# The Falling Factorial of a Sum 

Lukas Bulwahn

September 13, 2023


#### 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

## 1 Proving Falling Factorial of a Sum with Combinatorics 1

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

2 Proving Falling Factorial of a Sum with Induction 9
3 Proving Falling Factorial of a Sum with Vandermonde Identity

4 Note on Copyright Licensing 13

## 1 Proving Falling Factorial of a Sum with Combinatorics

theory Falling-Factorial-Sum-Combinatorics
imports

Discrete－Summation．Factorials
begin

## 1．1 Preliminaries

## 1．1．1 Addition to Factorials Theory

```
lemma card-lists-distinct-length-eq:
    assumes finite A
    shows card {xs.length xs = n ^ distinct xs ^ set xs \subseteqA} = ffact n (card A)
proof cases
    assume n \leqcard A
    have card {xs. length xs = n ^ distinct xs ^ set xs \subseteqA}=\prod{card A - n+
1..card A}
    using <finite A〉\langlen\leq card A> by (rule card-lists-distinct-length-eq)
    also have ... = ffact n (card A)
        using <n \leq card A> by (simp add: prod-rev-ffact-nat'[symmetric])
    finally show ?thesis.
next
    assume }\negn\leq\operatorname{card}
    from this<finite A〉 have }\forallxs\mathrm{ . length xs = n ^ distinct xs ^ set xs }\subseteqA
False
        by (metis card-mono distinct-card)
    from this have eq-empty: {xs. length xs =n\wedge distinct xs ^ set xs\subseteqA}={}
        using <finite A> by auto
    from <\neg n \leq card A〉 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 [] ys ys
| interleavings xs [] xs
| interleavings xs ys zs $\Longrightarrow$ interleavings $(x \# x s)$ ys $(x \# z s)$
$\mid$ interleavings $x s$ ys $z s \Longrightarrow$ interleavings $x s(y \# y s)(y \# z s)$
lemma interleaving-Nil-implies-eq1:
assumes interleavings xs ys zs
assumes $x s=[]$
shows $y s=z s$
using assms by (induct rule: interleavings.induct) auto
lemma interleaving-Nil-iff1:
interleavings [] ys zs $\longleftrightarrow(y s=z s)$
using interleaving-Nil-implies-eq1
by (auto simp add: interleavings.intros(1))

```
lemma interleaving-Nil-implies-eq2:
    assumes interleavings xs ys zs
    assumes \(y s=[]\)
    shows \(x s=z s\)
using assms by (induct rule: interleavings.induct) auto
lemma interleaving-Nil-iff2:
    interleavings \(x s[] z s \longleftrightarrow(x s=z s)\)
using interleaving-Nil-implies-eq2
by (auto simp add: interleavings.intros(2))
lemma interleavings-Cons:
    \(\{z s\). interleavings \((x \# x s)(y \# y s) z s\}=\)
        \(\{x \# z s \mid z s\). interleavings \(x s(y \# y s) z s\} \cup\{y \# z s \mid z s\). interleavings \((x \# x s)\) ys \(z s\}\)
    (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 \(z s \subseteq X \cup Y\)
    shows interleavings \([z \leftarrow z s . z \in X][z \leftarrow z s . z \in Y] z s\)
using assms by (induct zs) (auto intro: interleavings.intros)
lemma interleavings-filter-eq1:
    assumes interleavings xs ys zs
    assumes \((\forall x \in\) set \(x s . P x) \wedge(\forall y \in\) set ys. \(\neg P y)\)
    shows filter \(P\) zs \(=x s\)
using assms by (induct rule: interleavings.induct) auto
lemma interleavings-filter-eq2:
    assumes interleavings xs ys zs
    assumes \((\forall x \in\) set \(x s . \neg P x) \wedge(\forall y \in\) set ys. \(P y)\)
    shows filter \(P\) zs \(=y s\)
using assms by (induct rule: interleavings.induct) auto
lemma interleavings-length:
    assumes interleavings xs ys zs
    shows length \(x s+\) length \(y s=\) length zs
using assms by (induct xs ys zs rule: interleavings.induct) auto
lemma interleavings-set:
    assumes interleavings xs ys zs
    shows set \(x s \cup\) set \(y s=\) set \(z s\)
using assms by (induct xs ys zs rule: interleavings.induct) auto
```

