

# The Falling Factorial of a Sum

Lukas Bulwahn

June 16, 2019

## Abstract

This entry shows that the falling factorial of a sum can be computed with an expression using binomial coefficients and the falling factorial of its summands. The entry provides three different proofs: a combinatorial proof, an induction proof and an algebraic proof using the Vandermonde identity.

The three formalizations try to follow their informal presentations from a Mathematics Stack Exchange page [1, 2, 3, 4] as close as possible. The induction and algebraic formalization end up to be very close to their informal presentation, whereas the combinatorial proof first requires the introduction of list interleavings, and significant more detail than its informal presentation.

## Contents

<b>1</b>	<b>Proving Falling Factorial of a Sum with Combinatorics</b>	<b>1</b>
1.1	Preliminaries . . . . .	2
1.1.1	Addition to Factorials Theory . . . . .	2
1.2	Interleavings of Two Lists . . . . .	2
1.3	Cardinality of Distinct Fixed-Length Lists from a Union of Two Sets . . . . .	5
<b>2</b>	<b>Proving Falling Factorial of a Sum with Induction</b>	<b>9</b>
<b>3</b>	<b>Proving Falling Factorial of a Sum with Vandermonde Identity</b>	<b>13</b>
<b>4</b>	<b>Note on Copyright Licensing</b>	<b>13</b>

## 1 Proving Falling Factorial of a Sum with Combinatorics

```
theory Falling-Factorial-Sum-Combinatorics
imports
```

*Discrete-Summation.Factorials*  
*Card-Partitions.Injectivity-Solver*

**begin**

## 1.1 Preliminaries

### 1.1.1 Addition to Factorials Theory

**lemma** *card-lists-distinct-length-eq*:

**assumes** *finite A*

**shows**  $\text{card } \{xs. \text{length } xs = n \wedge \text{distinct } xs \wedge \text{set } xs \subseteq A\} = \text{ffact } n (\text{card } A)$

**proof** *cases*

**assume**  $n \leq \text{card } A$

**have**  $\text{card } \{xs. \text{length } xs = n \wedge \text{distinct } xs \wedge \text{set } xs \subseteq A\} = \prod \{\text{card } A - n + 1 .. \text{card } A\}$

**using**  $\langle \text{finite } A \rangle \langle n \leq \text{card } A \rangle$  **by** (*rule card-lists-distinct-length-eq*)

**also have**  $\dots = \text{ffact } n (\text{card } A)$

**using**  $\langle n \leq \text{card } A \rangle$  **by** (*simp add: prod-rev-ffact-nat'[symmetric]*)

**finally show** *?thesis* .

**next**

**assume**  $\neg n \leq \text{card } A$

**from this**  $\langle \text{finite } A \rangle$  **have**  $\forall xs. \text{length } xs = n \wedge \text{distinct } xs \wedge \text{set } xs \subseteq A \longrightarrow \text{False}$

**by** (*metis card-mono distinct-card*)

**from this** **have** *eq-empty*:  $\{xs. \text{length } xs = n \wedge \text{distinct } xs \wedge \text{set } xs \subseteq A\} = \{\}$

**using**  $\langle \text{finite } A \rangle$  **by** *auto*

**from**  $\langle \neg n \leq \text{card } A \rangle$  **show** *?thesis*

**by** (*simp add: ffact-nat-triv eq-empty*)

**qed**

## 1.2 Interleavings of Two Lists

**inductive** *interleavings* :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  'a list  $\Rightarrow$  bool

**where**

*interleavings*  $\square$  *ys ys*

| *interleavings xs*  $\square$  *xs*

| *interleavings xs ys zs*  $\Longrightarrow$  *interleavings (x#xs) ys (x#zs)*

| *interleavings xs ys zs*  $\Longrightarrow$  *interleavings xs (y#ys) (y#zs)*

**lemma** *interleaving-Nil-implies-eq1*:

**assumes** *interleavings xs ys zs*

**assumes**  $xs = \square$

**shows**  $ys = zs$

**using** *assms* **by** (*induct rule: interleavings.induct*) *auto*

**lemma** *interleaving-Nil-iff1*:

*interleavings*  $\square$  *ys zs*  $\longleftrightarrow$  ( $ys = zs$ )

**using** *interleaving-Nil-implies-eq1*

**by** (*auto simp add: interleavings.intros(1)*)

**lemma** *interleaving-Nil-implies-eq2*:  
**assumes** *interleavings xs ys zs*  
**assumes**  $ys = []$   
**shows**  $xs = zs$   
**using** *assms* **by** (*induct rule: interleavings.induct*) *auto*

**lemma** *interleaving-Nil-iff2*:  
*interleavings xs [] zs*  $\longleftrightarrow$  ( $xs = zs$ )  
**using** *interleaving-Nil-implies-eq2*  
**by** (*auto simp add: interleavings.intros(2)*)

