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"
  by (simp add: fallfactpow_def)

lemma fallfactpow_1[simp]: "fallfactpow a 1 = a" by (simp add: fallfactpow_def)
lemma fallfactpow_Suc0[simp]: "fallfactpow a (Suc 0) = a" by (simp add:
fallfactpow_def)

lemma fallfactpow_Suc_setprod: "fallfactpow a (Suc n) = prod (λn. a -
of_nat n) {0 .. n}"
  by (simp add: fallfactpow_def)
```

1.1 Falling factorial expressed recursively

```
lemma fallfactpow_Suc: "fallfactpow a (Suc n) = fallfactpow a n * (a
- of_nat n)"
proof-
  {assume "n=0" then have ?thesis by simp}
  moreover
  {fix m assume m: "n = Suc m"
    have ?thesis  unfolding m fallfactpow_Suc_setprod prod.atLeast0_atMost_Suc
by blast }
  ultimately show ?thesis by (cases n, auto)
qed


lemma fallfactpow_rec: "fallfactpow a (Suc n) = a * fallfactpow (a -
1) n"
proof-
  {assume "n=0"
    then have ?thesis
      by (simp add: fallfactpow_Suc_setprod)}
  moreover
  {assume n0: "n ≠ 0"
    have th0: "finite {1 .. n}" "0 ∉ {1 .. n}" by auto
    have eq: "insert 0 {1 .. n} = {0..n}" by auto
    have th1: "(∏n∈{1::nat..n}. a - of_nat n) =
      (∏n∈{0::nat..n - 1}. a - (1 + of_nat n))"
      apply (rule prod.reindex_cong [where l = Suc])
      using n0 by (auto simp add: fun_eq_iff field_simps)
    have ?thesis
```

```
      apply (simp add: fallfactpow_def)
    unfolding prod.insert[OF th0, unfolded eq]
    using th1 by (simp add: field_simps)}
  ultimately show ?thesis
    by blast
qed

lemma fallfactpow_base_zero[simp]: "fallfactpow 0 (Suc n) = 0"
by (simp add: fallfactpow_rec)

lemma fallfactpow_fact: "of_nat(fact n) =  fallfactpow (of_nat n) n"
proof (induct n)
  case 0
  then show ?case
    by simp
next
  case (Suc n)
  then show ?case
    by (metis add_diff_cancel_left' fact_Suc fallfactpow_rec id_apply

        of_nat_Suc of_nat_eq_id of_nat_mult)
qed

lemma fallfactpow_altdef:
  "fallfactpow a k = (∏i<k. a - of_nat i)"
proof (induct k)
  case 0
  then show ?case by simp
next
  case (Suc k)
  then show ?case
    by (simp add: fallfactpow_Suc)
qed
```

The falling factorial and Pochammer functions

```
lemma pochhammer_fallfactpow_eq:
  "pochhammer a n = fallfactpow (a + of_nat n - 1) n"
proof (induct n)
  case 0
  then show ?case
    by simp
next
  case (Suc n)
  then show ?case
    by (simp add: fallfactpow_rec pochhammer_rec')
qed

lemma fallfactpow_of_nat_eq_0_lemma [simp]:
  assumes kn: "k > n"
```

```
  shows "fallfactpow (of_nat n :: 'a:: idom) k = 0"
  by (metis cancel_comm_monoid_add_class.diff_cancel fallfactpow_altdef

      finite_lessThan kn lessThan_iff prod_zero_iff)

lemma setprod_zero_of_nat_diff_lemma:
  "a < k ⟹ (∏ i<k. of_nat (a - i)) = (0::'a::comm_semiring_1)"
by (force intro: prod_zero)

lemma fallfactpow_of_nat_altdef:
  "fallfactpow (of_nat a :: 'a :: idom) k = ((∏ i<k. of_nat (a - i)) ::
'a :: idom)"
proof (cases "k > a")
  case True
  then show ?thesis
    by (simp add:  setprod_zero_of_nat_diff_lemma)
next
  case False
  then show ?thesis
    by (auto intro!: prod.cong simp add: fallfactpow_altdef)
qed

lemma fallfactpow_ge_zero:
  "of_nat k < j ⟹ 0 ≤ fallfactpow (j::'a::{linordered_idom}) k"
proof (induct k)
  case 0
  then show ?case
    by simp
next
  case (Suc k)
  then show ?case
    by (simp add: fallfactpow_Suc)
qed

lemma fallfactpow_le_power_self [simp]:
  "fallfactpow (of_nat k ::'a::{linordered_idom}) k ≤ of_nat k ^ k"
proof(induct k)
  case 0
  then show ?case
    by simp
next
  case (Suc k)
  then show ?case
  proof (auto simp add: fallfactpow_rec mult_le_cancel_left)
    assume "(0::'a::linordered_idom) < 1 + of_nat k"
       have "of_nat k ^ k ≤ ((1::'a::linordered_idom) + of_nat k) ^ k"
         by (simp add: power_mono)
       then show "fallfactpow (of_nat k) k ≤ ((1::'a::linordered_idom)
+ of_nat k) ^ k"
```

```
        using Suc by linarith
    next
      assume "1 + of_nat k < (0::'a::linordered_idom)"
      then show "(1 + of_nat k) ^ k ≤ fallfactpow (of_nat k) k"
        by (metis of_nat_1 of_nat_add of_nat_less_0_iff)
  qed
qed


lemma fallfactpow_le_power:
  assumes "of_nat k ≤ j"
  shows "fallfactpow (j::'a::{linordered_idom}) k ≤ j ^ k"
proof cases
  assume "of_nat k = j"
  then show ?thesis
    using fallfactpow_le_power_self by blast
next
  assume "of_nat k ≠ j"
  then have "of_nat k < j"
    by (simp add: assms order.not_eq_order_implies_strict)
  then show ?thesis
  proof(induct k)
    case 0
    then show ?case
      by simp
  next
    case (Suc k)
    then have "fallfactpow j k ≤ j ^ k"
      by simp
    moreover have "of_nat k + 1 < j"
      using Suc.prems by auto
    ultimately have "(j - of_nat k) * fallfactpow j k ≤ j * j ^ k"
      by (auto intro!: mult_mono' fallfactpow_ge_zero)
    then show ?case
      by (simp add: fallfactpow_Suc mult.commute)
  qed
qed

lemma fallfactpow_ge_zero' [simp]:
  "fallfactpow (of_nat n) k ≥ (0::'a::linordered_idom)"
  by (metis eq_iff fallfactpow_fact fallfactpow_ge_zero fallfactpow_of_nat_eq_0_lemma

      less_imp_of_nat_less less_linear of_nat_0_le_iff)
```

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"
```

```
  by (simp add: binomial_gbinomial gbinomial_pochhammer' pochhammer_fallfactpow_eq)

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(induction n)
  case 0 show ?case by simp
next
  case (Suc n)
  have "(∑k = 0..Suc n.
               of_nat (Suc n choose k) * fallfactpow a (Suc n - k) * fallfactpow
b k) =
        fallfactpow a (Suc n) +
        (∑k = 1..Suc n.
               of_nat (Suc n choose k) * fallfactpow a (Suc n - k) * fallfactpow
b k)"
          by (simp add: sum.atLeast_Suc_atMost)
  moreover have "... = fallfactpow a (Suc n) +
                  (∑k = 1..Suc n. (of_nat(n choose (k - 1)) + of_nat(n
choose k)) * fallfactpow a (Suc n - k) * fallfactpow b k)"
          by (simp add: sum.shift_bounds_cl_Suc_ivl del: sum.cl_ivl_Suc)
  moreover have "... = fallfactpow a (Suc n) +
                  (∑k = 1..Suc n. of_nat(n choose (k - 1)) * fallfactpow
a (Suc n - k) * fallfactpow b k) +
                  (∑k = 1..Suc n. of_nat(n choose k) * fallfactpow a
(Suc n - k) * fallfactpow b k)"
          by (simp add: sum.distrib [symmetric] distrib_right del: sum.cl_ivl_Suc)
  moreover have "... = fallfactpow a (Suc n) +
                  (∑k = 0..n. of_nat(n choose k) * fallfactpow a (n -
k) * fallfactpow b (k + 1)) +
                  (∑k = 1..Suc n. of_nat(n choose k) * fallfactpow a
(Suc n - k) * fallfactpow b k)"
          by (simp add: sum.shift_bounds_cl_Suc_ivl  del: sum.cl_ivl_Suc)
  moreover have "... = fallfactpow a (Suc n) +
                  (∑k = 0..n. of_nat(n choose k) * fallfactpow a (n -
k) * fallfactpow b (k + 1)) +
                  (∑k = 1..Suc n. of_nat(n choose k) * fallfactpow a
(Suc n - k) * fallfactpow b k)"
          by (simp add: sum.shift_bounds_cl_Suc_ivl  del: sum.cl_ivl_Suc)
  moreover have "... = (∑k = 0..n. of_nat(n choose k) * fallfactpow
a (n - k) * fallfactpow b (k + 1)) +
                  (∑k = 0..Suc n. of_nat(n choose k) * fallfactpow a
(Suc n - k) * fallfactpow b k)"
          by (simp add: sum.atLeast_Suc_atMost)
  moreover have "... = (∑k = 0..n. of_nat(n choose k) * fallfactpow
a (n - k) * fallfactpow b (k + 1)) +
                    (∑k = 0..n. of_nat(n choose k) * fallfactpow a
(Suc (n - k)) * fallfactpow b k)"
```

```
          by (simp add:  binomial_eq_0 Suc_diff_le)
  moreover have "... = (∑k = 0..n. of_nat(n choose k) * fallfactpow
a (n - k) * fallfactpow b k * (b - of_nat k)) +
                      (∑k = 0..n. of_nat(n choose k) * fallfactpow a
(n - k) * fallfactpow b k * (a - of_nat (n - k)))"
          by (simp add:  fallfactpow_Suc algebra_simps)
  moreover have "... = (∑k = 0..n. of_nat(n choose k) * fallfactpow
a (n - k) * fallfactpow b k *
                        ((a - of_nat n + of_nat k) + (b - of_nat k)))"
          by (simp add: sum.distrib [symmetric] algebra_simps)
  moreover have "... = (∑k = 0..n. of_nat(n choose k) * fallfactpow
a (n - k) * fallfactpow b k *
                        (a + b - of_nat n))"
          by (simp add: algebra_simps)
  moreover have "... = (∑k = 0..n. of_nat(n choose k) * fallfactpow
a (n - k) * fallfactpow b k) * (a + b - of_nat n)"
          by (simp add: sum_distrib_right)
  moreover have "... = fallfactpow (a + b) n * (a + b - of_nat n)" us-
ing Suc  by metis
  ultimately show ?case by (metis fallfactpow_Suc)
qed


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 -
   have "(r + s)^ k  =  (∑i = 0..k. of_nat (k choose i) * r ^ i * s
^ (k - i))"
     using binomial_ring atLeast0AtMost by auto
   then have  "(r + s)^ k = (∑i = 0..k.  fact k / (fact i * fact (k
- i)) * r ^ i * s ^ (k - i))"
     using binomial_fact sum.cong atLeastAtMost_iff by (metis (no_types,
lifting))
   then show ?thesis by (simp add: field_simps sum_distrib_left)
qed

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)"
  by (metis add.commute atLeastAtMost_iff sum.atLeastAtMost_rev sum.cong)

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 -
```

```
  have "fallfactpow (r + s) k  =  (∑i = 0..k. of_nat (k choose i) *
fallfactpow r i * fallfactpow s (k - i))"
    using binomial_fallfactpow_ring
     by (metis (no_types, lifting) add.left_neutral binomial_symmetric
diff_diff_cancel
        natsum_reverse_index)
  then have "fallfactpow (r + s) k =
           (∑i = 0..k.  fact k / (fact i * fact (k - i)) * fallfactpow
r i * fallfactpow s (k - i))"
     using binomial_fact sum.cong atLeastAtMost_iff  by (metis (no_types,
lifting))
  then show ?thesis by (simp add: field_simps sum_distrib_left)
qed

lemma gbinomial_fallfactpow: "a gchoose n = fallfactpow a n / fact n"
  by (simp add: gbinomial_pochhammer' pochhammer_fallfactpow_eq)

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)))"
by (simp add: binomial_fallfactpow_vandermonde gbinomial_fallfactpow)

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 {}) = {}"
by (transfer empty_def) simp

lemma Iset_star_n_empty [simp]: "Iset (star_n (λn. {})) = {}"
  by (simp add: star_of_def [symmetric])

lemma Iset_eq_cancel:
  "(Iset (star_n X) = Iset (star_n Y)) = (eventually (λn. X n = Y n) 𝒰)"
  by (simp add: transfer_set_eq)

lemma Collect_starP_starset_eq: "{x. (*p* P) x} = *s* {x. P x}"
by transfer simp
```

Hypernat.thy lemmas

```
lemma Nats_hypnat_of_nat_iff: "(n ∈ Nats) = (∃m. n = hypnat_of_nat m)"
```

```
  by (auto simp add: Nats_def)

lemma HNatInfinite_add_one_cancel:
   "N + 1 ∈ HNatInfinite ⟹ N ∈ HNatInfinite"
  by (drule HNatInfinite_diff [of _ 1], auto)

lemma of_hypnat_of_nat [simp]: "of_hypnat(hypnat_of_nat n) = of_nat n"
  by (metis of_hypnat_def star_of_nat_def starfun_star_of)

lemma of_nat_hypreal_of_hypnat:
     "of_nat n = of_hypnat (hypnat_of_nat n)"
  by simp

lemma hSuc_eq_add_one:
  "⋀n. hSuc n = n + 1"
by transfer simp

lemma HNatInfinite_hSuc_diff_one [simp]:
  "N ∈ HNatInfinite ⟹ hSuc (N - 1) = N"
by (metis hSuc_eq_add_one hypnat_le_add_diff_inverse2 one_le_HNatInfinite)

lemma HNatInfinite_atLeastAtMost_hSuc:
  "N ∈ HNatInfinite ⟹ {M..<N} = {M..<hSuc(N - 1)}"
by (metis HNatInfinite_hSuc_diff_one)
```

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)"
by (auto simp only: of_nat_hypreal_of_hypnat of_hypnat_less_iff)

lemma hypreal_of_nat_le_hypreal_of_hypnat_iff:
      "⋀a b.(hypreal_of_nat a ≤ hypreal_of_hypnat b) = (hypnat_of_nat
a ≤  b)"
  by (auto simp only: of_nat_hypreal_of_hypnat of_hypnat_le_iff)

lemma hyperpow_zero [simp]: "x pow 0 = 1"
by (metis hyperpow_hypnat_of_nat power_0 star_of_simps(9))

lemma hyperpow_of_nat: "⋀x. x pow of_nat n = x ^ n"
by transfer simp

lemma hyperpow_hSuc_zero [simp]:
  "⋀N. (0::'a::{power, semiring_0} star) pow hSuc N = 0"
by transfer simp

lemma starfun_power: "⋀n. (*f* (^) x) n = star_of x pow n"
by transfer simp

lemma hyperpow_diff:
```

```
  "⋀x n k. n ≥ k ⟹ x ≠ 0 ⟹ ((x::'a::field star) pow n)/(x pow k)
= x pow (n - k)"
  by transfer (metis power_diff)

lemma hyperpow_divide:
  "⋀x y n. ((x::'a::field star)/y) pow n = (x pow n)/(y pow n)"
by transfer (metis power_divide)

lemma hyperpow_diff_cancel:
  "k ≤ n ⟹ x pow k * (x::'a::monoid_mult star) pow (n - k) = x pow
n"
  by (simp add: hyperpow_add [symmetric])

lemma hyperpow_0_left:
  "⋀n. 0 pow n = (if n = 0 then 1 else (0::'a::{power, semiring_1} star))"
by transfer (simp add: power_0_left)

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)"
  by transfer simp

lemma hyperpow_less_zero_eq:
  "⋀x n. (x pow n < (0::'a::{linordered_idom} star)) = (¬ 2 dvd n ∧ x
< 0)"
by transfer (metis power_less_zero_eq)

lemma star_n_divide: "star_n X / star_n Y = star_n (λn. X n / Y n)"
  by (simp add: star_divide_def starfun2_star_n)
```

NSA.thy lemmas

```
lemma approx_zero_abs_zero_iff [iff]:
  "(abs x ≈ 0) = ((x::hypreal) ≈ 0)"
by (metis Infinitesimal_hrabs_iff mem_infmal_iff)

lemma Infinitesimal_divide_HFinite:
  "⟦ (x::'a::real_normed_div_algebra star) ∈ Infinitesimal; y ∈ HFinite
- Infinitesimal ⟧
      ⟹ x/y ∈ Infinitesimal"
by (metis HFinite_inverse2 Infinitesimal_HFinite_mult divide_inverse)

lemma approx_1_not_Infinitesimal:
  "x ≈ (1 ::'a::real_normed_div_algebra star) ⟹ x ∉ Infinitesimal"
  using approx_sym approx_trans mem_infmal_iff one_not_Infinitesimal
  by blast

lemma HFinite_add_imp_HFinite:
  "⟦ x ≥ (0::hypreal); y ≥ 0; x + y ∈ HFinite ⟧ ⟹ y ∈ HFinite"
```

```
  using HFinite_not_HInfinite HInfinite_add_ge_zero2 not_HFinite_HInfinite

  by blast

lemma Infinitesimal_hrealpow_cancel [simp]:
  assumes "n > 0"
  shows "(x ^ n ∈ Infinitesimal) = ((x::'a::real_normed_div_algebra star)
∈ Infinitesimal)"
proof (insert assms, induct n)
  case 0
  then show ?case
    by simp
next
  case (Suc n)
  then show ?case
    using Infinitesimal_mult not_Infinitesimal_mult
    by force
qed

lemma hrealpow_approx:
   "⟦ (x::'a::{real_normed_algebra, one, monoid_mult} star) ≈ b;  b ∈
HFinite ⟧ ⟹ x ^ n ≈ b ^ n"
by (induct n, auto dest: hrealpow_HFinite approx_mult_HFinite [of x])

lemma hrealpow_approx_one:
   "(x::'a::{real_normed_algebra, one, monoid_mult} star) ≈ 1 ⟹ x ^
n ≈ 1"
by (auto dest: hrealpow_approx)

lemma zero_not_HInfinite [simp]: "0 ∉ HInfinite"
by (simp add: HFinite_not_HInfinite)

lemma HInfinite_hyperpow_not_zero:
  "(x::'a::{real_normed_field} star) ∈ HInfinite ⟹ x pow n ≠ 0"
  by (metis hyperpow_not_zero zero_not_HInfinite)

lemma HFinite_divide:
  "⟦ (x::'a::real_normed_div_algebra star) ∈ HFinite; y ∉ Infinitesimal
⟧ ⟹ x/y ∈ HFinite"
by (metis HFinite_mult Infinitesimal_inverse_HFinite divide_inverse)

lemma HFinite_hyperpow_HFinite:
  "⟦ (x::'a::{monoid_mult, real_normed_algebra} star) ∈ HFinite; n ∈ ℕ
⟧ ⟹ x pow n ∈ HFinite"
  by (metis Nats_cases hrealpow_HFinite hyperpow_hypnat_of_nat of_nat_eq_star_of)

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"
by (metis HFinite_a approx_HFinite approx_ab hrealpow_approx hyperpow_of_nat)

lemma Infinitesimal_approx_hyperpow:
      assumes Infinitesimal_a: "a ∈ Infinitesimal"
      and approx_ab: "(a::hypreal) ≈ b"
      shows "a pow n ≈ b pow n"
  by (metis Infinitesimal_a Infinitesimal_approx Infinitesimal_hyperpow
Infinitesimal_monad_zero_iff
      approx_ab approx_mem_monad_zero approx_monad_iff hyperpow_zero hypnat_neq0_conv)

lemma hnorm_hypreal_of_hypnat [simp]:
  "⋀n. hnorm (of_hypnat n::'a::real_normed_algebra_1 star) = of_hypnat
n"
  by transfer simp

lemma HFinite_of_nat [simp]: "of_nat n ∈ HFinite"
by (metis HFinite_star_of star_of_nat_def)

lemma HInfinite_def2: "HInfinite = {x. ∀n ∈ Nats. hypreal_of_hypnat
n < hnorm x}"
proof (auto simp add: HInfinite_def SHNat_eq SReal_def)
   fix x::"'a :: real_normed_vector star" and N
   assume "∀r. (∃ra. r = hypreal_of_real ra) ⟶ r < hnorm x"
   then show "hypreal_of_nat N < hnorm x"
     using star_of_nat_def by blast
next
   fix x::"'a :: real_normed_vector star" and ra
   assume "∀n. (∃N. n = hypnat_of_nat N) ⟶ hypreal_of_hypnat n < hnorm
x"
   then show "hypreal_of_real ra < hnorm x"
     by (metis of_nat_hypreal_of_hypnat less_trans reals_Archimedean2

         star_of_less star_of_nat_def)
 qed

lemma HFinite_def2: "HFinite = {x. ∃n ∈ Nats. hnorm x < hypreal_of_hypnat
n}"
proof (auto simp add: HFinite_def)
  fix x::"'a :: real_normed_vector star" and r
  assume r: "r ∈ ℝ" and hnormx: "hnorm x < r"
  then obtain s where "r = of_real s"
    using Reals_cases by blast
  moreover obtain n where "s < of_nat n"
    using reals_Archimedean2 by blast
  ultimately have "hnorm x < hypreal_of_hypnat (of_nat n)"
    using hnormx
    by (metis dual_order.strict_trans of_hypnat_def of_nat_eq_star_of
of_real_eq_star_of star_of_less
```

```
        starfun_star_of)
  then show "∃n∈ℕ. hnorm x < hypreal_of_hypnat n"
    by fastforce
next
  fix x::"'a :: real_normed_vector star" and n
  assume n: "n ∈ ℕ" and "hnorm x < hypreal_of_hypnat n"
  then have "hnorm x < hypreal_of_hypnat n"
    by blast
  moreover have "hypreal_of_hypnat n ∈ ℝ"
    using n by (auto simp add: SHNat_eq)
  ultimately show "∃r∈ℝ. hnorm x < r"
    by fastforce
qed

lemma HFinite_less_Nat_Ex:
  "(x::hypreal) ∈ HFinite ⟹ ∃n∈Nats. x < n"
  unfolding Nats_def HFinite_def2
  by simp (metis abs_ge_self order_less_le xt1(10))

lemma HNatInfinite_of_hypnat_HInfinite:
  "x ∈ HNatInfinite ⟹ (of_hypnat x :: 'a::real_normed_algebra_1 star)
∈ HInfinite"
proof (auto simp add: HInfinite_def2 Nats_def)
  fix n
  assume "x ∈ HNatInfinite"
  then show "hypreal_of_nat n < hypreal_of_hypnat x"
    by (metis hypreal_of_nat_less_hypreal_of_hypnat_iff star_of_less_HNatInfinite)
qed

lemma HInfinite_ge_HNatInfinite:
  "⟦ n ∈ HNatInfinite; hypreal_of_hypnat n ≤ x ⟧ ⟹ x ∈ HInfinite"
  using HInfinite_ge_HInfinite HNatInfinite_of_hypnat_HInfinite of_hypnat_0_le_iff

  by blast

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

  then show "n ∈ HNatInfinite"
    by (auto simp add: HInfinite_def2 HNatInfinite_def)
next
  assume "n ∈ HNatInfinite"
  then show "(of_hypnat n :: 'a::real_normed_algebra_1 star) ∈ HInfinite"

    using HNatInfinite_of_hypnat_HInfinite by blast
qed
```

```
lemma HNatInfinite_inverse_of_hypnat_Infinitesimal_iff:
  "N ≠ 0 ⟹ (1/hypreal_of_hypnat N ∈ Infinitesimal) = (N ∈ HNatInfinite)"
  by (simp add: HNatInfinite_of_hypnat_HInfinite_iff divide_inverse inverse_Infinitesimal_i

lemma Nats_hypreal_of_hypnat_HFinite:
  "n ∈ Nats ⟹ of_hypnat n ∈ HFinite"
  using Nats_hypnat_of_nat_iff by auto

lemma approx_divide_HFinite_diff_Infinitesimal:
   "⟦(x :: 'a::{real_normed_div_algebra, field} star) ≈ y;
     z ∈ HFinite - Infinitesimal
    ⟧ ⟹ x/z ≈ y/z"
by (metis HFinite_inverse2 approx_mult1 divide_inverse)

lemma HFinite_diff:
  "⟦ x ∈ HFinite; y ∈ HFinite ⟧ ⟹ x - y ∈ HFinite"
by (metis HFinite_add HFinite_minus_iff add_uminus_conv_diff)

lemma Infinitesimal_interval2a:
     "⟦ e ∈ Infinitesimal; e' ∈ Infinitesimal;
        hnorm e' ≤ hnorm x ; hnorm x ≤ hnorm e ⟧ ⟹ x ∈ Infinitesimal"
  by (auto simp add: Infinitesimal_def)

lemma Infinitesimal_HInfinite_divide:
  "⟦ (x::'a::{real_normed_div_algebra, field} star) ∈ HFinite; y ∈ HInfinite
⟧ ⟹ x/y ∈ Infinitesimal"
by (simp add: HInfinite_inverse_Infinitesimal Infinitesimal_HFinite_mult2
divide_inverse)

lemma approx_divide_HInfinite:
  "⟦(x :: 'a::{real_normed_div_algebra, field} star) ≈ y; z ∈ HInfinite⟧
⟹ x/z ≈ y/z"
by (metis Infinitesimal_HInfinite_divide approx_minus_iff approx_star_of_HFinite

          diff_divide_distrib mem_infmal_iff star_zero_def)

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 (rule ccontr)
  assume "y ∉ HFinite"
  then have HInfinite_y: "y ∈ HInfinite"
    using HFinite_HInfinite_iff
    by blast
  have HFinite_x: "x ∈ HFinite"
    by (metis approx_1 approx_star_of_HFinite star_one_def)
```

```
  have "x ∉ Infinitesimal"
    using approx_1 approx_1_not_Infinitesimal
    by blast
  then have "x * y ∈ HInfinite"
    using  HInfinite_y HFinite_x HInfinite_HFinite_not_Infinitesimal_mult2

           HInfinite_diff_HFinite_Infinitesimal_disj
    by blast
  thus "False"
    using HFinite_not_HInfinite HFinite_prod
    by blast
qed
```

Star.thy lemmas

```
instance star :: (semiring_modulo) semiring_modulo
  by (intro_classes, transfer) (simp add: div_mult_mod_eq)

declare of_bool_def [transfer_unfold]

instance star :: (semiring_parity) semiring_parity
proof
  show "⋀a ::'a :: semiring_parity star. a mod 2 = of_bool (¬ 2 dvd a)"
    by transfer (simp add: mod2_eq_if)
next
  show "¬ (2::'a :: semiring_parity star) dvd 1"
    by transfer simp
next
  show "⋀a ::'a :: semiring_parity star. 2 dvd a ⟹ (1 + a) div 2 =
a div 2"
    by transfer simp
qed

lemma starfun_starfun_n_eq:
  "*f* f = *fn* (λn. f)"
  by (metis starfun_n_starfun)

lemma starfun_n_zero:
  "(*fn* (λm n::'b. 0::'a::zero)) = (λn. star_of 0)"
  by (metis starfun_const_fun starfun_starfun_n_eq)

lemma InternalFuns_starfun_n [simp]: "*fn* f ∈ InternalFuns"
by (force simp add: InternalFuns_def)

lemma InternalFuns_starfun [simp]: "*f* f ∈ InternalFuns"
by (simp add: starfun_starfun_n_eq)

lemma InternalFuns_starfun2 [simp]: "(*f2* f) a ∈ InternalFuns"
  by (metis InternalFuns_starfun_n star_cases starfun2_def starfun_n_def)
```

```
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 (cases z)
  case (star_n X)
   assume Z: "z = star_n X"
   then have "(λn. (*f2* (^)) ((*fn* f) n) ((*fn* g) n)) z = (*f2* (^))
((*fn* f) (star_n X)) ((*fn* g) (star_n X))"
     by simp
   also have "... = (*f2* (^)) (star_n (λn. f n (X n))) (star_n (λn. g
n (X n)))"
     by (simp add: Ifun_star_n starfun_n_def)
   also have "... = star_n (λn. f n (X n) ^ g n (X n))"
     by (simp add: starfun2_star_n)
   also have "... =  (*fn* (λi x. f i x ^ g i x)) (star_n X)"
     by (simp add: Ifun_star_n starfun_n_def)
   finally show ?thesis using Z [symmetric] by simp
 qed

lemma InternalFuns_hSuc [simp]: "hSuc ∈ InternalFuns"
by (simp add: hSuc_def)

lemma InternalSets_starset_n [simp]: "*sn* X ∈ InternalSets"
by (auto simp add: InternalSets_def)

lemma InternalSets_singleton [simp]: "{x} ∈ InternalSets"
proof -
  {fix X assume "x = star_n X"
    have "{star_n X} = Iset (star_n (λn. {X n}))"
    proof
       show "{star_n X} ⊆ Iset (star_n (λn. {X n}))"
         by (simp add: Iset_star_n)
    next
      show "Iset (star_n (λn. {X n})) ⊆ {star_n X}"
      proof
        fix y
        assume yin: "y ∈ Iset (star_n (λn. {X n}))"
        then show  "y ∈ {star_n X}"
          using yin
          by (auto intro: star_cases [of y]
                    simp add: Iset_def starP2_star_n star_n_eq_iff)
        qed
      qed
   }
   then show ?thesis
     by (force intro: star_cases simp add: InternalSets_def starset_n_def)
 qed

lemma starset_n_singleton [simp]: "*sn* (λn. {X n}) =  {star_n X}"
proof  (auto simp add: starset_n_def)
```

```
  fix x
  assume xinX: "x ∈ Iset (star_n (λn. {X n}))"
  then show "x = star_n X"
  proof (cases x)
    case (star_n Y)
    then show ?thesis
      using xinX by (auto simp add: Iset_star_n star_n_eq_iff)
  qed
next
  show "star_n X ∈ Iset (star_n (λn. {X n}))"
    by (simp add: Iset_star_n)
qed

lemma starset_n_singleton2:"*sn* (λn. {X n}) = {x. x = star_n X}"
by simp

lemma starset_n_empty [simp]: "*sn* (λn. {}) = {}"
by (simp add: starset_n_def )

lemma InternalSets_empty [simp]: "{} ∈ InternalSets"
  by (metis InternalSets_starset_n starset_n_empty)

lemma starset_n_subset:
   "(*sn* X ⊆ *sn* Y) = (eventually (λn. X n ⊆ Y n) 𝒰)"
proof
  assume "*sn* X ⊆ *sn* Y"
  then have "∀x∈Iset (star_n X). x ∈ Iset (star_n Y)"
    by (simp add: set_mp starset_n_def)
  moreover
   have "∀Xa. star_n Xa ∈ Iset (star_n Y) = (∀_F n in 𝒰. Xa n ∈ Y n)"
     by (simp add: Iset_star_n)
   ultimately have "∀_F n in 𝒰. ∀x∈X n. x ∈ Y n"
     by (force intro: transfer_ball [THEN meta_eq_to_obj_eq, THEN iffD1])
   then show "∀_F n in 𝒰. X n ⊆ Y n"
     by (simp add: subset_eq)
 next
   assume  subsetF: "∀_F n in 𝒰. X n ⊆ Y n"
   show "*sn* X ⊆ *sn* Y"
   proof
     fix x
     assume xX: "x ∈ *sn* X"
     show "x ∈ *sn* Y"
     proof (cases x)
       case (star_n X)
       then show ?thesis
         using xX subsetF by (auto elim: eventually_elim2 simp add: Iset_star_n
starset_n_def)
     qed
   qed
```

qed

lemma *starfun_n_mult_star_n_right:*
*"(*fn* F) n * star_n X = (*fn* (λn m. F n m * X n)) n"*
by *(cases n, simp add: Ifun_star_n star_n_mult starfun_n_def)*

lemma *starfun_n_mult_star_n_left:*
*"star_n X * (*fn* F) n = (*fn* (λn m. X n * F n m)) n"*
by *(cases n, simp add: Ifun_star_n star_n_mult starfun_n_def)*

lemma *starfun_n_diff:*
*"(*fn* f) z - (*fn* g) z = (*fn* (λi x. f i x - g i x)) z"*
by *(cases z, simp add: Ifun_star_n star_n_minus star_n_diff starfun_n_def)*

lemma *starfun_n_inverse: "inverse ((*fn* f) x) = (*fn* (λi x. inverse ((f i) x))) x"*
by *(cases x, simp add: Ifun_star_n star_n_minus star_n_inverse starfun_n_def)*

lemma *InternalFuns_const_fun [simp]:*
"(λn. c) ∈ InternalFuns"
proof *(cases c, simp add: InternalFuns_def)*
case *(star_n X)*
have *"(λn. c) = *fn* (λm n. X m)"*
proof
fix *n :: "'c star"*
obtain *N* **where** *"n = star_n N"*
using *star_cases* **by** *blast*
then show *"c = (*fn* (λm n. X m)) n"*
by *(simp add: star_n starfun_n_def transfer_Ifun)*
qed
then show *"∃F. (λn. star_n X) = *fn* F"*
using *star_n* **by** *blast*
qed

lemma *starfun_n_divide:*
*"((*fn* f) z ::'a::division_ring star)/ (*fn* g) z = (*fn* (λi x. f i x / g i x)) z"*
by *(cases z, simp add: Ifun_star_n divide_inverse starfun_n_def star_n_inverse star_n_mult)*

lemma *InternalFuns_divide:*
"⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧
⟹ (λn. (f n ::'a::division_ring star)/g n) ∈ InternalFuns"
by *(auto simp add: InternalFuns_def starfun_n_divide)*

lemma *InternalFuns_id_fun [simp]: "(λn. n) ∈ InternalFuns"*
proof -
have *id: "(λn. n) = (*fn* (λn m. m))"*
by *(metis ext starfun_Id starfun_starfun_n_eq)*

```
   then show ?thesis
     using InternalFuns_def by blast
 qed


lemma FreeUltrafilterNat_star_n_Infinitesimal:
  assumes "eventually (λn. norm (X n) < inverse(Suc n)) 𝒰"
  shows "star_n X ∈ Infinitesimal"
proof (auto simp add: Infinitesimal_def hnorm_def starfun Reals_def
       star_of_def star_less_def starP2_star_n star_n_zero_num)
  fix r
  assume "∀_F n in 𝒰. 0 < (r::real)"
  moreover have "eventually (λn. inverse (real (Suc n)) < r) 𝒰"
    using FreeUltrafilterNat_inverse_real_of_posnat calculation transfer_bool
by blast
  ultimately show "∀_F n in 𝒰. norm (X n) < r" using assms
    by (force elim: eventually_elim2)
qed

lemma starset_n_atLeastLessThan_eq:
  "*sn* (λn. {(X n)..<(Y n)}) = {star_n X..<star_n Y}"
proof (auto simp add: starset_n_def)
  fix x
  assume x: "x ∈ Iset (star_n (λn. {X n..<Y n}))"
  then show "star_n X ≤ x"
  proof (cases x)
    case (star_n X)
    then show ?thesis
      using x by (simp add: Iset_star_n eventually_mono star_n star_n_le)

  qed
next
  fix x
  assume x: "x ∈ Iset (star_n (λn. {X n..<Y n}))"
  then show "x < star_n Y"
  proof (cases x)
    case (star_n X)
    then show ?thesis
      using x by (simp add: Iset_star_n eventually_mono star_n_less)
  qed
next
  fix x
  assume x: "star_n X ≤ x" "x < star_n Y"
  then show "x ∈ Iset (star_n (λn. {X n..<Y n}))"
  proof (cases x)
    case (star_n X)
    then show ?thesis
      using x by (simp add: Iset_star_n starP2_star_n star_n_less
           star_le_def eventually_conj_iff)
```

```
  qed
qed

lemma starset_n_atLeast_zero_LessThan_eq:
  "*sn* (λn. {0..<(X n)}) = {0..<star_n X}"
by (simp add: starset_n_atLeastLessThan_eq [of "λn. 0"] star_n_zero_num)

lemma starset_atLeastAtMost:
  "{of_nat m .. of_nat n} = *s* {m .. n}"
proof
  show "{of_nat m..of_nat n} ⊆ *s* {m..n}"
  proof
    have "⋀x. hypnat_of_nat m ≤ x ∧ x ≤ hypnat_of_nat n ⟶ x ∈ *s*
{m..n}"
      by (transfer, simp)
    then
    show "⋀x. x ∈ {of_nat m..of_nat n} ⟹ x ∈ *s* {m..n}"
      by simp
  qed
next
  show "*s* {m..n} ⊆ {of_nat m..of_nat n}"
  proof
    have "⋀x. x ∈ *s* {m..n} ⟹ hypnat_of_nat m ≤ x"
      by (transfer, simp)
    moreover have "⋀x. x ∈ *s* {m..n} ⟹ x ≤ hypnat_of_nat n"
      by (transfer, simp)
    ultimately show "⋀x. x ∈ *s* {m..n} ⟹ x ∈ {of_nat m..of_nat n}"
      by simp
  qed
qed

lemma starset_n_atLeastAtMost:
  "*sn* (λn. {(X n)..(Y n)}) = {star_n X..star_n Y}"
proof (auto simp add: starset_n_def)
  fix x
  assume x_in_ISet: "x ∈ Iset (star_n (λn. {X n..Y n}))"
  have "star_n X ≤ x ∧ x ≤ star_n Y"
  proof (cases x)
    case (star_n X)
    then show ?thesis
      using x_in_ISet
      by (auto simp add: star_le_def starP2_star_n Iset_star_n
            eventually_conj_iff)
  qed
  then show "star_n X ≤ x" and "x ≤ star_n Y"
    by auto
next
  fix x
  assume x_bounds: "star_n X ≤ x" "x ≤ star_n Y"
```

```
  then show "x ∈ Iset (star_n (λn. {X n..Y n}))"
  proof (cases x)
    case (star_n X)
    then show ?thesis
      using x_bounds
      by (auto simp add: star_le_def starP2_star_n Iset_star_n
            eventually_conj_iff)
  qed
qed

lemma starset_n_atLeast_zero_AtMost:
  "*sn* (λn. {0..(X n)}) = {0..star_n X}"
by (simp add: starset_n_atLeastAtMost [of "λn. 0"] star_n_zero_num)

lemma InternalSets_atLeastLessThan [simp]: "{M..<N} ∈ InternalSets"
  by (metis InternalSets_starset_n star_cases starset_n_atLeastLessThan_eq)


lemma InternalSets_atLeastAtMost [simp]: "{M..N} ∈ InternalSets"
proof -
  have "∃f. {M..N} = *sn* f"
    by (metis star_cases starset_n_atLeastAtMost)
  then show ?thesis
    by (simp add: InternalSets_def)
qed

lemma le_inverse_HNatInfinite_Infinitesimal:
      "⟦n ∈ HNatInfinite; hnorm x ≤ inverse (hypreal_of_hypnat n)⟧ ⟹
x ∈ Infinitesimal"
by (metis HNatInfinite_inverse_Infinitesimal hnorm_le_Infinitesimal)


lemma InternalFuns_hyperpow:
    "f ∈ InternalFuns ⟹ (g:: 'a star ⇒ nat star) ∈ InternalFuns
     ⟹ (λn. f n pow g n) ∈ InternalFuns"
proof (unfold hyperpow_def)
  assume "f ∈ InternalFuns"
  then obtain f' where F':"f = *fn* f'"
    using InternalFuns_def by auto
  moreover assume"(g:: 'a star ⇒ nat star) ∈ InternalFuns"
  moreover then obtain g' where G':"g = *fn* g'"
    using InternalFuns_def by auto
  ultimately have "(λn. (*f2* (^)) ((*fn* f') n) ((*fn* g') n))
                      = (*fn* (λ i x. (f' i x) ^ (g' i x)))"
    using starfun_n_hyperpow by auto
  thus "(λn. (*f2* (^)) (f n) (g n)) ∈ InternalFuns"
    using F' G' InternalFuns_starfun_n by auto
qed
```

```
lemma InternalFuns_hyperpowf [simp]:
  "f ∈ InternalFuns ⟹ (λn. x pow (f n)) ∈ InternalFuns"
  by (simp add: InternalFuns_hyperpow)

lemma InternalFuns_hyperpow_simple [simp]: "(λn. x pow n) ∈ InternalFuns"
  by simp
```

NatStar.thy lemmas

```
lemma InternalFuns_add:
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧ ⟹ (λn. f n + g n) ∈ InternalFuns"
by (auto simp add: InternalFuns_def starfun_n_add)

lemma InternalFuns_mult:
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧ ⟹ (λn. f n * g n) ∈ InternalFuns"
by (auto simp add: InternalFuns_def starfun_n_mult)

lemma InternalFuns_diff:
  "⟦ f ∈ InternalFuns; g ∈ InternalFuns ⟧ ⟹ (λn. f n - g n) ∈ InternalFuns"
  by (auto simp add: InternalFuns_def starfun_n_diff)

lemma InternalFuns_minus:
  "f ∈ InternalFuns  ⟹ (λn. - f n ) ∈ InternalFuns"
by (auto simp add: InternalFuns_def starfun_n_minus)

lemma InternalFuns_mult_const_left:
  "f ∈ InternalFuns ⟹ (λn. c * f n) ∈ InternalFuns"
by (force dest: InternalFuns_mult [OF InternalFuns_const_fun])

lemma InternalFuns_mult_const_right:
  "f ∈ InternalFuns ⟹ (λn. f n * c) ∈ InternalFuns"
by (force dest: InternalFuns_mult [OF _ InternalFuns_const_fun])

lemma InternalFuns_mult_of_hypnat [simp]:
      "X ∈ InternalFuns ⟹ (λn. of_hypnat n * X n) ∈ InternalFuns"
by (auto simp add: InternalFuns_def of_hypnat_def starfun_starfun_n_eq
starfun_n_mult)

lemma finite_InternalSets:
  assumes "finite X"
  shows "X ∈ InternalSets"
proof (insert assms, induct set: finite)
  case empty
  then show ?case by simp
next
  case (insert x F)
  then show ?case
    by (auto intro: insert_def [THEN ssubst] intro!: InternalSets_Un)
qed
```

```
lemma InternalFuns_of_hypnat:
  assumes "f ∈ InternalFuns"
  shows "(λn. of_hypnat (f n)) ∈ InternalFuns"
proof (insert assms, auto simp add: InternalFuns_def)
  fix F
  have "(λn. of_hypnat ((*fn* (F::nat ⇒ 'a ⇒ nat)) n)) = *fn* (λn m.
of_nat(F n m))"
  proof
    fix n
    obtain N where "(n::'a star) = star_n N"
      using star_cases by blast
    then show " of_hypnat ((*fn* F) n) = (*fn* (λn m. of_nat (F n m)))
n"
      by (simp add: starfun_n of_hypnat_def starfun)
  qed
  then show "∃Fa. (λn. of_hypnat ((*fn* F) n)) = *fn* Fa"
    by blast
qed

lemma insert_star_n_starset_n:
  "insert (star_n X) (*sn* A) = *sn* (λn. insert (X n) (A n))"
proof (unfold insert_def)
  have "{x. x = star_n X} ∪ *sn* A = *sn* (λn. {X n} ∪ A n)"
    by (simp only: starset_n_singleton2 [symmetric] starset_n_Un [symmetric])
  then show "{x. x = star_n X} ∪ *sn* A = *sn* (λn. {x. x = X n} ∪ A
n)"
    by simp
qed

lemma inj_starfun_n:
  "inj (*fn* f) =  (∀_F n in 𝒰. inj (f n))"
proof
  assume "inj (*fn* f)"
  then have "∀x y. (*fn* f) x = (*fn* f) y ⟶ x = y "
    by (simp add: inj_eq)
  then have "∀Y Ya. ∀_F n in 𝒰. f n (Y n) = f n (Ya n) ⟶ Y n = Ya n"
    by (metis  FreeUltrafilterNat.eventually_imp_iff star_n_eq_iff starfun_n)
  then show "∀_F n in 𝒰. inj (f n)"
    by (auto simp add: inj_def FreeUltrafilterNat.eventually_all_iff)
next
  assume "∀_F n in 𝒰. inj (f n)"
  then have eventually_inj: "∀Y Ya. ∀_F n in 𝒰. f n (Y n) = f n (Ya n)
⟶ Y n = Ya n"
    by (simp add: eventually_mono inj_def)
  then show "inj (*fn* f)"
  proof(auto simp add: inj_def)
    fix x y
    assume "(*fn* f) x = (*fn* f) y"
    then show "x = y"
```

```
    using eventually_inj
    by (metis not_eventually_impI star_cases star_n_eq_iff starfun_n)
  qed
qed


lemma surj_starfun_n:
  "surj (*fn* f) =  (∀F n in 𝒰. surj (f n))"
proof
  assume "surj (*fn* f)"
  then have "∀y. ∃x. y = (*fn* f) x"
    by (simp add: surjD)
  then have "∀Y. ∃Ya. ∀F n in 𝒰. Y n = f n (Ya n)"
    by (metis Ifun_star_n star_cases star_n_eq_iff starfun_n_def)
  then have "∀Y. ∀F n in 𝒰. ∃x. Y n = f n x"
    by (simp add: eventually_ex)
  then show "∀F n in 𝒰. surj (f n)"
    by (auto simp add: surj_def FreeUltrafilterNat.eventually_all_iff)
next
  assume "∀F n in 𝒰. surj (f n)"
  then have eventually_surj: "∀Y. ∃Ya. ∀F n in 𝒰. Y n = f n (Ya n)"
    by (simp add: surj_def FreeUltrafilterNat.eventually_all_iff eventually_ex)
  then show "surj (*fn* f)"
  proof(auto simp add: surj_def)
    fix y
    show "∃x. y = (*fn* f) x"
      by (metis eventually_surj star_cases star_n_eq_iff starfun_n)
  qed
qed

lemma bij_starfun_n:
  "bij (*fn* f) =  (∀F n in 𝒰. bij (f n))"
  unfolding bij_def by (simp add: eventually_conj_iff inj_starfun_n surj_starfun_n)

lemma
  assumes "bij (*fn* f)"
  shows "(inv (*fn* f)) = *fn* (λn. inv (f n))"
proof
  have inj_fn: "inj (*fn* f)"
    using assms bij_betw_def by blast
  have surj_fn: "surj (*fn* f)"
    using assms
    by (simp add: bij_betw_def)
  fix x
  have "(*fn* f) ((*fn* (λn. inv (f n))) x) = x"
  proof (cases x)
    case (star_n X)
    then have " ∀F n in 𝒰. f n (inv (f n) (X n)) = X n"
      using surj_fn
```

```
    by (auto simp add:  surj_starfun_n eventually_mono surj_f_inv_f)
  then show ?thesis
    by (simp add: star_n star_n_eq_iff starfun_n)
qed
then show "inv (*fn* f) x = (*fn* (λn. inv (f n))) x"
  by (metis inj_fn inv_f_eq)
qed

lemma starfun_n_o:
  "(λi. (*fn* F) ((*fn* G) i)) = (*fn* (λi m. F i (G i m)))"
proof (rule ext)
  fix i
  show "(*fn* F) ((*fn* G) i) = (*fn* (λi m. F i (G i m))) i"
    by (case_tac i, auto simp add: starfun_n)
qed

lemma starfun_starfun_n_o:
  "(λi. (*f* f) ((*fn* F) i)) = (*fn* (λi m. f (F i m)))"
by (simp add: starfun_starfun_n_eq starfun_n_o)

lemma InternalFuns_o2:
  "f ∈ InternalFuns ⟹ g ∈ InternalFuns ⟹ (λn. f (g n)) ∈ InternalFuns"
  unfolding InternalFuns_def using starfun_n_o by blast

lemma InternalFuns_abs_starfun_n [simp]:
  "(λi. abs ((*fn* F) i)) ∈ InternalFuns"
by (metis InternalFuns_starfun_n starfun_rabs_hrabs starfun_starfun_n_o)

lemma InternalFuns_abs:
  "f ∈ InternalFuns ⟹ (λi. abs (f i)) ∈ InternalFuns"
  by (metis InternalFuns_o2 InternalFuns_starfun starfun_rabs_hrabs)
```

HSEQ.thy lemmas

```
lemma LIMSEQ_0_Infinitesimal:
  assumes "X ——→ 0"
  shows "star_n X ∈ Infinitesimal"
proof -
  have "∀N∈HNatInfinite. (*f* X) N ≈ 0"
    using assms LIMSEQ_NSLIMSEQ NSLIMSEQ_D by fastforce
  then have "∀N∈HNatInfinite.
            ∀n. hnorm ((*f* X) N) < inverse (1 + hypreal_of_nat n)"
    using mem_infmal_iff [symmetric] Infinitesimal_hypreal_of_nat_iff
by auto
  then have "∀n. hnorm ((*f* X) whn) < inverse (1 + hypreal_of_nat n)"
by force
  then show ?thesis
    by (auto simp add: hypnat_omega_def starfun Infinitesimal_hypreal_of_nat_iff)
qed
```

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

the AFP session eventually.

