Euler's Exponential Series as an Infinite Polynomial

Jacques D. Fleuriot
University of Edinburgh

June 25, 2026

Abstract

In this formalisation, we reconstruct Euler's derivation of the power series for the exponential function as expounded in his famous *Introductio in analysin infinitorum*, first published in 1748. Using nonstandard analysis, we mechanize his mixture of infinitesimal and infinite 'algebraic' reasoning in the proof assistant Isabelle. In so doing, we demonstrate that the gist of his arguments can be reconstructed formally, with Isabelle and nonstandard analysis shoring up crucial aspects of his reasoning that some historians have qualified as being "more a matter of faith than science".

Contents

1 Generalized falling factorial

theory *FallFactorial*
imports *"HOL.Binomial" "HOL.Rat"*
begin

definition
"fallfactpow (a::'a::comm_ring_1) n = (if n = 0 then 1 else prod (λn. a - of_nat n) {0 .. n - 1})"

lemma *fallfactpow_0[simp]: "fallfactpow a 0 = 1"*
⟨*proof*⟩

lemma *fallfactpow_1[simp]: "fallfactpow a 1 = a"* ⟨*proof*⟩
lemma *fallfactpow_Suc0[simp]: "fallfactpow a (Suc 0) = a"* ⟨*proof*⟩

lemma *fallfactpow_Suc_setprod: "fallfactpow a (Suc n) = prod (λn. a - of_nat n) {0 .. n}"*
⟨*proof*⟩

1.1 Falling factorial expressed recursively

lemma *fallfactpow_Suc: "fallfactpow a (Suc n) = fallfactpow a n * (a - of_nat n)"*
⟨*proof*⟩

lemma *fallfactpow_rec: "fallfactpow a (Suc n) = a * fallfactpow (a - 1) n"*
⟨*proof*⟩

lemma *fallfactpow_base_zero[simp]: "fallfactpow 0 (Suc n) = 0"*
⟨*proof*⟩

lemma *fallfactpow_fact: "of_nat(fact n) = fallfactpow (of_nat n) n"*
⟨*proof*⟩

lemma *fallfactpow_altdef:*
"fallfactpow a k = (∏ i<k. a - of_nat i)"
⟨*proof*⟩

The falling factorial and Pochammer functions

lemma *pochhammer_fallfactpow_eq:*
"pochhammer a n = fallfactpow (a + of_nat n - 1) n"
⟨*proof*⟩

lemma *fallfactpow_of_nat_eq_0_lemma [simp]:*
assumes *kn: "k > n"*
shows *"fallfactpow (of_nat n :: 'a:: idom) k = 0"*
⟨*proof*⟩

lemma *setprod_zero_of_nat_diff_lemma:*
```
"a < k ⟹ (∏i<k. of_nat (a - i)) = (0::'a::comm_semiring_1)"
```
⟨*proof*⟩

lemma *fallfactpow_of_nat_altdef:*
```
"fallfactpow (of_nat a :: 'a :: idom) k = ((∏i<k. of_nat (a - i)) ::
'a :: idom)"
```
⟨*proof*⟩

lemma *fallfactpow_ge_zero:*
```
"of_nat k < j ⟹ 0 ≤ fallfactpow (j::'a::{linordered_idom}) k"
```
⟨*proof*⟩

lemma *fallfactpow_le_power_self [simp]:*
```
"fallfactpow (of_nat k ::'a::{linordered_idom}) k ≤ of_nat k ^ k"
```
⟨*proof*⟩

lemma *fallfactpow_le_power:*
assumes *"of_nat k ≤ j"*
shows *"fallfactpow (j::'a::{linordered_idom}) k ≤ j ^ k"*
⟨*proof*⟩

lemma *fallfactpow_ge_zero' [simp]:*
```
"fallfactpow (of_nat n) k ≥ (0::'a::linordered_idom)"
```
⟨*proof*⟩

1.2 Falling factorial and binomial coefficients

lemma *binomial_fallfactpow_altdef_of_nat:*
```
"of_nat (n choose k) =
        (fallfactpow (of_nat n) k :: 'a :: {field_char_0})/fact k"
```
⟨*proof*⟩

lemma *binomial_fallfactpow_ring:*
```
"fallfactpow (a + b :: 'a::{comm_ring_1}) n =
(∑k=0..n. of_nat(n choose k) * fallfactpow a (n - k) * fallfactpow
b k)"
```
⟨*proof*⟩

lemma *binomial_vandermonde:*
```
"(r + s)^ k/fact k =
 (∑i = 0..k. (r::'a :: field_char_0) ^ i / fact i *  s ^ (k - i) /
fact (k - i))"
```
⟨*proof*⟩

lemma *natsum_reverse_index:*

fixes *m::nat*
shows *"(⋀k. m ≤ k ⟹ k ≤ n ⟹ g k = f (m + n - k)) ⟹ (∑k=m..n. f k) = (∑k=m..n. g k)"*
⟨*proof*⟩

lemma *binomial_fallfactpow_vandermonde:*
"fallfactpow (r + s) k/fact k =
*(∑i = 0..k. fallfactpow (r::'a :: field_char_0) i / fact i * fallfactpow s (k - i) / fact (k - i))"*
⟨*proof*⟩

lemma *gbinomial_fallfactpow: "a gchoose n = fallfactpow a n / fact n"*
⟨*proof*⟩

lemma *binomial_fallfactpow_vandermonde_alt:*
"fallfactpow (r + s) k/fact k =
*(∑i = 0..k. ((r::'a :: field_char_0) gchoose i) * (s gchoose (k - i)))"*
⟨*proof*⟩

end

2 Various supporting lemmas

theory *AuxiliaryNSA*
imports *"HOL-Nonstandard_Analysis.HSeries" "Real_Power.Log"*
begin

These results can be moved to the indicated theories.

StarDef.thy lemmas

lemma *Iset_star_of_empty [simp]: "Iset(star_of {}) = {}"*
⟨*proof*⟩

lemma *Iset_star_n_empty [simp]: "Iset (star_n (λn. {})) = {}"*
⟨*proof*⟩

lemma *Iset_eq_cancel:*
"(Iset (star_n X) = Iset (star_n Y)) = (eventually (λn. X n = Y n) 𝒰)"
⟨*proof*⟩

lemma *Collect_starP_starset_eq: "{x. (*p* P) x} = *s* {x. P x}"*
⟨*proof*⟩

Hypernat.thy lemmas

lemma *Nats_hypnat_of_nat_iff: "(n ∈ Nats) = (∃m. n = hypnat_of_nat m)"*
⟨*proof*⟩

lemma *HNatInfinite_add_one_cancel:*

```
  "N + 1 ∈ HNatInfinite ⟹ N ∈ HNatInfinite"
```
⟨*proof*⟩

lemma *of_hypnat_of_nat [simp]: "of_hypnat(hypnat_of_nat n) = of_nat n"*
⟨*proof*⟩

lemma *of_nat_hypreal_of_hypnat:*
```
    "of_nat n = of_hypnat (hypnat_of_nat n)"
```
⟨*proof*⟩

lemma *hSuc_eq_add_one:*
```
  "⋀n. hSuc n = n + 1"
```
⟨*proof*⟩

lemma *HNatInfinite_hSuc_diff_one [simp]:*
```
  "N ∈ HNatInfinite ⟹ hSuc (N - 1) = N"
```
⟨*proof*⟩

lemma *HNatInfinite_atLeastAtMost_hSuc:*
```
  "N ∈ HNatInfinite ⟹ {M..<N} = {M..<hSuc(N - 1)}"
```
⟨*proof*⟩

HyperDef.thy lemmas

lemma *hypreal_of_nat_less_hypreal_of_hypnat_iff:*
```
     "⋀a b.(hypreal_of_nat a < of_hypnat b) = (hypnat_of_nat a < b)"
```
⟨*proof*⟩

lemma *hypreal_of_nat_le_hypreal_of_hypnat_iff:*
```
     "⋀a b.(hypreal_of_nat a ≤ hypreal_of_hypnat b) = (hypnat_of_nat
a ≤  b)"
```
⟨*proof*⟩

lemma *hyperpow_zero [simp]: "x pow 0 = 1"*
⟨*proof*⟩

lemma *hyperpow_of_nat: "⋀x. x pow of_nat n = x ^ n"*
⟨*proof*⟩

lemma *hyperpow_hSuc_zero [simp]:*
```
  "⋀N. (0::'a::{power, semiring_0} star) pow hSuc N = 0"
```
⟨*proof*⟩

lemma *starfun_power: "⋀n. (*f* (^) x) n = star_of x pow n"*
⟨*proof*⟩

lemma *hyperpow_diff:*
```
  "⋀x n k. n ≥ k ⟹ x ≠ 0 ⟹ ((x::'a::field star) pow n)/(x pow k)
= x pow (n - k)"
```
⟨*proof*⟩

lemma *hyperpow_divide:*
"⋀x y n. ((x::'a::field star)/y) pow n = (x pow n)/(y pow n)"
⟨*proof*⟩

lemma *hyperpow_diff_cancel:*
"k ≤ n ⟹ x pow k * (x::'a::monoid_mult star) pow (n - k) = x pow n"
⟨*proof*⟩

lemma *hyperpow_0_left:*
"⋀n. 0 pow n = (if n = 0 then 1 else (0::'a::{power, semiring_1} star))"
⟨*proof*⟩

lemma *hyperpow_eq_0_iff [simp]:*
"⋀x n. (x pow n = 0) ⟷
(x = (0::'a::{mult_zero,zero_neq_one,semiring_1_no_zero_divisors,power} star) ∧ n ≠ 0)"
⟨*proof*⟩

lemma *hyperpow_less_zero_eq:*
"⋀x n. (x pow n < (0::'a::{linordered_idom} star)) = (¬ 2 dvd n ∧ x < 0)"
⟨*proof*⟩

lemma *star_n_divide:* "star_n X / star_n Y = star_n (λn. X n / Y n)"
⟨*proof*⟩

NSA.thy lemmas

lemma *approx_zero_abs_zero_iff [iff]:*
"(abs x ≈ 0) = ((x::hypreal) ≈ 0)"
⟨*proof*⟩

lemma *Infinitesimal_divide_HFinite:*
"⟦ (x::'a::real_normed_div_algebra star) ∈ Infinitesimal; y ∈ HFinite - Infinitesimal ⟧
⟹ x/y ∈ Infinitesimal"
⟨*proof*⟩

lemma *approx_1_not_Infinitesimal:*
"x ≈ (1 ::'a::real_normed_div_algebra star) ⟹ x ∉ Infinitesimal"
⟨*proof*⟩

lemma *HFinite_add_imp_HFinite:*
"⟦ x ≥ (0::hypreal); y ≥ 0; x + y ∈ HFinite ⟧ ⟹ y ∈ HFinite"
⟨*proof*⟩

lemma *Infinitesimal_hrealpow_cancel [simp]:*
assumes "n > 0"

shows *"(x ^ n ∈ Infinitesimal) = ((x::'a::real_normed_div_algebra star) ∈ Infinitesimal)"*
⟨*proof*⟩

lemma *hrealpow_approx:*
"⟦ (x::'a::{real_normed_algebra, one, monoid_mult} star) ≈ b; b ∈ HFinite ⟧ ⟹ x ^ n ≈ b ^ n"
⟨*proof*⟩

lemma *hrealpow_approx_one:*
"(x::'a::{real_normed_algebra, one, monoid_mult} star) ≈ 1 ⟹ x ^ n ≈ 1"
⟨*proof*⟩

lemma *zero_not_HInfinite [simp]: "0 ∉ HInfinite"*
⟨*proof*⟩

lemma *HInfinite_hyperpow_not_zero:*
"(x::'a::{real_normed_field} star) ∈ HInfinite ⟹ x pow n ≠ 0"
⟨*proof*⟩

lemma *HFinite_divide:*
"⟦ (x::'a::real_normed_div_algebra star) ∈ HFinite; y ∉ Infinitesimal ⟧ ⟹ x/y ∈ HFinite"
⟨*proof*⟩

lemma *HFinite_hyperpow_HFinite:*
"⟦ (x::'a::{monoid_mult, real_normed_algebra} star) ∈ HFinite; n ∈ ℕ ⟧ ⟹ x pow n ∈ HFinite"
⟨*proof*⟩

lemma *hyperpow_approx:*
assumes *HFinite_a: "a ∈ HFinite"*
and *approx_ab: "(a::hypreal) ≈ b"*
shows *"a pow of_nat n ≈ b pow of_nat n"*
⟨*proof*⟩

lemma *Infinitesimal_approx_hyperpow:*
assumes *Infinitesimal_a: "a ∈ Infinitesimal"*
and *approx_ab: "(a::hypreal) ≈ b"*
shows *"a pow n ≈ b pow n"*
⟨*proof*⟩

lemma *hnorm_hypreal_of_hypnat [simp]:*
"⋀n. hnorm (of_hypnat n::'a::real_normed_algebra_1 star) = of_hypnat n"
⟨*proof*⟩

lemma *HFinite_of_nat [simp]: "of_nat n ∈ HFinite"*

⟨*proof*⟩

lemma *HInfinite_def2: "HInfinite = {x. ∀n ∈ Nats. hypreal_of_hypnat n < hnorm x}"*
⟨*proof*⟩

lemma *HFinite_def2: "HFinite = {x. ∃n ∈ Nats. hnorm x < hypreal_of_hypnat n}"*
⟨*proof*⟩

lemma *HFinite_less_Nat_Ex:*
"(x::hypreal) ∈ HFinite ⟹ ∃n∈Nats. x < n"
⟨*proof*⟩

lemma *HNatInfinite_of_hypnat_HInfinite:*
"x ∈ HNatInfinite ⟹ (of_hypnat x :: 'a::real_normed_algebra_1 star) ∈ HInfinite"
⟨*proof*⟩

lemma *HInfinite_ge_HNatInfinite:*
"⟦ n ∈ HNatInfinite; hypreal_of_hypnat n ≤ x ⟧ ⟹ x ∈ HInfinite"
⟨*proof*⟩

lemma *HNatInfinite_of_hypnat_HInfinite_iff:*
"(of_hypnat n :: 'a::real_normed_algebra_1 star) ∈ HInfinite = (n ∈ HNatInfinite)"
⟨*proof*⟩

lemma *HNatInfinite_inverse_of_hypnat_Infinitesimal_iff:*
"N ≠ 0 ⟹ (1/hypreal_of_hypnat N ∈ Infinitesimal) = (N ∈ HNatInfinite)"
⟨*proof*⟩

lemma *Nats_hypreal_of_hypnat_HFinite:*
"n ∈ Nats ⟹ of_hypnat n ∈ HFinite"
⟨*proof*⟩

lemma *approx_divide_HFinite_diff_Infinitesimal:*
"⟦(x :: 'a::{real_normed_div_algebra, field} star) ≈ y;
z ∈ HFinite - Infinitesimal
⟧ ⟹ x/z ≈ y/z"
⟨*proof*⟩

lemma *HFinite_diff:*
"⟦ x ∈ HFinite; y ∈ HFinite ⟧ ⟹ x - y ∈ HFinite"
⟨*proof*⟩

lemma *Infinitesimal_interval2a:*
"⟦ e ∈ Infinitesimal; e' ∈ Infinitesimal;
hnorm e' ≤ hnorm x ; hnorm x ≤ hnorm e ⟧ ⟹ x ∈ Infinitesimal"

⟨*proof*⟩

lemma *Infinitesimal_HInfinite_divide:*
"⟦ *(x::'a::{real_normed_div_algebra, field} star) ∈ HFinite; y ∈ HInfinite* ⟧ ⟹ *x/y ∈ Infinitesimal*"
⟨*proof*⟩

lemma *approx_divide_HInfinite:*
"⟦*(x :: 'a::{real_normed_div_algebra, field} star) ≈ y; z ∈ HInfinite*⟧ ⟹ *x/z ≈ y/z*"
⟨*proof*⟩

lemma *approx_1_mult_HFinite_HFinite:*
assumes *approx_1: "x ≈ (1 ::'a::real_normed_div_algebra star)"*

and *HFinite_prod: "x*y ∈ HFinite"*
shows *"y ∈ HFinite"*
⟨*proof*⟩

Star.thy lemmas

instance *star :: (semiring_modulo) semiring_modulo*
⟨*proof*⟩

declare *of_bool_def [transfer_unfold]*

instance *star :: (semiring_parity) semiring_parity*
⟨*proof*⟩

lemma *starfun_starfun_n_eq:*
"*f* f = *fn* (λn. f)"
⟨*proof*⟩

lemma *starfun_n_zero:*
"(*fn* (λm n::'b. 0::'a::zero)) = (λn. star_of 0)"
⟨*proof*⟩

lemma *InternalFuns_starfun_n [simp]: "*fn* f ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_starfun [simp]: "*f* f ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_starfun2 [simp]: "(*f2* f) a ∈ InternalFuns"*
⟨*proof*⟩

lemma *starfun_n_hyperpow: "(λn. (*f2* (^)) ((*fn* f) n) ((*fn* g) n)) z = (*fn* (λ i x. (f i x) ^ (g i x))) z"*
⟨*proof*⟩

lemma *InternalFuns_hSuc [simp]: "hSuc ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalSets_starset_n [simp]: "*sn* X ∈ InternalSets"*
⟨*proof*⟩

lemma *InternalSets_singleton [simp]: "{x} ∈ InternalSets"*
⟨*proof*⟩

lemma *starset_n_singleton [simp]: "*sn* (λn. {X n}) = {star_n X}"*
⟨*proof*⟩

lemma *starset_n_singleton2:"*sn* (λn. {X n}) = {x. x = star_n X}"*
⟨*proof*⟩

lemma *starset_n_empty [simp]: "*sn* (λn. {}) = {}"*
⟨*proof*⟩

lemma *InternalSets_empty [simp]: "{} ∈ InternalSets"*
⟨*proof*⟩

lemma *starset_n_subset:*
*"(*sn* X ⊆ *sn* Y) = (eventually (λn. X n ⊆ Y n) 𝒰)"*
⟨*proof*⟩

lemma *starfun_n_mult_star_n_right:*
*"(*fn* F) n * star_n X = (*fn* (λn m. F n m * X n)) n"*
⟨*proof*⟩

lemma *starfun_n_mult_star_n_left:*
*"star_n X * (*fn* F) n = (*fn* (λn m. X n * F n m)) n"*
⟨*proof*⟩

lemma *starfun_n_diff:*
*"(*fn* f) z - (*fn* g) z = (*fn* (λi x. f i x - g i x)) z"*
⟨*proof*⟩

lemma *starfun_n_inverse: "inverse ((*fn* f) x) = (*fn* (λi x. inverse ((f i) x))) x"*
⟨*proof*⟩

lemma *InternalFuns_const_fun [simp]:*
"(λn. c) ∈ InternalFuns"
⟨*proof*⟩

lemma *starfun_n_divide:*
*"((*fn* f) z ::'a::division_ring star)/ (*fn* g) z = (*fn* (λi x. f i x / g i x)) z"*
⟨*proof*⟩

lemma *InternalFuns_divide:*
"⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧
⟹ (λn. (f n ::'a::division_ring star)/g n) ∈ InternalFuns"
⟨*proof*⟩

lemma *InternalFuns_id_fun [simp]: "(λn. n) ∈ InternalFuns"*
⟨*proof*⟩

lemma *FreeUltrafilterNat_star_n_Infinitesimal:*
assumes *"eventually (λn. norm (X n) < inverse(Suc n)) 𝒰"*
shows *"star_n X ∈ Infinitesimal"*
⟨*proof*⟩

lemma *starset_n_atLeastLessThan_eq:*
*"*sn* (λn. {(X n)..<(Y n)}) = {star_n X..<star_n Y}"*
⟨*proof*⟩

lemma *starset_n_atLeast_zero_LessThan_eq:*
*"*sn* (λn. {0..<(X n)}) = {0..<star_n X}"*
⟨*proof*⟩

lemma *starset_atLeastAtMost:*
*"{of_nat m .. of_nat n} = *s* {m .. n}"*
⟨*proof*⟩

lemma *starset_n_atLeastAtMost:*
*"*sn* (λn. {(X n)..(Y n)}) = {star_n X..star_n Y}"*
⟨*proof*⟩

lemma *starset_n_atLeast_zero_AtMost:*
*"*sn* (λn. {0..(X n)}) = {0..star_n X}"*
⟨*proof*⟩

lemma *InternalSets_atLeastLessThan [simp]: "{M..<N} ∈ InternalSets"*
⟨*proof*⟩

lemma *InternalSets_atLeastAtMost [simp]: "{M..N} ∈ InternalSets"*
⟨*proof*⟩

lemma *le_inverse_HNatInfinite_Infinitesimal:*
"⟦n ∈ HNatInfinite; hnorm x ≤ inverse (hypreal_of_hypnat n)⟧ ⟹
x ∈ Infinitesimal"
⟨*proof*⟩

lemma *InternalFuns_hyperpow:*

```
    "f ∈ InternalFuns ⟹ (g:: 'a star ⇒ nat star) ∈ InternalFuns
     ⟹ (λn. f n pow g n) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_hyperpowf [simp]:*
```
  "f ∈ InternalFuns ⟹ (λn. x pow (f n)) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_hyperpow_simple [simp]: "(λn. x pow n) ∈ InternalFuns"*
⟨*proof*⟩

NatStar.thy lemmas

lemma *InternalFuns_add:*
```
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧ ⟹ (λn. f n + g n) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_mult:*
```
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧ ⟹ (λn. f n * g n) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_diff:*
```
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧ ⟹ (λn. f n - g n) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_minus:*
```
  "f ∈ InternalFuns  ⟹ (λn. - f n ) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_mult_const_left:*
```
  "f ∈ InternalFuns ⟹ (λn. c * f n) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_mult_const_right:*
```
  "f ∈ InternalFuns ⟹ (λn. f n * c) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_mult_of_hypnat [simp]:*
```
      "X ∈ InternalFuns ⟹ (λn. of_hypnat n * X n) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *finite_InternalSets:*
```
  assumes "finite X"
  shows "X ∈ InternalSets"
```
⟨*proof*⟩

lemma *InternalFuns_of_hypnat:*
```
  assumes "f ∈ InternalFuns"
  shows "(λn. of_hypnat (f n)) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *insert_star_n_starset_n:*
*"insert (star_n X) (*sn* A) = *sn* (λn. insert (X n) (A n))"*
⟨*proof*⟩

lemma *inj_starfun_n:*
*"inj (*fn* f) = (∀$_F$ n in 𝒰. inj (f n))"*
⟨*proof*⟩

lemma *surj_starfun_n:*
*"surj (*fn* f) = (∀$_F$ n in 𝒰. surj (f n))"*
⟨*proof*⟩

lemma *bij_starfun_n:*
*"bij (*fn* f) = (∀$_F$ n in 𝒰. bij (f n))"*
⟨*proof*⟩

lemma
assumes *"bij (*fn* f)"*
shows *"(inv (*fn* f)) = *fn* (λn. inv (f n))"*
⟨*proof*⟩

lemma *starfun_n_o:*
*"(λi. (*fn* F) ((*fn* G) i)) = (*fn* (λi m. F i (G i m)))"*
⟨*proof*⟩

lemma *starfun_starfun_n_o:*
*"(λi. (*f* f) ((*fn* F) i)) = (*fn* (λi m. f (F i m)))"*
⟨*proof*⟩

lemma *InternalFuns_o2:*
"f ∈ InternalFuns ⟹ g ∈ InternalFuns ⟹ (λn. f (g n)) ∈ InternalFuns"
⟨*proof*⟩

lemma *InternalFuns_abs_starfun_n [simp]:*
*"(λi. abs ((*fn* F) i)) ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_abs:*
"f ∈ InternalFuns ⟹ (λi. abs (f i)) ∈ InternalFuns"
⟨*proof*⟩

