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** $n > 0$
  ⟨*proof*⟩

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

lemma *ffact-0* [*simp*]:
  $ffact\ 0 = (\lambda x.\ 1)$
  ⟨*proof*⟩

lemma *ffact-Suc*:
  $ffact\ (Suc\ n)\ a = a * ffact\ n\ (a - 1)$
    for $a :: 'a :: comm\text{-}ring\text{-}1$
  ⟨*proof*⟩

lemma *ffact-Suc-rev*:
  $ffact\ (Suc\ n)\ m = (m - of\text{-}nat\ n) * ffact\ n\ m$
    for $m :: 'a :: \{comm\text{-}semiring\text{-}1\text{-}cancel,\ ab\text{-}group\text{-}add\}$
⟨*proof*⟩

lemma *ffact-nat-triv*:
  $ffact\ n\ m = 0$ **if** $m < n$

⟨*proof*⟩

lemma *ffact-Suc-nat*:
ffact (*Suc n*) *m* = *m* ∗ *ffact n* (*m* − *1*)
for *m* :: *nat*
⟨*proof*⟩

lemma *ffact-Suc-rev-nat*:
ffact (*Suc n*) *m* = (*m* − *n*) ∗ *ffact n m*
⟨*proof*⟩

lemma *fact-div-fact-ffact*:
fact n div fact m = *ffact* (*n* − *m*) *n* **if** *m* ≤ *n*
⟨*proof*⟩

lemma *fact-div-fact-ffact-nat*:
fact n div fact (*n* − *k*) = *ffact k n* **if** *k* ≤ *n*
⟨*proof*⟩

lemma *ffact-fact*:
ffact n (*of-nat n*) = (*of-nat* (*fact n*) :: *'a* :: *comm-ring-1*)
⟨*proof*⟩

lemma *ffact-add-diff-assoc*:
(*a* − *of-nat n*) ∗ *ffact n a* + *of-nat n* ∗ *ffact n a* = *a* ∗ *ffact n a*
for *a* :: *'a* :: *comm-ring-1*
⟨*proof*⟩

lemma *mult-ffact*:
a ∗ *ffact n a* = *ffact* (*Suc n*) *a* + *of-nat n* ∗ *ffact n a*
for *a* :: *'a* :: *comm-ring-1*
⟨*proof*⟩

lemma *prod-ffact*:
fixes *m* :: *'a* :: {*ord*, *ring-1*, *comm-monoid-mult*, *comm-semiring-1-cancel*}
shows (∏ *i* = *0*..<*n*. *m* − *of-nat i*) = *ffact n m*
⟨*proof*⟩

lemma *prod-ffact-nat*:
fixes *m* :: *nat*
shows (∏ *i* = *0*..<*n*. *m* − *i*) = *ffact n m*
⟨*proof*⟩

lemma *prod-rev-ffact*:
fixes *m* :: *'a* :: {*ord*, *ring-1*, *comm-monoid-mult*, *comm-semiring-1-cancel*}
shows (∏ *i* = *1*..*n*. *m* − *of-nat n* + *of-nat i*) = *ffact n m*
⟨*proof*⟩

lemma *prod-rev-ffact-nat*:
fixes m :: *nat*
assumes $n \leq m$
shows $(\prod i = 1..n.\ m - n + i) = ffact\ n\ m$
⟨*proof*⟩

lemma *prod-rev-ffact-nat′*:
fixes m :: *nat*
assumes $n \leq m$
shows $\prod \{m - n + 1..m\} = ffact\ n\ m$
⟨*proof*⟩

lemma *ffact-eq-fact-mult-binomial*:
$ffact\ k\ n = fact\ k * (n\ choose\ k)$
⟨*proof*⟩

lemma *of-nat-ffact*:
$of\text{-}nat\ (ffact\ n\ m) = ffact\ n\ (of\text{-}nat\ m :: 'a :: comm\text{-}ring\text{-}1)$
⟨*proof*⟩

lemma *of-int-ffact*:
$of\text{-}int\ (ffact\ n\ k) = ffact\ n\ (of\text{-}int\ k :: 'a :: comm\text{-}ring\text{-}1)$
⟨*proof*⟩

lemma *ffact-minus*:
fixes $x :: 'a :: comm\text{-}ring\text{-}1$
shows $ffact\ n\ (-\ x) = (-\ 1)\ \hat{}\ n * pochhammer\ x\ n$
⟨*proof*⟩

