Diplomarbeit

Algorithmen für mehrfache Summen

von
Torsten Sprenger
Universität Kassel

eingereicht bei
Prof. Dr. Wolfram Koepf

Juli 2004

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Algorithmen für mehrfache Summen

In den letzten Jahren wurden dank einiger Arbeiten von Herbert S. Wilf und Doron Zeilberger neue Algorithmen zur hypergeometrischen Summation entwickelt. Der Zeilberger-Algorithmus ist dabei wohl einer der effizientesten Algorithmen und ein wichtiges Hilfsmittel zum Beweisen von hypergeometrischen Identitäten, in denen einfache Summen auftreten. Leider stellt sich heraus, dass im Fall von mehrfachen Summen, wie z.B. Doppelsummen, kein vergleichbares Instrument zur Verfügung steht. Eine Möglichkeit, eine Rekursionsgleichung einer mehrfachen Summe zu ermitteln, besteht darin, den Zeilberger-Algorithmus iterativ anzuwenden. Allerdings ist dieses Vorhaben meistens nicht von Erfolg gekrönt, da man oftmals keine geschlossene Form für die innere Summe erhält. Aus diesem Grund betrachten wir einen Algorithmus zur mehrfachen hypergeometrischen Summation, der von Wilf und Zeilberger ([Wil92]) vorgeschlagen wurde und der, unter gewissen Voraussetzungen, theoretisch immer zum Erfolg führt. Dieser Algorithmus, der auf dem Fasenmyer-Algorithmus („Sister Celine's technique") basiert, ist aber leider sehr ineffizient und dadurch in der Praxis kaum nutzbar. Kurt Wegschaider ([Weg97]) untersuchte diesen Algorithmus genauer und fand einige Verbesserungen, die diesen durchführbar machen. Mit Hilfe seiner Implementation in Mathematica kann man zahlreiche hypergeometrische Identitäten mit Mehrfachsummen beweisen. In dieser Diplomarbeit werden Wegschaider's Ergebnisse aus [Weg97] dargestellt, wobei besonderer Wert auf den algorithmischen Aspekt gelegt wird. Außerdem wird ein Software-Paket für Maple zur mehrfachen Summation präsentiert, in dem auch eine C-Bibliothek und die Methode des automatischen Filterns zur Effizienzsteigerung zum Einsatz kommen. Es wird ferner eine neue Heuristik zur systematischen Suche nach Rekursionsgleichungen vorgestellt.
Um einen Überblick über die Wirkungsweise der Algorithmen und deren Nutzen zu bekommen, betrachten wir das folgende allgemeine Problem: Wir möchten die Identität

$$\sum_{k_1}\sum_{k_2}\cdots\sum_{k_r} F(n,k_1,k_2,\ldots,k_r) = G(n) \tag{1.1}$$

für alle nichtnegative, ganzzahlige n beweisen, wobei $F(n, k_1, k_2, \ldots, k_r)$ und $G(n)$ hypergeometrische Terme[1] sein sollen. Mit den Algorithmen, die in dieser Arbeit entwickelt werden, können wir eine Rekursionsgleichung für den Term $F(n, k_1, k_2, \ldots, k_r)$ aufstellen. Aus dieser Rekursion ermitteln wir, unter bestimmten Voraussetzungen, eine homogene und lineare Rekursionsgleichung

$$a_0(n)S(n) + a_1(n)S(n-1) + \cdots + a_m(n)S(n-m) = 0$$

für die mehrfache Summe $S(n)$ der linken Seite von (1.1). Erfüllt $G(n)$ diese Rekursionsgleichung und sind hinreichend viele Anfangswerte identisch ($S(i) = G(i)$ für $i = 0, \ldots, m-1$), so folgt die Identität (1.1). Ist $G(n)$ auch eine Summe, so bestimmt man für $G(n)$ ebenfalls eine Rekursionsgleichung. Wenn die Rekursionen und die Anfangswerte übereinstimmen, ist wiederum die Gleichheit von $S(n)$ und $G(n)$ bewiesen.
Diese Arbeit ist in fünf Kapitel aufgeteilt. In diesem Kapitel werden zunächst einige grundlegende Definitionen zusammengestellt, die in den darauf folgenden Abschnitten benötigt werden. Unser Basis-Algorithmus von Fasenmyer und die zugrunde liegende Theorie werden im zweiten Teil dieser Arbeit erläutert. Im dritten Kapitel werden wesentliche Verbesserungen des Algorithmus aufgezeigt. Eine Beschreibung des Maple-Packages `multsum`, welches sämtliche Algorithmen, die wir behandeln werden, beinhaltet, ist im vorletzten Kapitel zu finden. Abschließend werden einige Anwendungsbeispiele der Algorithmen aufgeführt.

1.2 Grundlagen und Notationen

In diesem Abschnitt werden Notationen und Definitionen vorgestellt, die in der vorliegenden Diplomarbeit Verwendung finden. Wir führen Operatoren und einige Kurzschreibweisen für Vektoren und mehrfache Summen ein, die sich später als sehr nützlich erweisen. Außerdem werden an dieser Stelle verschiedene grundlegende Begriffe, wie z.B. Polynom oder Binomialkoeffizient, definiert, die Grundvoraussetzung für das Studium der nächsten Kapitel sind.

1.2.1 Zahlenbereiche

Wir verwenden folgende Mengen:
die Menge der ganzen Zahlen $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$,
die Menge der natürlichen Zahlen $\mathbb{N} = \{1, 2, 3, \ldots\}$,
die Menge der nichtnegativen ganzen Zahlen $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$,
die Menge der negativen ganzen Zahlen $-\mathbb{N} = \{-1, -2, -3, \ldots\}$,
die Menge der nichtpositiven ganzen Zahlen $-\mathbb{N}_0 = -\mathbb{N} \cup \{0\} = \mathbb{Z} \setminus \mathbb{N}$,
die Menge der rationalen Zahlen $\mathbb{Q}$,
die Menge der reellen Zahlen $\mathbb{R}$ und
die Menge der komplexen Zahlen $\mathbb{C}$.
Das ganzzahlige Intervall mit den Grenzen i und j wird als $[i..j]$ geschrieben, d.h. es ist $[i..j] = \{k \in \mathbb{Z} \mid i \le k \le j\}$. Wir bezeichnen mit $\mathbb{K}$ einen Körper der Charakteristik 0. Für alle $\mathbb{K}$ soll dann $\mathbb{K}^* = \mathbb{K} \setminus \{0\}$ sein.

[1] siehe Seite 12

1.2.2 Mengen

Sei M eine Menge auf der eine Addition $+$ definiert ist und S_1,S_2 Teilmengen von M. Ferner sei $k \in M$. Wir definieren[2]:

$$\begin{aligned} S_1 + S_2 &:= \{s_1 + s_2 \mid s_1 \in S_1 \text{ und } s_2 \in S_2\} \\ S_1 + k &:= \{s_1 + k \mid s_1 \in S_1\}. \end{aligned}$$

Die Anzahl der Elemente einer Menge S wird mit $|S|$ bezeichnet. $|S|$ nennt man auch die **Größe** von S.

1.2.3 Vektoren

Ein n-**Vektor v** ist ein n-Tupel $(v_1, \ldots, v_n)$, wobei $n \in \mathbb{N}$ und die v_i's Variablen oder Zahlen darstellen. Demnach erscheinen Vektoren immer fett gedruckt[3], wohingegen Variablen, wie üblich, kursiv dargestellt werden. Die i-te Komponente eines n-Vektors $\mathbf{v}$ bezeichnet man mit v_i und die i-te Komponente eines k-ten n-Vektors $\mathbf{v}_k$ mit v_{ki}. Oftmals lassen wir n weg und sagen einfach $\mathbf{v}$ ist ein Vektor.
Da sehr häufig Vektoren in dieser Arbeit auftauchen, ist es sinnvoll, einige Kurzschreibweisen einzuführen. Seien dazu $\mathbf{i}$ und $\mathbf{j}$ n-Vektoren. Wir definieren:

$$\begin{aligned} \mathbf{i} \leq \mathbf{j} &:\Leftrightarrow i_r \leq j_r \quad \forall\, r \in [1..n] \\ \mathbf{i}^{\mathbf{j}} &:= i_1^{j_1} \cdot \ \cdots \ \cdot i_n^{j_n} \\ \mathbf{i} \cdot \mathbf{j} &:= i_1 j_1 + \cdots + i_n j_n \\ [\mathbf{i}..\mathbf{j}] &:= \{\mathbf{k} \in \mathbb{Z}^n \mid \mathbf{i} \leq \mathbf{k} \leq \mathbf{j}\}. \end{aligned}$$

Ferner gelte bspw. $(i, \mathbf{j}) \in \mathbb{Z}^{1+n}$ und $(i, \mathbf{j}) = (i, j_1, \ldots, j_n)$, wenn $i \in \mathbb{Z}$ und $\mathbf{j} \in \mathbb{Z}^n$.

1.2.4 Mehrfache Summen

Sei $S \subseteq \mathbb{Z}^n$ und f eine Funktion, die für alle $\mathbf{i} \in S$ definiert ist. Dann definieren wir die **mehrfache Summe von** f **über** S rekursiv mit Hilfe von Vektoren wie folgt:

$$\sum_{\mathbf{i} \in S} f(\mathbf{i}) := \sum_{i_1 \in S_1} \sum_{\bar{\mathbf{i}} \in S(i_1)} f(i_1, \bar{\mathbf{i}}),$$

mit $S_1 = \{i_1 \in \mathbb{Z} \mid \exists\, \bar{\mathbf{i}} \in \mathbb{Z}^{n-1} \text{ mit } (i_1, \bar{\mathbf{i}}) \in S\}$ und $S(i_1) = \{\bar{\mathbf{i}} \in \mathbb{Z}^{n-1} \mid (i_1, \bar{\mathbf{i}}) \in S\}$. Ist $\mathbf{0}, \mathbf{I} \in \mathbb{Z}^n$, dann schreiben wir gelegentlich auch

$$\sum_{\mathbf{i}=\mathbf{0}}^{\mathbf{I}} f(\mathbf{i}) := \sum_{\mathbf{i} \in [\mathbf{0}..\mathbf{I}]} f(\mathbf{i}).$$

[2] $\emptyset + S = \emptyset$ und $\emptyset + k = \emptyset$

[3] Der Nullvektor wird geschrieben als $\mathbf{0}$

1.2.5 Polynome und rationale Funktionen

Sei R ein Ring. Ein **Polynom p in x mit Koeffizienten in R** ist definiert als

$$p(x) := \sum_{k=0}^{n} a_k x^k$$

mit $a_k \in R$ und $n \in \mathbb{N}_0$. Wir schreiben dann auch $p \in R[x]$, wobei $R[x]$ der **Polynomring über R in der Variablen x** ist. Für $p \in R[x]$ definiere $\deg_x(p) := \max\{k \mid a_k \neq 0\}$ den **Grad** des Polynoms p in x, wenn p verschieden ist vom Nullpolynom[4].
Diese Definitionen lassen sich für den Fall mehrerer Variablen übernehmen. Wir schreiben für ein Polynom p in $x_1, \ldots, x_m$ mit Koeffizienten in R

$$p(\mathbf{x}) = \sum_{\mathbf{k}=\mathbf{0}}^{\mathbf{n}} a_{\mathbf{k}} \mathbf{x}^{\mathbf{k}}$$

mit $\mathbf{x} = (x_1, \ldots, x_m)$, $a_{\mathbf{k}} \in R$ und $\mathbf{n} \in \mathbb{N}_0^m$. Der Polynomring über R in $x_1, \ldots, x_m$ wird mit $R[x_1, \ldots, x_m]$ oder $R[\mathbf{x}]$ bezeichnet und der Grad des vom Nullpolynom verschiedenen Polynoms p in $x_1, \ldots, x_m$ mit $\deg_{\mathbf{x}}(p) := \max\{k_1 + \cdots + k_m \mid a_{\mathbf{k}} \neq 0\}$.
Der Quotientenkörper von $R[x]$ bzw. $R[\mathbf{x}]$ ist der **Körper der rationalen Funktionen über R** und wird geschrieben als $R(x)$ bzw. $R(\mathbf{x})$. Ist $r \in R(\mathbf{x})$, dann ist r eine **rationale Funktion über R in $\mathbf{x}$** und r lässt sich darstellen als Quotient von Polynomen $p, q \in R[\mathbf{x}]$, wobei $q \not\equiv 0$ sein muss.