**lemma** *interleavings-Cons*:  
 $\{zs. interleavings (x\#xs) (y\#ys) zs\} =$   
 $\{x\#zs | zs. interleavings xs (y\#ys) zs\} \cup \{y\#zs | zs. interleavings (x\#xs) ys zs\}$   
**(is**  $?S = ?expr$ **)**  
**proof**  
**show**  $?S \subseteq ?expr$   
**by** (*auto elim: interleavings.cases*)  
**next**  
**show**  $?expr \subseteq ?S$   
**by** (*auto intro: interleavings.intros*)  
**qed**

**lemma** *interleavings-filter*:  
**assumes**  $X \cap Y = \{\}$  *set*  $zs \subseteq X \cup Y$   
**shows** *interleavings*  $[z \leftarrow zs . z \in X]$   $[z \leftarrow zs . z \in Y]$  *zs*  
**using** *assms* **by** (*induct zs*) (*auto intro: interleavings.intros*)

**lemma** *interleavings-filter-eq1*:  
**assumes** *interleavings xs ys zs*  
**assumes**  $(\forall x \in set\ xs. P\ x) \wedge (\forall y \in set\ ys. \neg P\ y)$   
**shows** *filter*  $P\ zs = xs$   
**using** *assms* **by** (*induct rule: interleavings.induct*) *auto*

**lemma** *interleavings-filter-eq2*:  
**assumes** *interleavings xs ys zs*  
**assumes**  $(\forall x \in set\ xs. \neg P\ x) \wedge (\forall y \in set\ ys. P\ y)$   
**shows** *filter*  $P\ zs = ys$   
**using** *assms* **by** (*induct rule: interleavings.induct*) *auto*

**lemma** *interleavings-length*:  
**assumes** *interleavings xs ys zs*  
**shows**  $length\ xs + length\ ys = length\ zs$   
**using** *assms* **by** (*induct xs ys zs rule: interleavings.induct*) *auto*

**lemma** *interleavings-set*:  
**assumes** *interleavings xs ys zs*  
**shows**  $set\ xs \cup set\ ys = set\ zs$   
**using** *assms* **by** (*induct xs ys zs rule: interleavings.induct*) *auto*

**lemma** *interleavings-distinct*:  
**assumes** *interleavings xs ys zs*  
**shows**  $\text{distinct } xs \wedge \text{distinct } ys \wedge \text{set } xs \cap \text{set } ys = \{\}$   $\longleftrightarrow$  *distinct zs*  
**using** *assms interleavings-set* **by** (*induct xs ys zs rule: interleavings.induct*) *fast-force+*

**lemma** *two-mutual-lists-induction*:  
**assumes**  $\bigwedge ys. P \ [] \ ys$   
**assumes**  $\bigwedge xs. P \ xs \ []$   
**assumes**  $\bigwedge x \ xs \ y \ ys. P \ xs \ (y\#\ys) \implies P \ (x\#xs) \ ys \implies P \ (x\#xs) \ (y\#\ys)$   
**shows**  $P \ xs \ ys$   
**using** *assms* **by** (*induction-schema*) (*pat-completeness, lexicographic-order*)

**lemma** *finite-interleavings*:  
*finite {zs. interleavings xs ys zs}*  
**proof** (*induct xs ys rule: two-mutual-lists-induction*)  
**case** (*1 ys*)  
**show** *?case* **by** (*simp add: interleaving-Nil-iff1*)  
**next**  
**case** (*2 xs*)  
**then show** *?case* **by** (*simp add: interleaving-Nil-iff2*)  
**next**  
**case** (*3 x xs y ys*)  
**then show** *?case* **by** (*simp add: interleavings-Cons*)  
**qed**

**lemma** *card-interleavings*:  
**assumes**  $\text{set } xs \cap \text{set } ys = \{\}$   
**shows**  $\text{card } \{zs. \text{interleavings } xs \ ys \ zs\} = (\text{length } xs + \text{length } ys \text{ choose } (\text{length } xs))$   
**using** *assms*  
**proof** (*induct xs ys rule: two-mutual-lists-induction*)  
**case** (*1 ys*)  
**have**  $\text{card } \{zs. \text{interleavings } [] \ ys \ zs\} = \text{card } \{ys\}$   
**by** (*simp add: interleaving-Nil-iff1*)  
**also have**  $\dots = (\text{length } [] + \text{length } ys \text{ choose } (\text{length } []))$  **by** *simp*  
**finally show** *?case* .  
**next**  
**case** (*2 xs*)  
**have**  $\text{card } \{zs. \text{interleavings } xs \ [] \ zs\} = \text{card } \{xs\}$   
**by** (*simp add: interleaving-Nil-iff2*)  
**also have**  $\dots = (\text{length } xs + \text{length } [] \text{ choose } (\text{length } xs))$  **by** *simp*  
**finally show** *?case* .  
**next**  
**case** (*3 x xs y ys*)  
**have**  $\text{card } \{zs. \text{interleavings } (x \# xs) \ (y \# ys) \ zs\} =$   
 $\text{card } (\{x\#zs \mid zs. \text{interleavings } xs \ (y\#\ys) \ zs\} \cup \{y\#zs \mid zs. \text{interleavings } (x\#xs) \ ys \ zs\})$

