-zero-cancel-iff* [*simp*]: $(-x \text{ @} = 0) = (x \text{ @} = 0)$
by (*simp add: approx-def*)

lemma *Infinesimal-hcm-add-diff*:

$u \text{ @} = 0 \implies \text{hcm}od(x + u) - \text{hcm}od\ x \in \text{Infinesimal}$
apply (*drule approx-approx-zero-iff [THEN iffD1]*)
apply (*rule-tac e = hcm u and e' = - hcm u in Infinesimal-interval2*)
apply (*auto simp add: mem-infmal-iff [symmetric] diff-minus*)
apply (*rule-tac c1 = hcm x in add-le-cancel-left [THEN iffD1]*)
apply (*auto simp add: diff-minus [symmetric]*)
done

lemma *approx-hcm-add-hcm*: $u \text{ @} = 0 \implies \text{hcm}od(x + u) \text{ @} = \text{hcm}od\ x$
apply (*rule approx-minus-iff [THEN iffD2]*)
apply (*auto intro: Infinesimal-hcm-add-diff simp add: mem-infmal-iff [symmetric] diff-minus [symmetric]*)
done

17.5 Zero is the Only Infinitesimal Complex Number

lemma *Infinesimal-less-SComplex*:

$[x \in \text{SComplex}; y \in \text{Infinesimal}; 0 < \text{hcm}od\ x] \implies \text{hcm}od\ y < \text{hcm}od\ x$
by (*auto intro: Infinesimal-less-SReal SComplex-hcm-SReal simp add: Infinesimal-hcm-iff*)

lemma *SComplex-Int-Infinesimal-zero*: $\text{SComplex Int Infinesimal} = \{0\}$
by (*auto simp add: Standard-def Infinesimal-hcm-iff*)

lemma *SComplex-Infinesimal-zero*:

$[x \in \text{SComplex}; x \in \text{Infinesimal}] \implies x = 0$
by (*cut-tac SComplex-Int-Infinesimal-zero, blast*)

lemma *SComplex-HFinite-diff-Infinesimal*:

$[x \in \text{SComplex}; x \neq 0] \implies x \in \text{HFinite} - \text{Infinesimal}$
by (*auto dest: SComplex-Infinesimal-zero Standard-subset-HFinite [THEN subsetD]*)

lemma *hcomplex-of-complex-HFinite-diff-Infinesimal*:

$\text{hcomplex-of-complex}\ x \neq 0$
 $\implies \text{hcomplex-of-complex}\ x \in \text{HFinite} - \text{Infinesimal}$
by (*rule SComplex-HFinite-diff-Infinesimal, auto*)

lemma *numeral-not-Infinesimal* [*simp*]:

$\text{numeral}\ w \neq (0::\text{hcomplex}) \implies (\text{numeral}\ w::\text{hcomplex}) \notin \text{Infinesimal}$
by (*fast dest: Standard-numeral [THEN SComplex-Infinesimal-zero]*)

lemma *approx-SComplex-not-zero*:

$\llbracket y \in SComplex; x @= y; y \neq 0 \rrbracket \implies x \neq 0$

by (*auto dest: SComplex-Infinesimal-zero approx-sym [THEN mem-infmal-iff [THEN iffD2]]*)

lemma *SComplex-approx-iff*:

$\llbracket x \in SComplex; y \in SComplex \rrbracket \implies (x @= y) = (x = y)$

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

lemma *numeral-Infinesimal-iff [simp]*:

$((\text{numeral } w :: hcomplex) \in \text{Infinesimal}) =$
 $(\text{numeral } w = (0 :: hcomplex))$

apply (*rule iffI*)

apply (*fast dest: Standard-numeral [THEN SComplex-Infinesimal-zero]*)

apply (*simp (no-asm-simp)*)

done

lemma *approx-unique-complex*:

$\llbracket r \in SComplex; s \in SComplex; r @= x; s @= x \rrbracket \implies r = s$

by (*blast intro: SComplex-approx-iff [THEN iffD1] approx-trans2*)