1.2.6 Die Gamma-Funktion

Nach Euler ist die **Gamma-Funktion** definiert als

$$\Gamma(z) := \int_0^\infty t^{z-1} e^{-t}\, dt.$$

Das parameterabhängige uneigentliche Integral existiert für alle $z \in \mathbb{C}$ mit $\operatorname{Re} z > 0$. Es gilt die **Funktionalgleichung** der Gamma-Funktion

$$\Gamma(z+1) = z\,\Gamma(z). \tag{1.2}$$

Man erhält (1.2) durch partielle Integration. Aus der Funktionalgleichung folgt

$$\Gamma(z) = \frac{\Gamma(z+1)}{z}, \tag{1.3}$$

wobei die rechte Seite von (1.3) für jedes $z \in \mathbb{C} \setminus \{0\}$ mit $\operatorname{Re} z > -1$ erklärt ist. Es liegt also nahe die Funktion $z \mapsto \Gamma(z)$ zunächst auf $\{z \in \mathbb{C} \setminus \{0\} \mid \operatorname{Re} z > -1\}$ fortzusetzen. Wir wenden (1.3) erneut auf die Erweiterung der Gamma-Funktion an und setzen diese auf $\{z \in \mathbb{C} \setminus \{0, -1\} \mid \operatorname{Re} z > -2\}$ fort. Fahren wir so fort, können wir schließlich die Gamma-Funktion mittels Funktionalgleichung für alle $z \in \mathbb{C} \setminus -\mathbb{N}_0$ definieren. Alle $k \in -\mathbb{N}_0$ sind Polstellen erster Ordnung von Γ und die Funktion $\frac{1}{\Gamma}$ ist wegen $\frac{1}{\Gamma(k)} = 0$ für alle $k \in -\mathbb{N}_0$ im funktionentheoretischen Sinne eine ganze Funktion.

[4] $\deg_x(p) := -\infty$ für $p \equiv 0$

1.2.7 Fakultäten

Mit Hilfe der Funktionalgleichung der Gamma-Funktion und des Anfangswertes

$$\Gamma(1) = \int_0^\infty e^{-t}\,dt = -e^{-t}\Big|_0^\infty = 1$$

folgt per vollständiger Induktion

$$\Gamma(k+1) = k! \tag{1.4}$$

für alle $k \in \mathbb{N}_0$. Wir definieren demzufolge die **Fakultät** von $k \in \mathbb{C} \setminus -\mathbb{N}$ durch (1.4).

1.2.8 Pochhammersymbole

Sei $z \in \mathbb{C}$ und $k \in \mathbb{Z}$. Das **Pochhammersymbol** $(z)_k$ ist gegeben durch

$$(z)_k := \begin{cases} z(z+1)\cdots(z+k-1) & ,k \in \mathbb{N} \\ 1 & ,k = 0 \\ \frac{1}{(z-1)(z-2)\cdots(z+k)} & ,k \in -\mathbb{N} \text{ und } z \notin \{1,2,\ldots,-k\} \end{cases}$$

Insbesondere ergibt sich aus der Definition, dass $(z)_k = 0$, wenn $z \in -\mathbb{N}_0$ und $z+k > 0$. Durch Induktion bekommen wir ausgehend von der Funktionalgleichung (1.2) für $k \in \mathbb{N}$, $z \in \mathbb{C}$ und $z + k \notin -\mathbb{N}_0$

$$\frac{\Gamma(z+k)}{\Gamma(z)} = z(z+1)\cdots(z+k-1) = (z)_k. \tag{1.5}$$

Für $k \in -\mathbb{N}$, $z \in \mathbb{C}$ und $z + k \notin -\mathbb{N}_0$[5] gilt mit (1.5)

$$\frac{\Gamma(z+k)}{\Gamma(z)} = \frac{1}{\frac{\Gamma(z)}{\Gamma(z+k)}} = \frac{1}{(z+k)(z+k+1)\cdots(z-1)} = (z)_k.$$

Zusammenfassend gilt also

$$\frac{\Gamma(z+k)}{\Gamma(z)} = (z)_k \tag{1.6}$$

für alle $k \in \mathbb{Z}$, $z \in \mathbb{C}$ und $z + k \notin -\mathbb{N}_0$. Die Gleichung (1.6) zeigt den Zusammenhang der Gamma-Funktion mit dem Pochhammersymbol auf. Außerdem gilt die Gleichung

$$(z)_k = (-1)^k(1-z-k)_k,$$

die wiederum durch eine Fallunterscheidung über die Definition des Pochhammersymbols bewiesen wird. Ersetzt man in der letzten Gleichung $(z)_k$ durch die linke Seite von (1.6), so erhält man

$$\frac{\Gamma(z+k)}{\Gamma(z)} = (-1)^k \frac{\Gamma(1-z)}{\Gamma(1-z-k)}. \tag{1.7}$$

[5] also $z \notin -\mathbb{N}_0 - k$ und insbesondere $z \notin -\mathbb{N}_0$

Eine weitere wichtige Identität für Pochhammersymbole ist

$$\frac{(z)_j}{(z)_k} = (z+k)_{j-k}. \tag{1.8}$$

Für $j = 0$ bekommt man

$$\frac{1}{(z)_k} = (z+k)_{-k}. \tag{1.9}$$

1.2.9 Binomialkoeffizienten

Der **Binomialkoeffizient** für $z, k \in \mathbb{C}$ ist definiert als

$$\binom{z}{k} := \frac{\Gamma(z+1)}{\Gamma(k+1)\Gamma(z-k+1)} \tag{1.10}$$

unter der Voraussetzung, dass $z \notin -\mathbb{N}$ ist. Für $z \in -\mathbb{N}$ ist

$$\binom{z}{k} := \begin{cases} 0 & , k \in -\mathbb{N} \\ (-1)^k \frac{\Gamma(k-z)}{\Gamma(k+1)\Gamma(-z)} & , k \in \mathbb{N}_0. \end{cases}$$

Insbesondere folgt für $z \in \mathbb{N}_0$ aus (1.10), dass $\binom{z}{k} = 0$, wenn $k \in -\mathbb{N}$ oder $k - z \in \mathbb{N}$ ist. Weitere wichtige Gleichungen im Zusammenhang mit Binomialkoeffizienten sind

$$\binom{z}{k} = (-1)^k \binom{k-z-1}{k} = \frac{(-1)^k}{k!}(-z)_k \tag{1.11}$$

für alle $k \in \mathbb{N}_0$ und $z \in \mathbb{C}$, der bekannte Additionssatz

$$\binom{z}{k} = \binom{z-1}{k-1} + \binom{z-1}{k} \tag{1.12}$$

und das Additionstheorem

$$\binom{z}{0} + \binom{z+1}{1} + \binom{z+2}{2} + \cdots + \binom{z+k}{k} = \binom{z+k+1}{k} \tag{1.13}$$

für alle $z, k \in \mathbb{C}$, für die die auftretenden Binomialkoeffizienten definiert sind.

1.2.10 Rekursionsgleichungen und Operatoren

Sei R ein Ring, $r \in \mathbb{N}$ und $F : D \subseteq \mathbb{C}^{1+r} \to R$ eine Funktion in den Veränderlichen n und $\mathbf{k} = (k_1, \ldots, k_r)$. Ferner sei S eine nichtleere und endliche Menge mit $S \subseteq \mathbb{Z}^{1+r}$ und $a_{i\mathbf{j}} \in R[n, \mathbf{k}]$ für $(i, \mathbf{j}) \in S$ mit $a_{i\mathbf{j}} \not\equiv 0$ für mindestens ein $(i, \mathbf{j}) \in S$. Gilt

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n, \mathbf{k}) F(n-i, \mathbf{k}-\mathbf{j}) = 0 \tag{1.14}$$

für alle $(n, \mathbf{k}) \in D$ mit $(n-i, \mathbf{k}-\mathbf{j}) \in D$ für alle $(i, \mathbf{j}) \in S$, so erfüllt F eine **homogene und lineare Rekursionsgleichung in** n und $\mathbf{k}$ **mit polynomialen Koeffizienten**. Man spricht dann auch von einer **holonomen Rekursion**. Die

Ordnung der Rekursion **in** n ist gegeben durch $\max\{i_1 - i_2 \mid (i_1, \mathbf{j}_1), (i_2, \mathbf{j}_2) \in S \text{ und } a_{i_1\mathbf{j}_1}, a_{i_2\mathbf{j}_2} \not\equiv 0\}$.

Um eine elegantere Schreibweise für Rekursionen zu erhalten, führen wir Operatoren ein. Ein (rückwärtiger) **Verschiebungs-Operator** L_n bzw. L_{k_m} für eine Funktion F in n und $\mathbf{k} = (k_1, \ldots, k_m, \ldots, k_r)$ ist definiert als

$$L_n F(n, \mathbf{k}) := F(n-1, \mathbf{k})$$

bzw.

$$L_{k_m} F(n, k_1, \ldots, k_m, \ldots, k_r) := F(n, k_1, \ldots, k_m - 1, \ldots, k_r),$$

wobei $m \in [1..r]$ ist.
Operatoren kann man addieren, multiplizieren (im Sinne von hintereinander schalten) und invertieren. Für zwei Operatoren L_n und L_k für eine Funktion F in n und k gelten bspw. folgende Rechenregeln:

$$\begin{aligned}(L_n + L_k)F(n,k) &= F(n-1,k) + F(n,k-1)\\ L_n L_k F(n,k) &= F(n-1,k-1)\\ L_n^{-1} F(n,k) &= F(n+1,k).\end{aligned}$$

Die Rekursionsgleichung

$$\sum_{(i,j)\in S} a_{ij}(n,k) F(n-i, k-j) = 0$$

für F in n und k kann man mit Hilfe des **Rekursions-Operators**

$$P(n, k, L_n, L_k) = \sum_{(i,j)\in S} a_{ij}(n,k) L_n^i L_k^j$$

als

$$P(n, k, L_n, L_k) F(n,k) = 0$$

schreiben. P ist dann ein **Rekursions-Operator für** F. Die Menge aller $\sum_{(i,j)\in S} a_{ij}(n,k) L_n^i L_k^j$ bildet einen nichtkommutativen Ring, den **Ring der polynomialen Rekursions-Operatoren in** n und k. Wir bezeichnen ihn mit $R[n,k]\langle L_n, L_k\rangle$. Für eine Funktion in n und $\mathbf{k}$ schreibt man entsprechend $R[n,\mathbf{k}]\langle L_n, \mathbf{L_k}\rangle$. Die Nichtkommutativität der Multiplikation zweier Rekursions-Operatoren drückt sich in den beiden Gleichungen

$$nL_nF(n,k) = nF(n-1,k) = L_n(n+1)F(n,k) \quad \text{, also} \quad nL_n = L_n(n+1)$$

und

$$kL_kF(n,k) = kF(n,k-1) = L_k(k+1)F(n,k) \quad \text{, also} \quad kL_k = L_k(k+1)$$

aus.
Außerdem benötigen wir für bestimmte Rekursionen, die wir später einführen werden, einen **Differenzen-Operator**, den wir als $\Delta_{k_m} := 1 - L_{k_m}$ definieren, wobei L_{k_m} wiederum für einen Operator steht, der auf der Variablen k_m operiert und 1 der identische Operator ist.