by (simp add: interleavings-Cons)  
 also have ... = card  $\{x\#zs \mid zs. \text{interleavings } xs \ (y\#ys) \ zs\} + \text{card } \{y\#zs \mid zs. \text{interleavings } (x\#xs) \ ys \ zs\}$   
 proof -  
   have finite  $\{x \# zs \mid zs. \text{interleavings } xs \ (y \# ys) \ zs\}$   
   by (simp add: finite-interleavings)  
   moreover have finite  $\{y \# zs \mid zs. \text{interleavings } (x \# xs) \ ys \ zs\}$   
   by (simp add: finite-interleavings)  
   moreover have  $\{x \# zs \mid zs. \text{interleavings } xs \ (y \# ys) \ zs\} \cap \{y \# zs \mid zs. \text{interleavings } (x \# xs) \ ys \ zs\} = \{\}$   
   using  $\langle \text{set } (x \# xs) \cap \text{set } (y \# ys) = \{\} \rangle$  by auto  
   ultimately show ?thesis by (simp add: card-Un-disjoint)  
 qed  
 also have ... = card  $((\lambda zs. x \# zs) \ ` \ \{zs. \text{interleavings } xs \ (y \# ys) \ zs\}) + \text{card } ((\lambda zs. y \# zs) \ ` \ \{zs. \text{interleavings } (x\#xs) \ ys \ zs\})$   
   by (simp add: setcompr-eq-image)  
 also have ... = card  $\{zs. \text{interleavings } xs \ (y \# ys) \ zs\} + \text{card } \{zs. \text{interleavings } (x\#xs) \ ys \ zs\}$   
   by (simp add: card-image)  
 also have ... =  $(\text{length } xs + \text{length } (y \# ys) \ \text{choose } \text{length } xs) + (\text{length } (x \# xs) + \text{length } ys \ \text{choose } \text{length } (x \# xs))$   
   using 3 by simp  
 also have ... =  $\text{length } (x \# xs) + \text{length } (y \# ys) \ \text{choose } \text{length } (x \# xs)$  by simp  
 finally show ?case .  
 qed

### 1.3 Cardinality of Distinct Fixed-Length Lists from a Union of Two Sets

lemma lists-distinct-union-by-interleavings:

assumes  $X \cap Y = \{\}$   
 shows  $\{zs. \text{length } zs = n \wedge \text{distinct } zs \wedge \text{set } zs \subseteq X \cup Y\} = \text{do } \{$   
    $k \leftarrow \{0..n\};$   
    $xs \leftarrow \{xs. \text{length } xs = k \wedge \text{distinct } xs \wedge \text{set } xs \subseteq X\};$   
    $ys \leftarrow \{ys. \text{length } ys = n - k \wedge \text{distinct } ys \wedge \text{set } ys \subseteq Y\};$   
    $\{zs. \text{interleavings } xs \ ys \ zs\}$   
 } (is ?S = ?expr)

proof

show ?S  $\subseteq$  ?expr

proof

fix zs

assume  $zs \in ?S$

from this have  $\text{length } zs = n$  and  $\text{distinct } zs$  and  $\text{set } zs \subseteq X \cup Y$  by auto

define xs where  $xs = \text{filter } (\lambda z. z \in X) \ zs$

define ys where  $ys = \text{filter } (\lambda z. z \in Y) \ zs$

have eq:  $[z \leftarrow zs . z \in Y] = [z \leftarrow zs . z \notin X]$

using  $\langle \text{set } zs \subseteq X \cup Y \rangle \langle X \cap Y = \{\} \rangle$

by (auto intro: filter-cong)

```

have length xs ≤ n ∧ distinct xs ∧ set xs ⊆ X
  using ⟨length zs = n⟩ ⟨distinct zs⟩ unfolding xs-def by auto
moreover have length ys = n - length xs
  using ⟨set zs ⊆ X ∪ Y⟩ ⟨length zs = n⟩
  unfolding xs-def ys-def eq
  by (metis diff-add-inverse sum-length-filter-compl)
moreover have distinct ys ∧ set ys ⊆ Y
  using ⟨distinct zs⟩ unfolding ys-def by auto
moreover have interleavings xs ys zs
  using xs-def ys-def ⟨X ∩ Y = {}⟩ ⟨set zs ⊆ X ∪ Y⟩
  by (simp add: interleavings-filter)
ultimately show zs ∈ ?expr by force
qed
next
show ?expr ⊆ ?S
proof
  fix zs
  assume zs ∈ ?expr
  from this obtain xs ys where length xs ≤ n distinct xs set xs ⊆ X
    and length ys = n - length xs distinct ys set ys ⊆ Y interleavings xs ys zs
by auto
  have length zs = n
    using ⟨length xs ≤ n⟩ ⟨length ys = n - length xs⟩ ⟨interleavings xs ys zs⟩
    using interleavings-length by force
  moreover have distinct zs
    using ⟨distinct xs⟩ ⟨distinct ys⟩ ⟨interleavings xs ys zs⟩ ⟨set xs ⊆ X⟩ ⟨set ys ⊆
Y⟩
    using ⟨X ∩ Y = {}⟩ interleavings-distinct by fastforce
  moreover have set zs ⊆ X ∪ Y
    using ⟨interleavings xs ys zs⟩ ⟨set xs ⊆ X⟩ ⟨set ys ⊆ Y⟩ interleavings-set by
blast
  ultimately show zs ∈ ?S by blast
qed
qed

lemma interleavings-inject:
  assumes (set xs ∪ set xs') ∩ (set ys ∪ set ys') = {}
  assumes interleavings xs ys zs interleavings xs' ys' zs'
  assumes zs = zs'
  shows xs = xs' and ys = ys'
proof -
  have xs = filter (λz. z ∈ set xs ∪ set xs') zs
    using ⟨(set xs ∪ set xs') ∩ (set ys ∪ set ys') = {}⟩ ⟨interleavings xs ys zs⟩
    by (auto intro: interleavings-filter-eq1[symmetric])
  also have ... = filter (λz. z ∈ set xs ∪ set xs') zs'
    using ⟨zs = zs'⟩ by simp
  also have ... = xs'
    using ⟨(set xs ∪ set xs') ∩ (set ys ∪ set ys') = {}⟩ ⟨interleavings xs' ys' zs'⟩
    by (auto intro: interleavings-filter-eq1)