17.6 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 *Infinesimal-hRe*: $x \in \text{Infinesimal} \implies hRe\ x \in \text{Infinesimal}$

apply (*rule InfinesimalI2, simp*)

apply (*rule order-le-less-trans [OF abs-hRe-le-hcmod]*)

apply (*erule (1) InfinesimalD2*)

done

lemma *Infinesimal-hIm*: $x \in \text{Infinesimal} \implies hIm\ x \in \text{Infinesimal}$

apply (*rule InfinesimalI2, simp*)

apply (*rule order-le-less-trans [OF abs-hIm-le-hcmod]*)

apply (*erule (1) InfinesimalD2*)

done

lemma *real-sqrt-lessI*: $\llbracket 0 < u; x < u^2 \rrbracket \implies \text{sqrt } x < u$

by (*frule real-sqrt-less-mono*) *simp*

lemma *hypreal-sqrt-lessI*:

$\bigwedge x\ u. \llbracket 0 < u; x < u^2 \rrbracket \implies (*f* \text{sqrt})\ x < u$

by *transfer (rule real-sqrt-lessI)*

lemma *hypreal-sqrt-ge-zero*: $\bigwedge x. 0 \leq x \implies 0 \leq (*f* \text{ sqrt}) x$
by *transfer* (rule *real-sqrt-ge-zero*)

lemma *Infinitesimal-sqrt*:
 $\llbracket x \in \text{Infinitesimal}; 0 \leq x \rrbracket \implies (*f* \text{ sqrt}) x \in \text{Infinitesimal}$
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*:
 $\llbracket x \in \text{Infinitesimal}; y \in \text{Infinitesimal} \rrbracket \implies \text{HComplex } x y \in \text{Infinitesimal}$
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*:
 $(x \in \text{Infinitesimal}) = (\text{hRe } x \in \text{Infinitesimal} \wedge \text{hIm } x \in \text{Infinitesimal})$
apply (*safe intro!: Infinitesimal-hRe Infinitesimal-hIm*)
apply (drule (1) *Infinitesimal-HComplex*, *simp*)
done

lemma *hRe-diff* [*simp*]: $\bigwedge x y. \text{hRe } (x - y) = \text{hRe } x - \text{hRe } y$
by *transfer* (rule *complex-Re-diff*)

lemma *hIm-diff* [*simp*]: $\bigwedge x y. \text{hIm } (x - y) = \text{hIm } x - \text{hIm } y$
by *transfer* (rule *complex-Im-diff*)

lemma *approx-hRe*: $x \approx y \implies \text{hRe } x \approx \text{hRe } y$
unfolding *approx-def* **by** (drule *Infinitesimal-hRe*) *simp*

lemma *approx-hIm*: $x \approx y \implies \text{hIm } x \approx \text{hIm } y$
unfolding *approx-def* **by** (drule *Infinitesimal-hIm*) *simp*

lemma *approx-HComplex*:
 $\llbracket a \approx b; c \approx d \rrbracket \implies \text{HComplex } a c \approx \text{HComplex } b d$
unfolding *approx-def* **by** (*simp add: Infinitesimal-HComplex*)

lemma *hcomplex-approx-iff*:
 $(x \approx y) = (\text{hRe } x \approx \text{hRe } y \wedge \text{hIm } x \approx \text{hIm } y)$
unfolding *approx-def* **by** (*simp add: hcomplex-Infinitesimal-iff*)

lemma *HFinite-hRe*: $x \in \text{HFinite} \implies \text{hRe } x \in \text{HFinite}$
apply (*auto simp add: HFinite-def SReal-def*)

apply (*rule-tac* $x = \text{star-of } r$ **in** exI , *simp*)
apply (*erule* *order-le-less-trans* [*OF* *abs-hRe-le-hcmod*])
done

lemma *HFinite-hIm*: $x \in HFinite \implies hIm\ x \in HFinite$
apply (*auto* *simp* *add*: *HFinite-def* *SReal-def*)
apply (*rule-tac* $x = \text{star-of } r$ **in** exI , *simp*)
apply (*erule* *order-le-less-trans* [*OF* *abs-hIm-le-hcmod*])
done

lemma *HFinite-HComplex*:
 $\llbracket x \in HFinite; y \in HFinite \rrbracket \implies HComplex\ x\ y \in HFinite$
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*:
 $(x \in HInfinite) = (hRe\ x \in HInfinite \vee hIm\ x \in HInfinite)$
by (*simp* *add*: *HInfinite-HFinite-iff* *hcomplex-HFinite-iff*)

lemma *hcomplex-of-hypreal-approx-iff* [*simp*]:
 $(hcomplex\ of\ hypreal\ x\ @ = hcomplex\ of\ hypreal\ z) = (x\ @ = z)$
by (*simp* *add*: *hcomplex-approx-iff*)

lemma *Standard-HComplex*:
 $\llbracket x \in Standard; y \in Standard \rrbracket \implies HComplex\ x\ y \in Standard$
by (*simp* *add*: *HComplex-def*)

lemma *stc-part-Ex*: $x : HFinite \implies \exists t \in SComplex. x\ @ = t$
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*: $x : HFinite \implies EX! t. t \in SComplex \ \& \ x\ @ = t$
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*]