```
lemma interleavings-distinct:
    assumes interleavings xs ys zs
    shows distinct xs ^ distinct ys ^ set xs \cap 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 \ys.P[] ys
    assumes \xs.P xs []
    assumes \x xs y ys.P xs (y#ys)\LongrightarrowP(x#xs) ys \LongrightarrowP(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 (3x xs y ys)
    then show ?case by (simp add: interleavings-Cons)
qed
lemma card-interleavings:
    assumes set xs \cap set ys = {}
    shows card {zs. interleavings xs ys zs} = (length xs + length ys choose (length
xs))
using assms
proof (induct xs ys rule: two-mutual-lists-induction)
    case (1 ys)
    have card {zs. interleavings [] ys zs} = card {ys}
    by (simp add: interleaving-Nil-iff1)
    also have ... = (length [] + length ys choose (length [])) by simp
    finally show ?case.
next
    case (2 xs)
    have card {zs. interleavings xs [] zs} = card {xs}
        by (simp add: interleaving-Nil-iff2)
    also have ... = (length xs + length [] choose (length xs)) by simp
    finally show ?case .
next
    case (3 x xs y ys)
    have card {zs. interleavings (x # xs) (y# ys) zs} =
        card ({x#zs|zs.interleavings xs (y#ys)zs} \cup{y#zs|zs.interleavings (x#xs) ys
zs})
```