```
lemma powrat_def2: "x pow_Q r = root (nat (snd(quotient_of r))) (x powi
(fst(quotient_of r)))"
proof (cases "r > 0")
  case True
  then have "x ^ nat (fst (quotient_of r)) = x powi fst (quotient_of
r)"
    by (metis Fract_quotient_of order_less_le power_int_nonneg_exp quotient_of_denom_pos'
zero_le_Fract_iff)
  then show ?thesis
    by (simp add: True powrat_def)
next
  case False
  then have "1 / x ^ nat (- fst (quotient_of r)) = x powi fst (quotient_of
r)"
    by (metis Rat_cases add.inverse_inverse linorder_not_le nat_eq_iff2
neg_less_0_iff_less normalize_stable
       power_int_minus_divide power_int_of_nat prod.sel(1) quotient_of_Fract
zero_less_Fract_iff)
  then show ?thesis
    by (simp add: False powrat_def)
qed

lemma powrat_powi: "x pow_Q (of_int n) = x powi n"
  by (simp add: powrat_def2)

lemma real_root_eq_powrat_inverse': "n > 0 ⟹ x pow_Q (1 / of_nat n)
= root n x"
  by (simp add: inverse_eq_divide real_root_eq_powrat_inverse)

lemma "a powa x = lim (λn. a pow_Q (incseq_of x n))"
  by (simp add: Lim_def powa_def)
```

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 -
  have "(*f* (λn. x ^ n)) N ≈ 0"
    using NSLIMSEQ_def LIMSEQ_realpow_zero LIMSEQ_NSLIMSEQ_iff assms by
force
  thus ?thesis using starfun_pow assms by simp
qed

lemma HNatInfinite_pow_Infinitesimal:
  assumes "0 < (x::hypreal)" and "x < 1" and "¬(x ≈ 1)" and "N ∈ HNatInfinite"
  shows "x pow N ≈ 0"
proof -
  from assms have "x ∈ HFinite"
```

```
    using HInfinite_gt_zero_gt_one not_HFinite_HInfinite order.asym by
blast
  then obtain x' where st_x: "star_of x' ≈ x"
    using st_SReal st_approx_self SReal_iff by fastforce
  then have "x' ≠ 1" using assms(3) by auto
  then have "x' < 1" using st_x assms(2)
    by (metis star_of_1_le star_of_approx_one_iff approx_le_bound less_imp_le
not_less)
  then  obtain y where y: "x' < y ∧ y < 1"
    using dense by blast
  then have x_less_y: "x < star_of y"
    using st_x
    by (metis Infinitesimal_add_hypreal_of_real_less approx_sym bex_Infinitesimal_iff2)

  then have "x pow N ≤ star_of y pow N"
    by (simp add: assms(1) hyperpow_le less_imp_le)
  moreover have "0 < x pow N"
    by (simp add: assms(1) hyperpow_gt_zero)
  moreover have "star_of y pow N ≈ 0"
    using assms(1,4) HNatInfinite_pow_real_Infinitesimal x_less_y y by
auto
  ultimately show ?thesis
    using approx_le_bound approx_trans3 less_imp_le by blast
qed

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 -
  have "hnorm x pow N ≈ 0"
    using HNatInfinite_pow_Infinitesimal assms
    by force
  then show ?thesis
    by (metis Infinitesimal_hnorm_iff hnorm_hyperpow mem_infmal_iff)
qed
```

HSeries.thy lemmas

```
lemma NSsums_sumhr_HNatInfinite_approx_iff:
  "(f NSsums s) = (∀N∈HNatInfinite. sumhr(0, N, f) ≈ of_real s)"
by (simp add: NSsums_def sumhr_app NSLIMSEQ_def star_of_zero [symmetric]

       atLeast0LessThan del: star_of_zero)

lemma sumhr_approx:
  "sumhr (0, M, f) ≈ sumhr (0, N, f) ⟹ sumhr (M, N, f) ≈ 0"
by (metis approx_zero_abs_zero_iff sumhr_hrabs_approx)
```

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"*
by *(metis mult_less_cancel_left_disj of_nat_0_less_iff)*

lemma *hypint_mult_less_mono2_lemma:*
     *"⋀i j k. (i::hypint)<j ⟹ 0<k ⟹ of_hypnat k * i < of_hypnat k*
** j"*
by *transfer (rule zmult_zless_mono2_lemma)*

lemma *zero_le_imp_eq_hypint: "⋀k. (0::hypint) ≤ k ⟹ ∃n. k = of_hypnat*
n"
by *transfer (rule zero_le_imp_eq_int)*

lemma *zero_less_imp_eq_hypint: "⋀k. (0::hypint) < k ⟹ ∃n>0. k = of_hypnat*
n"
by *transfer (rule zero_less_imp_eq_int)*

lemma *hypint_less_mono2: "⋀i j k. ⟦ i<j; (0::hypint) < k ⟧ ⟹ k*i <*
*k*j"*
by *transfer (rule zmult_zless_mono2)*

lemma *hypint_less_imp_add1_le: "⋀w z. w < z ⟹ w + (1::hypint) ≤ z"*
by *transfer (rule zless_imp_add1_zle)*

lemma *hypint_less_iff_hSuc_add:*
  "⋀w z. (w :: hypint) < z ⟷ (∃n. z = w + of_hypnat (hSuc n))"
by *transfer (rule zless_iff_Suc_zadd)*

lemma *hypint_le_iff_add:*
"⋀w z. (w :: hypint) ≤ z ⟷ (∃n. z = w + of_hypnat n)"
by *transfer (simp add: zle_iff_zadd)*

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

lemma *hypint_less_zero_eq_minus_of_hypnat: "x < (0::hypint) ⟹ ∃n. x = - hypint_of_hypnat (n)"*
by *(metis minus_minus neg_0_less_iff_less zero_less_imp_eq_hypint)*

lemma *hypint_less_add1_eq: "⋀w z. (w < z + (1::hypint)) = (w<z | w=z)"*
by *transfer (rule zless_add1_eq)*

lemma *add1_hypint_le_eq: "⋀w z. (w + (1::hypint)≤ z) = (w<z)"*
by *transfer (rule add1_zle_eq)*

lemma *hypint_le_diff1_eq [simp]: "⋀w z. (w ≤ z - (1::hypint)) = (w<z)"*
by *transfer (rule zle_diff1_eq)*

lemma *hypint_le_add1_eq_le [simp]: "⋀w z. (w < z + (1::hypint)) = (w ≤ z)"*
by *transfer (rule zle_add1_eq_le)*

lemma *hypint_one_le_iff_zero_less: "⋀z. ((1::hypint) ≤ z) = (0 < z)"*
by *transfer (rule int_one_le_iff_zero_less)*

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"*
by *transfer simp*

lemma *of_hypint_hypint_of_int [simp]: "of_hypint (hypint_of_int z) = of_int z"*
by *transfer simp*

lemma *of_hypint_of_hypnat [simp]: "⋀z. of_hypint (of_hypnat z) = of_hypnat z"*
by *transfer simp*

lemma *of_hypint_of_nat [simp]: "of_hypint (of_nat z) = of_nat z"*
by *transfer simp*

lemma *of_hypint_0 [simp]: "of_hypint 0 = 0"*
by *transfer (rule of_int_0)*

lemma *of_hypint_1 [simp]: "of_hypint 1 = 1"*
by *transfer (rule of_int_1)*

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

lemma *of_hypint_minus [simp]: "⋀z. of_hypint (-z) = - (of_hypint z)"*
by *transfer (rule of_int_minus)*

lemma *of_hypint_diff [simp]: "⋀w z. of_hypint (w - z) = of_hypint w - of_hypint z"*
by *transfer (rule of_int_diff)*

lemma *of_hypint_mult [simp]: "⋀w z. of_hypint (w*z) = of_hypint w * of_hypint z"*
by *transfer (rule of_int_mult)*

lemma *of_hypint_of_nat_eq [simp]: "of_hypint (of_nat n) = of_nat n"*
by *(induct n) auto*

lemma *of_hypint_nat_power: "of_hypint (z ^ n) = of_hypint z ^ n"*
by *(induct n) simp_all*

lemma *of_hypint_eq_iff [simp]:*
"⋀w z. of_hypint w = of_hypint z ⟷ w = z"
by *transfer (rule of_int_eq_iff)*

lemma *of_hypint_eq_0_iff [simp]:*
"⋀z. of_hypint z = 0 ⟷ z = 0"
by *transfer (rule of_int_eq_0_iff)*

lemma *of_hypint_0_eq_iff [simp]:*
"⋀z. 0 = of_hypint z ⟷ z = 0"
by *transfer (rule of_int_0_eq_iff)*

lemma *of_hypint_le_iff [simp]:*
"⋀ w z. of_hypint w ≤ of_hypint z ⟷ w ≤ z"
by *transfer (rule of_int_le_iff)*

lemma *of_hypint_less_iff [simp]:*
"⋀w z. of_hypint w < of_hypint z ⟷ w < z"
by *transfer (rule of_int_less_iff)*

lemma *of_hypint_0_le_iff [simp]:*
"⋀z. 0 ≤ of_hypint z ⟷ 0 ≤ z"

by *transfer (rule of_int_0_le_iff)*

lemma *of_hypint_le_0_iff [simp]:*
"⋀z. of_hypint z ≤ 0 ⟷ z ≤ 0"
by *transfer (rule of_int_le_0_iff)*

lemma *of_hypint_0_less_iff [simp]:*
"⋀z. 0 < of_hypint z ⟷ 0 < z"
by *transfer (rule of_int_0_less_iff)*

lemma *of_hypint_less_0_iff [simp]:*
"⋀z. of_hypint z < 0 ⟷ z < 0"
by *transfer (rule of_int_less_0_iff)*

lemma *of_hypint_abs: "⋀z. of_hypint (abs z) = abs(of_hypint z)"*
by *transfer (simp add: abs_if)*

lemma *of_hypint_add_one: "of_hypint z + (1 :: real star) = of_hypint (z + 1)"*
by *(metis of_hypint_1 of_hypint_add)*

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}"*
by *(simp add: Nats_def image_def)*

lemma *hypint_of_nat:*
"hypint_of_nat m = star_n (%n. int m)"
by *(fold star_of_def) simp*

lemma *HypintNat_ge_zero: "n ∈ ℕ ⟹ 0 ≤ (n :: hypint)"*
by *(auto simp add: Nats_def)*

lemma *hypint_of_hypnat_hypint_of_nat_eq [simp]:*
"⋀n. hypint_of_hypnat (hypnat_of_nat n) = hypint_of_nat n"
by *(simp add: of_hypnat_def)*

lemma *abs_hypint_of_hypnat [simp]: "|hypint_of_hypnat n| = hypint_of_hypnat n"*
by *(auto simp add: abs_if)*

lemma *hypint_abs_less_one_iff [simp]: "⋀z. (|z| < 1) = (z = (0::hypint))"*
by *transfer simp*

lemma *abs_hypint_mult_eq_1:*

```
    "⋀m n. |m * n| = 1 ⟹ |m| = (1::hypint)"
by transfer (rule abs_zmult_eq_1)

lemma pos_hypint_mult_eq_1_iff_lemma:
    "⋀m n. (m * n = 1) ==> m = (1::hypint) | m = -1"
by transfer (rule pos_zmult_eq_1_iff_lemma)

lemma pos_hypint_mult_eq_1_iff:
   "⋀m n. 0 < (m::hypint) ⟹ (m * n = 1) = (m = 1 ∧ n = 1)"
by transfer (rule pos_zmult_eq_1_iff)

lemma hypint_mult_eq_1_iff:
   "⋀m n. (m*n = (1::hypint)) = ((m = 1 ∧ n = 1) ∨ (m = -1 ∧ n = -1))"
by transfer (rule zmult_eq_1_iff)
```

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"
by transfer simp

lemma hypnat_zero [simp]: "hypnat 0 = 0"
by transfer simp

lemma hypnat_one [simp]: "hypnat 1 = 1"
by (metis hypnat_hypint of_hypnat_1)

lemma hypint_hypnat_eq [simp]: "⋀z. of_hypnat (hypnat z) = (if 0 ≤ z
then z else 0)"
by transfer simp

corollary hypnat_0_le: "0 ≤ z ⟹ of_hypnat (hypnat z) = z"
by simp

lemma hypnat_le_0 [simp]: "⋀z. z ≤ 0 ⟹ hypnat z = 0"
by transfer simp

lemma hypnat_le_eq_zle: "⋀w z. 0 < w ∨ 0 ≤ z ⟹ (hypnat w  ≤ hypnat
z) = (w ≤ z)"
by transfer (rule nat_le_eq_zle)

corollary hypnat_mono_iff: "0 < z ⟹ (hypnat w < hypnat z) = (w < z)"
by (simp add: hypnat_le_eq_zle linorder_not_le [symmetric])

corollary hypnat_less_eq_less: "0 ≤ w ⟹ (hypnat w < hypnat z) = (w<z)"
```

by *(simp add: hypnat_le_eq_zle linorder_not_le [symmetric])*

lemma *less_hypnat_conj [simp]: "⋀w z. (hypnat w < hypnat z) = (0 < z ∧ w < z)"*
by *transfer (rule zless_nat_conj)*

lemma *nonneg_eq_hypint:*
fixes *z :: hypint*
assumes *"0 ≤ z"* **and** *"⋀m. z = of_hypnat m ⟹ P"*
shows *P*
using *assms* **by** *(blast dest: hypnat_0_le sym)*

lemma *hypnat_eq_iff:*
"⋀ m w. (hypnat w = m) = (if 0 ≤ w then w = of_hypnat m else m=0)"
by *transfer (rule nat_eq_iff)*

corollary *hypnat_eq_iff2: "(m = hypnat w) = (if 0 ≤ w then w = of_hypnat m else m=0)"*
by *(simp only: eq_commute [of m] hypnat_eq_iff)*

lemma *hypnat_less_iff: "⋀m w. 0 ≤ w ⟹ (hypnat w < m) = (w < of_hypnat m)"*
by *transfer (rule nat_less_iff)*

lemma *hypnat_0_iff[simp]: "⋀i. hypnat i = 0 ⟷ i≤0"*
by *transfer simp*

lemma *hypint_eq_iff: "(of_hypnat m = z) = (m = hypnat z ∧ 0 ≤ z)"*
by *(auto simp add: hypnat_eq_iff2)*

lemma *zero_less_hypnat_eq [simp]: "(0 < hypnat z) = (0 < z)"*
by *(insert less_hypnat_conj [of 0], auto)*

lemma *hypnat_add_distrib:*
"⋀z z'. ⟦ (0::hypint) ≤ z; 0 ≤ z' ⟧ ⟹ hypnat (z+z') = hypnat z + hypnat z'"
by *transfer (rule nat_add_distrib)*

lemma *hypnat_diff_distrib:*
"⋀z z'. ⟦ (0::hypint) ≤ z'; z' ≤ z ⟧ ⟹ hypnat (z-z') = hypnat z - hypnat z'"
by *transfer (rule nat_diff_distrib)*

lemma *hypnat_minus_hypint [simp]: "⋀n. hypnat (- (of_hypnat n)) = 0"*
by *transfer simp*

lemma *less_hypnat_eq_hypint_less: "⋀m z. (m < hypnat z) = (of_hypnat m < z)"*
by *transfer (rule zless_nat_eq_int_zless)*

lemma *hypnat_add_one: "0 ≤ z ⟹ hypnat z + (1 :: nat star) = hypnat (z + 1)"*
by *(metis hypint_eq_iff hypnat_add_distrib of_hypnat_1)*

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)"
by *transfer (rule zdvd_antisym_nonneg)*

lemma *hypint_dvd_antisym_abs:*
"⋀a b. ⟦ (a::hypint) dvd b; b dvd a ⟧ ⟹ |a| = |b|"
by *transfer (rule zdvd_antisym_abs)*

lemma *hyperint_dvd_diffD: "⋀k m n. k dvd m - n ⟹ k dvd n ⟹ k dvd (m::hypint)"*
by *transfer (rule zdvd_zdiffD)*

lemma *hypint_dvd_reduce: "⋀k m n. (k dvd n + k * m) = (k dvd (n::hypint))"*
by *transfer (rule zdvd_reduce)*

lemma *hypint_dvd_imp_le_hypint:*
"⋀d i. ⟦ i ≠ 0; d dvd (i::hypint) ⟧ ⟹ |d| ≤ |i|"
by *transfer (rule dvd_imp_le_int)*

lemma *hypint_dvd_not_less:*
"⋀m n. ⟦ 0 < m; m < (n::hypint) ⟧ ⟹ ¬ n dvd m"
by *transfer (rule zdvd_not_zless)*

lemma *hypint_dvd_mult_cancel:*
*"⋀k m n. ⟦ k * m dvd k * n; k ≠ (0::hypint) ⟧ ⟹ m dvd n"*
by *transfer (rule zdvd_mult_cancel)*

theorem *dvd_hypint_of_hypnat:*
"⋀x y. ((x::hypnat) dvd y) = ((hypint_of_hypnat x) dvd (hypint_of_hypnat y))"
by *transfer simp*

lemma *hypint_dvd1_eq[simp]: "⋀x. (x::hypint) dvd 1 = (|x| = 1)"*
by *transfer (rule zdvd1_eq)*

lemma *hypint_dvd_mult_cancel1:*
*"⋀m n. m ≠ (0::hypint) ⟹ (m * n dvd m) = (|n| = 1)"*
by *transfer (rule zdvd_mult_cancel1)*

lemma *hypint_of_hypnat_dvd_iff:*
"⋀m z. ((of_hypnat m) dvd z) = (m dvd hypnat (abs z))"
using *dvd_hypint_of_hypnat* **by** *auto*

lemma *dvd_hypint_iff: "⋀m z. (z dvd of_hypnat m) = (hypnat (abs z) dvd m)"*
using *nat_abs_dvd_iff* **by** *transfer blast*

lemma *hypnat_dvd_iff: "(hypnat z dvd m) = (if 0 ≤ z then (z dvd of_hypnat m) else m = 0)"*
by *(auto simp add: dvd_hypint_iff)*

lemma *eq_hypnat_hypnat_iff:*
"0 ≤ z ⟹ 0 ≤ z' ⟹ hypnat z = hypnat z' ⟷ z = z'"
by *(auto elim!: nonneg_eq_hypint)*

lemma *hypnat_power_eq:*
"⋀z. 0 ≤ z ⟹ hypnat (z ^ n) = hypnat z ^ n"
by *transfer (rule nat_power_eq)*

lemma *hypint_dvd_imp_le:*
"⋀n z. ⟦ z dvd n; 0 < n ⟧ ⟹ z ≤ (n::hypint)"
by *transfer (rule zdvd_imp_le)*

lemma *hypint_period:*
*"⋀a c d t x. (a::hypint) dvd d ⟹ a dvd (x + t) ⟷ a dvd ((x + c * d) + t)"*
by *transfer (rule zdvd_period)*

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)"*
by *transfer (simp add: zdiv_int)*

lemma *hypint_of_hypnat_div: "⋀m n. hypint_of_hypnat(m div n) = hypint_of_hypnat m div (hypint_of_hypnat n)"*
by *transfer (simp add: zdiv_int)*

lemma *HypintNats_div: "⟦(n::hypint) ∈ ℕ; m ∈ ℕ ⟧ ⟹ n div m ∈ ℕ"*
by *(metis Nats_cases hypint_of_nat_div of_nat_in_Nats)*

lemma *hypint_div_mono1: "⋀a a' b. ⟦ a ≤ a'; 0 < (b::hypint) ⟧ ⟹ a div b ≤ a' div b"*
by *transfer (rule zdiv_mono1)*

lemma *hypint_div_neg_pos_less0: "⋀a b. ⟦ a < (0::hypint); 0 < b ⟧ ⟹ a div b < 0"*
by *transfer (rule div_neg_pos_less0)*

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

by *transfer (rule div_nonneg_neg_le0)*

lemma *hypint_div_nonpos_pos_le0:* "⋀a b. ⟦ *(a::hypint)* ≤ *0; b > 0* ⟧ ⟹ *a div b* ≤ *0*"
by *transfer (rule div_nonpos_pos_le0)*

lemma *hypint_pos_imp_div_nonneg_iff:* "⋀a b. *(0::hypint) < b* ⟹ *(0* ≤ *a div b) = (0* ≤ *a)*"
by *transfer (rule pos_imp_zdiv_nonneg_iff)*

lemma *hypint_neg_imp_div_nonneg_iff:*
"⋀a b. *b < (0::hypint)* ⟹ *(0* ≤ *a div b) = (a* ≤ *(0::hypint))*"
by *transfer (rule neg_imp_zdiv_nonneg_iff)*

lemma *hypint_pos_imp_div_neg_iff:* "⋀a b. *(0::hypint) < b* ⟹ *(a div b < 0) = (a < 0)*"
by *transfer (rule pos_imp_zdiv_neg_iff)*

lemma *hypint_neg_imp_div_neg_iff:* "⋀a b. *b < (0::hypint)* ⟹ *(a div b < 0) = (0 < a)*"
by *transfer (rule neg_imp_zdiv_neg_iff)*

lemma *hypint_nonneg1_imp_div_pos_iff:*
"⋀a b. *(0::hypint)* ≤ *a* ⟹ *(a div b > 0) = (a* ≥ *b* ∧ *b > 0)*"
by *transfer (rule nonneg1_imp_zdiv_pos_iff)*

3.7 Properties of the set of embedded integers

lemma *of_int_eq_star_of [simp]:* "of_int = star_of"
proof
fix *z :: int*
show "of_int z = star_of z" **by** *transfer simp*
qed

lemma *Ints_eq_Standard:* "(ℤ :: int star set) = Standard"
by *(auto simp add: Ints_def Standard_def)*

lemma *hypint_of_int_mem_Ints [simp]:* "hypint_of_int z ∈ ℤ"
by *(simp add: Ints_eq_Standard)*

lemma *hypint_of_nat_Suc [simp]:*
"hypint_of_nat (Suc n) = hypint_of_nat n + (1::hypint)"
by *transfer simp*

lemma *of_int_eq_add [rule_format]:*
"∀ *d::hypint. of_int m = of_int n + d* ⟶ *d* ∈ *range of_int*"
by *(metis add_diff_cancel_left' of_int_diff rangeI)*

3.8 Infinite hyperintegers

definition

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

lemma Ints_not_HIntInfinite_iff: "(x ∈ Ints) = (x ∉ HIntInfinite)"
by (simp add: HIntInfinite_def)

lemma HIntInfinite_not_Ints_iff: "(x ∈ HIntInfinite) = (x ∉ Ints)"
by (simp add: HIntInfinite_def)

lemma star_of_neq_HIntInfinite: "Z ∈ HIntInfinite ==> star_of n ≠ Z"
by (metis HIntInfinite_not_Ints_iff Ints_of_int of_hypint_hypint_of_int

     of_hypint_of_int of_int_eq_iff star_of_int_def)

lemma HIntInfinite_minus_iff [simp]:
  "(- Z ∈ HIntInfinite) =  (Z ∈ HIntInfinite)"
by (metis Ints_minus Ints_not_HIntInfinite_iff minus_minus)


lemma star_of_less_pos_HIntInfinite:
  assumes "0 < Z" and "Z ∈ HIntInfinite"
  shows "star_of z < Z"
proof (cases "z ≥ 0")
  case True
  then obtain n where z: "z = int n"
    using zero_le_imp_eq_int by blast
  have "hypint_of_int (int n) < Z"
  proof (induct n)
    case 0
    then show ?case
      by (simp add: assms(1))
  next
    case (Suc n)
    then show ?case
      by (metis assms(2) hypint_less_imp_add1_le hypint_of_nat_Suc
          less_le star_of_neq_HIntInfinite star_of_of_nat)
  qed
  then show ?thesis
    using z by simp
next
  case False
  then obtain n where "z = - int (Suc n)"
    by (metis int_cases of_nat_0_le_iff)
  then show ?thesis
    by (metis False assms(1) le_less less_le_trans not_less star_of_less_0)
```

qed

lemma *star_of_gt_neg_HIntInfinite:*
  assumes *"Z < 0"*
and *"Z ∈ HIntInfinite"*
shows *"Z < star_of z"*
proof -
  have *"0 < - Z"*
    by *(simp add: assms(1))*
  then show *?thesis* **using** *star_of_less_pos_HIntInfinite [of _ "-z"]*
    using *assms(2)* **by** *fastforce*
qed

lemma *zero_not_HIntInfinite [simp]: "0 ∉ HIntInfinite"*
by *(metis HIntInfinite_not_Ints_iff Ints_0)*

lemma *star_of_less_abs_HIntInfinite:*
  "Z ∈ HIntInfinite ⟹ star_of z < abs Z"
by *(auto intro: star_of_less_pos_HIntInfinite simp add: abs_if not_less le_less)*

lemma *hypint_of_nat_less_abs_HIntInfinite:*
  "y ∈ HIntInfinite ⟹ hypint_of_nat n < |y|"
by *(metis star_of_less_abs_HIntInfinite star_of_nat_def)*

lemma *Ints_less_pos_HIntInfinite: "⟦ 0 < y; x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ x < y"*
by *(auto simp add: Ints_def star_of_less_pos_HIntInfinite)*

lemma *Ints_gt_neg_HIntInfinite: "⟦ y < 0; x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ y < x"*
by *(auto simp add: Ints_def star_of_gt_neg_HIntInfinite)*

lemma *Ints_less_abs_HIntInfinite: "⟦ x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ x < abs y"*
by *(auto simp add: Ints_def star_of_less_abs_HIntInfinite)*

lemma *Ints_le_abs_HIntInfinite: "⟦ x ∈ ℤ; y ∈ HIntInfinite ⟧ ⟹ x ≤ abs y"*
by *(metis Ints_less_abs_HIntInfinite less_imp_le)*

lemma *Nats_less_abs_HIntInfinite: "⟦ x ∈ ℕ; y ∈ HIntInfinite ⟧ ⟹ x < abs y"*
by *(metis Nats_cases hypint_of_nat_less_abs_HIntInfinite)*

lemma *Nats_le_abs_HIntInfinite: "⟦ x ∈ ℕ; y ∈ HIntInfinite ⟧ ⟹ x < abs y"*
by *(metis Nats_less_abs_HIntInfinite)*

```
lemma Ints_pos_downward_closed:
  "⟦ 0 ≤ y; x ∈ Ints; (y::hypint) ≤ x ⟧ ⟹ y ∈ ℤ"
  using HIntInfinite_not_Ints_iff Ints_le_abs_HIntInfinite by fastforce

lemma Ints_neg_upward_closed:
  "⟦ y ≤ 0; x ∈ Ints; x ≤ (y::hypint) ⟧ ⟹ y ∈ ℤ"
  by (metis HIntInfinite_not_Ints_iff Ints_0 Ints_gt_neg_HIntInfinite
linorder_not_less order_le_less)

lemma HIntInfinite_pos_upward_closed:
  "⟦ 0 ≤ x; x ∈ HIntInfinite; x ≤ y ⟧ ⟹ y ∈ HIntInfinite"
by (auto intro: Ints_pos_downward_closed simp only: HIntInfinite_not_Ints_iff)

lemma HIntInfinite_neg_downward_closed:
  "⟦ x ≤ 0; x ∈ HIntInfinite; y ≤ x ⟧ ⟹ y ∈ HIntInfinite"
by (auto intro: Ints_neg_upward_closed simp only: HIntInfinite_not_Ints_iff)

lemma HIntInfinite_pos_add:
  "⟦ 0 ≤ x; 0 ≤ y; x ∈ HIntInfinite ⟧ ⟹ x + y ∈ HIntInfinite"
by (metis HIntInfinite_pos_upward_closed add_increasing2 order_refl)

lemma HIntInfinite_neg_add:
  "⟦ x ≤ 0; y ≤ 0; x ∈ HIntInfinite ⟧ ⟹ x + y ∈ HIntInfinite"
by (metis HIntInfinite_neg_downward_closed add_0_iff add_le_cancel_left)

lemma HIntInfinite_add_Ints:
  "⟦ x ∈ HIntInfinite; y ∈ Ints ⟧ ⟹ x + y ∈ HIntInfinite"
  by (metis HIntInfinite_not_Ints_iff Ints_diff add.commute add_diff_cancel_left')

lemma HIntInfinite_diff:
  "⟦ x ∈ HIntInfinite; y ∈ Ints ⟧ ⟹ x - y ∈ HIntInfinite"
by (metis HIntInfinite_not_Ints_iff Ints_add diff_add_cancel)

lemma Ints_def2: "ℤ = {z. ∃Z. z = hypint_of_int Z}"
by (auto simp add: Ints_def)

lemma HIntInfinite_hypint_of_hypnat_iff:
  "(hypint_of_hypnat n ∈ HIntInfinite) = (n ∈ HNatInfinite)"
proof
  assume "hypint_of_hypnat n ∈ HIntInfinite"
  then show "n ∈ HNatInfinite"
    using HNatInfinite_not_Nats_iff SHNat_eq hypint_of_nat_less_abs_HIntInfinite
by force
next
  assume "n ∈ HNatInfinite"
  then have "∀N. n ≠ hypnat_of_nat N"
    using star_of_neq_HNatInfinite by blast
  then show "hypint_of_hypnat n ∈ HIntInfinite"
    by (metis Ifun_star_of Ints_cases Ints_not_HIntInfinite_iff hypnat_def
```

```
        hypnat_hypint of_int_eq_star_of starfun_def)
qed

lemma two_hyperpow_ge_one [simp]: "(1::hypint)  ≤ 2 pow n"
  by (simp add: hyperpow_gt_zero hypint_one_le_iff_zero_less)
```

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"
by transfer simp

lemma real_of_int_le_real_of_nat_iff: "(real_of_int a ≤ real_of_nat
b) = (a ≤ int b)"
proof
  assume "real_of_int a ≤ real b" then show "a ≤ int b"
    by auto
next
  assume "a ≤ int b" then show "real_of_int a ≤ real b"
    by linarith
qed

lemma minus_real_of_int_le_real_of_nat_iff: "(- real_of_int a ≤ real_of_nat
b) = (- a ≤ int b)"
  by linarith

lemma hypreal_of_hypint_le_hypreal_of_hypnat_iff:
      "⋀a b.(hypreal_of_hypint a ≤ of_hypnat b) = (a ≤ of_hypnat b)"
by transfer (auto simp add: real_of_int_le_real_of_nat_iff)

lemma minus_hypreal_of_hypint_le_hypreal_of_hypnat_iff:
      "⋀a b.(- hypreal_of_hypint a ≤ hypreal_of_hypnat b) = (- a ≤ of_hypnat
b)"
by transfer (auto simp add: minus_real_of_int_le_real_of_nat_iff)

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

lemma of_hypnat_hypnat [simp]: "⋀z. 0 ≤ z ⟹ of_hypnat(hypnat z) =
of_hypint z"
  by transfer simp
```

```
lemma HInfinite_def3: "HInfinite = {x. ∀z ∈ Ints. hypreal_of_hypint
z < hnorm x}"
proof (auto simp add: HInfinite_def2)
  fix x::"'a :: real_normed_vector star" and z::"int star"
  assume less_hnorm: "∀n∈ℕ. hypreal_of_hypnat n < hnorm x" and z: "z
∈ ℤ"
  then obtain z' where "z = of_int z'"
    using Ints_cases by blast
  moreover have  "∀n. hypreal_of_nat n < hnorm x"
    using less_hnorm of_nat_in_Nats by fastforce
  moreover have "of_int z' < hnorm x" using less_hnorm
    by (metis calculation(2) hnorm_ge_zero less_le_trans nonneg_int_cases
not_le
        of_int_less_0_iff of_int_of_nat_eq)
  ultimately show "hypreal_of_hypint z < hnorm x"
    by simp
next
  fix x::"'a :: real_normed_vector star" and n::"nat star"
  assume less_hnorm: "∀z∈ℤ. hypreal_of_hypint z < hnorm x" and n: "n
∈ ℕ"
  then have "hypint_of_hypnat n ∈ ℤ"
    by (metis Nats_cases hypint_of_hypnat_hypint_of_nat_eq hypint_of_int_mem_Ints
star_of_nat_def)
  then have "hypreal_of_hypint (hypint_of_hypnat n) < hnorm x"
    using  less_hnorm by blast
  then show "hypreal_of_hypnat n < hnorm x"
    by simp
qed

lemma HInfinite_of_hypint_HNatInfinite_hypnat:
  "⟦ (of_hypint x :: 'a::{real_normed_algebra_1,linordered_idom} star)
∈ HInfinite; x > 0 ⟧
       ⟹ hypnat x ∈ HNatInfinite"
by (rule nonneg_eq_hypint [of x], auto simp add: HNatInfinite_of_hypnat_HInfinite_iff)

lemma HNatInfinite_hypnat_HInfinite_of_hypint:
  "hypnat x ∈ HNatInfinite
      ⟹ (of_hypint x :: 'a::{real_normed_algebra_1,linordered_idom}
star) ∈ HInfinite"
  by (metis HNatInfinite_of_hypnat_HInfinite_iff hypnat_eq_iff2 of_hypint_of_hypnat

      zero_not_mem_HNatInfinite)

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)"*
by *(case_tac x, auto simp add: n_star_def star_n_eq_iff)*

lemma *star_n_star [simp]:*
"star_n (n_star x) = x"
by *(metis (full_types) n_star_def someI_ex star_cases)*

lemma *n_star_star_n_eq_ultra:*
"eventually (λn. n_star (star_n X) n = X n) 𝒰"
using *star_n_eq_iff* **by** *fastforce*

lemma *n_star_star_of_eq_ultra:*
"eventually (λn. n_star (star_of X) n = X) 𝒰"
using *star_of_def star_n_eq_iff* **by** *fastforce*

lemma *n_starset_def2: "*ns* S = (SOME Sn. S = *sn* Sn)"*
by *(simp add: n_starset_def starset_n_def)*

lemma *Iset_n_starset_eq [simp]:*
assumes *"X ∈ InternalSets"*

```
  shows "Iset (star_n (*ns* X)) = X"
proof -
  have "∃As. X = *sn* As" using assms
    using InternalSets_def by blast
  then have "Iset (star_n (SOME As. X = Iset (star_n As))) = X"
    by (metis (mono_tags) someI_ex starset_n_def)
  then show ?thesis
    by (simp add: n_starset_def)
qed

lemma Iset_n_starset_empty [simp]:
  "Iset(star_n (*ns* {})) = {}"
by simp

lemma n_starset_eq_ultra:
  "eventually (λn. (*ns* (Iset (star_n X))) n = X n) 𝒰"
proof (simp add: n_starset_def)
  have "∀_F n in 𝒰. (SOME As. Iset (star_n X) = Iset (star_n As)) n =
X n"
  proof  (rule someI2_ex [of "λx. Iset (star_n X) = Iset (star_n x)"])
    show "∃a. Iset (star_n X) = Iset (star_n a)"
      by force
    next fix x assume "Iset (star_n X) = Iset (star_n x)"
    then show "∀_F n in 𝒰. x n = X n"
      using Iset_eq_cancel by blast
  qed
  then show "∀_F n in 𝒰. (SOME As. Iset (star_n X) = Iset (star_n As))
n = X n" .
qed

lemma n_starset_starset_n_eq_ultra:
  "eventually (λn. (*ns* *sn* X) n = X n) 𝒰"
  by (simp add: starset_n_def n_starset_eq_ultra)

lemma n_starset_starset_n_eq_ultra2:
  "eventually (λn. X n = (*ns* *sn* X) n) 𝒰"
by (metis n_starset_starset_n_eq_ultra star_n_eq_iff)

lemma n_starset_empty_ultra:
      "eventually (λn. (*ns* {}) n = {}) 𝒰"
proof -
  have "∀_F n in 𝒰. (*ns* *sn* (λn. {})) n = {}"
    using n_starset_starset_n_eq_ultra by blast
  then show ?thesis
    using starset_n_empty by simp
qed

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