17.7 Theorems About Monads

lemma *monad-zero-hcmod-iff*: $(x \in \text{monad } 0) = (\text{hcmod } x:\text{monad } 0)$
by (*simp add: Infinitesimal-monad-zero-iff [symmetric] Infinitesimal-hcmod-iff*)

17.8 Theorems About Standard Part

lemma *stc-approx-self*: $x \in \text{HFinite} \implies \text{stc } x @= x$
apply (*simp add: stc-def*)
apply (*frule stc-part-Ex, safe*)
apply (*rule someI2*)
apply (*auto intro: approx-sym*)
done

lemma *stc-SComplex*: $x \in \text{HFinite} \implies \text{stc } x \in \text{SComplex}$
apply (*simp add: stc-def*)
apply (*frule stc-part-Ex, safe*)
apply (*rule someI2*)
apply (*auto intro: approx-sym*)
done

lemma *stc-HFinite*: $x \in \text{HFinite} \implies \text{stc } x \in \text{HFinite}$
by (*erule stc-SComplex [THEN Standard-subset-HFinite [THEN subsetD]]*)

lemma *stc-unique*: $\llbracket y \in \text{SComplex}; y \approx x \rrbracket \implies \text{stc } x = y$
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*]: $x \in \text{SComplex} \implies \text{stc } x = x$
apply (*erule stc-unique*)
apply (*rule approx-refl*)
done

lemma *stc-hcomplex-of-complex*:
 $\text{stc } (\text{hcomplex-of-complex } x) = \text{hcomplex-of-complex } x$
by *auto*

lemma *stc-eq-approx*:
 $\llbracket x \in \text{HFinite}; y \in \text{HFinite}; \text{stc } x = \text{stc } y \rrbracket \implies x @= y$
by (*auto dest!: stc-approx-self elim!: approx-trans3*)

lemma *approx-stc-eq*:

$$[[x \in HFinite; y \in HFinite; x @= y]] ==> stc\ x = stc\ y$$
by (*blast intro: approx-trans approx-trans2 SComplex-approx-iff [THEN iffD1]*
dest: stc-approx-self stc-SComplex)

lemma *stc-eq-approx-iff*:

$$[[x \in HFinite; y \in HFinite]] ==> (x @= y) = (stc\ x = stc\ y)$$
by (*blast intro: approx-stc-eq stc-eq-approx*)

lemma *stc-Infinitesimal-add-SComplex*:

$$[[x \in SComplex; e \in Infinitesimal]] ==> stc(x + e) = x$$
apply (*erule stc-unique*)
apply (*erule Infinitesimal-add-approx-self*)
done

lemma *stc-Infinitesimal-add-SComplex2*:

$$[[x \in SComplex; e \in Infinitesimal]] ==> stc(e + x) = x$$
apply (*erule stc-unique*)
apply (*erule Infinitesimal-add-approx-self2*)
done

lemma *HFinite-stc-Infinitesimal-add*:

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

lemma *stc-add*:

$$[[x \in HFinite; y \in HFinite]] ==> stc(x + y) = stc(x) + stc(y)$$
by (*simp add: stc-unique stc-SComplex stc-approx-self approx-add*)

lemma *stc-numeral [simp]*: $stc(\text{numeral } w) = \text{numeral } w$

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*: $y \in HFinite ==> stc(-y) = -stc(y)$

by (*simp add: stc-unique stc-SComplex stc-approx-self approx-minus*)

lemma *stc-diff*:

$$[[x \in HFinite; y \in HFinite]] ==> stc(x - y) = stc(x) - stc(y)$$
by (*simp add: stc-unique stc-SComplex stc-approx-self approx-diff*)

lemma *stc-mult*:

$$[[x \in HFinite; y \in HFinite]] ==> stc(x * y) = stc(x) * stc(y)$$
by (*simp add: stc-unique stc-SComplex stc-approx-self approx-mult-HFinite*)

lemma *stc-Infinitesimal*: $x \in \text{Infinitesimal} \implies \text{stc } x = 0$
by (*simp add: stc-unique mem-infmal-iff*)