```

**finally show**  $xs = xs'$  **by** *simp*  
**have**  $ys = \text{filter } (\lambda z. z \in \text{set } ys \cup \text{set } ys') \text{ } zs$   
**using**  $\langle (\text{set } xs \cup \text{set } xs') \cap (\text{set } ys \cup \text{set } ys') = \{\} \rangle \langle \text{interleavings } xs \text{ } ys \text{ } zs \rangle$   
**by** (*auto intro: interleavings-filter-eq2[symmetric]*)  
**also have**  $\dots = \text{filter } (\lambda z. z \in \text{set } ys \cup \text{set } ys') \text{ } zs'$   
**using**  $\langle zs = zs' \rangle$  **by** *simp*  
**also have**  $\dots = ys'$   
**using**  $\langle (\text{set } xs \cup \text{set } xs') \cap (\text{set } ys \cup \text{set } ys') = \{\} \rangle \langle \text{interleavings } xs' \text{ } ys' \text{ } zs' \rangle$   
**by** (*auto intro: interleavings-filter-eq2*)  
**finally show**  $ys = ys'$  .  
**qed**

**lemma** *injectivity*:

**assumes**  $X \cap Y = \{\}$   
**assumes**  $k \in \{0..n\} \wedge k' \in \{0..n\}$   
**assumes**  $(\text{length } xs = k \wedge \text{distinct } xs \wedge \text{set } xs \subseteq X) \wedge (\text{length } xs' = k' \wedge \text{distinct } xs' \wedge \text{set } xs' \subseteq X)$   
**assumes**  $(\text{length } ys = n - k \wedge \text{distinct } ys \wedge \text{set } ys \subseteq Y) \wedge (\text{length } ys' = n - k' \wedge \text{distinct } ys' \wedge \text{set } ys' \subseteq Y)$   
**assumes**  $\text{interleavings } xs \text{ } ys \text{ } zs \wedge \text{interleavings } xs' \text{ } ys' \text{ } zs'$   
**assumes**  $zs = zs'$   
**shows**  $k = k'$  **and**  $xs = xs'$  **and**  $ys = ys'$

**proof** –

**from** *assms(1,3,4)* **have**  $(\text{set } xs \cup \text{set } xs') \cap (\text{set } ys \cup \text{set } ys') = \{\}$  **by** *blast*  
**from** *this assms(5)*  $\langle zs = zs' \rangle$  **show**  $xs = xs'$  **and**  $ys = ys'$   
**using** *interleavings-inject by fastforce+*  
**from** *this assms(3)* **show**  $k = k'$  **by** *auto*

**qed**

**lemma** *card-lists-distinct-length-eq-union*:

**assumes** *finite X finite Y*  $X \cap Y = \{\}$   
**shows**  $\text{card } \{zs. \text{length } zs = n \wedge \text{distinct } zs \wedge \text{set } zs \subseteq X \cup Y\} =$   
 $(\sum k=0..n. (n \text{ choose } k) * \text{ffact } k (\text{card } X) * \text{ffact } (n - k) (\text{card } Y))$   
**(is** *card ?S = -*)

**proof** –

**let** *?expr* = *do* {  
 $k \leftarrow \{0..n\}$ ;  
 $xs \leftarrow \{xs. \text{length } xs = k \wedge \text{distinct } xs \wedge \text{set } xs \subseteq X\}$ ;  
 $ys \leftarrow \{ys. \text{length } ys = n - k \wedge \text{distinct } ys \wedge \text{set } ys \subseteq Y\}$ ;  
 $\{zs. \text{interleavings } xs \text{ } ys \text{ } zs\}$   
}

**from**  $\langle X \cap Y = \{\} \rangle$  **have**  $\text{card } ?S = \text{card } ?expr$   
**by** (*simp add: lists-distinct-union-by-interleavings*)