1.2.11 Hypergeometrische Terme

Sei R ein Ring, $r \in \mathbb{N}$ und $F : D \subseteq \mathbb{C}^{1+r} \to R$ eine Funktion in den Variablen n und $\mathbf{k} = (k_1, \ldots, k_r)$. Gilt

$$\frac{F(n-1,\mathbf{k})}{F(n,\mathbf{k})} \in R(n,\mathbf{k}) \quad \text{und} \quad \frac{F(n,k_1,\ldots,k_m-1,\ldots,k_r)}{F(n,k_1,\ldots,k_m,\ldots,k_r)} \in R(n,\mathbf{k}),$$

für alle $m \in [1..r]$, dann nennen wir F einen **hypergeometrischen Term** (oder **hypergeometrisch**) und $n, k_1, \ldots, k_r$ **hypergeometrische Variablen**. Ein hypergeometrischer Term $F(n,\mathbf{k})$ besitzt einen **endlichen Träger** $T_F(n)$, wenn für jedes $n \in \mathbb{N}_0$ die Menge

$$T_F(n) = \{\mathbf{k} \mid (n,\mathbf{k}) \in D \cap (\mathbb{N}_0 \times \mathbb{Z}^r) \text{ und } F(n,\mathbf{k}) \neq 0\}$$

endlich ist. Beispiele für hypergeometrische Terme sind Polynome, rationale Funktionen, Gamma-Funktionen, Pochhammersymbole, Fakultäten und Binomialkoeffizienten oder Kombinationen von diesen. Gamma-Funktionen, Pochhammersymbole, Fakultäten und Binomialkoeffizienten sind allerdings nur hypergeometrisch, wenn die Argumente **ganzzahlig-linear** in den auftretenden Variablen $n, k_1, \ldots, k_r$ sind, d.h. die Argumente lassen sich in der Form $an + \mathbf{b} \cdot \mathbf{k} + c$ mit $a \in \mathbb{Z}$, $\mathbf{b} \in \mathbb{Z}^r$ und $c \in \mathbb{K}$ schreiben. In den folgenden Kapiteln wählen wir $R = \mathbb{C}$ und $\mathbb{K} = \mathbb{C}$. Unsere Aufgabe wird es sein, Summen der Art

$$S(n) = \sum_{\mathbf{k}} F(n,\mathbf{k}) = \sum_{k_1=-\infty}^{\infty} \cdots \sum_{k_r=-\infty}^{\infty} F(n,k_1,\ldots,k_r),$$

zu bestimmen oder zu vereinfachen, wobei F ein hypergeometrischer Term ist. Man nennt n die **Hauptvariable** und $k_1, \ldots, k_r$ die **Summationsvariablen**. In den zahlreichen Beispielen der folgenden Kapitel und in der Praxis treten in einem hypergeometrischen Term oft zusätzliche Parameter $\boldsymbol{\alpha}$ auf. Alle Sätze und Definitionen, die wir bzgl. hypergeometrischer Terme $F(n,\mathbf{k})$ formulieren, können wir auf Terme $F(n,\mathbf{k},\boldsymbol{\alpha})$ übertragen. Folglich beschränken wir uns der Einfachheit halber auf hypergeometrische Terme $F(n,\mathbf{k})$ ohne optionale Parameter.

Kapitel 2

Der Fasenmyer-Algorithmus

Auf den Algorithmus von Celine Fasenmyer beziehen sich alle Algorithmen, die wir in dieser Diplomarbeit behandeln. Wilf und Zeilberger betrachteten diesen Algorithmus als Erste in Bezug auf mehrfache Summen. Der Algorithmus bestimmt eine Rekursionsgleichung für einen hypergeometrischen Term F in n und $\mathbf{k}$. Dabei ist zu bemerken, dass die Koeffizienten der Rekursion Polynome in n sind und nicht mehr von den Summationsvariablen $\mathbf{k}$ abhängen. In diesem Kapitel werden wir den Algorithmus angeben und sehen, dass er nicht bei allen hypergeometrischen Termen eine Rekursionsgleichung bestimmen kann. Diese Tatsache führt uns zu dem Begriff des zulässigen hypergeometrischen Terms. Wir werden am Ende des Kapitels zeigen können, dass diese speziellen hypergeometrischen Terme immer einer Rekursionsgleichung genügen, deren Koeffizienten Polynome in n sind.

2.1 k-freie Rekursionsgleichungen

Bevor wir näher auf den Algorithmus von Celine Fasenmyer eingehen, definieren wir zunächst, was der Algorithmus als Ergebnis liefert.

Definition 2.1 *Sei $F : D \subseteq \mathbb{C}^{1+r} \to \mathbb{C}$ ein hypergeometrischer Term in n und $\mathbf{k} = (k_1, \ldots, k_r)$ und S eine* **Strukturmenge für** *F, d.h. S sei eine nichtleere, endliche Menge mit $S \subseteq \mathbb{Z}^{1+r}$. Ferner sei $a_{i\mathbf{j}} \in \mathbb{C}[n]$ für alle $(i, \mathbf{j}) \in S$ und $a_{i\mathbf{j}} \not\equiv 0$ für mindestens ein $(i, \mathbf{j}) \in S$. Es gelte*

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n) F(n - i, \mathbf{k} - \mathbf{j}) = 0 \tag{2.1}$$

für alle $(n, \mathbf{k}) \in D$ mit $(n - i, \mathbf{k} - \mathbf{j}) \in D$ für alle $(i, \mathbf{j}) \in S$. Dann ist (2.1) eine **k-freie Rekursionsgleichung** *(oder kurz* **k-freie Rekursion***)* **für** *F. Wenn F einer $\mathbf{k}$-freien Rekursionsgleichung bzgl. der Strukturmenge S genügt, so nennen wir S eine* **Struktur für** *F.*

2.2 Der Algorithmus

Wir erläutern den Fasenmyer-Algorithmus an einem einfachen Beispiel. Gesucht ist eine k-freie Rekursionsgleichung für den Binomialkoeffizienten $\binom{n}{k}$.

Beispiel 2.1 Sei

$$F(n,k) = \binom{n}{k}.$$

Wir machen den allgemeinen Ansatz

$$\sum_{(i,j)\in S} a_{ij}(n)F(n-i,k-j) = 0$$

mit einer Strukturmenge S für F und rationalen Funktionen a_{ij} in n. Wir setzen $S = \{(0,0),(0,1),(1,0),(1,1)\}$ und bekommen[1]

$$a_{00}F(n,k) + a_{01}F(n,k-1) + a_{10}F(n-1,k) + a_{11}F(n-1,k-1) = 0. \quad (2.2)$$

Wir teilen die Gleichung (2.2) durch $F(n,k)$ und erhalten

$$a_{00} + a_{01}\frac{F(n,k-1)}{F(n,k)} + a_{10}\frac{F(n-1,k)}{F(n,k)} + a_{11}\frac{F(n-1,k-1)}{F(n,k)} = 0.$$

Durch Einsetzen von $\binom{n}{k}$ für $F(n,k)$ und Vereinfachen ergibt sich

$$a_{00} + a_{01}\frac{k}{n-k+1} + a_{10}\frac{n-k}{n} + a_{11}\frac{k}{n} = 0.$$

Multiplikation mit einem gemeinsamen Nenner, in diesem Fall $n(n-k+1)$, bringt

$$a_{00}n(n-k+1) + a_{01}kn + a_{10}(n-k)(n-k+1) + a_{11}k(n-k+1) = 0.$$

Wir multiplizieren die Terme aus und betrachten die linke Seite der letzten Gleichung als Polynom in k und erhalten somit

$$\begin{gathered}(a_{10}-a_{11})k^2 \\ +(-na_{00}+na_{01}-(2n+1)a_{10}+(n+1)a_{11})k \\ +n(n+1)a_{00}+n(n+1)a_{10} = 0.\end{gathered}$$

Durch Koeffizientenvergleich bekommen wir das lineare Gleichungssystem

$$\begin{array}{rcrcrcrcl} & & & & a_{10} & - & a_{11} & = & 0 \\ -na_{00} & + & na_{01} & - & (2n+1)a_{10} & + & (n+1)a_{11} & = & 0 \\ n(n+1)a_{00} & & & + & n(n+1)a_{10} & & & = & 0. \end{array}$$

Wir lösen das lineare Gleichungssystem mit Maple und erhalten

```
> solve({a[1,0]-a[1,1],
> -n*a[0,0]+n*a[0,1]-(2*n+1)*a[1,0]+(n+1)*a[1,1],
> n*(n+1)*a[0,0]+n*(n+1)*a[1,0]},{a[0,0],a[0,1],a[1,0],a[1,1]});
```

$$\{a_{1,1} = -a_{0,0},\ a_{1,0} = -a_{0,0},\ a_{0,0} = a_{0,0},\ a_{0,1} = 0\}$$

Einsetzen der Lösung in (2.2) resultiert in der Rekursionsgleichung

$$a_{00}F(n,k) - a_{00}F(n-1,k) - a_{00}F(n-1,k-1) = 0.$$

[1] wir vernachlässigen in den folgenden Betrachtungen das Argument n der a_{ij}'s

Für $a_{00}(n) = 1$ bekommen wir den Additionssatz für Binomialkoeffizienten[2]

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \tag{2.3}$$

der eine homogene und lineare Rekursionsgleichung der Ordnung 1 in n darstellt.

Mit Hilfe der Rekursionsgleichung (2.3), die wir mit dem Fasenmyer-Algorithmus erhalten haben, können wir dann leicht

$$S(n) = \sum_{k=-\infty}^{\infty} F(n,k)$$

bestimmen, indem wir die Rekursion über alle $k \in \mathbb{Z}$ summieren. Es ergibt sich

$$S(n) = S(n-1) + S(n-1) = 2S(n-1).$$

Mit dem Anfangswert $S(0) = 1$ bekommt man schließlich durch Induktion

$$S(n) = \sum_{k=-\infty}^{\infty} \binom{n}{k} = \sum_{k=0}^{n} \binom{n}{k} = 2^n$$

für alle $n \in \mathbb{N}_0$.

Der Algorithmus von Fasenmyer lässt sich allgemein folgendermaßen beschreiben:

Algorithmus 2.1 (Fasenmyer-Algorithmus)

1. **Eingabe**: ein hypergeometrischer Term $F(n, \mathbf{k})$ mit $\mathbf{k} = (k_1, \ldots, k_r)$ eine Strukturmenge S für F.

2. Mache den Ansatz

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n) F(n-i, \mathbf{k}-\mathbf{j}) = 0, \tag{2.4}$$

   wobei $a_{i\mathbf{j}}$ rationale Funktionen in n sind.

3. Teile (2.4) durch $F(n, \mathbf{k})$ und vereinfache die auftretenden Quotienten $F(n-i, \mathbf{k}-\mathbf{j})/F(n, \mathbf{k})$. Man erhält

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n) R_{i\mathbf{j}}(n, \mathbf{k}) = 0, \tag{2.5}$$

   wobei $R_{i\mathbf{j}}$ rationale Funktionen in n und $\mathbf{k}$ sind.

[2] siehe auch (1.12)

4. Multipliziere mit einem gemeinsamen Nenner, um die Gleichung

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n)p_{i\mathbf{j}}(n,\mathbf{k}) = 0 \tag{2.6}$$

zu bekommen, bei der $p_{i\mathbf{j}}$ Polynome in n und $\mathbf{k}$ sind.

5. Betrachte die linke Seite der Polynomgleichung als Polynom in $\mathbf{k}$ und vergleiche die Koeffizienten der $k_1^{l_1}\cdots k_r^{l_r}$ mit 0, um ein homogenes lineares Gleichungssystem für die $a_{i\mathbf{j}}$'s über dem Körper der rationalen Funktionen in n zu erhalten.

6. Wenn nur die triviale Lösung $a_{i\mathbf{j}} \equiv 0$ für alle $(i,\mathbf{j}) \in S$ existiert, dann gibt es keine $\mathbf{k}$-freie Rekursionsgleichung für F bzgl. S.
Ausgabe: „Es existiert keine $\mathbf{k}$-freie Rekursion für F bzgl. S."
Andernfalls setze die erhaltene Lösung in (2.4) ein und multipliziere mit einem gemeinsamen Nenner.
Ausgabe: die durch den letzten Schritt entstandene, i. allg. nicht eindeutige $\mathbf{k}$-freie Rekursion für F bzgl. der Struktur S.

Beweis Da F ein hypergeometrischer Term ist, erhält man durch Induktion, dass alle $F(n-i,\mathbf{k}-\mathbf{j})/F(n,\mathbf{k})$ rationale Funktionen in n und $\mathbf{k}$ sind. Also folgt die Gleichung (2.5), die zu (2.4) äquivalent ist. Multiplizieren wir (2.5) mit einem gemeinsamen Nenner, so erhält man die zu (2.5) äquivalente Polynomgleichung (2.6). Die linke Seite der Gleichung (2.6) ist genau dann 0, wenn sie das Nullpolynom in $\mathbf{k}$ ist, was den Koeffizientenvergleich in Schritt 5 rechtfertigt. Demzufolge besitzt das daraus resultierende homogene und lineare Gleichungssystem für die $a_{i\mathbf{j}}$'s über dem Körper der rationalen Funktionen in n genau dann eine nichttriviale Lösung, wenn es eine $\mathbf{k}$-freie Rekursionsgleichung für F bzgl. S gibt. □

Algorithmus 2.1 ist in der Funktion `find_krec` aus dem im vierten Kapitel beschriebenen Maple-Package `multsum` implementiert. Wir verifizieren unser Ergebnis von Beispiel 2.1[3]

```
> find_krec(binomial(n,k),n,k,{[0,0],[0,1],[1,0],[1,1]});
number of equations: 3  number of variables: 4
```

$$\{-\mathrm{F}(n,\,k)+\mathrm{F}(n-1,\,k)+\mathrm{F}(n-1,\,k-1)=0\}$$

Unser eigentliches Ziel wird es sein, diesen Algorithmus auf hypergeometrische Terme F in n und $\mathbf{k}=(k_1,\ldots,k_r)$ anzuwenden, bei denen $r \geq 2$ ist. Für den Fall $r=1$ ist der Zeilberger-Algorithmus dem Fasenmyer-Algorithmus vorzuziehen, da er wesentlich effizienter ist.