lemma *stc-not-Infinitesimal*: $\text{stc}(x) \neq 0 \implies x \notin \text{Infinitesimal}$
by (*fast intro: stc-Infinitesimal*)

lemma *stc-inverse*:
 $\llbracket x \in \text{HFinite}; \text{stc } x \neq 0 \rrbracket$
 $\implies \text{stc}(\text{inverse } x) = \text{inverse } (\text{stc } x)$
apply (*drule stc-not-Infinitesimal*)
apply (*simp add: stc-unique stc-SComplex stc-approx-self approx-inverse*)
done

lemma *stc-divide* [*simp*]:
 $\llbracket x \in \text{HFinite}; y \in \text{HFinite}; \text{stc } y \neq 0 \rrbracket$
 $\implies \text{stc}(x/y) = (\text{stc } x) / (\text{stc } y)$
by (*simp add: divide-inverse stc-mult stc-not-Infinitesimal HFinite-inverse stc-inverse*)

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

lemma *HFinite-HFinite-hcomplex-of-hypreal*:
 $z \in \text{HFinite} \implies \text{hcomplex-of-hypreal } z \in \text{HFinite}$
by (*simp add: hcomplex-HFinite-iff*)

lemma *SComplex-SReal-hcomplex-of-hypreal*:
 $x \in \text{Reals} \implies \text{hcomplex-of-hypreal } x \in \text{SComplex}$
apply (*rule Standard-of-hypreal*)
apply (*simp add: Reals-eq-Standard*)
done

lemma *stc-hcomplex-of-hypreal*:
 $z \in \text{HFinite} \implies \text{stc}(\text{hcomplex-of-hypreal } z) = \text{hcomplex-of-hypreal } (\text{st } z)$
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*]:
 $(\text{hcnj } z \in \text{Infinitesimal}) = (z \in \text{Infinitesimal})$
by (*simp add: Infinitesimal-hcmmod-iff*)

lemma *Infinitesimal-hcomplex-of-hypreal-epsilon* [*simp*]:
 $\text{hcomplex-of-hypreal } \text{epsilon} \in \text{Infinitesimal}$
by (*simp add: Infinitesimal-hcmmod-iff*)

end

18 CStar: Star-transforms in NSA, Extending Sets of Complex Numbers and Complex Functions

theory CStar
imports NSCA
begin

18.1 Properties of the *-Transform Applied to Sets of Reals

lemma STARC-hcomplex-of-complex-Int:

$$*s* X \text{ Int } S\text{Complex} = h\text{complex-of-complex } ' X$$
 by (auto simp add: Standard-def)

lemma lemma-not-hcomplexA:

$$x \notin h\text{complex-of-complex } ' A \implies \forall y \in A. x \neq h\text{complex-of-complex } y$$
 by auto

18.2 Theorems about Nonstandard Extensions of Functions

lemma starfunC-hcpow: $!!Z. (*f* (\%z. z \wedge n)) Z = Z \text{ pow hypnat-of-nat } n$
 by transfer (rule refl)

lemma starfunCR-cmod: $*f* \text{ cmod} = \text{hcmmod}$
 by transfer (rule refl)

18.3 Internal Functions - Some Redundancy With *f* Now

lemma starfun-Re: $(*f* (\lambda x. \text{Re } (f x))) = (\lambda x. h\text{Re } ((*f* f) x))$
 by transfer (rule refl)

lemma starfun-Im: $(*f* (\lambda x. \text{Im } (f x))) = (\lambda x. h\text{Im } ((*f* f) x))$
 by transfer (rule refl)

lemma starfunC-eq-Re-Im-iff:

$$((*f* f) x = z) = (((*f* (\%x. \text{Re}(f x))) x = h\text{Re } (z)) \& \\ ((*f* (\%x. \text{Im}(f x))) x = h\text{Im } (z)))$$
 by (simp add: hcomplex-hRe-hIm-cancel-iff starfun-Re starfun-Im)

lemma starfunC-approx-Re-Im-iff:

$$((*f* f) x @= z) = (((*f* (\%x. \text{Re}(f x))) x @= h\text{Re } (z)) \& \\ ((*f* (\%x. \text{Im}(f x))) x @= h\text{Im } (z)))$$
 by (simp add: hcomplex-approx-iff starfun-Re starfun-Im)

end

19 CLim: 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]:
   $x \neq (0::\text{complex}) \implies (x * \text{inverse}(x) ^ 2) = \text{inverse } x$ 
by (simp add: numeral-2-eq-2)
```

Changing the quantified variable. Install earlier?

