

Discrete Summation

Florian Haftmann
with contributions by Amine Chaieb

March 17, 2025

Abstract

These theories introduce basic concepts and proofs about discrete summation: shifts, formal summation, falling factorials and stirling numbers. As proof of concept, a simple summation conversion is provided.

1 Falling factorials

```
theory Factorials
  imports Complex-Main HOL-Combinatorics.Stirling
begin

lemma pochhammer-0 [simp]: — TODO move
  pochhammer 0 n = (0::nat) if n > 0
  using that by (simp add: pochhammer-prod)

definition ffact :: nat ⇒ 'a::comm-semiring-1-cancel ⇒ 'a
  where ffact n a = pochhammer (a + 1 - of-nat n) n

lemma ffact-0 [simp]:
  ffact 0 = (λx. 1)
  by (simp add: fun-eq-iff ffact-def)

lemma ffact-Suc:
  ffact (Suc n) a = a * ffact n (a - 1)
  for a :: 'a :: comm-ring-1
  by (simp add: ffact-def pochhammer-prod prod.atLeast0-lessThan-Suc algebra-simps)

lemma ffact-Suc-rev:
  ffact (Suc n) m = (m - of-nat n) * ffact n m
  for m :: 'a :: {comm-semiring-1-cancel, ab-group-add}
  unfolding ffact-def pochhammer-rec by (simp add: diff-add-eq)

lemma ffact-nat-triv:
  ffact n m = 0 if m < n
```

```

using that by (simp add: ffact-def)

lemma ffact-Suc-nat:
  ffact (Suc n) m = m * ffact n (m - 1)
    for m :: nat
proof (cases n ≤ m)
  case True
  then show ?thesis
    by (simp add: ffact-def pochhammer-prod algebra-simps prod.atLeast0-lessThan-Suc)
next
  case False
  then have m < n
    by simp
  then show ?thesis
    by (simp add: ffact-nat-triv)
qed

lemma ffact-Suc-rev-nat:
  ffact (Suc n) m = (m - n) * ffact n m
proof (cases n ≤ m)
  case True
  then show ?thesis
    by (simp add: ffact-def pochhammer-rec Suc-diff-le)
next
  case False
  then have m < n by simp
  then show ?thesis by (simp add: ffact-nat-triv)
qed

lemma fact-div-fact-ffact:
  fact n div fact m = ffact (n - m) n if m ≤ n
proof -
  from that have fact n = ffact (n - m) n * fact m
    by (simp add: ffact-def pochhammer-product pochhammer-fact)
  moreover have fact m dvd (fact n :: nat)
    using that by (rule fact-dvd)
  ultimately show ?thesis
    by simp
qed

lemma fact-div-fact-ffact-nat:
  fact n div fact (n - k) = ffact k n if k ≤ n
using that by (simp add: fact-div-fact-ffact)

lemma ffact-fact:
  ffact n (of-nat n) = (of-nat (fact n) :: 'a :: comm-ring-1)
  by (induct n) (simp-all add: algebra-simps ffact-Suc)

lemma ffact-add-diff-assoc:

```

```


$$(a - \text{of-nat } n) * \text{ffact } n a + \text{of-nat } n * \text{ffact } n a = a * \text{ffact } n a$$

for  $a :: 'a :: \text{comm-ring-1}$ 
by (simp add: algebra-simps)

```

lemma *mult-ffact*:

```


$$a * \text{ffact } n a = \text{ffact } (\text{Suc } n) a + \text{of-nat } n * \text{ffact } n a$$

for  $a :: 'a :: \text{comm-ring-1}$ 

```

proof –

```

have  $\text{ffact } (\text{Suc } n) a + \text{of-nat } n * (\text{ffact } n a) = (a - \text{of-nat } n) * (\text{ffact } n a) +$ 

$$\text{of-nat } n * (\text{ffact } n a)$$

using ffact-Suc-rev [of n] by auto
also have ... =  $a * \text{ffact } n a$  using ffact-add-diff-assoc by (simp add: algebra-simps)
finally show ?thesis by simp

```

qed

lemma *prod-ffact*:

```

fixes  $m :: 'a :: \{\text{ord, ring-1, comm-monoid-mult, comm-semiring-1-cancel}\}$ 
shows  $(\prod i = 0..<n. m - \text{of-nat } i) = \text{ffact } n m$ 