HSEQ.thy lemmas

lemma *LIMSEQ_0_Infinitesimal:*
assumes *"X ⟶ 0"*
shows *"star_n X ∈ Infinitesimal"*
⟨*proof*⟩

Alternative definitions for exponential and real powers to replace the one in

the AFP session eventually.

lemma *powrat_def2: "x pow$_{\mathbb{Q}}$ r = root (nat (snd(quotient_of r))) (x powi (fst(quotient_of r)))"*
⟨*proof*⟩

lemma *powrat_powi: "x pow$_{\mathbb{Q}}$ (of_int n) = x powi n"*
⟨*proof*⟩

lemma *real_root_eq_powrat_inverse': "n > 0 ⟹ x pow$_{\mathbb{Q}}$ (1 / of_nat n) = root n x"*
⟨*proof*⟩

lemma *"a powa x = lim (λn. a pow$_{\mathbb{Q}}$ (incseq_of x n))"*
⟨*proof*⟩

Next two proofs by Zuzana Frankovska

lemma *HNatInfinite_pow_real_Infinitesimal:*
assumes *"0 ≤ (x::real)"* **and** *"x < 1"* **and** *"N ∈ HNatInfinite"*
shows *"(star_of x) pow N ≈ 0"*
⟨*proof*⟩

lemma *HNatInfinite_pow_Infinitesimal:*
assumes *"0 < (x::hypreal)"* **and** *"x < 1"* **and** *"¬(x ≈ 1)"* **and** *"N ∈ HNatInfinite"*
shows *"x pow N ≈ 0"*
⟨*proof*⟩

lemma *Infinitesimal_hyperpow_HNatInfinite:*
assumes *"hnorm (x::'a::real_normed_div_algebra star) < 1"*
and *"N ∈ HNatInfinite"*
and *"¬(hnorm x ≈ 1)"*
shows *"x pow N ∈ Infinitesimal"*
⟨*proof*⟩

HSeries.thy lemmas

lemma *NSsums_sumhr_HNatInfinite_approx_iff:*
"(f NSsums s) = (∀N∈HNatInfinite. sumhr(0, N, f) ≈ of_real s)"
⟨*proof*⟩

lemma *sumhr_approx:*
"sumhr (0, M, f) ≈ sumhr (0, N, f) ⟹ sumhr (M, N, f) ≈ 0"
⟨*proof*⟩

end

3 Hyperinteger numbers

theory *HyperInt*
imports *AuxiliaryNSA "HOL-Nonstandard_Analysis.NSA"*

begin

type_synonym *hypint = "int star"*

abbreviation
hypint_of_int :: "int ⇒ int star" **where**
"hypint_of_int ≡ star_of"

abbreviation
hypint_of_hypnat :: "hypnat ⇒ hypint" **where**
"hypint_of_hypnat ≡ of_hypnat"

3.1 Properties of Hyperintegers

Many properties hold by transfer from the integers

lemma *hypint_nat_mult_less_mono2_lemma:*
*"(i::hypint)<j ⟹ 0<k ⟹ of_nat k * i < of_nat k * j"*
⟨*proof*⟩

lemma *hypint_mult_less_mono2_lemma:*
*"⋀i j k. (i::hypint)<j ⟹ 0<k ⟹ of_hypnat k * i < of_hypnat k*
** j"*
⟨*proof*⟩

lemma *zero_le_imp_eq_hypint: "⋀k. (0::hypint) ≤ k ⟹ ∃n. k = of_hypnat*
n"
⟨*proof*⟩

lemma *zero_less_imp_eq_hypint: "⋀k. (0::hypint) < k ⟹ ∃n>0. k = of_hypnat*
n"
⟨*proof*⟩

lemma *hypint_less_mono2: "⋀i j k. ⟦ i<j; (0::hypint) < k ⟧ ⟹ k*i <*
*k*j"*
⟨*proof*⟩

lemma *hypint_less_imp_add1_le: "⋀w z. w < z ⟹ w + (1::hypint) ≤ z"*
⟨*proof*⟩

lemma *hypint_less_iff_hSuc_add:*
"⋀w z. (w :: hypint) < z ⟷ (∃n. z = w + of_hypnat (hSuc n))"
⟨*proof*⟩

lemma *hypint_le_iff_add:*
"⋀w z. (w :: hypint) ≤ z ⟷ (∃n. z = w + of_hypnat n)"
⟨*proof*⟩

lemma *hypint_le_iff_diff:*
"⋀w z. (w :: hypint) ≤ z ⟷ (∃n. w = z - of_hypnat n)"

⟨*proof*⟩

lemma *hypint_less_zero_eq_minus_of_hypnat: "x < (0::hypint) ⟹ ∃n. x = - hypint_of_hypnat (n)"*
⟨*proof*⟩

lemma *hypint_less_add1_eq: "⋀w z. (w < z + (1::hypint)) = (w<z | w=z)"*
⟨*proof*⟩

lemma *add1_hypint_le_eq: "⋀w z. (w + (1::hypint)≤ z) = (w<z)"*
⟨*proof*⟩

lemma *hypint_le_diff1_eq [simp]: "⋀w z. (w ≤ z - (1::hypint)) = (w<z)"*
⟨*proof*⟩

lemma *hypint_le_add1_eq_le [simp]: "⋀w z. (w < z + (1::hypint)) = (w ≤ z)"*
⟨*proof*⟩

lemma *hypint_one_le_iff_zero_less: "⋀z. ((1::hypint) ≤ z) = (0 < z)"*
⟨*proof*⟩

3.2 Embedding of the hyperintegers into other types

definition
of_hypint :: "hypint => 'a::linordered_idom star" **where**
*of_hypint_def [transfer_unfold]: "of_hypint = *f* of_int"*

lemma *of_hypint_of_int [simp]: "of_hypint (of_int z) = of_int z"*
⟨*proof*⟩

lemma *of_hypint_hypint_of_int [simp]: "of_hypint (hypint_of_int z) = of_int z"*
⟨*proof*⟩

lemma *of_hypint_of_hypnat [simp]: "⋀z. of_hypint (of_hypnat z) = of_hypnat z"*
⟨*proof*⟩

lemma *of_hypint_of_nat [simp]: "of_hypint (of_nat z) = of_nat z"*
⟨*proof*⟩

lemma *of_hypint_0 [simp]: "of_hypint 0 = 0"*
⟨*proof*⟩

lemma *of_hypint_1 [simp]: "of_hypint 1 = 1"*
⟨*proof*⟩

lemma *of_hypint_add [simp]: "⋀w z. of_hypint (w+z) = of_hypint w + of_hypint*

z"
⟨proof⟩

lemma of_hypint_minus [simp]: "⋀z. of_hypint (-z) = - (of_hypint z)"
⟨proof⟩

lemma of_hypint_diff [simp]: "⋀w z. of_hypint (w - z) = of_hypint w - of_hypint z"
⟨proof⟩

lemma of_hypint_mult [simp]: "⋀w z. of_hypint (w*z) = of_hypint w * of_hypint z"
⟨proof⟩

lemma of_hypint_of_nat_eq [simp]: "of_hypint (of_nat n) = of_nat n"
⟨proof⟩

lemma of_hypint_nat_power: "of_hypint (z ^ n) = of_hypint z ^ n"
⟨proof⟩

lemma of_hypint_eq_iff [simp]:
"⋀w z. of_hypint w = of_hypint z ⟷ w = z"
⟨proof⟩

lemma of_hypint_eq_0_iff [simp]:
"⋀z. of_hypint z = 0 ⟷ z = 0"
⟨proof⟩

lemma of_hypint_0_eq_iff [simp]:
"⋀z. 0 = of_hypint z ⟷ z = 0"
⟨proof⟩

lemma of_hypint_le_iff [simp]:
"⋀ w z. of_hypint w ≤ of_hypint z ⟷ w ≤ z"
⟨proof⟩

lemma of_hypint_less_iff [simp]:
"⋀w z. of_hypint w < of_hypint z ⟷ w < z"
⟨proof⟩

lemma of_hypint_0_le_iff [simp]:
"⋀z. 0 ≤ of_hypint z ⟷ 0 ≤ z"
⟨proof⟩

lemma of_hypint_le_0_iff [simp]:
"⋀z. of_hypint z ≤ 0 ⟷ z ≤ 0"
⟨proof⟩

lemma of_hypint_0_less_iff [simp]:

```
  "⋀z. 0 < of_hypint z ⟷ 0 < z"
```
⟨*proof*⟩

lemma *of_hypint_less_0_iff [simp]:*
```
  "⋀z. of_hypint z < 0 ⟷ z < 0"
```
⟨*proof*⟩

lemma *of_hypint_abs:* `"⋀z. of_hypint (abs z) = abs(of_hypint z)"`
⟨*proof*⟩

lemma *of_hypint_add_one:* `"of_hypint z + (1 :: real star) = of_hypint (z + 1)"`
⟨*proof*⟩

3.3 Embedding the naturals into the hyperintegers

abbreviation
```
  hypint_of_nat :: "nat => hypint" where
  "hypint_of_nat == of_nat"
```

lemma *HypintNat_eq:* `"Nats = {z. ∃N. z = hypint_of_nat N}"`
⟨*proof*⟩

lemma *hypint_of_nat:*
```
     "hypint_of_nat m = star_n (%n. int m)"
```
⟨*proof*⟩

lemma *HypintNat_ge_zero:* `"n ∈ ℕ ⟹ 0 ≤ (n :: hypint)"`
⟨*proof*⟩

lemma *hypint_of_hypnat_hypint_of_nat_eq [simp]:*
```
     "⋀n. hypint_of_hypnat (hypnat_of_nat n) = hypint_of_nat n"
```
⟨*proof*⟩

lemma *abs_hypint_of_hypnat [simp]:* `"|hypint_of_hypnat n| = hypint_of_hypnat n"`
⟨*proof*⟩

lemma *hypint_abs_less_one_iff [simp]:* `"⋀z. (|z| < 1) = (z = (0::hypint))"`
⟨*proof*⟩

lemma *abs_hypint_mult_eq_1:*
```
    "⋀m n. |m * n| = 1 ⟹ |m| = (1::hypint)"
```
⟨*proof*⟩

lemma *pos_hypint_mult_eq_1_iff_lemma:*
```
    "⋀m n. (m * n = 1) ==> m = (1::hypint) | m = -1"
```
⟨*proof*⟩

lemma *pos_hypint_mult_eq_1_iff:*
"⋀m n. 0 < (m::hypint) ⟹ (m * n = 1) = (m = 1 ∧ n = 1)"
⟨*proof*⟩

lemma *hypint_mult_eq_1_iff:*
"⋀m n. (m*n = (1::hypint)) = ((m = 1 ∧ n = 1) ∨ (m = -1 ∧ n = -1))"
⟨*proof*⟩

3.4 Magnitude of a hyperinteger

definition
hypnat :: "hypint => hypnat"
where
*[transfer_unfold]: "hypnat ≡ *f* nat"*

lemma *hypnat_hypint [simp]: "⋀n. hypnat (of_hypnat n) = n"*
⟨*proof*⟩

lemma *hypnat_zero [simp]: "hypnat 0 = 0"*
⟨*proof*⟩

lemma *hypnat_one [simp]: "hypnat 1 = 1"*
⟨*proof*⟩

lemma *hypint_hypnat_eq [simp]: "⋀z. of_hypnat (hypnat z) = (if 0 ≤ z then z else 0)"*
⟨*proof*⟩

corollary *hypnat_0_le: "0 ≤ z ⟹ of_hypnat (hypnat z) = z"*
⟨*proof*⟩

lemma *hypnat_le_0 [simp]: "⋀z. z ≤ 0 ⟹ hypnat z = 0"*
⟨*proof*⟩

lemma *hypnat_le_eq_zle: "⋀w z. 0 < w ∨ 0 ≤ z ⟹ (hypnat w ≤ hypnat z) = (w ≤ z)"*
⟨*proof*⟩

corollary *hypnat_mono_iff: "0 < z ⟹ (hypnat w < hypnat z) = (w < z)"*
⟨*proof*⟩

corollary *hypnat_less_eq_less: "0 ≤ w ⟹ (hypnat w < hypnat z) = (w<z)"*
⟨*proof*⟩

lemma *less_hypnat_conj [simp]: "⋀w z. (hypnat w < hypnat z) = (0 < z ∧ w < z)"*
⟨*proof*⟩

lemma *nonneg_eq_hypint:*

fixes *z :: hypint*
assumes *"0 ≤ z"* **and** *"⋀m. z = of_hypnat m ⟹ P"*
shows *P*
⟨*proof*⟩

lemma *hypnat_eq_iff:*
"⋀ m w. (hypnat w = m) = (if 0 ≤ w then w = of_hypnat m else m=0)"
⟨*proof*⟩

corollary *hypnat_eq_iff2: "(m = hypnat w) = (if 0 ≤ w then w = of_hypnat m else m=0)"*
⟨*proof*⟩

lemma *hypnat_less_iff: "⋀m w. 0 ≤ w ⟹ (hypnat w < m) = (w < of_hypnat m)"*
⟨*proof*⟩

lemma *hypnat_0_iff[simp]: "⋀i. hypnat i = 0 ⟷ i≤0"*
⟨*proof*⟩

lemma *hypint_eq_iff: "(of_hypnat m = z) = (m = hypnat z ∧ 0 ≤ z)"*
⟨*proof*⟩

lemma *zero_less_hypnat_eq [simp]: "(0 < hypnat z) = (0 < z)"*
⟨*proof*⟩

lemma *hypnat_add_distrib:*
"⋀z z'. ⟦ (0::hypint) ≤ z; 0 ≤ z' ⟧ ⟹ hypnat (z+z') = hypnat z + hypnat z'"
⟨*proof*⟩

lemma *hypnat_diff_distrib:*
"⋀z z'. ⟦ (0::hypint) ≤ z'; z' ≤ z ⟧ ⟹ hypnat (z-z') = hypnat z - hypnat z'"
⟨*proof*⟩

lemma *hypnat_minus_hypint [simp]: "⋀n. hypnat (- (of_hypnat n)) = 0"*
⟨*proof*⟩

lemma *less_hypnat_eq_hypint_less: "⋀m z. (m < hypnat z) = (of_hypnat m < z)"*
⟨*proof*⟩

lemma *hypnat_add_one: "0 ≤ z ⟹ hypnat z + (1 :: nat star) = hypnat (z + 1)"*
⟨*proof*⟩

3.5 The divides relation

lemma *hypint_dvd_antisym_nonneg:*
"⋀m n. 0 <= m ⟹ 0 <= n ⟹ m dvd n ⟹ n dvd m ⟹ m = (n::hypint)"
⟨*proof*⟩

lemma *hypint_dvd_antisym_abs:*
"⋀a b. ⟦ (a::hypint) dvd b; b dvd a ⟧ ⟹ |a| = |b|"
⟨*proof*⟩

lemma *hyperint_dvd_diffD:* "⋀k m n. k dvd m - n ⟹ k dvd n ⟹ k dvd (m::hypint)"
⟨*proof*⟩

lemma *hypint_dvd_reduce:* "⋀k m n. (k dvd n + k * m) = (k dvd (n::hypint))"
⟨*proof*⟩

lemma *hypint_dvd_imp_le_hypint:*
"⋀d i. ⟦ i ≠ 0; d dvd (i::hypint) ⟧ ⟹ |d| ≤ |i|"
⟨*proof*⟩

lemma *hypint_dvd_not_less:*
"⋀m n. ⟦ 0 < m; m < (n::hypint) ⟧ ⟹ ¬ n dvd m"
⟨*proof*⟩

lemma *hypint_dvd_mult_cancel:*
"⋀k m n. ⟦ k * m dvd k * n; k ≠ (0::hypint) ⟧ ⟹ m dvd n"
⟨*proof*⟩

theorem *dvd_hypint_of_hypnat:*
"⋀x y. ((x::hypnat) dvd y) = ((hypint_of_hypnat x) dvd (hypint_of_hypnat y))"
⟨*proof*⟩

lemma *hypint_dvd1_eq[simp]:* "⋀x. (x::hypint) dvd 1 = (|x| = 1)"
⟨*proof*⟩

lemma *hypint_dvd_mult_cancel1:*
"⋀m n. m ≠ (0::hypint) ⟹ (m * n dvd m) = (|n| = 1)"
⟨*proof*⟩

lemma *hypint_of_hypnat_dvd_iff:*
"⋀m z. ((of_hypnat m) dvd z) = (m dvd hypnat (abs z))"
⟨*proof*⟩

lemma *dvd_hypint_iff:* "⋀m z. (z dvd of_hypnat m) = (hypnat (abs z) dvd m)"
⟨*proof*⟩

lemma *hypnat_dvd_iff:* "(hypnat z dvd m) = (if 0 ≤ z then (z dvd of_hypnat

m) else m = 0)"
⟨proof⟩

lemma eq_hypnat_hypnat_iff:
"0 ≤ z ⟹ 0 ≤ z' ⟹ hypnat z = hypnat z' ⟷ z = z'"
⟨proof⟩

lemma hypnat_power_eq:
"⋀z. 0 ≤ z ⟹ hypnat (z ^ n) = hypnat z ^ n"
⟨proof⟩

lemma hypint_dvd_imp_le:
"⋀n z. ⟦ z dvd n; 0 < n ⟧ ⟹ z ≤ (n::hypint)"
⟨proof⟩

lemma hypint_period:
"⋀a c d t x. (a::hypint) dvd d ⟹ a dvd (x + t) ⟷ a dvd ((x + c
* d) + t)"
⟨proof⟩

3.6 Division on *hypint*

lemma hypint_of_nat_div: "hypint_of_nat(m div n) = hypint_of_nat m div
(hypint_of_nat n)"
⟨proof⟩

lemma hypint_of_hypnat_div: "⋀m n. hypint_of_hypnat(m div n) = hypint_of_hypnat
m div (hypint_of_hypnat n)"
⟨proof⟩

lemma HypintNats_div: "⟦(n::hypint) ∈ ℕ; m ∈ ℕ ⟧ ⟹ n div m ∈ ℕ"
⟨proof⟩

lemma hypint_div_mono1: "⋀a a' b. ⟦ a ≤ a'; 0 < (b::hypint) ⟧ ⟹ a
div b ≤ a' div b"
⟨proof⟩

lemma hypint_div_neg_pos_less0: "⋀a b. ⟦ a < (0::hypint); 0 < b ⟧ ⟹
a div b < 0"
⟨proof⟩

lemma hypint_div_nonneg_neg_le0: "⋀a b. ⟦ (0::hypint) ≤ a; b < 0 ⟧ ⟹
a div b ≤ 0"
⟨proof⟩

lemma hypint_div_nonpos_pos_le0: "⋀a b. ⟦ (a::hypint) ≤ 0; b > 0 ⟧ ⟹
a div b ≤ 0"
⟨proof⟩

lemma *hypint_pos_imp_div_nonneg_iff: "⋀a b. (0::hypint) < b ⟹ (0 ≤ a div b) = (0 ≤ a)"*
⟨*proof*⟩

lemma *hypint_neg_imp_div_nonneg_iff:*
"⋀a b. b < (0::hypint) ⟹ (0 ≤ a div b) = (a ≤ (0::hypint))"
⟨*proof*⟩

lemma *hypint_pos_imp_div_neg_iff: "⋀a b. (0::hypint) < b ⟹ (a div b < 0) = (a < 0)"*
⟨*proof*⟩

lemma *hypint_neg_imp_div_neg_iff: "⋀a b. b < (0::hypint) ⟹ (a div b < 0) = (0 < a)"*
⟨*proof*⟩

lemma *hypint_nonneg1_imp_div_pos_iff:*
"⋀a b. (0::hypint) ≤ a ⟹ (a div b > 0) = (a ≥ b ∧ b > 0)"
⟨*proof*⟩

3.7 Properties of the set of embedded integers

lemma *of_int_eq_star_of [simp]: "of_int = star_of"*
⟨*proof*⟩

lemma *Ints_eq_Standard: "(ℤ :: int star set) = Standard"*
⟨*proof*⟩

lemma *hypint_of_int_mem_Ints [simp]: "hypint_of_int z ∈ ℤ"*
⟨*proof*⟩

lemma *hypint_of_nat_Suc [simp]:*
"hypint_of_nat (Suc n) = hypint_of_nat n + (1::hypint)"
⟨*proof*⟩

lemma *of_int_eq_add [rule_format]:*
"∀d::hypint. of_int m = of_int n + d ⟶ d ∈ range of_int"
⟨*proof*⟩