```
lemma all-shift:  $(\forall x::'a::\text{comm-ring-1}. P x) = (\forall x. P (x-a))$ 
apply auto
apply (drule-tac x=x+a in spec)
apply (simp add: diff-minus add-assoc)
done
```

```
lemma complex-add-minus-iff [simp]:  $(x + - a = (0::\text{complex})) = (x=a)$ 
by (simp add: diff-eq-eq diff-minus [symmetric])
```

```
lemma complex-add-eq-0-iff [iff]:  $(x+y = (0::\text{complex})) = (y = -x)$ 
apply auto
apply (drule sym [THEN diff-eq-eq [THEN iffD2]], auto)
done
```

19.1 Limit of Complex to Complex Function

```
lemma NSLIM-Re:  $f \text{ -- } a \text{ --NS} > L \implies (\%x. \text{Re}(f x)) \text{ -- } a \text{ --NS} > \text{Re}(L)$ 
by (simp add: NSLIM-def starfunC-approx-Re-Im-iff
  hRe-hcomplex-of-complex)
```

```
lemma NSLIM-Im:  $f \text{ -- } a \text{ --NS} > L \implies (\%x. \text{Im}(f x)) \text{ -- } a \text{ --NS} > \text{Im}(L)$ 
by (simp add: NSLIM-def starfunC-approx-Re-Im-iff
  hIm-hcomplex-of-complex)
```

```
lemma LIM-Re:
  fixes  $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$ 
  shows  $f \text{ -- } a \text{ --} > L \implies (\%x. \text{Re}(f x)) \text{ -- } a \text{ --} > \text{Re}(L)$ 
by (simp add: LIM-NSLIM-iff NSLIM-Re)
```