```

proof –

```

have inj-on  $(\lambda j. j - 1) \{1..n\}$ 
by (force intro: inj-on-diff-nat)
moreover have  $\{0..<n\} = (\lambda j. j - 1) ` \{1..n\}$ 

```

proof –

```

have  $i \in (\lambda j. j - 1) ` \{1..n\}$  if  $i \in \{0..<n\}$  for  $i$ 
using that by (auto intro: image-eqI[where x=i + 1])
from this show ?thesis by auto

```

qed

```

moreover have  $m - \text{of-nat } (i - 1) = m + 1 - \text{of-nat } n + \text{of-nat } (n - i)$  if  $i \in \{1..n\}$  for  $i$ 
using that by (simp add: of-nat-diff)
ultimately have  $(\prod i = 0..<n. m - \text{of-nat } i) = (\prod i = 1..n. m + 1 - \text{of-nat } n + \text{of-nat } (n - i))$ 
by (rule prod.reindex-cong)
from this show ?thesis

```

```

unfolding ffact-def by (simp only: pochhammer-prod-rev)

```

qed

lemma *prod-ffact-nat*:

```

fixes  $m :: \text{nat}$ 
shows  $(\prod i = 0..<n. m - i) = \text{ffact } n m$ 

```

proof cases

```

assume  $n \leq m$ 
have inj-on  $(\lambda j. j - 1) \{1..n\}$  by (force intro: inj-on-diff-nat)
moreover have  $\{0..<n\} = (\lambda j. j - 1) ` \{1..n\}$ 

```

proof –

```

have  $i \in (\lambda j. j - 1) ` \{1..n\}$  if  $i \in \{0..<n\}$  for  $i$ 
using that by (auto intro: image-eqI[where x=i + 1])

```

```

from this show ?thesis by auto
qed
ultimately have  $(\prod i = 0..n. m - i) = (\prod i = 1..n. (m + 1) - i)$ 
  by (auto intro: prod.reindex-cong[where l= $\lambda i. i - 1$ ])
from this ⟨ $n \leq m$ ⟩ show ?thesis
  unfolding ffact-def by (simp add: pochhammer-prod-rev)
next
assume  $\neg n \leq m$ 
from this show ?thesis by (auto simp add: ffact-nat-triv)
qed

lemma prod-rev-ffact:
fixes m :: 'a :: {ord, ring-1, comm-monoid-mult, comm-semiring-1-cancel}
shows  $(\prod i = 1..n. m - of\text{-}nat n + of\text{-}nat i) = ffact n m$ 
proof -
have inj-on  $(\lambda i. i + 1) \{0..n\}$  by simp
moreover have  $\{1..n\} = (\lambda i. i + 1) \{0..n\}$  by auto
moreover have  $m - of\text{-}nat n + of\text{-}nat (i + 1) = m + 1 - of\text{-}nat n + of\text{-}nat i$ 
for i by simp
ultimately have  $(\prod i = 1..n. m - of\text{-}nat n + of\text{-}nat i) = (\prod i = 0..n. m +$ 
 $1 - of\text{-}nat n + of\text{-}nat i)$ 
  by (rule prod.reindex-cong[where l= $\lambda i. i + 1$ ])
from this show ?thesis
  unfolding ffact-def by (simp only: pochhammer-prod)
qed

lemma prod-rev-ffact-nat:
fixes m :: nat
assumes n ≤ m
shows  $(\prod i = 1..n. m - n + i) = ffact n m$ 
proof -
have inj-on  $(\lambda i. i + 1) \{0..n\}$  by simp
moreover have  $\{1..n\} = (\lambda i. i + 1) \{0..n\}$  by auto
moreover have  $m - n + (i + 1) = m + 1 - n + i$  for i
  using ⟨ $n \leq m$ ⟩ by auto
ultimately have  $(\prod i = 1..n. m - n + i) = (\prod i = 0..n. m + 1 - n + i)$ 
  by (rule prod.reindex-cong)
from this show ?thesis
  unfolding ffact-def by (simp only: pochhammer-prod of-nat-id)
qed

lemma prod-rev-ffact-nat':
fixes m :: nat
assumes n ≤ m
shows  $\prod \{m - n + 1..m\} = ffact n m$ 
proof -
have inj-on  $(\lambda i. m - n + i) \{1..n\}$  by (auto intro: inj-onI)
moreover have  $\{m - n + 1..m\} = (\lambda i. m - n + i) \{1..n\}$ 