**let** *?S*  $\gg=$  *?comp* = *?expr*

{  
**fix** *k*  
**assume**  $k \in ?S$   
**let** *?expr* = *?comp k*  
**let** *?S*  $\gg=$  *?comp* = *?expr*

```

from ⟨finite X⟩ have finite ?S by auto
moreover {
  fix xs
  assume xs: xs ∈ ?S
  let ?expr = ?comp xs
  let ?S ≫= ?comp = ?expr
  from ⟨finite Y⟩ have finite ?S by auto
  moreover {
    fix ys
    assume ys: ys ∈ ?S
    let ?expr = ?comp ys
    have finite ?expr
    by (simp add: finite-interleavings)
    moreover have card ?expr = (n choose k)
    using xs ys ⟨X ∩ Y = {}⟩ ⟨k ∈ -⟩
    by (subst card-interleavings) auto
    ultimately have finite ?expr ∧ card ?expr = (n choose k) ..
  }
  moreover have disjoint-family-on ?comp ?S
  using ⟨k ∈ {0..n}⟩ ⟨xs ∈ {xs. length xs = k ∧ distinct xs ∧ set xs ⊆ X}⟩
  by (injectivity-solver rule: injectivity(3))[OF ⟨X ∩ Y = {}⟩]
  moreover have card ?S = ffact (n - k) (card Y)
  using ⟨finite Y⟩ by (simp add: card-lists-distinct-length-eq)
  ultimately have card ?expr = (n choose k) * ffact (n - k) (card Y)
  by (subst card-bind-constant) auto
  moreover have finite ?expr
  using ⟨finite ?S⟩ by (auto intro!: finite-bind finite-interleavings)
  ultimately have finite ?expr ∧ card ?expr = (n choose k) * ffact (n - k)
(card Y)
  by blast
}
moreover have disjoint-family-on ?comp ?S
using ⟨k ∈ {0..n}⟩
by (injectivity-solver rule: injectivity(2))[OF ⟨X ∩ Y = {}⟩]
moreover have card ?S = ffact k (card X)
using ⟨finite X⟩ by (simp add: card-lists-distinct-length-eq)
ultimately have card ?expr = (n choose k) * ffact k (card X) * ffact (n - k)
(card Y)
by (subst card-bind-constant) auto
moreover have finite ?expr
using ⟨finite ?S⟩ ⟨finite Y⟩ by (auto intro!: finite-bind finite-interleavings)
ultimately have finite ?expr ∧ card ?expr = (n choose k) * ffact k (card X) *
ffact (n - k) (card Y)
by blast
}
moreover have disjoint-family-on ?comp ?S
by (injectivity-solver rule: injectivity(1))[OF ⟨X ∩ Y = {}⟩]
ultimately have card ?expr = (∑ k=0..n. (n choose k) * ffact k (card X) *
ffact (n - k) (card Y))

```



```

    by (auto simp add: card-bind)
  from ⟨card - = card ?expr⟩ this show ?thesis by simp
qed

```

**lemma**

$$\text{ffact } n (x + y) = \left( \sum_{k=0..n} (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y \right)$$

**proof** -

```

  define X where X = {.. $x$ }
  define Y where Y = {.. $x+y$ }
  have finite X and card X =  $x$  unfolding X-def by auto
  have finite Y and card Y =  $y$  unfolding Y-def by auto
  have  $X \cap Y = \{\}$  unfolding X-def Y-def by auto
  have  $\text{ffact } n (x + y) = \text{ffact } n (\text{card } X + \text{card } Y)$ 
    using ⟨card X =  $x$ ⟩ ⟨card Y =  $y$ ⟩ by simp
  also have ... =  $\text{ffact } n (\text{card } (X \cup Y))$ 
    using ⟨ $X \cap Y = \{\}$ ⟩ ⟨finite X⟩ ⟨finite Y⟩ by (simp add: card-Un-disjoint)
  also have ... = card { $xs.$  length  $xs = n \wedge$  distinct  $xs \wedge$  set  $xs \subseteq X \cup Y$ }
    using ⟨finite X⟩ ⟨finite Y⟩ by (simp add: card-lists-distinct-length-eq)
  also have ... =  $\left( \sum_{k=0..n} (n \text{ choose } k) * \text{ffact } k (\text{card } X) * \text{ffact } (n - k) (\text{card } Y) \right)$ 
    using ⟨ $X \cap Y = \{\}$ ⟩ ⟨finite X⟩ ⟨finite Y⟩ by (simp add: card-lists-distinct-length-eq-union)
  also have ... =  $\left( \sum_{k=0..n} (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y \right)$ 
    using ⟨card X =  $x$ ⟩ ⟨card Y =  $y$ ⟩ by simp
  finally show ?thesis .

```

qed

end

## 2 Proving Falling Factorial of a Sum with Induction

**theory** *Falling-Factorial-Sum-Induction*

**imports**

*Discrete-Summation.Factorials*

**begin**

Note the potentially special copyright license condition of the following proof.