```
𝒰)"
by (metis n_starset_starset_n_eq_ultra2 starP_star_n star_n_eq_iff)

lemma InternalSets_starset_n_starset [simp]:
  assumes "A ∈ InternalSets"
  shows "*sn* (*ns* A) = A"
proof -
  have "∃a. A = Iset (star_n a)"
    using Iset_n_starset_eq assms by blast
  moreover {fix x
  assume "A = Iset (star_n x)"
  then have "*sn* x = A"
    by (simp add: starset_n_def)}
  ultimately have "*sn* (SOME As. A = Iset (star_n As)) = A"
    by (rule someI2_ex, simp)
  then show ?thesis
    by (simp add: n_starset_def)
qed

lemma starset_n_star_n:
  "(star_n X ∈ *sn* As) =  (eventually (λn. (X n) ∈ (As n)) 𝒰)"
by (metis Iset_star_n starset_n_def)

lemma n_starfun_def2: "*nf* F = (SOME Fn. F = *fn* Fn)"
  by (simp add:  n_starfun_def starfun_n_def)

lemma InternalFuns_starfun_n_starfun [simp]:
  assumes "F ∈ InternalFuns"
  shows "*fn* (*nf* F) = F"
proof -
  have "∃a. F = Ifun (star_n a)"
    using  assms
    by (simp add: InternalFuns_def starfun_n_def)
  moreover {fix x
  assume "F = Ifun (star_n x)"
  then have "*fn* x = F"
    by (simp add: starfun_n_def)}
  ultimately have "*fn* (SOME Fs. F = Ifun (star_n Fs)) = F"
    by (rule someI2_ex, simp)
  then show ?thesis
    by (simp add: n_starfun_def)
qed

lemma Ifun_eq_cancel:
  "(Ifun (star_n F) = Ifun (star_n G)) = (eventually (λn. F n = G n) 𝒰)"
by (auto intro!: transfer_fun_eq [THEN meta_eq_to_obj_eq]
     simp add: Ifun_star_n star_n_eq_iff)

lemma starfun_n_eq_cancel:
```

```
  "(*fn* F = *fn* G) = (eventually (λn. F n = G n) 𝒰)"
by (simp add: starfun_n_def Ifun_eq_cancel)

lemma n_starfun_starfun_n_eq_ultra:
  "eventually (λn. (*nf* (*fn* F)) n = F n) 𝒰"
using InternalFuns_def InternalFuns_starfun_n_starfun starfun_n_eq_cancel
by auto

lemma n_starfun_starfun_eq_ultra:
  "eventually (λn. (*nf* (*f* f)) n = f) 𝒰"
by (simp add: starfun_starfun_n_eq n_starfun_starfun_n_eq_ultra)

lemma P_n_starfun_starfun_n_ultra_iff:
   "eventually (λn. P ((*nf* *fn* X) n)) 𝒰 = (eventually (λn. P (X n))
𝒰)"
by (metis n_starfun_starfun_n_eq_ultra starP_star_n starP_star_n star_n_eq_iff)

lemma starfun2_eq_starfun2n: "*f2* f = *fn2* (λn. f)"
by (simp add: starfun2_n_def starfun2_def star_of_def)

lemma starfun2_n_Ifun_starfun_n: "*fn2* f = (λx. Ifun ((*fn* f) x))"
by (simp add: starfun2_n_def starfun_n_def)

lemma starfun2_n_Ifun_starfun_n2: "( *fn2* f) x y = ((*fn* f) x) ⋆ y"
by (simp add: starfun2_n_Ifun_starfun_n)
```

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))"
by (simp add: starfun2_n_def Ifun_star_n)

lemma n_starfun2_starfun2_n_eq_ultra:
  "eventually (λn. (*nf2* (*fn2* F)) n = F n) 𝒰"
proof -
  {
    fix x
    assume "(λy. Ifun (star_n F ⋆ y)) = (λxa. Ifun (star_n x ⋆ xa))"
    then have "∀X Xa. ∀_F n in 𝒰. F n (X n) (Xa n) = x n (X n) (Xa n)"
      by (auto simp add: fun_eq_iff all_star_eq Ifun_star_n star_n_eq_iff)
    then have "⋀X. ∀_F n in 𝒰. ∀xb. F n (X n) xb = x n (X n) xb"
      using transfer_all [THEN meta_eq_to_obj_eq, THEN iffD1, where  p2="λx.
True"]
      by fastforce
    then have "∀_F n in 𝒰. ∀ xa xb. F n xa xb = x n xa xb"
      using transfer_all [THEN meta_eq_to_obj_eq, THEN iffD1,
                          where P2="λn xa. ∀xb. F n xa xb = x n xa xb"
and p2="λx. True"]
      by force
    moreover have "∀_F n in 𝒰. (∀xa xb. F n xa xb = x n xa xb) ⟶ F
```

```
n = x n"
      by (simp add: ext)
    ultimately have "∀_F n in 𝒰. F n = x n"
      using not_eventually_impI
      by force
  }
  then have ev_fn2: ‹⋀x. (λy. Ifun (star_n F ⋆ y)) = (λxa. Ifun (star_n
x ⋆ xa)) ⟹ ∀_F n in 𝒰. F n = x n› .

  show ?thesis
    unfolding n_starfun2_def starfun2_n_def
    using someI2 [of "λGs. (λx. Ifun (star_n F ⋆ x)) = (λx. Ifun (star_n
Gs ⋆ x))",
                  OF _ ev_fn2, where x1="λx. x"]
    eventually_mono
    by fastforce
qed

lemma n_starfun2_starfun2_eq_ultra: "eventually (λn. (*nf2* (*f2* f))
n = f) 𝒰"
by (simp add: starfun2_eq_starfun2n n_starfun2_starfun2_n_eq_ultra)

lemma InternalFuns2_starfun2_n_starfun2 [simp]:
  assumes "f ∈ InternalFuns2"
  shows "*fn2* (*nf2* f) = f"
proof -
  have "∃a. f = (λx. Ifun (star_n a ⋆ x))"
    using assms
    by (simp add: InternalFuns2_def starfun2_n_def)
  moreover {fix x
  assume "f = (λxa. Ifun (star_n x ⋆ xa))"
  then have "*fn2* x = f"
    by (simp add: starfun2_n_def)}
  ultimately have "*fn2* (SOME Gs. f = (λx. Ifun (star_n Gs ⋆ x))) = f"
    by (rule someI2_ex, simp)
  then show ?thesis
    by (simp add: n_starfun2_def)
qed

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"
by (simp only: fun_eq_iff transfer_all)

lemma starfun2_n_eq_cancel:
  "(*fn2* F = *fn2* G) = (eventually (λn. F n = G n) 𝒰)"
by (auto intro!: transfer_fun2_eq [THEN meta_eq_to_obj_eq] simp add: starfun2_n_def
Ifun_star_n star_n_eq_iff)
```

```
lemma n_starfun_starfun2_n_eq_ultra:
  "∀_F n in 𝒰. (*nf* (*fn2* f) (star_n X)) n = f n (X n)"
proof -
  have "Ifun (star_n f ⋆ star_n X) = Ifun (star_n (λn. f n (X n)))"
    by (simp add: Ifun_star_n)
  then show ?thesis
    by (metis n_starfun_starfun_n_eq_ultra starfun2_n_def starfun_n_def)
qed

lemma starfun2_n_minus: "- (*fn2* f) x y = (*fn2* (λi j x. - (f i j)
x)) x y"
proof (cases x)
  case (star_n X)
  assume x_star: "x = star_n X"
  then show ?thesis
  proof (cases y)
    case (star_n Y)
    then show ?thesis
      using x_star by (simp add: starfun2_n star_n_minus star_n_diff)
  qed
qed

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 (cases y)
  case (star_n Y)
  assume y_star: "y = star_n Y"
  then show ?thesis
  proof(cases z)
    case (star_n Z)
    then show ?thesis
      using y_star by (simp add: starfun2_n star_n_minus star_n_diff)
  qed
qed

lemma starfun2_n_inverse: "inverse ((*fn2* f) x y) = (*fn2* (λi j x.
inverse ((f i j) x))) x y"
proof (cases x)
  case (star_n X)
  assume x_star: "x = star_n X"
  then show ?thesis
  proof (cases y)
    case (star_n Y)
    then show ?thesis
      using x_star by (auto simp add: starfun2_n star_n_inverse)
  qed
qed
```

```
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 (cases y)
  case (star_n Y)
  assume y_star: "y = star_n Y"
  then show ?thesis
  proof(cases z)
    case (star_n Z)
    then show ?thesis
      using y_star by (simp add: starfun2_n star_n_mult)
  qed
qed

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"
  by (simp add: divide_inverse starfun2_n_mult starfun2_n_inverse)

lemma starfun_n_starfun2_omega: "(*fn* F) z = (*f2* F) whn z"
by (cases z, auto simp add:  starfun2_star_n starfun_n star_n_eq_iff hypnat_omega_def)

lemma InternalFuns2_divide:
  "⟦ f ∈ InternalFuns2; g ∈ InternalFuns2 ⟧
       ⟹ (λm n. (f m n ::'a::division_ring star)/g m n) ∈ InternalFuns2"
  by (auto simp add: InternalFuns2_def starfun2_n_divide )

lemma InternalFuns2_mult:
  "⟦ f ∈ InternalFuns2; g ∈ InternalFuns2 ⟧ ⟹ (λm n. f m n * g m n)
∈ InternalFuns2"
by (auto simp add: InternalFuns2_def starfun2_n_mult)

lemma InternalFuns2_diff:
  "⟦ f ∈ InternalFuns2; g ∈ InternalFuns2 ⟧ ⟹ (λm n. f m n - g m n)
∈ InternalFuns2"
by (auto simp add: InternalFuns2_def starfun2_n_diff)

lemma InternalFuns2_minus:
  "f ∈ InternalFuns2  ⟹ (λm n. - f m n) ∈ InternalFuns2"
by (auto simp add: InternalFuns2_def starfun2_n_minus)

lemma InternalFuns_starfun2_n [simp]: "*fn2* f ∈ InternalFuns2"
  using InternalFuns2_def by auto

lemma ext_obj: "(∀x y. f x y = g x y) ⟹ f = g"
  using ext by blast
```

```
lemma InternalFuns2_const_fun: "(λn m. c) ∈ InternalFuns2"
proof (cases c, simp add: InternalFuns2_def)
  case (star_n X)
    show "∃F. (λn m. star_n X) = *fn2* F"
    proof (rule exI [of _ "λn m p. X n"])
      show "(λn m. star_n X) = *fn2* (λn m p. X n)"
      proof (rule ext)+
        fix n::"'f star" and m::"'g star"
        show "star_n X = (*fn2* (λn m p. X n)) n m"
          by (force intro: star_cases [of n] star_cases [of m]
              simp add: starfun2_n star_n_eq_iff)
      qed
    qed
qed

lemma InternalFuns2_InternalFuns_iff:
  "((λm n. f m) ∈ InternalFuns2) = (f ∈ InternalFuns)"
proof (auto simp add: InternalFuns_def InternalFuns2_def)
  fix F :: "nat ⇒ 'a ⇒ 'b ⇒ 'c"
  assume fn2F: "(λm n. f m) = *fn2* F"
  show "∃F. f = *fn* F"
  proof (rule exI [of _ "λm n. F m n p"], rule ext)
    {fix x
    have "f x = (*fn* (λm n. F m n p)) x"
    proof -
      have fx: "f x = (*fn2* F) x (star_of p)"
        using fn2F by metis
      then show ?thesis
      proof (cases x)
        case (star_n X)
        then show ?thesis
          using fx by (simp add: starfun_n starfun2_n star_n_eq_iff star_of_def)
      qed
    qed}
  then show "⋀x. f x = (*fn* (λm n. F m n p)) x"
    by blast
qed
next
  fix F
  assume "f = *fn* F"
  show "∃Fa. (λm n. (*fn* F) m) = *fn2* Fa"
  proof (rule exI [of _ "λn j k. F n j"],  (rule ext)+)
    {fix m::"'a star" and n::"'e star"
    have "(*fn* F) m = (*fn2* (λn j k. F n j)) m n"
    proof (cases m)
      case (star_n X)
      assume m: "m = star_n X"
      then show ?thesis
      proof (cases n)
```

```
      case (star_n Y)
      then show ?thesis by (auto simp add: m starfun_n starfun2_n)
    qed
  qed}
  then show "⋀m n. (*fn* F) m = (*fn2* (λn j k. F n j)) m n"
    by blast
 qed
qed

lemma InternalSets_hnorm_InternalFuns_gt [simp]:
  assumes "X ∈ InternalFuns"
  shows "{n. hnorm (X n) > m} ∈ InternalSets"
proof (cases m)
  case (star_n Y)
  assume m_star: "m = star_n Y"
  obtain A where internal_X: "X = *fn* A" using assms
    by (metis InternalFuns_starfun_n_starfun)
  have "{n. star_n Y < hnorm ((*fn* A) n)} =
            {x. (*p2* (∈)) x (star_n (λm. {n. Y m < norm (A m n)}))}"
  proof (safe)
    fix x
    assume less_hnorm: "star_n Y < hnorm ((*fn* A) x)"
    then show "(*p2* (∈)) x (star_n (λm. {n. Y m < norm (A m n)}))"
    proof(cases x)
      case (star_n X)
      then show ?thesis using less_hnorm
        by (simp add: starP2_star_n hnorm_def starfun_n starfun star_less_def)
    qed
  next
    fix x
    assume set_mem: "(*p2* (∈)) x (star_n (λm. {n. Y m < norm (A m n)}))"
    then show "star_n Y < hnorm ((*fn* A) x)"
    proof (cases x)
      case (star_n X)
      then show ?thesis using set_mem
        by (simp add: starP2_star_n hnorm_def starfun_n starfun star_less_def)
    qed
  qed
  then show ?thesis
    using InternalSets_def Iset_def internal_X star_n starset_n_def by
fastforce
qed

lemma InternalSets_hnorm_starfun_gt [simp]:
      "⋀m. {n. hnorm((*f* X) n) > m} ∈ InternalSets"
by (metis InternalFuns_starfun InternalSets_hnorm_InternalFuns_gt)
```

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 (cases "∀n. hnorm(X n) ≤ m")
  case True
  then show ?thesis
    using HNatInfinite_whn by blast
next
  case False
  have "{n. m < hnorm (X n)} ∈ InternalSets"
    using InternalSets_hnorm_InternalFuns_gt assms(1) by blast
  then have "∃n∈{n. m < hnorm (X n)}. ∀m∈{n. m < hnorm (X n)}. n ≤ m"
    by (metis False empty_iff leI mem_Collect_eq nonempty_InternalNatSet_has_least)
  then obtain na where na_prop: "m < hnorm (X na)" "∀m∈{n. m < hnorm
(X n)}. na ≤ m" by blast
  have "na ∈ HNatInfinite"
  proof (rule ccontr)
    assume "na ∉ HNatInfinite"
    then have "na ∈ ℕ"
      using HNatInfinite_not_Nats_iff by blast
    then show False
      using na_prop(1) assms(2) not_less by blast
  qed
  moreover have "∀n<na. hnorm (X n) ≤ m"
  proof (safe)
    fix n
    assume n_l_na: "n < na"
    show "hnorm (X n) ≤ m"
    proof (rule ccontr)
      assume "¬ hnorm (X n) ≤ m"
      then have "m < hnorm (X n)" by simp
      then have "na ≤ n" using na_prop(2)
        by blast
      then show False using n_l_na by simp
    qed
  qed
  ultimately show ?thesis by blast
qed

lemma starfun_least:
   "∀n∈Nats. hnorm((*f* X) n) ≤ m ⟹ ∃v∈HNatInfinite. ∀n<v. hnorm(
(*f* X) n) ≤ m"
by (metis InternalFuns_ub InternalFuns_starfun)
```

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 -
  have "∀n∈Nats. hnorm(of_hypnat n * f n) ≤ 1"
  proof
    fix n
    assume "(n :: nat star) ∈ ℕ"
    then have "f n * of_hypnat n ∈ Infinitesimal"
       using f_approx_zero Infinitesimal_HFinite_mult mem_infmal_iff
       by (metis HFinite_star_of Nats_hypnat_of_nat_iff of_hypnat_def
starfun_star_of)
    then show "hnorm (of_hypnat n * f n) ≤ 1"
      by (metis Infinitesimal_hnorm_iff hnorm_le_Infinitesimal hnorm_mult

          hnorm_one le_cases mult.commute one_not_Infinitesimal)
  qed
  then obtain v where Inf_v : "v∈HNatInfinite" and hnorm_le_1: "∀n<v.
hnorm (of_hypnat n * f n) ≤ 1"
    using Int_f  InternalFuns_ub [OF InternalFuns_mult_of_hypnat] by blast
  show ?thesis
  proof (rule bexI [of _ "v"])
    show "v ∈ HNatInfinite"
      by (simp add: Inf_v)
  next
    {fix n
    assume n_less_v: "n < v"
    then have "f n ≈ 0"
    proof (cases "n ∈ Nats")
      case True
      then show ?thesis
        using f_approx_zero by blast
    next
      case False
      then have "hypreal_of_hypnat n ≠ 0"
        by force
      then have "hnorm (f n) ≤ 1 / hypreal_of_hypnat n"
        using hnorm_le_1 n_less_v
        by (force simp add: zero_less_HNatInfinite le_divide_eq hnorm_mult
mult.commute)
      then show ?thesis
        by (metis False HNatInfinite_inverse_Infinitesimal HNatInfinite_not_Nats_iff

            hnorm_le_Infinitesimal inverse_eq_divide mem_infmal_iff)
```

```
    qed}
    then show "∀n<v. f n ≈ 0"
      by blast
  qed
qed

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 -
  have "(λn. f n - g n) ∈ InternalFuns"
    using assms(1,2) by (force simp add: InternalFuns_def starfun_n_diff)
  moreover have "∀n∈ℕ. f n - g n ≈ 0"
    using approx_minus_iff assms(3) by blast
  ultimately show ?thesis using InternalFuns_Sequential_Theorem
    by (metis (no_types, lifting) approx_minus_iff)
qed

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"
by (metis InternalFuns_Sequential_Theorem InternalFuns_starfun)

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"
by (metis InternalFuns_Sequential_Theorem2 InternalFuns_starfun)

lemma hyperpower_starfun_n:
  "(x::'a::power star) pow n = ( *fn* (λn m. (n_star x) n ^ m)) n"
  by (metis hyperpow star_n_star starfun_n)

lemma InternalFuns2_hyperpower [simp]: "(λm. (pow) x) ∈ InternalFuns2"
proof(cases x, simp add: InternalFuns2_def)
  case (star_n X)
   show "∃F. (λm. (pow) (star_n X)) = *fn2* F"
   proof(rule exI [where x="(λm n. (^) (X m))"],(rule ext)+)
    fix m::"'c star" and x
     show "star_n X pow x = (*fn2* (λm n. (^) (X m))) m x"
     proof(cases m, cases x)
       fix Xa Xaa
       assume nx: "m = star_n Xa" "x = star_n Xaa"
       then show ?thesis
         by (auto simp add: hyperpow starfun2_n)
     qed
```

```
  qed
qed


lemma hyperpow_approx_one_ex:
  assumes "x ≈ 1"
  shows "∃N∈HNatInfinite.∀n<N. x pow n ≈ (1::'a::real_normed_div_algebra
star)"
proof -
  have "∀n∈ℕ. x pow n ≈ 1"
    by (metis Nats_hypnat_of_nat_iff assms hrealpow_approx_one hyperpow_hypnat_of_nat)
  then show ?thesis
    using  InternalFuns_hyperpow
    by (force intro!: InternalFuns_Sequential_Theorem2)
qed

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)"
by (simp add: hyperfinite_def unstar_def star_of_def star_n_eq_iff)

lemma hyperfinite_starset_n_iff:
  "hyperfinite (*sn* X) =  (eventually (λn. finite (X n)) 𝒰)"
by (simp add: hyperfinite_def unstar_def star_of_def star_n_eq_iff
      P_n_starset_starset_n_ultra_iff)

lemma hyperfinite_empty [simp]: "hyperfinite {}"
  using eventually_mono hyperfinite_starset_n_iff n_starset_empty_ultra

  by force

lemma hyperfinite_Iset_iff_starP_finite:
   "hyperfinite (Iset X) = (*p* finite) X"
  by (metis (full_types) hyperfinite_starset_n_iff starP_star_n
      star_n_star starset_n_def)
```

```
lemma finite_hyperfinite_starset:
   "finite X ⟹ hyperfinite (*s* X)"
by (simp add: hyperfinite_Iset_iff_starP_finite starset_def)

lemma hyperfinite_starset_finite:
   "hyperfinite (*s* X) ⟹ finite X"
by (simp add: hyperfinite_Iset_iff_starP_finite starset_def)

lemma hyperfinite_starset_iff_finite:
   "hyperfinite (*s* X) = finite X"
by (blast intro: finite_hyperfinite_starset hyperfinite_starset_finite)
```

5.1 Properties of hyperfinite sets

```
lemma hyperfinite_UnI:
  "⟦ X ∈ InternalSets; Y ∈ InternalSets;
     hyperfinite X; hyperfinite Y ⟧ ⟹ hyperfinite (X ∪ Y)"
by (auto simp add: hyperfinite_iff InternalSets_def starset_n_Un [symmetric]

       P_n_starset_starset_n_ultra_iff eventually_conj_iff)

lemma hyperfinite_starset_n_UnI:
  "⟦ hyperfinite (*sn* X); hyperfinite (*sn* Y) ⟧ ⟹ hyperfinite (*sn*
X ∪ *sn* Y)"
by (auto intro: hyperfinite_UnI)

lemma hyperfinite_starset_n_Un [iff]:
  "hyperfinite ((*sn* X) ∪ (*sn* Y)) ⟷ hyperfinite (*sn* X) ∧ hyperfinite
(*sn* Y)"
by (simp add: starset_n_Un [symmetric] hyperfinite_starset_n_iff eventually_conj_iff)

lemma hyperfinite_singleton [simp]: "hyperfinite {x}"
proof -
  obtain X where starx: "x = star_n X"
    using star_cases by blast
  have "⋀f fa. ∀F n in fa. finite {f (n::nat)::'a}"
    by simp
  then have "hyperfinite (*sn* (λn. {X n}))"
    using hyperfinite_starset_n_iff by blast
  then show ?thesis
    by (simp add: starx)
qed

lemma finite_hyperfinite:
  assumes "finite X"
  shows "hyperfinite X"
proof (rule finite_induct)
  show "finite X" using assms .
next
```

```
  show "hyperfinite {}" by simp
next
  fix x::"'a star" and A::"'a star set"
  assume As: "finite A" "x ∉ A" "hyperfinite A"
  then have "A ∈ InternalSets" using finite_InternalSets by blast
  moreover have "{xa. xa = x} ∈ InternalSets" by simp
  moreover have "hyperfinite {xa. xa = x}" by simp
  ultimately have " hyperfinite ({xa. xa = x} ∪ A)"
    using hyperfinite_UnI As by blast
  then show "hyperfinite (insert x A)" by simp
qed

lemma hyperfinite_insert [simp]:
   "A ∈ InternalSets ⟹ hyperfinite (insert a A) ⟷ hyperfinite A"
by (metis InternalSets_singleton InternalSets_starset_n_starset
          hyperfinite_singleton hyperfinite_starset_n_Un insert_def singleton_conv)

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

lemma hyperfinite_subset:
  assumes "A ∈ InternalSets" and "B ∈ InternalSets" and
          "hyperfinite B" and "A ⊆ B"
  shows "hyperfinite A"
proof -
  have "∀S. (S::'a star set) ∉ InternalSets ∨ *sn* *ns* S = S"
    using InternalSets_starset_n_starset by blast
  then show ?thesis
    by (metis (no_types) assms hyperfinite_starset_n_Un le_iff_sup)
qed

lemma hyperfinite_starset_n_subset:
  "⟦ hyperfinite (*sn* Y); (*sn* X) ⊆ (*sn* Y) ⟧ ⟹ hyperfinite (*sn*
X)"
by (force intro: hyperfinite_subset)

lemma hyperfinite_Int [simp, intro]:
  "⟦ X ∈ InternalSets; Y ∈ InternalSets;
     hyperfinite X ∨ hyperfinite Y ⟧ ⟹ hyperfinite (X ∩ Y)"
 by (blast intro: hyperfinite_subset InternalSets_Int)


lemma hyperfinite_IntI1:
  "⟦ X ∈ InternalSets; Y ∈ InternalSets; hyperfinite X ⟧ ⟹ hyperfinite
(X ∩ Y)"
by (blast intro: hyperfinite_subset InternalSets_Int)

lemma hyperfinite_IntI2:
```

```
  "⟦ X ∈ InternalSets; Y ∈ InternalSets; hyperfinite Y ⟧ ⟹ hyperfinite
(X ∩ Y)"
by (blast intro: hyperfinite_subset InternalSets_Int)

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

lemma hyperfinite_starset_n_IntI1:
  "hyperfinite (*sn* X) ⟹ hyperfinite (*sn* X ∩ *sn* Y)"
by simp

lemma hyperfinite_starset_n_IntI2:
  "hyperfinite (*sn* Y) ⟹ hyperfinite (*sn* X ∩ *sn* Y)"
by simp

lemma hyperfinite_Diff [simp, intro]:
  "⟦ X ∈ InternalSets; Y ∈ InternalSets; hyperfinite X ⟧ ⟹ hyperfinite
(X - Y)"
by (blast intro: hyperfinite_subset InternalSets_diff)

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

lemma hyperfinite_Diff2 [simp]:
  assumes "X ∈ InternalSets"
  and "Y ∈ InternalSets"
  and "hyperfinite Y"
  shows "hyperfinite (X - Y) ⟷ hyperfinite X"
proof
  assume hyp: "hyperfinite (X - Y)"
  from assms obtain Xa Ya where XY: "X = *sn* Xa" "Y = *sn* Ya"
    by (metis InternalSets_starset_n_starset)
  then have "hyperfinite (*sn* Xa ∩ *sn* Ya)"
    using assms(3) by blast
  moreover have "hyperfinite (*sn* Xa - *sn* Ya)"
    using hyp XY by simp
  ultimately have " hyperfinite (*sn* (λn. Xa n - Ya n) ∪ *sn* (λn. Xa
n ∩ Ya n))"
    using hyperfinite_starset_n_UnI
    by (force simp only: starset_n_diff [symmetric]  starset_n_Int [symmetric])
  then  show "hyperfinite X" using XY(1)
    by (auto simp only: starset_n_Un [symmetric] Un_Diff_Int Int_commute)
next
  assume "hyperfinite X"
  then show "hyperfinite (X - Y)"
    by (simp add: assms(1) assms(2))
```

qed

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

lemma *hyperfinite_Diff_insert [iff]:*
"⟦ X ∈ InternalSets; Y ∈ InternalSets ⟧
⟹ hyperfinite (X - insert a Y) ⟷ hyperfinite (X - Y)"
by *(metis Diff_insert InternalSets_diff InternalSets_singleton*
hyperfinite_singleton hyperfinite_Diff2)

lemma *hyperfinite_starset_n_Diff_insert [iff]:*
*"hyperfinite (*sn* X - insert a (*sn* Y)) ⟷ hyperfinite (*sn* X -*
**sn* Y)"*
by *simp*

lemma *hyperfinite_Compl[simp]:*
"⟦ X ∈ InternalSets; hyperfinite (X :: 'a star set) ⟧
⟹ hyperfinite (- X) ⟷ hyperfinite (UNIV :: 'a star set)"
by *(metis Compl_eq_Diff_UNIV NatStar.InternalSets_starset_n hyperfinite_Diff2*
starset_UNIV)

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

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)"
by *(simp add: Collect_neg_eq)*

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

lemma *hypnat_hyperfinite_atLeastLessThan [simp]:*
"hyperfinite {M::hypnat..<N}"
proof *(cases M, cases N, simp)*
fix *X Xa*
assume *"M = star_n X" "N = star_n Xa"*
then show *"hyperfinite {star_n X..<star_n Xa}"*
by *(simp add: hyperfinite_iff starset_n_atLeastLessThan_eq [symmetric]*

```
            P_n_starset_starset_n_ultra_iff)
qed

lemma hyperfinite_hypnat_set_atLeastAtMost:
   "hyperfinite ({of_nat m .. of_nat n} :: hypnat set)"
by (metis finite_atLeastAtMost hyperfinite_starset_iff_finite starset_atLeastAtMost)
```

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(simp add: hyperfold_def star_n_eq_iff)
  have "∀_F n in 𝒰. (*nf2* *fn2* Fs) n = Fs n" (is "eventually ?A 𝒰")
    by (simp add: n_starfun2_starfun2_n_eq_ultra)
  moreover have "∀_F n in 𝒰. (*ns* *sn* As) n = As n" (is "eventually
?B 𝒰")
    by (simp add: n_starset_starset_n_eq_ultra)
  moreover have "∀_F n in 𝒰. n_star (star_n Zs) n = Zs n" (is "eventually
?C 𝒰")
    by (simp add: n_star_star_n_eq_ultra)
  ultimately have "eventually (λn. ?A n ∧ ?B n ∧ ?C n) 𝒰"
    by (simp add: eventually_conj_iff)
  then show "∀_F n in 𝒰.
       Finite_Set.fold ((*nf2* *fn2* Fs) n) (n_star (star_n Zs) n) ((*ns*
*sn* As) n) =
       Finite_Set.fold (Fs n) (Zs n) (As n)"
    using eventually_mono by force
qed


lemma hyperfold_empty [simp]: "f ∈ InternalFuns2 ⟹ hyperfold f z {}
= z"
proof -
  obtain Z where starz: "z = star_n Z"
    using star_cases by blast
  moreover assume "f ∈ InternalFuns2"
  then obtain F where starf: "f = *fn2* F"
    by (metis InternalFuns2_starfun2_n_starfun2)
  moreover have "∀_F n in 𝒰. n_star (star_n Z) n = Z n"
    using n_star_star_n_eq_ultra by blast
  moreover have "∀_F n in 𝒰. (*nf2* *fn2* F) n = F n"
    by (simp add: n_starfun2_starfun2_n_eq_ultra)
```

```
  moreover have "eventually (λn. (*ns* {}) n = ({} :: 'a set)) 𝒰"
    using n_starset_empty_ultra by blast
  ultimately show ?thesis
    by (auto elim!: eventually_rev_mp simp add: hyperfold_def star_n_eq_iff
)
qed

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 (simp add: hypersum_def star_n_eq_iff)
  have "∀F n in 𝒰. (*nf* *fn* f) n = f n"
    using n_starfun_starfun_n_eq_ultra by blast
  moreover have "∀F n in 𝒰. (*ns* *sn* A) n = A n"
    using n_starset_starset_n_eq_ultra by blast
  ultimately have "∀F n in 𝒰. (*nf* *fn* f) n = f n ∧ (*ns* *sn* A) n
= A n"
      by (simp add: eventually_conj_iff)
  then show "∀F n in 𝒰. sum ((*nf* *fn* f) n) ((*ns* *sn* A) n) = sum
(f n) (A n)"
    using eventually_mono by force
qed

lemma hypersum_starfun2_n:
  "hypersum ((*fn2* f) (star_n X)) (*sn* A) = star_n (λn. sum (f n (X
n)) (A n))"
proof (simp add: hypersum_def star_n_eq_iff)
   have "∀F n in 𝒰. (*ns* *sn* A) n = A n"
```

```
    by (simp add: n_starset_starset_n_eq_ultra)
  moreover have "∀_F n in 𝒰. (*nf* (*fn2* f) (star_n X)) n = f n (X
n)"
    by (simp add: n_starfun_starfun2_n_eq_ultra)
  ultimately have "∀_F n in 𝒰.
                   (*ns* *sn* A) n = A n ∧
                   (*nf* (*fn2* f) (star_n X)) n = f n (X n)"
    by (simp add: eventually_conj_iff)
  then show "∀_F n in 𝒰.
                  sum ((*nf* (*fn2* f) (star_n X)) n) ((*ns* *sn* A)
n) =
                  sum (f n (X n)) (A n)"
   using eventually_mono by force
qed
```

6.1 Hypersum properties

```
lemma hypersum_empty [simp]:
  assumes "f ∈ InternalFuns" shows "hypersum f {} = 0"
proof -
  obtain F where fF: "f = *fn* F"
    by (metis InternalFuns_starfun_n_starfun assms)
  then have "∀_F n in 𝒰. (*nf* *fn* F) n = F n"
    using n_starfun_starfun_n_eq_ultra by blast
  then have "∀_F n in 𝒰. sum ((*nf* *fn* F) n) ((*ns* {}) n) = 0"
    using n_starset_empty_ultra  eventually_mono sum.empty
    by (metis (mono_tags, lifting))
  then show ?thesis using fF
    by (auto simp add: hypersum_def InternalFuns_def star_of_def
        star_n_eq_iff star_zero_def)
qed

lemma hypersum_starfun_n_empty [simp]: " hypersum (*fn* f) {} = 0"
by (metis InternalFuns_starfun_n hypersum_empty)


lemma setsum_singleton [simp]: "sum f {x} = f x"
by simp

lemma hypersum_singleton [simp]:
  assumes "f ∈ InternalFuns"
  shows "hypersum f {a} = f a"
proof (cases a)
  case (star_n X)
  assume a: "a = star_n X"
  have X: "{star_n X} = *sn* (λn. {X n})"
    by simp
  moreover obtain F where fF: "f = *fn* F"
    by (metis InternalFuns_starfun_n_starfun assms)
```

```
  ultimately have "∀F n in 𝒰. sum (F n) {X n} = F n (X n)"
    by (meson eventuallyI setsum_singleton)
  then show ?thesis using a fF X
    by (metis (full_types) hypersum star_n_eq_iff starfun_n)
qed

lemma hypersum_F_neutral_ivl [simp]:
  "hypersum (λn. 0) {m..<n} = 0"
proof -
  have f_0: "(λn. 0) = *fn* (λn m. 0)"
    by force
  have "{m..<n} ∈ InternalSets"
    by simp
  then show ?thesis
    by (auto simp add: InternalSets_def f_0 hypersum star_n_zero_num [symmetric])
qed

lemma hypersum_head_hSuc:
  assumes "f ∈ InternalFuns" and "m ≤ n"
  shows "hypersum f {m..n} = f m + hypersum f {hSuc m..n}"
proof -
  obtain F where f: "f = *fn* F" using assms(1)
    by (metis InternalFuns_starfun_n_starfun)
  then have "hypersum (*fn* F) {m..n} =
             (*fn* F) m + hypersum (*fn* F) {hSuc m..n}"
  proof (cases m, cases n)
    fix X Xa
    assume mn: "m = star_n X" "n = star_n Xa"
    then have " hypersum (*fn* F) {star_n X..star_n Xa} =
        (*fn* F) (star_n X) + hypersum (*fn* F) {hSuc (star_n X)..star_n
Xa}"
      using assms(2)
      by (auto  simp add: star_le_def starP2_star_n hSuc_def
            starset_n_atLeastAtMost [symmetric] starfun_star_n hypersum
            starfun_n star_n_add star_n_eq_iff eventually_mono  sum.atLeast_Suc_atMost)
    then show ?thesis using mn by simp
  qed
  then show ?thesis
    using f by simp
qed

lemma hypersum_head_upt_hSuc:
  assumes " f ∈ InternalFuns" and "m < n"
  shows "hypersum f {m..<n} = f m + hypersum f {hSuc m..<n}"
proof -
  obtain F where f: "f = *fn* F" using assms(1)
    by (metis InternalFuns_starfun_n_starfun)
  then have "hypersum (*fn* F) {m..<n} =
             (*fn* F) m + hypersum (*fn* F) {hSuc m..<n}"
```

```
  proof (cases m, cases n)
    fix X Xa
    assume mn: "m = star_n X" "n = star_n Xa"
    then have " hypersum (*fn* F) {star_n X..<star_n Xa} =
        (*fn* F) (star_n X) + hypersum (*fn* F) {hSuc (star_n X)..<star_n
Xa}"
      using assms(2)
      by (auto  simp add: star_less_def starP2_star_n hSuc_def starfun_star_n
            starset_n_atLeastLessThan_eq [symmetric] hypersum starfun_n
            star_n_add star_n_eq_iff  eventually_mono sum.atLeast_Suc_lessThan)
    then show ?thesis using mn by simp
  qed
  then show ?thesis
    using f by simp
qed

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 (cases m, cases n, cases p)
  fix X Xa Xb
  assume mnp: "m = star_n X" "n = star_n Xa" "p = star_n Xb"
  then have ev_le: "∀_F n in 𝒰. X n ≤ Xa n + 1" using assms(2)
    by (simp add: star_add_def star_n_le star_n_one_num starfun2_star_n)
  moreover
    obtain F where internal_f: "f = *fn* F"
    by (metis InternalFuns_starfun_n_starfun assms(1))
  then
   have " ∀_F n in 𝒰.
          sum (F n) {X n..Xa n + Xb n} =
          sum (F n) {X n..Xa n} + sum (F n) {Xa n + 1..Xa n + Xb n}"
     using sum.ub_add_nat eventually_mono [OF ev_le]
     by (metis (mono_tags, lifting))
  then have "hypersum (*fn* F) {m..n + p} =
           hypersum (*fn* F) {m..n} + hypersum (*fn* F) {n + 1..n +
p}"
    using mnp
    by (auto simp only: star_n_add starset_n_atLeastAtMost [symmetric]
hypersum
           star_n_one_num star_le_def starP2_star_n star_n_eq_iff )
  then show ?thesis
    using internal_f by blast
qed


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 (cases m, cases n)
  fix X Xa
  assume mn: "m = star_n X" "n = star_n Xa"
  obtain F where internal_f: "f = *fn* F"
    by (metis InternalFuns_starfun_n_starfun assms(1))
  then show ?thesis
  proof (cases "n < m")
    case True
    then show ?thesis
      using mn internal_f
      by (auto simp add: starset_n_atLeastLessThan_eq [symmetric] hSuc_def
starfun hypersum
           star_n_eq_iff star_less_def starP2_star_n star_n_zero_num
eventually_mono)
  next
    case False
    then show ?thesis
      using mn internal_f
      by (auto simp add: starset_n_atLeastLessThan_eq [symmetric] hSuc_def
starfun hypersum
            star_n_eq_iff star_less_def starP2_star_n star_n_add starfun_n

            FreeUltrafilterNat.eventually_imp_iff )
  qed
qed

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 (cases a)
  case (star_n X)
  assume a: "a = star_n X"
  obtain F As where f: "f = *fn* F" and A: "A = *sn* As"
    by (metis InternalFuns_starfun_n_starfun InternalSets_starset_n_starset
assms(1) assms(2))
  then have "∀_F n in 𝒰. finite (As n)"
    using assms(3) hyperfinite_starset_n_iff by blast
  moreover have "∀_F x in 𝒰. X x ∉ As x"
    using A assms(4) FreeUltrafilterNat.ultra star_n starset_n_star_n
by blast
  ultimately have "∀_F n in 𝒰. sum (F n) (insert (X n) (As n)) = F n (X
n) + sum (F n) (As n)"
    using eventually_elim2 by fastforce
  then show ?thesis
```

```
    using f A a
    by (auto simp add: insert_star_n_starset_n hypersum starfun_n star_n_eq_iff
star_n_add)
qed

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 -
  obtain F As Bs where f: "f = *fn* F" and A: "A = *sn* As" and B: "B
= *sn* Bs"
    by (metis InternalFuns_starfun_n_starfun InternalSets_starset_n_starset
assms(1) assms(2) assms(3))
  then have "∀_F n in 𝒰. Bs n ⊆ As n"
    using assms(4) by (auto simp add: starset_n_subset)
  moreover have "∀_F n in 𝒰. finite (As n)"
    using A assms(5) hyperfinite_starset_n_iff by blast
  ultimately have "∀_F n in 𝒰. sum (F n) (As n) = sum (F n) (As n - Bs
n) + sum (F n) (Bs n)"
    using eventually_elim2 by (fastforce simp add: sum.subset_diff)
  then show ?thesis
    using f A B by  (auto simp add: hypersum starset_n_diff [symmetric]
star_n_add star_n_eq_iff)
qed

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))"
  by (metis add_diff_cancel assms hypersum_subset_diff)

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 -
  obtain F M N P
    where f: "f = *fn* F" and m: "m = star_n M" and n: "n = star_n N"
and p: "p = star_n P"
    by (metis InternalFuns_starfun_n_starfun assms(1) star_cases)
  then have "∀_F n in 𝒰. M n ≤ N n"
```

```
    using assms(2) star_n_le by blast
  moreover have "∀_F n in 𝒰. N n ≤ P n"
    using assms(3) n p star_n_le by blast
  ultimately have "∀_F n in 𝒰. sum (F n) {M n..<N n} + sum (F n) {N n..<P
n}
                                = sum (F n) {M n..<P n}"
    using eventually_elim2
    by (fastforce simp add: sum.atLeastLessThan_concat)
  then show ?thesis
    using f m n p by (auto simp only: starset_n_atLeastLessThan_eq [symmetric]

                      hypersum star_n_add star_n_eq_iff star_le_def starP2_star_n)
qed

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}"
  by (metis add_diff_cancel_left' hypersum_add_hypnat_ivl)

lemma InternalFuns2_hypersum:
  assumes "(λj i. f j i) ∈ InternalFuns2"
  shows "(λi. hypersum (f i) {m..n}) ∈ InternalFuns"
proof -
  obtain F M N where f: "f = *fn2* F" and m: "m = star_n M" and n: "n
= star_n N"
    by (metis InternalFuns2_starfun2_n_starfun2 assms star_cases)
   have "∃Fa. (λi. hypersum ((*fn2* F) i) {m..n}) = *fn* Fa"
   proof (rule exI [where x="λn i. sum (F n i) {M n..N n}"], rule ext)
     fix i::"'a star"
     obtain I where i: "i = star_n I"
       using star_cases by blast
      then show  "hypersum ((*fn2* F) i) {m..n} = (*fn* (λn i. sum (F
n i) {M n..N n})) i"
        using m n
        by (auto simp add: starfun_n  starset_n_atLeastAtMost [symmetric]
hypersum_starfun2_n)
    qed
    then show ?thesis using f by auto
qed

lemma hypersum_starfun_atLeastAtMost:
  "hypersum (*f* f) {M..N} = (*f2* (λm n. sum f {m..n})) M N"
proof (cases M, cases N)
  fix X Xa
  assume "M = star_n X" "N = star_n Xa"
  then show "hypersum (*f* f) {M..N} = (*f2* (λm n. sum f {m..n})) M
N"
    by (auto simp add: starset_n_atLeastAtMost [symmetric]
```

```
        hypersum starfun_starfun_n_eq starfun2_star_n)
qed

lemma hypersum_starfun_atLeastLessThan:
  "hypersum (*f* f) {M..<N} = (*f2* (λm n. sum f {m..<n})) M N"
proof (cases M, cases N)
  fix X Xa
  assume "M = star_n X" "N = star_n Xa"
  then show "hypersum (*f* f) {M..<N} = (*f2* (λm n. sum f {m..<n}))
M N "
    by (auto simp add: starset_n_atLeastLessThan_eq [symmetric]
        hypersum starfun_starfun_n_eq starfun2_star_n)
qed

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 -
  obtain F M N where f: "f = *fn* F" and m: "m = star_n M" and n: "n
= star_n N"
    by (metis InternalFuns_starfun_n_starfun assms star_cases)
   have "∀_F n in 𝒰.
              (∑ i = M n..N n. F n (Suc i)) =
              (∑ m = M n..N n. F n (Suc m))"
    by simp
  then show ?thesis using f m n
    by (auto simp only: hSuc_def starfun starset_n_atLeastAtMost [symmetric]
starfun_n
        starfun_starfun_n_eq starfun_n_o hypersum star_n_eq_iff  sum.shift_bounds_cl_Suc_iv

qed

lemma hypersum_shift_bounds_hSuc_ivl:
  assumes "f ∈ InternalFuns"
  shows "hypersum f {hSuc m..<hSuc n} = hypersum (λi. f(hSuc i)){m..<n}"
proof -
  obtain F M N where f: "f = *fn* F" and m: "m = star_n M" and n: "n
= star_n N"
    by (metis InternalFuns_starfun_n_starfun assms star_cases)
  have "∀_F n in 𝒰.
              (∑ i = M n..<N n. F n (Suc i)) =
              (∑ m = M n..<N n. F n (Suc m))"
    by simp
  then show ?thesis using f m n
    by (auto simp only: hSuc_def starfun starset_n_atLeastLessThan_eq
[symmetric] starfun_n
        starfun_starfun_n_eq starfun_n_o hypersum star_n_eq_iff sum.shift_bounds_Suc_ivl)
qed
```

```
lemma lemma_n_starfun2_add_eq:
"eventually (λn. ((*nf2* (λx. (+) ((*fn* f) x))) n) = (λx. (+) (f n x)))
𝒰"
proof (simp add: n_starfun2_def)
  show "∀_F n in 𝒰.
       (SOME Gs.
           (λx. (+) ((*fn* f) x)) = (λx. Ifun (star_n Gs ⋆ x)))
        n =
       (λx. (+) (f n x))"
  proof (rule someI2 [of "(λGs. (λx. (+) ((*fn* f) x)) = (λx. Ifun (star_n
Gs ⋆ x)))", of "λn. (λx. (+) (f n x))"])
    show  "(λx. (+) ((*fn* f) x)) = (λx. Ifun (star_n (λn x. (+) (f n
x)) ⋆ x))"
    proof (rule ext)+
      fix x::"'a star" and xa::"'b star"
      obtain X Xa where x: "x = star_n X" and xa: "xa = star_n Xa"
        by (metis star_n_star)
      then show " (*fn* f) x + xa = star_n (λn x. (+) (f n x)) ⋆ x ⋆
xa"
        by (simp add: Ifun_star_n starfun_n star_n_add)
    qed
  next
    fix x
    assume assm: "(λx. (+) ((*fn* f) x)) = (λxa. Ifun (star_n x ⋆ xa))"
    then have "∀xa xb. star_of (+) ⋆ (star_n f ⋆ xa) ⋆ xb = star_n x
⋆ xa ⋆ xb"
      by (metis star_add_def starfun2_def starfun_n_def)
      {fix xa xb
        have "star_n x ⋆ xa ⋆ xb = star_n (λn x. (+) (f n x)) ⋆ xa ⋆ xb"
          by (metis assm star_add_def starfun2_def starfun_def starfun_n_def
starfun_starfun_n_o)
      }
      then have "*fn2* x = *fn2* (λn x. (+) (f n x))"
        by (auto simp add: star_add_def starfun2_def starfun_n_def fun_eq_iff
starfun2_n_def)
      then show "∀_F n in 𝒰. x n = (λx. (+) (f n x))"
        by (rule starfun2_n_eq_cancel [THEN iffD1])
  qed
qed