```

```

proof -
  have  $i \in (\lambda i. m + i - n) ` \{Suc 0..n\}$  if  $i \in \{m - n + 1..m\}$  for  $i$ 
    using that  $\langle n \leq m \rangle$  by (auto intro!: image-eqI[where x=i - (m - n)])
    with  $\langle n \leq m \rangle$  show ?thesis by auto
  qed
  moreover have  $m - n + i = m - n + i$  for  $i ..$ 
  ultimately have  $\prod \{m - n + (1::nat)..m\} = (\prod i = 1..n. m - n + i)$ 
    by (rule prod.reindex-cong)
  from this show ?thesis
    using  $\langle n \leq m \rangle$  by (simp only: prod-rev-ffact-nat)
  qed

lemma ffact-eq-fact-mult-binomial:
  ffact k n = fact k * (n choose k)
proof cases
  assume k ≤ n
  have ffact k n = fact n div fact (n - k)
    using  $\langle k \leq n \rangle$  by (simp add: fact-div-fact-ffact-nat)
  also have ... = fact k * (n choose k)
    using  $\langle k \leq n \rangle$  by (simp add: binomial-fact-lemma[symmetric])
  finally show ?thesis .
next
  assume ¬ k ≤ n
  from this ffact-nat-triv show ?thesis by force
qed

lemma of-nat-ffact:
  of-nat (ffact n m) = ffact n (of-nat m :: 'a :: comm-ring-1)
proof (induct n arbitrary: m)
  case 0
  then show ?case
    by simp
next
  case (Suc n)
  show ?case
  proof (cases m)
    case 0
    then show ?thesis
      by (simp add: ffact-Suc-nat ffact-Suc)
next
  case (Suc m)
  with Suc.hyps show ?thesis
    by (simp add: algebra-simps ffact-Suc-nat ffact-Suc)
qed
qed

lemma of-int-ffact:
  of-int (ffact n k) = ffact n (of-int k :: 'a :: comm-ring-1)
proof (induct n arbitrary: k)

```

```

case 0 then show ?case by simp
next
  case (Suc n k)
  then have of-int (ffact n (k - 1)) = ffact n (of-int (k - 1) :: 'a) .
  then show ?case
    by (simp add: ffact-Suc-nat ffact-Suc)
qed

lemma ffact-minus:
  fixes x :: 'a :: comm-ring-1
  shows ffact n (- x) = (- 1) ^ n * pochhammer x n
proof -
  have ffact n (- x) = pochhammer (- x + 1 - of-nat n) n
  unfolding ffact-def ..
  also have ... = pochhammer (- x - of-nat n + 1) n
  by (simp add: diff-add-eq)
  also have ... = (- 1) ^ n * pochhammer (- (- x)) n
  by (rule pochhammer-minus')
  also have ... = (- 1) ^ n * pochhammer x n by simp
  finally show ?thesis .
qed