**lemma** *ffact-add-nat*:

$$\text{ffact } n (x + y) = \left( \sum_{k=0..n} (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y \right)$$

**proof** (*induct n*)

**case** 0

**show** ?case **by** *simp*

**next**

**case** (*Suc n*)

**let** ?s =  $\lambda k. (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y$

**let** ?t =  $\lambda k. \text{ffact } k x * \text{ffact } (\text{Suc } n - k) y$

**let** ?u =  $\lambda k. \text{ffact } (\text{Suc } k) x * \text{ffact } (n - k) y$

**have**  $\text{ffact } (\text{Suc } n) (x + y) = (x + y - n) * \text{ffact } n (x + y)$   
**by** (*simp add: ffact-Suc-rev-nat*)  
**also have**  $\dots = (x + y - n) * (\sum k = 0..n. (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y)$   
**using** *Suc.hyps* **by** *simp*  
**also have**  $\dots = (\sum k = 0..n. ?s k * (x + y - n))$   
**by** (*simp add: mult.commute sum-distrib-left*)  
**also have**  $\dots = (\sum k = 0..n. ?s k * ((y + k - n) + (x - k)))$   
**proof** -  
**have**  $?s k * (x + y - n) = ?s k * ((y + k - n) + (x - k))$  **for**  $k$   
**by** (*cases k ≤ x ∨ n - k ≤ y*) (*auto simp add: ffact-nat-triv*)  
**from this show** *?thesis*  
**by** (*auto intro: sum.cong simp only: refl*)  
**qed**  
**also have**  $\dots = (\sum k = 0..n. (n \text{ choose } k) * (?t k + ?u k))$   
**by** (*auto intro!: sum.cong simp add: Suc-diff-le ffact-Suc-rev-nat*) *algebra*  
**also have**  $\dots = (\sum k = 0..n. (n \text{ choose } k) * ?t k) + (\sum k = 0..n. (n \text{ choose } k) * ?u k)$   
**by** (*simp add: sum.distrib add-mult-distrib2 mult.commute mult.left-commute*)  
**also have**  $\dots = ?t 0 + (\sum k = 0..n. (n \text{ choose } k + (n \text{ choose } \text{Suc } k)) * ?u k)$   
**proof** -  
**have**  $\dots = (?t 0 + (\sum k = 0..n. (n \text{ choose } \text{Suc } k) * ?u k)) + (\sum k = 0..n. (n \text{ choose } k) * ?u k)$   
**proof** -  
**have**  $(\sum k = \text{Suc } 0..n. (n \text{ choose } k) * ?t k) = (\sum k = 0..n. (n \text{ choose } \text{Suc } k) * ?u k)$   
**proof** -  
**have**  $(\sum k = \text{Suc } 0..n. (n \text{ choose } k) * ?t k) = (\sum k = \text{Suc } 0..\text{Suc } n. (n \text{ choose } k) * ?t k)$   
**by** *simp*  
**also have**  $\dots = (\text{sum } ((\lambda k. (n \text{ choose } k) * ?t k) \text{ o } \text{Suc}) \{0..n\})$   
**by** (*simp only: sum.reindex[symmetric, of Suc] inj-Suc image-Suc-atLeastAtMost*)  
**also have**  $\dots = (\sum k = 0..n. (n \text{ choose } \text{Suc } k) * ?u k)$   
**by** *simp*  
**finally show** *?thesis* .  
**qed**  
**from this show** *?thesis*  
**by** (*simp add: sum.atLeast-Suc-atMost[of - - λk. (n choose k) \* ?t k]*)  
**qed**  
**also have**  $\dots = ?t 0 + (\sum k = 0..n. (n \text{ choose } k + (n \text{ choose } \text{Suc } k)) * ?u k)$   
**by** (*simp add: distrib-right sum.distrib*)  
**finally show** *?thesis* .  
**qed**  
**also have**  $\dots = (\sum k = 0..\text{Suc } n. (\text{Suc } n \text{ choose } k) * \text{ffact } k x * \text{ffact } (\text{Suc } n - k) y)$   
**proof** -  
**let**  $?v = \lambda k. (\text{Suc } n \text{ choose } k) * \text{ffact } k x * \text{ffact } (\text{Suc } n - k) y$   
**have**  $\dots = ?v 0 + (\sum k = 0..n. (\text{Suc } n \text{ choose } (\text{Suc } k)) * ?u k)$   
**by** *simp*

**also have** ... = ?v 0 + ( $\sum k = \text{Suc } 0.. \text{Suc } n. ?v k$ )  
**by** (*simp only: sum.shift-bounds-cl-Suc-ivl diff-Suc-Suc mult.assoc*)  
**also have** ... = ( $\sum k = 0.. \text{Suc } n. (\text{Suc } n \text{ choose } k) * \text{ffact } k x * \text{ffact } (\text{Suc } n - k) y$ )  
**by** (*simp add: sum.atLeast-Suc-atMost*)  
**finally show** ?thesis .  
**qed**  
**finally show** ?case .  
**qed**