```
    by (simp add: interleavings-Cons)
    also have \(\ldots=\operatorname{card}\{x \# z s \mid z s\). interleavings \(x s(y \# y s) z s\}+\operatorname{card}\{y \# z s \mid z s\).
interleavings ( \(x \# x s\) ) ys zs\}
    proof -
    have finite \(\{x \# z s \mid z s\). interleavings \(x s(y \# y s) z s\}\)
            by (simp add: finite-interleavings)
    moreover have finite \(\{y \# z s \mid z s\). interleavings \((x \# x s)\) ys \(z s\}\)
        by (simp add: finite-interleavings)
    moreover have \(\{x \#\) zs \(\mid z s\). interleavings \(x s(y \# y s) z s\} \cap\{y \# z s \mid z s\).
interleavings ( \(x\) \# xs) ys zs \(\}=\{ \}\)
        using 〈set \((x \# x s) \cap \operatorname{set}(y \# y s)=\{ \}\) 〉 by auto
    ultimately show ?thesis by (simp add: card-Un-disjoint)
    qed
    also have \(\ldots=\operatorname{card}((\lambda z s . x \# z s)\) ' \(\{z s\). interleavings \(x s(y \# y s) z s\})+\)
    card (( \(\lambda z s . y \# z s)\) ' \(\{z s\). interleavings \((x \# x s) y s z s\})\)
    by (simp add: setcompr-eq-image)
    also have \(\ldots=\) card \(\{z s\). interleavings \(x s(y \# y s) z s\}+\) card \(\{z s\). interleavings
( \(x \# x s\) ) ys zs\}
    by (simp add: card-image)
    also have \(\ldots=(\) length \(x s+\) length \((y \# y s)\) choose length \(x s)+(\) length \((x \#\)
\(x s)+\) length ys choose length \((x \# x s))\)
    using 3 by \(\operatorname{simp}\)
    also have \(\ldots=\) length \((x \# x s)+\) length \((y \# y s)\) choose length \((x \# x s)\) 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 $\{z s$. length $z s=n \wedge$ distinct $z s \wedge$ set $z s \subseteq X \cup Y\}=d o\{$
$k \leftarrow\{0 . . n\} ;$
$x s \leftarrow\{x s$. length $x s=k \wedge$ distinct $x s \wedge$ set $x s \subseteq X\}$;
$y s \leftarrow\{$ ys. length ys $=n-k \wedge$ distinct ys $\wedge$ set $y s \subseteq Y\} ;$
$\{z s$. interleavings xs ys $z s\}$
$\}($ is ? $S=?$ expr $)$
proof
show ? $S \subseteq$ ? expr
proof
fix $z s$
assume $z s \in$ ? $S$
from this have length $z s=n$ and distinct $z s$ and set $z s \subseteq X \cup Y$ by auto
define $x s$ where $x s=$ filter $(\lambda z . z \in X) z s$
define $y s$ where $y s=$ filter $(\lambda z . z \in Y) z s$
have eq: $[z \leftarrow z s . z \in Y]=[z \leftarrow z s, z \notin X]$ using «set zs $\subseteq X \cup Y\rangle\langle X \cap Y=\{ \}\rangle$ by (auto intro: filter-cong)

```
    have length xs \leqn^ distinct xs ^ set xs \subseteqX
    using <length zs = n`<distinct zs` unfolding xs-def by auto
    moreover have length ys =n-length xs
        using <set zs \subseteqX\cupY><length zs = n>
        unfolding xs-def ys-def eq
        by (metis diff-add-inverse sum-length-filter-compl)
    moreover have distinct ys }\wedge\mathrm{ set ys }\subseteq
        using <distinct zs> unfolding ys-def by auto
    moreover have interleavings xs ys zs
        using xs-def ys-def〈X\capY={}〉\langleset zs\subseteqX\cupY〉
        by (simp add: interleavings-filter)
    ultimately show zs \in? expr by force
    qed
next
    show ?expr \subseteq?S
    proof
    fix zs
    assume zs \in? expr
    from this obtain xs ys where length xs \leqn distinct xs set xs \subseteqX
        and length ys =n - length xs distinct ys set ys }\subseteqY\mathrm{ interleavings xs ys zs by
auto
    have length zs = n
            using <length xs \leqn><length ys = n - length xs`<interleavings xs ys zs`
            using interleavings-length by force
    moreover have distinct zs
        using <distinct xs\rangle<distinct ys\rangle\langleinterleavings xs ys zs\rangle<set xs \subseteqX\rangle<set ys
\subseteq Y >
            using <X\capY={}> interleavings-distinct by fastforce
    moreover have set zs\subseteqX\cupY
        using <interleavings xs ys zs\rangle\langleset xs \subseteqX\rangle<set ys\subseteqY〉 interleavings-set by
blast
    ultimately show zs \in?S by blast
    qed
qed
lemma interleavings-inject:
    assumes (set xs \cup set xs') \cap (set ys \cup set ys')={}
    assumes interleavings xs ys zs interleavings xs' ys'zs'
    assumes zs = zs'
    shows xs = xs'' and ys = ys'
proof -
    have xs = filter (\lambdaz.z\in set xs \cup set xs') zs
        using <(set xs \cup set xs') \cap (set ys \cup set ys')={}>〈interleavings xs ys zs〉
        by (auto intro: interleavings-filter-eq1[symmetric])
    also have ... = filter ( }\lambdaz.z\in\mathrm{ set xs U set xs') zs'
        using <zs = zs'〉 by simp
    also have ... = xs'
        using <(set xs \cup set xs') \cap (set ys U set ys')={}>〈interleavings xs' ys'zs'>
        by (auto intro: interleavings-filter-eq1)
```

```
    finally show \(x s=x s^{\prime}\) by \(\operatorname{simp}\)
    have \(y s=\) filter \(\left(\lambda z . z \in\right.\) set \(y s \cup\) set \(\left.y s^{\prime}\right) z s\)
    using 〈 (set \(x s \cup\) set \(\left.x s^{\prime}\right) \cap\left(\right.\) set \(y s \cup\) set \(\left.y s^{\prime}\right)=\{ \}\) 〉〈interleavings xs ys zs〉