```

Conversion of natural potences into falling factorials and back

```

lemma monomial-ffact:
  a ^ n = (∑ k = 0..n. of-nat (Stirling n k) * ffact k a)
  for a :: 'a :: comm-ring-1
proof (induct n)
  case 0 then show ?case by simp
next
  case (Suc n)
  then have a ^ Suc n = a * (∑ k = 0..n. of-nat (Stirling n k) * ffact k a)
  by simp
  also have ... = (∑ k = 0..n. of-nat (Stirling n k) * (a * ffact k a))
  by (simp add: sum-distrib-left algebra-simps)
  also have ... = (∑ k = 0..n. of-nat (Stirling n k) * ffact (Suc k) a) +
  (∑ k = 0..n. of-nat (Stirling n k) * (of-nat k * ffact k a))
  by (simp add: sum.distrib algebra-simps mult-ffact)
  also have ... = (∑ k = 0..Suc n. of-nat (Stirling n k) * ffact (Suc k) a) +
  (∑ k = 0..Suc n. of-nat ((Suc k) * (Stirling n (Suc k))) * (ffact (Suc k) a))
proof -
  have (∑ k = 0..n. of-nat (Stirling n k) * (of-nat k * ffact k a)) =
  (∑ k = 0..n+2. of-nat (Stirling n k) * (of-nat k * ffact k a)) by simp
  also have ... = (∑ k = Suc 0 .. Suc (Suc n). of-nat (Stirling n k) * (of-nat k
  * ffact k a))
  by (simp only: sum.atLeast-Suc-atMost [of 0 n + 2]) simp
  also have ... = (∑ k = 0 .. Suc n. of-nat (Stirling n (Suc k)) * (of-nat (Suc
  k) * ffact (Suc k) a))
  by (simp only: image-Suc-atLeastAtMost sum.shift-bounds-cl-Suc-ivl)
  also have ... = (∑ k = 0 .. Suc n. of-nat ((Suc k) * Stirling n (Suc k)) * ffact

```

```

(Suc k) a)
  by (simp only: of-nat-mult algebra-simps)
  finally have ( $\sum k = 0..n. \text{of-nat}(\text{Stirling } n k) * (\text{of-nat } k * \text{ffact } k a)$ ) =
    ( $\sum k = 0..n. \text{of-nat}(\text{Suc } k * \text{Stirling } n (\text{Suc } k)) * \text{ffact } (\text{Suc } k) a$ )
    by simp
  then show ?thesis by simp
qed
also have ... = ( $\sum k = 0..n. \text{of-nat}(\text{Stirling } (\text{Suc } n) (\text{Suc } k)) * \text{ffact } (\text{Suc } k)$ )
a)
  by (simp add: algebra-simps sum.distrib)
  also have ... = ( $\sum k = 0..n. \text{of-nat}(\text{Stirling } (\text{Suc } n) k) * \text{ffact } k a$ )
    by (simp only: image-Suc-atLeastAtMost sum.shift-bounds-cl-Suc-ivl)
  also have ... = ( $\sum k = 0..n. \text{of-nat}(\text{Stirling } (\text{Suc } n) k) * \text{ffact } k a$ )
    by (simp only: sum.atLeast-Suc-atMost [of 0 Suc n]) simp
  finally show ?case by simp
qed

lemma ffact-monomial:
  ffact n a = ( $\sum k = 0..n. (-1)^k * \text{of-nat}(\text{stirling } n k) * a^k$ )
  for a :: 'a :: comm-ring-1
proof (induct n)
  case 0 show ?case by simp
next
  case (Suc n)
  then have ffact (Suc n) a = (a - of-nat n) * ( $\sum k = 0..n. (-1)^k * \text{of-nat}(\text{stirling } n k) * a^k$ )
    by (simp add: ffact-Suc-rev)
  also have ... = ( $\sum k = 0..n. (-1)^k * \text{of-nat}(\text{stirling } n k) * a^k$ ) +
    ( $\sum k = 0..n. (-1)^k * \text{of-nat}(n * (\text{stirling } n k)) * a^k$ )
    by (simp only: diff-conv-add-uminus distrib-right) (simp add: sum-distrib-left
field-simps)
  also have ... = ( $\sum k = 0..n. (-1)^k * (\text{Suc } n - \text{Suc } k) * \text{of-nat}(\text{stirling } n k) * a^k$ ) +
    ( $\sum k = 0..n. (-1)^k * (\text{Suc } n - \text{Suc } k) * \text{of-nat}(n * \text{stirling } n (\text{Suc } k)) * a^k$ )
  proof -
    have ( $\sum k = 0..n. (-1)^k * \text{of-nat}(n * \text{stirling } n k) * a^k$ ) =
      ( $\sum k = 0..n. (-1)^k * (\text{Suc } n - k) * \text{of-nat}(n * \text{stirling } n k) * a^k$ )
      by (simp add: Suc-diff-le)
    also have ... = ( $\sum k = 0..n. (-1)^k * (\text{Suc } n - k) * \text{of-nat}(n * \text{stirling } n k) * a^k$ )
      by (simp add: sum.atLeast-Suc-atMost) (cases n; simp)
    also have ... = ( $\sum k = 0..n. (-1)^k * (\text{Suc } n - k) * \text{of-nat}(n * \text{stirling } n k) * a^k$ )
      by (simp only: sum.shift-bounds-cl-Suc-ivl)
    finally show ?thesis by simp
qed