**lemma** *ffact-add*:

**fixes**  $x y :: 'a::\{ab\text{-group-add, comm-semiring-1-cancel, ring-1}\}$   
**shows**  $\text{ffact } n (x + y) = (\sum k=0..n. \text{of-nat } (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y)$   
**proof** (*induct n*)  
**case** 0  
**show** ?case **by** *simp*  
**next**  
**case** (*Suc n*)  
**let** ?s =  $\lambda k. \text{of-nat } (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y$   
**let** ?t =  $\lambda k. \text{ffact } k x * \text{ffact } (\text{Suc } n - k) y$   
**let** ?u =  $\lambda k. \text{ffact } (\text{Suc } k) x * \text{ffact } (n - k) y$   
**have**  $\text{ffact } (\text{Suc } n) (x + y) = (x + y - \text{of-nat } n) * \text{ffact } n (x + y)$   
**by** (*simp add: ffact-Suc-rev*)  
**also have** ... =  $(x + y - \text{of-nat } n) * (\sum k = 0..n. \text{of-nat } (n \text{ choose } k) * \text{ffact } k x * \text{ffact } (n - k) y)$   
**using** *Suc.hyps* **by** *simp*  
**also have** ... =  $(\sum k = 0..n. ?s k * (x + y - \text{of-nat } n))$   
**by** (*simp add: mult.commute sum-distrib-left*)  
**also have** ... =  $(\sum k = 0..n. ?s k * ((y + \text{of-nat } k - \text{of-nat } n) + (x - \text{of-nat } k)))$   
**by** (*auto intro: sum.cong simp add: diff-add-eq add-diff-eq add.commute*)  
**also have** ... =  $(\sum k = 0..n. \text{of-nat } (n \text{ choose } k) * (?t k + ?u k))$   
**proof** –  
{  
**fix**  $k$   
**assume**  $k \leq n$   
**have** ?u k =  $\text{ffact } k x * \text{ffact } (n - k) y * (x - \text{of-nat } k)$   
**by** (*simp add: ffact-Suc-rev Suc-diff-le of-nat-diff mult.commute mult.left-commute*)  
**moreover from**  $\langle k \leq n \rangle$  **have** ?t k =  $\text{ffact } k x * \text{ffact } (n - k) y * (y + \text{of-nat } k - \text{of-nat } n)$   
**by** (*simp add: ffact-Suc-rev Suc-diff-le of-nat-diff diff-diff-eq2 mult.commute mult.left-commute*)  
**ultimately have**  
?s k \*  $((y + \text{of-nat } k - \text{of-nat } n) + (x - \text{of-nat } k)) = \text{of-nat } (n \text{ choose } k)$   
\*  $(?t k + ?u k)$   
**by** (*metis (no-types, lifting) distrib-left mult.assoc*)  
}  
}

**from this show** *?thesis* **by** (*auto intro: sum.cong*)  
**qed**  
**also have**  $\dots = (\sum k = 0..n. \text{of-nat } (n \text{ choose } k) * ?t k) + (\sum k = 0..n. \text{of-nat } (n \text{ choose } k) * ?u k)$   
**by** (*simp add: sum.distrib distrib-left mult.commute mult.left-commute*)  
**also have**  $\dots = ?t 0 + (\sum k = 0..n. \text{of-nat } (n \text{ choose } k + (n \text{ choose } \text{Suc } k)) * ?u k)$   
**proof** –  
**have**  $\dots = (?t 0 + (\sum k = 0..n. \text{of-nat } (n \text{ choose } \text{Suc } k) * ?u k)) + (\sum k = 0..n. \text{of-nat } (n \text{ choose } k) * ?u k)$   
**proof** –  
**have**  $(\sum k = \text{Suc } 0..n. \text{of-nat } (n \text{ choose } k) * ?t k) = (\sum k = 0..n. \text{of-nat } (n \text{ choose } \text{Suc } k) * ?u k)$   
**proof** –  
**have**  $(\sum k = \text{Suc } 0..n. \text{of-nat } (n \text{ choose } k) * ?t k) = (\sum k = \text{Suc } 0..\text{Suc } n. \text{of-nat } (n \text{ choose } k) * ?t k)$   
**by** (*simp add: binomial-eq-0*)  
**also have**  $\dots = (\text{sum } ((\lambda k. \text{of-nat } (n \text{ choose } k) * ?t k) \text{ o } \text{Suc}) \{0..n\})$   
**by** (*simp only: sum.reindex[symmetric, of Suc] inj-Suc image-Suc-atLeastAtMost*)  
**also have**  $\dots = (\sum k = 0..n. \text{of-nat } (n \text{ choose } \text{Suc } k) * ?u k)$   
**by** *simp*  
**finally show** *?thesis* .  
**qed**  
**from this show** *?thesis*  
**by** (*simp add: sum.atLeast-Suc-atMost[of - -  $\lambda k. \text{of-nat } (n \text{ choose } k) * ?t k]$* )  
**qed**  
**also have**  $\dots = ?t 0 + (\sum k = 0..n. \text{of-nat } (n \text{ choose } k + (n \text{ choose } \text{Suc } k)) * ?u k)$   
**by** (*simp add: distrib-right sum.distrib*)  
**finally show** *?thesis* .  
**qed**  
**also have**  $\dots = (\sum k = 0..\text{Suc } n. \text{of-nat } (\text{Suc } n \text{ choose } k) * \text{ffact } k \ x * \text{ffact } (\text{Suc } n - k) \ y)$   
**proof** –  
**let**  $?v = \lambda k. \text{of-nat } (\text{Suc } n \text{ choose } k) * \text{ffact } k \ x * \text{ffact } (\text{Suc } n - k) \ y$   
**have**  $\dots = ?v 0 + (\sum k = 0..n. \text{of-nat } (\text{Suc } n \text{ choose } (\text{Suc } k)) * ?u k)$   
**by** *simp*  
**also have**  $\dots = ?v 0 + (\sum k = \text{Suc } 0..\text{Suc } n. ?v k)$   
**by** (*simp only: sum.shift-bounds-cl-Suc-ivl diff-Suc-Suc mult.assoc*)  
**also have**  $\dots = (\sum k = 0..\text{Suc } n. \text{of-nat } (\text{Suc } n \text{ choose } k) * \text{ffact } k \ x * \text{ffact } (\text{Suc } n - k) \ y)$   
**by** (*simp add: sum.atLeast-Suc-atMost*)  
**finally show** *?thesis* .  
**qed**  
**finally show** *?case* .  
**qed**  
**end**

