

Discrete Summation

Florian Haftmann
with contributions by Amine Chaieb

March 8, 2026

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* \Rightarrow *'a::comm-semiring-1-cancel* \Rightarrow *'a*
where *ffact n a = pochhammer (a + 1 - of-nat n) n*

lemma *ffact-0* [*simp*]:
ffact 0 = ($\lambda x. 1$)
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-rem-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 $\dots = a * \text{ffact } n \ a$ **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 $\text{inj-on } (\lambda j. j - 1) \ \{1..n\}$

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 $\text{inj-on } (\lambda j. j - 1) \ \{1..n\}$ **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  $\langle n \leq m \rangle$  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  $(\prod i = 1..n. m - \text{of-nat } n + \text{of-nat } i) = \text{ffact } n \ m$ 
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  $m - \text{of-nat } n + \text{of-nat } (i + 1) = m + 1 - \text{of-nat } n + \text{of-nat } i$ 
  for i by simp
  ultimately have  $(\prod i = 1..n. m - \text{of-nat } n + \text{of-nat } i) = (\prod i = 0..<n. m + 1 - \text{of-nat } n + \text{of-nat } i)$ 
    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  $n \leq m$ 
  shows  $(\prod i = 1..n. m - n + i) = \text{ffact } n \ m$ 
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  $\langle n \leq m \rangle$  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  $n \leq m$ 
  shows  $\prod \{m - n + 1..m\} = \text{ffact } n \ m$ 
proof -
  have inj-on  $(\lambda i. m - n + i) \{1::nat..n\}$  by (auto intro: inj-onI)
  moreover have  $\{m - n + 1..m\} = (\lambda i. m - n + i) \{1::nat..n\}$ 

```

proof –
have $i \in (\lambda i. m + i - n) \cdot \{ \text{Suc } 0..n \}$ **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** $\prod \{m - n + (1::\text{nat})..m\} = (\prod i = 1..n. m - n + i)$
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 $k \leq n$

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** $... = \text{fact } k * (n \text{ choose } k)$

using $\langle k \leq n \rangle$ **by** (*simp add*: *binomial-fact-lemma*[*symmetric*])

finally **show** *?thesis* .

next

assume $\neg k \leq n$

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

lemma *ffact-monomial:*

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

also have ... = $(\sum k = 0..n. (-1)^{\wedge}(Suc\ n - Suc\ k) * of\text{-}nat\ (n * stirling\ n\ (Suc\ k) + stirling\ n\ k) * a^{\wedge} Suc\ k)$
by (*simp add: sum.distrib algebra-simps*)
also have ... = $(\sum k = 0..n. (-1)^{\wedge}(Suc\ n - Suc\ k) * of\text{-}nat\ (stirling\ (Suc\ n)\ (Suc\ k)) * a^{\wedge} Suc\ k)$
by (*simp only: stirling.simps*)
also have ... = $(\sum k = Suc\ 0..Suc\ n. (-1)^{\wedge}(Suc\ n - k) * of\text{-}nat\ (stirling\ (Suc\ n)\ k) * a^{\wedge} k)$
by (*simp only: sum.shift-bounds-cl-Suc-ivl*)
also have ... = $(\sum k = 0..Suc\ n. (-1)^{\wedge}(Suc\ n - k) * of\text{-}nat\ (stirling\ (Suc\ n)\ k) * a^{\wedge} k)$
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*:
 $sum\ f\ \{k..<j\} + sum\ f\ \{l..<k\} = sum\ f\ \{l..<k\} + sum\ f\ \{k..<j\}$
by (*fact add.commute*)

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

2.2 The shift operator

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

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

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

let $?F = \lambda k. f (of-int\ k)$
let $?G = \lambda k. g (of-int\ k)$
from *assms* **have** $\bigwedge k. \Delta f (of-int\ k) = \Delta g (of-int\ k)$ **by** *simp*
then have $k-incr: \bigwedge k. ?F (k + 1) - ?G (k + 1) = ?F\ k - ?G\ k$
by (*simp add: Δ -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 $k-decr: \bigwedge k. ?F (k - 1) - ?G (k - 1) = ?F\ k - ?G\ k$
by *simp*
have $\bigwedge k. ?F\ k - ?G\ k = ?F\ 0 - ?G\ 0$
proof -
fix k
show $?F\ k - ?G\ k = ?F\ 0 - ?G\ 0$
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 $\bigwedge k. (plus (?G\ 0 - ?F\ 0) \circ ?F) k = ?G\ k$
by (*simp add: algebra-simps*)
then have $plus (?G\ 0 - ?F\ 0) \circ ?F = ?G\ ..$
then have $plus (?G\ 0 - ?F\ 0) \circ f \circ of-int = g \circ of-int$
by (*simp only: comp-def*)
then show *?thesis ..*
qed

lemma Δ -*add*:
 $\Delta (\lambda k. f\ k + g\ k) k = \Delta f\ k + \Delta g\ k$
by (*simp add: Δ -def*)

lemma Δ -*factor*:
 $\Delta (\lambda k. c * k) k = c$
by (*simp add: Δ -def algebra-simps*)

2.3 The formal sum operator

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

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

lemma Σ -*same* [*simp*]:
 $\Sigma f\ j\ j = 0$
by (*simp add: Σ -def*)

lemma Σ -*positive*:
 $j < l \implies \Sigma f\ j\ l = sum\ (f \circ of-int)\ \{j..<l\}$
by (*simp add: Σ -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  $\Delta$ - $\Sigma$  have  $\Delta (\Sigma (\Delta f) j) = \Delta f$  .
then obtain  $k$  where  $plus\ k \circ \Sigma (\Delta f) j \circ of-int = f \circ of-int$ 
  by (blast dest:  $\Delta$ -same-shift)
then have  $\bigwedge q. f (of-int\ q) = k + \Sigma (\Delta f) j\ q$ 
  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