3.8 Infinite hyperintegers

definition

HIntInfinite :: "hypint set" **where**
"HIntInfinite = {z. z ∉ ℤ}"

lemma *Ints_not_HIntInfinite_iff: "(x ∈ Ints) = (x ∉ HIntInfinite)"*
⟨*proof*⟩

lemma *HIntInfinite_not_Ints_iff: "(x ∈ HIntInfinite) = (x ∉ Ints)"*

⟨*proof*⟩

lemma *star_of_neq_HIntInfinite: "Z ∈ HIntInfinite ==> star_of n ≠ Z"*
⟨*proof*⟩

lemma *HIntInfinite_minus_iff [simp]:*
"(- Z ∈ HIntInfinite) = (Z ∈ HIntInfinite)"
⟨*proof*⟩

lemma *star_of_less_pos_HIntInfinite:*
assumes *"0 < Z"* **and** *"Z ∈ HIntInfinite"*
shows *"star_of z < Z"*
⟨*proof*⟩

lemma *star_of_gt_neg_HIntInfinite:*
assumes *"Z < 0"*
and *"Z ∈ HIntInfinite"*
shows *"Z < star_of z"*
⟨*proof*⟩

lemma *zero_not_HIntInfinite [simp]: "0 ∉ HIntInfinite"*
⟨*proof*⟩

lemma *star_of_less_abs_HIntInfinite:*
"Z ∈ HIntInfinite ⟹ star_of z < abs Z"
⟨*proof*⟩

lemma *hypint_of_nat_less_abs_HIntInfinite:*
"y ∈ HIntInfinite ⟹ hypint_of_nat n < |y|"
⟨*proof*⟩

lemma *Ints_less_pos_HIntInfinite: "⟦ 0 < y; x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ x < y"*
⟨*proof*⟩

lemma *Ints_gt_neg_HIntInfinite: "⟦ y < 0; x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ y < x"*
⟨*proof*⟩

lemma *Ints_less_abs_HIntInfinite: "⟦ x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ x < abs y"*
⟨*proof*⟩

lemma *Ints_le_abs_HIntInfinite: "⟦ x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ x ≤ abs y"*
⟨*proof*⟩

lemma *Nats_less_abs_HIntInfinite: "⟦ x ∈ ℕ; y ∈ HIntInfinite ⟧ ⟹ x*

```
< abs y"
```
⟨*proof*⟩

lemma *Nats_le_abs_HIntInfinite:* "⟦ *x* ∈ ℕ; *y* ∈ *HIntInfinite* ⟧ ⟹ *x* < *abs y*"
⟨*proof*⟩

lemma *Ints_pos_downward_closed:*
"⟦ *0* ≤ *y*; *x* ∈ *Ints*; *(y::hypint)* ≤ *x* ⟧ ⟹ *y* ∈ ℤ"
⟨*proof*⟩

lemma *Ints_neg_upward_closed:*
"⟦ *y* ≤ *0*; *x* ∈ *Ints*; *x* ≤ *(y::hypint)* ⟧ ⟹ *y* ∈ ℤ"
⟨*proof*⟩

lemma *HIntInfinite_pos_upward_closed:*
"⟦ *0* ≤ *x*; *x* ∈ *HIntInfinite*; *x* ≤ *y* ⟧ ⟹ *y* ∈ *HIntInfinite*"
⟨*proof*⟩

lemma *HIntInfinite_neg_downward_closed:*
"⟦ *x* ≤ *0*; *x* ∈ *HIntInfinite*; *y* ≤ *x* ⟧ ⟹ *y* ∈ *HIntInfinite*"
⟨*proof*⟩

lemma *HIntInfinite_pos_add:*
"⟦ *0* ≤ *x*; *0* ≤ *y*; *x* ∈ *HIntInfinite* ⟧ ⟹ *x* + *y* ∈ *HIntInfinite*"
⟨*proof*⟩

lemma *HIntInfinite_neg_add:*
"⟦ *x* ≤ *0*; *y* ≤ *0*; *x* ∈ *HIntInfinite* ⟧ ⟹ *x* + *y* ∈ *HIntInfinite*"
⟨*proof*⟩

lemma *HIntInfinite_add_Ints:*
"⟦ *x* ∈ *HIntInfinite*; *y* ∈ *Ints* ⟧ ⟹ *x* + *y* ∈ *HIntInfinite*"
⟨*proof*⟩

lemma *HIntInfinite_diff:*
"⟦ *x* ∈ *HIntInfinite*; *y* ∈ *Ints* ⟧ ⟹ *x* - *y* ∈ *HIntInfinite*"
⟨*proof*⟩

lemma *Ints_def2:* "ℤ = {*z*. ∃*Z*. *z* = *hypint_of_int Z*}"
⟨*proof*⟩

lemma *HIntInfinite_hypint_of_hypnat_iff:*
"*(hypint_of_hypnat n* ∈ *HIntInfinite) = (n* ∈ *HNatInfinite)*"
⟨*proof*⟩

lemma *two_hyperpow_ge_one [simp]:* "*(1::hypint)* ≤ *2 pow n*"
⟨*proof*⟩

3.9 Embedding of hypertintegers in hyperreals

abbreviation
hypreal_of_hypint :: "hypint ⇒ hypreal" **where**
"hypreal_of_hypint ≡ of_hypint"

lemma *of_hypint_numeral [simp]: "of_hypint (numeral v) = numeral v"*
⟨*proof*⟩

lemma *real_of_int_le_real_of_nat_iff: "(real_of_int a ≤ real_of_nat b) = (a ≤ int b)"*
⟨*proof*⟩

lemma *minus_real_of_int_le_real_of_nat_iff: "(- real_of_int a ≤ real_of_nat b) = (- a ≤ int b)"*
⟨*proof*⟩

lemma *hypreal_of_hypint_le_hypreal_of_hypnat_iff:*
"⋀a b.(hypreal_of_hypint a ≤ of_hypnat b) = (a ≤ of_hypnat b)"
⟨*proof*⟩

lemma *minus_hypreal_of_hypint_le_hypreal_of_hypnat_iff:*
"⋀a b.(- hypreal_of_hypint a ≤ hypreal_of_hypnat b) = (- a ≤ of_hypnat b)"
⟨*proof*⟩

lemma *hnorm_of_hypint [simp]:*
"⋀z. hnorm (of_hypint z::'a::{real_normed_algebra_1, linordered_idom} star) = abs(of_hypint z)"
⟨*proof*⟩

lemma *of_hypnat_hypnat [simp]: "⋀z. 0 ≤ z ⟹ of_hypnat(hypnat z) = of_hypint z"*
⟨*proof*⟩

lemma *HInfinite_def3: "HInfinite = {x. ∀z ∈ Ints. hypreal_of_hypint z < hnorm x}"*
⟨*proof*⟩

lemma *HInfinite_of_hypint_HNatInfinite_hypnat:*
"⟦ (of_hypint x :: 'a::{real_normed_algebra_1,linordered_idom} star) ∈ HInfinite; x > 0 ⟧
⟹ hypnat x ∈ HNatInfinite"
⟨*proof*⟩

lemma *HNatInfinite_hypnat_HInfinite_of_hypint:*
"hypnat x ∈ HNatInfinite
⟹ (of_hypint x :: 'a::{real_normed_algebra_1,linordered_idom} star) ∈ HInfinite"

⟨*proof*⟩

end

4 Internal sets and functions

theory *Internal*
imports *"HOL-Nonstandard_Analysis.NatStar" HyperInt AuxiliaryNSA*
begin

definition
n_star :: "'a star ⇒ (nat ⇒ 'a)" **where**
"n_star x = (SOME y. x = star_n y)"

definition
*n_starset :: "'a star set ⇒ (nat ⇒ 'a set)" ("*ns* _" [80] 80)* **where**
*"*ns* A = (SOME As. A = Iset (star_n As))"*

definition
*n_starfun :: "('a star ⇒ 'b star) ⇒ (nat ⇒ ('a ⇒ 'b))" ("*nf* _" [80] 80)* **where**
*"*nf* F = (SOME Fs. F = Ifun (star_n Fs))"*

definition
*starfun2_n :: "(nat ⇒ ('a ⇒ 'b ⇒ 'c)) ⇒ ('a star ⇒ 'b star ⇒ 'c star)" ("*fn2* _" [80] 80)* **where**
*"*fn2* F = (λx. Ifun (star_n F ⋆ x))"*

definition
*n_starfun2 :: "('a star ⇒ 'b star ⇒ 'c star) ⇒ (nat ⇒ ('a ⇒ 'b ⇒ 'c))" ("*nf2* _" [80] 80)* **where**
*"*nf2* F = (SOME Gs. F = (λx. Ifun (star_n Gs ⋆ x)))"*

definition
InternalFuns2 :: "('a star ⇒ 'b star ⇒ 'c star) set" **where**
*"InternalFuns2 = {X. ∃F. X = *fn2* F}"*

lemma *n_star_def2: "n_star x = (SOME y. y ∈ Rep_star x)"*
⟨*proof*⟩

lemma *star_n_star [simp]:*
"star_n (n_star x) = x"
⟨*proof*⟩

lemma *n_star_star_n_eq_ultra:*
"eventually (λn. n_star (star_n X) n = X n) 𝒰"
⟨*proof*⟩

lemma *n_star_star_of_eq_ultra:*

```
  "eventually  (λn. n_star (star_of X) n = X)  𝒰"
```
⟨*proof*⟩

lemma `n_starset_def2: "*ns* S = (SOME Sn. S = *sn* Sn)"`
⟨*proof*⟩

lemma `Iset_n_starset_eq [simp]:`
assumes `"X ∈ InternalSets"`
shows `"Iset (star_n (*ns* X)) = X"`
⟨*proof*⟩

lemma `Iset_n_starset_empty [simp]:`
`"Iset(star_n (*ns* {})) = {}"`
⟨*proof*⟩

lemma `n_starset_eq_ultra:`
`"eventually (λn. (*ns* (Iset (star_n X))) n = X n) 𝒰"`
⟨*proof*⟩

lemma `n_starset_starset_n_eq_ultra:`
`"eventually (λn. (*ns* *sn* X) n = X n) 𝒰"`
⟨*proof*⟩

lemma `n_starset_starset_n_eq_ultra2:`
`"eventually (λn. X n = (*ns* *sn* X) n) 𝒰"`
⟨*proof*⟩

lemma `n_starset_empty_ultra:`
`"eventually (λn. (*ns* {}) n = {}) 𝒰"`
⟨*proof*⟩

lemma `P_n_starset_starset_n_ultra_iff:`
`"eventually (λn. P ((*ns* *sn* X) n)) 𝒰 = (eventually (λn. P (X n)) 𝒰)"`
⟨*proof*⟩

lemma `InternalSets_starset_n_starset [simp]:`
assumes `"A ∈ InternalSets"`
shows `"*sn* (*ns* A) = A"`
⟨*proof*⟩

lemma `starset_n_star_n:`
`"(star_n X ∈ *sn* As) =  (eventually (λn. (X n) ∈ (As n)) 𝒰)"`
⟨*proof*⟩

lemma `n_starfun_def2: "*nf* F = (SOME Fn. F = *fn* Fn)"`
⟨*proof*⟩

lemma `InternalFuns_starfun_n_starfun [simp]:`

assumes *"F ∈ InternalFuns"*
shows *"*fn* (*nf* F) = F"*
⟨*proof*⟩

lemma *Ifun_eq_cancel:*
"(Ifun (star_n F) = Ifun (star_n G)) = (eventually (λn. F n = G n) 𝒰)"
⟨*proof*⟩

lemma *starfun_n_eq_cancel:*
*"(*fn* F = *fn* G) = (eventually (λn. F n = G n) 𝒰)"*
⟨*proof*⟩

lemma *n_starfun_starfun_n_eq_ultra:*
*"eventually (λn. (*nf* (*fn* F)) n = F n) 𝒰"*
⟨*proof*⟩

lemma *n_starfun_starfun_eq_ultra:*
*"eventually (λn. (*nf* (*f* f)) n = f) 𝒰"*
⟨*proof*⟩

lemma *P_n_starfun_starfun_n_ultra_iff:*
*"eventually (λn. P ((*nf* *fn* X) n)) 𝒰 = (eventually (λn. P (X n)) 𝒰)"*
⟨*proof*⟩

lemma *starfun2_eq_starfun2n: "*f2* f = *fn2* (λn. f)"*
⟨*proof*⟩

lemma *starfun2_n_Ifun_starfun_n: "*fn2* f = (λx. Ifun ((*fn* f) x))"*
⟨*proof*⟩

lemma *starfun2_n_Ifun_starfun_n2: "(*fn2* f) x y = ((*fn* f) x) ⋆ y"*
⟨*proof*⟩

The definition of binary internal functions is as expected

lemma *starfun2_n: "(*fn2* f) (star_n X) (star_n Y) = star_n (λn. f n (X n) (Y n))"*
⟨*proof*⟩

lemma *n_starfun2_starfun2_n_eq_ultra:*
*"eventually (λn. (*nf2* (*fn2* F)) n = F n) 𝒰"*
⟨*proof*⟩

lemma *n_starfun2_starfun2_eq_ultra: "eventually (λn. (*nf2* (*f2* f)) n = f) 𝒰"*
⟨*proof*⟩

lemma *InternalFuns2_starfun2_n_starfun2 [simp]:*
assumes *"f ∈ InternalFuns2"*

shows *"*fn2* (*nf2* f) = f"*
⟨*proof*⟩

lemma *transfer_fun2_eq [transfer_intro]:*
"⟦ ⋀X Y. f (star_n X) (star_n Y) = g (star_n X) (star_n Y)
≡ eventually (λn. F n (X n) (Y n) = G n (X n) (Y n)) 𝒰 ⟧
⟹ f = g ≡ eventually (λn. F n = G n) 𝒰"
⟨*proof*⟩

lemma *starfun2_n_eq_cancel:*
*"(*fn2* F = *fn2* G) = (eventually (λn. F n = G n) 𝒰)"*
⟨*proof*⟩

lemma *n_starfun_starfun2_n_eq_ultra:*
*"∀ F n in 𝒰. (*nf* (*fn2* f) (star_n X)) n = f n (X n)"*
⟨*proof*⟩

lemma *starfun2_n_minus: "- (*fn2* f) x y = (*fn2* (λi j x. - (f i j) x)) x y"*
⟨*proof*⟩

lemma *starfun2_n_diff:*
*"(*fn2* f) y z - (*fn2* g) y z = (*fn2* (λi j x. f i j x - g i j x)) y z"*
⟨*proof*⟩

lemma *starfun2_n_inverse: "inverse ((*fn2* f) x y) = (*fn2* (λi j x. inverse ((f i j) x))) x y"*
⟨*proof*⟩

lemma *starfun2_n_mult:*
*"(*fn2* f) y z * (*fn2* g) y z = (*fn2* (λi j x. f i j x * g i j x)) y z"*
⟨*proof*⟩

lemma *starfun2_n_divide:*
*"((*fn2* f) y z ::'a::division_ring star)/ (*fn2* g) y z =*
*(*fn2* (λi j x. f i j x / g i j x)) y z"*
⟨*proof*⟩

lemma *starfun_n_starfun2_omega: "(*fn* F) z = (*f2* F) whn z"*
⟨*proof*⟩

lemma *InternalFuns2_divide:*
"⟦ f ∈ InternalFuns2; g ∈ InternalFuns2 ⟧
⟹ (λm n. (f m n ::'a::division_ring star)/g m n) ∈ InternalFuns2"
⟨*proof*⟩

lemma *InternalFuns2_mult:*

"⟦ f ∈ InternalFuns2; g ∈ InternalFuns2 ⟧ ⟹ (λm n. f m n * g m n) ∈ InternalFuns2"
⟨proof⟩

lemma InternalFuns2_diff:
"⟦ f ∈ InternalFuns2; g ∈ InternalFuns2 ⟧ ⟹ (λm n. f m n - g m n) ∈ InternalFuns2"
⟨proof⟩

lemma InternalFuns2_minus:
"f ∈ InternalFuns2 ⟹ (λm n. - f m n) ∈ InternalFuns2"
⟨proof⟩

lemma InternalFuns_starfun2_n [simp]: "*fn2* f ∈ InternalFuns2"
⟨proof⟩

lemma ext_obj: "(∀x y. f x y = g x y) ⟹ f = g"
⟨proof⟩

lemma InternalFuns2_const_fun: "(λn m. c) ∈ InternalFuns2"
⟨proof⟩

lemma InternalFuns2_InternalFuns_iff:
"((λm n. f m) ∈ InternalFuns2) = (f ∈ InternalFuns)"
⟨proof⟩

lemma InternalSets_hnorm_InternalFuns_gt [simp]:
assumes "X ∈ InternalFuns"
shows "{n. hnorm (X n) > m} ∈ InternalSets"
⟨proof⟩

lemma InternalSets_hnorm_starfun_gt [simp]:
"⋀m. {n. hnorm((*f* X) n) > m} ∈ InternalSets"
⟨proof⟩

4.1 Overspill theorem

Theorem 3.3.18, Nonstandard Analysis, A. Robinson

lemma InternalFuns_ub:
assumes "X ∈ InternalFuns"
and "∀n∈ℕ. hnorm(X n) ≤ m"
shows "∃v∈HNatInfinite. ∀n<v. hnorm(X n) ≤ m"
⟨proof⟩

lemma starfun_least:
"∀n∈Nats. hnorm((*f* X) n) ≤ m ⟹ ∃v∈HNatInfinite. ∀n<v. hnorm((*f* X) n) ≤ m"
⟨proof⟩

4.2 The Sequential Theorem

Theorem 3.3.20, Nonstandard Analysis, A. Robinson

lemma *InternalFuns_Sequential_Theorem:*
assumes *Int_f: "f ∈ InternalFuns"*
and *f_approx_zero: "∀n∈Nats. f n ≈ (0:: 'a::{real_normed_div_algebra, semiring_1_cancel} star)"*
shows *"∃N∈HNatInfinite.∀n<N. f n ≈ 0"*
⟨*proof*⟩

lemma *InternalFuns_Sequential_Theorem2:*
assumes *"f ∈ InternalFuns" "g ∈ InternalFuns"*
"∀n∈Nats. f n ≈ (g n :: 'a::{real_normed_div_algebra, semiring_1_cancel} star)"
shows *"∃N∈HNatInfinite.∀n<N. f n ≈ g n"*
⟨*proof*⟩

lemma *Sequential_Theorem:*
*"∀n∈Nats. (*f* f) n ≈ (0:: 'a::{real_normed_div_algebra, semiring_1_cancel} star)*
*⟹ ∃N∈HNatInfinite.∀n<N. (*f* f) n ≈ 0"*
⟨*proof*⟩

lemma *Sequential_Theorem2:*
*"∀n∈Nats. (*f* f) n ≈ ((*f* g) n :: 'a::{real_normed_div_algebra, semiring_1_cancel} star)*
*⟹ ∃N∈HNatInfinite.∀n<N. (*f* f) n ≈ (*f* g) n"*
⟨*proof*⟩

lemma *hyperpower_starfun_n:*
*"(x::'a::power star) pow n = (*fn* (λn m. (n_star x) n ^ m)) n"*
⟨*proof*⟩

lemma *InternalFuns2_hyperpower [simp]: "(λm. (pow) x) ∈ InternalFuns2"*
⟨*proof*⟩

lemma *hyperpow_approx_one_ex:*
assumes *"x ≈ 1"*
shows *"∃N∈HNatInfinite.∀n<N. x pow n ≈ (1::'a::real_normed_div_algebra star)"*
⟨*proof*⟩

end

theory *Hyperfinite_Set*
imports *"HOL-Nonstandard_Analysis.NatStar" Internal*
begin

5 Hyperfinite sets

definition
hyperfinite :: "'a star set ⇒ bool" **where**
*"hyperfinite S ≡ unstar (star_n (λn. finite ((*ns* S) n))) "*

Usual definition of a hyperfinite set

lemma *hyperfinite_iff:*
*"hyperfinite S = (eventually (λn. finite ((*ns* S) n)) 𝒰)"*
⟨*proof*⟩

lemma *hyperfinite_starset_n_iff:*
*"hyperfinite (*sn* X) = (eventually (λn. finite (X n)) 𝒰)"*
⟨*proof*⟩

lemma *hyperfinite_empty [simp]: "hyperfinite {}"*
⟨*proof*⟩

lemma *hyperfinite_Iset_iff_starP_finite:*
*"hyperfinite (Iset X) = (*p* finite) X"*
⟨*proof*⟩

lemma *finite_hyperfinite_starset:*
*"finite X ⟹ hyperfinite (*s* X)"*
⟨*proof*⟩

lemma *hyperfinite_starset_finite:*
*"hyperfinite (*s* X) ⟹ finite X"*
⟨*proof*⟩

lemma *hyperfinite_starset_iff_finite:*
*"hyperfinite (*s* X) = finite X"*
⟨*proof*⟩

5.1 Properties of hyperfinite sets

lemma *hyperfinite_UnI:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets;
hyperfinite X; hyperfinite Y ⟧ ⟹ hyperfinite (X ∪ Y)"
⟨*proof*⟩

lemma *hyperfinite_starset_n_UnI:*
*"⟦ hyperfinite (*sn* X); hyperfinite (*sn* Y) ⟧ ⟹ hyperfinite (*sn* X ∪ *sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_starset_n_Un [iff]:*
*"hyperfinite ((*sn* X) ∪ (*sn* Y)) ⟷ hyperfinite (*sn* X) ∧ hyperfinite (*sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_singleton [simp]: "hyperfinite {x}"*
⟨*proof*⟩

lemma *finite_hyperfinite:*
assumes *"finite X"*
shows *"hyperfinite X"*
⟨*proof*⟩

lemma *hyperfinite_insert [simp]:*
"A ∈ InternalSets ⟹ hyperfinite (insert a A) ⟷ hyperfinite A"
⟨*proof*⟩

lemma *hyperfinite_starset_n_insert [simp]:*
*"hyperfinite (insert a (*sn* A)) ⟷ hyperfinite (*sn* A)"*
⟨*proof*⟩

lemma *hyperfinite_subset:*
assumes *"A ∈ InternalSets"* **and** *"B ∈ InternalSets"* **and**
"hyperfinite B" **and** *"A ⊆ B"*
shows *"hyperfinite A"*
⟨*proof*⟩

lemma *hyperfinite_starset_n_subset:*
*"⟦ hyperfinite (*sn* Y); (*sn* X) ⊆ (*sn* Y) ⟧ ⟹ hyperfinite (*sn* X)"*
⟨*proof*⟩

lemma *hyperfinite_Int [simp, intro]:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets;
hyperfinite X ∨ hyperfinite Y ⟧ ⟹ hyperfinite (X ∩ Y)"
⟨*proof*⟩

lemma *hyperfinite_IntI1:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets; hyperfinite X ⟧ ⟹ hyperfinite (X ∩ Y)"
⟨*proof*⟩

lemma *hyperfinite_IntI2:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets; hyperfinite Y ⟧ ⟹ hyperfinite (X ∩ Y)"
⟨*proof*⟩

lemma *hyperfinite_starset_n_Int [simp, intro]:*
*"hyperfinite (*sn* X) ∨ hyperfinite (*sn* Y) ⟹ hyperfinite (*sn* X ∩ *sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_starset_n_IntI1:*
*"hyperfinite (*sn* X) ⟹ hyperfinite (*sn* X ∩ *sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_starset_n_IntI2:*
*"hyperfinite (*sn* Y) ⟹ hyperfinite (*sn* X ∩ *sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_Diff [simp, intro]:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets; hyperfinite X ⟧ ⟹ hyperfinite (X - Y)"
⟨*proof*⟩

lemma *hyperfinite_starset_n_Diff [simp, intro]:*
*"hyperfinite (*sn* X) ⟹ hyperfinite (*sn* X - *sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_Diff2 [simp]:*
assumes *"X ∈ InternalSets"*
and *"Y ∈ InternalSets"*
and *"hyperfinite Y"*
shows *"hyperfinite (X - Y) ⟷ hyperfinite X"*
⟨*proof*⟩

lemma *hyperfinite_starset_n_Diff2 [simp]:*
*"hyperfinite (*sn* X)*
*⟹ hyperfinite (*sn* X - *sn* Y) ⟷ hyperfinite (*sn* X)"*
⟨*proof*⟩

lemma *hyperfinite_Diff_insert [iff]:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets ⟧
⟹ hyperfinite (X - insert a Y) ⟷ hyperfinite (X - Y)"
⟨*proof*⟩

lemma *hyperfinite_starset_n_Diff_insert [iff]:*
*"hyperfinite (*sn* X - insert a (*sn* Y)) ⟷ hyperfinite (*sn* X - *sn* Y)"*
⟨*proof*⟩

lemma *hyperfinite_Compl[simp]:*
"⟦ X ∈ InternalSets; hyperfinite (X :: 'a star set) ⟧
⟹ hyperfinite (- X) ⟷ hyperfinite (UNIV :: 'a star set)"
⟨*proof*⟩

lemma *hyperfinite_starset_n_Compl[simp]:*
*"hyperfinite (*sn* X :: 'a star set)*
*⟹ hyperfinite (- (*sn* X)) ⟷ hyperfinite (UNIV :: 'a star set)"*
⟨*proof*⟩

lemma *hyperfinite_Collect_not[simp]:*
"⟦ {x :: 'a star. P x} ∈ InternalSets; hyperfinite {x :: 'a star. P x} ⟧ ⟹ hyperfinite {x. ¬ P x} ⟷ hyperfinite (UNIV :: 'a star set)"
⟨*proof*⟩

lemma *hyperfinite_starset_Collect_not[simp]:*
*"hyperfinite {x :: 'a star. (*p* P) x} ⟹ hyperfinite {x. ¬ (*p* P) x} ⟷ hyperfinite (UNIV :: 'a star set)"*
⟨*proof*⟩

lemma *hypnat_hyperfinite_atLeastLessThan [simp]:*
"hyperfinite {M::hypnat..<N}"
⟨*proof*⟩

lemma *hyperfinite_hypnat_set_atLeastAtMost:*
"hyperfinite ({of_nat m .. of_nat n} :: hypnat set)"
⟨*proof*⟩