    by (auto intro: interleavings-filter-eq2[symmetric])
    also have \(\ldots=\) filter \(\left(\lambda z . z \in\right.\) set \(y s \cup\) set \(\left.y s^{\prime}\right) z s^{\prime}\)
    using \(\left\langle z s=z s^{\prime}\right\rangle\) by simp
    also have \(\ldots=y s^{\prime}\)
    using 〈 \(\left(\right.\) set \(x s \cup\) set \(\left.x s^{\prime}\right) \cap\left(\right.\) set \(y s \cup\) set \(\left.y s^{\prime}\right)=\{ \}\) 〈interleavings \(\left.x s^{\prime} y s^{\prime} z s^{\prime}\right\rangle\)
    by (auto intro: interleavings-filter-eq2)
    finally show \(y s=y s^{\prime}\).
qed
lemma injectivity:
    assumes \(X \cap Y=\{ \}\)
    assumes \(k \in\{0 . . n\} \wedge k^{\prime} \in\{0 . . n\}\)
    assumes (length \(x s=k \wedge\) distinct \(x s \wedge\) set \(x s \subseteq X) \wedge\left(\right.\) length \(x s^{\prime}=k^{\prime} \wedge\) distinct
\(x s^{\prime} \wedge\) set \(\left.x s^{\prime} \subseteq X\right)\)
    assumes (length ys \(=n-k \wedge\) distinct ys \(\wedge\) set \(y s \subseteq Y) \wedge\left(\right.\) length \(y s^{\prime}=n-k^{\prime}\)
\(\wedge\) distinct \(y s^{\prime} \wedge\) set \(\left.y s^{\prime} \subseteq Y\right)\)
    assumes interleavings xs ys zs \(\wedge\) interleavings \(x s^{\prime} y^{\prime}{ }^{\prime} z s^{\prime}\)
    assumes \(z s=z s^{\prime}\)
    shows \(k=k^{\prime}\) and \(x s=x s^{\prime}\) and \(y s=y s^{\prime}\)
proof -
    from \(\operatorname{assms}(1,3,4)\) have \(\left(\right.\) set \(x s \cup\) set \(\left.x s^{\prime}\right) \cap\left(\right.\) set \(y s \cup\) set \(\left.y s^{\prime}\right)=\{ \}\) by blast
    from this assms(5) \(\left\langle z s=z s^{\prime}\right\rangle\) show \(x s=x s^{\prime}\) and \(y s=y s^{\prime}\)
    using interleavings-inject by fastforce+
    from this assms(3) show \(k=k^{\prime}\) by auto
qed
lemma card-lists-distinct-length-eq-union:
    assumes finite \(X\) finite \(Y X \cap Y=\{ \}\)
    shows card \(\{z s\). length \(z s=n \wedge\) distinct \(z s \wedge\) set \(z s \subseteq X \cup Y\}=\)
        \(\left(\sum k=0 . . n .(n\right.\) choose \(k) *\) ffact \(k(\operatorname{card} X) *\) ffact \(\left.(n-k)(\operatorname{card} Y)\right)\)
    (is card ? \(S=-\) )
proof -
    let ? expr \(=d o\{\)
    \(k \leftarrow\{0 . . n\} ;\)
    \(x s \leftarrow\{x s\). length \(x s=k \wedge\) distinct \(x s \wedge\) set \(x s \subseteq X\}\);
    \(y s \leftarrow\{y s\). length \(y s=n-k \wedge\) distinct \(y s \wedge\) set \(y s \subseteq Y\} ;\)
    \(\{z s\). interleavings xs ys zs\}
    \}
    from \(\langle X \cap Y=\{ \}\rangle\) have card ? \(S=\) 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 \in?S
    let ? expr = ?comp xs
    let ?S >> ?comp = ? expr
    from〈finite Y〉 have finite ?S by auto
    moreover {
        fix ys
        assume ys:ys \in?S
        let ?expr = ?comp ys
        have finite ?expr
            by (simp add: finite-interleavings)
        moreover have card ? expr = ( n choose k)
            using xs ys < X \cap Y={}><k \in ->
            by (subst card-interleavings) auto
        ultimately have finite ? expr ^ card ? expr = (n choose k)..
    }
    moreover have disjoint-family-on ?comp ?S
        using <k\in{0..n}>\langlexs \in{xs. length xs = k ^ distinct xs ^ set xs\subseteqX}>
        by (injectivity-solver rule: injectivity(3)[OF<X\capY={}>])
    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)(\mathrm{ 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\in{0..n}>
        by (injectivity-solver rule: injectivity(2)[OF <X \capY={}>])
    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\capY={}>])
    ultimately have card ? expr = ( \sumk=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
    ffact n (x+y)=(\sumk=0..n. (n choose k)* ffact k x * ffact (n-k) y)
proof -
    define }X\mathrm{ where }X={..<x
    define }Y\mathrm{ where }Y={x..<x+y
    have finite }X\mathrm{ and card }X=x\mathrm{ unfolding X-def by auto
    have finite Y and card Y=y unfolding Y-def by auto
    have }X\capY={}\mathrm{ unfolding X-def Y-def by auto
    have ffact n (x+y) = ffact n (card X + card Y)
    using <card X = x\rangle\langlecard Y = y> by simp
    also have ... = ffact n (card ( X\cupY))
        using <X\capY={}><finite X\rangle\langlefinite Y> by (simp add:card-Un-disjoint)
    also have ... = card {xs. length xs = n ^ distinct xs ^ set xs\subseteqX\cupY}
    using <finite X〉<finite Y> by (simp add: card-lists-distinct-length-eq)
    also have \ldots. = (\sumk=0..n. (n choose k)* ffact k (card X)* ffact (n-k) (card
Y))
    using <X\capY={}><finite X><finite Y> by (simp add:card-lists-distinct-length-eq-union)
    also have ... = (\sumk=0..n. (n choose k)* ffact kx* ffact (n-k) y)
    using <card X = x\rangle\langlecard Y = y> by simp
    finally show ?thesis.
qed
end
```