Conversion of natural potences into falling factorials and back

lemma *monomial-ffact*:
$a\ \hat{}\ n = (\sum k = 0..n.\ of\text{-}nat\ (Stirling\ n\ k) * ffact\ k\ a)$
for $a :: 'a :: comm\text{-}ring\text{-}1$
⟨*proof*⟩

lemma *ffact-monomial*:
$ffact\ n\ a = (\sum k = 0..n.\ (-\ 1)\ \hat{}\ (n - k) * of\text{-}nat\ (stirling\ n\ k) * a\ \hat{}\ k)$
for $a :: 'a :: comm\text{-}ring\text{-}1$
⟨*proof*⟩

end

2 Some basic facts about discrete summation

theory *Discrete-Summation*
imports *Main*
begin

2.1 Auxiliary

lemma *add-sum-orient*:
$sum\ f\ \{k..<j\} + sum\ f\ \{l..<k\} = sum\ f\ \{l..<k\} + sum\ f\ \{k..<j\}$
⟨*proof*⟩

lemma *add-sum-int*:
fixes $j\ k\ l :: int$
shows $j < k \Longrightarrow k < l \Longrightarrow$
$sum\ f\ \{j..<k\} + sum\ f\ \{k..<l\} = sum\ f\ \{j..<l\}$
⟨*proof*⟩

2.2 The shift operator

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

lemma Δ-*shift*:
$\Delta\ (\lambda k.\ l + f\ k) = \Delta\ f$
⟨*proof*⟩

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*⟩

lemma Δ-*add*:
$\Delta\ (\lambda k.\ f\ k + g\ k)\ k = \Delta\ f\ k + \Delta\ g\ k$
⟨*proof*⟩

lemma Δ-*factor*:
$\Delta\ (\lambda k.\ c * k)\ k = c$
⟨*proof*⟩

2.3 The formal sum operator

definition $\Sigma :: ('b::ring\text{-}1 \Rightarrow 'a::ab\text{-}group\text{-}add) \Rightarrow int \Rightarrow int \Rightarrow 'a$
where
$\Sigma\ f\ j\ l = (if\ j < l\ then\ sum\ (f \circ of\text{-}int)\ \{j..<l\}$
$else\ if\ j > l\ then\ -\ sum\ (f \circ of\text{-}int)\ \{l..<j\}$
$else\ 0)$

lemma Σ-*same* [*simp*]:
$\Sigma\ f\ j\ j = 0$
⟨*proof*⟩

lemma Σ-*positive*:
$j < l \Longrightarrow \Sigma\ f\ j\ l = sum\ (f \circ of\text{-}int)\ \{j..<l\}$
⟨*proof*⟩

lemma Σ-*negative*:
$j > l \Longrightarrow \Sigma\ f\ j\ l = -\ \Sigma\ f\ l\ j$
$\langle proof \rangle$

lemma Σ-*comp-of-int*:
$\Sigma\ (f \circ \textit{of-int}) = \Sigma\ f$
$\langle proof \rangle$

lemma Σ-*const*:
$\Sigma\ (\lambda k.\ c)\ j\ l = \textit{of-int}\ (l - j) * c$
$\langle proof \rangle$

lemma Σ-*add*:
$\Sigma\ (\lambda k.\ f\ k + g\ k)\ j\ l = \Sigma\ f\ j\ l + \Sigma\ g\ j\ l$
$\langle proof \rangle$

lemma Σ-*factor*:
$\Sigma\ (\lambda k.\ c * f\ k)\ j\ l = (c::'a::ring) * \Sigma\ (\lambda k.\ f\ k)\ j\ l$
$\langle proof \rangle$

lemma Σ-*concat*:
$\Sigma\ f\ j\ k + \Sigma\ f\ k\ l = \Sigma\ f\ j\ l$
$\langle proof \rangle$

lemma Σ-*incr-upper*:
$\Sigma\ f\ j\ (l + 1) = \Sigma\ f\ j\ l + f\ (\textit{of-int}\ l)$
$\langle proof \rangle$