5.2 Hyperfold function

definition *hyperfold :: "('a star ⇒ 'b star ⇒ 'b star) ⇒ 'b star ⇒ 'a star set => 'b star"* **where**
*"hyperfold f z A = star_n (λn. Finite_Set.fold ((*nf2* f) n) (n_star z n) ((*ns* A) n))"*

hypefold is indeed an internal function

lemma *hyperfold:*
*"hyperfold (*fn2* Fs) (star_n Zs) (*sn* As) = star_n (λn. Finite_Set.fold (Fs n) (Zs n) (As n))"*
⟨*proof*⟩

lemma *hyperfold_empty [simp]: "f ∈ InternalFuns2 ⟹ hyperfold f z {} = z"*
⟨*proof*⟩

definition *hyperfold_image ::*
"('b star ⇒ 'b star ⇒ 'b star) ⇒ ('a star ⇒ 'b star) ⇒ 'b star ⇒ 'a star set ⇒ 'b star"
where *"hyperfold_image f g = hyperfold (λx y. f (g x) y)"*

end

6 Hyperfinite summation

theory *HyperSum*
imports *"HOL-Nonstandard_Analysis.HSeries" Hyperfinite_Set HyperInt*

begin

definition (in *comm_monoid_add*) *hypersum :: "('b star ⇒ 'a star) ⇒ 'b star set ⇒ 'a star"* **where**
*"hypersum f A = star_n (λn. sum ((*nf* f) n) ((*ns* A) n))"*

notation *hypersum ("∑h")*
syntax *"_hypersum" :: "pttrn ⇒ 'b star set ⇒ 'c star ⇒ 'c star" ("(2∑h (_/∈_)./ _)" [0, 51, 10] 10)*
translations *"∑h i∈A. b" == "CONST hypersum (λi. b) A"*

lemma *hypersum:*
*"hypersum (*fn* f) (*sn* A) = star_n (λn. sum (f n) (A n))"*
⟨*proof*⟩

lemma *hypersum_starfun2_n:*
*"hypersum ((*fn2* f) (star_n X)) (*sn* A) = star_n (λn. sum (f n (X n)) (A n))"*
⟨*proof*⟩

6.1 Hypersum properties

lemma *hypersum_empty [simp]:*
assumes *"f ∈ InternalFuns"* **shows** *"hypersum f {} = 0"*
⟨*proof*⟩

lemma *hypersum_starfun_n_empty [simp]: " hypersum (*fn* f) {} = 0"*
⟨*proof*⟩

lemma *setsum_singleton [simp]: "sum f {x} = f x"*
⟨*proof*⟩

lemma *hypersum_singleton [simp]:*
assumes *"f ∈ InternalFuns"*
shows *"hypersum f {a} = f a"*
⟨*proof*⟩

lemma *hypersum_F_neutral_ivl [simp]:*
"hypersum (λn. 0) {m..<n} = 0"
⟨*proof*⟩

lemma *hypersum_head_hSuc:*
assumes *"f ∈ InternalFuns"* **and** *"m ≤ n"*
shows *"hypersum f {m..n} = f m + hypersum f {hSuc m..n}"*
⟨*proof*⟩

lemma *hypersum_head_upt_hSuc:*
assumes *" f ∈ InternalFuns"* **and** *"m < n"*

shows *"hypersum f {m..<n} = f m + hypersum f {hSuc m..<n}"*
⟨*proof*⟩

lemma *hypersum_ub_add_hypernat:*
assumes *"f ∈ InternalFuns"*
and *"(m::hypnat) ≤ n + 1"*
shows *"hypersum f {m..n + p} = hypersum f {m..n} + hypersum f {n + 1..n + p}"*
⟨*proof*⟩

lemma *hypersum_op_ivl_Suc[simp]:*
assumes *"f ∈ InternalFuns"*
shows *"hypersum f {m..<hSuc n} = (if n < m then 0 else hypersum f {m..<n} + f(n))"*
⟨*proof*⟩

lemma *hypersum_insert:*
assumes *"f ∈ InternalFuns"*
and *"A ∈ InternalSets"*
and *"hyperfinite A"*
and *"a ∉ A"*
shows *"hypersum f (insert a A) = f a + hypersum f A"*
⟨*proof*⟩

lemma *hypersum_subset_diff:*
assumes *"f ∈ InternalFuns"*
and *"A ∈ InternalSets"*
and *"B ∈ InternalSets"*
and *"B ⊆ A"*
and *"hyperfinite A"*
shows *"hypersum f A = hypersum f (A - B) + hypersum f B"*
⟨*proof*⟩

lemma *hypersum_diff:*
assumes *"f ∈ InternalFuns"*
and *"A ∈ InternalSets"*
and *"B ∈ InternalSets"*
and *"B ⊆ A"*
and *"hyperfinite A"*
shows *"hypersum f (A - B) = hypersum f A - ((hypersum f B)::('a::ab_group_add star))"*
⟨*proof*⟩

lemma *hypersum_add_hypnat_ivl:*
assumes *"f ∈ InternalFuns"*
and *"m ≤ n"* **and** *"n ≤ p"*
shows *"hypersum f {m..<n} + hypersum f {n..<p} = hypersum f {m..<p::hypnat}"*
⟨*proof*⟩

lemma *hypersum_diff_hypnat_ivl:*
"⟦ (f::hypnat⇒'a::ab_group_add star) ∈ InternalFuns; m ≤ n; n ≤ p
⟧
⟹ hypersum f {m..<p} - hypersum f {m..<n} = hypersum f {n..<p::hypnat}"
⟨*proof*⟩

lemma *InternalFuns2_hypersum:*
assumes *"(λj i. f j i) ∈ InternalFuns2"*
shows *"(λi. hypersum (f i) {m..n}) ∈ InternalFuns"*
⟨*proof*⟩

lemma *hypersum_starfun_atLeastAtMost:*
*"hypersum (*f* f) {M..N} = (*f2* (λm n. sum f {m..n})) M N"*
⟨*proof*⟩

lemma *hypersum_starfun_atLeastLessThan:*
*"hypersum (*f* f) {M..<N} = (*f2* (λm n. sum f {m..<n})) M N"*
⟨*proof*⟩

lemma *hypersum_shift_bounds_cl_hSuc_ivl:*
assumes *"f ∈ InternalFuns"*
shows *"hypersum f {hSuc m..hSuc n} = hypersum (λi. f(hSuc i)){m..n}"*
⟨*proof*⟩

lemma *hypersum_shift_bounds_hSuc_ivl:*
assumes *"f ∈ InternalFuns"*
shows *"hypersum f {hSuc m..<hSuc n} = hypersum (λi. f(hSuc i)){m..<n}"*
⟨*proof*⟩

lemma *lemma_n_starfun2_add_eq:*
*"eventually (λn. ((*nf2* (λx. (+) ((*fn* f) x))) n) = (λx. (+) (f n x)))*
𝒰"
⟨*proof*⟩

lemma (**in** *comm_monoid_add) setsum_def2:*
"sum f A = (if finite A then (Finite_Set.fold (λx y. (+) (f x) y) 0
A) else 0)"
⟨*proof*⟩

lemma *hypersum_def2:*
assumes *"f ∈ InternalFuns"*
shows *"hypersum f A = (if hyperfinite A then hyperfold_image (+) f 0*
A else 0)"
⟨*proof*⟩

6.2 The sumhr function as a hypersum.

lemma *sumhr_hypersum_eq:*
*"sumhr(M,N,f) = hypersum (*f* f) {M..<N}"*
⟨*proof*⟩

lemma *NSsums_hypersum_HNatInfinite_approx_iff:*
*"(f NSsums s) = (∀N∈HNatInfinite. hypersum (*f* f) {0..<N} ≈ of_real s)"*
⟨*proof*⟩

6.3 Hyper-convergence

definition
HyperCauchy :: "(hypnat ⇒ 'a::real_normed_vector star) ⇒ bool" **where**
"HyperCauchy X = (∀M∈HNatInfinite. ∀N∈HNatInfinite. X M ≈ X N)"

definition
HyperSummable :: "(hypnat ⇒ 'a::real_normed_vector star) ⇒ bool" **where**
"HyperSummable F = HyperCauchy (λN. hypersum F {0..<N})"

lemma *NSCauchy_HyperCauchy_starfun_iff:*
*"NSCauchy X = HyperCauchy (*f* X)"*
⟨*proof*⟩

lemma *Cauchy_HyperCauchy_starfun_iff:*
*"Cauchy X = HyperCauchy (*f* X)"*
⟨*proof*⟩

lemma *NSsummable_HyperSummable_starfun_iff:*
*"NSsummable f = HyperSummable (*f* f)"*
⟨*proof*⟩

lemma *starfun_setsum_atLeastLessThan_eq_hypesetsum_fun:*
*"(*f* (λn. sum f {k n..<g n})) = (λN. hypersum (*f* f) {((*f* k) N)..<((*f* g) N)})"*
⟨*proof*⟩

lemma *starfun_setsum_atLeast_zero_hypesetsum_fun:*
*"(*f* (λn. sum f {0..<g n})) = (λN. hypersum (*f* f) {0..< ((*f* g) N)})"*
⟨*proof*⟩

lemma *starfun_setsum_atLeast_zero_hypesetsum:*
*"(*f* (λn. sum f {0..<n})) = (λN. hypersum (*f* f) {0..<N})"*
⟨*proof*⟩

lemma *hypersum_atLeast_zero_starfun:*
*"hypersum (*f* f) {0..<N} = (*f* (λn. sum f {0..<n})) N"*

⟨*proof*⟩

lemma *hypersum_starfun_atLeast0LessThan:*
*"hypersum (*f* f) {0..<N::nat star} = (*f* (λn. ∑i<n. f i)) N"*
⟨*proof*⟩

lemma *hypersum_atLeast_zero_star_n_starfun_n:*
*"hypersum (*fn* f) {0..<star_n X} = star_n((λn. sum (f n) {0..<X n}))"*
⟨*proof*⟩

lemma *hypersum_atLeast_zero_starfun_n:*
*"hypersum (*fn* f) {0..<N} = (*fn* (λn m. sum (f n) {0..<m})) N"*
⟨*proof*⟩

lemma *hypersum_geometric:*
assumes *"x ≠ 1"*
shows *"hypersum (λn. x pow n) {0..<n} = (x pow n - 1) / (x - 1::'a::field star)"*
⟨*proof*⟩

lemma *HyperSummable_geometric:*
assumes *"hnorm (x::'a::{real_normed_field} star) < 1"*
and *"¬hnorm x ≈ 1"*
shows *"HyperSummable (λN. x pow N)"*
⟨*proof*⟩

lemma *summable_convergent_sumr_iff:*
"summable f = convergent (λn. sum f {0..<n})"
⟨*proof*⟩

lemma *HyperSummable_starfun_summable_iff:*
*"(HyperSummable ((*f* f)::hypnat => 'a::banach star)) = (summable f)"*
⟨*proof*⟩

lemma *HyperSummable_def2:*
"HyperSummable f = (∃s. ∀N∈HNatInfinite. hypersum f {0..<N} ≈ s)"
⟨*proof*⟩

lemma *HyperSummable_def3:*
assumes *"f ∈ InternalFuns"*
shows *"HyperSummable f = (∀M∈HNatInfinite.∀N∈HNatInfinite. hypersum f {M..<N} ≈ 0)"*
⟨*proof*⟩

lemma *HyperSummable_starfun_n:*
*"HyperSummable (*fn* f) = (∀M∈HNatInfinite.∀N∈HNatInfinite. hypersum*

*(*fn* f) {M..<N} ≈ 0)"*
⟨*proof*⟩

lemma *atLeastLessThanhSuc_atLeastAtMost:*
"⋀m n. {m..<hSuc n} = {m..n}"
⟨*proof*⟩

lemma *hypersum_op_ivl_Suc2[simp]:*
*"hypersum (*fn* f) {m..<hSuc n} = (if n < m then 0 else hypersum (*fn* f) {m..<n} + (*fn*f) n)"*
⟨*proof*⟩

lemma *hypersum_op_ivl_Suc2':*
*"hypersum (*fn* f) {m..(n::hypnat)} = (if n < m then 0 else hypersum (*fn* f) {m..<n} + (*fn*f) n)"*
⟨*proof*⟩

lemma *HyperSummable_starfun_n_HNatInfinite:*
assumes *"HyperSummable (*fn* f)"*
and *"K∈HNatInfinite"*
shows *"(*fn* f) K ≈ 0"*
⟨*proof*⟩

lemma *HyperSummable_def': "HyperSummable F = HyperCauchy (λN. hypersum F {0..N})"*
⟨*proof*⟩

lemma *HyperSummable_def3':*
assumes *"f ∈ InternalFuns"*
shows *"HyperSummable f = (∀M∈HNatInfinite.∀N∈HNatInfinite. hypersum f {M..N} ≈ 0)"*
⟨*proof*⟩

lemma *HyperSummable_def4:*
*"HyperSummable (*fn* f) = (∃s. ∀N∈HNatInfinite. hypersum (*fn* f) {0..N} ≈ s)"*
⟨*proof*⟩

lemma *HyperSummable_shift_hSuc:*
assumes *"f ∈ InternalFuns"*
and *"HyperSummable f"*
shows *"HyperSummable (λn. f (hSuc n))"*
⟨*proof*⟩

lemma *InternalFuns_hypersum_starfun_n_Interval:*
*"(λN. hypersum (*fn* F) {M..<N}) ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_hypersum:*
assumes *"f ∈ InternalFuns"*
shows *"(λn. hypersum f {m..<n}) ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_hypersum_starfun_n_Interval2:*
*"(λN. hypersum (*fn* F) {M..N}) ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_hypersum2:*
"f ∈ InternalFuns ⟹ (λn. hypersum f {m..n}) ∈ InternalFuns"
⟨*proof*⟩

lemma *NSsummable_NSCauchy_setsum:*
"NSsummable f = NSCauchy (λn. sum f {0..<n})"
⟨*proof*⟩

lemma *Hypersum_Comparison_Theorem_Nats:*
assumes *"f ∈ InternalFuns" "g ∈ InternalFuns"*
"HyperSummable f" "HyperSummable g"
"∀n∈Nats. f n ≈ g n"
"N ∈ Nats"
shows *"hypersum f {0..<N} ≈ hypersum g {0..<N}"*
⟨*proof*⟩

lemma *Hypersum_Comparison_Theorem_HNatInfinite:*
assumes *int_f: "f ∈ InternalFuns"*
and *int_g: "(g::nat star ⇒ 'a::{real_normed_div_algebra, semiring_1_cancel} star) ∈ InternalFuns"*
and *hyperSums: "HyperSummable f" "HyperSummable g"*
and *inf_close_fg: "∀n∈Nats. f n ≈ g n"*
and *inf_N: "N ∈ HNatInfinite"*
shows *"hypersum f {0..<N} ≈ hypersum g {0..<N}"*
⟨*proof*⟩

Full version of the Summation Comparison Theorem

lemma *Hypersum_Comparison_Theorem:*
"⟦ f ∈ InternalFuns;
(g::nat star ⇒ 'a::{real_normed_div_algebra, semiring_1_cancel} star) ∈ InternalFuns;
HyperSummable f; HyperSummable g; ∀n∈Nats. f n ≈ g n ⟧
⟹ hypersum f {0..<N} ≈ hypersum g {0..<N}"
⟨*proof*⟩

lemma *Hypersum_Comparison_Theorem':*
"⟦ f ∈ InternalFuns;
(g::nat star ⇒ 'a::{real_normed_div_algebra, semiring_1_cancel}

```
star) ∈ InternalFuns;
     HyperSummable f; HyperSummable g; ∀n∈Nats. f n ≈ g n ⟧
       ⟹ hypersum f {0..N} ≈ hypersum g {0..N}"
```
⟨*proof*⟩

lemma *HyperSummable_hypersum_approx:*
```
  "⟦ HyperSummable f; M ∈ HNatInfinite; N ∈ HNatInfinite ⟧
   ⟹ hypersum f {0..M} ≈ hypersum f {0..N}"
```
⟨*proof*⟩

lemma *hnorm_hypersum:*
```
  "⟦ f ∈ InternalFuns; A ∈ InternalSets ⟧ ⟹ hnorm (hypersum f A) ≤
hypersum (λi. hnorm (f i)) A"
```
⟨*proof*⟩

lemma *hypersum_mono':*
```
  assumes internal_f: "f ∈ InternalFuns"
      and internal_g: "g ∈ InternalFuns"
      and internal_setK: "K ∈ InternalSets"
      and fg_mono: "(∀i. i ∈ K ⟶ f i ≤ ((g i)::('b::ordered_comm_monoid_add
star)))"
    shows "hypersum f K ≤ hypersum g K"
```
⟨*proof*⟩

lemma *hypersum_mono:*
```
   "⟦ f ∈ InternalFuns; g ∈ InternalFuns; K ∈ InternalSets;
      (⋀i. i ∈ K ⟹ f i ≤ ((g i)::('b::ordered_comm_monoid_add star)))
⟧
    ⟹ hypersum f K ≤ hypersum g K"
```
⟨*proof*⟩

lemma *hypersum_nonneg:*
```
  assumes "f ∈ InternalFuns" "A ∈ InternalSets"
          "∀x∈A. (0::'a::ordered_comm_monoid_add star) ≤ f x"
  shows "0 ≤ hypersum f A"
```
⟨*proof*⟩

lemma *abs_hypersum_abs [simp]:*
```
  "⟦ (f :: 'a star ⇒ ('b::ordered_ab_group_add_abs) star) ∈ InternalFuns;
A ∈ InternalSets ⟧
       ⟹ abs(hypersum (λi. abs(f i)) A) = hypersum (λi. abs(f i)) A"
```
⟨*proof*⟩

lemma *InternalFuns_hnorm_starfun_n [simp]:*
```
  "(λi. hnorm ((*fn* F) i)) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *InternalFuns_hnorm:*

```
  "f ∈ InternalFuns ⟹ (λn. hnorm (f n)) ∈ InternalFuns"
```
⟨*proof*⟩

lemma *HyperSummable_Cauchy:*
```
   "HyperSummable (*fn* F) ⟹ (∀e∈Reals. e>0 ⟶ (∃N. ∀m≥N. ∀n. hnorm
(hypersum (*fn* F) {m..<n}) < e))"
```
⟨*proof*⟩

lemma *Cauchy_HyperSummable:*
```
   "(∀e∈Reals. e>0 ⟶ (∃N∈Nats. ∀m≥N. ∀n. hnorm (hypersum (*fn* F)
{m..<n}) < e)) ⟹ HyperSummable (*fn* F)"
```
⟨*proof*⟩

lemma *Cauchy_HyperSummable':*
```
   "(∀e∈Reals. e>0 ⟶ (∃N∈Nats. ∀m≥N. ∀n≥N. hnorm (hypersum (*fn*
F) {m..<n}) < e)) ⟹
    HyperSummable (*fn* F)"
```
⟨*proof*⟩

lemma *HyperSummable_Cauchy':*
```
   assumes H: "HyperSummable (*fn* F)"
   shows "(∀e∈Reals. e>0 ⟶ (∃N. ∀m≥N. ∀n≥N. hnorm (hypersum (*fn*
F) {m..<n}) < e))"
```
⟨*proof*⟩

lemma *HyperSummable_comparison_test:*
```
  assumes "f ∈ InternalFuns" "g ∈ InternalFuns"
          "∃k∈Nats. ∀n. k ≤ n ⟶ hnorm ((f::hypnat => 'a::banach star)
n) ≤ g n"
          "HyperSummable g"
  shows "HyperSummable f"
```
⟨*proof*⟩

lemma *HyperSummable_hypreal_comparison_test:*
```
     "⟦ (f::hypnat ⇒ hypreal) ∈ InternalFuns;
        g ∈ InternalFuns;
        ∀n. f n ≥ 0;
        ∃k∈Nats. ∀n. k ≤ n ⟶ f n ≤ g n;
        HyperSummable g ⟧ ⟹ HyperSummable f"
```
⟨*proof*⟩

lemma *HyperSummable_comparison_test_norm:*
```
     "f ∈ InternalFuns
      ⟹ HyperSummable (λn. hnorm ((f::hypnat => 'a::banach star) n))
⟹  HyperSummable f"
```
⟨*proof*⟩

lemma *hypersum_atLeastLessThan_starfun_n:*
```
  "hypersum (*fn* f) {M..<N} =  (*fn2* (λi m n. sum (f i) {m..<n})) M
```