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 (auto simp add: sum.eq_fold)
  assume "finite A"
  then show "Finite_Set.fold ((+) ∘ f) 0 A = Finite_Set.fold (λx. (+)
(f x)) 0 A"
    by (meson comp_apply)
```

qed

lemma *hypersum_def2:*
  assumes *"f ∈ InternalFuns"*
  shows *"hypersum f A = (if hyperfinite A then hyperfold_image (+) f 0 A else 0)"*
proof -
  obtain *F* **where** *f: "f = *fn* F"*
    by *(metis InternalFuns_starfun_n_starfun assms)*
  then show *?thesis*
  proof *(cases "hyperfinite A")*
    assume *"hyperfinite A"*
    then have *"∀_F n in 𝒰. finite ((*ns* A) n)"*
      using *hyperfinite_iff* **by** *blast*
    moreover have *"∀_F x in 𝒰. n_star (star_n (λn. 0 :: 'b)) x = 0"*
      by *(metis n_star_star_of_eq_ultra star_of_def)*
    ultimately have *"∀_F x in 𝒰. finite ((*ns* A) x) ∧*
                        *(*nf* *fn* F) x = F x ∧*
                        *(*nf2* (λx. (+) ((*fn* F) x))) x = (λxa. (+) (F x xa)) ∧*
                        n_star (star_n (λn. 0 :: 'b)) x = 0"
      by *(auto intro!: eventually_conj simp add: lemma_n_starfun2_add_eq n_starfun_starfun_n_eq_ultra hyperfinite_iff)*
    then have *"∀_F n in 𝒰.*
                *sum ((*nf* *fn* F) n) ((*ns* A) n) =*
                *Finite_Set.fold ((*nf2* (λx. (+) ((*fn* F) x))) n)*
                 *(n_star (star_n (λn. 0)) n) ((*ns* A) n)"*
      by *(fastforce intro: eventually_mono simp add: setsum_def2)*
    then show *?thesis* **using** *f*
      by *(simp add: <hyperfinite A> hyperfold_def hyperfold_image_def hypersum_def star_n_eq_iff star_n_zero_num)*
    next
      assume *not_hyperfiniteA: "¬ hyperfinite A"*
      then have *"∀_F n in 𝒰. ¬ finite ((*ns* A) n)"*
        using *FreeUltrafilterNat.ultra hyperfinite_iff* **by** *blast*
      then have *"∀_F n in 𝒰. sum ((*nf* *fn* F) n) ((*ns* A) n) = 0"*
        using *eventually_mono* **by** *fastforce*
      then show *?thesis*
        by *(auto simp add: f not_hyperfiniteA hypersum_def star_zero_def star_of_def star_n_eq_iff)*
  qed
qed

6.2 The sumhr function as a hypersum.

lemma *sumhr_hypersum_eq:*
  *"sumhr(M,N,f) = hypersum (*f* f) {M..<N}"*
  by *(simp add: hypersum_starfun_atLeastLessThan sumhr_app)*

```
lemma NSsums_hypersum_HNatInfinite_approx_iff:
  "(f NSsums s) = (∀N∈HNatInfinite. hypersum (*f* f) {0..<N} ≈ of_real
s)"
proof (auto simp add: NSsums_def NSLIMSEQ_def)
  fix N
  assume "∀N∈HNatInfinite. (*f* (λn. sum f {..<n})) N ≈ hypreal_of_real
s"
  and "N ∈ HNatInfinite"
  then have starfun_sum: "(*f* (λn. sum f {..<n})) N ≈ hypreal_of_real
s" by blast
  then show "hypersum (*f* f) {0..<N} ≈ hypreal_of_real s"
  proof (cases N)
    case (star_n X)
    then show ?thesis using starfun_sum
      by (auto simp add: starset_n_atLeast_zero_LessThan_eq [symmetric]

          starfun_starfun_n_eq hypersum starfun_n atLeast0LessThan)
  qed
next
  fix N
  assume  "∀N∈HNatInfinite. hypersum (*f* f) {0..<N} ≈ hypreal_of_real
s"
  and "N ∈ HNatInfinite"
  then have hypsetsum: "hypersum (*f* f) {0..<N} ≈ hypreal_of_real s"
by blast
  then show "(*f* (λn. sum f {..<n})) N ≈ hypreal_of_real s"
  proof (cases N)
    case (star_n X)
    then show ?thesis using  hypsetsum
      by (auto simp add: starset_n_atLeast_zero_LessThan_eq [symmetric]

          starfun_starfun_n_eq hypersum starfun_n atLeast0LessThan)
  qed
qed
```

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)"
by (simp add: HyperCauchy_def NSCauchy_def)
```

```
lemma Cauchy_HyperCauchy_starfun_iff:
  "Cauchy X = HyperCauchy (*f* X)"
by (metis NSCauchy_Cauchy_iff NSCauchy_HyperCauchy_starfun_iff)

lemma NSsummable_HyperSummable_starfun_iff:
  "NSsummable f = HyperSummable (*f* f)"
  by (smt HyperCauchy_def HyperSummable_def NSsummable_NSCauchy NSsummable_NSsums
      NSsums_sumhr_HNatInfinite_approx_iff approx_trans2 sumhr_hrabs_approx
sumhr_hypersum_eq)

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 (rule ext)
  fix N
  show "(*f* (λn. sum f {k n..<g n})) N = hypersum (*f* f) {(*f* k) N..<(*f*
g) N}"
  proof (cases N)
    case (star_n X)
    then show ?thesis
      by (auto simp add:  starfun_starfun_n_eq starfun_n hypersum
       starset_n_atLeastLessThan_eq [symmetric])
  qed
qed


lemma starfun_setsum_atLeast_zero_hypesetsum_fun:
  "(*f* (λn. sum f {0..<g n})) = (λN. hypersum (*f* f) {0..< ((*f* g)
N)})"
proof (rule ext)
  fix N
  show "(*f* (λn. sum f {0..<g n})) N = hypersum (*f* f) {0..<(*f* g)
N}"
  proof (cases N)
    case (star_n X)
    then show ?thesis
      by (auto simp add:  starfun_starfun_n_eq starfun_n hypersum
       starset_n_atLeast_zero_LessThan_eq [symmetric])
  qed
qed

lemma starfun_setsum_atLeast_zero_hypesetsum:
  "(*f* (λn. sum f {0..<n})) = (λN. hypersum (*f* f) {0..<N})"
using starfun_setsum_atLeast_zero_hypesetsum_fun [where g="λx. x"] by
auto

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

```
lemma hypersum_starfun_atLeast0LessThan:
  "hypersum (*f* f) {0..<N::nat star} = (*f* (λn. ∑i<n. f i)) N"
  using hypersum_atLeast_zero_starfun lessThan_atLeast0 by auto


lemma hypersum_atLeast_zero_star_n_starfun_n:
  "hypersum (*fn* f) {0..<star_n X} = star_n((λn. sum (f n) {0..<X n}))"
  by (auto simp add: starset_n_atLeast_zero_LessThan_eq [symmetric] hypersum
star_n_eq_iff)


lemma hypersum_atLeast_zero_starfun_n:
  "hypersum (*fn* f) {0..<N} =  (*fn* (λn m. sum (f n) {0..<m})) N"
proof (cases N)
  case (star_n X)
  then show ?thesis
    by (auto simp add: hypersum_atLeast_zero_star_n_starfun_n starfun_n)
qed

lemma hypersum_geometric:
  assumes "x ≠ 1"
  shows "hypersum (λn. x pow n) {0..<n} = (x pow n - 1) / (x - 1::'a::field
star)"
proof -
  obtain X N where xn: "x = star_n X" "n = star_n N"
    by (metis star_n_star)
  then have "∀F n in 𝒰. X n ≠ 1" using  assms
    by (simp add: star_n_one_num star_n_eq_iff
                  FreeUltrafilterNat.eventually_not_iff [symmetric])
  moreover have "∀F n in 𝒰. n_star (star_n X) n = X n"
    by (simp add: n_star_star_n_eq_ultra)
  ultimately have "∀F n in 𝒰.
              sum ((^) (n_star (star_n X) n)) {0..<N n} =
              (n_star (star_n X) n ^ N n - 1) / (X n - 1)"
    by (force elim: eventually_elim2  simp add: atLeast0LessThan geometric_sum)

  then show ?thesis
    by (simp add: xn hyperpower_starfun_n hypersum_atLeast_zero_starfun_n
starfun_n
          star_n_eq_iff star_n_one_num star_n_diff star_n_divide)
qed

lemma HyperSummable_geometric:
  assumes "hnorm (x::'a::{real_normed_field} star) < 1"
  and "¬hnorm x ≈ 1"
  shows "HyperSummable (λN. x pow N)"
proof (auto simp add: HyperSummable_def HyperCauchy_def)
  fix M N
```

```
  assume MN: "M ∈ HNatInfinite" "N ∈ HNatInfinite"
  then show "hypersum ((pow) x) {0..<M} ≈ hypersum ((pow) x) {0..<N}"
  proof (cases "x = 1")
    case True
    then show ?thesis
      using assms(2) by simp
  next
    case False
    assume x_not_1: "x ≠ 1"
    have "x ∈ HFinite" using assms
      by (force intro: bexI [of _ 2] simp add:  HFinite_def)
    then have "x - 1 ∈ HFinite"
      by (simp add: HFinite_diff)
    moreover have "x - 1 ∉ Infinitesimal"
      using assms(2) bex_Infinitesimal_iff
      by (metis approx_hnorm hnorm_one)
    moreover have "x pow M - 1 ≈ x pow N - 1" using assms MN
      by (metis Infinitesimal_approx Infinitesimal_hyperpow_HNatInfinite
approx_diff approx_refl)
    ultimately have "(x pow M - 1) / (x - 1) ≈ (x pow N - 1) / (x - 1)"
      by (simp add: approx_divide_HFinite_diff_Infinitesimal)
    then show ?thesis
      by (simp add: x_not_1 hypersum_geometric)
  qed
qed

lemma summable_convergent_sumr_iff:
 "summable f = convergent (λn. sum f {0..<n})"
  by (simp add: atLeast0LessThan summable_iff_convergent)


lemma HyperSummable_starfun_summable_iff:
  "(HyperSummable ( (*f* f)::hypnat => 'a::banach star)) = (summable f)"
by (simp add: HyperSummable_def summable_convergent_sumr_iff Cauchy_convergent_iff
[symmetric]
            Cauchy_HyperCauchy_starfun_iff starfun_setsum_atLeast_zero_hypesetsum)

lemma HyperSummable_def2:
  "HyperSummable f = (∃s. ∀N∈HNatInfinite. hypersum f {0..<N} ≈ s)"
proof (auto simp add: HyperSummable_def HyperCauchy_def)
  assume "∀M∈HNatInfinite.
       ∀N∈HNatInfinite. hypersum f {0..<M} ≈ hypersum f {0..<N}"
  then show "∃s. ∀N∈HNatInfinite. hypersum f {0..<N} ≈ s"
    by metis
next
  fix M N s
  assume "M ∈ HNatInfinite" "N ∈ HNatInfinite"
         "∀N∈HNatInfinite. hypersum f {0..<N} ≈ s"
  then show "hypersum f {0..<M} ≈ hypersum f {0..<N}"
```

```
    using approx_trans2 by blast
qed

lemma HyperSummable_def3:
  assumes "f ∈ InternalFuns"
  shows "HyperSummable f = (∀M∈HNatInfinite.∀N∈HNatInfinite. hypersum
f {M..<N} ≈ 0)"
proof (simp add: HyperSummable_def HyperCauchy_def, safe)
  fix M N
  assume approxsums: "∀M∈HNatInfinite.
          ∀N∈HNatInfinite. hypersum f {0..<M} ≈ hypersum f {0..<N}"
         and "M ∈ HNatInfinite" "N ∈ HNatInfinite"
  then have "hypersum f {0..<M} ≈ hypersum f {0..<N}"
    by blast
  then have sums_diff: "hypersum f {0..<M} - hypersum f {0..<N} ≈ 0"
    using approx_minus_iff by blast
  then show "hypersum f {M..<N} ≈ 0"
  proof(cases "M < N")
    case True
     then show ?thesis using  sums_diff hypersum_diff_hypnat_ivl
      by (metis approx_minus_iff approx_sym assms  hypnat_le0 less_imp_le)

  next
    case False
    then show ?thesis
      using assms by auto
  qed
next
  fix M N
  assume approx_sum: "∀M∈HNatInfinite.
               ∀N∈HNatInfinite. hypersum f {M..<N} ≈ 0"
         and infs: "M ∈ HNatInfinite" "N ∈ HNatInfinite"
   show "hypersum f {0..<M} ≈ hypersum f {0..<N}"
  proof (cases "N < M")
    case True
    then show ?thesis using approx_sum hypersum_diff_hypnat_ivl infs
assms
      by (metis approx_minus_iff less_imp_le zero_less_HNatInfinite)
  next
    case False
    then have "hypersum f {0..<N} - hypersum f {0..<M} ≈ hypersum f {M..<N}"
      using assms hypersum_diff_hypnat_ivl by fastforce
    then show ?thesis
      using approx_minus_iff approx_sum approx_sym approx_trans3 infs
by blast
  qed
qed

lemma HyperSummable_starfun_n:
```

```
  "HyperSummable (*fn* f) = (∀M∈HNatInfinite.∀N∈HNatInfinite. hypersum
(*fn* f) {M..<N} ≈ 0)"
by (metis HyperSummable_def3 InternalFuns_starfun_n)


lemma atLeastLessThanhSuc_atLeastAtMost:
  "⋀m n. {m..<hSuc n} = {m..n}"
proof (auto)
  show "⋀ m n x. ⟦m ≤ x; x < hSuc n⟧ ⟹ x ≤ n"
    by transfer simp
next
  show " ⋀m n x. ⟦m ≤ x; x ≤ n⟧ ⟹ x < hSuc n"
    by transfer simp
qed

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)"
by (metis InternalFuns_starfun_n hypersum_op_ivl_Suc)

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)"
  by (metis atLeastLessThanhSuc_atLeastAtMost hypersum_op_ivl_Suc2)

lemma HyperSummable_starfun_n_HNatInfinite:
  assumes "HyperSummable (*fn* f)"
  and "K∈HNatInfinite"
  shows "(*fn* f) K ≈ 0"
proof -
  have "∀M∈HNatInfinite. ∀N∈HNatInfinite. hypersum (*fn* f) {M..<N}
≈ 0"
    using assms(1) HyperSummable_starfun_n by blast
  moreover have "hSuc K ∈ HNatInfinite" using assms(2)
    by (metis HNatInfinite_add hSuc_eq_add_one hSuc_eq_add_one)
  ultimately have "hypersum (*fn* f) {K..<hSuc K} ≈ 0"
    using assms(2) by force
  thus ?thesis by simp
qed


lemma  HyperSummable_def': "HyperSummable F = HyperCauchy (λN. hypersum
F {0..N})"
  by (metis (no_types, lifting) HNatInfinite_add HNatInfinite_add_one_cancel

      HNatInfinite_is_Suc HyperCauchy_def HyperSummable_def
      atLeastLessThanhSuc_atLeastAtMost hSuc_eq_add_one)

lemma HyperSummable_def3':
```

```
  assumes "f ∈ InternalFuns"
  shows "HyperSummable f = (∀M∈HNatInfinite.∀N∈HNatInfinite. hypersum
f {M..N} ≈ 0)"
  by (metis HNatInfinite_add HNatInfinite_add_one_cancel HNatInfinite_is_Suc
HyperSummable_def3 assms
      atLeastLessThanhSuc_atLeastAtMost hSuc_eq_add_one)

lemma HyperSummable_def4:
  "HyperSummable (*fn* f) = (∃s. ∀N∈HNatInfinite. hypersum (*fn* f) {0..N}
≈ s)"
proof
  assume hsummable_f: "HyperSummable (*fn* f)"
  then obtain s where hsum_approx: "∀N∈HNatInfinite. hypersum (*fn*
f) {0..<N} ≈ s"
    using HyperSummable_def2 by blast
  {fix N
  assume infN: "N∈HNatInfinite"
  then have "hypersum (*fn* f) {N..<N+1} ≈ 0"
    using HNatInfinite_add HyperSummable_starfun_n hsummable_f by blast
  then have "hypersum (*fn* f) {0..N} ≈ s"
    by (metis HNatInfinite_add atLeastLessThanhSuc_atLeastAtMost hSuc_eq_add_one
hsum_approx infN)}
  then show "∃s. ∀N∈HNatInfinite. hypersum (*fn* f) {0..N} ≈ s"
    by blast
next
  assume "∃s. ∀N∈HNatInfinite. hypersum (*fn* f) {0..N} ≈ s"
  then obtain s where hsum_approx: "∀N∈HNatInfinite. hypersum (*fn*
f) {0..N} ≈ s"
    by clarify
  then have "∀N∈HNatInfinite. hypersum (*fn* f) {0..<N+1} ≈ s"
    using atLeastLessThanhSuc_atLeastAtMost hSuc_eq_add_one by auto
  then show "HyperSummable (*fn* f)"
    using HNatInfinite_add_one_cancel HNatInfinite_is_Suc HyperSummable_def2
by blast
qed

lemma HyperSummable_shift_hSuc:
  assumes "f ∈ InternalFuns"
      and "HyperSummable f"
    shows "HyperSummable (λn. f (hSuc n))"
proof (subst HyperSummable_def3)
  show "(λn. f (hSuc n)) ∈ InternalFuns"
    by (simp add: InternalFuns_o2 assms(1))
next
  show "∀M∈HNatInfinite. ∀N∈HNatInfinite. hypersum (λn. f (hSuc n))
{M..<N} ≈ 0"
    by (metis HNatInfinite_add HyperSummable_def3 assms(1,2) hSuc_eq_add_one
        hypersum_shift_bounds_hSuc_ivl)
qed
```

```
lemma InternalFuns_hypersum_starfun_n_Interval:
    "(λN. hypersum (*fn* F) {M..<N}) ∈ InternalFuns"
proof (simp add: InternalFuns_def)
  {fix X
    assume M_seq: "M = star_n X"
    {fix N
      have "hypersum (*fn* F) {M..<N} = (*fn* (λn N. sum (F n) {X n..<N}))
N"
      proof (cases N)
        case (star_n Y)
        then show ?thesis using M_seq
          by (auto simp add: starfun_n hypersum starset_n_atLeastLessThan_eq
[symmetric])
      qed}
    then have "(λN. hypersum (*fn* F) {M..<N}) =  *fn* (λn N. sum (F
n) {X n..<N})"
      using ext by blast}
    then show "∃Fa. (λN. hypersum (*fn* F) {M..<N}) = *fn* Fa"
      using star_cases by metis
qed

lemma InternalFuns_hypersum:
  assumes "f ∈ InternalFuns"
  shows "(λn. hypersum f {m..<n}) ∈ InternalFuns"
  using assms InternalFuns_hypersum_starfun_n_Interval
  by (metis InternalFuns_starfun_n_starfun)

lemma InternalFuns_hypersum_starfun_n_Interval2:
    "(λN. hypersum (*fn* F) {M..N}) ∈ InternalFuns"
proof (simp add: InternalFuns_def)
  {fix X
   assume M_seq: "M = star_n X"
    {fix N
      have "hypersum (*fn* F) {M..N} = (*fn* (λn N. sum (F n) {X n..N}))
N"
      proof (cases N)
        case (star_n Y)
        then show ?thesis using M_seq
          by (auto simp add: starset_n_atLeastAtMost [symmetric] hypersum
starfun_n)
      qed}
    then have "(λN. hypersum (*fn* F) {M..N}) =  *fn* (λn N. sum (F n)
{X n..N})"
      using ext by blast}
    then show "∃Fa. (λN. hypersum (*fn* F) {M..N}) = *fn* Fa"
      using star_cases by metis
  qed
```

```
lemma InternalFuns_hypersum2:
  "f ∈ InternalFuns ⟹ (λn. hypersum f {m..n}) ∈ InternalFuns"
  using InternalFuns_hypersum_starfun_n_Interval2
  by (metis InternalFuns_starfun_n_starfun)

lemma NSsummable_NSCauchy_setsum:
  "NSsummable f = NSCauchy (λn. sum f {0..<n})"
proof (simp add: NSCauchy_def NSsummable_NSCauchy starfunNat_sumr, safe)

  fix M N
  assume "∀M∈HNatInfinite. ∀N∈HNatInfinite. sumhr (M, N, f) ≈ 0"
         "M ∈ HNatInfinite" "N ∈ HNatInfinite"
  then show "sumhr (0, M, f) ≈ sumhr (0, N, f)"
    using approx_minus_iff approx_refl approx_sym sumhr_split_diff
    by (metis less_linear)
next
  fix M N
  assume "∀M∈HNatInfinite.
              ∀N∈HNatInfinite. sumhr (0, M, f) ≈ sumhr (0, N, f)"
         "M ∈ HNatInfinite" "N ∈ HNatInfinite"
  then show "sumhr (M, N, f) ≈ 0"
    using sumhr_approx by blast
qed


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 -
  {fix m
  have "hypersum f {0..<hypnat_of_nat m} ≈
            hypersum g {0..<hypnat_of_nat m}"
  proof(induct m)
    case 0
    then show ?case
      using assms(1) assms(2) by auto
  next
    case (Suc m)
    then show ?case using assms(1,2,5)
      by (auto intro: approx_add simp add: hSuc_eq_add_one [symmetric])
  qed}
  then show ?thesis using assms(6) Nats_hypnat_of_nat_iff by auto
qed

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 -
  have "∀ n∈Nats.  hypersum f {0..<n} ≈ hypersum g {0..<n}"
    using Hypersum_Comparison_Theorem_Nats hyperSums(1) hyperSums(2) inf_close_fg
int_f int_g by blast
  then obtain M where inf_M: "M ∈ HNatInfinite"
           and inf_close_hsum: "∀ n<M. hypersum f {0..<n} ≈ hypersum g
{0..<n}"
    using InternalFuns_Sequential_Theorem2 InternalFuns_hypersum int_f
int_g by blast
  then show ?thesis
  proof (cases "N < M")
    case True
    then show ?thesis using inf_close_hsum by blast
  next
    case False
    have "M - 1 < M"  using inf_M
      by (metis hypnat_add_one_self_less hypnat_le_add_diff_inverse2 one_le_HNatInfinite)
    then have hsum1: "hypersum f {0..<M - 1} ≈ hypersum g {0..<M - 1}"

      using inf_close_hsum by blast
    have "hypersum f {M - 1..<N} ≈ 0"
      using HNatInfinite_diff HyperSummable_def3 Nats_1 hyperSums(1) inf_M
inf_N int_f by blast
    moreover have "hypersum g {M - 1..<N} ≈ 0"
      using HNatInfinite_diff HyperSummable_def3 Nats_1 hyperSums(2) inf_M
inf_N int_g by blast
    ultimately have "hypersum f {M - 1..<N} ≈ hypersum g {M - 1..<N}"
      using approx_trans2 by blast
    moreover have "hypersum f {0..<N} ≈ hypersum f {0..<M - 1} + hypersum
f {M - 1..<N}"
      using False ‹M - 1 < M› hypersum_add_hypnat_ivl int_f by fastforce
    moreover have "hypersum g {0..<N} ≈ hypersum g {0..<M - 1} + hypersum
g {M - 1..<N}"
      using False ‹M - 1 < M› hypersum_add_hypnat_ivl int_g by fastforce
    ultimately show ?thesis using hsum1 approx_add
      by (metis (no_types, lifting) approx_monad_iff)
  qed
qed
```

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}"
  by (metis HNatInfinite_not_Nats_iff Hypersum_Comparison_Theorem_HNatInfinite

      Hypersum_Comparison_Theorem_Nats)

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}"
by (metis Hypersum_Comparison_Theorem atLeastLessThanhSuc_atLeastAtMost)

lemma HyperSummable_hypersum_approx:
  "⟦ HyperSummable f; M ∈ HNatInfinite; N ∈ HNatInfinite ⟧
   ⟹ hypersum f {0..M} ≈ hypersum f {0..N}"
by (metis HNatInfinite_upward_closed HyperSummable_def2 approx_monad_iff

      atLeastLessThan_empty_iff atLeastLessThanhSuc_atLeastAtMost
      atLeastatMost_empty_iff less_imp_le order_refl)

lemma hnorm_hypersum:
  "⟦ f ∈ InternalFuns; A ∈ InternalSets ⟧ ⟹ hnorm (hypersum f A) ≤
hypersum (λi. hnorm (f i)) A"
by (auto simp add: InternalSets_def InternalFuns_def hypersum hnorm_def
starfun_starfun_n_o starfun
       star_n_eq_iff star_le_def starP2_star_n norm_sum)

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 -
  obtain F Fa As
    where fnf: "f = *fn* F" and fng: "g = *fn* Fa" and snk: "K = *sn*
As"
    and "∀F x in 𝒰. ∀y. y ∈ As x ⟶ F x y ≤ Fa x y"
    using internal_f internal_g internal_setK fg_mono
    by (auto dest!: FreeUltrafilterNat.eventually_all_iff [THEN iffD2]

          simp add: InternalSets_def InternalFuns_def  star_le_def starP2_star_n
                    all_star_eq starfun_n starset_n_star_n
                    FreeUltrafilterNat.eventually_imp_iff [symmetric])
  then have "∀F n in 𝒰. sum (F n) (As n) ≤ sum (Fa n) (As n)"
```

```
    by (simp add: eventually_mono sum_mono)
  then show ?thesis
    by (simp add: fnf fng hypersum snk star_n_le)
qed

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"
by (metis hypersum_mono')


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 -
  obtain F S where "f = *fn* F" and "A = *sn* S"
    using assms
    by (metis InternalFuns_starfun_n_starfun InternalSets_starset_n_starset)

  then have " ∀_F x in 𝒰. ∀y. y ∈ S x ⟶ 0 ≤ F x y"
    using assms(3)
    by (force dest!:  FreeUltrafilterNat.eventually_all_iff [THEN iffD2]
              simp add: star_n_zero_num star_n_le_def starP2_star_n
                        all_star_eq Ball_def starset_n_star_n
                        FreeUltrafilterNat.eventually_imp_iff [symmetric]
starfun_n)
  then have "∀_F n in 𝒰. 0 ≤ sum (F n) (S n)"
    by (force intro: eventually_mono simp add: sum_nonneg)
  then show ?thesis
    by (simp add: <A = *sn* S> <f = *fn* F> hypersum star_n_le star_n_zero_num)

qed

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"
by (auto simp add: InternalSets_def InternalFuns_def hypersum star_abs_def

           starfun_starfun_n_o starfun star_n_eq_iff)

lemma InternalFuns_hnorm_starfun_n [simp]:
  "(λi. hnorm ((*fn* F) i)) ∈ InternalFuns"
proof (auto simp add: hnorm_def)
  show "(λi. (*f* norm) ((*fn* F) i)) ∈ InternalFuns"
    by (metis InternalFuns_starfun_n starfun_starfun_n_o)
```

qed

lemma *InternalFuns_hnorm:*
"f ∈ InternalFuns ⟹ (λn. hnorm (f n)) ∈ InternalFuns"
by *(metis InternalFuns_hnorm_starfun_n InternalFuns_starfun_n_starfun)*

lemma *HyperSummable_Cauchy:*
*"HyperSummable (*fn* F) ⟹ (∀e∈Reals. e>0 ⟶ (∃N. ∀m≥N. ∀n. hnorm (hypersum (*fn* F) {m..<n}) < e))"*
unfolding *HyperSummable_starfun_n mem_infmal_iff [symmetric]*
by *(metis HNatInfinite_upward_closed HNatInfinite_whn InfinitesimalD atLeastLessThan_empty nle_le)*

lemma *Cauchy_HyperSummable:*
*"(∀e∈Reals. e>0 ⟶ (∃N∈Nats. ∀m≥N. ∀n. hnorm (hypersum (*fn* F) {m..<n}) < e)) ⟹ HyperSummable (*fn* F)"*
by *(metis HyperSummable_starfun_n InfinitesimalI Nats_le_HNatInfinite mem_infmal_iff)*

lemma *Cauchy_HyperSummable':*
*"(∀e∈Reals. e>0 ⟶ (∃N∈Nats. ∀m≥N. ∀n≥N. hnorm (hypersum (*fn* F) {m..<n}) < e)) ⟹*
*HyperSummable (*fn* F)"*
by *(metis HyperSummable_starfun_n InfinitesimalI Nats_le_HNatInfinite mem_infmal_iff)*

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 *(safe)*
fix *e::hypreal* **assume** *e: "0 < e" "e ∈ ℝ"*
have *"∃N. ∀m≥N. ∀n≥N. hnorm (hypersum (*fn* F) {m..<n}) < e"*
proof *(intro exI allI impI)*
fix *M* **assume** *"whn ≤ M"*
with *HNatInfinite_whn* **have** *M: "M ∈ HNatInfinite"*
by *(rule HNatInfinite_upward_closed)*
fix *N* **assume** *"whn ≤ N"*
with *HNatInfinite_whn* **have** *N: "N ∈ HNatInfinite"*
by *(rule HNatInfinite_upward_closed)*
from *H M N* **have** *"hypersum (*fn* F) {M..<N} ≈ 0"*
using *HyperSummable_starfun_n* **by** *blast*
then show *"hnorm (hypersum (*fn* F) {M..<N}) < e"*
by *(simp add: InfinitesimalD e(1) e(2) mem_infmal_iff)*
qed
then show
*"(∃N. ∀m≥N. ∀n≥N. hnorm (hypersum (*fn* F) {m..<n}) < e)"* **by** *blast*
qed

```
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 -
  obtain F G where Int_f: "f = *fn* F" and Int_g: "g = *fn* G"
    by (metis InternalFuns_starfun_n_starfun assms(1) assms(2))
  {fix M N
  assume inf_M: "M∈HNatInfinite" and inf_N: "N∈HNatInfinite"
  then have Infinitesimal_hSum_g: "hypersum (*fn* G) {M..<N} ∈ Infinitesimal"
    using HyperSummable_def3 Int_g assms(2) assms(4) mem_infmal_iff by
blast
  have hSum_G_ge_0: "0 ≤ hypersum (*fn* G) {M..<N}"
  proof -
    {fix x
     assume "M ≤ x" and "x < N"
     then have "(*fn* G) x ≥ (0::real star)"
       using assms(3) Int_g inf_M by (metis  Nats_le_HNatInfinite hnorm_ge_zero
order_trans)
    }
    then show "0 ≤ hypersum (*fn* G) {M..<N}"
      using assms hypersum_nonneg[intro!] by force
  qed
  have "hnorm (hypersum (*fn* F) {M..<N}) ≤ hnorm (hypersum (*fn* G)
{M..<N})"
  proof -
    {fix i
     assume "M ≤ i" and "i < N"
     then have "hnorm ((*fn* F) i) ≤ (*fn* G) i"
       using Int_f Int_g Nats_le_HNatInfinite assms(3) dual_order.trans
inf_M by blast
    }
    then have X: "hypersum (λi. hnorm ((*fn* F) i)) {M..<N} ≤  hypersum
(*fn* G) {M..<N}"
       by (force intro!: hypersum_mono)
     then show ?thesis
       using hSum_G_ge_0 by (force intro: order_trans [OF hnorm_hypersum])
   qed
   then have "hypersum (*fn* F) {M..<N} ∈ Infinitesimal"
      using  Infinitesimal_hSum_g by (fastforce intro: Infinitesimal_interval2a
[of _ 0])
  }
  then show ?thesis
    using assms(1) Int_f
    by (simp add:  InternalFuns_def HyperSummable_starfun_n mem_infmal_iff
[symmetric] )
qed
```

```
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"
by (metis HyperSummable_comparison_test abs_of_nonneg hypreal_hnorm_def)

lemma HyperSummable_comparison_test_norm:
     "f ∈ InternalFuns
      ⟹ HyperSummable (λn. hnorm ((f::hypnat => 'a::banach star) n))
⟹  HyperSummable f"
by (auto intro: HyperSummable_comparison_test InternalFuns_hnorm Nats_0)

lemma hypersum_atLeastLessThan_starfun_n:
  "hypersum (*fn* f) {M..<N} =  (*fn2* (λi m n. sum (f i) {m..<n})) M
N"
proof (cases M, cases N)
  fix X Xa
  assume "M = star_n X" "N = star_n Xa"
  then show ?thesis
    by (auto simp add: hypersum starfun2_n starfun_n
        starset_n_atLeastLessThan_eq [symmetric])
qed

lemma hypersum_minus:
  "⟦ f ∈ InternalFuns; A ∈ InternalSets ⟧
   ⟹ hypersum (λx. - (f x)::'a::ab_group_add star) A = - hypersum f
A"
by (auto simp add: InternalFuns_def InternalSets_def starfun_n_minus hypersum

       star_n_minus star_n_eq_iff sum_negf)

lemma HyperSummable_minus:
  "⟦ f ∈ InternalFuns; HyperSummable f ⟧ ⟹  HyperSummable (λn. - f n)"
by (auto simp add: HyperSummable_def HyperCauchy_def hypersum_minus)

lemma HyperSummable_negf_iff:
  "f ∈ InternalFuns ⟹ (HyperSummable (λn. - f n)) =  (HyperSummable
f)"
by (auto simp add: HyperSummable_def HyperCauchy_def hypersum_minus)

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 (cases r)
  case (star_n X)
```

```
  then show ?thesis
    using assms
    by (auto simp add: InternalFuns_def InternalSets_def hypersum  star_n_mult

       starfun_n_mult_star_n_right star_n_eq_iff sum_distrib_right)
qed

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 (cases r)
  case (star_n X)
  then show ?thesis
    using assms
    by (auto simp add: InternalFuns_def InternalSets_def hypersum  star_n_mult

       starfun_n_mult_star_n_left star_n_eq_iff sum_distrib_left)
qed

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"
  by  (auto simp add: InternalFuns_def InternalSets_def hypersum star_n_mult

        sum_product starfun_n_mult hypersum_right_distrib [symmetric]

        starfun_n_mult_star_n_right star_n_eq_iff sum_distrib_right)

lemma starfun2_to_starfun_n_lemma:
  "(λn. (*f2* f) (star_n X) ((*fn* F) n)) = *fn* (λn m. f (X n) (F n m))"
by (rule ext, case_tac n, auto simp add: starfun_n starfun2_star_n)

lemma starfun2_to_starfun_n_lemma2:
  "(λn. (*f2* f) ((*fn* F) n) (star_n X)) = *fn* (λn m. f (F n m) (X n))"
by (rule ext, case_tac n, auto simp add: starfun_n starfun2_star_n)

lemma hypersum_divide_distrib:
  assumes "f ∈ InternalFuns"
          "A ∈ InternalSets "
  shows "hypersum f A / (r::'a::field star) = hypersum (λn. f n / r)
A"
proof (cases r)
  case (star_n X)
  then show ?thesis using assms
    by (auto simp add: InternalFuns_def InternalSets_def hypersum star_divide_def
```

```
      starfun2_star_n starfun2_to_starfun_n_lemma2 star_n_eq_iff sum_divide_distrib)
qed

lemma HyperSummable_HFinite_mult_left:
  "⟦ f ∈ InternalFuns; (c::'a::{real_normed_algebra} star) ∈ HFinite;
HyperSummable f ⟧
   ⟹ HyperSummable (λn. c * f n)"
by (auto intro: approx_mult2 simp add: HyperSummable_def HyperCauchy_def

            hypersum_right_distrib [symmetric])

lemma HyperSummable_divide:
  assumes "f ∈ InternalFuns"
          "HyperSummable f"
          "r ∉ Infinitesimal"
  shows "HyperSummable (λn. f n / (r::'a::{real_normed_field} star))"
proof -
  have "1/r ∈ HFinite"
    by (metis Infinitesimal_inverse_HFinite assms(3) inverse_eq_divide)
  then have "HyperSummable (λn. 1/r * f n)"
    using HyperSummable_HFinite_mult_left assms(1) assms(2) by blast
  then show ?thesis by simp
qed


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))"
  by (simp add: hfloor_def starfun_star_n)

lemma hfloor_correct: "⋀x. of_hypint (hfloor x) ≤ x ∧ x < of_hypint
(hfloor x + 1)"
by transfer (rule floor_correct)
```

lemma *hfloor_unique: "⋀x z. ⟦ of_hypint z ≤ x; x < of_hypint z + 1 ⟧ ⟹ hfloor x = z"*
by *transfer (blast intro: floor_unique)*

lemma *hfloor_eq_zero [simp]: "⟦ 0 ≤ x; x < 1⟧ ⟹ hfloor x = 0"*
by *(simp add: hfloor_unique)*

lemma *of_hypint_floor_le: "of_hypint (hfloor x) ≤ x"*
using *hfloor_correct* **..**

lemma *le_hfloor_iff: "⋀x z. z ≤ hfloor x ⟷ of_hypint z ≤ x"*
by *transfer (rule le_floor_iff)*

lemma *hfloor_less_iff: "⋀x z. hfloor x < z ⟷ x < of_hypint z"*
by *transfer (rule floor_less_iff)*

lemma *less_hfloor_iff: "⋀x z. z < hfloor x ⟷ of_hypint z + 1 ≤ x"*
by *transfer (rule less_floor_iff)*

lemma *hfloor_le_iff: "⋀x z. hfloor x ≤ z ⟷ x < of_hypint z + 1"*
by *transfer (rule floor_le_iff)*

lemma *hfloor_mono: "⋀x y. x ≤ y ⟹ hfloor x ≤ hfloor y"*
by *transfer (rule floor_mono)*

lemma *hfloor_less_cancel: "⋀x y. hfloor x < hfloor y ⟹ x < y"*
by *transfer (rule floor_less_cancel)*

lemma *hfloor_of_hypint [simp]: "⋀z. hfloor (of_hypint z) = z"*
by *transfer (rule floor_of_int)*

lemma *hfloor_of_int [simp]: "hfloor (of_int z) = hypint_of_int z"*
by *(rule hfloor_unique) auto*

lemma *hfloor_of_hypnat [simp]: "⋀n. hfloor (of_hypnat n) = of_hypnat n"*
by *transfer (rule floor_of_nat)*

lemma *hfloor_of_nat [simp]: "hfloor (of_nat n) = of_nat n"*
using *hfloor_of_hypint [of "of_nat n"]* **by** *simp*

Floor with numerals

lemma *hfloor_zero [simp]: "hfloor 0 = 0"*
using *hfloor_of_int [of 0]* **by** *simp*

lemma *hfloor_one [simp]: "hfloor 1 = 1"*
using *hfloor_of_int [of 1]* **by** *simp*

lemma *hfloor_star_of [simp]: "hfloor (star_of x) = star_of x"*
by *transfer simp*

lemma *star_of_hfloor [simp]: "star_of(floor x) = hfloor (star_of x)"*
by *(metis hfloor_def starfun_star_of)*

lemma *hfloor_number_of [simp]: "hfloor (numeral v :: hypreal) = (numeral v :: hypint)"*
by *(metis hfloor_of_int of_int_numeral star_numeral_def)*

lemma *zero_le_hfloor [simp]: "0 ≤ hfloor x ⟷ 0 ≤ x"*
by *(simp add: le_hfloor_iff)*

lemma *one_le_hfloor [simp]: "1 ≤ hfloor x ⟷ 1 ≤ x"*
by *(simp add: le_hfloor_iff)*

lemma *numeral_le_hfloor [simp]: "⋀x. numeral v ≤ hfloor x ⟷ numeral v ≤ x"*
by *transfer simp*

lemma *zero_less_hfloor [simp]: "0 < hfloor x ⟷ 1 ≤ x"*
by *(simp add: less_hfloor_iff)*

lemma *one_less_hfloor [simp]: "1 < hfloor x ⟷ 2 ≤ x"*
by *(simp add: less_hfloor_iff)*

lemma *numeral_less_hfloor [simp]:*
"⋀x. numeral v < hfloor x ⟷ numeral v + 1 ≤ x"
by *transfer simp*

lemma *hfloor_le_zero [simp]: "hfloor x ≤ 0 ⟷ x < 1"*
by *(simp add: hfloor_le_iff)*

lemma *hfloor_le_one [simp]: "hfloor x ≤ 1 ⟷ x < 2"*
by *(simp add: hfloor_le_iff)*

lemma *hfloor_le_numeral [simp]:*
"⋀x. hfloor x ≤ numeral v ⟷ x < numeral v + 1"
by *transfer simp*

lemma *hfloor_less_zero [simp]: "hfloor x < 0 ⟷ x < 0"*
by *(simp add: hfloor_less_iff)*

lemma *hfloor_less_one [simp]: "hfloor x < 1 ⟷ x < 1"*
by *(simp add: hfloor_less_iff)*

lemma *hfloor_less_numeral [simp]:*
"⋀x. hfloor x < numeral v ⟷ x < numeral v"
by *transfer simp*

lemma *hfloor_add_of_hypint [simp]: "hfloor (x + of_hypint z) = hfloor x + z"*
using *hfloor_correct [of x]* **by** *(simp add: hfloor_unique)*

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

lemma *hfloor_add_numeral [simp]:*
"hfloor (x + numeral v) = hfloor x + numeral v"
using *hfloor_add_of_int [of x "numeral v"]* **by** *simp*

lemma *hfloor_add_one [simp]: "hfloor (x + 1) = hfloor x + 1"*
using *hfloor_add_of_int [of x 1]* **by** *simp*

lemma *hfloor_diff_of_hypint [simp]: "hfloor (x - of_hypint z) = hfloor x - z"*
using *hfloor_add_of_hypint [of x "- z"]* **by** *(simp add: algebra_simps)*

lemma *hfloor_diff_of_int [simp]: "hfloor (x - of_int z) = hfloor x - of_int z"*
using *hfloor_add_of_int [of x "- z"]* **by** *(simp add: algebra_simps)*

lemma *hfloor_diff_numeral [simp]:*
"hfloor (x - numeral v) = hfloor x - numeral v"
using *hfloor_diff_of_int [of x "numeral v"]* **by** *simp*

lemma *hfloor_diff_one [simp]: "hfloor (x - 1) = hfloor x - 1"*
using *hfloor_diff_of_int [of x 1]* **by** *simp*

lemma *hfloor_add_one_gt_zero:*
"⋀y. 0 < y ⟹ 0 < hfloor y + 1"
by *(metis hypint_le_add1_eq_le le_less zero_le_hfloor)*

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)"*
unfolding *hceiling_def* **using** *hfloor_correct [of "- x"]* **by** *simp*

lemma *hceiling_unique: "⟦of_hypint z - 1 < x; x ≤ of_hypint z⟧ ⟹ hceiling x = z"*
unfolding *hceiling_def* **using** *hfloor_unique [of "- z" "- x"]* **by** *simp*

lemma *le_of_int_hceiling: "x ≤ of_hypint (hceiling x)"*
using *hceiling_correct* **..**

lemma *hceiling_le_iff: "hceiling x ≤ z ⟷ x ≤ of_hypint z"*
unfolding *hceiling_def* **using** *le_hfloor_iff [of "- z" "- x"]* **by** *auto*

lemma *less_hceiling_iff: "z < hceiling x ⟷ of_hypint z < x"*
by *(simp add: not_le [symmetric] hceiling_le_iff)*

lemma *hceiling_less_iff: "hceiling x < z ⟷ x ≤ of_hypint z - 1"*
using *hceiling_le_iff [of x "z - 1"]* **by** *simp*

lemma *le_hceiling_iff: "z ≤ hceiling x ⟷ of_hypint z - 1 < x"*
by *(simp add: not_less [symmetric] hceiling_less_iff)*

lemma *hceiling_mono: "x ≥ y ⟹ hceiling x ≥ hceiling y"*
unfolding *hceiling_def* **by** *(simp add: hfloor_mono)*

lemma *hceiling_less_cancel: "hceiling x < hceiling y ⟹ x < y"*
by *(auto simp add: not_le [symmetric] hceiling_mono)*

lemma *hceiling_of_hypint [simp]: "hceiling (of_hypint z) = z"*
by *(rule hceiling_unique) simp_all*

lemma *hceiling_of_int [simp]: "hceiling (of_int z) = of_int z"*
by *(rule hceiling_unique) simp_all*

lemma *hceiling_of_hypnat [simp]: "hceiling (of_hypnat n) = of_hypnat n"*
using *hceiling_of_hypint [of "of_hypnat n"]* **by** *simp*

lemma *hceiling_of_nat [simp]: "hceiling (of_nat n) = of_nat n"*
using *hceiling_of_int [of "of_nat n"]* **by** *simp*