Beispiel 2.2 Wir verwenden nun `find_krec`, um für den hypergeometrischen Term

$$F(n,i,j) = \binom{n}{j}\binom{j}{i}x^i y^{j-i} z^{n-j}$$

[3]Mit der Strukturmenge S={(0,0),(1,0),(1,1)} erhält man dasselbe Ergebnis

eine (i,j)-freie Rekursionsgleichung zu finden. Wir erhalten für die Strukturmenge $S = \{(0,0,0),(1,0,0),(1,0,1),(1,1,1)\}$

```
> find_krec(binomial(n,j)*binomial(j,i)*x^i*y^(j-i)*z^(n-j),
> n,[i,j],{[0,0,0],[1,0,0],[1,0,1],[1,1,1]});

number of equations: 3  number of variables: 4
```

$$\{\mathrm{F}(n,\,i,\,j) - z\,\mathrm{F}(n-1,\,i,\,j) - y\,\mathrm{F}(n-1,\,i,\,j-1) - x\,\mathrm{F}(n-1,\,i-1,\,j-1) = 0\}$$

Für jede andere Wahl von S mit $|S| < 4$ besitzt das zustande kommende lineare Gleichungssystem keine nichttriviale Lösung und demzufolge auch keine (i,j)-freie Rekursion bzgl. S. Um eine geschlossene Form für

$$S(n) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} F(n,i,j)$$

zu bekommen, summieren wir die mit `find_krec` bestimmte Rekursionsgleichung über alle $i,j \in \mathbb{Z}$. Dann folgt

$$S(n) = (x+y+z)S(n-1)$$

und mit dem Anfangswert $S(0) = 1$ schließlich

$$S(n) = (x+y+z)^n$$

für alle $n \in \mathbb{N}_0$.

2.3 Zulässige hypergeometrische Terme

Wir wenden uns nun der Frage zu, ob zu jedem hypergeometrischen Term $F(n,\mathbf{k})$ eine Strukturmenge S existiert, so dass F einer $\mathbf{k}$-freien Rekursionsgleichung bzgl. S genügt. Eine Antwort darauf gibt uns folgendes

Lemma 2.1 *Für den hypergeometrischen Term $F(n,k) = 1/(n^2+k^2)$ existiert keine k-freie Rekursionsgleichung.*

Beweis Sei also $F(n,k) = 1/(n^2+k^2)$. $F(n,k)$ ist offenbar hypergeometrisch. Annahme: Es existiert eine k-freie Rekursionsgleichung für F.
Dann gilt

$$\sum_{(i,j)\in S} \frac{a_{ij}(n)}{(n-i)^2+(k-j)^2} = 0$$

für eine Struktur S für F, wobei nicht alle $a_{ij} \equiv 0$ sein dürfen. Die linke Seite der Rekursionsgleichung ist demzufolge eine nichttriviale Summe von meromorphen Termen bzgl. k. Betrachten wir die Rekursion an einer Polstelle eines Summanden, so bleiben alle anderen Summanden endlich, was zu dem Widerspruch $\infty =$ endlich führt. □

Der Fasenmyer-Algorithmus kann demnach bei bestimmten hypergeometrischen Termen nicht zum Erfolg führen. Aus diesem Grund betrachten wir im Folgenden hypergeometrische Terme, die sich aus einem Polynom, einem Ausdruck mit Gamma-Funktionen und aus Potenzen zusammenstellen. Diese speziellen hypergeometrischen Terme definieren wir wie folgt

Definition 2.2 *Seien n und $\mathbf{k} = (k_1, \ldots, k_r)$ Variablen mit $r \in \mathbb{N}$. Ferner sei $pp, qq \in \mathbb{N}_0$. Es gelte außerdem*

- $P \in \mathbb{C}[n, \mathbf{k}]$
- $a_p \in \mathbb{Z}, \mathbf{b}_p \in \mathbb{Z}^r$ *und* $c_p \in \mathbb{C}$ *für alle* $p \in [1..pp]$
- $u_q \in \mathbb{Z}, \mathbf{v}_q \in \mathbb{Z}^r$ *und* $w_q \in \mathbb{C}$ *für alle* $q \in [1..qq]$
- $x_0, x_1, \ldots, x_r \in \mathbb{C}$.

Dann ist die Funktion $F : D \subseteq \mathbb{C}^{1+r} \to \mathbb{C}$ definiert als

$$F(n, \mathbf{k}) := P(n, \mathbf{k}) \frac{\prod_{p=1}^{pp} \Gamma(a_p n + \mathbf{b}_p \cdot \mathbf{k} + c_p)}{\prod_{q=1}^{qq} \Gamma(u_q n + \mathbf{v}_q \cdot \mathbf{k} + w_q)} x_0^n x_1^{k_1} \cdots x_r^{k_r} \tag{2.7}$$

ein **zulässiger hypergeometrischer Term in** *n und $\mathbf{k}$. Wir nennen die Argumente $a_p n + \mathbf{b}_p \cdot \mathbf{k} + c_p$* **Zähler-Fakultätsterme von** *F und die Argumente $u_q n + \mathbf{v}_q \cdot \mathbf{k} + w_q$* **Nenner-Fakultätsterme von** *F. Wir sagen F ist* **wohldefiniert in** *$(n_0, \mathbf{k}_0)$, wenn $a_p n_0 + \mathbf{b}_p \cdot \mathbf{k}_0 + c_p \notin -\mathbb{N}_0$ für alle $p \in [1..pp]$ und $x_0 x_1 \cdots x_r \neq 0$. Die* **Menge der wohldefinierten Werte von** *F ist gegeben durch*

$$D_F := \left\{ (n_0, \mathbf{k}_0) \in \mathbb{C}^{1+r} \mid F \text{ ist wohldefiniert in } (n_0, \mathbf{k}_0) \right\}.$$

Man beachte, dass x_0^n bzw. $x_m^{k_m}$ als $e^{n \log(x_0)}$ bzw. $e^{k_m \log(x_m)}$ zu interpretieren ist, wobei log der Hauptwert des Logarithmus und $m \in [1..r]$ ist. Der hypergeometrische Term $1/(n^2 + k^2)$ ist nach obiger Definition nicht zulässig. Allerdings kann man gewisse rationale Funktionen, nämlich diejenigen, deren Nennerpolynom $Q(n, \mathbf{k})$ in ganzzahlig-lineare Faktoren zerfällt, in einen zulässigen hypergeometrischen Term umwandeln. Ist nämlich $un + \mathbf{v} \cdot \mathbf{k} + w$ ein Faktor von $Q(n, \mathbf{k})$, so kann man diesen Faktor auf den Gamma-Ausdruck wie folgt umwälzen

$$\frac{1}{(un + \mathbf{v} \cdot \mathbf{k} + w)} = \frac{\Gamma(un + \mathbf{v} \cdot \mathbf{k} + w)}{\Gamma(un + \mathbf{v} \cdot \mathbf{k} + w + 1)}.$$

Beispiel 2.3 Sei

$${}_pF_q(x) = \sum_{k=0}^{\infty} \frac{(a_1 n + b_1)_k \cdots (a_p n + b_p)_k}{(u_1 n + v_1)_k \cdots (u_q n + v_q)_k} \frac{x^k}{k!}$$

die verallgemeinerte hypergeometrische Funktion, wobei alle a_i's und u_i's ganze Zahlen sein sollen. Dann ist der Summand von ${}_pF_q(x)$ ein zulässiger hypergeometrischer Term, denn er lässt sich schreiben als

$$\frac{\Gamma(a_1 n + b_1 + k) \cdots \Gamma(a_p n + b_p + k) \Gamma(u_1 n + v_1) \cdots \Gamma(u_q n + v_q)}{\Gamma(a_1 n + b_1) \cdots \Gamma(a_p n + b_p) \Gamma(u_1 n + v_1 + k) \cdots \Gamma(u_q n + v_q + k) \Gamma(k + 1)} x^k.$$

Man beachte, dass $\Gamma(a_i n + b_i + k)/\Gamma(a_i n + b_i)$ gemäß (1.7) umformuliert werden muss, falls $a_i n + b_i \in -\mathbb{N}_0$. Ist letzteres für eine ungerade Anzahl von i's der Fall, so ändert sich die Potenz x^k und aus x^k wird $(-x)^k$.

Beispiel 2.4 Der Binomialkoeffizient $\binom{an+b}{uk+v}$ mit $a, u \in \mathbb{Z}$ lässt sich darstellen als

$$\binom{an+b}{uk+v} = \frac{\Gamma(an+b+1)}{\Gamma(uk+v+1)\Gamma(an-uk+b-v+1)},$$

sofern $an+b \notin -\mathbb{N}$ ist. Somit ist die rechte Seite der letzten Gleichung der zu dem Binomialkoeffizenten gehörige zulässige hypergeometrische Term. Es ist anzumerken, dass der zulässige Term für negative $an+b$ nicht definiert ist, der Binomialkoeffizient hingegen schon.

Mit Hilfe der Funktionen `check_hyper` bzw. `check_proper` kann man überprüfen, ob ein Term hypergeometrisch bzw. zulässig ist. Außerdem ist es mit der Funktion `properterm` möglich, einen zulässigen Term in seine Bestandteile zu zerlegen. So erhalten wir bspw. für[4]

```
> properterm((-1)^k*(n+k)*binomial(n,k),n,k);
```

$$[n+k,\ [[1]],\ [[0]],\ [1],\ [1,\ 0],\ [[-1,\ 1]],\ [1,\ 1]],\ (-1)^k].$$

2.4 Der Existenzsatz für k-freie Rekursionen

In den folgenden Betrachtungen gehen wir immer von zulässigen hypergeometrischen Termen aus. Ziel dieses Abschnitts ist es, zu zeigen, dass zu jedem beliebigen zulässigen Term eine **k**-freie Rekursionsgleichung existiert.

2.4.1 Die rationalen Funktionen $R^F_{i\mathbf{j}}$

Wenden wir den Fasenmyer-Algorithmus auf zulässige Terme an, so können wir die rationalen Funktionen, die im dritten Schritt des Algorithmus 2.1 entstehen, explizit angeben.

Definition 2.3 *Sei $(i, \mathbf{j}) \in \mathbb{Z}^{1+r}$ und F ein zulässiger hypergeometrischer Term in n und $\mathbf{k} = (k_1, \ldots, k_r)$ wie in Definition 2.2. Wir definieren die rationale Funktion $R^F_{i\mathbf{j}}$ in n und $\mathbf{k}$ als*

$$R^F_{i\mathbf{j}}(n, \mathbf{k}) := \frac{P(n-i, \mathbf{k}-\mathbf{j})}{P(n, \mathbf{k})} \frac{\prod_{p=1}^{pp}(a_p n + \mathbf{b}_p \cdot \mathbf{k} + c_p)_{-ia_p - \mathbf{j}\cdot\mathbf{b}_p}}{\prod_{q=1}^{qq}(u_q n + \mathbf{v}_q \cdot \mathbf{k} + w_q)_{-iu_q - \mathbf{j}\cdot\mathbf{v}_q}} x_0^{-i} x_1^{-j_1} \cdots x_r^{-j_r}. \tag{2.8}$$

Man beachte, dass das Pochhammersymbol im Zähler als Ausdruck im Nenner auftritt, falls $-ia_p - \mathbf{j}\cdot\mathbf{b}_p \in -\mathbb{N}$ ist[5]. Umgekehrt gilt für das Pochhammersymbol im Nenner, dass es tatsächlich Teil des Zählers ist, falls $-iu_q - \mathbf{j}\cdot\mathbf{v}_q \in -\mathbb{N}$ ist.

Lemma 2.2 *Sei $(i, \mathbf{j}) \in \mathbb{Z}^{1+r}$ und F ein zulässiger hypergeometrischer Term in n und $\mathbf{k} = (k_1, \ldots, k_r)$ wie in Definition 2.2. Es gilt*

$$\frac{F(n-i, \mathbf{k}-\mathbf{j})}{F(n, \mathbf{k})} = R^F_{i\mathbf{j}}(n, \mathbf{k}) \tag{2.9}$$

für alle $(n, \mathbf{k}) \in D_F$ mit $F(n, \mathbf{k}) \neq 0$ und $(n-i, \mathbf{k}-\mathbf{j}) \in D_F$.