```
N"
```
⟨*proof*⟩

lemma *hypersum_minus:*
```
  "⟦ f ∈ InternalFuns; A ∈ InternalSets ⟧
   ⟹ hypersum (λx. - (f x)::'a::ab_group_add star) A = - hypersum f
A"
```
⟨*proof*⟩

lemma *HyperSummable_minus:*
```
  "⟦ f ∈ InternalFuns; HyperSummable f ⟧ ⟹  HyperSummable (λn. - f n)"
```
⟨*proof*⟩

lemma *HyperSummable_negf_iff:*
```
  "f ∈ InternalFuns ⟹ (HyperSummable (λn. - f n)) =  (HyperSummable
f)"
```
⟨*proof*⟩

lemma *hypersum_left_distrib:*
  assumes *"f ∈ InternalFuns" "A ∈ InternalSets"*
  shows *"hypersum f A * (r::'a::semiring_0 star) = hypersum (λn. f n
* r) A"*
⟨*proof*⟩

lemma *hypersum_right_distrib:*
  assumes *"f ∈ InternalFuns" "A ∈ InternalSets"*
  shows *"(r::'a::semiring_0 star) * hypersum f A = hypersum (λn. r *
f n) A"*
⟨*proof*⟩

lemma *hypersum_product:*
```
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns; A ∈ InternalSets;  B ∈ InternalSets
⟧
   ⟹ hypersum f A * hypersum g B =
       hypersum (λi. hypersum (λj. f i * (g j::'a::semiring_0 star)) B)
A"
```
  ⟨*proof*⟩

lemma *starfun2_to_starfun_n_lemma:*
```
  "(λn. (*f2* f) (star_n X) ((*fn* F) n)) = *fn* (λn m. f (X n) (F n m))"
```
⟨*proof*⟩

lemma *starfun2_to_starfun_n_lemma2:*
```
  "(λn. (*f2* f) ((*fn* F) n) (star_n X)) = *fn* (λn m. f (F n m) (X n))"
```
⟨*proof*⟩

lemma *hypersum_divide_distrib:*
  assumes *"f ∈ InternalFuns"*
```
         "A ∈ InternalSets "
```

shows *"hypersum f A / (r::'a::field star) = hypersum (λn. f n / r) A"*
⟨*proof*⟩

lemma *HyperSummable_HFinite_mult_left:*
*"⟦ f ∈ InternalFuns; (c::'a::{real_normed_algebra} star) ∈ HFinite; HyperSummable f ⟧ ⟹ HyperSummable (λn. c * f n)"*
⟨*proof*⟩

lemma *HyperSummable_divide:*
assumes *"f ∈ InternalFuns"*
"HyperSummable f"
"r ∉ Infinitesimal"
shows *"HyperSummable (λn. f n / (r::'a::{real_normed_field} star))"*
⟨*proof*⟩

end

7 The floor and ceiling functions extended to hypernumbers

theory *HyperArchimedean*
imports *HyperInt*
begin

7.1 The nonstandard floor function

definition
hfloor :: "'a::floor_ceiling star ⇒ hypint" (<(<open_block notation=<mixfix floor>>⌊_⌋)>) **where**
*[transfer_unfold]: "hfloor x = (*f* floor) x"*

lemma *hfloor:*
"hfloor (star_n X) = star_n (λn. floor (X n))"
⟨*proof*⟩

lemma *hfloor_correct: "⋀x. of_hypint (hfloor x) ≤ x ∧ x < of_hypint (hfloor x + 1)"*
⟨*proof*⟩

lemma *hfloor_unique: "⋀x z. ⟦ of_hypint z ≤ x; x < of_hypint z + 1 ⟧ ⟹ hfloor x = z"*
⟨*proof*⟩

lemma *hfloor_eq_zero [simp]: "⟦ 0 ≤ x; x < 1⟧ ⟹ hfloor x = 0"*
⟨*proof*⟩

lemma *of_hypint_floor_le: "of_hypint (hfloor x) ≤ x"*
⟨*proof*⟩

lemma *le_hfloor_iff: "⋀x z. z ≤ hfloor x ⟷ of_hypint z ≤ x"*
⟨*proof*⟩

lemma *hfloor_less_iff: "⋀x z. hfloor x < z ⟷ x < of_hypint z"*
⟨*proof*⟩

lemma *less_hfloor_iff: "⋀x z. z < hfloor x ⟷ of_hypint z + 1 ≤ x"*
⟨*proof*⟩

lemma *hfloor_le_iff: "⋀x z. hfloor x ≤ z ⟷ x < of_hypint z + 1"*
⟨*proof*⟩

lemma *hfloor_mono: "⋀x y. x ≤ y ⟹ hfloor x ≤ hfloor y"*
⟨*proof*⟩

lemma *hfloor_less_cancel: "⋀x y. hfloor x < hfloor y ⟹ x < y"*
⟨*proof*⟩

lemma *hfloor_of_hypint [simp]: "⋀z. hfloor (of_hypint z) = z"*
⟨*proof*⟩

lemma *hfloor_of_int [simp]: "hfloor (of_int z) = hypint_of_int z"*
⟨*proof*⟩

lemma *hfloor_of_hypnat [simp]: "⋀n. hfloor (of_hypnat n) = of_hypnat n"*
⟨*proof*⟩

lemma *hfloor_of_nat [simp]: "hfloor (of_nat n) = of_nat n"*
⟨*proof*⟩

Floor with numerals

lemma *hfloor_zero [simp]: "hfloor 0 = 0"*
⟨*proof*⟩

lemma *hfloor_one [simp]: "hfloor 1 = 1"*
⟨*proof*⟩

lemma *hfloor_star_of [simp]: "hfloor (star_of x) = star_of x"*
⟨*proof*⟩

lemma *star_of_hfloor [simp]: "star_of(floor x) = hfloor (star_of x)"*
⟨*proof*⟩

lemma *hfloor_number_of [simp]: "hfloor (numeral v :: hypreal) = (numeral*

```
v :: hypint)"
```
⟨*proof*⟩

lemma *zero_le_hfloor [simp]: "0 ≤ hfloor x ⟷ 0 ≤ x"*
⟨*proof*⟩

lemma *one_le_hfloor [simp]: "1 ≤ hfloor x ⟷ 1 ≤ x"*
⟨*proof*⟩

lemma *numeral_le_hfloor [simp]: "⋀x. numeral v ≤ hfloor x ⟷ numeral v ≤ x"*
⟨*proof*⟩

lemma *zero_less_hfloor [simp]: "0 < hfloor x ⟷ 1 ≤ x"*
⟨*proof*⟩

lemma *one_less_hfloor [simp]: "1 < hfloor x ⟷ 2 ≤ x"*
⟨*proof*⟩

lemma *numeral_less_hfloor [simp]:*
"⋀x. numeral v < hfloor x ⟷ numeral v + 1 ≤ x"
⟨*proof*⟩

lemma *hfloor_le_zero [simp]: "hfloor x ≤ 0 ⟷ x < 1"*
⟨*proof*⟩

lemma *hfloor_le_one [simp]: "hfloor x ≤ 1 ⟷ x < 2"*
⟨*proof*⟩

lemma *hfloor_le_numeral [simp]:*
"⋀x. hfloor x ≤ numeral v ⟷ x < numeral v + 1"
⟨*proof*⟩

lemma *hfloor_less_zero [simp]: "hfloor x < 0 ⟷ x < 0"*
⟨*proof*⟩

lemma *hfloor_less_one [simp]: "hfloor x < 1 ⟷ x < 1"*
⟨*proof*⟩

lemma *hfloor_less_numeral [simp]:*
"⋀x. hfloor x < numeral v ⟷ x < numeral v"
⟨*proof*⟩

lemma *hfloor_add_of_hypint [simp]: "hfloor (x + of_hypint z) = hfloor x + z"*
⟨*proof*⟩

lemma *hfloor_add_of_int [simp]: "⋀x. hfloor (x + of_int z) = hfloor x + of_int z"*

⟨*proof*⟩

lemma *hfloor_add_numeral [simp]:*
"hfloor (x + numeral v) = hfloor x + numeral v"
⟨*proof*⟩

lemma *hfloor_add_one [simp]: "hfloor (x + 1) = hfloor x + 1"*
⟨*proof*⟩

lemma *hfloor_diff_of_hypint [simp]: "hfloor (x - of_hypint z) = hfloor x - z"*
⟨*proof*⟩

lemma *hfloor_diff_of_int [simp]: "hfloor (x - of_int z) = hfloor x - of_int z"*
⟨*proof*⟩

lemma *hfloor_diff_numeral [simp]:*
"hfloor (x - numeral v) = hfloor x - numeral v"
⟨*proof*⟩

lemma *hfloor_diff_one [simp]: "hfloor (x - 1) = hfloor x - 1"*
⟨*proof*⟩

lemma *hfloor_add_one_gt_zero:*
"⋀y. 0 < y ⟹ 0 < hfloor y + 1"
⟨*proof*⟩

7.2 The nonstandard ceiling function

definition
hceiling :: "'a::floor_ceiling star => hypint" (‹(‹open_block notation=‹mixfix ceiling››⌈_⌉)›) **where**
[transfer_unfold]: "hceiling x = - hfloor (- x)"

lemma *hceiling_correct: "of_hypint (hceiling x) - 1 < x ∧ x ≤ of_hypint (hceiling x)"*
⟨*proof*⟩

lemma *hceiling_unique: "⟦of_hypint z - 1 < x; x ≤ of_hypint z⟧ ⟹ hceiling x = z"*
⟨*proof*⟩

lemma *le_of_int_hceiling: "x ≤ of_hypint (hceiling x)"*
⟨*proof*⟩

lemma *hceiling_le_iff: "hceiling x ≤ z ⟷ x ≤ of_hypint z"*
⟨*proof*⟩

lemma *less_hceiling_iff: "z < hceiling x ⟷ of_hypint z < x"*
⟨*proof*⟩

lemma *hceiling_less_iff: "hceiling x < z ⟷ x ≤ of_hypint z - 1"*
⟨*proof*⟩

lemma *le_hceiling_iff: "z ≤ hceiling x ⟷ of_hypint z - 1 < x"*
⟨*proof*⟩

lemma *hceiling_mono: "x ≥ y ⟹ hceiling x ≥ hceiling y"*
⟨*proof*⟩

lemma *hceiling_less_cancel: "hceiling x < hceiling y ⟹ x < y"*
⟨*proof*⟩

lemma *hceiling_of_hypint [simp]: "hceiling (of_hypint z) = z"*
⟨*proof*⟩

lemma *hceiling_of_int [simp]: "hceiling (of_int z) = of_int z"*
⟨*proof*⟩

lemma *hceiling_of_hypnat [simp]: "hceiling (of_hypnat n) = of_hypnat n"*
⟨*proof*⟩

lemma *hceiling_of_nat [simp]: "hceiling (of_nat n) = of_nat n"*
⟨*proof*⟩

Ceiling with numerals

lemma *hceiling_zero [simp]: "hceiling 0 = 0"*
⟨*proof*⟩

lemma *hceiling_one [simp]: "hceiling 1 = 1"*
⟨*proof*⟩

lemma *hceiling_numeral [simp]: "hceiling (numeral v) = numeral v"*
⟨*proof*⟩

lemma *hceiling_le_zero [simp]: "hceiling x ≤ 0 ⟷ x ≤ 0"*
⟨*proof*⟩

lemma *hceiling_le_one [simp]: "hceiling x ≤ 1 ⟷ x ≤ 1"*
⟨*proof*⟩

lemma *hceiling_le_numeral [simp]:*
"⋀x. hceiling x ≤ numeral v ⟷ x ≤ numeral v"
⟨*proof*⟩

lemma *hceiling_less_zero [simp]: "hceiling x < 0 ⟷ x ≤ -1"*

⟨*proof*⟩

lemma *hceiling_less_one [simp]: "hceiling x < 1 ⟷ x ≤ 0"*
⟨*proof*⟩

lemma *hceiling_less_numeral [simp]:*
"⋀x. hceiling x < numeral v ⟷ x ≤ numeral v - 1"
⟨*proof*⟩

lemma *zero_le_hceiling [simp]: "0 ≤ hceiling x ⟷ -1 < x"*
⟨*proof*⟩

lemma *one_le_hceiling [simp]: "1 ≤ hceiling x ⟷ 0 < x"*
⟨*proof*⟩

lemma *numeral_le_hceiling [simp]:*
"⋀x. numeral v ≤ hceiling x⟷ numeral v - 1 < x"
⟨*proof*⟩

lemma *zero_less_hceiling [simp]: "0 < hceiling x ⟷ 0 < x"*
⟨*proof*⟩

lemma *one_less_hceiling [simp]: "1 < hceiling x ⟷ 1 < x"*
⟨*proof*⟩

lemma *numeral_less_hceiling [simp]:*
"⋀x. numeral v < hceiling x ⟷ numeral v < x"
⟨*proof*⟩

lemma *hceiling_add_of_hypint [simp]: "hceiling (x + of_hypint z) = hceiling x + z"*
⟨*proof*⟩

lemma *hceiling_add_of_int [simp]: "hceiling (x + of_int z) = hceiling x + of_int z"*
⟨*proof*⟩

lemma *hceiling_add_numeral [simp]:*
"hceiling (x + numeral v) = hceiling x + numeral v"
⟨*proof*⟩

lemma *hceiling_add_one [simp]: "hceiling (x + 1) = hceiling x + 1"*
⟨*proof*⟩

lemma *hceiling_diff_of_hypint [simp]: "hceiling (x - of_hypint z) = hceiling x - z"*
⟨*proof*⟩

lemma *hceiling_diff_of_int [simp]: "hceiling (x - of_int z) = hceiling*