```

```

   $\Sigma$ -const [summation]  $\Sigma$ -add [summation]
   $\Sigma$ -factor [summation] monomial-ffact [summation]

```

```

lemma intervall-simps [summation]:

```

```

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

```

```

lemma  $\Delta$ -ffact:

```

```

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

```

```

proof (induct n)

```

```

  case 0 then show ?case
    by (simp add:  $\Delta$ -def ffact-Suc)

```

```

next

```

```

  case (Suc n)
    obtain  $m$  where  $m = Suc\ n$  by blast
    have  $\Delta (ffact (Suc\ m))\ k =$ 
       $ffact (Suc\ m) (of-int (k + 1)) - ffact (Suc\ m) (of-int k :: 'a)$ 
      by (simp add:  $\Delta$ -def)
    also have  $\dots = of-int (k + 1) * ffact\ m (of-int\ k)$ 
       $- (ffact\ m (of-int\ k) * (of-int\ k - of-nat\ m))$ 
      using ffact-Suc-rev [of m (of-int k :: 'a :: comm-ring-1)]
      by (simp add: ac-simps ffact-Suc)
    also have  $\dots = (of-int\ k + 1 - of-int\ k + of-nat\ m) * ffact\ m (of-int\ k)$ 
      by (simp add: algebra-simps)
    also have  $\dots = of-nat (Suc\ m) * ffact\ m (of-int\ k)$  by simp
    also have  $\dots = of-nat (Suc\ m) * ffact (Suc\ m - 1) (of-int\ k)$  by simp
    finally show ?case by (simp only:  $\langle m = Suc\ n \rangle$  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 ($\lambda k. \Delta$ (ffact (Suc n)) k) j l

by (simp add: Σ - Δ)

also have ... = Σ ($\lambda k. \text{of-nat (Suc } n) * \text{ffact } n \text{ (of-int } k)$) j l

by (simp add: Δ -ffact)

also have ... = of-nat (Suc n) * Σ (ffact n \circ 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 \Rightarrow nat) \Rightarrow int \Rightarrow int

where

$$\text{lift-nat } f = \text{int} \circ f \circ \text{nat}$$

definition $\Sigma\text{-nat} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \langle \langle \Sigma_{\mathbb{N}} \rangle \rangle$

where

$$[\text{summation}]: \Sigma_{\mathbb{N}} f m n = \text{nat} (\Sigma (\text{lift-nat } f) (\text{int } m) (\text{int } n))$$

definition $\text{pos-id} :: \text{int} \Rightarrow \text{int}$

where

$$\text{pos-id } k = (\text{if } k < 0 \text{ then } 0 \text{ else } k)$$

lemma $\Sigma\text{-pos-id}$ $[\text{summation}]$:

$$0 \leq k \implies 0 \leq l \implies \Sigma (\lambda r. f (\text{pos-id } r)) k l = \Sigma f k l$$

by (*simp add: $\Sigma\text{-def pos-id-def}$*)

lemma $[\text{summation}]$:

$$\begin{aligned} (0::\text{int}) &\leq 0 \\ (0::\text{int}) &\leq 1 \\ (0::\text{int}) &\leq \text{numeral } m \\ (0::\text{int}) &\leq \text{int } n \end{aligned}$$

by *simp-all*

lemma $[\text{summation}]$:

$$\text{lift-nat } (\lambda n. m) = (\lambda k. \text{int } m)$$

by (*simp add: lift-nat-def fun-eq-iff*)

lemma $[\text{summation}]$:

$$\text{lift-nat } (\lambda n. n) = \text{pos-id}$$

by (*simp add: lift-nat-def fun-eq-iff pos-id-def*)

lemma $[\text{summation}]$:

$$\text{lift-nat } (\lambda n. f n + g n) = (\lambda k. \text{lift-nat } f k + \text{lift-nat } g k)$$

by (*simp add: lift-nat-def fun-eq-iff*)

lemma $[\text{summation}]$:

$$\text{lift-nat } (\lambda n. m * f n) = (\lambda k. \text{int } m * \text{lift-nat } f k)$$

by (*simp add: lift-nat-def fun-eq-iff*)

lemma $[\text{summation}]$:

$$\text{lift-nat } (\lambda n. f n * m) = (\lambda k. \text{lift-nat } f k * \text{int } m)$$

by (*simp add: lift-nat-def fun-eq-iff*)

lemma $[\text{summation}]$:

$$\text{lift-nat } (\lambda n. f n \hat{=} m) = (\lambda k. \text{lift-nat } f k \hat{=} m)$$

by (*simp add: lift-nat-def fun-eq-iff*)

Generic conversion

ML \langle

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_simps3 =
    @{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_simps3
      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 semir-
ing_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  $\Sigma$  ( $\lambda q::\text{rat. } q \wedge \text{Suc (Suc (Suc 0))} + 3$ ) 0 j}
>

ML <
  Summation.conv @{context}
  @{cterm  $\Sigma$  ( $\lambda x::\text{real. } x \wedge \text{Suc (Suc (Suc 0))} + 3$ ) 0 j}
>

ML <
  Summation.conv @{context}
  @{cterm  $\Sigma$  ( $\lambda k::\text{int. } k \wedge \text{Suc (Suc (Suc 0))} + 3$ ) 0 j}
>

ML <
  Summation.conv @{context}
  @{cterm  $\Sigma_{\mathbb{N}}$  ( $\lambda n::\text{nat. } n \wedge \text{Suc (Suc (Suc 0))} + 3$ ) 0 m}
>

end

```