Ceiling with numerals

lemma *hceiling_zero [simp]: "hceiling 0 = 0"*
using *hceiling_of_int [of 0]* **by** *simp*

lemma *hceiling_one [simp]: "hceiling 1 = 1"*
using *hceiling_of_int [of 1]* **by** *simp*

lemma *hceiling_numeral [simp]: "hceiling (numeral v) = numeral v"*
using *hceiling_of_int [of "numeral v"]* **by** *simp*

lemma *hceiling_le_zero [simp]: "hceiling x ≤ 0 ⟷ x ≤ 0"*
by *(simp add: hceiling_le_iff)*

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

by *(simp add: hceiling_le_iff)*

lemma *hceiling_le_numeral [simp]:*
"⋀x. hceiling x ≤ numeral v ⟷ x ≤ numeral v"
by *transfer (simp add: floor_minus)*

lemma *hceiling_less_zero [simp]: "hceiling x < 0 ⟷ x ≤ -1"*
by *(simp add: hceiling_less_iff)*

lemma *hceiling_less_one [simp]: "hceiling x < 1 ⟷ x ≤ 0"*
by *(simp add: hceiling_less_iff)*

lemma *hceiling_less_numeral [simp]:*
"⋀x. hceiling x < numeral v ⟷ x ≤ numeral v - 1"
by *transfer(metis ceiling_def ceiling_less_numeral)*

lemma *zero_le_hceiling [simp]: "0 ≤ hceiling x ⟷ -1 < x"*
by *(simp add: le_hceiling_iff)*

lemma *one_le_hceiling [simp]: "1 ≤ hceiling x ⟷ 0 < x"*
by *(simp add: le_hceiling_iff)*

lemma *numeral_le_hceiling [simp]:*
"⋀x. numeral v ≤ hceiling x⟷ numeral v - 1 < x"
by *(meson hceiling_less_numeral leD not_le_imp_less)*

lemma *zero_less_hceiling [simp]: "0 < hceiling x ⟷ 0 < x"*
by *(simp add: less_hceiling_iff)*

lemma *one_less_hceiling [simp]: "1 < hceiling x ⟷ 1 < x"*
by *(simp add: less_hceiling_iff)*

lemma *numeral_less_hceiling [simp]:*
"⋀x. numeral v < hceiling x ⟷ numeral v < x"
by *(metis hceiling_le_numeral not_less)*

lemma *hceiling_add_of_hypint [simp]: "hceiling (x + of_hypint z) = hceiling x + z"*
using *hceiling_correct [of x]* **by** *(simp add: hceiling_unique)*

lemma *hceiling_add_of_int [simp]: "hceiling (x + of_int z) = hceiling x + of_int z"*
using *hceiling_correct [of x]* **by** *(simp add: hceiling_unique)*

lemma *hceiling_add_numeral [simp]:*
"hceiling (x + numeral v) = hceiling x + numeral v"
using *hceiling_add_of_int [of x "numeral v"]* **by** *simp*

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

using *hceiling_add_of_int [of x 1]* **by** *simp*

lemma *hceiling_diff_of_hypint [simp]: "hceiling (x - of_hypint z) = hceiling x - z"*
using *hceiling_add_of_hypint [of x "- z"]* **by** *(simp add: algebra_simps)*

lemma *hceiling_diff_of_int [simp]: "hceiling (x - of_int z) = hceiling x - of_int z"*
using *hceiling_add_of_int [of x "- z"]* **by** *(simp add: algebra_simps)*

lemma *hceiling_diff_numeral [simp]:*
"hceiling (x - numeral v) = hceiling x - numeral v"
using *hceiling_diff_of_int [of x "numeral v"]* **by** *simp*

lemma *hceiling_diff_one [simp]: "hceiling (x - 1) = hceiling x - 1"*
using *hceiling_diff_of_int [of x 1]* **by** *simp*

lemma *hfloor_minus: "hfloor (- x) = - hceiling x"*
unfolding *hceiling_def* **by** *simp*

lemma *hceiling_minus: "hceiling (- x) = - hfloor x"*
unfolding *hceiling_def* **by** *simp*

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"*
by *(simp add: hpart_def)*

lemma *hpart_ge_zero [simp]: "0 ≤ hpart x"*
using *hfloor_correct [of x]* **by** *(simp add: hpart_def)*

lemma *hpart_less_one [simp]: "hpart x < 1"*
using *hfloor_correct [of x]* **by** *(simp add: hpart_def)*

lemma *hpart_le_one [simp]: "hpart x ≤ 1"*
by *(simp add: less_imp_le)*

```
lemma hypreal_eq_hfloor_hpart_add:
      "x = of_hypint (hfloor x) + hpart x"
by (simp add: hpart_def)

lemma hfloor_eq_self_diff_hpart: "of_hypint (hfloor x) = x - hpart x"
by (simp add: hpart_def)

lemma HFinite_hpart [simp]: "⋀x. hpart (x::hypreal) ∈ HFinite"
  using HFinite_1 HFinite_bounded hpart_ge_zero hpart_le_one by blast

lemma hpart_not_less_one [simp]: "¬ 1 < ⦃x⦄"
by (simp add: not_less)

lemma hpart_mult_less_cancel [simp]: "(x * ⦃y⦄ < x) = (0 < x)"
by (auto simp add: mult_less_cancel_left2)

lemma hfloor_of_hypint_divide [simp]:
  "⋀x y. hfloor ((of_hypint x :: hypreal)/ of_hypint y) = (x div y)"
by transfer (simp add: floor_divide_of_int_eq)

lemma hfloor_of_hypint_of_hypnat_divide [simp]:
  "⋀x y. hfloor ((of_hypint x :: hypreal)/ of_hypnat y) = (x div (of_hypnat
y))"
by (metis hfloor_of_hypint_divide of_hypint_of_hypnat)

lemma floor_real_of_nat_real_of_int_divide [simp]:
  "floor (real x / real y) = int x div y"
by (metis floor_divide_of_int_eq of_int_of_nat_eq)

lemma hfloor_of_hypnat_of_hypint_divide [simp]:
  "⋀x y. hfloor ((of_hypnat x:: hypreal) / of_hypint y) = (of_hypnat
x) div y"
by (metis hfloor_of_hypint_divide of_hypint_of_hypnat)

lemma of_hypnat_hypnat_of_hypint [simp]:
  "0 ≤ x ⟹ of_hypnat(hypnat(hfloor x)) = of_hypint (hfloor x)"
  by simp
```

7.4 Miscellaneous properties

```
lemma HFinite_hypnat_hfloor_Nat:
  assumes "(y::hypreal) ∈ HFinite"
  shows "hypnat (hfloor y) ∈ ℕ"
proof -
  have ybounds: "hypreal_of_hypint (hfloor y) ≤ y" "y < hypreal_of_hypint
(hfloor y + 1)"
    using hfloor_correct by blast+
  then show ?thesis
  proof (cases "y > 0")
```

```
    case True
    assume y0: "y > 0"
    show ?thesis
    proof (rule ccontr)
      assume "hypnat (hfloor y) ∉ ℕ"
      then have "hypnat (hfloor y) ∈ HNatInfinite"
        using HNatInfinite_not_Nats_iff by blast
      moreover have "hypreal_of_hypnat (hypnat (hfloor y)) ≤ y"
        by (metis hypint_hypnat_eq less_imp_le of_hypint_of_hypnat y0
ybounds(1) zero_le_hfloor)
      ultimately show False
        using HInfinite_HFinite_iff HInfinite_ge_HNatInfinite assms by
blast
    qed
  next
    case False
    then show ?thesis by simp
  qed
qed

lemma HFinite_between_Nats:
  assumes "(x::hypreal) ∈ HFinite"
  and "x > 0"
  shows "∃n∈ℕ. n ≤ x ∧ x < n + 1"
proof -
  have x1: "hypreal_of_hypint (hfloor x) ≤ x" and x2: "x < hypreal_of_hypint
(hfloor x + 1)"
    using hfloor_correct by auto
  moreover obtain m where "hfloor x = hypint_of_hypnat m"
    using assms(2) nonneg_eq_hypint zero_le_hfloor less_imp_le by metis
  moreover have "m ∈ ℕ"
    using HFinite_hypnat_hfloor_Nat assms(1) calculation(3) by force
  ultimately show ?thesis
    by (auto simp add: Nats_def)
qed

lemma hfloor_eq_self_diff_hpart2:
      "⋀x. 0 ≤ x ⟹ of_hypnat (hypnat (hfloor x)) = x - hpart x"
  by (metis hfloor_eq_self_diff_hpart hypint_hypnat_eq of_hypint_of_hypnat
zero_le_hfloor)

lemma hypreal_HInfinite_hfloor:
  "(x::hypreal) ∈ HInfinite ⟹ hypreal_of_hypint (hfloor x) ∈ HInfinite"
  by (metis HFinite_hpart HInfinite_HFinite_add_cancel diff_add_cancel
hfloor_eq_self_diff_hpart)

lemma hypreal_HInfinite_hfloor_cancel:
  "hypreal_of_hypint (hfloor x) ∈ HInfinite ⟹ (x::hypreal) ∈ HInfinite"
by (metis HFinite_hpart HInfinite_HFinite_add diff_add_cancel hfloor_eq_self_diff_hpart)
```

```
lemma hypreal_HNatInfinite_hfloor:
  "⟦ (x::hypreal) ∈ HInfinite; x > 0 ⟧ ⟹ hypnat (hfloor x) ∈ HNatInfinite"
by (metis HInfinite_gt_zero_gt_one hypreal_HInfinite_hfloor
    HInfinite_of_hypint_HNatInfinite_hypnat less_imp_le zero_less_hfloor)

lemma hypreal_HNatInfinite_hfloor_cancel:
  "hypnat (hfloor x) ∈ HNatInfinite ⟹ (x::hypreal) ∈ HInfinite"
by (metis hypreal_HInfinite_hfloor_cancel HNatInfinite_hypnat_HInfinite_of_hypint)

lemma hypreal_HNatInfinite_hfloor_inverse_Infinitesimal:
   "⟦ (e::hypreal) ∈ Infinitesimal; e > 0 ⟧ ⟹ hypnat (hfloor (1 / e))
∈ HNatInfinite"
by  (auto intro!: hypreal_HNatInfinite_hfloor Infinitesimal_inverse_HInfinite

        simp add: divide_inverse)

lemma hypreal_Infinitesimal_HNatInfinite_hfloor_inverse_iff:
      assumes pos: "e > 0"
      shows "((e::hypreal) ∈ Infinitesimal) = (hypnat (hfloor (1 / e))
∈ HNatInfinite)" (is "?L = ?R")
proof
  assume ?L
  then show ?R using pos hypreal_HNatInfinite_hfloor_inverse_Infinitesimal
by blast
next
  assume ?R then have "1/e ∈ HInfinite" using hypreal_HNatInfinite_hfloor_cancel
by blast
  then show ?L using HInfinite_inverse_Infinitesimal by fastforce
qed

lemma hypreal_Infinitesimal_hfloor_inverse_HNatInfinite:
   "hypnat (hfloor (1 / e)) ∈ HNatInfinite ⟹ (e::hypreal) ∈ Infinitesimal"
  using HInfinite_inverse_Infinitesimal hypreal_HNatInfinite_hfloor_cancel
by force

lemma inverse_hfloor_inverse_approx:
  "e ∈ Infinitesimal ⟹ inverse (hypreal_of_hypint(hfloor(1 / e))) ≈
e"
proof -
  assume a1: "e ∈ Infinitesimal"
  then have f2: "e = 0 ∨ 1 / e ∈ HInfinite"
    by (metis (no_types) Infinitesimal_inverse_HInfinite inverse_eq_divide)
  then have "1 / hypreal_of_hypint (hfloor (1 / e)) ≈ e"
    by (metis HInfinite_inverse_Infinitesimal Infinitesimal_approx a1
div_by_0 hfloor_zero
        hypreal_HInfinite_hfloor inverse_eq_divide of_hypint_0)
  then show ?thesis
    by (simp add: inverse_eq_divide)
```

qed

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 -
have *infmal_hpart_mult: "x * hpart (z/ x) ≈ 0"*
using *infmal_x*
by *(simp add: Infinitesimal_HFinite_mult Infinitesimal_approx)*
have *"x * of_hypint(hfloor (z/x)) = x * (z/x - hpart (z/x))"*
using *hfloor_eq_self_diff_hpart*
by *auto*
also have *"... = z - x * hpart (z/x)"*
by *(simp add: not_zero_x right_diff_distrib)*
finally show *?thesis*
using *approx_diff infmal_hpart_mult*
by *force*
qed

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"*
by *(simp add: Infinitesimal_hfloor_divide_mult infmal_x inverse_eq_divide not_zero_x)*

lemma *Infinitesimal_hfloor_inverse_mult_self_pos:*
*"⟦ e ∈ Infinitesimal; e > (0::hypreal) ⟧ ⟹ e * of_hypint(hfloor(1/e)) ≈ 1"*
by *(auto dest: Infinitesimal_hfloor_inverse_mult_self simp add: inverse_eq_divide)*

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)"*
by *(metis hyperbinomial_def starfun2_star_n)*

lemma *star_of_choose [simp]: "star_of (n choose k) = (star_of n) hchoose (star_of k)"*
by *transfer simp*

lemma *star_choose_zero_hypnat [simp]: "⋀n. (n::hypnat) hchoose 0 = 1"*
by *transfer simp*

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

lemma *star_choose_reduce_hypnat:*
"⋀n k. ⟦0 < (n::hypnat); 0 < k⟧
⟹ n hchoose k = n - 1 hchoose k + (n - 1 hchoose (k - 1))"
by *transfer (simp add: choose_reduce_nat)*

lemma *star_choose_plus_one_hypnat: "⋀n k. ((n::hypnat) + 1) hchoose (k + 1) =*
(n hchoose (k + 1)) + (n hchoose k)"
by *transfer simp*

lemma *star_choose_hSuc_hypnat: "⋀n k. (hSuc n) hchoose (hSuc k) =*
(n hchoose (hSuc k)) + (n hchoose k)"
by *transfer simp*

lemma *star_choose_self_hypnat [simp]: "⋀n. ((n::hypnat) hchoose n) = 1"*
by *transfer simp*

lemma *star_choose_one_hypnat [simp]: "⋀n. (n::hypnat) hchoose 1 = n"*
by *transfer simp*

lemma *star_plus_one_choose_self_hypnat [simp]: "⋀n. (n::hypnat) + 1 hchoose n = n + 1"*
by *transfer simp*

lemma *hSuc_choose_self_hypnat [simp]: "⋀n. (hSuc n) hchoose n = hSuc n"*
by *transfer simp*

lemma *choose_pos_nat [rule_format]: "⋀k n. k <= (n::hypnat) ⟹ (n hchoose k) > 0"*
by *transfer simp*

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))"*
by *transfer (blast intro: binomial_fact')*

lemma *star_choose_dvd_hypnat: "⋀n k. (k::hypnat) ≤ n ⟹ hfact k * hfact (n - k) dvd hfact n"*
by *transfer (metis binomial_fact_lemma dvdI of_nat_fact of_nat_mult)*

lemma *hfact_of_nat: "hfact (hypnat_of_nat n) = of_nat (fact n)"*
by *(simp add: hfact_def star_of_nat_def)*

lemma *HFinite_hfact_of_nat [simp]:*
"of_hypnat (hfact (hypnat_of_nat n)) ∈ HFinite"
by *(simp add: hfact_of_nat of_hypnat_def)*

lemma *hfact_nat_in_Nats: "n ∈ ℕ ⟹ hfact (n::hypnat)∈ ℕ "*
by *(metis Nats_hypnat_of_nat_iff fact_in_Nats hfact_of_nat of_nat_fact)*

lemma *star_choose_le_hyperpow:*
"⋀r n. r ≤ n ⟹ n hchoose r ≤ n pow r"
by *transfer (rule binomial_le_pow)*

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}"
using *starset_n_atLeast_zero_AtMost starset_n_def* **by** *blast*

lemma *n_star_interval_ultra: "eventually (λn. (*ns* {0..star_n X}) n = {0..X n}) 𝒰"*
by *(simp add: Iset_interval [symmetric] n_starset_eq_ultra)*

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
fix *K*
show *"(*fn* (λn k. of_nat (Xb n choose k) * X n ^ k * Xa n ^ (Xb n - k))) K =*
*of_hypnat ((*f2* (choose)) (star_n Xb) K) * (*f2* (^)) (star_n X) K **

```
        (*f2* (^)) (star_n Xa) (star_n Xb - K)"
  proof (cases K)
    case (star_n X)
    then show ?thesis
      by (simp add: starfun2_star_n star_n_diff of_hypnat_def starfun_star_n
star_n_mult starfun_n)
  qed
qed
```

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 -
  obtain As Bs Ns where abN: "a = star_n As" "b = star_n Bs" "N = star_n
Ns"
    by (meson star_cases)
  then have star_n_sum:
         "∀f. star_n (λn. sum ((*nf* *fn* f) n) ((*ns* {0..star_n Ns})
n)) =
               star_n (λn. sum (f n) {0..Ns n})"
    by (metis (full_types) hypersum hypersum_def starset_n_atLeast_zero_AtMost)
  show ?thesis
    by (simp add: abN star_n_sum hypersum_def hyperpow_def hyperbinomial_def
atLeast0AtMost
                  lemma_starfun_n_binomial [symmetric] starfun2_star_n
star_n_add binomial_ring)
qed

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 (subst add.commute)
  show "(a + 1) pow N = hypersum (λi. of_hypnat (N hchoose i) * a pow
i) {0..N}"
    by (simp add: hyperbinomial_ring)
qed

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))"
  by (simp add: starfun_n hyperpow star_choose of_hypnat_def
       starfun star_mult_def star_diff_def starfun2_star_n)

lemma InternalFuns_hyperbinomial [simp]:
```

```
    "(λK. hypreal_of_hypnat (N hchoose K) * a pow K * b pow (N - K)) ∈
InternalFuns"
proof (simp add: InternalFuns_def)
  obtain As Bs Ns where AbN: "a = star_n As" "b = star_n Bs" "N = star_n
Ns"
    by (meson star_cases)
   show "∃F. (λK. hypreal_of_hypnat (N hchoose K) * a pow K * b pow (N
- K)) = *fn* F"
    proof (rule exI [of _ "(λn k. real (Ns n choose k) * As n ^ k * Bs
n ^ (Ns n - k))"], rule ext)
      fix K
      obtain Ks where "(K::nat star) = star_n Ks"
        using star_cases by blast
      then show "hypreal_of_hypnat (N hchoose K) * a pow K * b pow (N
- K) =
                  (*fn* (λn k. real (Ns n choose k) * As n ^ k *  Bs n
^ (Ns n - k))) K"
        by (simp add: AbN hyperbinomial_def hyperpow_def lemma_starfun_n_binomial)
    qed
  qed
```

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"
by *transfer simp*

lemma *hfallfactpow_1[simp]*: "⋀a. hfallfactpow a 1 = a" **by** *transfer simp*
lemma *hfallfactpow_hSuc0[simp]*: "⋀a. hfallfactpow a (hSuc 0) = a" **by** *transfer simp*

lemma *hfallfactpow_hSuc*: "⋀a n. hfallfactpow a (hSuc n) = hfallfactpow
a n * (a - of_hypnat n)"
by *transfer (simp add: fallfactpow_Suc)*

lemma *hfallfactpow_hSuc'*: "hfallfactpow a (n + 1) = hfallfactpow a n
* (a - of_hypnat n)"
by *(metis hSuc_eq_add_one hfallfactpow_hSuc)*

lemma *hfallfactpow_rec*: "⋀a n. hfallfactpow a (hSuc n) = a * hfallfactpow
(a - 1) n"
by *transfer (simp add: fallfactpow_rec)*

lemma *hfallfactpow_rec'*: "⋀a n. hfallfactpow a (n + 1) = a * hfallfactpow
(a - 1) n"
by *(metis hSuc_eq_add_one hfallfactpow_rec)*

```
lemma hfallfactpow_base_zero[simp]: "⋀n. hfallfactpow 0 (hSuc n) = 0"
by transfer simp

lemma hfallfactpow_base_zero'[simp]: "⋀n. hfallfactpow 0 (n + 1) = 0"
by (metis hSuc_eq_add_one hfallfactpow_base_zero)

lemma hfallfactpow_fact: "⋀n. of_hypnat(hfact n) =  hfallfactpow (of_hypnat
n) n"
by transfer (simp add: fallfactpow_fact)

lemma hfallfactpow_of_nat_eq_0_lemma:
  "⋀k n. k > n ⟹ hfallfactpow (of_hypnat n :: 'a:: idom star) k = 0"
by transfer simp

lemma hyperbinomial_hfallfactpow:
 "⋀k n. of_hypnat (n hchoose k) =
  (hfallfactpow (of_hypnat n) k ::'a::{field_char_0} star)/hfact k"
by transfer (metis binomial_fallfactpow_altdef_of_nat)

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}"
by (simp add: hyperbinomial_ring hyperbinomial_hfallfactpow)

lemma hfallfactpow_le_power_self [simp]:
  "⋀k. hfallfactpow (of_hypnat k ::'a::{linordered_idom} star) k ≤ of_hypnat
k pow k"
by transfer (metis fallfactpow_le_power_self)

lemma hfallfactpow_le_hyperpow:
  "⋀k j. of_hypnat k ≤ j ⟹ hfallfactpow (j::'a::{linordered_idom} star)
k ≤ j pow k"
by transfer (metis fallfactpow_le_power)

lemma hfallfactpow_ge_zero [simp]:
  "⋀k n. hfallfactpow (of_hypnat n) k ≥ (0::'a::linordered_idom star)"
by transfer simp


lemma fact_Suc_ge_two_pow: "fact (n + 1) ≥ (2::nat)^n"
proof (induct n)
  case 0 thus ?case
    by simp
next
  case (Suc n)
  have "fact (Suc n + 1) = (Suc n + 1) * fact (Suc n)"
    by simp
```

```
  also have "...  ≥ 2 * fact (Suc n)"
    by (simp add: mult_le_mono)
  also have "... ≥ 2 * 2^n"
    using Suc by simp
  then show ?case
    by simp
qed

lemma hfact_hSuc_ge_two_pow:
  "⋀n. (hfact (n + 1) :: hypreal) ≥ 2 pow n"
  by transfer (metis fact_Suc_ge_two_pow numeral_power_le_of_nat_cancel_iff
of_nat_fact)

lemma HInfinite_diff_of_nat_divide_approx_one:
  assumes "x ∈ HInfinite"
  shows "(x - of_nat k)/x ≈ (1::'a::real_normed_field star)"
proof (cases "x=0")
  case True
  then show ?thesis
    using assms zero_not_HInfinite by blast
next
  case False
  then have "(x - of_nat k)/x = 1 - of_nat k/x"
    by (simp add: diff_divide_eq_iff)
  moreover have "of_nat k/x ≈ 0"
    by (metis HFinite_star_of Infinitesimal_HInfinite_divide assms mem_infmal_iff
star_of_nat_def)
  ultimately show ?thesis
    using approx_diff by force
qed

lemma HInfinite_hfallfactpow_divide_hrealpow:
  assumes "j ∈ HInfinite"
  shows "hfallfactpow j (of_nat k)/(j ^ k) ≈ (1::'a::real_normed_field
star)"
proof (induct k)
  case 0
  then show ?case by simp
next
  case (Suc k)
  assume IH: "hfallfactpow j (of_nat k) / j ^ k ≈ 1"
  have "(j - of_nat k)/ j ≈ 1"
    by (simp add: HInfinite_diff_of_nat_divide_approx_one assms)
  then have "(hfallfactpow j (of_nat k) / j ^ k) * (j - of_nat k)/ j
≈ 1"
    using IH approx_mult_HFinite
    by (metis (no_types, lifting) HFinite_1 mult.left_neutral times_divide_eq_right)

  then show ?case
```

```
    by (auto simp add: hfallfactpow_hSuc' mult.commute)
qed

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)"
by (metis HInfinite_hfallfactpow_divide_hrealpow hyperpow_of_nat)

lemma HInfinite_hfallfactpow_divide_hyperpow_Nats:
"⟦ k∈ℕ; j∈HInfinite ⟧ ⟹ hfallfactpow j k/(j pow k) ≈ (1::'a::real_normed_field
star)"
by (metis HInfinite_hfallfactpow_divide_hyperpow_of_nat Nats_cases)
```

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)"
by (metis HInfinite_hfallfactpow_divide_hyperpow_of_nat HNatInfinite_of_hypnat_HInfinite)


lemma hfallfactpow_base_starfun2:
  "hfallfactpow a = (*f2* fallfactpow) a"
proof
  fix x
  show "hfallfactpow a x = (*f2* fallfactpow) a x"
    by (simp add: hfallfactpow_def)
qed

lemma InternalFuns_hfallfactpow [simp]: "hfallfactpow a ∈ InternalFuns"
  by (simp add: hfallfactpow_base_starfun2)

lemma hfallfactpow_starfun2:
  "hfallfactpow = *f2* fallfactpow"
proof (rule ext)
  fix x
  show "hfallfactpow (x::'a star)  = (*f2* fallfactpow) x "
    by (simp add: hfallfactpow_base_starfun2)
qed

lemma InternalFuns2_hfallfactpow [simp]: "hfallfactpow ∈ InternalFuns2"
  by (auto simp add: InternalFuns2_def hfallfactpow_starfun2 starfun2_eq_starfun2n)

lemma InternalFuns2_hfallfactpow_base [simp]: "(λn. hfallfactpow a) ∈
InternalFuns2"
proof (cases a, simp add: InternalFuns2_def)
  case (star_n X)
```

```
  show "∃F. (λn. hfallfactpow (star_n X)) = *fn2* F"
  proof(rule exI [where x="(λm n. fallfactpow (X m))"], (rule ext)+)
    fix n::"'c star" and x
    show "hfallfactpow (star_n X) x =
           (*fn2* (λm n. fallfactpow (X m))) n x"
    proof (cases n, cases x)
      fix Xa Xaa
      assume nx: "n = star_n Xa" "x = star_n Xaa"
      then show ?thesis
        by (auto simp add: hfallfactpow_def starfun2_star_n starfun2_n)
    qed
  qed
qed

lemma hfallfactpow: "hfallfactpow (star_n X) (star_n Y) = star_n (λn.
fallfactpow (X n) (Y n))"
by (auto simp add:  hfallfactpow_def starfun2_star_n)

lemma InternalFuns_hfallfactpow_fun:
  assumes "f ∈ InternalFuns"
  shows "(λn. hfallfactpow a (f n)) ∈ InternalFuns"
proof (insert assms, cases a, simp add: InternalFuns_def)
  case (star_n X)
  show "∃F. (λn. hfallfactpow (star_n X) (f n)) = *fn* F"
    by (metis (no_types) InternalFuns_hfallfactpow InternalFuns_starfun_n_starfun
assms starfun_n_o)
qed

lemma InternalFuns_hfallfactpow_divide:
  "(λk. (hfallfactpow (of_hypnat j) k/(of_hypnat j pow k)) :: 'a :: {comm_ring_1,
division_ring} star ) ∈ InternalFuns"
  by (simp add: InternalFuns_divide)

lemma hfact:
  "hfact (star_n X) = star_n(λn. fact (X n))"
by (auto simp add: hfact_def starfun star_n_eq_iff)

lemma InternalFuns_hfact_fun [simp]:
  "f ∈ InternalFuns ⟹ (λn. hfact (f n)) ∈ InternalFuns"
  by (metis (no_types) InternalFuns_starfun InternalFuns_starfun_n InternalFuns_starfun_n_s

      hfact_def starfun_n_o)

lemma InternalFuns_hfact [simp]: "(λn. hfact n) ∈ InternalFuns"
by (simp add: hfact_def)

lemma InternalFuns2_hfact [simp]: "(λm. hfact) ∈ InternalFuns2"
proof (simp add: InternalFuns2_def hfact_def)
  show "∃F. (λm. *f* fact) = *fn2* F"
```

```
  proof(rule exI [where x="λm n p. fact p"], (rule ext)+)
    fix m::"'c star" and x
    show "(*f* fact) x = (*fn2* (λm n. fact)) m x"
    proof(cases m, cases x)
      fix X Xa
      assume "m = star_n X" "x = star_n Xa"
      then show ?thesis
        by (auto simp add:  starfun2_n starfun)
    qed
  qed
qed

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 -
  have "(^) x ----> 0" using LIMSEQ_power_zero
    using assms star_of_less_1 by fastforce
  then have "star_of x pow n ≈ 0"
    by (metis LIMSEQ_NSLIMSEQ_iff NSLIMSEQ_def assms(2) star_zero_def
starfun_power)
  then show ?thesis
    by (simp add: mem_infmal_iff)
qed


lemma InternalFuns_abs_pow_fun:
  assumes Int_f: "f ∈ InternalFuns" shows "(λn. |x pow f n|) ∈ InternalFuns"
  by (simp add: InternalFuns_abs  assms)

lemma hfact_gt_zero [simp]: "⋀n. (0 :: 'a :: linordered_semidom star)
< hfact (n::hypnat)"
by transfer simp

lemma hypnat_fact_zero [simp] : "hfact (0 :: nat star) = 1"
by transfer simp
```

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)"
  by (metis add.commute id_apply le_add_diff_inverse of_nat_eq_id pochhammer_fact
```

```
pochhammer_product')

lemma fact_fact_hpochhammer_mult:
  "⋀n k. n ≥ k ⟹ hfact n = (hfact k) * hpochhammer (k + 1) (n - k)"
by transfer (metis fact_fact_pochhammer_mult)

lemma power_le_pochhammer:
  assumes "0 ≤ x"
  shows "(x ^ n :: 'a :: linordered_semidom) ≤ pochhammer x n"
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then have "x ^ n * x ≤ pochhammer x n * (x + of_nat n)"
    by (simp add: assms mult_mono')
  then show ?case
    by (simp add: mult.commute pochhammer_Suc)
qed

lemma hyperpow_le_hpochhammer:
  "⋀x n. 0 ≤ x ⟹ (x pow n :: 'a :: linordered_semidom star) ≤ hpochhammer
x n"
by transfer (metis power_le_pochhammer)

lemma of_nat_pochhammer_of_nat:
  assumes "0 ≤ z" shows "of_nat (pochhammer (nat z) n) = pochhammer
(of_int z) n"
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case
    by (simp add: pochhammer_Suc assms)
qed

lemma of_hypnat_hpochhammer_of_hypnat:
  "⋀z n. 0 ≤ z ⟹ of_hypnat (hpochhammer (hypnat z) n) = hpochhammer
(of_hypint z) n"
by transfer (metis of_nat_pochhammer_of_nat)

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 -
  have hpow_gt_zero: "0 < y pow n"
    by (simp add: assms(2) hyperpow_gt_zero)
  have hfl_y_2_ge_0: "0 < hypreal_of_hypint (hfloor y) + 2"
    by (simp add: add_nonneg_pos assms(2) less_imp_le)
  have hfl_y_ge0: "0 ≤ hfloor y"
    by (simp add: assms(2) less_imp_le)
  then have "hpochhammer (hypreal_of_hypint (hfloor y) + 2)
        (n - hypnat (hfloor y + 1))
       ≤ hypreal_of_hypnat
           (hpochhammer (hypnat (hfloor y + 1) + 1)
             (n - hypnat (hfloor y + 1)))"
    using hypnat_add_one of_hypint_add_one of_hypnat_hpochhammer_of_hypnat
    by (metis (no_types, lifting) add_nonneg_nonneg add.assoc le_less
one_add_one zero_less_one)
  moreover have "(hypreal_of_hypint (hfloor y) + 2) pow (n - hypnat (hfloor
y + 1))
                ≤ hpochhammer (hypreal_of_hypint (hfloor y) + 2)
                    (n - hypnat (hfloor y + 1))"
    using hfl_y_2_ge_0 hyperpow_le_hpochhammer less_imp_le by blast
  ultimately have "(hypreal_of_hypint (hfloor y) + 2) pow (n - hypnat
(hfloor y + 1))
                   ≤ hypreal_of_hypnat (hpochhammer (hypnat (hfloor y
+ 1) + 1) (n - hypnat (hfloor y + 1)))"
    by linarith
  then have "(hypreal_of_hypint (hfloor y) + 2) pow (n - hypnat (hfloor
y + 1)) *
             hypreal_of_hypnat (hfact (hypnat (hfloor y + 1)))
            ≤ hypreal_of_hypnat (hfact n)"
    using fact_fact_hpochhammer_mult [of "hypnat (hfloor y + 1)" n, OF
assms(3)]
    by simp
  moreover have "0 < (hypreal_of_hypint (hfloor y) + 2) pow
        (n - hypnat (hfloor y + 1)) *
        hypreal_of_hypnat (hfact (hypnat (hfloor y + 1)))"
  proof -
    have "hypreal_of_hypnat (hfact (hypnat (hfloor y + 1))) > 0"
      using hfact_gt_zero of_hypnat_0_less_iff by blast
    moreover have "(hypreal_of_hypint (hfloor y) + 2) pow (n - hypnat
(hfloor y + 1)) > 0"
      by (simp add: hfl_y_2_ge_0 hyperpow_gt_zero)
    ultimately show ?thesis
      using zero_less_mult_iff by blast
  qed
  ultimately show ?thesis
    by (simp add: divide_left_mono hpow_gt_zero less_imp_le)
qed
```

```
lemma hfloor_one_minus_epsilon: "hfloor(1 - ε) = 0"
proof (rule hfloor_unique)
  have "ε < 1" using Infinitesimal_less_SReal by auto
  then show "hypreal_of_hypint 0 ≤ 1 - ε" by simp
next
  show "1 - ε < hypreal_of_hypint 0 + 1"
    by (simp add: epsilon_gt_zero)
qed

lemma hypreal_hfloor_approx_zero [simp]: "⟦ (x::hypreal) ≈ 0; x > 0 ⟧
⟹ hfloor x = 0"
by (metis Infinitesimal_less_SReal Reals_1 hfloor_less_zero le_less less_add_one

    less_add_same_cancel1 mem_infmal_iff not_less zero_less_hfloor)

lemma hfloor_approx_zero [simp]: "x ≈ 0 ⟹ hfloor (hnorm x) = 0"
by (metis approx_hnorm hfloor_zero hnorm_eq_zero hypreal_hfloor_approx_zero
zero_less_hnorm_iff)


lemma InternalFuns_expf_term' [simp]:
  "(λN. y pow N / hypreal_of_hypnat (hfact N)) ∈ InternalFuns"
  by (simp add: InternalFuns_divide InternalFuns_of_hypnat)

lemma InternalFuns_expf_term [simp]:
  "(λN. y pow N / (hfact N :: real star)) ∈ InternalFuns"
  by (simp add: InternalFuns_divide)

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 -
  have "(λN. y pow N / hypreal_of_hypnat (hfact N)) ∈ InternalFuns"
    by simp
  moreover have "(λn. y pow hypnat (hfloor y) /
             hypreal_of_hypint (hfact (hypnat (hfloor y))) *
             ((y / hypreal_of_hypint (hfloor y + 1)) pow n /
              (y / hypreal_of_hypint (hfloor y + 1)) pow
              hypnat (hfloor y)))
        ∈ InternalFuns"
    using y_gt_0 y_approx_0 by simp
  moreover have "∃k∈ℕ. ∀n≥k. hnorm (y pow n / hypreal_of_hypnat (hfact
n))
                    ≤ y pow hypnat (hfloor y) /
                       hypreal_of_hypint
```

```
                        (hfact (hypnat (hfloor y))) *
                       ((y / hypreal_of_hypint (hfloor y + 1)) pow
                        n /
                        (y / hypreal_of_hypint (hfloor y + 1)) pow
                        hypnat (hfloor y))"
  proof (rule_tac bexI [of _ "hypnat (hfloor y)"],safe)
    fix n
    assume "hypnat (hfloor y) ≤ n"
    have "0 < y pow n"
      by (simp add: hyperpow_gt_zero y_gt_0)
    moreover have " 1 ≤ hypreal_of_hypnat (hfact n)"
      using zero_less_hfloor by fastforce
    ultimately have "y pow n / hypreal_of_hypnat (hfact n) ≤ y pow n"
      using divide_left_mono by fastforce
    then show "hnorm (y pow n / hypreal_of_hypnat (hfact n))
         ≤ y pow hypnat (hfloor y) /
            hypreal_of_hypint (hfact (hypnat (hfloor y))) *
            ((y / hypreal_of_hypint (hfloor y + 1)) pow n /
             (y / hypreal_of_hypint (hfloor y + 1)) pow
             hypnat (hfloor y))"
      by (simp add: <0 < y pow n> abs_of_pos y_approx_0 y_gt_0)
   next
    show "hypnat (hfloor y) ∈ ℕ"
      using HFinite_hypnat_hfloor_Nat HFinite_y by blast
  qed
  moreover have " HyperSummable (λn. y pow hypnat (hfloor y) /
             hypreal_of_hypint (hfact (hypnat (hfloor y))) *
             ((y / hypreal_of_hypint (hfloor y + 1)) pow n /
              (y / hypreal_of_hypint (hfloor y + 1)) pow
              hypnat (hfloor y)))"
  proof -
    have "(λn. (y / hypreal_of_hypint (hfloor y + 1)) pow n /
              (y / hypreal_of_hypint (hfloor y + 1)) pow
              hypnat (hfloor y)) ∈ InternalFuns"
      using  y_gt_0 y_approx_0 by auto
    moreover have "y pow hypnat (hfloor y) /
        hypreal_of_hypint (hfact (hypnat (hfloor y))) ∈ HFinite"
      using y_approx_0 y_gt_0 by auto
    moreover have "(pow) (y / hypreal_of_hypint (hfloor y + 1)) ∈ InternalFuns"
      using InternalFuns_hyperpow_simple by blast
    moreover have "HyperSummable ((pow) (y / hypreal_of_hypint (hfloor
y + 1)))"
      by (metis HyperSummable_geometric abs_of_pos add_cancel_right_left
approx_trans3 div_by_1
          hfloor_correct hfloor_one hypreal_hfloor_approx_zero hypreal_hnorm_def
of_hypint_1
          one_neq_zero y_approx_0 y_gt_0 zero_less_one)
    moreover have "(y / hypreal_of_hypint (hfloor y + 1)) pow hypnat
(hfloor y) ∉ Infinitesimal"
```

```
    using y_approx_0 y_gt_0 by auto
  ultimately show ?thesis
    using HyperSummable_HFinite_mult_left HyperSummable_divide by blast
 qed
 ultimately show ?thesis
  using HyperSummable_comparison_test by blast
qed

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 -
 have "(λN. y pow N / hypreal_of_hypnat (hfact N)) ∈ InternalFuns"
  by simp
 moreover have "(λn. y pow hypnat (hfloor y + 1) /
            hypreal_of_hypnat (hfact (hypnat (hfloor y + 1))) *
            ((y / hypreal_of_hypint (hfloor y + 2)) pow n /
             (y / hypreal_of_hypint (hfloor y + 2)) pow
             hypnat (hfloor y + 1))) ∈ InternalFuns"
  using y_gt_0 by (auto simp add: hyperpow_divide hyperpow_diff_cancel
InternalFuns_divide
       InternalFuns_mult_const_right InternalFuns_mult_const_left)
 moreover have "∃k∈ℕ. ∀n≥k. hnorm (y pow n / hypreal_of_hypnat (hfact
n))
                   ≤ y pow hypnat (hfloor y + 1) /
                     hypreal_of_hypnat
                      (hfact (hypnat (hfloor y + 1))) *
                     ((y / hypreal_of_hypint (hfloor y + 2)) pow
                      n /
                      (y / hypreal_of_hypint (hfloor y + 2)) pow
                      hypnat (hfloor y + 1))"
 proof (rule bexI [of _ "hypnat ((hfloor y) + 1)"],safe)
  fix n
  have "y / hypreal_of_hypint (hfloor y + 2) ≠ 0"
   by (metis divide_eq_0_iff le_less less_add_same_cancel1 not_less
of_hypint_0_eq_iff y_gt_0
       zero_le_hfloor zero_less_numeral)
  moreover assume ge_floor: "hypnat (hfloor y + 1) ≤ n"
  ultimately have "(y / hypreal_of_hypint (hfloor y + 2)) pow n /
       (y / hypreal_of_hypint (hfloor y + 2)) pow
       hypnat (hfloor y + 1) =
       (y / hypreal_of_hypint (hfloor y + 2)) pow
       (n - hypnat (hfloor y + 1))"
    using hyperpow_diff
    by blast
  moreover have "|y pow n| / hypreal_of_hypnat (hfact n)
     ≤ y pow hypnat (hfloor y + 1) *
```

```
          (y / (hypreal_of_hypint (hfloor y) + 2)) pow
          (n - hypnat (hfloor y + 1)) /
          hypreal_of_hypnat (hfact (hypnat (hfloor y + 1)))"
      by (simp add: HFinite_y ge_floor hyperpow_diff_cancel hyperpow_divide
hyperpow_divide_fact_le_lemma' y_gt_0)
    ultimately show " hnorm (y pow n / hypreal_of_hypnat (hfact n))
        ≤ y pow hypnat (hfloor y + 1) /
           hypreal_of_hypnat (hfact (hypnat (hfloor y + 1))) *
           ((y / hypreal_of_hypint (hfloor y + 2)) pow n /
            (y / hypreal_of_hypint (hfloor y + 2)) pow
            hypnat (hfloor y + 1))"
      by simp
  next
    show "hypnat (hfloor y + 1) ∈ ℕ"
     using HFinite_hypnat_hfloor_Nat HFinite_y
     by (metis HFinite_1 HFinite_add hfloor_add_one)
  qed
  moreover have "(λn. (y / hypreal_of_hypint (hfloor y + 2)) pow n /
        (y / hypreal_of_hypint (hfloor y + 2)) pow
        hypnat (hfloor y + 1))
    ∈ InternalFuns"
    using InternalFuns_const_fun InternalFuns_divide InternalFuns_hyperpow_simple

    by blast
  moreover have "y pow hypnat (hfloor y + 1) /
                hypreal_of_hypnat (hfact (hypnat (hfloor y + 1))) ∈ HFinite"
    using HFinite_divide hypnat_add_distrib HFinite_hyperpow_HFinite
          HFinite_hypnat_hfloor_Nat neq_iff  assms  hfloor_of_hypnat
          hfact_gt_zero hypreal_hfloor_approx_zero mem_infmal_iff of_hypnat_0_less_iff
    by (metis HFinite_1 HFinite_add hfloor_add_one)
  moreover have "(pow) (y / hypreal_of_hypint (hfloor y + 2)) ∈ InternalFuns"
    by simp
  moreover have "HyperSummable ((pow) (y / hypreal_of_hypint (hfloor
y + 2)))"
  proof -
    have gt0: "0 < hypreal_of_hypint (hfloor y) + 2"
      by (simp add: add_nonneg_pos less_imp_le y_gt_0)
    then have " hnorm (y / (hypreal_of_hypint (hfloor y) + 2)) < 1"
      using assms add_le_cancel_left hfloor_less_cancel less_hfloor_iff
one_le_numeral
      by (metis abs_of_pos divide_less_eq_1_pos divide_pos_pos hypreal_hnorm_def)
    moreover have "¬ hnorm (y / (hypreal_of_hypint (hfloor y) + 2)) ≈
1"
    proof
      assume "hnorm (y / (hypreal_of_hypint (hfloor y) + 2)) ≈ 1"
      then have "y / (hypreal_of_hypint (hfloor y) + 2) ≈ 1"
        by (simp add: abs_of_pos gt0 y_gt_0)
      moreover have "hypreal_of_hypint (hfloor y) + 2 ∈ HFinite"
        using assms
```