```
x - of_int z"
```
⟨*proof*⟩

lemma *hceiling_diff_numeral [simp]:*
```
"hceiling (x - numeral v) = hceiling x - numeral v"
```
⟨*proof*⟩

lemma *hceiling_diff_one [simp]: "hceiling (x - 1) = hceiling x - 1"*
⟨*proof*⟩

lemma *hfloor_minus: "hfloor (- x) = - hceiling x"*
⟨*proof*⟩

lemma *hceiling_minus: "hceiling (- x) = - hfloor x"*
⟨*proof*⟩

7.3 The nonstandard fractional part

definition
```
hpart :: "'a::floor_ceiling star => 'a star" where
[transfer_unfold]: "hpart x = x - of_hypint (hfloor x)"
```

notation
```
hpart  ("{_}")
```

notation (*HTML* **output**)
```
hpart  ("{_}")
```

lemma *hpart_zero [simp]: "hpart(0) = 0"*
⟨*proof*⟩

lemma *hpart_ge_zero [simp]: "0 ≤ hpart x"*
⟨*proof*⟩

lemma *hpart_less_one [simp]: "hpart x < 1"*
⟨*proof*⟩

lemma *hpart_le_one [simp]: "hpart x ≤ 1"*
⟨*proof*⟩

lemma *hypreal_eq_hfloor_hpart_add:*
```
      "x = of_hypint (hfloor x) + hpart x"
```
⟨*proof*⟩

lemma *hfloor_eq_self_diff_hpart: "of_hypint (hfloor x) = x - hpart x"*
⟨*proof*⟩

lemma *HFinite_hpart [simp]: "⋀x. hpart (x::hypreal) ∈ HFinite"*
⟨*proof*⟩

lemma *hpart_not_less_one [simp]: "¬ 1 < ⦃x⦄"*
⟨*proof*⟩

lemma *hpart_mult_less_cancel [simp]: "(x * ⦃y⦄ < x) = (0 < x)"*
⟨*proof*⟩

lemma *hfloor_of_hypint_divide [simp]:*
"⋀x y. hfloor ((of_hypint x :: hypreal)/ of_hypint y) = (x div y)"
⟨*proof*⟩

lemma *hfloor_of_hypint_of_hypnat_divide [simp]:*
"⋀x y. hfloor ((of_hypint x :: hypreal)/ of_hypnat y) = (x div (of_hypnat y))"
⟨*proof*⟩

lemma *floor_real_of_nat_real_of_int_divide [simp]:*
"floor (real x / real y) = int x div y"
⟨*proof*⟩

lemma *hfloor_of_hypnat_of_hypint_divide [simp]:*
"⋀x y. hfloor ((of_hypnat x:: hypreal) / of_hypint y) = (of_hypnat x) div y"
⟨*proof*⟩

lemma *of_hypnat_hypnat_of_hypint [simp]:*
"0 ≤ x ⟹ of_hypnat(hypnat(hfloor x)) = of_hypint (hfloor x)"
⟨*proof*⟩

7.4 Miscellaneous properties

lemma *HFinite_hypnat_hfloor_Nat:*
assumes *"(y::hypreal) ∈ HFinite"*
shows *"hypnat (hfloor y) ∈ ℕ"*
⟨*proof*⟩

lemma *HFinite_between_Nats:*
assumes *"(x::hypreal) ∈ HFinite"*
and *"x > 0"*
shows *"∃n∈ℕ. n ≤ x ∧ x < n + 1"*
⟨*proof*⟩

lemma *hfloor_eq_self_diff_hpart2:*
"⋀x. 0 ≤ x ⟹ of_hypnat (hypnat (hfloor x)) = x - hpart x"
⟨*proof*⟩

lemma *hypreal_HInfinite_hfloor:*

"(x::hypreal) ∈ HInfinite ⟹ hypreal_of_hypint (hfloor x) ∈ HInfinite"
⟨proof⟩

lemma *hypreal_HInfinite_hfloor_cancel:*
"hypreal_of_hypint (hfloor x) ∈ HInfinite ⟹ (x::hypreal) ∈ HInfinite"
⟨proof⟩

lemma *hypreal_HNatInfinite_hfloor:*
"⟦ (x::hypreal) ∈ HInfinite; x > 0 ⟧ ⟹ hypnat (hfloor x) ∈ HNatInfinite"
⟨proof⟩

lemma *hypreal_HNatInfinite_hfloor_cancel:*
"hypnat (hfloor x) ∈ HNatInfinite ⟹ (x::hypreal) ∈ HInfinite"
⟨proof⟩

lemma *hypreal_HNatInfinite_hfloor_inverse_Infinitesimal:*
"⟦ (e::hypreal) ∈ Infinitesimal; e > 0 ⟧ ⟹ hypnat (hfloor (1 / e))
∈ HNatInfinite"
⟨proof⟩

lemma *hypreal_Infinitesimal_HNatInfinite_hfloor_inverse_iff:*
assumes *pos:* "e > 0"
shows "((e::hypreal) ∈ Infinitesimal) = (hypnat (hfloor (1 / e))
∈ HNatInfinite)" (**is** "?L = ?R")
⟨proof⟩

lemma *hypreal_Infinitesimal_hfloor_inverse_HNatInfinite:*
"hypnat (hfloor (1 / e)) ∈ HNatInfinite ⟹ (e::hypreal) ∈ Infinitesimal"
⟨proof⟩

lemma *inverse_hfloor_inverse_approx:*
"e ∈ Infinitesimal ⟹ inverse (hypreal_of_hypint(hfloor(1 / e))) ≈
e"
⟨proof⟩

lemma *Infinitesimal_hfloor_divide_mult:*
assumes *infmal_x:* "x ∈ Infinitesimal"
and *not_zero_x:* " x ≠ (0::hypreal)"
shows "x * of_hypint(hfloor (z/x)) ≈ z"
⟨proof⟩

lemma *Infinitesimal_hfloor_inverse_mult_self:*
assumes *infmal_x:* "x ∈ Infinitesimal"
and *not_zero_x:* " x ≠ (0::hypreal)"
shows "x * of_hypint(hfloor (inverse x)) ≈ 1"
⟨proof⟩

lemma *Infinitesimal_hfloor_inverse_mult_self_pos:*

"⟦ e ∈ Infinitesimal; e > (0::hypreal) ⟧ ⟹ e * of_hypint(hfloor(1/e)) ≈ 1"
⟨*proof*⟩

end

8 The hyperfactorial and hyperbinomial coefficient functions

theory *HyperBinomial*
imports *FallFactorial HyperSum HyperArchimedean Internal*
begin

8.1 The hyperbinomial coefficients

definition
  hchoose :: "nat star ⇒ nat star ⇒ nat star" (**infixl** *"hchoose" 65*)
where
  hyperbinomial_def [transfer_unfold]: *"(hchoose) = *f2* (choose)"*

lemma *star_choose: "(star_n (N::nat⇒nat)) hchoose (star_n K) = star_n (λn. N n choose K n)"*
⟨*proof*⟩

lemma *star_of_choose [simp]: "star_of (n choose k) = (star_of n) hchoose (star_of k)"*
⟨*proof*⟩

lemma *star_choose_zero_hypnat [simp]: "⋀n. (n::hypnat) hchoose 0 = 1"*
⟨*proof*⟩

lemma *zero_star_choose_hypnat [simp]: "⋀n k. n < (k::hypnat) ⟹ n hchoose k = 0"*
⟨*proof*⟩

lemma *star_choose_reduce_hypnat:*
  "⋀n k. ⟦0 < (n::hypnat); 0 < k⟧
  ⟹ n hchoose k = n - 1 hchoose k + (n - 1 hchoose (k - 1))"
⟨*proof*⟩

lemma *star_choose_plus_one_hypnat: "⋀n k. ((n::hypnat) + 1) hchoose (k + 1) =*
  (n hchoose (k + 1)) + (n hchoose k)"
⟨*proof*⟩

lemma *star_choose_hSuc_hypnat: "⋀n k. (hSuc n) hchoose (hSuc k) =*
  (n hchoose (hSuc k)) + (n hchoose k)"
⟨*proof*⟩

lemma *star_choose_self_hypnat [simp]: "⋀n. ((n::hypnat) hchoose n) = 1"*
⟨*proof*⟩

lemma *star_choose_one_hypnat [simp]: "⋀n. (n::hypnat) hchoose 1 = n"*
⟨*proof*⟩

lemma *star_plus_one_choose_self_hypnat [simp]: "⋀n. (n::hypnat) + 1 hchoose n = n + 1"*
⟨*proof*⟩

lemma *hSuc_choose_self_hypnat [simp]: "⋀n. (hSuc n) hchoose n = hSuc n"*
⟨*proof*⟩

lemma *choose_pos_nat [rule_format]: "⋀k n. k <= (n::hypnat) ⟹ (n hchoose k) > 0"*
⟨*proof*⟩

8.2 The hyperfactorial function

definition
hfact_def [transfer_unfold]: *"hfact ≡ *f* fact"*

lemma *star_choose_altdef_hypnat:*
*"⋀n k. (k::hypnat) ≤ n ⟹ n hchoose k = hfact n div (hfact k * hfact (n - k))"*
⟨*proof*⟩

lemma *star_choose_dvd_hypnat: "⋀n k. (k::hypnat) ≤ n ⟹ hfact k * hfact (n - k) dvd hfact n"*
⟨*proof*⟩

lemma *hfact_of_nat: "hfact (hypnat_of_nat n) = of_nat (fact n)"*
⟨*proof*⟩

lemma *HFinite_hfact_of_nat [simp]:*
"of_hypnat (hfact (hypnat_of_nat n)) ∈ HFinite"
⟨*proof*⟩

lemma *hfact_nat_in_Nats: "n ∈ ℕ ⟹ hfact (n::hypnat)∈ ℕ "*
⟨*proof*⟩

lemma *star_choose_le_hyperpow:*
"⋀r n. r ≤ n ⟹ n hchoose r ≤ n pow r"
⟨*proof*⟩

The binomial theorem extended to hyperreal and hypernaturals, characterized via hyperfinite sums

lemma *Iset_interval:*
"Iset (star_n (λn. {0..X n})) = {0..star_n X}"
⟨*proof*⟩

lemma *n_star_interval_ultra: "eventually (λn. (*ns* {0..star_n X}) n = {0..X n}) 𝒰"*
⟨*proof*⟩

lemma *lemma_starfun_n_binomial:*
*"*fn* (λn k. of_nat (Xb n choose k) * X n ^ k * Xa n ^ (Xb n - k)) =*

*(λK. of_hypnat ((*f2* (choose)) (star_n Xb) K) * (*f2* (^)) (star_n X) K * (*f2* (^)) (star_n Xa) (star_n Xb - K))"*
⟨*proof*⟩

8.3 The hyperbinomial theorem

lemma *hyperbinomial_ring:*
*"(a + b::'a::{comm_semiring_1,semiring_1_cancel} star) pow N = hypersum (λK. of_hypnat (N hchoose K) * a pow K * b pow (N - K)) {0..N}"*
⟨*proof*⟩

lemma *hyperbinomial_simple:*
*"(1 + a::'a::{comm_semiring_1,semiring_1_cancel} star) pow N = hypersum (λi. of_hypnat (N hchoose i) * a pow i) {0..N}"*
⟨*proof*⟩

lemma *lemma_InternalFuns_hyperbinomial:*
*"hypreal_of_hypnat (star_n X hchoose star_n Xc) * star_n Xa pow star_n Xc * star_n Xb pow (star_n X - star_n Xc) = star_n (λn. real (X n choose Xc n) * Xa n ^ Xc n * Xb n ^ (X n - Xc n))"*
⟨*proof*⟩

lemma *InternalFuns_hyperbinomial [simp]:*
*"(λK. hypreal_of_hypnat (N hchoose K) * a pow K * b pow (N - K)) ∈ InternalFuns"*
⟨*proof*⟩

8.4 Nonstandard falling factorial

definition
hfallfactpow :: "'a::comm_ring_1 star ⇒ hypnat ⇒ 'a star" **where**
*[transfer_unfold]: "hfallfactpow a n = (*f2* fallfactpow) a n"*

lemma *hfallfactpow_0[simp]: "⋀a. hfallfactpow a 0 = 1"*
⟨*proof*⟩

lemma *hfallfactpow_1[simp]: "⋀a. hfallfactpow a 1 = a"* ⟨*proof*⟩
lemma *hfallfactpow_hSuc0[simp]: "⋀a. hfallfactpow a (hSuc 0) = a"* ⟨*proof*⟩

lemma *hfallfactpow_hSuc: "⋀a n. hfallfactpow a (hSuc n) = hfallfactpow a n * (a - of_hypnat n)"*
⟨*proof*⟩

lemma *hfallfactpow_hSuc': "hfallfactpow a (n + 1) = hfallfactpow a n * (a - of_hypnat n)"*
⟨*proof*⟩

lemma *hfallfactpow_rec: "⋀a n. hfallfactpow a (hSuc n) = a * hfallfactpow (a - 1) n"*
⟨*proof*⟩

lemma *hfallfactpow_rec': "⋀a n. hfallfactpow a (n + 1) = a * hfallfactpow (a - 1) n"*
⟨*proof*⟩

lemma *hfallfactpow_base_zero[simp]: "⋀n. hfallfactpow 0 (hSuc n) = 0"*
⟨*proof*⟩

lemma *hfallfactpow_base_zero'[simp]: "⋀n. hfallfactpow 0 (n + 1) = 0"*
⟨*proof*⟩

lemma *hfallfactpow_fact: "⋀n. of_hypnat(hfact n) = hfallfactpow (of_hypnat n) n"*
⟨*proof*⟩

lemma *hfallfactpow_of_nat_eq_0_lemma:*
"⋀k n. k > n ⟹ hfallfactpow (of_hypnat n :: 'a:: idom star) k = 0"
⟨*proof*⟩

lemma *hyperbinomial_hfallfactpow:*
"⋀k n. of_hypnat (n hchoose k) =
(hfallfactpow (of_hypnat n) k ::'a::{field_char_0} star)/hfact k"
⟨*proof*⟩

lemma *hyperbinomial3:*
"(a + b::'a::{field_char_0} star) pow n =
*hypersum (λk. hfallfactpow (of_hypnat n) k / hfact k * a pow k * b pow (n - k)) {0..n}"*
⟨*proof*⟩

lemma *hfallfactpow_le_power_self [simp]:*
"⋀k. hfallfactpow (of_hypnat k ::'a::{linordered_idom} star) k ≤ of_hypnat k pow k"
⟨*proof*⟩

lemma *hfallfactpow_le_hyperpow:*
"⋀k j. of_hypnat k ≤ j ⟹ hfallfactpow (j::'a::{linordered_idom} star) k ≤ j pow k"
⟨*proof*⟩

lemma *hfallfactpow_ge_zero [simp]:*
"⋀k n. hfallfactpow (of_hypnat n) k ≥ (0::'a::linordered_idom star)"
⟨*proof*⟩

lemma *fact_Suc_ge_two_pow: "fact (n + 1) ≥ (2::nat)^n"*
⟨*proof*⟩

lemma *hfact_hSuc_ge_two_pow:*
"⋀n. (hfact (n + 1) :: hypreal) ≥ 2 pow n"
⟨*proof*⟩

lemma *HInfinite_diff_of_nat_divide_approx_one:*
assumes *"x ∈ HInfinite"*
shows *"(x - of_nat k)/x ≈ (1::'a::real_normed_field star)"*
⟨*proof*⟩

lemma *HInfinite_hfallfactpow_divide_hrealpow:*
assumes *"j ∈ HInfinite"*
shows *"hfallfactpow j (of_nat k)/(j ^ k) ≈ (1::'a::real_normed_field star)"*
⟨*proof*⟩

lemma *HInfinite_hfallfactpow_divide_hyperpow_of_nat:*
"j ∈ HInfinite ⟹ hfallfactpow j (of_nat k)/(j pow of_nat k) ≈ (1::'a::real_normed_field star)"
⟨*proof*⟩

lemma *HInfinite_hfallfactpow_divide_hyperpow_Nats:*
"⟦ k∈ℕ; j∈HInfinite ⟧ ⟹ hfallfactpow j k/(j pow k) ≈ (1::'a::real_normed_field star)"
⟨*proof*⟩

This is the one that we want to deal with Euler's claim that $(j-1)(j-2)...(j-k)/j^k = 1$!

lemma *HNatInfinite_hfallfactpow_divide_hyperpow_of_nat:*
"j ∈ HNatInfinite ⟹ hfallfactpow (of_hypnat j) (of_nat k)/(of_hypnat j pow of_nat k) ≈ (1::'a::real_normed_field star)"
⟨*proof*⟩

lemma *hfallfactpow_base_starfun2:*
*"hfallfactpow a = (*f2* fallfactpow) a"*

⟨*proof*⟩

lemma *InternalFuns_hfallfactpow [simp]: "hfallfactpow a ∈ InternalFuns"*
⟨*proof*⟩

lemma *hfallfactpow_starfun2:*
*"hfallfactpow = *f2* fallfactpow"*
⟨*proof*⟩

lemma *InternalFuns2_hfallfactpow [simp]: "hfallfactpow ∈ InternalFuns2"*
⟨*proof*⟩

lemma *InternalFuns2_hfallfactpow_base [simp]: "(λn. hfallfactpow a) ∈ InternalFuns2"*
⟨*proof*⟩

lemma *hfallfactpow: "hfallfactpow (star_n X) (star_n Y) = star_n (λn. fallfactpow (X n) (Y n))"*
⟨*proof*⟩

lemma *InternalFuns_hfallfactpow_fun:*
assumes *"f ∈ InternalFuns"*
shows *"(λn. hfallfactpow a (f n)) ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_hfallfactpow_divide:*
"(λk. (hfallfactpow (of_hypnat j) k/(of_hypnat j pow k)) :: 'a :: {comm_ring_1, division_ring} star) ∈ InternalFuns"
⟨*proof*⟩

lemma *hfact:*
"hfact (star_n X) = star_n(λn. fact (X n))"
⟨*proof*⟩

lemma *InternalFuns_hfact_fun [simp]:*
"f ∈ InternalFuns ⟹ (λn. hfact (f n)) ∈ InternalFuns"
⟨*proof*⟩

lemma *InternalFuns_hfact [simp]: "(λn. hfact n) ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns2_hfact [simp]: "(λm. hfact) ∈ InternalFuns2"*
⟨*proof*⟩

lemma *Infinitesimal_hyperpow_star_of_less_one:*
assumes *"hnorm ((star_of x)::'a::real_normed_algebra_1 star) < 1"*
"n ∈ HNatInfinite"
shows *"((star_of x)::'a::real_normed_algebra_1 star) pow n ∈ Infinitesimal"*
⟨*proof*⟩

lemma *InternalFuns_abs_pow_fun:*
assumes *Int_f: "f ∈ InternalFuns"* **shows** *"(λn. |x pow f n|) ∈ InternalFuns"*
⟨*proof*⟩

lemma *hfact_gt_zero [simp]: "⋀n. (0 :: 'a :: linordered_semidom star)*
< hfact (n::hypnat)"
⟨*proof*⟩

lemma *hypnat_fact_zero [simp] : "hfact (0 :: nat star) = 1"*
⟨*proof*⟩

8.5 Nonstandard version of Pochammer's symbol i.e. the rising factorial

definition
hpochhammer :: "'a::comm_semiring_1 star ⇒ hypnat ⇒ 'a star" **where**
*[transfer_unfold]: "hpochhammer x n = (*f2* pochhammer) x n"*

lemma *fact_fact_pochhammer_mult:*
*"n ≥ k ⟹ fact n = (fact k) * pochhammer (k + 1) (n - k)"*
⟨*proof*⟩

lemma *fact_fact_hpochhammer_mult:*
*"⋀n k. n ≥ k ⟹ hfact n = (hfact k) * hpochhammer (k + 1) (n - k)"*
⟨*proof*⟩

lemma *power_le_pochhammer:*
assumes *"0 ≤ x"*
shows *"(x ^ n :: 'a :: linordered_semidom) ≤ pochhammer x n"*
⟨*proof*⟩

lemma *hyperpow_le_hpochhammer:*
"⋀x n. 0 ≤ x ⟹ (x pow n :: 'a :: linordered_semidom star) ≤ hpochhammer
x n"
⟨*proof*⟩

lemma *of_nat_pochhammer_of_nat:*
assumes *"0 ≤ z"* **shows** *"of_nat (pochhammer (nat z) n) = pochhammer*
(of_int z) n"
⟨*proof*⟩

lemma *of_hypnat_hpochhammer_of_hypnat:*
"⋀z n. 0 ≤ z ⟹ of_hypnat (hpochhammer (hypnat z) n) = hpochhammer
(of_hypint z) n"
⟨*proof*⟩

lemma *hyperpow_divide_fact_le_lemma':*
assumes *"y ∈ HFinite"*
and *"(0 :: real star) < y"*
and *"hypnat (hfloor y + 1) ≤ n"*
shows *"|y pow n| / hypreal_of_hypnat (hfact n)*
≤ y pow n /
((hypreal_of_hypint (hfloor y) + 2) pow
*(n - hypnat (hfloor y + 1)) **
hypreal_of_hypnat (hfact (hypnat (hfloor y + 1))))"
⟨*proof*⟩

lemma *hfloor_one_minus_epsilon: "hfloor(1 - ε) = 0"*
⟨*proof*⟩

lemma *hypreal_hfloor_approx_zero [simp]: "⟦ (x::hypreal) ≈ 0; x > 0 ⟧*
⟹ hfloor x = 0"
⟨*proof*⟩

lemma *hfloor_approx_zero [simp]: "x ≈ 0 ⟹ hfloor (hnorm x) = 0"*
⟨*proof*⟩

lemma *InternalFuns_expf_term' [simp]:*
"(λN. y pow N / hypreal_of_hypnat (hfact N)) ∈ InternalFuns"
⟨*proof*⟩

lemma *InternalFuns_expf_term [simp]:*
"(λN. y pow N / (hfact N :: real star)) ∈ InternalFuns"
⟨*proof*⟩

lemma *HyperSummable_hyperpow_divide_fact_Infinitesimal:*
assumes *HFinite_y: "(y::hypreal) ∈ HFinite"*
and *y_gt_0: "y > 0"*
and *y_approx_0: "y ≈ 0"*
shows *"HyperSummable (λN. y pow N/of_hypnat(hfact N))"*
⟨*proof*⟩

lemma *HyperSummable_hyperpow_divide_fact_not_Infinitesimal:*
assumes *HFinite_y: "(y::hypreal) ∈ HFinite"*
and *y_gt_0: "y > 0"*
and *y_not_approx_0: "¬ y ≈ 0"*
shows *"HyperSummable (λN. y pow N/of_hypnat(hfact N))"*
⟨*proof*⟩

lemma *HyperSummable_hyperpow_divide_fact_pos:*
"⟦ (y::hypreal) ∈ HFinite; y > 0 ⟧ ⟹ HyperSummable (λN. y pow N/of_hypnat(hfact N))"
⟨*proof*⟩

lemma *HyperSummable_hyperpow_divide_fact_neg:*
assumes *"(y::hypreal) ∈ HFinite"*
and *"y < 0"*
shows *"HyperSummable (λN. y pow N/of_hypnat(hfact N))"*
⟨*proof*⟩

lemma *InternalFuns_star_choose: "(hchoose) k ∈ InternalFuns"*
⟨*proof*⟩

lemma *InternalFuns_star_choose_from [simp]: "(λn. n hchoose k) ∈ InternalFuns"*
⟨*proof*⟩

lemma *hypnat_fact_not_zero [simp]: "hfact n ≠ (0::hypnat)"*
⟨*proof*⟩

lemma *hfact_nonzero [simp]:*
"⋀k. hfact k ≠ (0::'a::{semiring_char_0,semiring_no_zero_divisors}
star)"
⟨*proof*⟩

lemma *HyperSummable_geometric':*
assumes *"hnorm (x::'a::{real_normed_field} star) < 1"*
"¬hnorm x ≈ 1"
shows *"HyperSummable (λN. x pow (hSuc N))"*
⟨*proof*⟩

lemma *HyperSummable_hyperpow_divide_fact_zero:*
"HyperSummable (λn. 0 pow n / hypreal_of_hypnat (hfact n))"
⟨*proof*⟩

lemma *HyperSummable_exp':*
"(y::hypreal) ∈ HFinite ⟹ HyperSummable (λN. y pow N/of_hypnat(hfact
N))"
⟨*proof*⟩

lemma *of_hypnat_hfact:*
"⋀n. of_hypnat(hfact n) = hfact n"
⟨*proof*⟩

lemma *HyperSummable_exp:*
"(y::hypreal) ∈ HFinite ⟹ HyperSummable (λN. y pow N/hfact N)"
⟨*proof*⟩

lemma *lemma_n_starfun_extract_binomial_hfallfactpow:*
*"eventually (λn. ((*nf* (λk. of_hypnat ((*f2* (choose)) (star_n Xb)*
*k) **

```
                                (*f2* fallfactpow) (star_n X) (star_n Xb
- k) *
                                (*f2* fallfactpow) (star_n Xa) k)) n) =