[4] genaue Beschreibung siehe viertes Kapitel: Das Maple-Package `multsum`

[5] siehe (1.9)

Beweis Zunächst gilt

$$\begin{aligned} F(n,\mathbf{k}) \neq 0 \quad &\Rightarrow \quad P(n,\mathbf{k}) \neq 0 \text{ und } u_q n + \mathbf{v}_q \cdot \mathbf{k} + w_q \notin -\mathbb{N}_0 \text{ für alle } q \\ &\Rightarrow \quad P(n,\mathbf{k}) \neq 0 \text{ und } (u_q n + \mathbf{v}_q \cdot \mathbf{k} + w_q)_{-iu_q - \mathbf{j}\cdot\mathbf{v}_q} \neq 0 \text{ für alle } q \\ &\Rightarrow \quad \text{Der Nenner von } R^F_{i\mathbf{j}}(n,\mathbf{k}) \text{ ist von 0 verschieden.} \end{aligned}$$

Außerdem ist $a_p(n-i)+\mathbf{b}_p\cdot(\mathbf{k}-\mathbf{j})+c_p \notin -\mathbb{N}_0$, da F wohldefiniert in $(n-i,\mathbf{k}-\mathbf{j})$ ist und somit gilt

$$\begin{aligned} \frac{\Gamma(a_p(n-i)+\mathbf{b}_p\cdot(\mathbf{k}-\mathbf{j})+c_p)}{\Gamma(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)} &= \frac{\Gamma(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p-ia_p-\mathbf{j}\cdot\mathbf{b}_p)}{\Gamma(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)} \\ &\overset{(1.6)}{=} (a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-ia_p-\mathbf{j}\cdot\mathbf{b}_p} \end{aligned}$$

für alle $p \in [1..pp]$ und wegen $u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q \notin -\mathbb{N}_0$

$$\begin{aligned} \frac{\Gamma(u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q)}{\Gamma(u_q(n-i)+\mathbf{v}_q\cdot(\mathbf{k}-\mathbf{j})+w_q)} &= \frac{\Gamma(u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q)}{\Gamma(u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q-iu_q-\mathbf{j}\cdot\mathbf{v}_q)} \\ &\overset{(1.6)}{=} (u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q-iu_q-\mathbf{j}\cdot\mathbf{v}_q)_{iu_q+\mathbf{j}\cdot\mathbf{v}_q} \\ &\overset{(1.9)}{=} \frac{1}{(u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q)_{-iu_q-\mathbf{j}\cdot\mathbf{v}_q}} \end{aligned}$$

für alle $q \in [1..qq]$, was alles zeigt. □

Das Lemma beweist insbesondere, dass jeder zulässige Term tatsächlich hypergeometrisch ist, so dass Definition 2.2 gerechtfertigt ist.

2.4.2 Das assoziierte Polynom P^F_S

Haben wir die rationale Gleichung

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n) R^F_{i\mathbf{j}}(n,\mathbf{k}) = 0 \tag{2.10}$$

gegeben, können wir diese problemlos in eine polynomiale Gleichung überführen. Wir multiplizieren (2.10) einfach mit dem kleinsten gemeinsamen Vielfachen der Nennerpolynome und teilen (2.10) zusätzlich durch den größten gemeinsamen Teiler der Zählerpolynome und erhalten die Gleichung

$$P^F_S(n,\mathbf{k}) = \sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n) p_{i\mathbf{j}}(n,\mathbf{k}) = 0, \tag{2.11}$$

deren Polynom P^F_S wir in diesem Abschnitt untersuchen werden. Wir führen dazu einige Begriffe ein.

Definition 2.4 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$ wie in Definition 2.2 und S eine Strukturmenge für F. Ferner sei $p \in [1..pp]$ und $q \in [1..qq]$. Wir nennen einen Punkt $(i_0,\mathbf{j}_0) \in S$*

- **Zähler-Randpunkt von** F, *wenn*

$$i_0 a_p + \mathbf{j}_0 \cdot \mathbf{b}_p \geq i a_p + \mathbf{j} \cdot \mathbf{b}_p \qquad \textit{für alle } (i, \mathbf{j}) \in S.$$

- **Nenner-Randpunkt von** F, *wenn*

$$i_0 u_q + \mathbf{j}_0 \cdot \mathbf{v}_q \leq i u_q + \mathbf{j} \cdot \mathbf{v}_q \qquad \textit{für alle } (i, \mathbf{j}) \in S.$$

- **Differenz-Randpunkt von** F, *wenn*

$$i_0 \alpha_F + \mathbf{j}_0 \cdot \boldsymbol{\beta}_F \leq i \alpha_F + \mathbf{j} \cdot \boldsymbol{\beta}_F \qquad \textit{für alle } (i, \mathbf{j}) \in S,$$

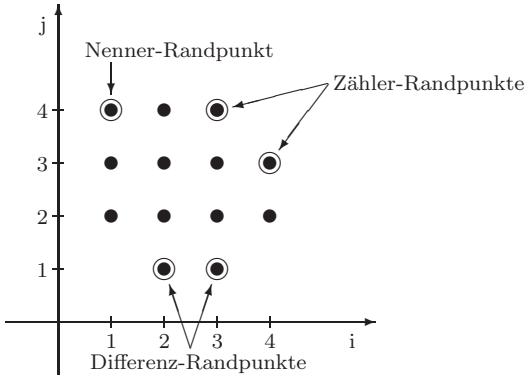
wobei

$$\alpha_F = \sum_{\substack{p=1 \\ \mathbf{b}_p \neq \mathbf{0}}}^{pp} a_p - \sum_{\substack{q=1 \\ \mathbf{v}_q \neq \mathbf{0}}}^{qq} u_q \quad \textit{und} \quad \boldsymbol{\beta}_F = \sum_{p=1}^{pp} \mathbf{b}_p - \sum_{q=1}^{qq} \mathbf{v}_q. \tag{2.12}$$

Einen Zähler-Randpunkt bezeichnen wir mit $(\overline{i}_p, \overline{\mathbf{j}}_p)$, *einen Nenner-Randpunkt mit* $(\underline{i}_q, \underline{\mathbf{j}}_q)$ *und einen Differenz-Randpunkt mit* $(i^-, \mathbf{j}^-)$. *Den Vektor* $(\alpha_F, \boldsymbol{\beta}_F)$ *nennen wir die* **Fakultäts-Differenz**.

Alle in Definition 2.4 eingeführten Punkte liegen auf dem Rand der konvexen Hülle von S und sind in diesem Sinne tatsächlich Randpunkte. Wir betrachten dazu

Beispiel 2.5 Sei $F(n,k) = \Gamma(n+k)/\Gamma(n-k)$ und $S = \{(1,2),(1,3),(1,4),(2,1),(2,2),(2,3),(2,4),(3,1),(3,2),(3,3),(3,4),(4,2),(4,3)\}$. Dann kann man alle Punkte von S in ein (i,j)-Koordinatensystem einzeichnen. Die umrandeten Punkte sind Randpunkte, die man leicht mit Hilfe von Definition 2.4 berechnen kann.



Mit den Randpunkten eines zulässigen Terms lässt sich die linke Seite von (2.11), also das Polynom P_S^F in n und $\mathbf{k}$, wie folgt angeben

Definition 2.5 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$ wie in Definition 2.2 und S eine Strukturmenge für F. Wir nennen das Polynom*

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n)P(n-i,\mathbf{k}-\mathbf{j})G^F_{i\mathbf{j}}(n,\mathbf{k})x_0^{i_{\max}-i}x_1^{j_{\max 1}-j_1}\cdots x_r^{j_{\max r}-j_r} \tag{2.13}$$

mit

$$G^F_{i\mathbf{j}}(n,\mathbf{k}) = \frac{\prod_{p=1}^{pp}(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p-\overline{i}_pa_p-\overline{\mathbf{j}}_p\cdot\mathbf{b}_p)_{(\overline{i}_p-i)a_p+(\overline{\mathbf{j}}_p-\mathbf{j})\cdot\mathbf{b}_p}}{\prod_{q=1}^{qq}(u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q-\underline{i}_qu_q-\underline{\mathbf{j}}_q\cdot\mathbf{v}_q)_{(\underline{i}_q-i)u_q+(\underline{\mathbf{j}}_q-\mathbf{j})\cdot\mathbf{v}_q}} \tag{2.14}$$

das **assoziierte Polynom von F bzgl.** *S und bezeichnen es mit P^F_S, wobei $i_{\max}=\max_{(i,\mathbf{j})\in S} i$ und $j_{\max m}=\max_{(i,\mathbf{j})\in S} j_m$ für $m\in[1..r]$.*

P^F_S ist offensichtlich ein Polynom, denn alle Indizes der Pochhammersymbole im Zähler sind nichtnegativ und alle Indizes der Pochhammersymbole im Nenner sind nichtpositiv.

Satz 2.3 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$ wie in Definition 2.2 und S eine Strukturmenge für F. Es gilt*

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n)R^F_{i\mathbf{j}}(n,\mathbf{k}) = 0 \Leftrightarrow P^F_S(n,\mathbf{k}) = 0$$

für alle $(n,\mathbf{k})\in D_F$ mit $F(n,\mathbf{k})\neq 0$ und $(n-i,\mathbf{k}-\mathbf{j})\in D_F$ für alle $(i,\mathbf{j})\in S$.

Beweis Zunächst multiplizieren wir jede rationale Funktion $R^F_{i\mathbf{j}}(n,\mathbf{k})$ mit $P(n,\mathbf{k})$ und $x_0^{i_{\max}}x_1^{j_{\max 1}}\cdots x_r^{j_{\max r}}$. Um die polynomiale Gleichung zu erhalten, multiplizieren wir die rationale Gleichung mit dem kleinsten gemeinsamen Vielfachen der Nennerpolynome und teilen durch den größten gemeinsamen Teiler der Zählerpolynome. Betrachten wir die Ausdrücke $(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-ia_p-\mathbf{j}\cdot\mathbf{b}_p}$.

- Gilt $\overline{i}_pa_p+\overline{\mathbf{j}}_p\cdot\mathbf{b}_p\leq 0$, dann folgt $-ia_p-\mathbf{j}\cdot\mathbf{b}_p\geq-\overline{i}_pa_p-\overline{\mathbf{j}}_p\cdot\mathbf{b}_p\geq 0$ für alle $(i,\mathbf{j})\in S$ und somit sind alle $(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-ia_p-\mathbf{j}\cdot\mathbf{b}_p}$ im Zähler. Wir teilen deshalb die rationale Gleichung durch den größten gemeinsamen Teiler $(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-\overline{i}_pa_p-\overline{\mathbf{j}}_p\cdot\mathbf{b}_p}$.

- Gilt $\overline{i}_pa_p+\overline{\mathbf{j}}_pb_p>0$, dann existieren $(i,\mathbf{j})\in S$, für die die Ausdrücke $(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-ia_p-\mathbf{j}\cdot\mathbf{b}_p}$ Polynome im Nenner sind. Das kleinste gemeinsame Vielfache dieser Polynome ist $1/(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-\overline{i}_pa_p-\overline{\mathbf{j}}_p\cdot\mathbf{b}_p}$. Wir multiplizieren die rationale Gleichung folglich mit diesem Polynom.

In beiden Fällen teilen wir die rationale Gleichung, also insbesondere jeden Ausdruck $(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-ia_p-\mathbf{j}\cdot\mathbf{b}_p}$, durch $(a_pn+\mathbf{b}_p\cdot\mathbf{k}+c_p)_{-\overline{i}_pa_p-\overline{\mathbf{j}}_p\cdot\mathbf{b}_p}$ und erhalten mittels (1.8) den Zähler von $G^F_{i\mathbf{j}}(n,\mathbf{k})$. Durch eine analoge Vorgehensweise für die Ausdrücke $(u_qn+\mathbf{v}_q\cdot\mathbf{k}+w_q)_{-iu_q-\mathbf{j}\cdot\mathbf{v}_q}$ bekommen wir schließlich mit Hilfe von den Nenner-Randpunkten den Nenner von $G^F_{i\mathbf{j}}(n,\mathbf{k})$. Damit ist alles bewiesen. □

Eine wichtige Eigenschaft zulässiger hypergeometrischer Terme wird im Folgenden festgehalten.