2.4 Fundamental lemmas: The relation between Δ and Σ

lemma Δ-Σ:
$\Delta\ (\Sigma\ f\ j) = f$
$\langle proof \rangle$

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

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*]:
$(\sum k::nat = 0..0.\ f\ k) = f\ 0$
$(\sum k::nat = 0..Suc\ n.\ f\ k) = f\ (Suc\ n) + (\sum k::nat = 0..n.\ f\ k)$
⟨*proof*⟩

lemma Δ-*ffact*:
$\Delta\ (ffact\ (Suc\ n))\ k = of\text{-}nat\ (Suc\ n) * ffact\ n\ (of\text{-}int\ k :: 'a :: comm\text{-}ring\text{-}1)$
⟨*proof*⟩

lemma Σ-*ffact-divide* [*summation*]:
$\Sigma\ (ffact\ n)\ j\ l =$
$(ffact\ (Suc\ n)\ (of\text{-}int\ l :: 'a :: \{idom\text{-}divide,\ semiring\text{-}char\text{-}0\}) - ffact\ (Suc\ n)$
$(of\text{-}int\ j))\ div\ of\text{-}nat\ (Suc\ n)$
⟨*proof*⟩

Various other rules

lemma *of-int-coeff*:
$(of\text{-}int\ l :: 'a::comm\text{-}ring\text{-}1) * numeral\ k = of\text{-}int\ (l * numeral\ k)$
⟨*proof*⟩

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*:
$(of\text{-}nat\ n :: 'a::comm\text{-}semiring\text{-}1) * numeral\ m = of\text{-}nat\ (n * numeral\ m)$
⟨*proof*⟩

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
  lift-nat f = *int* ∘ *f* ∘ *nat*

definition Σ-*nat* :: (*nat* ⇒ *nat*) ⇒ *nat* ⇒ *nat* ⇒ *nat* (‹$\Sigma_{\mathbb{N}}$›)
where
  [*summation*]: $\Sigma_{\mathbb{N}}$ *f m n* = *nat* (Σ (*lift-nat f*) (*int m*) (*int n*))

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

lemma Σ-*pos-id* [*summation*]:
  $0 \leq k \Longrightarrow 0 \leq l \Longrightarrow \Sigma\ (\lambda r.\ f\ (pos\text{-}id\ r))\ k\ l = \Sigma\ f\ k\ l$
  ⟨*proof*⟩

lemma [*summation*]:
  (*0*::*int*) ≤ *0*
  (*0*::*int*) ≤ *1*
  (*0*::*int*) ≤ *numeral m*
  (*0*::*int*) ≤ *int n*
  ⟨*proof*⟩

lemma [*summation*]:
  lift-nat (λ*n*. *m*) = (λ*k*. *int m*)
  ⟨*proof*⟩

lemma [*summation*]:
  lift-nat (λ*n*. *n*) = *pos-id*
  ⟨*proof*⟩

lemma [*summation*]:
  lift-nat (λ*n*. *f n* + *g n*) = (λ*k*. *lift-nat f k* + *lift-nat g k*)
  ⟨*proof*⟩

lemma [*summation*]:
  lift-nat (λ*n*. *m* ∗ *f n*) = (λ*k*. *int m* ∗ *lift-nat f k*)
  ⟨*proof*⟩

lemma [*summation*]:
  lift-nat (λ*n*. *f n* ∗ *m*) = (λ*k*. *lift-nat f k* ∗ *int m*)
  ⟨*proof*⟩

lemma [*summation*]:
  lift-nat (λ*n*. *f n* ^ *m*) = (λ*k*. *lift-nat f k* ^ *m*)
  ⟨*proof*⟩

Generic conversion

⟨*ML*⟩

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

end

4 Simple examples

theory *Examples*
imports *Summation-Conversion*
begin

⟨*ML*⟩

end