```

```

also have ... = ( $\sum k = 0..n. (-1)^k (\text{Suc } n - \text{Suc } k) * \text{of-nat} (n * \text{stirling } n (\text{Suc } k) + \text{stirling } n k) * a^k$ )
  by (simp add: sum.distrib algebra-simps)
also have ... = ( $\sum k = 0..n. (-1)^k (\text{Suc } n - \text{Suc } k) * \text{of-nat} (\text{stirling} (\text{Suc } n) (\text{Suc } k)) * a^k$ )
  by (simp only: stirling.simps)
also have ... = ( $\sum k = \text{Suc } 0..n. (-1)^k (\text{Suc } n - k) * \text{of-nat} (\text{stirling} (\text{Suc } n) k) * a^k$ )
  by (simp only: sum.shift-bounds-cl-Suc-ivl)
also have ... = ( $\sum k = 0..\text{Suc } n. (-1)^k (\text{Suc } n - k) * \text{of-nat} (\text{stirling} (\text{Suc } n) k) * a^k$ )
  by (simp add: sum.atLeast-Suc-atMost)
finally show ?case .
qed

end

```

2 Some basic facts about discrete summation

```

theory Discrete-Summation
imports Main
begin

```

2.1 Auxiliary

```

lemma add-sum-orient:
   $\text{sum } f \{k..<j\} + \text{sum } f \{l..<k\} = \text{sum } f \{l..<k\} + \text{sum } f \{k..<j\}$ 
  by (fact add.commute)

lemma add-sum-int:
  fixes j k l :: int
  shows  $j < k \implies k < l \implies \text{sum } f \{j..<k\} + \text{sum } f \{k..<l\} = \text{sum } f \{j..<l\}$ 
  by (simp-all add: sum.union-inter [symmetric] ivl-disj-un)

```

2.2 The shift operator

```

definition  $\Delta :: ('b::ring-1 \Rightarrow 'a::ab-group-add) \Rightarrow \text{int} \Rightarrow 'a$ 
where
 $\Delta f k = f(\text{of-int}(k+1)) - f(\text{of-int} k)$ 

```

```

lemma  $\Delta$ -shift:
 $\Delta(\lambda k. l + f k) = \Delta f$ 
by (simp add:  $\Delta$ -def fun-eq-iff)

```

```

lemma  $\Delta$ -same-shift:
assumes  $\Delta f = \Delta g$ 
shows  $\exists l. \text{plus } l \circ f \circ \text{of-int} = g \circ \text{of-int}$ 
proof -

```

```

let ?F =  $\lambda k. f \text{ (of-int } k)$ 
let ?G =  $\lambda k. g \text{ (of-int } k)$ 
from assms have  $\bigwedge k. \Delta f \text{ (of-int } k) = \Delta g \text{ (of-int } k)$  by simp
then have  $k\text{-incr}: \bigwedge k. ?F(k+1) - ?G(k+1) = ?Fk - ?Gk$ 
  by (simp add:  $\Delta\text{-def algebra-simps}$ )
then have  $\bigwedge k. ?F((k-1)+1) - ?G((k-1)+1) =$ 
   $?F(k-1) - ?G(k-1)$ 
  by blast
then have  $k\text{-decr}: \bigwedge k. ?F(k-1) - ?G(k-1) = ?Fk - ?Gk$ 
  by simp
have  $\bigwedge k. ?Fk - ?Gk = ?F0 - ?G0$ 
proof -
  fix k
  show  $?Fk - ?Gk = ?F0 - ?G0$ 
    by (induct k rule: int-induct)
      (simp-all add: k-incr k-decr del: of-int-add of-int-diff of-int-0)
qed
then have  $\bigwedge k. (\text{plus} (?G0 - ?F0) \circ ?F)k = ?Gk$ 
  by (simp add: algebra-simps)
then have plus (?G0 - ?F0)  $\circ ?F = ?G ..$ 
then have plus (?G0 - ?F0)  $\circ f \circ \text{of-int} = g \circ \text{of-int}$ 
  by (simp only: comp-def)
then show ?thesis ..
qed