Definition 2.6 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$ wie in Definition 2.2. Wir nennen F* **irreduzibel**, *wenn $P(n,\mathbf{k}) = 1$ ist und es keinen Zähler-Fakultätsterm s und keinen Nenner-Fakultätsterm t von F gibt, so dass $s - t \in \mathbb{Z}$. Ist F nicht irreduzibel, dann nennen wir F* **reduzibel**.

Die Polynomgleichung $P_S^F(n,\mathbf{k}) = 0$ kann für irreduzible zulässige Terme F nicht weiter vereinfacht werden, was uns folgender Hilfssatz lehrt.

Lemma 2.4 *Sei F ein irreduzibler, zulässiger Term in n und $\mathbf{k}$ und S eine Strukturmenge für F. Ferner sei $P_S^F = \sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n)p_{i\mathbf{j}}(n,\mathbf{k})$ das assoziierte Polynom von F bzgl. S. Dann existiert kein Polynom q in n und $\mathbf{k}$ mit $\deg_{n,\mathbf{k}}(q) \geq 1$, das alle $p_{i\mathbf{j}}(n,\mathbf{k})$ teilt.*

Beweis Sei s ein beliebiger Zähler-Fakultätsterm von dem irreduziblen Term F und $(\overline{i}_s, \overline{\mathbf{j}}_s)$ der entsprechende Zähler-Randpunkt. Betrachte nun $s + l$ für ein $l \in \mathbb{Z}$. Wir zeigen, dass es ein Polynom $p_{i\mathbf{j}}$ gibt, für das $s + l$ kein Teiler ist. Zunächst ist zu bemerken, dass für ein weiteren Zähler-Fakultätsterm s', für den $s - s' \in \mathbb{Z}$ gilt, der zugehörige Zähler-Randpunkt ebenfalls $(\overline{i}_s, \overline{\mathbf{j}}_s)$ ist. Daraus folgt, dass $s + l$ das Polynom $p_{\overline{i}_s\overline{\mathbf{j}}_s}$ nur in zwei Fällen teilt. Entweder ist $s + l$ ein Teiler von mindestens einem der Nennerausdrücke von $G^F_{\overline{i}_s\overline{\mathbf{j}}_s}(n,\mathbf{k})$ oder $s+l$ ist ein Teiler des Polynoms $P(n-\overline{i}_s, \mathbf{k}-\overline{\mathbf{j}}_s)$. Da jedoch F irreduzibel ist, gilt $s - t \notin \mathbb{Z}$ für alle Nenner-Fakultätsterme t, so dass der erste Fall ausgeschlossen ist. Der zweite Fall ist ebenfalls wegen der Irreduzibilität von F nicht möglich, da $P(n-\overline{i}_s, \mathbf{k}-\overline{\mathbf{j}}_s) = 1$ ist. Somit teilt $s + l$ nicht das Polynom $p_{\overline{i}_s\overline{\mathbf{j}}_s}$. Betrachten wir einen beliebigen Nenner-Fakultätsterm t, so erhalten wir durch eine analoge Schlußweise, dass $p_{\underline{i}_t,\underline{\mathbf{j}}_t}$ kein Vielfaches von $t + m$ für ein $m \in \mathbb{Z}$ sein kann. Somit existiert zu jedem Faktor eines Polynoms $p_{i\mathbf{j}}$ ein Polynom $p_{i'\mathbf{j}'}$, welches kein Vielfaches von diesem Faktor ist. □

Für reduzible Terme ist es ebenfalls nicht sehr wahrscheinlich, dass man P_S^F vereinfachen kann. Wir beschränken uns der Einfachheit halber in den kommenden Sätzen auf irreduzible Terme, da die folgende Gradformel nur für diese exakt ist. Man kann aber die nachstehenden Aussagen modifizieren, so dass der noch zu beweisende Existenzsatz auch für reduzible Terme gilt. Wir haben nun alle notwendigen Begriffe zusammen, um folgenden Satz formulieren zu können

Satz 2.5 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$ und S eine Strukturmenge für F. Der Grad des assoziierten Polynoms von F bzgl. S in $\mathbf{k}$, also $\deg_{\mathbf{k}}(P_S^F)$, lässt sich angeben als*

$$\deg_{\mathbf{k}}(P) - (i^-\alpha_F + \mathbf{j}^- \cdot \boldsymbol{\beta}_F) + \sum_{\substack{p=1\\ \mathbf{b}_p\neq\mathbf{0}}}^{pp} (\overline{i}_p a_p + \overline{\mathbf{j}}_p \cdot \mathbf{b}_p) - \sum_{\substack{q=1\\ \mathbf{v}_q\neq\mathbf{0}}}^{qq} (\underline{i}_q u_q + \underline{\mathbf{j}}_q \cdot \mathbf{v}_q) \quad (2.15)$$

und ist somit nur abhängig von den Randpunkten.

Beweis Sei $m \geq 0$. Dann beträgt der Grad von $(\mathbf{b}\cdot\mathbf{k})_m$ bzw. $1/(\mathbf{v}\cdot\mathbf{k})_{-m}$ in $\mathbf{k}$ offensichtlich m, sofern $\mathbf{b} \neq \mathbf{0}$ bzw. $\mathbf{v} \neq \mathbf{0}$ ist. Demnach ergibt sich der Grad

eines Summanden von P_S^F durch die Summe aller Indizes der Pochhammersymbole + dem Grad des Polynoms P[6]. Bildet man das Maximum der Grade der Summanden, erhält man schließlich den Grad des assoziierten Polynoms P_S^F in $\mathbf{k}$, also

$$\begin{aligned}
\deg_{\mathbf{k}}(P_S^F) &= \max_{(i,\mathbf{j})\in S}\Bigg(\deg_{\mathbf{k}}(P) + \sum_{\substack{p=1\\ \mathbf{b}_p\neq\mathbf{0}}}^{pp}((\overline{i}_p - i)a_p + (\overline{\mathbf{j}}_p - \mathbf{j})\cdot\mathbf{b}_p) \\
&\qquad - \sum_{\substack{q=1\\ \mathbf{v}_q\neq\mathbf{0}}}^{qq}((\underline{i}_q - i)u_q + (\underline{\mathbf{j}}_q - \mathbf{j})\cdot\mathbf{v}_q)\Bigg) \\
&= \deg_{\mathbf{k}}(P) + \sum_{\substack{p=1\\ \mathbf{b}_p\neq\mathbf{0}}}^{pp}(\overline{i}_p a_p + \overline{\mathbf{j}}_p\cdot\mathbf{b}_p) - \sum_{\substack{q=1\\ \mathbf{v}_q\neq\mathbf{0}}}^{qq}(\underline{i}_q u_q + \underline{\mathbf{j}}_q\cdot\mathbf{v}_q) \\
&\quad + \max_{(i,\mathbf{j})\in S}\Bigg(-i\Bigg(\sum_{\substack{p=1\\ \mathbf{b}_p\neq\mathbf{0}}}^{pp}a_p - \sum_{\substack{q=1\\ \mathbf{v}_q\neq\mathbf{0}}}^{qq}u_q\Bigg) - \mathbf{j}\cdot\Bigg(\sum_{p=1}^{pp}\mathbf{b}_p - \sum_{q=1}^{qq}\mathbf{v}_q\Bigg)\Bigg) \\
&= \deg_{\mathbf{k}}(P) + \sum_{\substack{p=1\\ \mathbf{b}_p\neq\mathbf{0}}}^{pp}(\overline{i}_p a_p + \overline{\mathbf{j}}_p\cdot\mathbf{b}_p) - \sum_{\substack{q=1\\ \mathbf{v}_q\neq\mathbf{0}}}^{qq}(\underline{i}_q u_q + \underline{\mathbf{j}}_q\cdot\mathbf{v}_q) \\
&\quad -(i^-\alpha_F + \mathbf{j}^-\boldsymbol{\beta}_F).
\end{aligned}$$

□

Wir sind nun in der Lage, nur durch Kenntnis des Grades von P_S^F, zu überprüfen, ob es eine $\mathbf{k}$-freie Rekursionsgleichung für F bzgl. der Strukturmenge S gibt. Es gilt nämlich folgendes

Theorem 2.6 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k} = (k_1,\ldots,k_r)$ und S eine Strukturmenge für F. Gilt*

$$\binom{\deg_{\mathbf{k}}(P_S^F)+r}{r} < |S|, \tag{2.16}$$

dann ist S eine Struktur für F.

Beweis S ist eine Struktur von F, wenn das lineare Gleichungssystem, dass man durch Vergleichen der Koeffizienten aller $k_1^{l_1}\cdots k_r^{l_r}$ von P_S^F mit 0 erhält, mehr Variablen als Gleichungen besitzt. In diesem Fall bekommt man eine nichttriviale Lösung und demzufolge eine $\mathbf{k}$-freie Rekursionsgleichung für F. Die Anzahl der Variablen ist offenbar $|S|$. Zu zeigen bleibt, dass die maximale Anzahl der Gleichungen $\binom{d+r}{r}$ entspricht, wobei $d = \deg_{\mathbf{k}}(P_S^F)$ ist. Die maximale Anzahl der Produkte $k_1^{l_1}\cdots k_r^{l_r}$ mit $l_1+\cdots+l_r = c$ und $0\le c\le d$ beträgt $\binom{r+c-1}{c}$. Also folgt mit Hilfe von (1.13) und dem Symmetriesatz für Binomialkoeffizienten

$$\sum_{c=0}^{d}\binom{r-1+c}{c} = \binom{r+d}{d} = \binom{d+r}{r}$$

und damit die Behauptung. □

Dieses Theorem können wir nun dazu verwenden, die zentrale Aussage dieses Abschnittes zu beweisen. Doch zunächst

[6] es gilt $\deg_{\mathbf{k}}(P(n,\mathbf{k})) = \deg_{\mathbf{k}}(P(n-i,\mathbf{k}-\mathbf{j}))$ für alle $(i,\mathbf{j})\in S$

Definition 2.7 *Wir bezeichnen mit* $S_{I\mathbf{J}} := \{(i,\mathbf{j}) \in \mathbb{Z}^{1+r} \mid i \in [0..I] \text{ und } \mathbf{j} \in [\mathbf{0}..\mathbf{J}]\}$ *eine* **rechteckige Strukturmenge** *(oder* **Standard-Strukturmenge***) für einen zulässigen Term* F *in* n *und* $\mathbf{k} = (k_1, \ldots, k_r)$*, wobei* $I \in \mathbb{N}_0$ *und* $\mathbf{J} \in \mathbb{N}_0^r$ *ist.*

Eine weitere Vorbereitung auf den Existenzsatz stellt folgendes Lemma dar

Lemma 2.7 *Sei* F *ein zulässiger hypergeometrischer Term in* n *und* $\mathbf{k}$ *und* $S_{I\mathbf{J}}$ *eine rechteckige Strukturmenge für* F*. Der Grad des assoziierten Polynoms von* F *bzgl.* $S_{I\mathbf{J}}$ *in* $\mathbf{k}$*, also* $\deg_{\mathbf{k}}(P^F_{S_{I\mathbf{J}}})$*, lässt sich dann angeben als*

$$\begin{aligned}\deg_{\mathbf{k}}(P) \quad &+ \quad I\left((-\alpha_F)^+ + \sum_{\substack{p=1\\ \mathbf{b}_p \neq \mathbf{0}}}^{pp} a_p^+ + \sum_{\substack{q=1\\ \mathbf{v}_q \neq \mathbf{0}}}^{qq} (-u_q)^+\right)\\ &+ \quad \mathbf{J}\cdot\left((-\boldsymbol{\beta}_F)^+ + \sum_{p=1}^{pp} \mathbf{b}_q^+ + \sum_{q=1}^{qq} (-\mathbf{v}_q)^+\right),\end{aligned}$$

wobei $\mathbf{x}^+ := (\max(0,x_1), \ldots, \max(0,x_m))$ *ist.*

Beweis Für rechteckige Strukturmengen lassen sich die Randpunkte einfach ausrechnen.

- Für die Zähler-Randpunkte $(\overline{i}_p, \overline{\mathbf{j}}_p)$ gilt

  $$\overline{i}_p = \begin{cases} I & , a_p \geq 0 \\ 0 & , a_p < 0 \end{cases} \qquad \text{und} \qquad \overline{j}_{pm} = \begin{cases} J_m & , b_{pm} \geq 0 \\ 0 & , b_{pm} < 0 \end{cases}$$

  für alle $m \in [1..r]$, also $\overline{i}_p a_p = I a_p^+$ und $\overline{\mathbf{j}}_p \cdot \mathbf{b}_p = \mathbf{J} \cdot \mathbf{b}_p^+$.