                    (λk. of_nat(Xb n choose k) *
                        fallfactpow (X n) (Xb n - k) *
                        fallfactpow (Xa n) k)) 𝒰"
```
⟨*proof*⟩

lemma *hyperbinomial_fallfactpow_ring:*
```
  "hfallfactpow (a + b :: 'a::{comm_ring_1} star) n =
  (hypersum (λk. of_hypnat(n hchoose k) * hfallfactpow a (n - k) * hfallfactpow
b k)) {0..n}"
```
⟨*proof*⟩

lemma *hyperbinomial_fallfactpow_ring2:*
```
  "⋀a b n. hfallfactpow (a + b :: 'a::{comm_ring_1} star) n =
  (hypersum (λk. of_hypnat(n hchoose k) * hfallfactpow a k * hfallfactpow
b (n - k))) {0..n}"
```
⟨*proof*⟩

lemma *hyperbinomial_hfact:*
```
 "⋀n k. k ≤ n ⟹ of_hypnat (n hchoose k) = (hfact n :: 'a:: field_char_0
star) / (hfact k * hfact (n - k))"
```
⟨*proof*⟩

lemma *hypersum_cong:*
assumes *ifun_g:* `"g ∈ InternalFuns"`
and *ifun_h:* `"h ∈ InternalFuns"`
and *iset_A:* `"A ∈ InternalSets"`
and `"A = B"`
and *g_h:* `"⋀x. x ∈ B ⟹ g x = h x"`
shows `"hypersum g A = hypersum h B"`
⟨*proof*⟩

lemma *hyperbinomial_vandermonde:*
```
  "(r + s) pow k/hfact k =
   hypersum (λi. (r::'a :: field_char_0 star) pow i / hfact i *  s pow
(k - i) / hfact (k - i)) {0..k}"
```
⟨*proof*⟩

lemma *hyperbinomial_hfallfactpow_vandermonde:*
```
  "hfallfactpow (r + s) k/hfact k =
   hypersum (λi. hfallfactpow (r::'a :: field_char_0 star)  i / hfact
i *  hfallfactpow s  (k - i) / hfact (k - i)) {0..k}"
```
⟨*proof*⟩

lemma *hyperbinomial_fallfactpow_ring3:*
"hfallfactpow (a + b :: 'a::{field_char_0} star) n =
*(hypersum (λk. hfallfactpow (of_hypnat n) k / hfact k * hfallfactpow a (n - k) * hfallfactpow b k)) {0..n}"*
⟨*proof*⟩

Setting up a few lemmas to prove properties about infinitesimal exponents

lemma *binomial_expand_first_two_terms:*
*"0 ≤ u ⟹ ∃x≥0. (1+ (u::'a::{linordered_idom,power}))^n = 1 + of_nat n * u + x"*
⟨*proof*⟩

lemma *hyperbinomial_expand_first_two_terms:*
*"⋀u n. 0 ≤ u ⟹ ∃x≥0. (1 + (u::'a::{linordered_idom,power} star)) pow n = 1 + of_hypnat n * u + x"*
⟨*proof*⟩

lemma *hyperbinomial_expand_first_two_terms':*
*"⋀u n. 0 ≤ u ⟹ ∃x≥0. ((u::'a::{linordered_idom,power} star) + 1) pow n = 1 + of_hypnat n * u + x"*
⟨*proof*⟩

end

9 Hyperreal powers

theory *HyperrealPower*
imports *"Real_Power.Log" "HOL-Nonstandard_Analysis.NatStar" HyperBinomial*
begin

definition
hpow :: "[hypreal,hypreal] ⇒ hypreal" (**infixr** *"hpow" 80*) **where**
[transfer_unfold]: "x hpow a = starfun2 (pow$_{\mathbb{R}}$) x a"

lemma *"(hpow) ∈ InternalFuns2"*
⟨*proof*⟩

lemma *hpow: "(star_n X) hpow (star_n Y) = star_n (λn. (X n) pow$_{\mathbb{R}}$ (Y n))"*
⟨*proof*⟩

9.1 Tranferred properties

lemma *hpow_one_eq_one [simp]: "⋀a. 1 hpow a = 1"*
⟨*proof*⟩

lemma *hpow_zero_eq_one [simp]: "⋀a. a > 0 ⟹ a hpow 0 = 1"*
⟨*proof*⟩

lemma *hpow_minus_one:* "⋀a. 0 < a ⟹ a hpow -1 = inverse a"
⟨*proof*⟩

lemma *hpow_one [simp]:* "⋀a. a > 0 ⟹ a hpow 1 = a"
⟨*proof*⟩

lemma *hpow_gt_zero:* "⋀a x. a > 0 ⟹ a hpow x > 0"
⟨*proof*⟩

lemma *hpow_not_zero:* "⋀a x. a > 0 ⟹ a hpow x ≠ 0"
⟨*proof*⟩

lemma *hpow_minus:* "⋀a x. a > 0 ⟹ a hpow (-x) = inverse (a hpow x)"
⟨*proof*⟩

lemma *hpow_inverse:*
"⋀a x. a > 0 ⟹ inverse (a hpow x) = (inverse a) hpow x"
⟨*proof*⟩

lemma *hpow_mult_base:*
"⋀a b x. ⟦ 0 < a; 0 < b ⟧ ⟹ (a * b) hpow x = (a hpow x) * (b hpow x)"
⟨*proof*⟩

lemma *hpow_mult:*
"⋀a x y. 0 < a ⟹ (a hpow x) hpow y = a hpow (x * y)"
⟨*proof*⟩

lemma *hpow_divide:*
"⋀a b x. ⟦ 0 < a; 0 < b ⟧ ⟹ (a/b) hpow x = (a hpow x) / (b hpow x)"
⟨*proof*⟩

lemma *hpow_divide2:* "⋀a x y. a > 0 ⟹ a hpow (x - y) = (a hpow x)/(a hpow y)"
⟨*proof*⟩

lemma *hpow_ge_one:* "⋀a x. a ≥ 1 ⟹ x ≥ 0 ⟹ a hpow x ≥ 1"
⟨*proof*⟩

lemma *hpow_ge_one2:*
"⋀a x. ⟦ 0 < a; a < 1; x ≤ 0 ⟧ ⟹ a hpow x ≥ 1"
⟨*proof*⟩

lemma *hpow_le_mono:*
"⋀r s a. ⟦ r ≤ s; a ≥ 1 ⟧ ⟹ a hpow r ≤ a hpow s"
⟨*proof*⟩

lemma *hpow_le_anti_mono:*
"⋀r s a. ⟦ r ≤ s; 0 < a; a < 1 ⟧ ⟹ a hpow r ≥ a hpow s"

⟨*proof*⟩

lemma *hpow_less_cancel:*
"⋀r s a. ⟦ a hpow r < a hpow s; a ≥ 1 ⟧ ⟹ r < s"
⟨*proof*⟩

lemma *hpow_less_anti_mono:*
"⋀r s a. ⟦ r < s; 0 < a; a < 1 ⟧ ⟹ a hpow r > a hpow s"
⟨*proof*⟩

lemma *hpow_less_mono:*
"⋀r s a. ⟦ r < s; a > 1 ⟧ ⟹ a hpow r < a hpow s"
⟨*proof*⟩

lemma *hpow_le_cancel:*
"⋀r s a. ⟦ a hpow r ≤ a hpow s; a > 1 ⟧ ⟹ r ≤ s"
⟨*proof*⟩

lemma *hpow_less_cancel_iff [simp]:*
"⋀r s a. 1 < a ⟹ (a hpow r < a hpow s) = (r < s)"
⟨*proof*⟩

lemma *hpow_le_cancel_iff [simp]:*
"⋀r s a. 1 < a ⟹ (a hpow r ≤ a hpow s) = (r ≤ s)"
⟨*proof*⟩

lemma *hpow_inject_exp1 [simp]:*
"⋀r s a. 1 < a ⟹ (a hpow r = a hpow s) = (s = r)"
⟨*proof*⟩

lemma *hpow_inject_exp2 [simp]:*
"⋀r s a. 0 < a ⟹ a < 1 ⟹ (a hpow r = a hpow s) = (s = r)"
⟨*proof*⟩

lemma *hpow_inject [simp]:*
"⋀r s a. 0 < a ⟹ a ≠ 1 ⟹ (a hpow r = a hpow s) = (s = r)"
⟨*proof*⟩

lemma *hpow_gt_one:* "⋀a x. a > 1 ⟹ x > 0 ⟹ a hpow x > 1"
⟨*proof*⟩

lemma *hpow_less_mono_base:*
"⋀a b r. ⟦ r > 0; 0 < a; a < b ⟧ ⟹ a hpow r < b hpow r"
⟨*proof*⟩

lemma *hpow_less_antimono_base:*
"⋀a b r. ⟦ r < 0; 0 < a; a < b ⟧ ⟹ a hpow r > b hpow r"
⟨*proof*⟩

lemma *hpow_hyperpow_eq:*
"⋀a n. a > 0 ⟹ a hpow (of_hypnat n) = a pow n"
⟨*proof*⟩

lemma *hpow_hyperpow_eq2:*
"⋀a n. 0 ≤ a ⟹ 0 < n ⟹ a pow n = (if (a = 0) then 0 else a hpow (of_hypnat n))"
⟨*proof*⟩

lemma *hpow_hypint:*
"⋀x i. x > 0 ⟹ x hpow (of_hypint i) = (if i ≥ 0 then x pow hypnat i else 1 / x pow hypnat (-i))"
⟨*proof*⟩

lemma *hpow_power_eq:*
"⋀a. a> 0 ⟹ a hpow hypreal_of_nat n = a ^ n"
⟨*proof*⟩

lemma *hpow_inverse_of_hypnat_gt_one:*
"⋀a N. ⟦ a > 1; N ≠ 0 ⟧ ⟹ a hpow (inverse(of_hypnat N)) > 1"
⟨*proof*⟩

lemma *hpow_inverse_of_nat_gt_one:*
"⋀a. ⟦ a > 1; N ≠ 0 ⟧ ⟹ a hpow (inverse(of_nat N)) > 1"
⟨*proof*⟩

lemma *Infinitesimal_hyperpow_less_one_lemma:*
"⋀x r. ⟦hnorm (x::'a::real_normed_algebra_1 star) < 1; 0 < r ⟧
⟹ ∃no. ∀n≥no. hnorm (x pow n) < r"
⟨*proof*⟩

lemma *hpow_add: "⋀a x y. 0 < a ⟹ a hpow (x + y) = a hpow x * a hpow y"*
⟨*proof*⟩

lemma *hpow_eq_square: "⋀x. 0 < x ⟹ x hpow 2 = x * x"*
⟨*proof*⟩

lemma *hpow_minus_eq_hpow:*
"⟦ 0 < a; 0 < b; a hpow -x = b hpow y ⟧ ⟹ a hpow x = b hpow - y"
⟨*proof*⟩

lemma *hpow_half_divide_eq:*
"0 < y ⟹ y hpow (1/2) / y = inverse(y hpow (1/2))"
⟨*proof*⟩

lemma *less_hpow_half_lemma:*
"⟦ 0 < x; 0 < y; x < y hpow (1/2) ⟧ ⟹ x/y < inverse(y hpow (1/2))"

⟨*proof*⟩

lemma *le_hpow_half_lemma:*
"⟦ 0 < x; 0 < y; x ≤ y hpow (1/2) ⟧ ⟹ x/y ≤ inverse(y hpow (1/2))"
⟨*proof*⟩

9.2 Relating pow and hpow

lemma *hpow_hyperpow_eq_hpow:* "0 < a ⟹ (a hpow x) pow n = a hpow (x * of_hypnat n)"
⟨*proof*⟩

lemma *hpow_divide_pow_cancel:*
assumes "a > 1" "j ≠ 0"
shows "(a hpow (z/of_hypnat j)) pow j = a hpow z"
⟨*proof*⟩

lemma *hpow_inverse_of_hypnat_cancel:*
"⟦ 0 < a; 0 < N ⟧ ⟹ a = (a hpow (inverse(of_hypnat N))) hpow (of_hypnat N)"
⟨*proof*⟩

lemma *hpow_inverse_of_hypnat_pow_cancel:*
"⟦ 0 < a; 0 < N ⟧ ⟹ (a hpow (inverse(of_hypnat N))) pow N = a"
⟨*proof*⟩

lemma *hpow_inverse_of_hypnat_pow_cancel2:*
"⟦ a > 1; N ∈ HNatInfinite ⟧ ⟹ a = (a hpow (inverse(of_hypnat N))) pow N"
⟨*proof*⟩

lemma *HNatInfinite_hpow_inverse_gt_one:*
"⟦ a > 1; N ∈ HNatInfinite ⟧ ⟹ a hpow (inverse(of_hypnat N)) > 1"
⟨*proof*⟩

lemma *HNatInfinite_hpow_eq_Ex:*
assumes "a > 1" **and** "N ∈ HNatInfinite"
shows "∃u>0. a hpow (inverse(of_hypnat N)) = 1 + u"
⟨*proof*⟩

9.3 Some nonstandardard theorems

lemma *hpow_inverse_hypreal_of_nat_approx_one_cancel:*
assumes "a > 0"
and "m > 0"
and "a hpow inverse (hypreal_of_nat m) ≈ 1"
shows "a ≈ 1"
⟨*proof*⟩

lemma *HNatInfinite_hpow_inverse_approx_one_cancel:*

"⟦ a > 0; ¬(a ≈ 1); (a hpow (inverse(of_hypnat N))) ≈ 1; N ≠ 0 ⟧ ⟹ N ∈ HNatInfinite"
⟨proof⟩

lemma *HFinite_hpow1:*
assumes "a ∈ HFinite"
and "x ∈ HFinite"
and "1 ≤ a"
shows "a hpow x ∈ HFinite"
⟨proof⟩

lemma *HFinite_hpow2:*
assumes "a ∈ HFinite - Infinitesimal"
and "x ∈ HFinite"
and "0 < a"
and "a < 1"
shows "a hpow x ∈ HFinite"
⟨proof⟩

lemma *HFinite_hpow3:*
"⟦ a ∈ HFinite - Infinitesimal; x ∈ HFinite; 0 < a ⟧ ⟹ a hpow x ∈ HFinite"
⟨proof⟩

lemma *HFinite_hpow_not_Infinitesimal:*
assumes "a ∈ HFinite"
and "x ∈ HFinite"
and "1 ≤ a"
shows "a hpow x ∉ Infinitesimal"
⟨proof⟩

lemma *HFinite_hpow_HFinite_Diff_Infinitesimal:*
"⟦ a ∈ HFinite; x ∈ HFinite; a ≥ 1 ⟧ ⟹ a hpow x ∈ HFinite - Infinitesimal"
⟨proof⟩

lemma *approx_hpow_minus_hpow:*
"⟦ 0 < a; 0 < b; a hpow -x ≈ b hpow y; a hpow x ∈ HFinite - Infinitesimal ⟧ ⟹ a hpow x ≈ b hpow - y"
⟨proof⟩

lemma *HNatInfinite_hpow_inverse_approx_one:*
assumes "a ∈ HFinite"
and "a > 1"
and "N ∈ HNatInfinite"
shows "(a hpow (inverse(of_hypnat N))) ≈ 1"
⟨proof⟩

lemma *HNatInfinite_hpow_inverse_diff_one_Infinitesimal:*

"⟦ a ∈ HFinite; a > 1; N ∈ HNatInfinite ⟧ ⟹ (a hpow (inverse(of_hypnat N))) - 1 ∈ Infinitesimal"
⟨*proof*⟩

lemma *HNatInfinite_hpow_inverse_approx_one2:*
"⟦ a ∈ HFinite; a > 1; N ∈ HNatInfinite ⟧ ⟹ (a hpow (1/of_hypnat N)) ≈ 1"
⟨*proof*⟩

One of the results that we want on our way to derive the exponential series a la Euler

lemma *Infinitesimal_pos_hpow_approx_one:*
assumes *HFinite_a:* "a ∈ HFinite"
and *a_gt1:* "a > 1"
and *e_gt0:* "e > 0"
and *Infinitesimal_e:* "e ∈ Infinitesimal"
shows "a hpow e ≈ 1"
⟨*proof*⟩

lemma *Infinitesimal_neg_hpow_approx_one:*
"⟦ a ∈ HFinite; a > 1; e > 0; e ∈ Infinitesimal⟧ ⟹ a hpow -e ≈ 1"
⟨*proof*⟩

And the next result, without e>0 restriction!

lemma *Infinitesimal_hpow_approx_one:*
assumes "a ∈ HFinite"
and "a > 1"
and " e ∈ Infinitesimal"
shows "a hpow e ≈ 1"
⟨*proof*⟩

lemma *hpow_HFinite_approx_1_lemma1:*
assumes *x_finite:* "x ∈ HFinite"
and *a_infclose_1:* "a ≈ 1"
and *a_ge_1:* "a ≥ 1" **and** *x_gt_0:* "x > 0"
shows "a hpow x ≈ 1"
⟨*proof*⟩

lemma *approx_hpow:*
assumes "y ∈ HFinite" **and** "a ∈ HFinite"
and "a > 1" **and** "x ≈ y"
shows "a hpow x ≈ a hpow y"
⟨*proof*⟩

end

10 The hyper logarithm

theory *HyperLog*

imports *"Real_Power.Log" "HOL-Nonstandard_Analysis.NatStar" HyperrealPower*
begin

definition
hLog :: "[hypreal,hypreal] => hypreal" **where**
[transfer_unfold]: "hLog a x = starfun2 Log a x"

lemma *hLog:*
"hLog (star_n X) (star_n Y) = star_n (λn. Log (X n) (Y n))"
⟨*proof*⟩

lemma *hpow_hLog_cancel [simp]:*
"⋀a x. ⟦ a > 0; a ≠ 1; x > 0 ⟧ ⟹ a hpow (hLog a x) = x"
⟨*proof*⟩

lemma *hLog_hpow_cancel [simp]:*
"⋀a y. ⟦ 0 < a; a ≠ 1 ⟧ ⟹ hLog a (a hpow y) = y"
⟨*proof*⟩

lemma *hLog_mult:*
*"⋀a x y. ⟦ 0 < a; a ≠ 1; 0 < x; 0 < y ⟧ ⟹ hLog a (x * y) = hLog a x + hLog a y"*
⟨*proof*⟩

lemma *hLog_one [simp]:*
"⋀a. ⟦ 0 < a; a ≠ 1 ⟧ ⟹ hLog a 1 = 0"
⟨*proof*⟩

lemma *hLog_eq_one [simp]:*
"⋀a. ⟦ 0 < a; a ≠ 1 ⟧ ⟹ hLog a a = 1"
⟨*proof*⟩

lemma *hLog_inverse:*
"⋀a x. ⟦ a > 0; a ≠ 1; x > 0 ⟧ ⟹ hLog a (inverse x) = - hLog a x"
⟨*proof*⟩

lemma *hLog_divide:*
"⋀a x y. ⟦ 0 < a; a ≠ 1; 0 < x; 0 < y ⟧ ⟹ hLog a (x/y) = hLog a x - hLog a y"
⟨*proof*⟩

lemma *hLog_less_cancel_iff [simp]:*
"⋀a x y. ⟦ 1 < a; 0 < x; 0 < y ⟧ ⟹ (hLog a x < hLog a y) = (x < y)"
⟨*proof*⟩

lemma *hLog_inj:* **assumes** *"1 < b"* **shows** *"inj_on (hLog b) {0 <..}"*
⟨*proof*⟩

lemma *hLog_le_cancel_iff [simp]:*
"⋀a x y. ⟦ 1 < a; 0 < x; 0 < y ⟧ ⟹ (hLog a x ≤ hLog a y) = (x ≤ y)"
⟨*proof*⟩

lemma *zero_less_hLog_cancel_iff [simp]:*
"⋀a x. 1 < a ⟹ 0 < x ⟹ 0 < hLog a x ⟷ 1 < x"
⟨*proof*⟩

lemma *zero_le_hLog_cancel_iff[simp]:* "1 < a ⟹ 0 < x ⟹ 0 ≤ hLog a x ⟷ 1 ≤ x"
⟨*proof*⟩

lemma *hLog_less_zero_cancel_iff[simp]:* "1 < a ⟹ 0 < x⟹ hLog a x < 0 ⟷ x < 1"
⟨*proof*⟩

lemma *hLog_le_zero_cancel_iff[simp]:* "1 < a ⟹ 0 < x ⟹ hLog a x ≤ 0 ⟷ x ≤ 1"
⟨*proof*⟩

lemma *one_less_hLog_cancel_iff[simp]:* "1 < a ⟹ 0 < x ==> 1 < hLog a x ⟷ a < x"
⟨*proof*⟩

lemma *one_le_hLog_cancel_iff[simp]:* "1 < a ==> 0 < x ⟹ 1 ≤ hLog a x ⟷ a ≤ x"
⟨*proof*⟩

lemma *hLog_less_one_cancel_iff[simp]:* "1 < a ⟹ 0 < x ⟹ hLog a x < 1 ⟷ x < a"
⟨*proof*⟩

lemma *hLog_le_one_cancel_iff[simp]:* "1 < a ⟹ 0 < x ⟹ hLog a x ≤ 1 ⟷ x ≤ a"
⟨*proof*⟩

lemma *hLog_hpow:* "⋀b x y. ⟦ 0 < x; 1 < b; b ≠ 1 ⟧ ⟹ hLog b (x hpow y) = y * hLog b x"
⟨*proof*⟩

lemma *hLog_nat_power:*
"⋀b x n. ⟦ 0 < x; 1 < b; b ≠ 1 ⟧ ⟹ hLog b (x pow n) = of_hypnat n * hLog b x"
⟨*proof*⟩

end

11 Deriving the exponential series Euler-style

theory *EulerExponential*
imports *HyperBinomial HyperLog*
begin

11.1 Euler's witness: Introductio, paragraph 114

We show explicitly that Euler's witness (that he denotes by k) is a finite quantity. The existence is merely stated by Euler and finiteness is only argued using an example, with Euler merely stating that:

"We see that k is a finite number which depends on the value of the base a".

lemma *Introductio_114_HFinite_witness:*
assumes *infinitesimal_exponent: "e ∈ Infinitesimal"*
and *exponent_gt_zero: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
shows *"(a hpow e - 1)/e ∈ HFinite"*
⟨*proof*⟩

Proposition 114, with some finite k.

lemma *Introductio_114_pos:*
assumes *infinitesimal_exponent: "e ∈ Infinitesimal"*
and *exponent_gt_zero: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
shows *"∃k∈HFinite. k > 0 ∧ a hpow e = 1 + k * e"*
⟨*proof*⟩

This case, not discussed explicitly by Euler, is also needed to eventually deal with finite, negative exponents.

lemma *Introductio_114_neg:*
assumes *infinitesimal_exponent: "e ∈ Infinitesimal"*
and *exponent_less_zero: "e < 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
shows *"∃k∈HFinite. k > 0 ∧ a hpow e = 1 + k * e"*
⟨*proof*⟩

lemma *Introductio_114_HFinite_witness2:*
assumes *infinitesimal_exponent: "e ∈ Infinitesimal"*
and *exponent_less_zero: "e < 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
shows *"(a hpow e - 1)/e ∈ HFinite"*
⟨*proof*⟩

lemma *Introductio_114_HFinite_witness_full:*

```
    assumes "x ∈ Infinitesimal"
    and "a ∈ HFinite"
    and "a > 1"
  shows "(a hpow x - 1)/x ∈ HFinite"
```
⟨*proof*⟩

11.2 Defining Euleur's finite witness

In fact, let us introduce a definition that will enable us to characterize Euler's k, (called the scaling factor by McKinzie and Tuckey but also left undefined by them).

```
definition eulerK :: "hypreal ⇒ hypreal ⇒ hypreal" where
  "eulerK a x ≡ (a hpow x - 1)/ x"
```

```
lemma eulerK_neg_exponent:
  assumes "a > 0" and "e ≠ 0"
  shows "eulerK a (-e) = eulerK a e / (a hpow e)"
```
⟨*proof*⟩

Equivalent multiplicative form.

```
corollary eulerK_neg_exponent_mult:
  assumes "a > 0" and "e ≠ 0"
  shows "eulerK a (-e) * (a hpow e) = eulerK a e"
```
⟨*proof*⟩

```
lemma eulerK_neg_approx:
  assumes "a ∈ HFinite" and "a > 1" and "x ∈ Infinitesimal"
  shows "eulerK a (-x) ≈ eulerK a x"
```
⟨*proof*⟩

```
corollary eulerK_epsilon_neg_approx:
  assumes "a ∈ HFinite" and "a > 1"
  shows "eulerK a (-ε) ≈ eulerK a ε"
```
⟨*proof*⟩

```
lemma Introductio_114_eulerK_witness:
  assumes "x ∈ Infinitesimal"
  and "a ∈ HFinite"
  and "a > 1"
  shows "eulerK a x ∈ HFinite"
```
⟨*proof*⟩

```
lemma HFinite_eulerK_inverse_whn:
  assumes "a ∈ HFinite"
  and "a > 1"
  shows "eulerK a (1/of_hypnat whn) ∈ HFinite"
```
⟨*proof*⟩

lemma *Introductio_114_eulerK_epsilon_witness:*
assumes *"a ∈ HFinite"*
and *"a > 1"*
shows *"eulerK a ε ∈ HFinite"*
⟨*proof*⟩

lemma *eulerK_neg_e_HFinite:*
assumes *"e ∈ Infinitesimal" "e > 0" "a ∈ HFinite" "a > 1"*
shows *"eulerK a (-e) ∈ HFinite"*
⟨*proof*⟩

lemma *Introductio_114_neg_eulerK_epsilon_witness:*
assumes *"a ∈ HFinite" "a > 1"*
shows *"eulerK a (-ε) ∈ HFinite"*
⟨*proof*⟩

lemma *eulerK_neg_e_gt_zero:*
assumes *"e ∈ Infinitesimal" "e > 0" "a ∈ HFinite" "a > 1"*
shows *"eulerK a (-e) > 0"*
⟨*proof*⟩

lemma *eulerK_neg_epsilon_gt_zero:*
assumes *"a ∈ HFinite" "a > 1"*
shows *"eulerK a (-ε) > 0"*
⟨*proof*⟩

lemma *Introductio_114_eulerK:*
shows *"a > 0 ⟹ a hpow e = 1 + eulerK a e * e"*
⟨*proof*⟩

lemma *Introductio_114_eulerK':*
shows *"e ≠ 0 ⟹ a hpow e = 1 + eulerK a e * e"*
⟨*proof*⟩

lemma *Introductio_114_eulerK_epsilon:*
shows *"a hpow ε = 1 + eulerK a ε * ε"*
⟨*proof*⟩

lemma *Introductio_114_epsilon:*
*"a > 1 ⟹ ε = hLog a (1 + eulerK a ε * ε)"*
⟨*proof*⟩

lemma *eulerK_product_neg_e:*
assumes *"a > 0"* **and** *"e ≠ 0"*
shows *"eulerK a (-e) * (-e) = eulerK a e * (-e) / (a hpow e)"*
⟨*proof*⟩

lemma *eulerK_pow_neg_e:*
assumes *"a > 0"* **and** *"e ≠ 0"*

shows *"(eulerK a (-e) * (-e)) pow of_nat n = (eulerK a e * (-e)) pow of_nat n / (a hpow e) ^ n"*
⟨*proof*⟩

11.3 Euler's Introductio, Paragraph 115: first expansion

This expansion is given by Euler who implicitly uses the Binomial theorem.

lemma *Introductio_115_expansion_hchoose:*
*"a > 0 ⟹ (a hpow x) pow j = hypersum (λi. of_hypnat (j hchoose i) * (eulerK a x * x) pow i) {0..j}"*
⟨*proof*⟩

lemma *Introductio_115_expansion_hffp:*
"a > 0 ⟹
(a hpow x) pow j =
*hypersum (λi. (hfallfactpow (of_hypnat j) i)/ hfact i * (eulerK a x * x) pow i) {0..j}"*
⟨*proof*⟩

lemma *Introductio_115_expansion_hchoose':*
*"e ≠ 0 ⟹ (a hpow e) pow j = hypersum (λi. of_hypnat (j hchoose i) * (eulerK a e * e) pow i) {0..j}"*
⟨*proof*⟩

lemma *Introductio_115_expansion_hchoose_epsilon:*
*"(a hpow ε) pow j = hypersum (λi. of_hypnat (j hchoose i) * (eulerK a ε * ε) pow i) {0..j}"*
⟨*proof*⟩

lemma *HFinite_Infinitesimal_hpow_cancel_approx:*
assumes *infinitesimal_exponent: "e ∈ Infinitesimal"*
and *exponent_not_zero: "e ≠ 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
and *exponentz_HFinite: "z ∈ HFinite"*
shows *"(a hpow e) hpow of_hypint(hfloor(z/e)) ≈ a hpow z"*
⟨*proof*⟩

lemma *hpow_hypersum_pos_hchoose:*
assumes *"e ∈ Infinitesimal"* **and** *"e > 0"*
and *"a ∈ HFinite"* **and** *"a > 1"*
and *"z ∈ HFinite"* **and** *"z > 0"*
defines *"N ≡ hypnat (hfloor (z / e))"*
shows *"a hpow z ≈ ($\sum_h$ i∈{0..N}. hypreal_of_hypnat (N hchoose i) * (eulerK a e * e) pow i)"*
⟨*proof*⟩

lemma *hpow_hypersum_pos_hffp:*

```
  assumes "e ∈ Infinitesimal" and "e > 0"
      and "a ∈ HFinite" and "a > 1"
      and "z ∈ HFinite" and "z > 0"
    defines "N ≡ hypnat (hfloor (z / e))"
    shows "a hpow z ≈
           (∑h i∈{0..N}. hfallfactpow (of_hypnat N) i / hfact i * (eulerK
a e * e) pow i)"
  ⟨proof⟩

lemma HFinite_Infinitesimal_hpow_hypersum_pos_eulerK_epsilon:
    assumes "a ∈ HFinite" and "a > 1"
    and "z ∈ HFinite" and "z > 0"
    defines "N ≡ hypnat (hfloor (z/ε))"
    shows "a hpow z ≈
             (∑h i∈{0..N}. hypreal_of_hypnat (N hchoose i) * (eulerK
a ε * ε) pow i)"
  ⟨proof⟩

lemma Infinitesimal_pow_hfallfactpow_le_pow:
      assumes infinitesimal_e: "e ∈ Infinitesimal"
      and e_gt_zero: "e > (0::hypreal)"
      and exponentz_HFinite: "z ∈ HFinite - Infinitesimal"
      and exponentz_gt_zero: "z > 0"
      defines "N ≡ hypnat (hfloor (z / e))"
      shows "e pow n * hfallfactpow (of_hypnat N) n ≤ z pow n"
⟨proof⟩

lemma HyperSummable_exp_terms:
      assumes infinitesimal_e: "e ∈ Infinitesimal"
      and e_gt_zero: "e > (0::hypreal)"
      and exponentz_HFinite: "z ∈ HFinite - Infinitesimal"
      and exponentz_gt_zero: "z > 0"
      and k_HFinite: "k ∈ HFinite"
      and k_gt_zero: "k > 0"
      defines "N ≡ hypnat (hfloor (z / e))"
      shows "HyperSummable
             (λi. hfallfactpow (of_hypnat N) i / hfact i * (k * e) pow
i)"
⟨proof⟩
```

We compare against the previous series this time to show hypersum for negative exponent is also summable