```

lemma $\Delta\text{-add:}$

```

 $\Delta(\lambda k. fk + gk)k = \Delta fk + \Delta gk$ 
by (simp add:  $\Delta\text{-def}$ )

```

lemma $\Delta\text{-factor:}$

```

 $\Delta(\lambda k. c * k)k = c$ 
by (simp add:  $\Delta\text{-def algebra-simps}$ )

```

2.3 The formal sum operator

definition $\Sigma :: ('b::ring-1 \Rightarrow 'a::ab-group-add) \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow 'a$
where

```

 $\Sigma f j l = (\text{if } j < l \text{ then sum } (f \circ \text{of-int}) \{j..<l\}$ 
   $\text{else if } j > l \text{ then } - \text{sum } (f \circ \text{of-int}) \{l..<j\}$ 
   $\text{else } 0)$ 

```

lemma $\Sigma\text{-same [simp]:}$

```

 $\Sigma f j j = 0$ 
by (simp add:  $\Sigma\text{-def}$ )

```

lemma $\Sigma\text{-positive:}$

```

 $j < l \implies \Sigma f j l = \text{sum } (f \circ \text{of-int}) \{j..<l\}$ 
by (simp add:  $\Sigma\text{-def}$ )

```

lemma Σ -negative:
 $j > l \implies \Sigma f j l = -\Sigma f l j$
by (*simp add: Σ -def*)

lemma Σ -comp-of-int:
 $\Sigma (f \circ \text{of-int}) = \Sigma f$
by (*simp add: Σ -def fun-eq-iff*)

lemma Σ -const:
 $\Sigma (\lambda k. c) j l = \text{of-int} (l - j) * c$
by (*simp add: Σ -def algebra-simps*)

lemma Σ -add:
 $\Sigma (\lambda k. f k + g k) j l = \Sigma f j l + \Sigma g j l$
by (*simp add: Σ -def sum.distrib*)

lemma Σ -factor:
 $\Sigma (\lambda k. c * f k) j l = (c :: 'a :: ring) * \Sigma (\lambda k. f k) j l$
by (*simp add: Σ -def sum-distrib-left*)

lemma Σ -concat:
 $\Sigma f j k + \Sigma f k l = \Sigma f j l$
by (*simp add: Σ -def algebra-simps add-sum-int*)
(*simp-all add: add-sum-orient [of $\lambda k. f (\text{of-int} k)$] k j l*
add-sum-orient [of $\lambda k. f (\text{of-int} k)$] j l k
add-sum-orient [of $\lambda k. f (\text{of-int} k)$] j k l add-sum-int)

lemma Σ -incr-upper:
 $\Sigma f j (l + 1) = \Sigma f j l + f (\text{of-int} l)$
proof –
have $\{l..<l+1\} = \{l\}$ **by** auto
then have $\Sigma f l (l + 1) = f (\text{of-int} l)$ **by** (*simp add: Σ -def*)
moreover have $\Sigma f j (l + 1) = \Sigma f j l + \Sigma f l (l + 1)$ **by** (*simp add: Σ -concat*)
ultimately show ?thesis **by** simp
qed

2.4 Fundamental lemmas: The relation between Δ and Σ

lemma Δ - Σ :
 $\Delta (\Sigma f j) = f$
proof
fix k
show $\Delta (\Sigma f j) k = f k$
by (*simp add: Δ -def Σ -incr-upper*)
qed

lemma Σ - Δ :
 $\Sigma (\Delta f) j l = f (\text{of-int} l) - f (\text{of-int} j)$
proof –

```

from Δ-Σ have Δ (Σ (Δ f) j) = Δ f .
then obtain k where plus k o Σ (Δ f) j o of-int = f o of-int
  by (blast dest: Δ-same-shift)
then have ⋀ q. f (of-int q) = k + Σ (Δ f) j q
  by (simp add: fun-eq-iff)
then show ?thesis by simp
qed

```

end

3 A barebone conversion for discrete summation

```

theory Summation-Conversion
imports Factorials Discrete-Summation
begin