```
      by (metis HFinite_add HFinite_bounded HFinite_numeral
            hfloor_add_one_gt_zero hfloor_correct hypint_le_add1_eq_le
of_hypint_0_le_iff)
    ultimately have "y ≈ hypreal_of_hypint (hfloor y) + 2"
      using gt0 assms approx_mult2
      by fastforce
    moreover have "hpart y ≈ 2"
      using approx_add_left_iff hypreal_eq_hfloor_hpart_add
      by (metis calculation)
    ultimately show False
      by (metis add.commute add_cancel_right_left approx_add_left_cancel

          approx_le_bound approx_sym hfloor_one hpart_le_one hypreal_hfloor_approx_zero

          one_add_one one_le_numeral zero_less_one zero_neq_one)
  qed
  ultimately show ?thesis
    by (simp add: HyperSummable_geometric)
 qed
 moreover have "(y / hypreal_of_hypint (hfloor y + 2)) pow hypnat (hfloor
y + 1) ∉ Infinitesimal"
 proof
  have "(hypreal_of_hypint (hfloor(y)) + 2) pow hypnat (hfloor(y) +
(1 :: int star)) ∈ HFinite"
    by (metis HFinite_1 HFinite_HInfinite_iff HFinite_add HFinite_hyperpow_HFinite

        HFinite_hypnat_hfloor_Nat HFinite_numeral HFinite_y hfloor_add_one
hypreal_HInfinite_hfloor_cancel)
  moreover assume contra: "(y / hypreal_of_hypint (hfloor y + 2))
pow hypnat (hfloor y + 1) ∈ Infinitesimal"
  ultimately have "y pow hypnat (hfloor y + 1) /
   (hypreal_of_hypint (hfloor y) + 2) pow hypnat (hfloor y + 1) *
   (hypreal_of_hypint (hfloor y) + 2) pow hypnat (hfloor y + 1) ∈ Infinitesimal"
    by (metis Infinitesimal_HFinite_mult hyperpow_divide of_hypint_add
of_hypint_numeral)
  then have "y pow hypnat (hfloor y + 1) ∈ Infinitesimal"
    by (metis add_nonneg_pos hyperpow_gt_zero less_imp_le nonzero_mult_div_cancel_right
of_hypint_0_le_iff
        order_less_irrefl times_divide_eq_left y_gt_0 zero_le_hfloor
zero_less_numeral)
  moreover have "hypnat ((hfloor y) + (1 :: int star)) ∈ ℕ"
    by (metis HFinite_1 HFinite_add HFinite_hypnat_hfloor_Nat HFinite_y
hfloor_add_one)
  moreover have "hypnat (hfloor y + 1) ≠ 0"
    using y_gt_0 by force
  moreover have "hypreal_of_hypint (hfloor y) + 2 ≠ 0"
    by (metis add_nonneg_pos less_imp_le less_numeral_extra(3) of_hypint_0_le_iff
y_gt_0
        zero_le_hfloor zero_less_numeral)
```

```
    ultimately show False
      using assms Nats_hypnat_of_nat_iff hrealpow_hyperpow_Infinitesimal_iff
[symmetric]
            mem_infmal_iff [symmetric] not_gr0 Infinitesimal_hrealpow_cancel
star_of_simps(9)
      by metis
  qed
  ultimately show ?thesis
    using HyperSummable_HFinite_mult_left
            [of _ "(y pow hypnat ((hfloor y) + 1))/of_hypnat(hfact (hypnat
((hfloor y) + 1)))"]
          HyperSummable_divide
          HyperSummable_comparison_test
    by blast
qed

lemma HyperSummable_hyperpow_divide_fact_pos:
  "⟦ (y::hypreal) ∈ HFinite; y > 0 ⟧ ⟹ HyperSummable (λN. y pow N/of_hypnat(hfact
N))"
by (blast intro: HyperSummable_hyperpow_divide_fact_Infinitesimal
            HyperSummable_hyperpow_divide_fact_not_Infinitesimal)


lemma HyperSummable_hyperpow_divide_fact_neg:
  assumes "(y::hypreal) ∈ HFinite"
  and "y < 0"
  shows "HyperSummable (λN. y pow N/of_hypnat(hfact N))"
proof -
  have "(λN. y pow N / hypreal_of_hypnat (hfact N)) ∈ InternalFuns"
    by simp
  moreover have "(λN. (- y) pow N / hypreal_of_hypnat (hfact N)) ∈ InternalFuns"
    by simp
  moreover have "∃k∈ℕ. ∀n≥k. hnorm (y pow n / hypreal_of_hypnat (hfact
n))
                    ≤ (- y) pow n / hypreal_of_hypnat (hfact n)"
    using abs_of_neg eq_iff hyperpow_hrabs
    by (metis Nats_1 assms(2) hnorm_divide hnorm_hypreal_of_hypnat hypreal_hnorm_def)
  moreover have "HyperSummable (λN. (- y) pow N / hypreal_of_hypnat (hfact
N))"
    by (simp add: HFinite_minus_iff HyperSummable_hyperpow_divide_fact_pos
assms(1) assms(2))
  ultimately show ?thesis
    using HyperSummable_comparison_test by blast
qed

lemma InternalFuns_star_choose: "(hchoose) k ∈ InternalFuns"
  by (simp add: hyperbinomial_def)

lemma InternalFuns_star_choose_from [simp]: "(λn. n hchoose k) ∈ InternalFuns"
```

```
proof (cases k, simp add: hyperbinomial_def InternalFuns_def)
  case (star_n X)
  show "∃F. (λn. (*f2* (choose)) n (star_n X)) = *fn* F"
  proof (rule exI [where x="λn m. m choose (X n)"], rule ext)
    fix n :: hypnat
    obtain Y where "n = star_n Y"
      using star_cases by auto
    then show "(*f2* (choose)) n (star_n X) = (*fn* (λn m. m choose X
n)) n"
      by (auto simp add: starfun2_star_n starfun_n)
  qed
qed


lemma hypnat_fact_not_zero [simp]: "hfact n ≠ (0::hypnat)"
by (metis hfact_gt_zero hypnat_not_less0)

lemma hfact_nonzero [simp]:
  "⋀k. hfact k ≠ (0::'a::{semiring_char_0,semiring_no_zero_divisors}
star)"
by transfer simp

lemma HyperSummable_geometric':
  assumes "hnorm (x::'a::{real_normed_field} star) < 1"
          "¬hnorm x ≈ 1"
        shows "HyperSummable (λN. x pow (hSuc N))"
  by (simp add: HyperSummable_geometric HyperSummable_shift_hSuc assms)

lemma HyperSummable_hyperpow_divide_fact_zero:
  "HyperSummable (λn. 0 pow n / hypreal_of_hypnat (hfact n))"
proof -
  {fix n
  assume infinite_n: "n ∈ HNatInfinite"
  then have internalf: "(λn. 0 pow n / hypreal_of_hypnat (hfact n)) ∈
InternalFuns"
    using InternalFuns_expf_term' by blast
  moreover have "n > 0" using infinite_n
    by (simp add: zero_less_HNatInfinite)
  moreover have "hypersum (λn. 0 pow n / hypreal_of_hypnat (hfact n))
{0..<n} ≈ 1"
  proof -
    have "hypersum (λi. 0 pow hSuc i / hypreal_of_hypnat (hfact (hSuc
i)))  {0..<n - 1} ≈ 0"
      by simp
    then have "hypersum (λn. 0 pow n / hypreal_of_hypnat (hfact n)) {hSuc
0..<hSuc (n - 1)} ≈ 0"
      using  hypersum_shift_bounds_hSuc_ivl [OF internalf] by simp
    then have "hypersum (λn. 0 pow n / hypreal_of_hypnat (hfact n)) {hSuc
0..<n} ≈ 0"
```

```
    by (simp add: infinite_n)
  then have "0 pow 0 / hypreal_of_hypnat (hfact 0) +
      hypersum (λn. 0 pow n / hypreal_of_hypnat (hfact n))
       {hSuc 0..<n} ≈
      1"
    using approx_minus_iff by force
  then show ?thesis
    by (simp add: calculation(2) hypersum_head_upt_hSuc)
 qed}
 then show ?thesis
  using HyperSummable_def2 by blast
qed

lemma HyperSummable_exp':
 "(y::hypreal) ∈ HFinite  ⟹ HyperSummable (λN. y pow N/of_hypnat(hfact
N))"
by (metis (no_types) HyperSummable_hyperpow_divide_fact_pos
     HyperSummable_hyperpow_divide_fact_neg HyperSummable_hyperpow_divide_fact_zero

     linorder_cases)

lemma of_hypnat_hfact:
 "⋀n. of_hypnat(hfact n) = hfact n"
by transfer simp

lemma HyperSummable_exp:
 "(y::hypreal) ∈ HFinite  ⟹ HyperSummable (λN. y pow N/hfact N)"
proof -
  assume fin: "(y::hypreal) ∈ HFinite"
  have "HyperSummable (λN. y pow N/of_hypnat (hfact N)) = HyperSummable
(λN. y pow N/hfact N)"
       by (subst of_hypnat_hfact) simp
  thus ?thesis using HyperSummable_exp' fin by blast
qed

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 -
 have "(λk. of_hypnat ((*f2* (choose)) (star_n Xb) k) *
       (*f2* fallfactpow) (star_n X) (star_n Xb - k) *
       (*f2* fallfactpow) (star_n Xa) k)
```

```
        ∈ InternalFuns"
    by (metis (no_types) InternalFuns_hfallfactpow_fun InternalFuns_mult
InternalFuns_of_hypnat
        InternalFuns_starfun2 hfallfactpow_starfun2 star_diff_def)
  moreover have "(λk. of_hypnat ((*f2* (choose)) (star_n Xb) k) *
                      (*f2* fallfactpow) (star_n X) (star_n Xb - k) *
                      (*f2* fallfactpow) (star_n Xa) k) =
                *fn* (λn k. of_nat (Xb n choose k) *
                            fallfactpow (X n) (Xb n - k) *
                            fallfactpow (Xa n) k)"
  proof
    fix k
     obtain Xc :: "nat ⇒ nat" where "k = star_n Xc"
      using star_cases by auto
    then
    show "of_hypnat ((*f2* (choose)) (star_n Xb) k) *
         (*f2* fallfactpow) (star_n X) (star_n Xb - k) *
         (*f2* fallfactpow) (star_n Xa) k =
         (*fn* (λn k. of_nat (Xb n choose k) *
                     fallfactpow (X n) (Xb n - k) *
                     fallfactpow (Xa n) k)) k"
      by (auto simp add: starfun2_star_n star_n_diff star_n_mult
          of_hypnat_def starfun_star_n starfun_n)
  qed
  ultimately show ?thesis
    using starfun_n_eq_cancel n_starfun_starfun_n_eq_ultra by auto
qed

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 (cases a, cases b, cases n)
  fix X Xa Xb
  assume star_assms: "a = star_n X" "b = star_n Xa" "n = star_n Xb"
  then have ev: "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"
    by (simp add: lemma_n_starfun_extract_binomial_hfallfactpow)
  moreover
  {fix x
    assume "(*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))x =
            (λk. of_nat (Xb x choose k) * fallfactpow (X x) (Xb x - k)
*
```

```
                fallfactpow (Xa x) k)"
    and "(*ns* {0..star_n Xb}) x = {0..Xb x}"
    then have "fallfactpow (X x + Xa x) (Xb x) =
           sum ((*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))
                x) ((*ns* {0..star_n Xb}) x)"
      using binomial_fallfactpow_ring by auto
  }
  then have "∀F x in U.
              (*ns* {0..star_n Xb}) x = {0..Xb x} ⟶
              fallfactpow (X x + Xa x) (Xb x) =
              sum ((*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))
                   x) ((*ns* {0..star_n Xb}) x)"
    using eventually_mono [OF ev]
    by simp
  then have "∀F x in U.
              fallfactpow (X x + Xa x) (Xb x) =
              sum ((*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))
                   x) ((*ns* {0..star_n Xb}) x)"
    by (rule eventually_mp [OF _ n_star_interval_ultra])
  then show ?thesis using star_assms
    by (simp add: hfallfactpow_starfun2 hyperbinomial_def hypersum_def
star_add_def star_n_eq_iff starfun2_star_n)
qed


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}"
  by (subst add.commute, simp add: hyperbinomial_fallfactpow_ring mult.commute
mult.left_commute)

lemma hyperbinomial_hfact:
 "⋀n k. k ≤ n ⟹ of_hypnat (n hchoose k) = (hfact n :: 'a:: field_char_0
star) / (hfact k * hfact (n - k))"
by (transfer, metis binomial_fact)
```

```
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 -
   obtain As Bs where isets: "A = *sn* As" "B = *sn* Bs"  "∀_F n in 𝒰.
As n = Bs n"
      using iset_A `A = B` by (force simp add: InternalSets_def)
   obtain G H where ifuns: "g = *fn* G" "h = *fn* H" using ifun_g ifun_h
by (force simp add: InternalFuns_def)
   {fix X assume "star_n X ∈ *sn* Bs"
    then have "(*fn* G)(star_n X) = (*fn* H)(star_n X)" using g_h isets
ifuns by blast
    then have "∀_F n in 𝒰. (G n)(X n) = (H n)(X n)" by (simp add: star_n_eq_iff
starfun_n)
   }
  then have "∀X. ∀_F n in 𝒰. (X n) ∈ (Bs n) ⟶ (G n)(X n) = (H n)(X
n)"
      by (simp add: FreeUltrafilterNat.eventually_imp_iff starset_n_star_n)
  then have ultra_g_h: "∀_F n in 𝒰. ∀x. x ∈ (Bs n) ⟶ (G n) x = (H n)
x"
      by (simp add: FreeUltrafilterNat.eventually_all_iff)
  have "∀_F n in 𝒰. (As n = Bs n ∧ (∀x. x ∈ (Bs n) ⟶ (G n) x = (H n)
x)
                          ⟶ sum (G n) (As n) = sum (H n) (Bs n))"
       by simp
   then have "∀_F n in 𝒰. sum (G n) (As n) = sum (H n) (Bs n)"
       using eventually_elim2 isets(3) ultra_g_h by fastforce
   thus ?thesis using isets(1,2) ifuns
       by (auto simp only: hypersum star_n_eq_iff )
qed

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 -
   have ifun_a: "(λi. of_hypnat (k hchoose i) * r pow i * s pow (k - i))
∈ InternalFuns"
     by (simp add: InternalFuns_diff InternalFuns_mult InternalFuns_of_hypnat
InternalFuns_star_choose)
   have ifun_b: "(λi. r pow i * s pow (k - i) / (hfact i * hfact (k -
i))) ∈ InternalFuns"
     by (simp add: InternalFuns_diff InternalFuns_divide InternalFuns_mult)
   then have ifun_c: "(λi. hfact k * r pow i * s pow (k - i) / (hfact
i * hfact (k - i))) ∈ InternalFuns"
```

```
    by (simp add: InternalFuns_diff InternalFuns_divide InternalFuns_mult)
   have "(r + s) pow k  =  hypersum (λi. of_hypnat (k hchoose i) * r
pow i * s pow (k - i)) {0..k}"
    using hyperbinomial_ring by blast
   also have  "... = hypersum (λi. hfact k / (hfact i * hfact (k - i))
* r pow i * s pow (k - i)) {0..k}"
      using ifun_a ifun_c by (auto intro!: hypersum_cong simp add: hyperbinomial_hfact)
   also have "... =  hfact k * hypersum (λi. r pow i * s pow (k - i)
/ (hfact i * hfact (k - i))) {0..k}"
      using  ifun_b by (subst hypersum_right_distrib, auto simp add:
field_simps)
  ultimately show ?thesis by auto
qed


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 -
  have ifun_a: "(λi. of_hypnat (k hchoose i) * hfallfactpow r i *  hfallfactpow
s (k - i)) ∈ InternalFuns"
    by (simp add: InternalFuns_diff InternalFuns_hfallfactpow_fun InternalFuns_mult
InternalFuns_of_hypnat
        InternalFuns_star_choose)
  have ifun_b: "(λi. hfallfactpow r i * hfallfactpow s (k - i) / (hfact
i * hfact (k - i))) ∈ InternalFuns"
    by (simp add: InternalFuns_divide InternalFuns_hfallfactpow_fun InternalFuns_mult
star_class_defs(4))
  then have ifun_c: "(λi. hfact k * hfallfactpow r i * hfallfactpow
s (k - i) / (hfact i * hfact (k - i))) ∈ InternalFuns"
    by (simp add: InternalFuns_diff InternalFuns_divide InternalFuns_hfallfactpow_fun
InternalFuns_mult)
  have "hfallfactpow (r + s) k  =  hypersum (λi. of_hypnat (k hchoose
i) * hfallfactpow r i * hfallfactpow s (k - i)) {0..k}"
    using hyperbinomial_fallfactpow_ring2 by blast
  also have  "... = hypersum (λi. hfact k / (hfact i * hfact (k - i))
* hfallfactpow r i * hfallfactpow s (k - i)) {0..k}"
      using ifun_a ifun_c by (auto intro!: hypersum_cong simp add: hyperbinomial_hfact)
  also have "... =  hfact k * hypersum (λi. hfallfactpow r i * hfallfactpow
s (k - i) / (hfact i * hfact (k - i))) {0..k}"
      using  ifun_b by (subst hypersum_right_distrib, auto simp add:
field_simps)
  ultimately show ?thesis by auto
qed

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 -
  have "(λk. of_hypnat (n hchoose k) * hfallfactpow a (n - k) * hfallfactpow
b k) ∈ InternalFuns"
    by (simp add: InternalFuns_hfallfactpow_fun InternalFuns_mult InternalFuns_of_hypnat

        InternalFuns_star_choose star_diff_def)
  moreover have "(λk. hfallfactpow (of_hypnat n) k * hfallfactpow a (n
- k) * hfallfactpow b k / of_hypnat (hfact k)) ∈ InternalFuns"
    by (simp add: InternalFuns_divide InternalFuns_mult InternalFuns_o2
InternalFuns_of_hypnat star_diff_def)
  ultimately show ?thesis
    by (force intro!: hypersum_cong simp add: hyperbinomial_fallfactpow_ring
hyperbinomial_hfallfactpow)
qed
```

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"
by (induct n, auto simp add: algebra_simps)

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"
by transfer (erule binomial_expand_first_two_terms)

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"
  by (metis add.commute hyperbinomial_expand_first_two_terms)

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_ℝ) x a"

lemma "(hpow) ∈ InternalFuns2"
  unfolding InternalFuns_def hpow_def
  by (simp add: starfun2_eq_starfun2n)
```

lemma *hpow: "(star_n X) hpow (star_n Y) = star_n (λn. (X n) pow$_{\mathbb{R}}$ (Y n))"*
by *(metis hpow_def starfun2_star_n)*

9.1 Tranferred properties

lemma *hpow_one_eq_one [simp]: "⋀a. 1 hpow a = 1"*
by *transfer simp*

lemma *hpow_zero_eq_one [simp]: "⋀a. a > 0 ⟹ a hpow 0 = 1"*
by *transfer simp*

lemma *hpow_minus_one: "⋀a. 0 < a ⟹ a hpow -1 = inverse a"*
by *transfer (simp add: powreal_minus_one)*

lemma *hpow_one [simp]: "⋀a. a > 0 ⟹ a hpow 1 = a"*
by *transfer simp*

lemma *hpow_gt_zero: "⋀a x. a > 0 ⟹ a hpow x > 0"*
by *(transfer, rule powreal_gt_zero)*

lemma *hpow_not_zero: "⋀a x. a > 0 ⟹ a hpow x ≠ 0"*
by *(transfer, rule powreal_not_zero)*

lemma *hpow_minus: "⋀a x. a > 0 ⟹ a hpow (-x) = inverse (a hpow x)"*
by *(transfer, rule powreal_minus)*

lemma *hpow_inverse:*
"⋀a x. a > 0 ⟹ inverse (a hpow x) = (inverse a) hpow x"
by *(transfer, rule powreal_inverse)*

lemma *hpow_mult_base:*
*"⋀a b x. ⟦ 0 < a; 0 < b ⟧ ⟹ (a * b) hpow x = (a hpow x) * (b hpow x)"*
by *transfer (simp add: powreal_mult_base)*

lemma *hpow_mult:*
*"⋀a x y. 0 < a ⟹ (a hpow x) hpow y = a hpow (x * y)"*
by *(transfer, rule powreal_mult)*

lemma *hpow_divide:*
"⋀a b x. ⟦ 0 < a; 0 < b ⟧ ⟹ (a/b) hpow x = (a hpow x) / (b hpow x)"
by *transfer (simp add: powreal_divide)*

lemma *hpow_divide2: "⋀a x y. a > 0 ⟹ a hpow (x - y) = (a hpow x)/(a hpow y)"*
by *transfer (simp add: powreal_divide2)*

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

by *(transfer, rule powreal_ge_one)*

lemma *hpow_ge_one2:*
"⋀a x. ⟦ 0 < a; a < 1; x ≤ 0 ⟧ ⟹ a hpow x ≥ 1"
by *(transfer, rule powreal_ge_one2)*

lemma *hpow_le_mono:*
"⋀r s a. ⟦ r ≤ s; a ≥ 1 ⟧ ⟹ a hpow r ≤ a hpow s"
by *(transfer, rule powreal_le_mono)*

lemma *hpow_le_anti_mono:*
"⋀r s a. ⟦ r ≤ s; 0 < a; a < 1 ⟧ ⟹ a hpow r ≥ a hpow s"
by *(transfer, rule powreal_le_anti_mono)*

lemma *hpow_less_cancel:*
"⋀r s a. ⟦ a hpow r < a hpow s; a ≥ 1 ⟧ ⟹ r < s"
by *(transfer, rule powreal_less_cancel)*

lemma *hpow_less_anti_mono:*
"⋀r s a. ⟦ r < s; 0 < a; a < 1 ⟧ ⟹ a hpow r > a hpow s"
by *(transfer, rule powreal_less_anti_mono)*

lemma *hpow_less_mono:*
"⋀r s a. ⟦ r < s; a > 1 ⟧ ⟹ a hpow r < a hpow s"
by *(transfer, rule powreal_less_mono)*

lemma *hpow_le_cancel:*
"⋀r s a. ⟦ a hpow r ≤ a hpow s; a > 1 ⟧ ⟹ r ≤ s"
by *(transfer, rule powreal_le_cancel)*

lemma *hpow_less_cancel_iff [simp]:*
"⋀r s a. 1 < a ⟹ (a hpow r < a hpow s) = (r < s)"
by *(transfer, rule powreal_less_cancel_iff)*

lemma *hpow_le_cancel_iff [simp]:*
"⋀r s a. 1 < a ⟹ (a hpow r ≤ a hpow s) = (r ≤ s)"
by *(transfer, rule powreal_le_cancel_iff)*

lemma *hpow_inject_exp1 [simp]:*
"⋀r s a. 1 < a ⟹ (a hpow r = a hpow s) = (s = r)"
by *(transfer, rule powreal_inject_exp1)*

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

lemma *hpow_inject [simp]:*
"⋀r s a. 0 < a ⟹ a ≠ 1 ⟹ (a hpow r = a hpow s) = (s = r)"
by *(transfer, rule powreal_inject)*

lemma *hpow_gt_one:* "⋀a x. a > 1 ⟹ x > 0 ⟹ a hpow x > 1"
by *(transfer, rule powreal_gt_one)*

lemma *hpow_less_mono_base:*
"⋀a b r. ⟦ r > 0; 0 < a; a < b ⟧ ⟹ a hpow r < b hpow r"
by *(transfer, rule powreal_less_mono_base)*

lemma *hpow_less_antimono_base:*
"⋀a b r. ⟦ r < 0; 0 < a; a < b ⟧ ⟹ a hpow r > b hpow r"
by *(transfer, rule powreal_less_antimono_base)*

lemma *hpow_hyperpow_eq:*
"⋀a n. a > 0 ⟹ a hpow (of_hypnat n) = a pow n"
by *(transfer, rule powreal_power_eq)*

lemma *hpow_hyperpow_eq2:*
"⋀a n. 0 ≤ a ⟹ 0 < n ⟹ a pow n = (if (a = 0) then 0 else a hpow (of_hypnat n))"
by *transfer (simp add: powreal_power_eq2)*

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))"
by *transfer (metis powreal_int)*

lemma *hpow_power_eq:*
"⋀a. a> 0 ⟹ a hpow hypreal_of_nat n = a ^ n"
by *transfer (simp add: powreal_power_eq)*

lemma *hpow_inverse_of_hypnat_gt_one:*
"⋀a N. ⟦ a > 1; N ≠ 0 ⟧ ⟹ a hpow (inverse(of_hypnat N)) > 1"
by *transfer (auto intro: powreal_inverse_of_nat_gt_one)*

lemma *hpow_inverse_of_nat_gt_one:*
"⋀a. ⟦ a > 1; N ≠ 0 ⟧ ⟹ a hpow (inverse(of_nat N)) > 1"
by *transfer (auto intro: powreal_inverse_of_nat_gt_one)*

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"
using *LIMSEQ_iff LIMSEQ_power_zero*
by *transfer force*

lemma *hpow_add:* "⋀a x y. 0 < a ⟹ a hpow (x + y) = a hpow x * a hpow y"
by *transfer (metis powreal_add)*

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

```
by (metis hpow_add hpow_one one_add_one)

lemma hpow_minus_eq_hpow:
  "⟦ 0 < a; 0 < b; a hpow -x = b hpow y ⟧ ⟹ a hpow x = b hpow - y"
by (metis hpow_minus inverse_inverse_eq)


lemma hpow_half_divide_eq:
  "0 < y ⟹ y hpow (1/2) / y = inverse(y hpow (1/2))"
proof -
  assume a1: "0 < y"
  then have "y = y hpow 1"
    by simp
  then have "y = y hpow (1/2) * y hpow (1/2)"
    by (metis a1 field_sum_of_halves hpow_add)
  moreover have "y hpow (1/2) > 0"
    by (simp add: a1 hpow_gt_zero)
  ultimately show ?thesis
    by (metis divide_eq_0_iff inverse_eq_divide nonzero_divide_mult_cancel_right)

qed

lemma less_hpow_half_lemma:
  "⟦ 0 < x; 0 < y; x < y hpow (1/2) ⟧ ⟹ x/y < inverse(y hpow (1/2))"
by (metis divide_strict_right_mono hpow_half_divide_eq)

lemma le_hpow_half_lemma:
  "⟦ 0 < x; 0 < y; x ≤ y hpow (1/2) ⟧ ⟹ x/y ≤ inverse(y hpow (1/2))"
by (metis divide_le_cancel hpow_half_divide_eq less_asym)
```

9.2 Relating pow and hpow

```
lemma hpow_hyperpow_eq_hpow: "0 < a ⟹ (a hpow x) pow n = a hpow (x
* of_hypnat n)"
by (metis hpow_gt_zero hpow_hyperpow_eq hpow_mult)

lemma hpow_divide_pow_cancel:
  assumes "a > 1" "j ≠ 0"
  shows "(a hpow (z/of_hypnat j)) pow j = a hpow z"
using assms hpow_hyperpow_eq_hpow by simp

lemma hpow_inverse_of_hypnat_cancel:
     "⟦ 0 < a; 0 < N ⟧ ⟹ a = (a hpow (inverse(of_hypnat N))) hpow (of_hypnat
N)"
by (simp add: hpow_mult)

lemma hpow_inverse_of_hypnat_pow_cancel:
   "⟦ 0 < a; 0 < N ⟧ ⟹ (a hpow (inverse(of_hypnat N))) pow N = a"
by (metis hpow_gt_zero hpow_hyperpow_eq hpow_inverse_of_hypnat_cancel)
```

lemma *hpow_inverse_of_hypnat_pow_cancel2:*
"⟦ a > 1; N ∈ HNatInfinite ⟧ ⟹ a = (a hpow (inverse(of_hypnat N))) pow N"
by *(metis hpow_inverse_of_hypnat_pow_cancel order_less_trans zero_less_HNatInfinite zero_less_one)*

lemma *HNatInfinite_hpow_inverse_gt_one:*
"⟦ a > 1; N ∈ HNatInfinite ⟧ ⟹ a hpow (inverse(of_hypnat N)) > 1"
by *(metis HNatInfinite_of_hypnat_gt_zero hpow_gt_one inverse_positive_iff_positive)*

lemma *HNatInfinite_hpow_eq_Ex:*
assumes *"a > 1"* **and** *"N ∈ HNatInfinite"*
shows *"∃u>0. a hpow (inverse(of_hypnat N)) = 1 + u"*
proof -
have *"a hpow inverse (hypreal_of_hypnat N) > 1"*
by *(simp add: HNatInfinite_hpow_inverse_gt_one assms)*
then have *"a hpow inverse (hypreal_of_hypnat N) - 1 > 0"*
by *simp*
then show *?thesis* **by** *fastforce*
qed

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 -
have *"(a hpow inverse (hypreal_of_nat m)) ^ m ≈ 1"*
using *hrealpow_approx_one assms(3)* **by** *blast*
moreover have *"(a hpow inverse (hypreal_of_hypnat (of_nat m))) pow of_nat m = a"*
using *assms(1) assms(2) hpow_inverse_of_hypnat_pow_cancel of_nat_0_less_iff*
by *blast*
ultimately show *?thesis*
by *(simp add: hyperpow_hypnat_of_nat)*
qed

lemma *HNatInfinite_hpow_inverse_approx_one_cancel:*
"⟦ a > 0; ¬(a ≈ 1); (a hpow (inverse(of_hypnat N))) ≈ 1; N ≠ 0 ⟧ ⟹ N ∈ HNatInfinite"
using *HNatInfinite_not_Nats_iff Nats_def hpow_inverse_hypreal_of_nat_approx_one_cancel*

by *(auto simp add: Nats_def)*

lemma *HFinite_hpow1:*
assumes *"a ∈ HFinite"*

```
      and "x ∈ HFinite"
      and "1 ≤ a"
  shows "a hpow x ∈ HFinite"
proof (cases "a = 1")
  case True
  then show ?thesis by simp
next
  case False
  then have a_gt_1: "a > 1"
    using assms(3) le_less by simp
  obtain n where "x < hypreal_of_nat n"
    using HFinite_less_Nat_Ex Nats_cases assms(2) by blast
  then have "a hpow x < a hpow hypreal_of_nat n"
    using hpow_less_mono a_gt_1
    by blast
  then have "a hpow x < a ^ n"
    using assms(3) hpow_power_eq by auto
  moreover have "a ^ n ∈ HFinite"
    by (simp add: assms(1) hrealpow_HFinite)
  ultimately show ?thesis
    by (meson HFinite_bounded assms(3) hpow_gt_zero less_imp_le less_le_trans
zero_less_one)
qed


lemma HFinite_hpow2:
  assumes "a ∈ HFinite - Infinitesimal"
      and "x ∈ HFinite"
      and "0 < a"
      and "a < 1"
  shows "a hpow x ∈ HFinite"
proof -
  have "1 < 1/a"
    by (simp add: assms(3) assms(4))
  moreover have "-x ∈ HFinite"
    by (simp add: HFinite_minus_iff assms(2))
  ultimately have "(1/a) hpow -x ∈ HFinite"
    by (metis HFinite_hpow1 HFinite_inverse2 assms(1) inverse_eq_divide
less_le)
  then show ?thesis
    by (simp add: assms(3) hpow_divide hpow_minus)
qed

lemma HFinite_hpow3:
   "⟦ a ∈ HFinite - Infinitesimal; x ∈ HFinite; 0 < a ⟧ ⟹ a hpow x ∈
HFinite"
by (metis DiffE HFinite_hpow1 HFinite_hpow2 linorder_not_less)

lemma HFinite_hpow_not_Infinitesimal:
```

```
  assumes "a ∈ HFinite"
  and "x ∈ HFinite"
  and "1 ≤ a"
shows "a hpow x ∉ Infinitesimal"
proof
  have "a hpow x ≠ 0"
    using assms(3) hpow_not_zero by auto
  moreover assume "a hpow x ∈ Infinitesimal"
  then have "inverse (a hpow x) ∈ HInfinite"
    using Infinitesimal_inverse_HInfinite calculation by blast
  then have "a hpow - x ∉ HFinite"
    using HFinite_not_HInfinite assms(3) hpow_minus by auto
  then show False
    using HFinite_hpow1 HFinite_minus_iff assms by blast
qed

lemma HFinite_hpow_HFinite_Diff_Infinitesimal:
  "⟦ a ∈ HFinite; x ∈ HFinite; a ≥ 1 ⟧ ⟹ a hpow x ∈ HFinite - Infinitesimal"
by (metis Diff_iff HFinite_hpow1 HFinite_hpow_not_Infinitesimal)

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"
  by (metis HFinite_not_Infinitesimal_inverse approx_inverse approx_reorient
hpow_minus inverse_inverse_eq)

lemma HNatInfinite_hpow_inverse_approx_one:
  assumes "a ∈ HFinite"
  and "a > 1"
  and "N ∈ HNatInfinite"
  shows "(a hpow (inverse(of_hypnat N))) ≈ 1"
proof -
  obtain u where u0: "u > 0" and lhs: "a hpow inverse (hypreal_of_hypnat
N) = 1 + u"
    using HNatInfinite_hpow_eq_Ex assms(2) assms(3) by blast
  then have "(a hpow inverse (hypreal_of_hypnat N)) pow N = (1 + u) pow
N"
    by simp
  also obtain x where "... = 1 + hypreal_of_hypnat N * u + x" and x0:
"x ≥ 0"
    by (metis add.commute hyperbinomial_expand_first_two_terms' less_imp_le
u0)
    then have "a = 1 + hypreal_of_hypnat N * u + x"
    using assms(2) assms(3) calculation hpow_inverse_of_hypnat_pow_cancel2
by auto
  then have "u ≤ (a - 1) / hypreal_of_hypnat N"
    using assms(2) assms(3)  u0 x0
    by (auto simp add: le_divide_eq zero_less_HNatInfinite)
  moreover have "(a - 1) / hypreal_of_hypnat N ∈ Infinitesimal"
```

```
    by (simp add: HFinite_diff Infinitesimal_HFinite_mult2 assms(1) assms(3)
divide_inverse)
  ultimately have "u ∈ Infinitesimal"
    using u0 Infinitesimal_interval2 Infinitesimal_zero
    by (metis less_imp_le)
  then show ?thesis
    using bex_Infinitesimal_iff2 lhs by blast
qed

lemma HNatInfinite_hpow_inverse_diff_one_Infinitesimal:
  "⟦ a ∈ HFinite; a > 1; N ∈ HNatInfinite ⟧ ⟹ (a hpow (inverse(of_hypnat
N))) - 1 ∈ Infinitesimal"
by (metis HNatInfinite_hpow_inverse_approx_one bex_Infinitesimal_iff)

lemma HNatInfinite_hpow_inverse_approx_one2:
  "⟦ a ∈ HFinite; a > 1; N ∈ HNatInfinite ⟧ ⟹ (a hpow (1/of_hypnat N))
≈ 1"
by (simp add: divide_inverse HNatInfinite_hpow_inverse_approx_one)
```

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 -
  have gt0: "0 < hypnat (hfloor (1 / e))"
    using e_gt0 Infinitesimal_e hypreal_HNatInfinite_hfloor_inverse_Infinitesimal
zero_less_HNatInfinite
    by blast
  have ele: "e ≤ 1/(of_hypint (hfloor(1/e)))"
    by (metis (no_types, lifting) div_by_1 divide_pos_pos e_gt0 gt0 hfloor_correct
inverse_divide
        inverse_le_iff_le of_hypint_0_less_iff zero_less_hypnat_eq zero_less_one)
  have "inverse(of_hypint (hfloor(1/e)) + 1) < e"
    by (simp add: e_gt0 hfloor_less_cancel inverse_eq_divide inverse_less_imp_less)

   then have "a hpow inverse (hypreal_of_hypnat (hypnat (hfloor (1 /
e)) + 1)) - 1 ≤ a hpow e - 1"
     by (simp add: a_gt1 e_gt0 less_imp_le)
   moreover have "a hpow e - 1 ≤ a hpow inverse (hypreal_of_hypnat (hypnat
(hfloor (1 / e)))) - 1"
     using assms gt0 ele
     by (simp add: inverse_eq_divide)
  moreover have "a hpow inverse (hypreal_of_hypnat (hypnat (hfloor (1
/ e)))) - 1 ∈ Infinitesimal"
    using HNatInfinite_hpow_inverse_diff_one_Infinitesimal assms
```

```
        hypreal_HNatInfinite_hfloor_inverse_Infinitesimal
    by blast
  ultimately have "a hpow e - 1 ∈ Infinitesimal"
    by (meson Infinitesimal_interval2 Infinitesimal_zero a_gt1 e_gt0 diff_ge_0_iff_ge

        hpow_ge_one less_imp_le)
  then show ?thesis
    using bex_Infinitesimal_iff
    by blast
qed

lemma Infinitesimal_neg_hpow_approx_one:
   "⟦ a ∈ HFinite; a > 1; e > 0; e ∈ Infinitesimal⟧ ⟹ a hpow -e ≈ 1"
  using Infinitesimal_pos_hpow_approx_one approx_inverse hpow_minus by
fastforce
```

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 -
  have "e < 0 ∨ e = 0 ∨ 0 < e"
    by (simp add: less_linear)
  then show ?thesis
  proof (safe)
    assume "e < 0"
    then show ?thesis
      using Infinitesimal_neg_hpow_approx_one assms
      by force
  next
    show "a hpow 0 ≈ 1"
      using assms(2) by auto
  next
    assume "e > 0"
    then show ?thesis
      using Infinitesimal_pos_hpow_approx_one assms
      by force
  qed
qed

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 (cases "a = 1")
  assume "a = 1"
```

```
  then show ?thesis
     by simp
next
  assume "a ≠ 1"
  then have a_gt_1: "a > 1"
    using a_ge_1 by simp
  obtain n where xn: "of_nat n ≤ x" and  xn1: "x < of_nat (n + 1)"
    using HFinite_between_Nats x_finite x_gt_0
    by (metis (no_types, lifting) Nats_cases of_nat_1 of_nat_add)
  then have "a ^ n ≤ a hpow x"
    by (metis a_ge_1 a_gt_1 hpow_le_mono hpow_power_eq less_trans zero_less_one)
  moreover have "a hpow x < a ^ (n + 1)"
    by (metis a_ge_1 a_gt_1 hpow_less_mono hpow_power_eq less_le_trans
xn1 zero_less_one)
  moreover have "a ^ n ≈ 1"
    by (simp add: a_infclose_1 hrealpow_approx_one)
  moreover have "a ^ (n + 1) ≈ 1"
    using a_infclose_1 hrealpow_approx_one by blast
  ultimately show ?thesis
    by (meson approx_le_bound approx_sym approx_trans less_imp_le)
qed

lemma approx_hpow:
  assumes "y ∈ HFinite" and "a ∈ HFinite"
      and "a > 1" and "x ≈ y"
    shows "a hpow x ≈ a hpow y"
proof -
  have "x - y ∈ Infinitesimal"
    using approx_def assms(4) by auto
  then have "a hpow (x - y) ≈ 1"
    using Infinitesimal_hpow_approx_one assms(2) assms(3) by blast
  then have axy_approx_1: "a hpow x / a hpow y ≈ 1"
    using assms(3) hpow_divide2 by auto
   have "a hpow y ∈ HFinite"
    by (simp add: HFinite_hpow1 assms(1,2,3) less_imp_le)
  then show ?thesis
    using approx_mult2 [OF axy_approx_1] hpow_not_zero assms(3)
    by fastforce
qed

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))"
by *(metis hLog_def starfun2_star_n)*

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

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

lemma *hLog_mult:*
*"⋀a x y. ⟦ 0 < a; a ≠ 1; 0 < x; 0 < y ⟧ ⟹ hLog a (x * y) = hLog a x + hLog a y"*
by *(transfer, rule Log_mult)*

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

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

lemma *hLog_inverse:*
"⋀a x. ⟦ a > 0; a ≠ 1; x > 0 ⟧ ⟹ hLog a (inverse x) = - hLog a x"
by *(transfer, rule Log_inverse)*

lemma *hLog_divide:*
"⋀a x y. ⟦ 0 < a; a ≠ 1; 0 < x; 0 < y ⟧ ⟹ hLog a (x/y) = hLog a x - hLog a y"
by *(transfer, rule Log_divide)*

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

lemma *hLog_inj:* **assumes** *"1 < b"* **shows** *"inj_on (hLog b) {0 <..}"*
proof *(rule inj_onI, simp)*
fix *x y* **assume** *pos: "0 < x" "0 < y"* **and** *∗: "hLog b x = hLog b y"*
show *"x = y"*
proof *(cases rule: linorder_cases)*
assume *"x < y"* **hence** *"hLog b x < hLog b y"*

```
      using hLog_less_cancel_iff[OF `1 < b`] pos by simp
    thus ?thesis using * by simp
  next
    assume "y < x" hence "hLog b y < hLog b x"
      using hLog_less_cancel_iff[OF `1 < b`] pos by simp
    thus ?thesis using * by simp
  qed simp
qed

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

lemma zero_less_hLog_cancel_iff [simp]:
  "⋀a x. 1 < a ⟹ 0 < x ⟹ 0 < hLog a x ⟷ 1 < x"
by transfer simp

lemma zero_le_hLog_cancel_iff[simp]: "1 < a ⟹ 0 < x ⟹ 0 ≤ hLog a
x ⟷ 1 ≤ x"
  using hLog_le_cancel_iff[of a 1 x] by simp

lemma hLog_less_zero_cancel_iff[simp]: "1 < a ⟹ 0 < x⟹ hLog a x
< 0 ⟷ x < 1"
  using hLog_less_cancel_iff[of a x 1] by simp

lemma hLog_le_zero_cancel_iff[simp]: "1 < a ⟹ 0 < x ⟹ hLog a x ≤
0 ⟷ x ≤ 1"
  using hLog_le_cancel_iff[of a x 1] by simp

lemma one_less_hLog_cancel_iff[simp]: "1 < a ⟹ 0 < x ==> 1 < hLog a
x ⟷ a < x"
  using hLog_less_cancel_iff[of a a x] by simp

lemma one_le_hLog_cancel_iff[simp]: "1 < a ==> 0 < x ⟹ 1 ≤ hLog a
x ⟷ a ≤ x"
  using hLog_le_cancel_iff[of a a x] by simp

lemma hLog_less_one_cancel_iff[simp]: "1 < a ⟹ 0 < x ⟹ hLog a x
< 1 ⟷ x < a"
  using hLog_less_cancel_iff[of a x a] by simp

lemma hLog_le_one_cancel_iff[simp]: "1 < a ⟹ 0 < x ⟹ hLog a x ≤
1 ⟷ x ≤ a"
  using hLog_le_cancel_iff[of a x a] by simp

lemma hLog_hpow: "⋀b x y. ⟦ 0 < x; 1 < b; b ≠ 1 ⟧ ⟹ hLog b (x hpow
y) = y * hLog b x"
by (transfer, rule Log_powreal)
```

```
lemma hLog_nat_power:
      "⋀b x n. ⟦ 0 < x; 1 < b; b ≠ 1 ⟧ ⟹ hLog b (x pow n) = of_hypnat
n * hLog b x"
by (transfer, simp add: Log_nat_power )

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 -
  have agt0: "a > 0"
    using base_greater_1 less_trans zero_less_one by blast
  let ?k = "(a hpow e - 1)/e"
  have "(a hpow e) pow hypnat(hfloor(1/e)) = (1 + ?k * e) pow hypnat(hfloor(1/e))"
    by (simp add: exponent_gt_zero neq_iff)
  then
  obtain x
    where xprops: "x ≥ 0"
                  "a hpow (e * of_hypnat(hypnat(hfloor(1/e)))) =
                   1 + of_hypnat(hypnat(hfloor(1/e))) * (?k * e) + x"

    using hyperbinomial_expand_first_two_terms hpow_hyperpow_eq_hpow
          [symmetric] base_greater_1 exponent_gt_zero agt0
    by (metis diff_ge_0_iff_ge divide_nonneg_pos hpow_ge_one mult_nonneg_nonneg

        order.strict_implies_order)
  then have "a ≈ 1 + of_hypnat(hypnat(hfloor(1/e))) * (?k * e) + x"
    using approx_hpow base_finite base_greater_1 exponent_gt_zero
          Infinitesimal_hfloor_inverse_mult_self_pos
    by (metis HFinite_1 agt0 approx_sym divide_pos_pos hpow_one infinitesimal_exponent
```