```
lemma HyperSummable_exp_terms2:
      assumes infinitesimal_e: "e ∈ Infinitesimal"
      and e_gt_zero: "e > (0::hypreal)"
      and exponentz_HFinite: "z ∈ HFinite - Infinitesimal"
      and exponentz_less_zero: "z < 0"
      and k_HFinite: "k ∈ HFinite"
      and k_gt_zero: "k > 0"
```

defines *"N ≡ hypnat (hfloor (-z / e))"*
shows *"HyperSummable (λi. hfallfactpow (of_hypnat N) i / hfact i * (k * -e) pow i)"*
⟨*proof*⟩

lemma *HyperSummable_binomial_neg_hchoose:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *infitesimal_e_gt_zero: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
and *exponentz_less_zero: "z < 0"*
defines *"N ≡ hypnat (hfloor(-z/e))"*
shows *"HyperSummable (λi. of_hypnat (N hchoose i) * (eulerK a e * -e) pow i)"*
⟨*proof*⟩

lemma *HyperSummable_binomial_neg_hchoose':*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *infitesima_e_gt_zero: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
and *exponentz_less_zero: "z < 0"*
defines *"N ≡ hypnat (hfloor(-z/e))"*
shows *"HyperSummable (λi. of_hypnat (N hchoose i) * (eulerK a (-e) * -e) pow i)"*
⟨*proof*⟩

lemma *hpow_hypersum_neg_hchoose:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *infinitesimal_e_gt_zero: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
and *exponentz_HFinite: "z ∈ HFinite"*
and *exponentz_less_zero: "z < 0"*
defines *"N ≡ hypnat (hfloor (- z/e))"*
shows *"a hpow z ≈ (∑h i∈{0..N}. of_hypnat (N hchoose i) * (eulerK a (- e) * - e) pow i)"*
⟨*proof*⟩

lemma *hyperpow_nat_approx:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_gt_0: "z > 0"*
shows *"z pow of_nat i ≈ (hfallfactpow (of_hypnat(hypnat(hfloor(z/e)))) (of_nat i)) * (e pow of_nat i)"*

⟨*proof*⟩

lemma *hyperpow_nat_approx':*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_gt_0: "z > 0"*
and *Nats_i: "i ∈ ℕ"*
shows *"z pow i ≈ (hfallfactpow (of_hypnat(hypnat(hfloor(z/e)))) i) * (e pow i)"*
⟨*proof*⟩

lemma *HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_neg:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_less_0: "z < 0"*
shows *"z pow of_nat i ≈ (hfallfactpow (of_hypnat(hypnat(hfloor(-z/e)))) (of_nat i)) * ((-e) pow of_nat i)"*
⟨*proof*⟩

lemma *HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow2:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_gt_0: "z > 0"*
and *HFinite_k: "k ∈ HFinite"*
shows *"(k * z) pow of_nat i ≈ (hfallfactpow (of_hypnat(hypnat(hfloor(z/e)))) (of_nat i)) * ((k * e) pow of_nat i)"*
⟨*proof*⟩

lemma *HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow2_neg:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_less_0: "z < 0"*
and *HFinite_k: "k ∈ HFinite"*
shows *"(k * z) pow of_nat i ≈ (hfallfactpow (of_hypnat(hypnat(hfloor(-z/e)))) (of_nat i)) * ((k * -e) pow of_nat i)"*
⟨*proof*⟩

lemma *HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_divide:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_gt_0: "z > 0"*
and *HFinite_k: "k ∈ HFinite"*

shows *"(k * z) pow of_nat i/fact i ≈*
*(hfallfactpow (of_hypnat(hypnat(hfloor(z/e)))) (of_nat i))/fact i * ((k * e) pow of_nat i)"*
⟨*proof*⟩

lemma *HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_divide_neg:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *e_gt_0: "e > (0::hypreal)"*
and *z_HFinite_not_Infinitesimal: "z ∈ HFinite - Infinitesimal"*
and *z_less_0: "z < 0"*
and *HFinite_k: "k ∈ HFinite"*
shows *"(k * z) pow of_nat i/fact i ≈*
*(hfallfactpow (of_hypnat(hypnat(hfloor(-z/e)))) (of_nat i))/fact i * ((k * -e) pow of_nat i)"*
⟨*proof*⟩

lemma *binomial_summand_HFinite_neg:*
assumes *"e ∈ Infinitesimal" "e > 0"*
"z ∈ HFinite - Infinitesimal" "z < 0"
"k ∈ HFinite" "k > 0"
"n = of_nat m"
shows *"hypreal_of_hypnat (hypnat (hfloor(-z/e)) hchoose n) * (k * -e) pow n ∈ HFinite"*
⟨*proof*⟩

These two binomial summands are infinitely close for standard n.

lemma *hchoose_approx_for_Nats:*
assumes *Infinitesimal_e: "e ∈ Infinitesimal"*
and *Infinitesima_e_gt0: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
and *exponentz_less_zero: "z < 0"*
shows *"∀n∈ℕ. hypreal_of_hypnat (hypnat (hfloor(-z/e)) hchoose n) * (eulerK a e * -e) pow n ≈*
*hypreal_of_hypnat (hypnat (hfloor(-z/e)) hchoose n) * (eulerK a (-e) * -e) pow n"*
⟨*proof*⟩

lemma *hypersetsetsum_neg_approx:*
assumes *infinitesimal_e: "e ∈ Infinitesimal"*
and *infinitesimal_e_gt_zero: "e > 0"*
and *base_finite: "a ∈ HFinite"*
and *base_greater_1: "a > 1"*
and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
and *exponentz_less_zero: "z < 0"*
shows *"hypersum*
*(λi. of_hypnat (hypnat (hfloor (- z/e)) hchoose i) *

```
                    (eulerK a e * -e) pow i)
                {0..hypnat (hfloor (- z/e))} ≈
                hypersum
                (λi. of_hypnat (hypnat (hfloor (- z/e)) hchoose i) *
                    (eulerK a (-e) * -e) pow i)
                {0..hypnat (hfloor (- z/e))}"
```

⟨*proof*⟩

lemma *hpow_hypersum_neg_hchoose':*
  assumes *"e ∈ Infinitesimal"*
      and *infinitesimal_e_gt_zero: "e > 0"*
      and *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
      and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
      and *exponentz_less_zero: "z < 0"*
      shows *"a hpow z ≈*
              hypersum (λi. of_hypnat (hypnat (hfloor (- z/e)) hchoose
*i) * (eulerK a e * -e) pow i)*
              {0..hypnat (hfloor (- z/e))}"

⟨*proof*⟩

Euler's series in Paragraph 115, with finite, non-infinitesimal z > 0:

lemma *hpow_approx_series_pos:*
      assumes *infinitesimal_e: "e ∈ Infinitesimal"*
      and *e_gt_zero: "e > 0"*
      and *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
      and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
      and *exponentz_gt_zero: "z > 0"*
    defines *"N ≡ hypnat(hfloor(z/e))"*
      shows *"a hpow z ≈ hypersum (λi. (eulerK a e * z) pow i/ hfact i)*
{0..N}"

⟨*proof*⟩

lemma *hpow_approx_series_pos_epsilon:*
  assumes *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
      and *exponentz_HFinite: "z ∈ HFinite - Infinitesimal"*
      and *exponentz_gt_zero: "z > 0"*
      shows *"a hpow z ≈*
             *hypersum (λi. (eulerK a ε * z) pow i/ hfact i) {0..hypnat(hfloor(z/ε))}"*

⟨*proof*⟩

Euler's series in Paragraph 115, with finite, non-infinitesimal z < 0:

lemma *hpow_approx_series_neg:*
      assumes *infinitesimal_e: "e ∈ Infinitesimal"*
      and *e_gt_zero: "e > 0"*
      and *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*

```
      and exponentz_HFinite: "z ∈ HFinite - Infinitesimal"
      and exponentz_less_zero: "z < 0"
      shows "a hpow z ≈
             hypersum (λi. (eulerK a e * z) pow i/ hfact i) {0..hypnat(hfloor(-z/e))}"
```

⟨*proof*⟩

Euler's series in Paragraph 115, with infinitesimal z:

```
lemma hpow_approx_series_infinitesimal':
    assumes infinitesimal_e: "e ∈ Infinitesimal"
      and base_finite: "a ∈ HFinite"
      and base_greater_1: "a > 1"
      and exponentz_Infinitesimal: "z ≈ 0"
      shows "a hpow z ≈
             hypersum (λi. (eulerK a e * z) pow i/ hfact i) {0..< hSuc
n}"
```

⟨*proof*⟩

```
lemma hpow_approx_series_infinitesimal:
  assumes "e ∈ Infinitesimal"
      and base_finite: "a ∈ HFinite"
      and base_greater_1: "a > 1"
      and exponentz_Infinitesimal: "z ≈ 0"
      shows "a hpow z ≈
             hypersum (λi. (eulerK a e * z) pow i / hfact i) {0..n}"
```

⟨*proof*⟩

```
lemma hpow_approx_series_infinitesimal_epsilon:
      assumes base_finite: "a ∈ HFinite"
      and base_greater_1: "a > 1"
      and exponentz_Infinitesimal: "z ≈ 0"
      shows "a hpow z ≈
             hypersum (λi. (eulerK a ε * z) pow i / hfact i) {0..n}"
```

⟨*proof*⟩

Euler's power series in Paragraph 115:

```
lemma hpow_approx_powseries:
      assumes infinitesimal_e: "e ∈ Infinitesimal"
      and e_gt_zero: "e > 0"
      and base_finite: "a ∈ HFinite"
      and base_greater_1: "a > 1"
      and exponentz_HFinite: "z ∈ HFinite"
      shows "a hpow z ≈
             hypersum (λi. (eulerK a e * z) pow i / hfact i) {0..hypnat(hfloor(|z|/e))}"
```

⟨*proof*⟩

11.4 Euler's Introductio, Paragraph 116: exponential function.

First version of main theorem, with Euler's k explicit:

lemma *Euler_hpow_approx_powseries':*
  assumes *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
      and *exponentz_HFinite: "z ∈ HFinite"*
      and *index_infinite: "n ∈ HNatInfinite"*
      shows *"a hpow z ≈*
            *hypersum (λi. (eulerK a (1 / of_hypnat n) * z) pow i / hfact i) {0..n}"*
⟨*proof*⟩

We can remove the dependency between the infinite sum and the scaling factor as the hypersequence is hypersummable (hyper Cauchy) and so infinitely close for any infinite m and n, giving us Euler's result for power series.

lemma *Euler_hpow_approx_powseries:*
  assumes *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
      and *exponentz_HFinite: "z ∈ HFinite"*
      and *pow_infinite: "n ∈ HNatInfinite"*
      and *index_infinite: "m ∈ HNatInfinite"*
      shows *"a hpow z ≈*
            *hypersum (λi. (eulerK a (1 / of_hypnat n) * z) pow i / hfact i) {0..m}"*
⟨*proof*⟩

lemma *epsilon_inverse_hSuc_whn:*
  "ε = 1/hypreal_of_hypnat(hSuc whn)"
  ⟨*proof*⟩

We derive another simpler version of Euler's result by fixing our positive infinitesimal.

lemma *Euler_hpow_approx_powseries_epsilon:*
  assumes *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
      and *exponentz_HFinite: "z ∈ HFinite"*
      shows *"a hpow z ≈*
            *hypersum (λi. (eulerK a ε * z) pow i/ hfact i) {0..whn}"*
  ⟨*proof*⟩

Theorem as stated by McKinzie and Tucker, 2001:

lemma *Euler_hpow_approx_powseries_MT:*
  assumes *base_finite: "a ∈ HFinite"*
      and *base_greater_1: "a > 1"*
    shows *"∃k∈HFinite.*
            ∀z∈HFinite.

```
                ∀n∈HNatInfinite.
                    a hpow z ≈ hypersum (λi. (k * z) pow i/ hfact i)
{0..n}"
```
⟨*proof*⟩

lemma *binomial_term_le_inv_fact:*
assumes *"k ≤ N"* **and** *"(0::hypreal) < of_hypnat N"*
shows *"hypreal_of_hypnat (N hchoose k) * (1 / of_hypnat N) pow k*
≤ 1 / hfact k"
⟨*proof*⟩

lemma *geometric_half_sum_lt_two:*
"∑h ((pow) (1 / 2)) {0..M} < (2::hypreal)"
⟨*proof*⟩

lemma *inv_fact_sum_le_three:*
"hypersum (λk. 1 / (hfact k)) {0..N} ≤ (3 ::hypreal)"
⟨*proof*⟩

lemma *HNatInfinite_one_plus_inv_pow_HFinite:*
assumes *"n ∈ HNatInfinite"*
shows *"(1 + 1 / hypreal_of_hypnat n) pow n ∈ HFinite"*
⟨*proof*⟩

lemma *hpow_approx_powseries_exp_lemma:*
assumes *exponentz_HFinite: "z ∈ HFinite"*
and *index_infinite: "m ∈ HNatInfinite"*
and *pow_infinite: "n ∈ HNatInfinite"*
shows *"((1 + 1/of_hypnat n) pow n) hpow z ≈*
hypersum (λi. z pow i/ hfact i) {0..m}"
⟨*proof*⟩

lemma *hpow_approx_powseries_exp_lemma2:*
assumes *"z ∈ HFinite"*
shows *"((1 + 1/of_hypnat whn) pow whn) hpow z ≈ hypersum (λi. z*
pow i/ hfact i) {0..whn}"
⟨*proof*⟩

Setting the exponent z=1, gives us the series to evaluate the base, which in Introductio 122, Euler denotes by the symbol e. We do the same below.

lemma *hpow_approx_powseries_exp_lemma3:*
"(1 + (1::hypreal)/of_hypnat whn) pow whn ≈ hypersum (λi. 1/
hfact i) {0..whn}"
⟨*proof*⟩

11.5 Euler's Introductio, Paragraph 122: exponential series expansion for the base $\mathfrak{e}$

We can define a Euler's $\mathfrak{e}$ as follows although to pin it down to a unique real number, we should use the standard part function.

definition *euler_e (<𝔢>)* **where**
"𝔢 = (1 + 1/of_hypnat whn) pow whn"

Basic properties of $\mathfrak{e}$.

lemma *euler_e_pos [simp]: "(𝔢::hypreal) > 0"*
⟨*proof*⟩

The (infinitely close) series for $\mathfrak{e}$

lemma *Euler_e:*
"𝔢 ≈ hypersum (λi. (1::hypreal)/ hfact i) {0..whn}"
⟨*proof*⟩

Euler's Exponential series as a hyperfinite hypersum i.e. an infinite polynomial:

lemma *Euler_exp_powseries:*
assumes *"z ∈ HFinite"*
shows *"𝔢 hpow z ≈*
hypersum (λi. z pow i/ hfact i) {0..whn}"
⟨*proof*⟩

lemma *euler_e_hpow_zero [simp]: "𝔢 hpow 0 = 1"*
⟨*proof*⟩

lemma *euler_e_hpow_one [simp]: "𝔢 hpow 1 = 𝔢"*
⟨*proof*⟩

lemma *hexp_gt_one: "0 < x ⟹ 1 < 𝔢 hpow x"*
⟨*proof*⟩

lemma *HFinite_euler_e [simp]:*
"(𝔢::real star) ∈ HFinite"
⟨*proof*⟩

lemma *euler_e_gt_one [simp]:*
"(𝔢::real star) > 1"
⟨*proof*⟩

lemma *euler_e_ne_one: "(𝔢::hypreal) ≠ 1"*
⟨*proof*⟩

lemma *"eulerK 𝔢 (1 / hypreal_of_hypnat whn) = 1"*
⟨*proof*⟩

lemma *euler_e_ge_two: "(𝔢::hypreal) ≥ 2"*
⟨*proof*⟩

lemma *euler_e_power_ge_two_power: "(𝔢::hypreal) ^ n ≥ 2 ^ n"*
⟨*proof*⟩

lemma *euler_e_power_bound:*
assumes *"a ∈ HFinite" "a ≥ 1"*
shows *"∃n. (𝔢::hypreal) ^ n ≥ a"*
⟨*proof*⟩

lemma *hpow_euler_e_eulerK_approx:*
assumes *"a ∈ HFinite"*
and *"a > 1"*
shows *"𝔢 hpow (eulerK a (1/of_hypnat whn)) ≈ a"*
⟨*proof*⟩

lemma *euler_e_hpow_not_approx_one_pos:*
assumes *"d ∈ HFinite" "d > 0" "d ∉ Infinitesimal"*
shows *"¬ (𝔢 hpow d ≈ 1)"*
⟨*proof*⟩

lemma *euler_e_hpow_not_approx_one_neg:*
assumes *"d ∈ HFinite" "d < 0" "d ∉ Infinitesimal"*
shows *"¬ (𝔢 hpow d ≈ 1)"*
⟨*proof*⟩

lemma *euler_e_hpow_not_approx_one:*
assumes *"d ∈ HFinite" "d ≠ 0" "d ∉ Infinitesimal"*
shows *"¬ (𝔢 hpow d ≈ 1)"*
⟨*proof*⟩

corollary *euler_e_hpow_approx_one_iff_Infinitesimal:*
assumes *"d ∈ HFinite"*
shows *"(𝔢 hpow d ≈ 1) ⟷ (d ∈ Infinitesimal)"*
⟨*proof*⟩

lemma *euler_e_hpow_approx_inject:*
assumes *"𝔢 hpow x ≈ 𝔢 hpow y"*
and *"x ∈ HFinite"*
and *"y ∈ HFinite"*
shows *"x ≈ y"*
⟨*proof*⟩

11.6 Hypernatural logarithm

Let us define the hypernatural logarithm of a hyperreal x

definition *Ln* **where**
"Ln x ≡ hLog 𝔢 x"

lemma *Ln_euler_e [simp]: "Ln 𝔢 = 1"*
⟨*proof*⟩

lemma *hpow_e_Ln [simp]:*
assumes *"a > 0"*
shows *"𝔢 hpow (Ln a) = a"*
⟨*proof*⟩

lemma *Ln_ge_zero: "a ≥ 1 ⟹ Ln a ≥ 0"*
⟨*proof*⟩

lemma *Ln_inverse: "x > 0 ⟹ Ln (inverse x) = - Ln x"*
⟨*proof*⟩

lemma *Ln_less_zero: "a > 0 ⟹ a < 1 ⟹ Ln a < 0"*
⟨*proof*⟩

lemma *Ln_HFinite:*
assumes *"a ∈ HFinite - Infinitesimal" "a > 0"*
shows *"Ln a ∈ HFinite"*
⟨*proof*⟩

lemma *Ln_eulerK_approx:*
assumes *"a ∈ HFinite" "a > 1"*
shows *"Ln a ≈ eulerK a (1 / hypreal_of_hypnat whn)"*
⟨*proof*⟩

This is just to show that natural logarithm would have the expected definition (if we take the standard part of the RHS)

lemma *Ln_approx_def:*
assumes *"a ∈ HFinite" "a > 1"*
shows *"Ln a ≈ (a hpow (1 / hypreal_of_hypnat whn) - 1) * of_hypnat whn"*
⟨*proof*⟩

Euler's power series in terms of hypernatural logarithm.

lemma *hpow_powseries_Ln:*
assumes *"a ∈ HFinite"*
and *"a > 1"*
and *"z ∈ HFinite"*
shows *"a hpow z ≈ (∑ h i∈{0..whn}. (Ln a * z) pow i / hfact i)"*
⟨*proof*⟩

11.7 Linking the Euler and Isabelle library exponential functions.

Just for the sake of it: we link our current derivation of the exponential series to the Isabelle library one.

lemma *exp_NSsums: "(λn. z ^ n / fact n) NSsums (exp z)"*
⟨*proof*⟩

lemma *hypersum_exp_approx:*
assumes *"N ∈ HNatInfinite"*
shows *"hypersum (*f* (λn. z ^ n / fact n)) {0..<N} ≈ of_real (exp z)"*
⟨*proof*⟩

lemma *starfun_exp_of_real: "(*f* exp) (of_real z) = of_real (exp z)"*
⟨*proof*⟩

lemma *starfun_exp_term_eq:*
*"(*f* (λn. z ^ n / fact n)) n = (of_real z) pow n / hfact n"*
⟨*proof*⟩

lemma *HyperSummable_starfun_real_exp:*
*"HyperSummable (*f* (λn. (z::real) ^ n / fact n))"*
⟨*proof*⟩

lemma *hypersum_starfun_exp_approx_euler_sum:*
*"hypersum (*f* (λn. z ^ n / fact n)) {0..N} ≈*
hypersum (λi. (of_real z) pow i / hfact i) {0..N}"
⟨*proof*⟩

Theorem linking the two representations of the exponential series.

theorem *euler_e_hpow_approx_starfun_exp:*
*"𝔢 hpow (of_real z) ≈ (*f* exp) (of_real z)"*
⟨*proof*⟩

corollary *euler_e_hpow_approx_exp:*
"𝔢 hpow (hypreal_of_real z) ≈ hypreal_of_real (exp z)"
⟨*proof*⟩

The standard part of our Euler exponential is (the hyperreak embedding of) Isabelle's exponential funtion

corollary *st_euler_e_hpow_eq_exp:*
"st(𝔢 hpow (hypreal_of_real z)) = hypreal_of_real (exp z)"
⟨*proof*⟩

We can show that the standard part of our Euler 𝔢 is (literally!) the real e.

corollary *euler_e_approx_exp1: "(𝔢::hypreal) ≈ of_real (exp 1)"*
⟨*proof*⟩

corollary *st_euler_e_approx_exp1: "st (𝔢::hypreal) = of_real (exp 1)"*
⟨*proof*⟩

11.8 Derivative of Euler's exponential function.

We can also show that our Euler exponential function is its own derivative.

lemma *euler_e_hpow_eulerK_approx_base:*
assumes *inf_e: "e ∈ Infinitesimal"*
and *e_pos: "e > 0"*
and *a_HF: "a ∈ HFinite"*
and *a_gt1: "a > 1"*
shows *"𝔢 hpow (eulerK a e) ≈ a"*
⟨*proof*⟩

lemma *eulerK_approx_Ln_pos:*
assumes *"e ∈ Infinitesimal" "e > 0" "a ∈ HFinite" "a > 1"*
shows *"eulerK a e ≈ Ln a"*
⟨*proof*⟩

Extend the previous lemma to deal with negative infinitesimals.

lemma *eulerK_approx_Ln_neg:*
assumes *"e ∈ Infinitesimal" "e < 0" "a ∈ HFinite" "a > 1"*
shows *"eulerK a e ≈ Ln a"*
⟨*proof*⟩

The combined result for any nonzero infinitesimal.

theorem *eulerK_approx_Ln:*
assumes *"e ∈ Infinitesimal" "e ≠ 0" "a ∈ HFinite" "a > 1"*
shows *"eulerK a e ≈ Ln a"*
⟨*proof*⟩

lemma *hpow_difference_quotient:*
assumes *"a > 0"*
shows *"(a hpow (x + e) - a hpow x) / e = a hpow x * eulerK a e"*
⟨*proof*⟩

Nonstandard derivative of the general exponential

theorem *NSderivative_hpow:*
assumes *a_HF: "a ∈ HFinite"*
and *a_gt1: "a > 1"*
and *x_HF: "x ∈ HFinite"*
and *e_inf: "e ∈ Infinitesimal"*
and *e_ne0: "e ≠ 0"*
shows *"(a hpow (x + e) - a hpow x) / e ≈ Ln a * (a hpow x)"*
⟨*proof*⟩

Euler's exponential function is its own (nonstandardly-expressed) derivative (as expected).

theorem *NSderivative_euler_e:*
assumes *"x ∈ HFinite" "e ∈ Infinitesimal" "e ≠ 0"*
shows *"(𝔢 hpow (x + e) - 𝔢 hpow x) / e ≈ 𝔢 hpow x"*
⟨*proof*⟩

lemma *NSderivative_hpow_at_zero:*
assumes *"a ∈ HFinite" "a > 1" "e ∈ Infinitesimal" "e ≠ 0"*
shows *"(a hpow e - 1) / e ≈ Ln a"*
⟨*proof*⟩

The derivative of 𝔢 hpow x at x=0 is 1.

lemma *NSderivative_euler_e_at_zero:*
assumes *"e ∈ Infinitesimal" "e ≠ 0"*
shows *"(𝔢 hpow e - 1) / e ≈ 1"*
⟨*proof*⟩

lemma *"(𝔢 hpow ε - 1) / ε ≈ 1"*
⟨*proof*⟩

end