- Für die Nenner-Randpunkte $(\underline{i}_q, \underline{\mathbf{j}}_q)$ gilt

  $$\underline{i}_q = \begin{cases} 0 & , u_q > 0 \\ I & , u_q \leq 0 \end{cases} \qquad \text{und} \qquad \underline{j}_{qm} = \begin{cases} 0 & , v_{qm} > 0 \\ J_m & , v_{qm} \leq 0 \end{cases}$$

  für alle $m \in [1..r]$, also $-\underline{i}_q u_q = I(-u_q)^+$ und $-\underline{\mathbf{j}}_q \cdot \mathbf{v}_q = \mathbf{J} \cdot (-\mathbf{v}_q)^+$.

- Für die Differenz-Randpunkte $(i^-, \mathbf{j}^-)$ gilt

  $$i^- = \begin{cases} 0 & , \alpha_F > 0 \\ I & , \alpha_F \leq 0 \end{cases} \qquad \text{und} \qquad j_m^- = \begin{cases} 0 & , \beta_{Fm} > 0 \\ J_m & , \beta_{Fm} \leq 0 \end{cases}$$

  für alle $m \in [1..r]$, also $-i^- \alpha_F = I(-\alpha_F)^+$ und $-\mathbf{j}^- \cdot \boldsymbol{\beta}_F = \mathbf{J} \cdot (-\boldsymbol{\beta}_F)^+$.

Die Behauptung folgt nun mit Satz 2.5. □

Formulieren wir nun

Theorem 2.8 (Existenzsatz) *Sei* F *ein zulässiger hypergeometrischer Term in* n *und* $\mathbf{k}$*. Dann existiert eine* $\mathbf{k}$*-freie Rekursionsgleichung für* F*, deren Ordnung in* n *höchstens*

$$\left\lceil \frac{1}{r!}\left(\mathbf{1}\cdot\left((-\boldsymbol{\beta}_F)^+ + \sum_{p=1}^{pp} \mathbf{b}_p^+ + \sum_{q=1}^{qq} (-\mathbf{v}_q)^+\right)\right)^r \right\rceil \tag{2.17}$$

beträgt.

Beweis Wir betrachten rechteckige Strukturmengen und zeigen, dass es $I \in \mathbb{N}_0$ und $\mathbf{J} \in \mathbb{N}_0^r$ gibt, für die $S_{I\mathbf{J}}$ eine Struktur für F ist. O.B.d.A. nehmen wir an, dass $\mathbf{J} = (J, J, \ldots, J)$ gilt. Dann folgt mit Lemma 2.7, dass $\deg_{\mathbf{k}}(P^F_{S_{I\mathbf{J}}})$ dem Ausdruck

$$\overbrace{\deg_{\mathbf{k}}(P)}^{\varepsilon:=} + I\overbrace{\left((-\alpha_F)^+ + \sum_{\substack{p=1\\ \mathbf{b}_p\neq\mathbf{0}}}^{pp} a_p^+ + \sum_{\substack{q=1\\ \mathbf{v}_q\neq\mathbf{0}}}^{qq}(-u_q)^+\right)}^{\gamma:=}$$
$$+ J\overbrace{\left(\mathbf{1}\cdot\left((-\boldsymbol{\beta}_F)^+ + \sum_{p=1}^{pp}\mathbf{b}_q^+ + \sum_{q=1}^{qq}(-\mathbf{v}_q)^+\right)\right)}^{\delta:=}$$

entspricht. Betrachten wir die folgenden Ausdrücke als Polynome in J, so erhalten wir

$$\begin{aligned}\binom{\deg_{\mathbf{k}}(P^F_{S_{I\mathbf{J}}})+r}{r} &= \binom{\gamma I+\delta J+\varepsilon+r}{r}\\ &= \frac{(\gamma I+\delta J+\varepsilon+r)\cdots(\gamma I+\delta J+\varepsilon+1)}{r!}\\ &= \frac{\delta^r}{r!}J^r+\ldots\end{aligned}$$

und

$$|S_{I\mathbf{J}}| = (I+1)(J+1)^r = (I+1)\,J^r+\ldots.$$

Setzen wir $I = \lceil\frac{\delta^r}{r!}\rceil$, dann folgt daraus für hinreichend große J

$$\binom{\deg_{\mathbf{k}}(P^F_{S_{I\mathbf{J}}})+r}{r} = \frac{\delta^r}{r!}J^r+\ldots \quad < \quad \left(\left\lceil\frac{\delta^r}{r!}\right\rceil+1\right)J^r+\ldots = |S_{I\mathbf{J}}|$$

und mit Theorem 2.6 die Behauptung. □

Die obere Schranke (2.17) kann man mit der Prozedur `max_order` ermitteln. Mit den neuen Begriffen können wir nun Algorithmus 2.1 modifizieren und erhalten

Algorithmus 2.2 (Fasenmyer-Algorithmus mit $S_{I\mathbf{J}}$)

1. **Eingabe**: ein zulässiger Term $F(n,\mathbf{k})$ mit $\mathbf{k} = (k_1,\ldots,k_r)$
   eine Strukturmenge $S_{I\mathbf{J}}$ für F.

2. Mache den Ansatz

$$\sum_{(i,\mathbf{j})\in S_{I\mathbf{J}}} a_{i\mathbf{j}}(n)F(n-i,\mathbf{k}-\mathbf{j}) = 0, \tag{2.18}$$

   wobei $a_{i\mathbf{j}}$ rationale Funktionen in n sind.

3. Bestimme das assoziierte Polynom $P^F_{S_{I\mathbf{J}}}$ gemäß (2.13). Es gilt dann

$$P^F_{S_{I\mathbf{J}}}(n,\mathbf{k}) = 0.$$

4. Betrachte $P^F_{S_{I\mathbf{J}}}$ als Polynom in $\mathbf{k}$ und vergleiche die Koeffizienten der $k_1^{l_1} \cdots k_r^{l_r}$ mit 0, um ein homogenes lineares Gleichungssystem für die $a_{i\mathbf{j}}$'s über dem Körper der rationalen Funktionen in n zu erhalten.

5. Wenn nur die triviale Lösung $a_{i\mathbf{j}} \equiv 0$ für alle $(i, \mathbf{j}) \in S_{I\mathbf{J}}$ existiert, dann gibt es keine $\mathbf{k}$-freie Rekursionsgleichung für F bzgl. $S_{I\mathbf{J}}$.
   Vergrößere $S_{I\mathbf{J}}$ und wiederhole Schritte 2-5.
   Andernfalls setze die erhaltene Lösung in (2.18) ein und multipliziere mit einem gemeinsamen Nenner.
   Ausgabe: die durch den letzten Schritt entstandene, i. allg. nicht eindeutige $\mathbf{k}$-freie Rekursion für F bzgl. der Struktur $S_{I\mathbf{J}}$.

Dieser Algorithmus ist ebenfalls in der Maple-Prozedur `find_krec` implementiert. Wir können nun, ohne eine Strukturmenge anzugeben, `find_krec` nach einer $\mathbf{k}$-freien Rekursionsgleichung suchen lassen.

```
> find_krec(binomial(n,j)*binomial(j,i)*x^i*y^(j-i)*z^(n-j),
> n,[i,j]);

structureset: [1, 0, 0]  equations: 2  variables: 2
structureset: [1, 1, 0]  equations: 5  variables: 4
structureset: [1, 1, 1]  equations: 9  variables: 8
```

$$\{\mathrm{F}(n,\, i,\, j) - z\,\mathrm{F}(n-1,\, i,\, j) - y\,\mathrm{F}(n-1,\, i,\, j-1) - x\,\mathrm{F}(n-1,\, i-1,\, j-1) = 0\}$$

Die Standard-Strukturmengen vergrößert man nach einer einfachen Strategie, die im nächsten Kapitel beschrieben wird. Der Existenzsatz zeigt zwar, dass es zu jedem zulässigen Term eine $\mathbf{k}$-freie Rekursionsgleichung gibt[7], allerdings ist es in vielen Fällen nicht möglich, diese zu berechnen. Die Komplexität des Fasenmyer-Algorithmus hängt überwiegend von der Größe des Gleichungssystems ab, so dass man bei großen Gleichungssystemen mit einer nichttrivialen Lösung in vernünftiger Zeit nicht rechnen kann. Tatsächlich kann man im Falle von Doppelsummen nur vereinzelte Beispiele angeben, für die der Fasenmyer-Algorithmus in wenigen Sekunden eine Rekursion liefert. Für unsere Betrachtungen von mehrfachen Summen ist Algorithmus 2.2 demzufolge nicht zufriedenstellend und in der Praxis kaum anwendbar. Wie man den Algorithmus erheblich verbessern kann, werden wir in dem nächsten Kapitel sehen.

[7] Algorithmus 2.2 terminiert also

Kapitel 3

Optimierung und Verallgemeinerung des Fasenmyer-Algorithmus

Dieses Kapitel beschäftigt sich mit der Optimierung des Fasenmyer-Algorithmus. Zunächst führen wir bestimmte Strukturmengen ein, die die Effizienz des Fasenmyer-Algorithmus erheblich steigern. Trotzdem werden wir sehen, dass diese Verbesserung noch nicht ausreicht, um für jeden zulässigen hypergeometrischen Term eine $\mathbf{k}$-freie Rekursionsgleichung berechnen zu können. Deshalb verallgemeinern wir den Fasenmyer-Algorithmus und verlangen nicht mehr, dass die zu bestimmende Rekursion $\mathbf{k}$-frei sein muss. Die nicht notwendig $\mathbf{k}$-freien Rekursionen sind wesentlich schneller zu berechnen und für unsere Zwecke, die Bestimmung von mehrfachen Summen, völlig ausreichend.

3.1 P-maximale Strukturmengen

In diesem Abschnitt werden Strukturmengen vorgestellt, die optimal für den Fasenmyer-Algorithmus sind. Diese Strukturmengen sind von Paul Verbaeten eingeführt worden und um sie definieren zu können, benötigen wir erst einmal

Definition 3.1 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$. Ferner sei $g \in \mathbb{Z}$ und $\mathbf{h} \in \mathbb{Z}^r$ mit $\gcd(g, \mathbf{h}) = 1$. Wir nennen*

$$s_F : \mathbb{R}^{1+r} \to \mathbb{R}, \quad (i, \mathbf{j}) \mapsto gi + \mathbf{h} \cdot \mathbf{j}$$

eine **Strukturfunktion von** *F, wenn mindestens eine der folgenden Voraussetzungen gilt*

- $g = a_p / \gcd(a_p, \mathbf{b}_p)$, $\mathbf{h} = \mathbf{b}_p / \gcd(a_p, \mathbf{b}_p)$ *für ein* $p \in [1..pp]$ *mit* $\mathbf{b}_p \neq \mathbf{0}$
- $g = -u_q / \gcd(u_q, \mathbf{v}_q)$, $\mathbf{h} = -\mathbf{v}_q / \gcd(u_q, \mathbf{v}_q)$ *für ein* $q \in [1..qq]$ *mit* $\mathbf{v}_q \neq \mathbf{0}$
- $g = -\alpha_F / \gcd(\alpha_F, \boldsymbol{\beta}_F)$, $\mathbf{h} = -\boldsymbol{\beta}_F / \gcd(\alpha_F, \boldsymbol{\beta}_F)$ *mit* $\alpha_F \neq 0$ *oder* $\boldsymbol{\beta}_F \neq \mathbf{0}$.

Eine Strukturfunktion s_F **korrespondiert mit** $a_p n + \mathbf{b}_p \cdot \mathbf{k} + c_p$ *(*$u_q n + \mathbf{v}_q \cdot \mathbf{k} + w_q$*,* $(\alpha_F, \boldsymbol{\beta}_F)$*), wenn die erste (zweite, dritte) Bedingung erfüllt ist. Die* **Vielfachheit von** s_F *definieren wir als*

$$\omega_{s_F} := \sum \gcd(a_p, \mathbf{b}_p) + \sum \gcd(u_q, \mathbf{v}_q) + \gcd(\alpha_F, \boldsymbol{\beta}_F),$$

wobei nur diejenigen Summanden auftreten, für die die entsprechenden Ausdrücke mit s_F *korrespondieren. Die Menge aller Strukturfunktionen von* F *bezeichnen wir mit* $\mathcal{S}_F$*.*

Wir geben im Folgenden den Algorithmus an, mit dem man die Strukturfunktionen für einen zulässigen Term F erhält.