```
      of_hypnat_hypnat_of_hypint zero_less_one less_le)
  then have "1 + of_hypnat(hypnat(hfloor(1/e))) * (?k * e) + x ∈ HFinite"
    using approx_HFinite base_finite
    by blast
  moreover have "of_hypnat(hypnat(hfloor(1/e))) * ?k * e ≥ 0"
    by (simp add: base_greater_1 exponent_gt_zero hpow_ge_one less_imp_le)
  ultimately have "of_hypnat(hypnat(hfloor(1/e))) * (?k * e) ∈ HFinite"

    using HFinite_add_imp_HFinite xprops(1) zero_le_one
    by (metis (no_types, lifting) add.commute add_nonneg_nonneg mult.assoc)

  then show "?k ∈ HFinite"
    using Infinitesimal_hfloor_inverse_mult_self_pos approx_1_mult_HFinite_HFinite

      infinitesimal_exponent exponent_gt_zero
      of_hypnat_hypnat_of_hypint
    by (metis (no_types, lifting) divide_pos_pos mult.assoc mult.commute
zero_less_one less_le)
qed
```

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 -
  have "(a hpow e - 1)/e ∈ HFinite" and "(a hpow e - 1)/e > 0"
    using Introductio_114_HFinite_witness assms hpow_gt_one
    by auto
  then show ?thesis
    using exponent_gt_zero le_add_diff_inverse
    by auto
qed
```

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 -
  obtain k where ks: "k ∈ HFinite" "k > 0" "a hpow -e = 1 + k * -e"
    using Introductio_114_pos assms
    by (meson Infinitesimal_minus_iff neg_0_less_iff_less)
  then have "(a hpow e) * (a hpow -e) = (1 + k * -e) * (a hpow e)"
```

```
    by simp
  then have "1 = a hpow e + (k * a hpow e) * -e"
    using base_greater_1 hpow_add[symmetric] hpow_zero_eq_one
    by (simp add: algebra_simps )
  then have a_hpow_e: "a hpow e = 1 + (k * (a hpow e)) * e"
    by linarith
  have HFinite_k_mult: "k * (a hpow e) ∈ HFinite"
    by (meson HFinite_0 HFinite_hpow1 HFinite_mult Infinitesimal_approx

        approx_HFinite base_finite base_greater_1 infinitesimal_exponent
ks(1)
        le_less not_Infinitesimal_not_zero)
  have "k * (a hpow e) > 0" using base_greater_1 hpow_gt_zero ks(2)
    by auto
  thus ?thesis using a_hpow_e HFinite_k_mult
    by blast
qed

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 -
  have e_ne_0: "e ≠ 0"
    using exponent_less_zero
    by simp
  obtain k where ks: "k ∈ HFinite" "k > 0" "a hpow e = 1 + k * e"
    using Introductio_114_neg assms
    by blast
  from ks(3) have "a hpow e - 1 = k * e"
    by simp
  then have "(a hpow e - 1)/e = k * e / e"
    by simp
  also have "... = k" using e_ne_0
    by simp
  finally show ?thesis using ks(1)
    by simp
qed

lemma Introductio_114_HFinite_witness_full:
      assumes "x ∈ Infinitesimal"
      and "a ∈ HFinite"
      and "a > 1"
    shows "(a hpow x - 1)/x ∈ HFinite"
  by (metis assms HFinite_0 Introductio_114_HFinite_witness Introductio_114_HFinite_witness

      divide_eq_0_iff linorder_neqE_linordered_idom)
```

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 -
  have ane0: "a hpow e ≠ 0"
    using assms(1) hpow_not_zero by blast
  have "eulerK a (-e) = (inverse (a hpow e) - 1) / (-e)"
    using assms(1) hpow_minus by (simp add: eulerK_def)
  also have "... = ((1 - a hpow e) / (a hpow e)) / (-e)"
    using ane0 by (simp add: inverse_eq_divide diff_divide_distrib)
  also have "... = (a hpow e - 1) / (a hpow e * e)"
    by (simp add: diff_divide_distrib)
  also have "... = ((a hpow e - 1) / e) / (a hpow e)"
    using ane0 by (simp add: field_simps)
  finally show ?thesis by (simp add: eulerK_def)
qed
```

Equivalent multiplicative form.

```
corollary eulerK_neg_exponent_mult:
  assumes "a > 0" and "e ≠ 0"
  shows "eulerK a (-e) * (a hpow e) = eulerK a e"
  using eulerK_neg_exponent assms hpow_not_zero by auto

lemma eulerK_neg_approx:
  assumes "a ∈ HFinite" and "a > 1" and "x ∈ Infinitesimal"
  shows "eulerK a (-x) ≈ eulerK a x"
proof -
  have "a > 0" using assms(2) by auto
  moreover have "eulerK a (-x) = eulerK a x / (a hpow x)"
    by (metis calculation add.inverse_neutral div_by_1 eulerK_neg_exponent
hpow_zero_eq_one)
  moreover have "a hpow x ≈ 1"
    using Infinitesimal_hpow_approx_one assms by blast
  moreover have "eulerK a x ∈ HFinite"
    by (simp add: Introductio_114_HFinite_witness_full assms eulerK_def)
  moreover have "a hpow x ∈ HFinite - Infinitesimal"
    by (meson HFinite_hpow_HFinite_Diff_Infinitesimal Infinitesimal_subset_HFinite
assms order_less_le
        subsetD)
  ultimately show ?thesis
```

```
    by (metis DiffE HFinite_divide Infinitesimal_zero approx_mult2 approx_reorient
        divide_eq_0_iff eulerK_def eulerK_neg_exponent_mult mem_infmal_iff
        mult.right_neutral)
qed

corollary eulerK_epsilon_neg_approx:
  assumes "a ∈ HFinite" and "a > 1"
  shows "eulerK a (-ε) ≈ eulerK a ε"
  by (simp add: assms  epsilon_not_zero eulerK_neg_approx)

lemma Introductio_114_eulerK_witness:
  assumes "x ∈ Infinitesimal"
  and "a ∈ HFinite"
  and "a > 1"
  shows "eulerK a x ∈ HFinite"
  using assms Introductio_114_HFinite_witness_full eulerK_def
  by auto

lemma HFinite_eulerK_inverse_whn:
  assumes "a ∈ HFinite"
  and "a > 1"
  shows "eulerK a (1/of_hypnat whn) ∈ HFinite"
  by (metis assms HNatInfinite_inverse_Infinitesimal HNatInfinite_whn
Introductio_114_HFinite_witness_full
      eulerK_def inverse_eq_divide)

lemma Introductio_114_eulerK_epsilon_witness:
      assumes "a ∈ HFinite"
      and     "a > 1"
      shows "eulerK a ε ∈ HFinite"
  using assms Introductio_114_HFinite_witness_full
  by (simp add: eulerK_def)

lemma eulerK_neg_e_HFinite:
  assumes "e ∈ Infinitesimal" "e > 0" "a ∈ HFinite" "a > 1"
  shows "eulerK a (-e) ∈ HFinite"
  by (metis Infinitesimal_minus_iff Introductio_114_HFinite_witness_full
assms(1,3,4) eulerK_def)

lemma Introductio_114_neg_eulerK_epsilon_witness:
  assumes "a ∈ HFinite" "a > 1"
  shows "eulerK a (-ε) ∈ HFinite"
  by (simp add: assms  epsilon_gt_zero eulerK_neg_e_HFinite)

lemma eulerK_neg_e_gt_zero:
  assumes "e ∈ Infinitesimal" "e > 0" "a ∈ HFinite" "a > 1"
  shows "eulerK a (-e) > 0"
```

```
proof -
  have a_gt_0: "a > 0" using assms(4) by auto
  have "a hpow e > 1" using assms(4) assms(2) hpow_gt_one by blast
  then have "a hpow (-e) < 1"
    using a_gt_0 hpow_minus by (simp add: inverse_less_1_iff hpow_gt_zero)
  then have "a hpow (-e) - 1 < 0" by simp
  moreover have "-e < 0" using assms(2) by simp
  ultimately show ?thesis
    by (metis eulerK_def zero_less_divide_iff)
qed

lemma eulerK_neg_epsilon_gt_zero:
  assumes "a ∈ HFinite" "a > 1"
  shows "eulerK a (-ε) > 0"
  by (simp add: assms epsilon_gt_zero eulerK_neg_e_gt_zero)

lemma Introductio_114_eulerK:
  shows "a > 0 ⟹ a hpow e = 1 + eulerK a e * e"
  by (simp add: eulerK_def)

lemma Introductio_114_eulerK':
  shows "e ≠ 0 ⟹ a hpow e = 1 + eulerK a e * e"
  by (simp add: eulerK_def)

lemma Introductio_114_eulerK_epsilon:
  shows "a hpow ε = 1 + eulerK a ε * ε"
  by (simp add: epsilon_not_zero eulerK_def)

lemma Introductio_114_epsilon:
  "a > 1 ⟹ ε = hLog a (1 + eulerK a ε * ε)"
  using Introductio_114_eulerK_epsilon eulerK_def by fastforce

lemma eulerK_product_neg_e:
  assumes "a > 0" and "e ≠ 0"
  shows "eulerK a (-e) * (-e) = eulerK a e * (-e) / (a hpow e)"
proof -
  have "eulerK a (-e) = eulerK a e / (a hpow e)"
    using eulerK_neg_exponent assms by blast
  then show ?thesis by (simp add: field_simps)
qed

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 -
  have "(eulerK a (-e) * (-e)) pow of_nat n = (eulerK a e * (-e) / (a
hpow e)) pow of_nat n"
    using eulerK_product_neg_e assms by simp
```

```
  also have "... = (eulerK a e * (-e)) pow of_nat n / (a hpow e) pow of_nat
n"
    using hyperpow_divide by blast
  also have "... = (eulerK a e * (-e)) pow of_nat n / (a hpow e) ^ n"
    using hyperpow_of_nat by auto
  finally show ?thesis .
qed
```

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}"
  using Introductio_114_eulerK hyperbinomial_simple by auto

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}"
  by (simp add: Introductio_114_eulerK hyperbinomial_hfallfactpow hyperbinomial_simple)

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}"
  using Introductio_114_eulerK' hyperbinomial_simple by auto

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}"
  using Introductio_114_eulerK_epsilon hyperbinomial_simple by auto


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 -
  have hpow_pow: "(a hpow e) hpow of_hypint(hfloor(z/e)) = a hpow (e *
of_hypint (hfloor(z/e)))"
    using base_greater_1 hpow_hyperpow_eq_hpow less_trans zero_less_one
hpow_mult
    by auto
  have "(e * of_hypint (hfloor(z/e))) ≈ z"
    using  Infinitesimal_hfloor_divide_mult exponent_not_zero
```

```
          infinitesimal_exponent  by simp
  then have "a hpow (e * of_hypint (hfloor(z/e))) ≈ a hpow z"
    using approx_hpow base_finite base_greater_1 exponentz_HFinite by
blast
  thus ?thesis by (simp add: hpow_pow)
qed

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 ≈ (∑h i∈{0..N}. hypreal_of_hypnat (N hchoose i)
* (eulerK a e * e) pow i)"
proof -
  have "(a hpow e) pow N =
          (∑h i∈{0..N}. of_hypnat (N hchoose i) * (eulerK a e * e)
pow i)"
  using Introductio_115_expansion_hchoose assms(4) by fastforce
  then show ?thesis
    using assms Infinitesimal_hfloor_divide_mult hpow_hyperpow_eq_hpow
approx_hpow
    unfolding N_def
    by (metis (lifting) approx_sym divide_pos_pos dual_order.strict_trans
of_hypnat_hypnat_of_hypint order_less_le
        zero_less_one)
qed

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)"
  by (metis (lifting) hpow_hypersum_pos_hchoose assms ext hyperbinomial_hfallfactpow)

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)"
  using hpow_hypersum_pos_hchoose Infinitesimal_epsilon assms epsilon_gt_zero
  by blast

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 -
    have "hfallfactpow (hypreal_of_hypnat N) n ≤ (of_hypnat N) pow n"

      by (metis hfallfactpow_le_hyperpow hfallfactpow_of_nat_eq_0_lemma

          hyperpow_ge_zero not_le_imp_less of_hypnat_0_le_iff of_hypnat_less_iff)
    then have le1:
          "e pow n * hfallfactpow (of_hypnat N) n ≤ e pow n *  (of_hypnat
N) pow n"
      by (simp add: e_gt_zero hyperpow_gt_zero)
    have hfloor_gt_0: "of_hypnat N > (0::hypreal)"
       by (simp add: field_simps N_def e_gt_zero,
           meson DiffD2 Infinitesimal_interval Infinitesimal_zero
           exponentz_HFinite exponentz_gt_zero infinitesimal_e not_le_imp_less)

     have "e * of_hypnat N ≤ z"
       unfolding N_def
      by (metis  divide_nonneg_nonneg  e_gt_zero exponentz_gt_zero
          leD less_imp_le mult.commute mult_imp_div_pos_less
          not_less of_hypint_floor_le of_hypnat_hypnat zero_le_hfloor)

     then have  "(e * of_hypnat N) pow n ≤ z pow n"
      using e_gt_zero hfloor_gt_0 hyperpow_le mult_pos_pos by blast
    thus ?thesis
      using le1 unfolding N_def
      by (metis (no_types, lifting) hyperpow_mult order_trans)
qed

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 -
  have internalf_summand:
          "(λi. hfallfactpow (of_hypnat N) i  / hfact i * (k * e) pow
i) ∈ InternalFuns"
```

```
    by (simp add: InternalFuns_divide  InternalFuns_mult InternalFuns_of_hypnat)
  have internalf_compare: "(λi. (k * z) pow i / hfact i) ∈ InternalFuns"
    by (simp add: InternalFuns_divide)
  {fix n :: "nat star" have
            "0 ≤ hfallfactpow (of_hypnat N) n / hfact n * (k * e) pow
n"
       by (meson divide_nonneg_pos e_gt_zero hfact_gt_zero hfallfactpow_ge_zero

           hyperpow_ge_zero k_gt_zero mult_nonneg_nonneg order_less_le)
   }
   then have summand_ge_0:
            "∀n. 0 ≤ hfallfactpow (of_hypnat N) n / hfact n * (k * e)
pow n"
     by blast
   have hypersummable_compare: "HyperSummable (λi. (k * z) pow i / hfact
i)"
     using HFinite_mult HyperSummable_exp exponentz_HFinite k_HFinite
by auto

   have  "∃k'∈ℕ. ∀n≥k'.
             hfallfactpow (of_hypnat N) n / hfact n * (k * e) pow n
             ≤ (k * z) pow n / hfact n"
   proof -
     {fix n :: "nat star" assume n_gt_0: "n ≥ 0"
       then have  "hfallfactpow (hypreal_of_hypnat N) n * e pow n ≤ z
pow n"
         by (metis N_def Infinitesimal_pow_hfallfactpow_le_pow e_gt_zero
exponentz_HFinite
             exponentz_gt_zero infinitesimal_e mult.commute)
       then have  "hfallfactpow (hypreal_of_hypnat N) n * (k * e) pow
n  ≤ (k * z) pow n"
         by (simp add: hyperpow_gt_zero hyperpow_mult k_gt_zero)
       }
       thus ?thesis using N_def Nats_0 divide_right_mono
         by (metis hfact_gt_zero order_less_le times_divide_eq_left)
   qed
   thus ?thesis using internalf_compare internalf_summand hypersummable_compare
summand_ge_0
                      HyperSummable_hypreal_comparison_test
     by (metis (no_types, lifting) times_divide_eq_left)
qed
```

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 -
  have internalf_summand:
         "(λi. hfallfactpow (of_hypnat N) i / hfact i * (k * -e) pow
i) ∈ InternalFuns"
    by (simp add: InternalFuns_divide  InternalFuns_mult InternalFuns_of_hypnat)
  have internalf_compare:
         "(λi. hfallfactpow (of_hypnat N) i / hfact i * (k * e) pow i)
∈ InternalFuns"
    by (simp add: InternalFuns_divide  InternalFuns_mult InternalFuns_of_hypnat)
  have zs: "- z ∈ HFinite - Infinitesimal" "-z > 0"
    using HFinite_minus_iff  Diff_iff Infinitesimal_minus_iff exponentz_HFinite
exponentz_less_zero
    by auto
    have le_terms:
         "∃ka∈ℕ. ∀n≥ka.
              hnorm (hfallfactpow (of_hypnat N) n / hfact n * (k * -
e) pow n)
              ≤ hfallfactpow (of_hypnat N) n / hfact n * (k * e) pow
n"
    proof -
    {fix n :: "nat star" assume n_gt_0: "n ≥ 0"
     then have "|hfallfactpow (of_hypnat N) n *  (k * - e) pow n|
                ≤  hfallfactpow (of_hypnat N) n * (k * e) pow n"
     proof (cases "hfallfactpow (of_hypnat N) n = (0::hypreal)")
       assume "hfallfactpow (of_hypnat N) n = (0::hypreal)"
       then show ?thesis by (simp add: N_def)
     next
       assume "hfallfactpow (of_hypnat N) n ≠ (0::hypreal)"
       then have "hfallfactpow (of_hypnat N) n > (0::hypreal)"
         by (simp add: order.not_eq_order_implies_strict N_def)
       then show ?thesis
         by (metis (no_types, lifting) abs_mult abs_of_nonneg e_gt_zero

             hrabs_hyperpow_minus hyperpow_ge_zero k_gt_zero le_less mult_minus_right

             mult_nonneg_nonneg)
     qed
     then have "hnorm(hfallfactpow (of_hypnat N) n *  (k * - e) pow n)
                ≤  hfallfactpow (of_hypnat N) n * (k * e) pow n"
       by simp
   }
   thus ?thesis
     using Nats_0 divide_right_mono of_hypnat_0_le_iff
     by (metis (no_types, lifting) hnorm_divide hnorm_hypreal_of_hypnat
```

```
        of_hypnat_hfact times_divide_eq_left)
  qed
  then show ?thesis
    using HyperSummable_comparison_test [OF internalf_summand internalf_compare]

          HyperSummable_exp_terms zs infinitesimal_e e_gt_zero k_HFinite
k_gt_zero
    unfolding N_def
    by blast
qed

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 -
  let ?k = "eulerK a e"
  have eq: "(λi. hypreal_of_hypnat (N hchoose i) * (?k * -e) pow i) =
            (λi. hfallfactpow (hypreal_of_hypnat N) i / hfact i *
                 (?k * -e) pow i)"
    by (simp add: hyperbinomial_hfallfactpow)
  moreover have "eulerK a e ∈ HFinite"
    by (simp add: Introductio_114_eulerK_witness base_finite base_greater_1
infinitesimal_e)
  moreover have "eulerK a e > 0"
    by (simp add: base_greater_1 eulerK_def hpow_gt_one infitesimal_e_gt_zero)
  ultimately show ?thesis
    unfolding N_def
    by (metis (no_types, lifting) ext HyperSummable_exp_terms2 exponentz_HFinite
exponentz_less_zero
        hyperbinomial_hfallfactpow infinitesimal_e infitesimal_e_gt_zero)
qed

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 -
  let ?k = "eulerK a (-e)"
  have k_HFinite: "?k ∈ HFinite"
    by (simp add: Introductio_114_eulerK_witness base_finite base_greater_1
infinitesimal_e)
  moreover have k_gt_zero: "?k > 0"
    by (simp add: base_finite base_greater_1 eulerK_neg_e_gt_zero infinitesimal_e
infitesima_e_gt_zero)
  moreover have eq:
          "(λi. of_hypnat (N hchoose i) * (?k * -e) pow i) =
           (λi. hfallfactpow (of_hypnat N) i / hfact i * (?k * -e) pow
i)"
    by (rule ext, simp add: hyperbinomial_hfallfactpow)
  ultimately show ?thesis
    using HyperSummable_exp_terms2 exponentz_HFinite exponentz_less_zero
infinitesimal_e
          infitesima_e_gt_zero
    unfolding N_def
    by presburger
qed

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 -
  have ze_pos: "- z/e ≥ 0"
    by (simp add: divide_nonpos_pos exponentz_less_zero infinitesimal_e_gt_zero
order_less_imp_le)
   have "(a hpow -e) pow N =
        hypersum (λi. of_hypnat (N hchoose i) * (eulerK a (-e) * -e) pow
i) {0..N}"
     using Introductio_115_expansion_hchoose' infinitesimal_e_gt_zero

     by force
   moreover have "a hpow (-e * of_hypnat N) ≈ a hpow z"
     unfolding N_def
     by (metis Infinitesimal_hfloor_divide_mult approx_hpow approx_minus_cancel
base_finite
         base_greater_1 exponentz_HFinite infinitesimal_e infinitesimal_e_gt_zero
mult_minus_left
```

```
        of_hypnat_hypnat_of_hypint order_less_le verit_minus_simplify(4)
ze_pos)
    ultimately show ?thesis
      by (metis approx_sym base_greater_1 dual_order.strict_trans hpow_hyperpow_eq_hpow
zero_less_one)
  qed

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 -
  have "z/e ∈ HInfinite"
      using infinitesimal_e z_HFinite_not_Infinitesimal e_gt_0
        by (metis DiffD2 HInfinite_HFinite_disj Infinitesimal_ratio less_numeral_extra(3))

      then have HNatInfinite_ze: "hypnat(hfloor(z/e)) ∈ HNatInfinite"

        using divide_pos_pos e_gt_0 hypreal_HNatInfinite_hfloor z_gt_0
        by blast
      then have hfall_div_pow_1: "(hfallfactpow (of_hypnat(hypnat(hfloor(z/e))))
(of_nat i)) /
                (hypreal_of_hypnat(hypnat(hfloor(z/e))) pow of_nat i)
≈ 1"
        using HNatInfinite_hfallfactpow_divide_hyperpow_of_nat
        by blast
      have z_pow_HFinite: "z pow of_nat i ∈ HFinite"
          by (metis DiffE hrealpow_HFinite hyperpow_of_nat z_HFinite_not_Infinitesimal)

      have "(e * of_hypnat(hypnat(hfloor(z/e)))) pow of_nat i ≈ z pow
of_nat i"
        by (metis DiffE HNatInfinite_of_hypnat_gt_zero HNatInfinite_ze
Infinitesimal_hfloor_divide_mult
            division_ring_divide_zero hfloor_zero hrealpow_approx hyperpow_of_nat
hypint_hypnat_eq
            infinitesimal_e less_numeral_extra(3) of_hypnat_hypnat z_HFinite_not_Infinitesi
      then have  "(hfallfactpow (of_hypnat(hypnat(hfloor(z/e)))) (of_nat
i)) /
                (hypreal_of_hypnat(hypnat(hfloor(z/e))) pow of_nat i)
*
             (e * of_hypnat(hypnat(hfloor(z/e)))) pow of_nat i ≈ z pow
of_nat i"
        by (metis HFinite_1 approx_mult_HFinite hfall_div_pow_1 mult.left_neutral
z_pow_HFinite)
      thus ?thesis
        using HNatInfinite_ze
```

```
        by (metis (no_types, lifting) hyperpow_divide nonzero_mult_div_cancel_right
            of_hypnat_eq_0_iff of_nat_0 of_nat_eq_star_of star_of_neq_HNatInfinite
            times_divide_eq_left times_divide_eq_right approx_sym)
qed

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)"
  using assms hyperpow_nat_approx Nats_hypnat_of_nat_iff by auto

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 -
  have "z/e ∈ HInfinite"
    using infinitesimal_e z_HFinite_not_Infinitesimal e_gt_0
    by (metis DiffD2 HInfinite_HFinite_disj Infinitesimal_ratio less_numeral_extra(3))

      then have HNatInfinite_ze: "hypnat(hfloor(-z/e)) ∈ HNatInfinite"
        by (simp add: HInfinite_minus_iff divide_neg_pos e_gt_0 hypreal_HNatInfinite_hfloor
z_less_0)
      then have hfall_div_pow_1: "(hfallfactpow (of_hypnat(hypnat(hfloor(-z/e))))
(of_nat i)) /
                (hypreal_of_hypnat(hypnat(hfloor(-z/e))) pow of_nat i)
≈ 1"
        using HNatInfinite_hfallfactpow_divide_hyperpow_of_nat
        by blast
      have z_pow_HFinite: "z pow of_nat i ∈ HFinite"
        by (metis DiffE hrealpow_HFinite hyperpow_of_nat z_HFinite_not_Infinitesimal)

      have "- e * hypreal_of_hypnat (hypnat (hfloor(-z/e))) ≈ z"
        using Infinitesimal_hfloor_divide_mult z_less_0 infinitesimal_e
e_gt_0
        by (metis HNatInfinite_of_hypnat_gt_zero HNatInfinite_ze add.inverse_inverse
approx_minus
            hypint_hypnat_eq linorder_neq_iff mult_minus_left of_hypint_of_hypnat)
      then have "(-e * of_hypnat(hypnat(hfloor(-z/e)))) pow of_nat i
≈ z pow of_nat i"
```

```
        by (meson DiffD1 approx_sym hyperpow_approx z_HFinite_not_Infinitesimal)
      then have  "(hfallfactpow (of_hypnat(hypnat(hfloor(-z/e)))) (of_nat
i)) /
                (hypreal_of_hypnat(hypnat(hfloor(-z/e))) pow of_nat i)
*
             (-e * of_hypnat(hypnat(hfloor(-z/e)))) pow of_nat i ≈ z pow
of_nat i"
        by (metis HFinite_1 approx_mult_HFinite hfall_div_pow_1 mult.left_neutral
z_pow_HFinite)
      thus ?thesis using HNatInfinite_ze
        by (metis (no_types, lifting) hyperpow_divide nonzero_mult_div_cancel_right

            of_hypnat_eq_0_iff of_nat_0 of_nat_eq_star_of star_of_neq_HNatInfinite

            times_divide_eq_left times_divide_eq_right approx_sym)
qed

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)"
       using hyperpow_nat_approx HFinite_k
proof -
  have "k pow of_nat i ∈ HFinite"
    by (metis (no_types) HFinite_k hrealpow_HFinite hyperpow_of_nat)
  then have "k pow of_nat i * z pow of_nat i ≈
             k pow of_nat i * (hfallfactpow (hypreal_of_hypnat (hypnat
(hfloor(z / e)))) (of_nat i) * e pow of_nat i)"
    by (meson hyperpow_nat_approx approx_mult_subst approx_refl e_gt_0
infinitesimal_e z_HFinite_not_Infinitesimal z_gt_0)
  then show ?thesis
    by (simp add: hyperpow_mult mult.left_commute)
qed

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)"
       using hyperpow_nat_approx HFinite_k
```

```
proof -
  have "k pow of_nat i ∈ HFinite"
    by (metis (no_types) HFinite_k hrealpow_HFinite hyperpow_of_nat)
  then have "k pow of_nat i * z pow of_nat i ≈
            k pow of_nat i * (hfallfactpow (hypreal_of_hypnat (hypnat
(hfloor(-z / e)))) (of_nat i) * (-e) pow of_nat i)"
    using HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_neg approx_mult2
e_gt_0 infinitesimal_e z_HFinite_not_Infinitesimal z_less_0 by blast
  then show ?thesis
    by (metis hyperpow_mult mult.left_commute)
qed

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)"
      using HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow2 HFinite_k
         by (metis HInfinite_diff_HFinite_Infinitesimal_disj Reals_of_nat

              SReal_Infinitesimal_zero approx_divide_HFinite_diff_Infinitesimal

              approx_divide_HInfinite e_gt_0 fact_nonzero infinitesimal_e
of_nat_fact
              times_divide_eq_left z_HFinite_not_Infinitesimal z_gt_0)


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)"
      using HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow2_neg
HFinite_k
         by (metis HInfinite_diff_HFinite_Infinitesimal_disj Reals_of_nat

              SReal_Infinitesimal_zero approx_divide_HFinite_diff_Infinitesimal

              approx_divide_HInfinite e_gt_0 fact_nonzero infinitesimal_e
of_nat_fact
              times_divide_eq_left z_HFinite_not_Infinitesimal z_less_0)
```

```
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 -
  let ?N = "hypnat (hfloor(-z/e))"
  have "hypreal_of_hypnat (?N hchoose n) * (k * -e) pow n =
        hfallfactpow (hypreal_of_hypnat ?N) n / hfact n * (k * -e) pow
n"
    by (simp add: hyperbinomial_hfallfactpow)
  also have "... ≈ (k * z) pow n / fact m"
    using HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_divide_neg
assms
    by (metis approx_sym hfact_of_nat of_nat_eq_star_of of_nat_fact)
  finally have approx_standard: "hypreal_of_hypnat (?N hchoose n) * (k
* -e) pow n ≈
                                 (k * z) pow n / fact m" .
  have "(k * z) pow n / fact m ∈ HFinite"
  proof -
    have "k * z ∈ HFinite" using assms(3,5) by (metis DiffD1 HFinite_mult)
    then have "(k * z) pow n ∈ HFinite"
      by (metis assms(7) hrealpow_HFinite hyperpow_of_nat)
    moreover have "(fact m :: hypreal) ≠ 0" by simp
    moreover have "(fact m :: hypreal) ∈ HFinite"
      by (metis HFinite_of_nat of_nat_fact)
    ultimately show ?thesis
      using HFinite_divide SReal_Infinitesimal_zero fact_in_Reals by blast
  qed
  then show ?thesis using approx_standard approx_HFinite
    using approx_sym by blast
qed
```

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
```

```
  fix n :: hypnat assume n_Nat: "n ∈ ℕ"
  then obtain m where n_eq: "n = of_nat m"
    using Nats_cases by blast

  let ?N = "hypnat (hfloor(-z/e))"
  let ?k1 = "eulerK a e"
  let ?k2 = "eulerK a (-e)"

  have a_gt_0: "a > 0" using base_greater_1 by auto

  — Both eulerK values are HFinite and positive, trivially
  have k1_HFinite: "?k1 ∈ HFinite"
    by (simp add: Infinitesimal_e Introductio_114_eulerK_witness base_finite
base_greater_1)
  have k1_gt_0: "?k1 > 0"
    by (simp add: Infinitesima_e_gt0 base_greater_1 eulerK_def hpow_gt_one)
  have k2_HFinite: "?k2 ∈ HFinite"
    by (simp add: Infinitesima_e_gt0 Infinitesimal_e base_finite base_greater_1
eulerK_neg_e_HFinite)
  have k2_gt_0: "?k2 > 0"
    by (simp add: Infinitesima_e_gt0 Infinitesimal_e base_finite base_greater_1
eulerK_neg_e_gt_zero)

  — Both sides approximate the same middle term via the falling factorial
  let ?mid1 = "(?k1 * z) pow n / fact m"
  let ?mid2 = "(?k2 * z) pow n / fact m"

  have lhs_approx: "hypreal_of_hypnat (?N hchoose n) * (?k1 * -e) pow
n ≈ ?mid1"
  proof -
    have "hypreal_of_hypnat (?N hchoose n) * (?k1 * -e) pow n =
          hfallfactpow (hypreal_of_hypnat ?N) n / hfact n * (?k1 * -e)
pow n"
      by (simp add: hyperbinomial_hfallfactpow)
    also have "... ≈ ?mid1"
      by (metis HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_divide_neg
Infinitesima_e_gt0
          Infinitesimal_e approx_reorient exponentz_HFinite exponentz_less_zero
hfact_of_nat
          k1_HFinite n_eq of_nat_eq_star_of of_nat_fact)
     finally show ?thesis .
  qed

  have rhs_approx: "hypreal_of_hypnat (?N hchoose n) * (?k2 * -e) pow
n ≈ ?mid2"
  proof -
    have "hypreal_of_hypnat (?N hchoose n) * (?k2 * -e) pow n =
          hfallfactpow (hypreal_of_hypnat ?N) n / hfact n * (?k2 * -e)
pow n"
```

```
    by (simp add: hyperbinomial_hfallfactpow)
  also have "... ≈ ?mid2"
    by (metis HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_divide_neg
Infinitesima_e_gt0
        Infinitesimal_e approx_reorient exponentz_HFinite exponentz_less_zero
hfact_of_nat
        k2_HFinite n_eq  of_nat_eq_star_of of_nat_fact)
  finally show ?thesis .
qed

— The two middle terms are infinitely close because for infinitesimal e, eulerK a
e is infinitely close to eulerK a (-e)
have k1_approx_k2: "?k1 ≈ ?k2"
  using Infinitesimal_e approx_sym base_finite base_greater_1 eulerK_neg_approx

  by auto

have k1z_approx_k2z: "?k1 * z ≈ ?k2 * z"
proof -
  have "z ∈ HFinite" using exponentz_HFinite
    by blast
  then show ?thesis using k1_approx_k2 approx_mult1
    by blast
qed

have k1z_HFinite: "?k1 * z ∈ HFinite"
  using k1_HFinite exponentz_HFinite
  by (metis DiffD1 HFinite_mult)

have mid_approx: "?mid1 ≈ ?mid2"
proof -
  have "(?k1 * z) pow n ≈ (?k2 * z) pow n"
    using k1z_approx_k2z k1z_HFinite n_eq
    by (metis approx_HFinite hrealpow_approx hyperpow_of_nat)
  then show ?thesis
    by (metis approx_divide_HFinite_diff_Infinitesimal fact_nonzero

        hypreal_of_real_HFinite_diff_Infinitesimal of_nat_fact star_of_nat)
qed

— Conclude by transitivity of infinitely close relation
show "hypreal_of_hypnat (?N hchoose n) * (?k1 * -e) pow n ≈
      hypreal_of_hypnat (?N hchoose n) * (?k2 * -e) pow n"
  using approx_trans2 lhs_approx mid_approx rhs_approx
  by blast
qed

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 (rule Hypersum_Comparison_Theorem')
  show "(λi. hypreal_of_hypnat (hypnat (hfloor (- z / e)) hchoose i)
* (eulerK a e * - e) pow i)
        ∈ InternalFuns"
    using InternalFuns_hyperpow_simple InternalFuns_mult InternalFuns_of_hypnat
InternalFuns_star_choose
    by blast
next
  show "(λi. hypreal_of_hypnat (hypnat (hfloor (- z / e)) hchoose i)
* (eulerK a (- e) * - e) pow i)
        ∈ InternalFuns"
    using InternalFuns_hyperpow_simple InternalFuns_mult InternalFuns_of_hypnat
InternalFuns_star_choose by blast
next
  show "HyperSummable
        (λi. hypreal_of_hypnat (hypnat (hfloor (- z / e)) hchoose i) *
(eulerK a e * - e) pow i)"
    using HyperSummable_binomial_neg_hchoose assms
    by blast
next
  show "HyperSummable
        (λi. hypreal_of_hypnat (hypnat (hfloor (- z / e)) hchoose i) *
(eulerK a (- e) * - e) pow i)"
    using HyperSummable_binomial_neg_hchoose' assms
    by blast
next
  show "∀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"
    using hchoose_approx_for_Nats assms
    by blast
qed

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 -
  have "hypersum
     (λi. hypreal_of_hypnat (hypnat (hfloor (- z / e)) hchoose i) *
          (eulerK a (- e) * - e) pow i) {0..hypnat (hfloor (- z / e))}
≈
         hypersum (λi. hypreal_of_hypnat (hypnat (hfloor (- z / e)) hchoose
i) *
          (eulerK a e * - e) pow i) {0..hypnat (hfloor (- z / e))}"
    using approx_sym assms hypersetsetsum_neg_approx
    by blast
  then show ?thesis
    using hpow_hypersum_neg_hchoose [where e=e] assms
    by (meson DiffE approx_trans)
qed
```

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 -
     have internalf_rhs_terms: "(λi. (eulerK a e * z) pow i / hfact i)
∈ InternalFuns"
       by (simp add: InternalFuns_divide)
     have hyperSummable_rhs: "HyperSummable (λi. (eulerK a e * z) pow
i / hfact i)"
       by (metis Diff_iff HFinite_mult HyperSummable_exp
           Introductio_114_eulerK_witness base_finite base_greater_1
           exponentz_HFinite infinitesimal_e)
     have "(λi. hypreal_of_hypnat (N hchoose i) * (eulerK a e * e) pow
i) ∈ InternalFuns"
       using InternalFuns_hyperpow_simple InternalFuns_mult InternalFuns_of_hypnat
InternalFuns_star_choose
       by blast
```

```
    then have internalf_lhs_terms: "(λi. hfallfactpow (hypreal_of_hypnat
N) i /
                                        hfact i * (eulerK a e * e) pow
i) ∈ InternalFuns"
      by (simp add: hyperbinomial_hfallfactpow [symmetric])

   {
     fix n :: "nat star" assume "n ∈ Nats"
     then obtain K where "n = hypnat_of_nat K"
       using Nats_hypnat_of_nat_iff by blast
     moreover have "hfallfactpow (hypreal_of_hypnat N) (hypnat_of_nat
K) *
                    (eulerK a e * e) pow hypnat_of_nat K ≈
                    (eulerK a e * z) pow hypnat_of_nat K"
       by (metis HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow2
Introductio_114_eulerK_witness
           N_def approx_reorient base_finite base_greater_1 e_gt_zero
exponentz_HFinite
           exponentz_gt_zero infinitesimal_e of_nat_eq_star_of)
     ultimately have "hfallfactpow (hypreal_of_hypnat (hypnat (hfloor(z/e))))
n / hfact n *
                      (eulerK a e * e) pow n ≈
                      (eulerK a e * z) pow n / hfact n"
       by (metis N_def Reals_of_nat SReal_HFinite_diff_Infinitesimal
approx_divide_HFinite_diff_Infinitesimal
           hfact_nonzero hfact_of_nat times_divide_eq_left)
    }
    then have approx_lhs_rhs:
              "∀n∈ℕ. hfallfactpow (hypreal_of_hypnat N) n /
                     hfact n * (eulerK a e * e) pow n ≈
                     (eulerK a e * z) pow n / hfact n"
       unfolding N_def by blast
    have "HyperSummable
            (λi. hfallfactpow (hypreal_of_hypnat N) i /
                 hfact i * (eulerK a e * e) pow i)"
      using HyperSummable_exp_terms Introductio_114_eulerK_witness N_def
      by (metis (no_types, lifting) ext base_finite base_greater_1 diff_gt_0_iff_gt
divide_pos_pos
          e_gt_zero eulerK_def exponentz_HFinite exponentz_gt_zero hpow_gt_one
infinitesimal_e)

    then have "hypersum (λi. hfallfactpow (of_hypnat N) i / hfact i
* (eulerK a e * e) pow i) {0..N} ≈
               hypersum (λi. (eulerK a e * z) pow i / hfact i) {0..N}"
        using Hypersum_Comparison_Theorem' approx_lhs_rhs hyperSummable_rhs
internalf_lhs_terms
              internalf_rhs_terms by blast
      then show ?thesis using approx_trans assms
        using hpow_hypersum_pos_hffp[of e a z] by blast
```

qed

```
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))}"
  using Infinitesimal_epsilon base_finite base_greater_1 epsilon_gt_zero
exponentz_HFinite
        exponentz_gt_zero hpow_approx_series_pos
  by blast
```

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 -
     have internalf_rhs_terms: "(λi. (eulerK a e * z) pow i / hfact i)
∈ InternalFuns"
       by (simp add: InternalFuns_divide)
     have hyperSummable_rhs: "HyperSummable (λi. (eulerK a e * z) pow
i / hfact i)"
       by (metis (lifting) ext Diff_iff HFinite_mult HyperSummable_exp
Introductio_114_eulerK_witness base_finite
           base_greater_1 exponentz_HFinite infinitesimal_e)
     have "(λi. hypreal_of_hypnat (hypnat  (hfloor(-z/e)) hchoose i) *
(eulerK a e * -e) pow i) ∈ InternalFuns"
       using InternalFuns_hyperpow_simple InternalFuns_mult InternalFuns_of_hypnat
InternalFuns_star_choose
       by blast
     then have internalf_lhs_terms:
          "(λi. hfallfactpow (hypreal_of_hypnat (hypnat (hfloor(-z/e))))
i / hfact i *
                (eulerK a e * -e) pow i) ∈ InternalFuns"
       by (simp add: InternalFuns_divide InternalFuns_mult)
     {
       fix n :: "nat star" assume "n ∈ Nats"
       then obtain N where n: "n = hypnat_of_nat N"
         using Nats_hypnat_of_nat_iff by blast
       moreover have "hfallfactpow (hypreal_of_hypnat (hypnat (hfloor
```

```
(- (z / e))))) (hypnat_of_nat N) *
                (- (eulerK a e * e)) pow hypnat_of_nat N ≈
                (eulerK a e * z) pow hypnat_of_nat N"
        by (metis (no_types, lifting) HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow

            Introductio_114_eulerK_witness approx_reorient base_finite
base_greater_1 e_gt_zero
            exponentz_HFinite exponentz_less_zero infinitesimal_e minus_divide_left
mult_minus_right
            of_nat_eq_star_of)
      ultimately have "hfallfactpow (hypreal_of_hypnat (hypnat (hfloor(-z/e))))
n / hfact n *
                       (eulerK a e * -e) pow n ≈
                       (eulerK a e * z) pow n / hfact n"
       by (metis HFinite_not_Infinitesimal_pow_of_nat_eq_hfallfactpow_divide_neg
Introductio_114_eulerK_witness
           approx_sym base_finite base_greater_1 e_gt_zero exponentz_HFinite
exponentz_less_zero hfact_of_nat
           infinitesimal_e of_nat_eq_star_of of_nat_fact)
     }
     then have approx_lhs_rhs:
               "∀n∈ℕ. hfallfactpow (hypreal_of_hypnat (hypnat (hfloor(-z/e))))
n / hfact n *
                       (eulerK a e * -e) pow n ≈
                       (eulerK a e * z) pow n / hfact n"
       by blast
     have "- z ∈ HFinite - Infinitesimal" "-z > 0"
       using HFinite_minus_iff  Diff_iff Infinitesimal_minus_iff exponentz_HFinite
exponentz_less_zero
       by auto
     then have hs: "HyperSummable
           (λi. hfallfactpow (hypreal_of_hypnat (hypnat (hfloor(-z/e))))
i / hfact i  * (eulerK a e * -e) pow i)"
      using HyperSummable_exp_terms2 [where z=z and e=e and k="eulerK
a e"]
            Introductio_114_eulerK_witness base_finite base_greater_1
e_gt_zero eulerK_def
           exponentz_HFinite hpow_gt_one infinitesimal_e
       by auto
     then have x: "hypersum (λi. hfallfactpow (of_hypnat(hypnat(hfloor(-z/e))))
i / hfact i *
                                (eulerK a e * -e) pow i) {0..hypnat(hfloor(-z/e))}
≈
                hypersum (λi. (eulerK a e * z) pow i / hfact i) {0..hypnat(hfloor(-z/e))
       using Hypersum_Comparison_Theorem' approx_lhs_rhs hyperSummable_rhs
internalf_lhs_terms
             internalf_rhs_terms
       by blast
     then show ?thesis
```