```

Extensible theorem collection for solving summation problems
named-theorems summation rules for solving summation problems

declare

```

Σ-const [summation] Σ-add [summation]
Σ-factor [summation] monomial-ffact [summation]

```

lemma intervall-simps [summation]:

```

(∑ k:nat = 0..0. f k) = f 0
(∑ k:nat = 0..Suc n. f k) = f (Suc n) + (∑ k:nat = 0..n. f k)
by (simp-all add: add.commute)

```

lemma Δ-ffact:

Δ (ffact (Suc n)) k = of-nat (Suc n) * ffact n (of-int k :: 'a :: comm-ring-1)

proof (induct n)

```

case 0 then show ?case
  by (simp add: Δ-def ffact-Suc)

```

next

case (Suc n)

obtain m where m = Suc n by blast

have Δ (ffact (Suc m)) k =

```

ffact (Suc m) (of-int (k + 1)) - ffact (Suc m) (of-int k :: 'a)
  by (simp add: Δ-def)

```

also have ... = of-int (k + 1) * ffact m (of-int k)

```

- (ffact m (of-int k) * (of-int k - of-nat m))

```

```

using ffact-Suc-rev [of m (of-int k :: 'a :: comm-ring-1)]
  by (simp add: ac-simps ffact-Suc)

```

also have ... = (of-int k + 1 - of-int k + of-nat m) * ffact m (of-int k)

```

  by (simp add: algebra-simps)

```

also have ... = of-nat (Suc m) * ffact m (of-int k) by simp

also have ... = of-nat (Suc m) * ffact (Suc m - 1) (of-int k) by simp

finally show ?case by (simp only: <m = Suc n> diff-Suc-1)

qed

```

lemma Σ-ffact-divide [summation]:
  Σ (ffact n) j l =
    (ffact (Suc n)) (of-int l :: 'a :: {idom-divide, semiring-char-0}) - ffact (Suc n)
  (of-int j)) div of-nat (Suc n)
proof -
  have *: (of-nat (Suc n) * Σ (ffact n) j l) div of-nat (Suc n) = (Σ (ffact n) j l :: 'a)
    using of-nat-neq-0 [where ?'a = 'a] by simp
  have ffact (Suc n) (of-int l :: 'a) - ffact (Suc n) (of-int j) =
    Σ (λk. Δ (ffact (Suc n)) k) j l
    by (simp add: Σ-Δ)
  also have ... = Σ (λk. of-nat (Suc n) * ffact n (of-int k)) j l
    by (simp add: Δ-ffact)
  also have ... = of-nat (Suc n) * Σ (ffact n o of-int) j l
    by (simp add: Σ-factor comp-def)
  finally show ?thesis by (simp only: Σ-comp-of-int * of-nat-eq-0-iff)
qed

```

Various other rules

```

lemma of-int-coeff:
  (of-int l :: 'a::comm-ring-1) * numeral k = of-int (l * numeral k)
  by simp

lemmas nat-simps =
  add-0-left add-0-right add-Suc add-Suc-right
  mult-Suc mult-Suc-right mult-zero-left mult-zero-right
  One-nat-def of-nat-simps

lemmas of-int-pull-out =
  of-int-add [symmetric] of-int-diff [symmetric] of-int-mult [symmetric]
  of-int-coeff

lemma of-nat-coeff:
  (of-nat n :: 'a::comm-semiring-1) * numeral m = of-nat (n * numeral m)
  by (induct n) simp-all

lemmas of-nat-pull-out =
  of-nat-add [symmetric] of-nat-mult [symmetric] of-nat-coeff

lemmas nat-pull-in =
  nat-int-add

lemmas of-int-pull-in =
  of-int-pull-out [symmetric] add-divide-distrib diff-divide-distrib of-int-power
  of-int-numeral of-int-neg-numeral times-divide-eq-left [symmetric]

Special for nat

definition lift-nat :: (nat ⇒ nat) ⇒ int ⇒ int