```
lemma LIM-Im:
  fixes  $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$ 
  shows  $f \text{ -- } a \text{ --} > L \implies (\%x. \text{Im}(f x)) \text{ -- } a \text{ --} > \text{Im}(L)$ 
```

by (simp add: LIM-NSLIM-iff NSLIM-Im)

lemma LIM-cnj:

fixes $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$
 shows $f \dashv\dashv a \dashv\dashv L \implies (\%x. \text{cnj } (f x)) \dashv\dashv a \dashv\dashv \text{cnj } L$
 by (simp add: LIM-eq complex-cnj-diff [symmetric])

lemma LIM-cnj-iff:

fixes $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$
 shows $(\%x. \text{cnj } (f x)) \dashv\dashv a \dashv\dashv \text{cnj } L = (f \dashv\dashv a \dashv\dashv L)$
 by (simp add: LIM-eq complex-cnj-diff [symmetric])

lemma starfun-norm: $(*f* (\lambda x. \text{norm } (f x))) = (\lambda x. \text{hnorm } ((*f* f) x))$
 by transfer (rule refl)

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

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

lemma NSCLIM-NSCRLIM-iff:

$(f \dashv\dashv x \dashv\dashv \text{NS} > L) = ((\%y. \text{cmod}(f y - L)) \dashv\dashv x \dashv\dashv \text{NS} > 0)$
 by (simp add: NSLIM-def starfun-norm
 approx-approx-zero-iff [symmetric] approx-minus-iff [symmetric])

lemma CLIM-CRLIM-iff:

fixes $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$
 shows $(f \dashv\dashv x \dashv\dashv L) = ((\%y. \text{cmod}(f y - L)) \dashv\dashv x \dashv\dashv 0)$
 by (simp add: LIM-eq)

lemma NSCLIM-NSCRLIM-iff2:

$(f \dashv\dashv x \dashv\dashv \text{NS} > L) = ((\%y. \text{cmod}(f y - L)) \dashv\dashv x \dashv\dashv \text{NS} > 0)$
 by (simp add: LIM-NSLIM-iff [symmetric] CLIM-CRLIM-iff)

lemma NSLIM-NSCRLIM-Re-Im-iff:

$(f \dashv\dashv a \dashv\dashv \text{NS} > L) = ((\%x. \text{Re}(f x)) \dashv\dashv a \dashv\dashv \text{NS} > \text{Re}(L) \ \&$
 $(\%x. \text{Im}(f x)) \dashv\dashv a \dashv\dashv \text{NS} > \text{Im}(L))$

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:

fixes $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$
 shows $(f \dashv\dashv a \dashv\dashv L) = ((\%x. \text{Re}(f x)) \dashv\dashv a \dashv\dashv \text{Re}(L) \ \&$
 $(\%x. \text{Im}(f x)) \dashv\dashv a \dashv\dashv \text{Im}(L))$

by (simp add: LIM-NSLIM-iff NSLIM-NSCRLIM-Re-Im-iff)

19.2 Continuity

lemma *NSLIM-isContc-iff*:

($f \text{ --- } a \text{ ---NS} > f a$) = (($\%h. f(a + h)$) --- 0 ---NS > $f a$)

by (rule *NSLIM-h-iff*)

19.3 Functions from Complex to Reals

lemma *isNSContCR-cmod* [simp]: *isNSCont cmod* (a)

by (auto intro: approx-hnorm

simp add: starfunCR-cmod hmod-hcomplex-of-complex [symmetric]
isNSCont-def)

lemma *isContCR-cmod* [simp]: *isCont cmod* (a)

by (simp add: isNSCont-isCont-iff [symmetric])

lemma *isCont-Re*:

fixes $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$

shows $\text{isCont } f a \implies \text{isCont } (\%x. \text{Re } (f x)) a$

by (simp add: isCont-def LIM-Re)

lemma *isCont-Im*:

fixes $f :: 'a::\text{real-normed-vector} \Rightarrow \text{complex}$

shows $\text{isCont } f a \implies \text{isCont } (\%x. \text{Im } (f x)) a$

by (simp add: isCont-def LIM-Im)

19.4 Differentiation of Natural Number Powers

lemma *CDERIV-pow* [simp]:

$\text{DERIV } (\%x. x \wedge n) x := (\text{complex-of-real } (\text{real } n)) * (x \wedge (n - \text{Suc } 0))$

apply (induct n)

apply (drule-tac [2] *DERIV-ident* [THEN *DERIV-mult*])

apply (auto simp add: left-distrib real-of-nat-Suc)

apply (case-tac n)

apply (auto simp add: mult-ac add-commute)

done

Nonstandard version

lemma *NSCDERIV-pow*:

$\text{NSDERIV } (\%x. x \wedge n) x := \text{complex-of-real } (\text{real } n) * (x \wedge (n - 1))$

by (simp add: NSDERIV-DERIV-iff)

Can't relax the premise $x \neq (0::'a)$: it isn't continuous at zero