## 2 Proving Falling Factorial of a Sum with Induction

## theory Falling-Factorial-Sum-Induction <br> imports <br> Discrete-Summation.Factorials <br> begin

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

```
lemma ffact-add-nat:
    ffact \(n(x+y)=\left(\sum k=0 . . n .(n\right.\) choose \(k) *\) ffact \(k x *\) ffact \(\left.(n-k) y\right)\)
proof (induct \(n\) )
    case 0
    show? case by simp
next
    case (Suc n)
    let ?s \(=\lambda k\). \((n\) choose \(k) *\) ffact \(k x *\) ffact \((n-k) y\)
    let ? \(t=\lambda k\). ffact \(k x *\) ffact (Suc \(n-k) y\)
    let ?u \(=\lambda k\). ffact \((\) Suc \(k) x *\) ffact \((n-k) y\)
```

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

```
    also have .. = ?v 0 + (\sumk=Suc 0..Suc n. ?v k)
    by (simp only: sum.shift-bounds-cl-Suc-ivl diff-Suc-Suc mult.assoc)
    also have ... = (\sumk=0..Suc n. (Suc n choose k)* ffact kx* ffact (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-group-add, comm-semiring-1-cancel, ring-1\}
shows ffact \(n(x+y)=\left(\sum k=0 . . n\right.\). of-nat \((n\) choose \(k) *\) ffact \(k x *\) ffact \((n-\) k) \(y\) )
proof (induct \(n\) )
case 0
show ? case by simp
```