```
        using assms approx_trans [OF hpow_hypersum_neg_hchoose'] hyperbinomial_hfallfactpow
        by (metis (no_types, lifting) ext)
qed
```

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 -
  have hsum_approx_0: "hypersum (λi. (eulerK a e * z) pow hSuc i / hfact
(hSuc i)) {0..< n} ≈ 0"
  proof -
    have internalf_shift_exp:
          "(λi. (eulerK a e * z) pow hSuc i / hfact (hSuc i)) ∈ InternalFuns"
      by (simp add: InternalFuns_divide InternalFuns_of_hypnat)
    have " HyperSummable (λi. (eulerK a e * z) pow i / hfact i)"
      by (metis HFinite_0 HFinite_mult HyperSummable_exp Introductio_114_eulerK_witness

          approx_HFinite approx_sym base_finite base_greater_1 exponentz_Infinitesimal
infinitesimal_e)
    then have hypersummable_shift_exp: "HyperSummable (λi. (eulerK a
e * z) pow hSuc i / hfact (hSuc i))"
      using HyperSummable_shift_hSuc InternalFuns_expf_term
      by blast
    {
      fix n :: "nat star"
      assume nat_n: "n ∈ ℕ"
      then have Infinitesimal_exponentz_pow: "(eulerK a e * z) pow hSuc
n ∈ Infinitesimal"
        by (metis Infinitesimal_HFinite_mult Infinitesimal_hyperpow Introductio_114_eulerK_

            base_finite base_greater_1 exponentz_Infinitesimal infinitesimal_e
mem_infmal_iff
            mult.commute zero_less_hSuc)
        have hfinite_hfact_suc_n: "hypreal_of_hypnat (hfact (hSuc n))
∈ HFinite"
          by (simp add: hfact_nat_in_Nats Nats_hypreal_of_hypnat_HFinite
nat_n hSuc_eq_add_one)
        have "hypreal_of_hypnat (hfact (hSuc n)) ∉ Infinitesimal"
          by (metis Infinitesimal_interval2 Infinitesimal_zero hfact_gt_zero
hypnat_gt_zero_iff
                    of_hypnat_1 of_hypnat_le_iff one_not_Infinitesimal
zero_le_one)
        then have "(eulerK a e * z) pow hSuc n / hypreal_of_hypnat (hfact
```

```
(hSuc n)) ∈ Infinitesimal"
          using hfinite_hfact_suc_n Infinitesimal_exponentz_pow Infinitesimal_divide_HFinit
by blast
    }
    then have "∀n∈ℕ. (eulerK a e * z) pow hSuc n / hfact (hSuc n) ≈
0"
       using Infinitesimal_approx
       by (simp add: mem_infmal_iff of_hypnat_hfact)
     thus ?thesis
       using Hypersum_Comparison_Theorem [where g="λn. 0", simplified]
             HyperSummable_def2 hypersummable_shift_exp internalf_shift_exp

       by fastforce
  qed
  have "hypersum (λi. (eulerK a e * z) pow i / hfact i) {0..<hSuc n} =
          (eulerK a e * z) pow 0 / hfact 0 +
             hypersum (λi. (eulerK a e * z) pow i / hfact i) {hSuc 0..<hSuc
n}"
    using InternalFuns_divide InternalFuns_hfact InternalFuns_hyperpow_simple
hypersum_head_upt_hSuc
    by blast
  moreover have hsum: "... = 1 + hypersum (λi. (eulerK a e * z) pow hSuc
i / hfact (hSuc i)) {0..<n}"
     by (subst hypersum_shift_bounds_hSuc_ivl, auto intro: InternalFuns_expf_term')
  then have hsum_approx_1:
       "hypersum (λi. (eulerK a e * z) pow i / hfact i) {0..<hSuc n} ≈
1"
    using approx_minus_iff calculation hsum_approx_0
    by fastforce
  have "a hpow z ≈ 1"
     using Infinitesimal_hpow_approx_one base_finite base_greater_1 exponentz_Infinitesimal

             lemma_approx_gt_zero by (simp add: mem_infmal_iff)
  then have "a hpow z ≈ hypersum (λi. (eulerK a e * z) pow i / hfact
i) {0..<hSuc n}"
    using hsum_approx_1 approx_trans2
    by blast
  thus ?thesis
    by blast
qed

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}"
  using assms(1) atLeastLessThanhSuc_atLeastAtMost base_finite base_greater_1
```

```
exponentz_Infinitesimal
    hpow_approx_series_infinitesimal'
  by presburger

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}"
  using Infinitesimal_epsilon atLeastLessThanhSuc_atLeastAtMost base_finite
base_greater_1
        exponentz_Infinitesimal hpow_approx_series_infinitesimal
  by presburger
```

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 (cases "z ≈ 0")
  assume "z ≈ 0"
  then show ?thesis
    using base_finite base_greater_1 hpow_approx_series_infinitesimal
infinitesimal_e
    by blast
next
   assume exponentz_not_approx_0: "¬ (z ≈ 0)"
    then have exponentz_not_zero: "z ≠ 0" by blast
    have z_not_Infinitesimal: "z ∉ Infinitesimal"
      by (simp add: mem_infmal_iff exponentz_not_approx_0)
    then show ?thesis
    proof (cases "z > 0")
      assume "z > 0" then show ?thesis
        by (simp add: base_finite base_greater_1 e_gt_zero exponentz_HFinite

                      hpow_approx_series_pos infinitesimal_e
                      z_not_Infinitesimal)
    next
      assume "¬ (z > 0)" then have "z < 0"
        using exponentz_not_zero
        by auto
      then show ?thesis
        using assms hpow_approx_series_neg
```

```
        by (simp add: z_not_Infinitesimal)
    qed
qed
```

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 -
    have inv_n_gt_zero: "1/hypreal_of_hypnat n > 0" (is "?epsilon > 0")
      using index_infinite HNatInfinite_of_hypnat_gt_zero zero_less_divide_1_iff

      by blast
    have infinitesimal_inv_n: "1/hypreal_of_hypnat n ∈ Infinitesimal"

      using index_infinite
      by (metis HNatInfinite_inverse_Infinitesimal inverse_eq_divide)

     show ?thesis
     proof (cases "z ≈ 0")
       assume "z ≈ 0" then show ?thesis
         by (metis (full_types) HNatInfinite_inverse_Infinitesimal base_finite
base_greater_1
             hpow_approx_series_infinitesimal index_infinite inverse_eq_divide)
     next
       assume exponentz_not_approx_0: "¬ (z ≈ 0)"
       have index2_infinite: "hypnat (hfloor (|z| / ?epsilon)) ∈ HNatInfinite"

         using  hypreal_HNatInfinite_hfloor
         by (metis HInfinite_HFinite_disj Infinitesimal_approx Infinitesimal_hrabs_iff

                   Infinitesimal_ratio Infinitesimal_zero divide_less_eq_1
divide_less_eq_1_pos
                   divide_pos_pos exponentz_not_approx_0 infinitesimal_inv_n
inv_n_gt_zero
                    not_one_less_zero zero_less_abs_iff)
       then have x: "a hpow z ≈
              hypersum (λi. (eulerK a (1 / hypreal_of_hypnat n) * z)
pow i / hfact i)
               {0..hypnat(hfloor(|z|/?epsilon))}"
```

```
        using assms(1-3) hpow_approx_powseries infinitesimal_inv_n inv_n_gt_zero

        by blast
      moreover have "hypersum (λi. (eulerK a (1 / hypreal_of_hypnat
n) * z) pow i / hfact i)
                    {0..hypnat(hfloor(|z|/?epsilon))} ≈
                    hypersum (λi. (eulerK a (1 / hypreal_of_hypnat n)
* z) pow i/ hfact i) {0..n}"
        by (metis (lifting) ext HFinite_mult HyperSummable_exp HyperSummable_hypersum_appr

            Introductio_114_eulerK_witness assms(1-3)  index2_infinite
index_infinite infinitesimal_inv_n)
      ultimately have "a hpow z ≈ hypersum (λi. (eulerK a (1 / hypreal_of_hypnat
n) * z) pow i/ hfact i) {0..n}"
      using approx_trans
      by blast
    then show ?thesis .
  qed
qed
```

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 -
  have "HyperSummable (λi. (eulerK a (1 / hypreal_of_hypnat n) * z) pow
i / hfact i)"
    by (metis (no_types, lifting) ext HFinite_mult HNatInfinite_inverse_Infinitesimal
HyperSummable_exp
       Introductio_114_eulerK_witness base_finite base_greater_1 divide_inverse
exponentz_HFinite
       lambda_one pow_infinite)
  then have "hypersum (λi. (eulerK a (1 / hypreal_of_hypnat n) * z) pow
i / hfact i) {0..m} ≈
            hypersum (λi. (eulerK a (1 / hypreal_of_hypnat n) * z) pow
i / hfact i) {0..n}"
    using HyperSummable_hypersum_approx index_infinite pow_infinite by
blast
  then show ?thesis
    using approx_trans2 base_finite base_greater_1 exponentz_HFinite Euler_hpow_approx_pows
pow_infinite
```

```
    by blast
qed

lemma epsilon_inverse_hSuc_whn:
  "ε = 1/hypreal_of_hypnat(hSuc whn)"
  by (simp add: epsilon_inverse_omega hSuc_eq_add_one inverse_eq_divide
whn_eq_ωm1)
```

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}"
  unfolding epsilon_inverse_hSuc_whn
  using assms Euler_hpow_approx_powseries
        HNatInfinite_add HNatInfinite_whn hSuc_eq_add_one
  by presburger
```

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 (rule bexI [where x="eulerK a (1 / hypreal_of_hypnat whn)"], safe)
  fix z :: "real star" and n :: "nat star"
  assume "z ∈ HFinite" "n ∈ HNatInfinite"
  then show "a hpow z ≈
              hypersum
               (λi. (eulerK a (1 / hypreal_of_hypnat whn) * z) pow i
/ hfact i) {0..n}"
    using HNatInfinite_whn base_finite base_greater_1 Euler_hpow_approx_powseries

    by blast
next
  show "eulerK a (1 / hypreal_of_hypnat whn) ∈ HFinite"
    by (metis HNatInfinite_inverse_Infinitesimal HNatInfinite_whn Introductio_114_eulerK_wi
base_finite
        base_greater_1 inverse_eq_divide)
qed

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 -

  have
    "hypreal_of_hypnat (N hchoose k) * (1 / of_hypnat N) pow k =
     hfallfactpow (of_hypnat N) k / (hfact k * of_hypnat N pow k)"
  proof -
    have "hypreal_of_hypnat (N hchoose k) =
          hfallfactpow (of_hypnat N) k / hfact k"
      using hyperbinomial_hfallfactpow by auto
      thus ?thesis
        by (simp add: hyperpow_divide)
  qed

  moreover have "hfallfactpow (of_hypnat N) k ≤ hypreal_of_hypnat N
pow k"
    by (simp add: assms(1) hfallfactpow_le_hyperpow)
  moreover have "(0 :: hypreal) < of_hypnat N pow k"
    using assms(2) hyperpow_gt_zero by blast
  moreover have "(0 :: hypreal) < hfact k"
    using hfact_gt_zero of_hypnat_0_less_iff by blast
  ultimately show ?thesis
    by (metis (no_types, lifting) divide_le_cancel linorder_not_less mult_less_0_iff
        nonzero_divide_mult_cancel_right order_less_le)
qed

lemma geometric_half_sum_lt_two:
  "∑h ((pow) (1 / 2)) {0..M} < (2::hypreal)"
proof -
  have sum_eq: "∑h ((pow) (1 / 2)) {0..<M + 1} = ((1 / (2::hypreal))
pow (M + 1) - 1) / (1 / 2 - 1)"
     by (simp add: hypersum_geometric)
  moreover have sum_eq2: "∑h ((pow) (1 / (2::hypreal))) {0..M} = ∑h
((pow) (1 / 2)) {0..<M + 1}"
    using atLeastLessThanhSuc_atLeastAtMost hSuc_eq_add_one by auto
  moreover have pow_pos: "(1/2 :: hypreal) pow (M + 1) > 0"
    by (simp add: hyperpow_gt_zero)
  moreover have denom: "(1/2 :: hypreal) - 1 = -1/2" by simp
  ultimately show ?thesis by simp
qed

lemma inv_fact_sum_le_three:
  "hypersum (λk. 1 /  (hfact k)) {0..N} ≤ (3 ::hypreal)"
proof -
  have zero_term: "1 / hfact 0 = (1::hypreal)"
    by simp
  have term_bound: "∀j. (1 :: hypreal) / hfact (j + 1) ≤ (1/2) pow j"
  proof
```

```
    fix j :: hypnat
    have hf_pos: "(0::hypreal) <  hfact (j+1)"
      using hfact_gt_zero of_hypnat_0_less_iff by blast
    moreover have "hfact (j+1) ≥ (2::hypreal) pow j"
      by (simp add: hfact_hSuc_ge_two_pow of_hypnat_hfact)
    ultimately show "(1::hypreal) / hfact (j + 1) ≤ (1/2) pow j"
      by (simp add: frac_le hyperpow_divide hyperpow_gt_zero)
  qed
  have internal_f: "(λa. (1::hypreal) / hfact a) ∈ InternalFuns"
      by (simp add: InternalFuns_divide InternalFuns_of_hypnat)
  have split: "(∑h k∈{0..N}. (1::hypreal) / hfact k) = 1 + (∑h k∈{0..<N}.
1 / hfact (hSuc k))"
  proof (subst hypersum_shift_bounds_hSuc_ivl [symmetric])
    show "(λa. (1::hypreal) / hfact a) ∈ InternalFuns"
      by (rule internal_f)
  next
    show "(∑h k∈{0..N}. (1::hypreal) / hfact k) = 1 + (∑h a∈{hSuc
0..<hSuc N}. 1 / hfact a)"
    proof(subst hypersum_head_hSuc)
      show "(λk. (1::hypreal) / hfact k) ∈ InternalFuns" "N ≥ 0"
        by (auto simp add: internal_f)
    next
      show "(1::hypreal) / hfact 0 + (∑h k∈{hSuc 0..N}. 1 / hfact k)
=
            1 + (∑h a∈{hSuc 0..<hSuc N}. 1 / hfact a)"
        by (simp add: atLeastLessThanhSuc_atLeastAtMost)
    qed
  qed
  have tail_bound:
    "(∑h k∈{0..<N}. (1::hypreal) / hfact (k + 1)) ≤
      ∑h ((pow) (1 / 2)) {0..<N}"
  proof (rule hypersum_mono)
    show "(λk. (1::hypreal) / hfact (k + 1)) ∈ InternalFuns"
      by (simp add: InternalFuns_add InternalFuns_divide InternalFuns_of_hypnat)
  next
    show "(pow) (1 / 2) ∈ InternalFuns" "{0..<N} ∈ InternalSets"
      by auto
  next
    fix i
    assume "i ∈ {0..<N}"
    show "(1::hypreal) / hfact (i + 1) ≤ (1 / 2) pow i"
      by (rule term_bound [THEN spec])
  qed
  have geo_bound: "∑h ((pow) (1 / 2)) {0..<N} < (2::hypreal)"
    using geometric_half_sum_lt_two [of "N - 1"]
    by (metis atLeastLessThanhSuc_atLeastAtMost diff_numeral_special(9)
divide_eq_0_iff divide_eq_eq_1
       hSuc_eq_add_one hyperpow_zero hypersum_geometric hypnat_le_add_diff_inverse2
hypnat_less_one not_less
```

```
      numeral_eq_one_iff verit_eq_simplify(10) zero_less_numeral)
  show ?thesis
    using split geo_bound hSuc_eq_add_one tail_bound by force
qed

lemma HNatInfinite_one_plus_inv_pow_HFinite:
  assumes "n ∈ HNatInfinite"
  shows "(1 + 1 / hypreal_of_hypnat n) pow n ∈ HFinite"
proof -
  have n_pos: "(0::hypreal) < of_hypnat n"
    using assms HNatInfinite_of_hypnat_gt_zero of_hypnat_0_less_iff by
blast

  have expand:
    "(1 + 1 / hypreal_of_hypnat n) pow n =
     hypersum (λk. of_hypnat (n hchoose k) * (1 / of_hypnat n) pow k)
{0..n}"
    using hyperbinomial_simple [of "1 / of_hypnat n" n] by simp

  have term_le:
    "∀k ∈ {0..n}. hypreal_of_hypnat (n hchoose k) * (1 / of_hypnat n)
pow k
                  ≤ 1 /  (hfact k)"
    using binomial_term_le_inv_fact n_pos by auto

  have sum_le:
    "(1 + 1 / hypreal_of_hypnat n) pow n ≤
     hypersum (λk. 1 / hfact k) {0..n}"
  proof (unfold expand, rule hypersum_mono)
    show "(λk. hypreal_of_hypnat (n hchoose k) * (1 / hypreal_of_hypnat
n) pow k) ∈ InternalFuns"
      by  (simp add: InternalFuns_mult InternalFuns_of_hypnat InternalFuns_star_choose)
  next
    show "(λk. 1 / hfact k) ∈ InternalFuns"
      by (simp add: InternalFuns_o2 star_divide_def)
  next
    show "{0..n} ∈ InternalSets"
      by simp
  next
    fix i
    assume "i ∈ {0..n}"
    then show "hypreal_of_hypnat (n hchoose i) * (1 / hypreal_of_hypnat
n) pow i ≤
          1 /  hfact i"
      using term_le by blast
  qed

  have sum_bound:
    "hypersum (λk. 1 / hfact k) {0..n} ≤ (3::hypreal)"
```

```
    using inv_fact_sum_le_three by blast

  have pos: "(1 + 1 / hypreal_of_hypnat n) pow n > 0"
    by (meson add_pos_pos hyperpow_gt_zero n_pos zero_less_divide_1_iff
zero_less_one)

  have le_three: "(1 + 1 / hypreal_of_hypnat n) pow n ≤ 3"
    using sum_le sum_bound by linarith
  then show ?thesis
    using HFinite_bounded HFinite_numeral order_less_imp_le pos by blast
qed

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 -
  have "n ≠ 0"
    by (metis pow_infinite zero_not_mem_HNatInfinite)
  then have eulerK_1: "eulerK ((1 + 1/of_hypnat n) pow  n) (1/of_hypnat
n) = 1"
    using hpow_hyperpow_eq [symmetric]
    by (simp add: eulerK_def  add_pos_pos hpow_mult field_simps)
  moreover have "(1 + 1 / hypreal_of_hypnat n) pow n ∈ HFinite"
    by (simp add: HNatInfinite_one_plus_inv_pow_HFinite pow_infinite)
  moreover have "1 < (1 + 1 / hypreal_of_hypnat n) pow n"
    by (metis HNatInfinite_of_hypnat_gt_zero add_pos_pos hpow_gt_one hpow_hyperpow_eq
pow_infinite
       less_add_same_cancel1 zero_less_divide_iff zero_less_one)
   ultimately show ?thesis
     using assms Euler_hpow_approx_powseries
     by force
 qed

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}"
  by (simp add: assms hpow_approx_powseries_exp_lemma)
```

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}"
  using hpow_approx_powseries_exp_lemma2 [where z=1,  simplified]
  by (simp add: add_pos_pos hyperpow_gt_zero)
```

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"*
by *(simp add: add_pos_pos euler_e_def hyperpow_gt_zero)*

The (infinitely close) series for $\mathfrak{e}$

lemma *Euler_e:*
"𝔢 ≈ hypersum (λi. (1::hypreal)/ hfact i) {0..whn}"
by *(simp add: euler_e_def hpow_approx_powseries_exp_lemma3)*

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}"
by *(simp add: assms euler_e_def hpow_approx_powseries_exp_lemma2)*

lemma *euler_e_hpow_zero [simp]: "𝔢 hpow 0 = 1"*
by *simp*

lemma *euler_e_hpow_one [simp]: "𝔢 hpow 1 = 𝔢"*
by *simp*

lemma *hexp_gt_one: "0 < x ⟹ 1 < 𝔢 hpow x"*
by *(metis add_pos_pos divide_pos_pos euler_e_def hpow_gt_one hpow_hyperpow_eq hypnat_zero_less_hypnat_omega*
less_add_same_cancel1 of_hypnat_0_less_iff zero_less_one)

lemma *HFinite_euler_e [simp]:*
"(𝔢::real star) ∈ HFinite"
by *(simp add: HNatInfinite_one_plus_inv_pow_HFinite euler_e_def)*

lemma *euler_e_gt_one [simp]:*
"(𝔢::real star) > 1"
using *hexp_gt_one [of 1]*
by *(simp add: euler_e_def hyperpow_gt_zero hypreal_add_zero_less_le_mono)*

lemma *euler_e_ne_one: "(𝔢::hypreal) ≠ 1"*
using *euler_e_gt_one*
by *linarith*

```
lemma "eulerK 𝔢 (1 / hypreal_of_hypnat whn) = 1"
    using hpow_hyperpow_eq [symmetric]
    by (simp add: eulerK_def euler_e_def add_pos_pos hpow_mult field_simps)

lemma euler_e_ge_two: "(𝔢::hypreal) ≥ 2"
proof -
  have whn_pos: "(0::hypreal) < of_hypnat whn"
    using HNatInfinite_of_hypnat_gt_zero HNatInfinite_whn by blast
  then have inv_pos: "(0::hypreal) ≤ 1 / of_hypnat whn"
    by auto
  obtain x :: hypreal where x_ge: "x ≥ 0" and
    expand: "(1 + 1 / of_hypnat whn) pow whn = 1 + of_hypnat whn * (1
/ of_hypnat whn) + x"
    using hyperbinomial_expand_first_two_terms [OF inv_pos, of whn] by
auto
  have "of_hypnat whn * (1 / of_hypnat whn) = (1::hypreal)"
    using whn_pos by auto
  then have "(1 + 1 / of_hypnat whn) pow whn = 2 + x"
    using expand by auto
  then show ?thesis
    using x_ge euler_e_def
    by (metis add.commute le_add_same_cancel2)
qed

lemma euler_e_power_ge_two_power: "(𝔢::hypreal) ^ n ≥ 2 ^ n"
  by (simp add: euler_e_ge_two power_mono)

lemma euler_e_power_bound:
  assumes "a ∈ HFinite" "a ≥ 1"
  shows "∃n. (𝔢::hypreal) ^ n ≥ a"
proof -
  obtain m where "a ≤ of_nat m"
    using HFinite_less_Nat_Ex assms(1) Nats_cases
    by (metis order_less_imp_le)
  moreover have "∃n. of_nat m ≤ (2::hypreal) ^ n"
    using self_le_ge2_pow by fastforce
  then obtain n where "of_nat m ≤ (2::hypreal) ^ n" by auto
  moreover have "(2::hypreal) ^ n ≤ 𝔢 ^ n"
    using euler_e_power_ge_two_power by auto
  ultimately show ?thesis by (metis order_trans)
qed

lemma hpow_euler_e_eulerK_approx:
  assumes "a ∈ HFinite"
      and "a > 1"
    shows "𝔢 hpow (eulerK a (1/of_hypnat whn)) ≈ a"
  using assms Euler_hpow_approx_powseries [where a=a and z=1 and n=whn
and m=whn, simplified]
```

```
      Euler_exp_powseries HFinite_eulerK_inverse_whn approx_trans2
  by blast

lemma euler_e_hpow_not_approx_one_pos:
  assumes "d ∈ HFinite" "d > 0" "d ∉ Infinitesimal"
  shows "¬ (𝔢 hpow d ≈ 1)"
proof -
  have d_HF_not_Inf: "d ∈ HFinite - Infinitesimal"
    using assms(1,3)
    by blast
  then have inv_d_HF: "inverse d ∈ HFinite"
    using HFinite_inverse2
    by blast
  then obtain m where m_bound: "inverse d < of_nat m"
    using HFinite_less_Nat_Ex Nats_cases
    by blast
  have m_pos: "m > 0"
  proof (rule ccontr)
    assume "¬ m > 0" then have "m = 0" by simp
    then have "inverse d < 0"
      using m_bound
      by simp
    then show False
      using assms(2)
      by fastforce
  qed
  have d_lower: "d > inverse (of_nat m)"
    by (simp add: assms(2) inverse_less_imp_less m_bound)
  have "𝔢 hpow d > 𝔢 hpow (inverse (of_nat m))"
    using d_lower euler_e_gt_one hpow_less_mono
    by blast
  moreover have "𝔢 hpow (inverse (of_nat m)) ≥ star_of 2 hpow (inverse
(of_nat m))"
  proof -
    have "inverse (of_nat m) > (0::hypreal)"
      using m_pos
      by simp
    moreover have "(star_of 2 :: hypreal) ≤ 𝔢"
      using euler_e_ge_two
      by simp
    moreover have "(0::hypreal) < star_of 2"
      by simp
    ultimately show ?thesis
      by (metis hpow_less_mono_base le_less less_imp_le)
  qed
  moreover have "star_of 2 hpow (inverse (of_nat m)) = star_of (2 pow_ℝ
(inverse (real m)))"
    by transfer simp
  moreover have "2 pow_ℝ (inverse (real m)) > 1"
```

```
    using m_pos
    by (simp add: powreal_gt_one)
  ultimately have lower: "𝔢 hpow d > star_of (2 pow_ℝ (inverse (real m)))"

    by linarith
  show "¬ (𝔢 hpow d ≈ 1)"
  proof
    assume "𝔢 hpow d ≈ 1"
    then have "𝔢 hpow d - 1 ∈ Infinitesimal"
      by (simp add: Infinitesimal_approx_minus)
    moreover have "𝔢 hpow d - 1 > star_of (2 pow_ℝ (inverse (real m)))
- 1"
      using lower by simp
    moreover have "star_of (2 pow_ℝ (inverse (real m))) - 1 > 0"
      using ‹2 pow_ℝ (inverse (real m)) > 1›
      by simp
    ultimately have "star_of (2 pow_ℝ (inverse (real m))) - 1 ∈ Infinitesimal"
      using Infinitesimal_interval
      by blast
    moreover have "2 pow_ℝ (inverse (real m)) - 1 ≠ 0"
      using ‹2 pow_ℝ (inverse (real m)) > 1›
      by force
    then have "star_of (2 pow_ℝ (inverse (real m)) - 1) ∉ Infinitesimal"
      by blast
    ultimately show False
      by simp
  qed
qed

lemma euler_e_hpow_not_approx_one_neg:
  assumes "d ∈ HFinite" "d < 0" "d ∉ Infinitesimal"
  shows "¬ (𝔢 hpow d ≈ 1)"
proof -
  have "-d > 0" "-d ∉ Infinitesimal" "-d ∈ HFinite"
    using assms by (auto simp: HFinite_minus_iff)
  then have neg_not_approx: "¬ (𝔢 hpow (-d) ≈ 1)"
    by (simp add: euler_e_hpow_not_approx_one_pos)
  show "¬ (𝔢 hpow d ≈ 1)"
  proof
    assume "𝔢 hpow d ≈ 1"
    have e_hpow_d_HF_not_Inf: "𝔢 hpow d ∈ HFinite - Infinitesimal"
      using HFinite_hpow_HFinite_Diff_Infinitesimal HFinite_euler_e euler_e_gt_one

            assms(1) less_imp_le
      by blast
    then have "inverse (𝔢 hpow d) ≈ inverse 1"
      using ‹𝔢 hpow d ≈ 1› approx_inverse approx_sym
      by blast
    then have "inverse (𝔢 hpow d) ≈ 1"
```

```
    by simp
  moreover have "𝔢 hpow (-d) = inverse (𝔢 hpow d)"
    using euler_e_pos hpow_minus
    by blast
  ultimately have "𝔢 hpow (-d) ≈ 1"
    by simp
  then show False
    by (simp add: neg_not_approx)
 qed
qed

lemma euler_e_hpow_not_approx_one:
 assumes "d ∈ HFinite" "d ≠ 0" "d ∉ Infinitesimal"
 shows "¬ (𝔢 hpow d ≈ 1)"
 by (meson assms euler_e_hpow_not_approx_one_neg euler_e_hpow_not_approx_one_pos
     linorder_neqE_linordered_idom)

corollary euler_e_hpow_approx_one_iff_Infinitesimal:
 assumes "d ∈ HFinite"
 shows "(𝔢 hpow d ≈ 1) ⟷ (d ∈ Infinitesimal)"
 using HFinite_euler_e Infinitesimal_hpow_approx_one assms euler_e_gt_one
euler_e_hpow_not_approx_one
 by blast

lemma euler_e_hpow_approx_inject:
 assumes "𝔢 hpow x ≈ 𝔢 hpow y"
     and "x ∈ HFinite"
     and "y ∈ HFinite"
 shows "x ≈ y"
proof -
 have "𝔢 hpow (x - y) = 𝔢 hpow x / 𝔢 hpow y"
   using euler_e_pos hpow_divide2
   by blast
 moreover have "𝔢 hpow x / 𝔢 hpow y ≈ 1"
 proof -
   have ey_HF: "𝔢 hpow y ∈ HFinite - Infinitesimal"
     using HFinite_hpow_HFinite_Diff_Infinitesimal HFinite_euler_e euler_e_gt_one

           assms(3) less_imp_le
     by blast
   then have "𝔢 hpow x / 𝔢 hpow y ≈ 𝔢 hpow y / 𝔢 hpow y"
     using assms(1) approx_divide_HFinite_diff_Infinitesimal by blast
   then show ?thesis
     by (simp add: hpow_not_zero)
 qed
 ultimately have hpow_diff: "𝔢 hpow (x - y) ≈ 1" by simp
 then show "x ≈ y"
   by (meson HFinite_diff assms(2,3) bex_Infinitesimal_iff euler_e_hpow_approx_one_iff_Inf
qed
```

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"*
by *(metis Ln_def add_pos_pos divide_pos_pos euler_e_def hLog_eq_one hexp_gt_one hpow_one hyperpow_gt_zero hypnat_zero_less_hypnat_omega less_numeral_extra(1) of_hypnat_0_less_iff order_less_irrefl)*

lemma *hpow_e_Ln [simp]:*
assumes *"a > 0"*
shows *"𝔢 hpow (Ln a) = a"*
by *(metis Ln_def add_pos_pos assms euler_e_def hexp_gt_one hpow_hLog_cancel hpow_one hyperpow_gt_zero hypnat_zero_less_hypnat_omega less_numeral_extra(1,4) of_hypnat_0_less_iff zero_less_divide_1_iff)*

lemma *Ln_ge_zero: "a ≥ 1 ⟹ Ln a ≥ 0"*
using *Ln_def* **by** *fastforce*

lemma *Ln_inverse: "x > 0 ⟹ Ln (inverse x) = - Ln x"*
by *(simp add: Ln_def euler_e_ne_one hLog_inverse)*

lemma *Ln_less_zero: "a > 0 ⟹ a < 1 ⟹ Ln a < 0"*
by *(simp add: Ln_def)*

lemma *Ln_HFinite:*
assumes *"a ∈ HFinite - Infinitesimal" "a > 0"*
shows *"Ln a ∈ HFinite"*
proof *(cases "a ≥ 1")*
case *True*
obtain *n* **where** *n_bound: "𝔢 ^ n ≥ a"*
using *euler_e_power_bound assms(1) True*
by *(metis DiffD1)*
then have *"𝔢 hpow (of_nat n) ≥ a"*
by *(simp add: hpow_power_eq)*
then have *"hLog 𝔢 (𝔢 hpow (of_nat n)) ≥ hLog 𝔢 a"*
using *assms(2)*
by *auto*
then have *"of_nat n ≥ hLog 𝔢 a"*
by *(simp add: euler_e_ne_one)*
then have *"Ln a ≤ of_nat n"*
by *(simp add: Ln_def)*
then show *?thesis*
using *HFinite_bounded HFinite_of_nat Ln_ge_zero True*
by *blast*

```
next
  case False
  then have a_lt_1: "a < 1"
    by simp
  then have inv_a_ge_1: "inverse a ≥ 1"
    using assms(2) one_le_inverse_iff
    by fastforce
  have inv_a_HFinite: "inverse a ∈ HFinite"
    using assms(1) HFinite_inverse2
    by blast
  have inv_a_HFinite_diff: "inverse a ∈ HFinite - Infinitesimal"
    using HFinite_not_Infinitesimal_inverse assms(1)
    by blast
  have inv_a_pos: "inverse a > 0"
    using assms(2) by simp
  have "Ln (inverse a) ∈ HFinite"
  proof -
    have "Ln (inverse a) ≥ 0"
      by (simp add: Ln_ge_zero inv_a_ge_1)
    moreover obtain n where "𝔢 ^ n ≥ inverse a"
      using euler_e_power_bound inv_a_HFinite inv_a_ge_1 by blast
    then have "𝔢 hpow (of_nat n) ≥ inverse a"
      using euler_e_pos hpow_power_eq
      by simp
    then have "hLog 𝔢 (𝔢 hpow (of_nat n)) ≥ hLog 𝔢 (inverse a)"
      using euler_e_pos hpow_gt_zero inv_a_pos
      by simp
    then have "of_nat n ≥ hLog 𝔢 (inverse a)"
      using euler_e_pos euler_e_ne_one
      by simp
    then have "Ln (inverse a) ≤ of_nat n"
      by (simp add: Ln_def)
    ultimately show ?thesis
      using HFinite_bounded HFinite_of_nat
      by blast
  qed
  moreover have "Ln a = - Ln (inverse a)"
    using euler_e_pos euler_e_ne_one assms(2)
    by (simp add: Ln_def hLog_inverse)
  ultimately show ?thesis
    by (simp add: HFinite_minus_iff)
qed


lemma Ln_eulerK_approx:
  assumes "a ∈ HFinite" "a > 1"
  shows "Ln a ≈ eulerK a (1 / hypreal_of_hypnat whn)"
  using assms euler_e_hpow_approx_inject hpow_euler_e_eulerK_approx hpow_e_Ln
Ln_HFinite
```

```
      HFinite_eulerK_inverse_whn
  by (metis Diff_iff Infinitesimal_interval Infinitesimal_zero approx_sym
dual_order.strict_trans
      one_not_Infinitesimal zero_less_one)
```

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"
  using Ln_eulerK_approx assms eulerK_def by auto
```

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 -
  have "(∑h i∈{0..whn}. (eulerK a (1 / hypreal_of_hypnat whn) * z) pow
i / hfact i) ≈
        (∑h i∈{0..whn}. (Ln a * z) pow i / hfact i)"
  proof (rule Hypersum_Comparison_Theorem')
    show "(λi. (eulerK a (1 / hypreal_of_hypnat whn) * z) pow i / hfact
i) ∈ InternalFuns"
      by simp
  next
    show "(λi. (Ln a * z) pow i / hfact i) ∈ InternalFuns"
      by simp
  next
    show "HyperSummable
           (λi. (eulerK a (1 / hypreal_of_hypnat whn) * z) pow i / hfact
i)"
      by (simp add: assms HFinite_eulerK_inverse_whn HFinite_mult HyperSummable_exp)
  next
    show "HyperSummable (λi. (Ln a * z) pow i / hfact i)"
      using assms
      by (metis HFinite_eulerK_inverse_whn HFinite_mult HyperSummable_exp
Ln_eulerK_approx
          approx_HFinite approx_reorient)
  next
    show "∀n∈ℕ. (eulerK a (1 / hypreal_of_hypnat whn) * z) pow n / hfact
n ≈
           (Ln a * z) pow n / hfact n"
    proof
      fix n :: hypnat
      assume Nat_n: "n ∈ ℕ"
```

```
      then have "(z * eulerK a (1 / hypreal_of_hypnat whn)) pow n ≈ (z
* Ln a) pow n"
        by (metis assms HFinite_eulerK_inverse_whn HFinite_mult Ln_eulerK_approx
Nats_hypnat_of_nat_iff approx_mult1
            approx_reorient hyperpow_approx mult.commute
            of_nat_eq_star_of)
      moreover have "(hfact n :: hypreal) ∈ HFinite - Infinitesimal"
        by (metis Nats_hypnat_of_nat_iff Reals_of_nat SReal_HFinite_diff_Infinitesimal
Nat_n hfact_nonzero
            hfact_of_nat )
      ultimately
      show "(eulerK a (1 / hypreal_of_hypnat whn) * z) pow n / hfact
n ≈
            (Ln a * z) pow n / hfact n"
        by (simp add: approx_divide_HFinite_diff_Infinitesimal mult.commute)
    qed
  qed
  then show ?thesis
    using Euler_hpow_approx_powseries [where n=whn and m=whn] assms HNatInfinite_whn
approx_trans
    by blast
qed
```

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 -
  have "(λn. z ^ n / fact n) sums (exp z)"
    using exp_converges [of z]
    by (simp add: inverse_eq_divide nonzero_divide_eq_eq)
  then show ?thesis
    by (simp add: sums_NSsums_iff)
qed

lemma hypersum_exp_approx:
  assumes "N ∈ HNatInfinite"
  shows "hypersum (*f* (λn. z ^ n / fact n)) {0..<N} ≈ of_real (exp z)"
  using exp_NSsums NSsums_hypersum_HNatInfinite_approx_iff assms
  by blast

lemma starfun_exp_of_real: "(*f* exp) (of_real z) = of_real (exp z)"
  by (metis exp_of_real star_of_real_def starfun_star_of)

lemma starfun_exp_term_eq:
  "(*f* (λn. z ^ n / fact n)) n = (of_real z) pow n / hfact n"
```

```
  by (simp add: hfact_def starfun_power)

lemma HyperSummable_starfun_real_exp:
  "HyperSummable (*f* (λn. (z::real) ^ n / fact n))"
proof -
  have "summable (λn. z ^ n / fact n)"
    using exp_NSsums sums_NSsums_iff sums_summable
    by blast
  then show ?thesis
    using HyperSummable_starfun_summable_iff
    by blast
qed

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}"
  by (metis (no_types, lifting) ext approx_refl starfun_exp_term_eq)
```

Theorem linking the two representations of the exponential series.

```
theorem euler_e_hpow_approx_starfun_exp:
  "℮ hpow (of_real z) ≈ (*f* exp) (of_real z)"
proof -
  let ?z = "of_real z :: hypreal"

  have hypsum_std_exp: "hypersum (*f* (λn. z ^ n / fact n)) {0..<hSuc
whn} ≈ of_real (exp z)"
    using hypersum_exp_approx [OF HNatInfinite_add [OF HNatInfinite_whn]]

    by (simp add: hSuc_eq_add_one)
  moreover have sums_approx:
    "hypersum (*f* (λn. z ^ n / fact n)) {0..<hSuc whn} ≈
     hypersum (λi. ?z pow i / hfact i) {0..whn}"
    using atLeastLessThanhSuc_atLeastAtMost hypersum_starfun_exp_approx_euler_sum

    by presburger
  ultimately have "of_real (exp z) ≈ hypersum (λi. ?z pow i / hfact i)
{0..whn}"
    using  sums_approx approx_sym approx_trans
    by blast
  then have "℮ hpow ?z ≈ of_real (exp z)"
    by (metis Euler_exp_powseries HFinite_star_of approx_trans2 of_real_eq_star_of)
  then show ?thesis
    using starfun_exp_of_real
    by simp
qed

corollary euler_e_hpow_approx_exp:
  "℮ hpow (hypreal_of_real z) ≈ hypreal_of_real (exp z)"
  using euler_e_hpow_approx_starfun_exp starfun_exp_of_real by simp
```

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)"
  by (simp add: approx_sym euler_e_hpow_approx_exp st_unique)
```

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)"
  using euler_e_hpow_approx_exp [of 1] by simp

corollary st_euler_e_approx_exp1: "st (𝔢::hypreal) = of_real (exp 1)"
  using euler_e_approx_exp1 st_eq_approx_iff by force
```

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 -
  let ?k = "eulerK a e"
  let ?N = "hypnat(hfloor(1/e))"

  have k_HF: "?k ∈ HFinite"
    using Introductio_114_eulerK_witness inf_e a_HF a_gt1
    by blast
  have N_inf: "?N ∈ HNatInfinite"
    using e_pos hypreal_HNatInfinite_hfloor_inverse_Infinitesimal inf_e

    by blast
  have "a hpow 1 ≈
       hypersum (λi. (?k * 1) pow i / hfact i) {0..?N}"
    using hpow_approx_powseries [OF inf_e e_pos a_HF a_gt1 HFinite_1]
    by simp
  then have sum_approx_a:
    "a ≈ hypersum (λi. ?k pow i / hfact i) {0..?N}"
    using a_gt1
    by simp
  have sum_approx_exp:
    "𝔢 hpow ?k ≈ hypersum (λi. ?k pow i / hfact i) {0..whn}"
    using Euler_exp_powseries k_HF
    by blast
  have summable: "HyperSummable (λi. ?k pow i / hfact i)"
    using HyperSummable_exp k_HF
    by blast
```

```
  then have sums_approx:
    "hypersum (λi. ?k pow i / hfact i) {0..?N} ≈
     hypersum (λi. ?k pow i / hfact i) {0..whn}"
    using HyperSummable_hypersum_approx N_inf HNatInfinite_whn
    by blast
  have "a ≈ hypersum (λi. ?k pow i / hfact i) {0..whn}"
    using sum_approx_a sums_approx approx_trans
    by blast
  then have "a ≈ 𝔢 hpow ?k"
    using sum_approx_exp approx_trans2
    by blast
  then show ?thesis
    using approx_sym
    by blast
qed

lemma eulerK_approx_Ln_pos:
  assumes "e ∈ Infinitesimal" "e > 0" "a ∈ HFinite" "a > 1"
  shows "eulerK a e ≈ Ln a"
proof -
  have k_HF: "eulerK a e ∈ HFinite"
    using Introductio_114_eulerK_witness assms
    by blast
  have a_pos: "a > 0"
    using assms(4) by (simp add: less_trans)
  have Ln_HF: "Ln a ∈ HFinite"
    using Ln_HFinite assms(3,4)
    by (metis Diff_iff Infinitesimal_interval Infinitesimal_zero
        dual_order.strict_trans one_not_Infinitesimal zero_less_one)
  have "𝔢 hpow (eulerK a e) ≈ a"
    using euler_e_hpow_eulerK_approx_base assms
    by blast
  moreover have "a = 𝔢 hpow (Ln a)"
    using hpow_e_Ln a_pos
    by simp
  ultimately have "𝔢 hpow (eulerK a e) ≈ 𝔢 hpow (Ln a)"
    by simp
  then show ?thesis
    using euler_e_hpow_approx_inject k_HF Ln_HF
    by blast
qed
```

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 -
  have neg_e_inf: "-e ∈ Infinitesimal"
    using assms(1) Infinitesimal_minus_iff
```

```
    by blast
  have "eulerK a (-(-e)) ≈ eulerK a (-e)"
    using eulerK_neg_approx [OF assms(3,4) neg_e_inf]
    by blast
  then have "eulerK a e ≈ eulerK a (-e)"
    by simp
  moreover have "eulerK a (-e) ≈ Ln a"
    using eulerK_approx_Ln_pos neg_e_inf assms
    by force
  ultimately show ?thesis
    using approx_trans
    by blast
qed
```

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"
  using assms eulerK_approx_Ln_neg eulerK_approx_Ln_pos
  by fastforce

lemma hpow_difference_quotient:
  assumes "a > 0"
  shows "(a hpow (x + e) - a hpow x) / e = a hpow x * eulerK a e"
proof -
  have "a hpow (x + e) = a hpow x * a hpow e"
    using hpow_add assms(1)
    by blast
  then have "a hpow (x + e) - a hpow x = a hpow x * (a hpow e - 1)"
    by (simp add: algebra_simps)
  then have "(a hpow (x + e) - a hpow x) / e = a hpow x * ((a hpow e
- 1) / e)"
    by simp
  then show ?thesis
    by (simp add: eulerK_def)
qed
```

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 -
  have "a > 0"
    using a_gt1 by (simp add: less_trans)
  then have factor: "(a hpow (x + e) - a hpow x) / e = a hpow x * eulerK
```

```
a e"
    using hpow_difference_quotient
    by blast
  have eulerK_Ln: "eulerK a e ≈ Ln a"
    by (simp add: a_HF a_gt1 e_inf e_ne0 eulerK_approx_Ln)
  have hpow_HF: "a hpow x ∈ HFinite"
    using HFinite_hpow1 a_HF x_HF a_gt1 less_imp_le
    by blast
  then have "a hpow x * eulerK a e ≈ a hpow x * Ln a"
    by (simp add: approx_mult2 eulerK_Ln)
  then show ?thesis
    by (simp add: factor mult.commute)
qed
```

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 -
  have "(𝔢 hpow (x + e) - 𝔢 hpow x) / e ≈ Ln 𝔢 * (𝔢 hpow x)"
    using NSderivative_hpow [OF HFinite_euler_e euler_e_gt_one assms]

    by blast
  then show ?thesis
    by simp
qed

lemma NSderivative_hpow_at_zero:
  assumes "a ∈ HFinite" "a > 1" "e ∈ Infinitesimal" "e ≠ 0"
  shows "(a hpow e - 1) / e ≈ Ln a"
proof -
  have "(a hpow (0 + e) - a hpow 0) / e ≈ Ln a * (a hpow 0)"
    using HFinite_0 NSderivative_hpow assms
    by blast
  then show ?thesis
    using assms(2)
    by (simp add: less_trans)
qed
```

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"
  using NSderivative_hpow_at_zero [OF HFinite_euler_e euler_e_gt_one assms]
  by simp

lemma "(𝔢 hpow ε - 1) / ε ≈ 1"
  by (simp add: NSderivative_euler_e_at_zero epsilon_not_zero)
```

end