lemma *NSCDERIV-inverse*:

$(x::\text{complex}) \neq 0 \implies \text{NSDERIV } (\%x. \text{inverse}(x)) x := -(\text{inverse } x \wedge 2)$

unfolding numeral-2-eq-2

by (rule *NSDERIV-inverse*)

lemma *CDERIV-inverse:*

$(x::\text{complex}) \neq 0 \implies \text{DERIV } (\%x. \text{inverse}(x)) x := (- (\text{inverse } x \wedge 2))$

unfolding *numeral-2-eq-2*

by (*rule DERIV-inverse*)

19.5 Derivative of Reciprocals (Function *inverse*)

lemma *CDERIV-inverse-fun:*

$[[\text{DERIV } f x := d; f(x) \neq (0::\text{complex})]]$

$\implies \text{DERIV } (\%x. \text{inverse}(f x)) x := (- (d * \text{inverse}(f(x) \wedge 2)))$

unfolding *numeral-2-eq-2*

by (*rule DERIV-inverse-fun*)

lemma *NSCDERIV-inverse-fun:*

$[[\text{NSDERIV } f x := d; f(x) \neq (0::\text{complex})]]$

$\implies \text{NSDERIV } (\%x. \text{inverse}(f x)) x := (- (d * \text{inverse}(f(x) \wedge 2)))$

unfolding *numeral-2-eq-2*

by (*rule NSDERIV-inverse-fun*)

19.6 Derivative of Quotient

lemma *CDERIV-quotient:*

$[[\text{DERIV } f x := d; \text{DERIV } g x := e; g(x) \neq (0::\text{complex})]]$

$\implies \text{DERIV } (\%y. f(y) / (g y)) x := (d * g(x) - (e * f(x))) / (g(x) \wedge 2)$

unfolding *numeral-2-eq-2*

by (*rule DERIV-quotient*)

lemma *NSCDERIV-quotient:*

$[[\text{NSDERIV } f x := d; \text{NSDERIV } g x := e; g(x) \neq (0::\text{complex})]]$

$\implies \text{NSDERIV } (\%y. f(y) / (g y)) x := (d * g(x) - (e * f(x))) / (g(x) \wedge 2)$

unfolding *numeral-2-eq-2*

by (*rule NSDERIV-quotient*)

19.7 Caratheodory Formulation of Derivative at a Point: Standard Proof

lemma *CARAT-CDERIVD:*

$(\forall z. f z - f x = g z * (z - x)) \ \& \ \text{isNSCont } g x \ \& \ g x = l$

$\implies \text{NSDERIV } f x := l$

by *clarify* (*rule CARAT-DERIVD*)

end

20 HLog: Logarithms: Non-Standard Version

theory *HLog*

imports *Log HTranscendental*

begin

lemma *epsilon-ge-zero* [*simp*]: $0 \leq \text{epsilon}$
by (*simp add: epsilon-def star-n-zero-num star-n-le*)

lemma *hpfinitive-witness*: $\text{epsilon} : \{x. 0 \leq x \ \& \ x : \text{HFinite}\}$
by *auto*

definition

powhr :: [*hypreal*,*hypreal*] => *hypreal* (**infixr** *powhr* 80) **where**
 [*transfer-unfold*]: $x \text{ powhr } a = \text{starfun2 } (\text{op } \text{powhr}) \ x \ a$

definition

hlog :: [*hypreal*,*hypreal*] => *hypreal* **where**
 [*transfer-unfold*]: $\text{hlog } a \ x = \text{starfun2 } \text{log } a \ x$

lemma *powhr*: $(\text{star-n } X) \ \text{powhr} \ (\text{star-n } Y) = \text{star-n } (\%n. (X \ n) \ \text{powhr} \ (Y \ n))$
by (*simp add: powhr-def starfun2-star-n*)

lemma *powhr-one-eq-one* [*simp*]: $!!a. 1 \ \text{powhr} \ a = 1$
by (*transfer, simp*)

lemma *powhr-mult*:

$!!a \ x \ y. [| 0 < x; 0 < y |] ==> (x * y) \ \text{powhr} \ a = (x \ \text{powhr} \ a) * (y \ \text{powhr} \ a)$
by (*transfer, rule powhr-mult*)

lemma *powhr-gt-zero* [*simp*]: $!!a \ x. 0 < x \ \text{powhr} \ a$
by (*transfer, simp*)

lemma *powhr-not-zero* [*simp*]: $x \ \text{powhr} \ a \neq 0$
by (*rule powhr-gt-zero [THEN hypreal-not-refl2, THEN not-sym]*)

lemma *powhr-divide*:

$!!a \ x \ y. [| 0 < x; 0 < y |] ==> (x / y) \ \text{powhr} \ a = (x \ \text{powhr} \ a) / (y \ \text{powhr} \ a)$
by (*transfer, rule powhr-divide*)

lemma *powhr-add*: $!!a \ b \ x. x \ \text{powhr} \ (a + b) = (x \ \text{powhr} \ a) * (x \ \text{powhr} \ b)$
by (*transfer, rule powhr-add*)

lemma *powhr-powhr*: $!!a \ b \ x. (x \ \text{powhr} \ a) \ \text{powhr} \ b = x \ \text{powhr} \ (a * b)$
by (*transfer, rule powhr-powhr*)

lemma *powhr-powhr-swap*: $!!a \ b \ x. (x \ \text{powhr} \ a) \ \text{powhr} \ b = (x \ \text{powhr} \ b) \ \text{powhr} \ a$
by (*transfer, rule powhr-powhr-swap*)