## next

```
case (Suc n)
let ?s \(=\lambda k\). of-nat ( \(n\) choose \(k) *\) ffact \(k x *\) ffact \((n-k) y\)
let ? \(t=\lambda k\). ffact \(k x *\) ffact \((\) Suc \(n-k) y\)
let ? \(u=\lambda k\). ffact (Suc k) \(x *\) ffact \((n-k) y\)
have ffact (Suc \(n)(x+y)=(x+y-\) of-nat \(n) *\) ffact \(n(x+y)\)
by (simp add: ffact-Suc-rev)
also have \(\ldots=(x+y-\) of-nat \(n) *\left(\sum k=0\right.\)..n. of-nat \((n\) choose \(k) *\) ffact \(k\)
\(x *\) ffact \((n-k) y)\)
using Suc.hyps by simp
also have \(\ldots=\left(\sum k=0\right.\)..n. ?s \(k *(x+y-\) of-nat \(\left.n)\right)\)
by (simp add: mult.commute sum-distrib-left)
also have \(\ldots=\left(\sum k=0\right.\)..n. ?s \(k *((y+\) of-nat \(k-\) of-nat \(n)+(x-\) of-nat k))
by (auto intro: sum.cong simp add: diff-add-eq add-diff-eq add.commute)
also have \(\ldots=\left(\sum k=0\right.\)..n. of-nat \((n\) choose \(k) *(\) ?t \(k+\) ?u \(\left.k)\right)\)
proof -
\{
fix \(k\)
assume \(k \leq n\)
have ?u \(k=\) ffact \(k x *\) ffact \((n-k) y *(x-\) 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 \(k=\) ffact \(k x *\) ffact \((n-k) y *(y+\)
of-nat \(k\) - 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 \(k *((y+\) of-nat \(k-\) of-nat \(n)+(x-\) of-nat \(k))=o f\)-nat \((n\) 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 ... = (\sumk=0..n. of-nat (n choose k)* ?t k) + (\sumk=0..n. of-nat
(n choose k) * ?u k)
    by (simp add: sum.distrib distrib-left mult.commute mult.left-commute)
    also have ... = ?t 0 + (\sumk=0..n. of-nat ( n choose k + ( n choose Suc k))*
?u k)
    proof -
    have ... = (?t 0 + (\sumk=0..n. of-nat (n choose Suc k)*?u k)) + (\sumk=
0..n. of-nat (n choose k) * ?u k)
    proof -
            have (\sumk=Suc 0..n. of-nat (n choose k)* ?t k)=(\sumk=0..n. of-nat ( }
choose Suc k) * ?u k)
            proof -
            have (\sumk= Suc 0..n. of-nat (n choose k)*?t k)=(\sumk=Suc 0..Suc n.
of-nat (n choose k) * ?t k)
            by (simp add: binomial-eq-0)
            also have ... = (sum ((\lambdak. of-nat (n choose k)* ?t k) o Suc) {0..n})
            by (simp only: sum.reindex[symmetric, of Suc] inj-Suc image-Suc-atLeastAtMost)
                    also have ... = (\sumk=0..n. of-nat (n choose Suc k)*?u k)
                    by simp
            finally show ?thesis.
            qed
            from this show ?thesis
            by (simp add: sum.atLeast-Suc-atMost[of - - \lambdak. of-nat (n choose k) * ?t k])
    qed
    also have .. = ?t 0 + (\sumk=0..n. of-nat ( n choose k + ( n choose Suc k))
* ?u k)
            by (simp add: distrib-right sum.distrib)
            finally show ?thesis .
qed
also have ... = (\sumk=0..Suc n. of-nat (Suc n choose k)* ffact kx * ffact (Suc
n-k) y)
    proof -
    let ?v = \lambdak. of-nat (Suc n choose k)*ffact kx*ffact (Suc n - k) y
    have ... = ?v 0 + (\sumk=0..n. of-nat (Suc n choose (Suc k))* ?u k)
        by simp
    also have .. = ?v 0 + (\sumk=Suc 0..Suc n. ?v k)
        by (simp only: sum.shift-bounds-cl-Suc-ivl diff-Suc-Suc mult.assoc)
    also have \ldots=( (\sumk=0..Suc n. of-nat (Suc n choose k) * ffact k x * ffact
(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<br>Discrete-Summation.Factorials<br>begin

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

```
lemma ffact-add-nat:
    shows ffact \(k(n+m)=\left(\sum i \leq k .(k\right.\) choose \(i) *\) ffact \(i n *\) ffact \(\left.(k-i) m\right)\)
proof -
    have ffact \(k(n+m)=\) fact \(k *((n+m)\) choose \(k)\)
        by (simp only: ffact-eq-fact-mult-binomial)
    also have \(\ldots=\) fact \(k *\left(\sum i \leq k .(n\right.\) choose \(i) *(m\) choose \(\left.(k-i))\right)\)
        by (simp only: vandermonde)
    also have \(\ldots=\left(\sum i \leq k\right.\). fact \(k *(n\) choose \(i) *(m\) choose \(\left.(k-i))\right)\)
        by (simp add: sum-distrib-left field-simps)
    also have \(\ldots=\left(\sum i \leq k\right.\). \((\) fact \(i *\) fact \((k-i) *(k\) choose \(i)) *(n\) choose \(i) *\)
( \(m\) choose \((k-i)\) )
        by (simp add: binomial-fact-lemma)
    also have \(\ldots=\left(\sum i \leq k .(k\right.\) choose \(i) *(\) fact \(i *(n\) choose \(i)) *(\) fact \((k-i) *\)
( \(m\) choose \((k-i)\) ))
        by (auto intro: sum.cong)
    also have \(\ldots=\left(\sum i \leq k\right.\). \((k\) choose \(i) *\) ffact \(i n *\) ffact \(\left.(k-i) m\right)\)
        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 AttributionShareAlike 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 seemless 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.