### 3 Proving Falling Factorial of a Sum with Vandermonde Identity

**theory** *Falling-Factorial-Sum-Vandermonde*

**imports**

*Discrete-Summation.Factorials*

**begin**

Note the potentially special copyright license condition of the following proof.

**lemma** *ffact-add-nat*:

**shows**  $\text{ffact } k (n + m) = (\sum i \leq k. (k \text{ choose } i) * \text{ffact } i n * \text{ffact } (k - i) m)$

**proof** –

**have**  $\text{ffact } k (n + m) = \text{fact } k * ((n + m) \text{ choose } k)$

**by** (*simp only: ffact-eq-fact-mult-binomial*)

**also have**  $\dots = \text{fact } k * (\sum i \leq k. (n \text{ choose } i) * (m \text{ choose } (k - i)))$

**by** (*simp only: vandermonde*)

**also have**  $\dots = (\sum i \leq k. \text{fact } k * (n \text{ choose } i) * (m \text{ choose } (k - i)))$

**by** (*simp add: sum-distrib-left field-simps*)

**also have**  $\dots = (\sum i \leq k. (\text{fact } i * \text{fact } (k - i) * (k \text{ choose } i) * (n \text{ choose } i) * (m \text{ choose } (k - i))))$

**by** (*simp add: binomial-fact-lemma*)

**also have**  $\dots = (\sum i \leq k. (k \text{ choose } i) * (\text{fact } i * (n \text{ choose } i)) * (\text{fact } (k - i) * (m \text{ choose } (k - i))))$

**by** (*auto intro: sum.cong*)

**also have**  $\dots = (\sum i \leq k. (k \text{ choose } i) * \text{ffact } i n * \text{ffact } (k - i) m)$

**by** (*simp only: ffact-eq-fact-mult-binomial*)

**finally show** *?thesis* .

**qed**

**end**

### 4 Note on Copyright Licensing

The initial material of the informal proof for this formalisation is provided on Mathematics Stack Exchange under the Creative Commons Attribution-ShareAlike 3.0 Unported license (CC BY-SA 3.0; <https://creativecommons.org/licenses/by-sa/3.0/>), which is pointed out on the the Mathematics Stack Exchange terms of use at <https://stackexchange.com/legal/terms-of-service>.

The two main proofs, the induction and the algebraic proof in this AFP entry are (even textually) very close to the initial material from Mathematics Stack Exchange.

In case the two Isabelle proofs are judged to build upon the main proofs from Mathematics Stack Exchange, the CC BY-SA 3.0 license requires that these proofs must be available under the same license, and hence, these

proofs are consequently licensed under CC BY-SA 3.0. In case the two Isabelle proofs are not judged to build upon the material from Mathematics Stack Exchange, I as an author provide them under the 3-Clause BSD License (<https://opensource.org/licenses/BSD-3-Clause>) to allow their seamless integration into the Isabelle repository at any point in time.

All other content that does not build upon the material from Mathematics Stack Exchange is licensed under the 3-clause BSD License, and can be copied, moved or integrated in other work licensed under the 3-clause BSD License without further consideration of the different obligations of the existing copyright licensing.

## References

- [1] ajotatxe. Combinatorial proof of falling factorial and binomial theorem. Mathematics Stack Exchange. <https://math.stackexchange.com/q/1271700> (version: 2015-05-07), author profile: <https://math.stackexchange.com/users/132456/ajotatxe>.
- [2] grapher. Combinatorial proof of falling factorial and binomial theorem. Mathematics Stack Exchange. <https://math.stackexchange.com/q/1271688> (version: 2016-08-27), author profile: <https://math.stackexchange.com/users/199155/grapher>.
- [3] F. John. Combinatorial proof of falling factorial and binomial theorem. Mathematics Stack Exchange. <https://math.stackexchange.com/q/2161558> (version: 2017-11-14), author profile: <https://math.stackexchange.com/users/302692/foobaz-john>.
- [4] B. M. Scott. Combinatorial proof of falling factorial and binomial theorem. Mathematics Stack Exchange. <https://math.stackexchange.com/q/1271983> (version: 2015-05-07), author profile: <https://math.stackexchange.com/users/12042/brian-m-scott>.