```

```

where
lift-nat f = int o f o nat

definition Σ-nat :: (nat ⇒ nat) ⇒ nat ⇒ nat ⇒ nat ( $\langle \Sigma_N \rangle$ )
where
[summation]:  $\Sigma_N f m n = \text{nat} (\Sigma (\text{lift-nat } f) (\text{int } m) (\text{int } n))$ 

definition pos-id :: int ⇒ int
where
pos-id k = (if k < 0 then 0 else k)

lemma Σ-pos-id [summation]:
 $0 \leq k \implies 0 \leq l \implies \Sigma (\lambda r. f (\text{pos-id } r)) k l = \Sigma f k l$ 
by (simp add: Σ-def pos-id-def)

lemma [summation]:
(0::int) ≤ 0
(0::int) ≤ 1
(0::int) ≤ numeral m
(0::int) ≤ int n
by simp-all

lemma [summation]:
lift-nat (λn. m) = (λk. int m)
by (simp add: lift-nat-def fun-eq-iff)

lemma [summation]:
lift-nat (λn. n) = pos-id
by (simp add: lift-nat-def fun-eq-iff pos-id-def)

lemma [summation]:
lift-nat (λn. f n + g n) = (λk. lift-nat f k + lift-nat g k)
by (simp add: lift-nat-def fun-eq-iff)

lemma [summation]:
lift-nat (λn. m * f n) = (λk. int m * lift-nat f k)
by (simp add: lift-nat-def fun-eq-iff)

lemma [summation]:
lift-nat (λn. f n * m) = (λk. lift-nat f k * int m)
by (simp add: lift-nat-def fun-eq-iff)

lemma [summation]:
lift-nat (λn. f n ^ m) = (λk. lift-nat f k ^ m)
by (simp add: lift-nat-def fun-eq-iff)

```

Generic conversion

```

ML ‹
signature SUMMATION =

```

```

sig
  val conv: Proof.context -> conv
end

structure Summation : SUMMATION =
struct

  val simps2 = @{thms Stirling.simps ffact-0 ffact-Suc nat-simps};
  val simpset3 =
    @{context}
    |> fold Simplifier.add-simp @{thms field-simps}
    |> Simplifier simpset-of;
  val simps4 = @{thms of-int-pull-out of-nat-pull-out nat-pull-in};
  val simps6 = @{thms of-int-pull-in};

  fun conv ctxt =
    let
      val ctxt1 =
        ctxt
        |> put-simpset HOL-basic-ss
        |> fold Simplifier.add-simp (Named-Theorems.get ctxt @{named-theorems summation})
      val ctxt2 =
        ctxt
        |> put-simpset HOL-basic-ss
        |> fold Simplifier.add-simp simps2
      val ctxt3 =
        ctxt
        |> put-simpset simpset3
      val ctxt4 =
        ctxt
        |> put-simpset HOL-basic-ss
        |> fold Simplifier.add-simp simps4
      val semiring-conv-base = Semiring-Normalizer.semiring-normalize-conv ctxt
        val semiring-conv = Conv.arg-conv (Conv.arg1-conv (Conv.arg-conv semiring-conv-base))
          else-conv Conv.arg1-conv (Conv.arg-conv semiring-conv-base)
          else-conv semiring-conv-base
      val ctxt6 =
        ctxt
        |> put-simpset HOL-basic-ss
        |> fold Simplifier.add-simp simps6
    in
      Simplifier.rewrite ctxt1
      then-conv Simplifier.rewrite ctxt2
      then-conv Simplifier.rewrite ctxt3
      then-conv Simplifier.rewrite ctxt4
      then-conv semiring-conv
      then-conv Simplifier.rewrite ctxt6
    end
end

```

```

end

end
>

hide-fact (open) nat-simps of-int-pull-out of-int-pull-in

end

```

4 Simple examples

```

theory Examples
imports Summation-Conversion
begin

ML ‹
  Summation.conv @{context}
  @{cterm Σ (λq::rat. q ^ Suc (Suc 0)) + 3) 0 j}
›

ML ‹
  Summation.conv @{context}
  @{cterm Σ (λx::real. x ^ Suc (Suc 0)) + 3) 0 j}
›

ML ‹
  Summation.conv @{context}
  @{cterm Σ (λk::int. k ^ Suc (Suc 0)) + 3) 0 j}
›

ML ‹
  Summation.conv @{context}
  @{cterm ΣN (λn::nat. n ^ Suc (Suc 0)) + 3) 0 m}
›

end

```