lemma *powhr-minus*: $!!a \ x. x \ \text{powhr} \ (-a) = \text{inverse } (x \ \text{powhr} \ a)$

by (*transfer*, *rule powr-minus*)

lemma *powhr-minus-divide*: $x \text{ powhr } (-a) = 1 / (x \text{ powhr } a)$

by (*simp add: divide-inverse powhr-minus*)

lemma *powhr-less-mono*: $!!a \ b \ x. [a < b; 1 < x] \implies x \text{ powhr } a < x \text{ powhr } b$

by (*transfer*, *simp*)

lemma *powhr-less-cancel*: $!!a \ b \ x. [x \text{ powhr } a < x \text{ powhr } b; 1 < x] \implies a < b$

by (*transfer*, *simp*)

lemma *powhr-less-cancel-iff* [*simp*]:

$$1 < x \implies (x \text{ powhr } a < x \text{ powhr } b) = (a < b)$$

by (*blast intro: powhr-less-cancel powhr-less-mono*)

lemma *powhr-le-cancel-iff* [*simp*]:

$$1 < x \implies (x \text{ powhr } a \leq x \text{ powhr } b) = (a \leq b)$$

by (*simp add: linorder-not-less [symmetric]*)

lemma *hlog*:

$$\begin{aligned} \text{hlog } (\text{star-n } X) (\text{star-n } Y) &= \\ \text{star-n } (\%n. \text{log } (X \ n) (Y \ n)) & \end{aligned}$$

by (*simp add: hlog-def starfun2-star-n*)

lemma *hlog-starfun-ln*: $!!x. (** \ln) x = \text{hlog } ((** \exp) 1) x$

by (*transfer*, *rule log-ln*)

lemma *powhr-hlog-cancel* [*simp*]:

$$!!a \ x. [0 < a; a \neq 1; 0 < x] \implies a \text{ powhr } (\text{hlog } a \ x) = x$$

by (*transfer*, *simp*)

lemma *hlog-powhr-cancel* [*simp*]:

$$!!a \ y. [0 < a; a \neq 1] \implies \text{hlog } a (a \text{ powhr } y) = y$$

by (*transfer*, *simp*)

lemma *hlog-mult*:

$$\begin{aligned} !!a \ x \ y. [0 < a; a \neq 1; 0 < x; 0 < y] & \\ \implies \text{hlog } a (x * y) = \text{hlog } a \ x + \text{hlog } a \ y & \end{aligned}$$

by (*transfer*, *rule log-mult*)

lemma *hlog-as-starfun*:

$$!!a \ x. [0 < a; a \neq 1] \implies \text{hlog } a \ x = (** \ln) x / (** \ln) a$$

by (*transfer*, *simp add: log-def*)

lemma *hlog-eq-div-starfun-ln-mult-hlog*:

$$\begin{aligned} !!a \ b \ x. [0 < a; a \neq 1; 0 < b; b \neq 1; 0 < x] & \\ \implies \text{hlog } a \ x = ((** \ln) b / (** \ln) a) * \text{hlog } b \ x & \end{aligned}$$

by (*transfer*, *rule log-eq-div-ln-mult-log*)

lemma *powhr-as-starfun*: !!*a x. x powhr a = (*f* exp) (a * (*f* ln) x)*
by (*transfer, simp add: powr-def*)

lemma *HInfinite-powhr*:

$[[x : HInfinite; 0 < x; a : HFinite - Infinitesimal; 0 < a]] ==> x \text{ powhr } a : HInfinite$

apply (*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)

done

lemma *hlog-hrabs-HInfinite-Infinitesimal*:

$[[x : HFinite - Infinitesimal; a : HInfinite; 0 < a]] ==> hlog a (abs x) : Infinitesimal$

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 hypreal-not-refl2 [THEN not-sym] divide-inverse)

done

lemma *hlog-HInfinite-as-starfun*:

$[[a : HInfinite; 0 < a]] ==> hlog a x = (*f* ln) x / (*f* ln) a$

by (*rule hlog-as-starfun, auto*)

lemma *hlog-one* [*simp*]: !!*a. hlog a 1 = 0*

by (*transfer, simp*)

lemma *hlog-eq-one* [*simp*]: !!*a. [[0 < a; a ≠ 1]] ==> hlog a a = 1*

by (*transfer, rule log-eq-one*)

lemma *hlog-inverse*:

$[[0 < a; a ≠ 1; 0 < x]] ==> hlog a (inverse x) = - hlog a x$

apply (*rule add-left-cancel [of hlog a x, THEN iffD1]*)

apply (*simp add: hlog-mult [symmetric]*)

done

lemma *hlog-divide*:

$[[0 < a; a ≠ 1; 0 < x; 0 < y]] ==> hlog a (x/y) = hlog a x - hlog a y$

by (*simp add: hlog-mult hlog-inverse divide-inverse*)

lemma *hlog-less-cancel-iff* [*simp*]:

!!*a x y. [[1 < a; 0 < x; 0 < y]] ==> (hlog a x < hlog a y) = (x < y)*

by (*transfer, simp*)

lemma *hlog-le-cancel-iff* [*simp*]:

$[[1 < a; 0 < x; 0 < y]] ==> (hlog a x ≤ hlog a y) = (x ≤ y)$

by (*simp add: linorder-not-less [symmetric]*)

end

theory *Hyperreal*
imports *Ln Deriv Taylor HLog*
begin

end

theory *Hypercomplex*
imports *CLim Hyperreal*
begin

end