Algorithmus 3.1 (Bestimmen von s_F**)**

1. **Eingabe**: ein zulässiger Term $F(n, \mathbf{k})$.
2. Bestimme die Zähler- und Nenner-Fakultätsterme von $F(n, \mathbf{k})$.
3. Bestimme die Fakultäts-Differenz gemäß (2.12).
4. **Ausgabe**: die mit Hilfe von Definition 3.1 bestimmten Strukturfunktionen.

Mit `structure_fct` können wir die Koeffizienten der Strukturfunktionen anzeigen lassen. Die Strukturfunktionen des Binomialkoeffizienten sind z.B. gegeben durch

```
> structure_fct(binomial(n,k),n,k);
```

$$[[-1, 1], [0, -1], [1, 0]]$$

Wir sind jetzt in der Lage, alle Randpunkte, die wir bereits im zweiten Kapitel kennen gelernt haben, zu vereinheitlichen und als Maximum von Strukturfunktionen zu definieren.

Definition 3.2 *Sei* F *ein zulässiger hypergeometrischer Term in* n *und* $\mathbf{k}$*,* S *eine Strukturmenge für* F *und* $s_F \in \mathcal{S}_F$*. Wir nennen einen Punkt* $(i_0, \mathbf{j}_0) \in S$ *einen* **Randpunkt von** s_F **bzgl.** S*, falls*

$$s_F(i_0, \mathbf{j}_0) \geq s_F(i, \mathbf{j}) \quad \textit{für alle } (i, \mathbf{j}) \in S.$$

Wir bezeichnen mit $(i_S^{s_F}, \mathbf{j}_S^{s_F})$ *einen Randpunkt von* s_F *bzgl.* S *und mit*

$$H_S^{s_F} := \{(i, \mathbf{j}) \in \mathbb{R}^{1+r} \mid s_F(i, \mathbf{j}) = s_F(i_S^{s_F}, \mathbf{j}_S^{s_F})\}$$

die **Struktur-Hyperebene von** s_F *und* S*. Für* $r = 1$ *ist eine Struktur-Hyperebene eine* **Strukturlinie**.

Hat man die Strukturfunktionen s_F für F bestimmt, so kann man die Randpunkte von allen s_F bzgl. einer rechteckigen Strukturmenge $S_{I\mathbf{J}}$ mit folgendem einfachen Algorithmus ausrechnen

Algorithmus 3.2 (Bestimmen von $(i^{s_F}_{S_{I\mathbf{J}}}, \mathbf{j}^{s_F}_{S_{I\mathbf{J}}})$)

1. **Eingabe**: ein zulässiger Term $F(n, \mathbf{k})$ mit $\mathbf{k} = (k_1, \ldots, k_r)$
   eine rechteckige Strukturmenge $S_{I\mathbf{J}}$
   eine Strukturfunktion $s_F(i, \mathbf{j}) = gi + \mathbf{h} \cdot \mathbf{j}$ von F.

2. Setze[1]

$$i^{s_F}_{S_{I\mathbf{J}}} = \begin{cases} I & , g \geq 0 \\ 0 & , g < 0 \end{cases} \qquad \text{und} \qquad j^{s_F}_{S_{I\mathbf{J}}m} = \begin{cases} J_m & , h_m \geq 0 \\ 0 & , h_m < 0 \end{cases}$$

   für $m \in [1..r]$.

3. **Ausgabe**: ein Randpunkt $(i^{s_F}_{S_{I\mathbf{J}}}, \mathbf{j}^{s_F}_{S_{I\mathbf{J}}})$.

Dieser Algorithmus ist in `boundpts_Sstd` implementiert und gibt nebst Koeffizienten der Strukturfunktionen, die Randpunkte von s_F aus. Im Falle von $F(n,k) = \binom{n}{k}$ und $S_{3,2}$ sieht das folgendermaßen aus

```
> boundpts_Sstd(binomial(n,k),n,k,[3,2]);
```

$$[[[-1, 1], [0, -1], [1, 0]], [[0, 2], [3, 0], [3, 2]]]$$

Mit Hilfe von den Randpunkten aller s_F's, können wir nun Satz 2.5 vereinfachen.

Satz 3.1 *Sei F ein zulässiger hypergeometrischer Term in n und $\mathbf{k}$ und S eine Strukturmenge für F. Es gilt*

$$\deg_{\mathbf{k}}(P^F_S) = \deg_{\mathbf{k}}(P) + \sum_{s_F \in \mathcal{S}_F} \omega_{s_F} s_F(i^{s_F}_S, \mathbf{j}^{s_F}_S). \tag{3.1}$$

Beweis Sei $s_F \in \mathcal{S}_F$. Dann gelten folgende Gleichungen für Ausdrücke, mit denen s_F korrespondiert

$$\begin{aligned} \bar{i}_p a_p + \bar{\mathbf{j}}_p \cdot \mathbf{b}_p &= \gcd(a_p, \mathbf{b}_p) s_F(\bar{i}_p, \bar{\mathbf{j}}_p) &&= \gcd(a_p, \mathbf{b}_p) s_F(i^{s_F}_S, \mathbf{j}^{s_F}_S) \\ -(\underline{i}_q u_q + \underline{\mathbf{j}}_q \cdot \mathbf{v}_q) &= \gcd(u_q, \mathbf{v}_q) s_F(\underline{i}_q, \underline{\mathbf{j}}_q) &&= \gcd(u_q, \mathbf{v}_q) s_F(i^{s_F}_S, \mathbf{j}^{s_F}_S) \\ -(i^- \alpha_F + \mathbf{j}^- \cdot \boldsymbol{\beta}_F) &= \gcd(\alpha_F, \boldsymbol{\beta}_F) s_F(i^-, \mathbf{j}^-) &&= \gcd(\alpha_F, \boldsymbol{\beta}_F) s_F(i^{s_F}_S, \mathbf{j}^{s_F}_S). \end{aligned}$$

Betrachten wir (2.15) und summieren über alle Strukturfunktionen von F anstatt über alle p und q, so erhalten wir (3.1). □

Ist eine Strukturmenge $S_{I\mathbf{J}}$ von F vorgegeben, so kann man mit obigen Algorithmen die Strukturfunktionen von F und deren Randpunkte bzgl. $S_{I\mathbf{J}}$ berechnen. Wir können dann weitere Punkte, nämlich alle Punkte $(i, \mathbf{j}) \in \mathbb{Z}^{1+r}$, die $s_F(i, \mathbf{j}) \leq s_F(i^{s_F}_{S_{I\mathbf{J}}}, \mathbf{j}^{s_F}_{S_{I\mathbf{J}}})$ für jede Strukturfunktion s_F erfüllen, zu $S_{I\mathbf{J}}$ hinzunehmen, und so die Anzahl der Variablen des Gleichungssystems erhöhen[2]. Der Grad des assoziierten Polynoms und damit die Anzahl der Gleichungen erhöht sich dadurch nicht, denn der Grad hängt nach Satz 3.1 nur von den Randpunkten der Strukturfunktionen ab. Diese Tatsache führt uns zu

[1] vergleiche mit Beweis von Lemma 2.7

[2] gilt natürlich für beliebige Strukturmengen S

Definition 3.3 *Sei F ein zulässiger Term in n und $\mathbf{k}$ und S eine Strukturmenge für F. Dann ist S eine* **P-maximale Strukturmenge**, *wenn es keine Strukturmenge S' mit $S \subsetneq S'$ gibt, für die $\deg_{\mathbf{k}} P_S^F = \deg_{\mathbf{k}} P_{S'}^F$ gilt.*

und schließlich zu

Satz 3.2 *Sei F ein zulässiger Term in n und $\mathbf{k} = (k_1, \ldots, k_r)$ und S eine Strukturmenge für F. S ist P-maximal, wenn*

$$S = \{(i, \mathbf{j}) \in \mathbb{Z}^{1+r} \mid s_F(i, \mathbf{j}) \leq s_F(i_S^{s_F}, \mathbf{j}_S^{s_F}) \text{ für alle } s_F \in \mathcal{S}_F\}$$

ist.

Betrachten wir nun

Beispiel 3.1 Sei

$$F(n,k) = \binom{3n}{k}^2 = \frac{\Gamma(3n+1)^2}{\Gamma(k+1)^2\,\Gamma(3n-k+1)^2}.$$

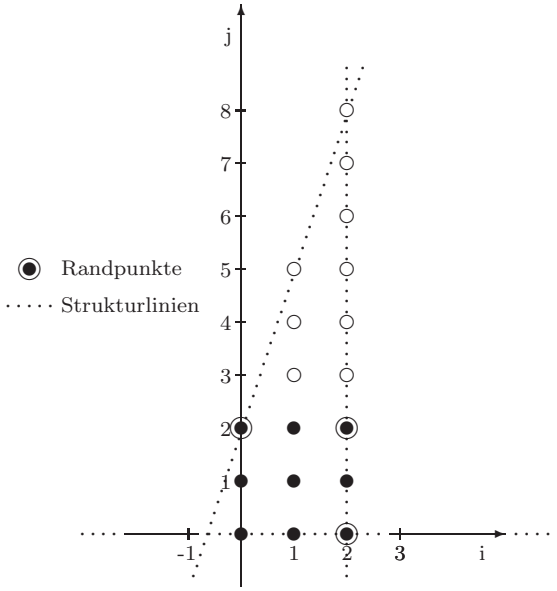
Es ergeben sich mittels Algorithmus 3.1 die Strukturfunktionen

$$s_{F1}(i,j) = -j \qquad s_{F2}(i,j) = -3i + j \qquad s_{F3}(i,j) = i$$

für F. Die ersten beiden Funktionen resultieren aus den Nenner-Fakultätstermen und die letzte aus der Fakultäts-Differenz $(-6, 0)$. Der Zähler-Fakultätsterm liefert keinen Beitrag, da $b_1 = 0$ ist. Sei nun $S_{2,2}$ die quadratische Strukturmenge mit 9 Elementen. Mit Algorithmus 3.2 bekommt man für $S_{2,2}$ die Randpunkte $\{(2,0),(0,2),(2,2)\}$, mit denen sich schließlich auch die Strukturlinien bestimmen lassen. Man erhält

$$\begin{aligned} H_{S_{2,2}}^{s_{F1}} &= \{(i,j) \in \mathbb{R}^2 \mid j = 0\} \\ H_{S_{2,2}}^{s_{F2}} &= \{(i,j) \in \mathbb{R}^2 \mid j = 3i + 2\} \\ H_{S_{2,2}}^{s_{F3}} &= \{(i,j) \in \mathbb{R}^2 \mid i = 2\}. \end{aligned}$$

Die folgende Abbildung verdeutlicht die Situation



Die schwarzen Punkte gehören zu der Strukturmenge $S_{2,2}$ und die weißen Punkte befinden sich in der, von den Strukturlinien eingegrenzten, konvexen Menge. Die Menge der weißen und schwarzen Punkte, nennen wir sie $S_{\max}$, ist demnach die kleinste P-maximale Strukturmenge, die $S_{2,2}$ enthält. Führen wir den Fasenmyer-Algorithmus für $S_{2,j}$ aus, dann ist $j = 8$ die kleinste Zahl, für die $S_{2,j}$ eine Struktur für $F(n,k)$ ist. Wir verwenden wieder die Funktion `find_krec` aus `multsum` und erhalten

```
> duration('find_krec(binomial(3*n,k)^2,n,k,[2,8])')[1];

number of equations: 29  number of variables: 27
                              16.514
```

Es muss also ein 29×27-Gleichungssystem gelöst werden. Maple benötigt dafür ungefähr 16 Sekunden[3]. Betrachten wir die Strukturmenge $S_{2,2}$ und verwenden die Prozedur `Smax`, die ausgehend von einer rechteckigen Strukturmenge $S_{I\mathbf{J}}$, die kleinste P-maximale Strukturmenge bestimmt, die $S_{I\mathbf{J}}$ enthält, so bekommen wir $S_{\max}$.

```
> S_max:=Smax(binomial(3*n,k)^2,n,k,[2,2]):
> duration('find_krec(binomial(3*n,k)^2,n,k,S_max)')[1];

number of equations: 17  number of variables: 18
                               1.493
```

Diesmal ist lediglich ein 17×18-Gleichungssystem zu lösen und Maple benötigt dafür nicht einmal zwei Sekunden. In beiden Fällen kommen wir aber auf die gleiche k-freie Rekursionsgleichung. Sie lautet

$$
\begin{aligned}
&\{2\,(-5+6\,n)\,(2673\,n^4-8910\,n^3+9669\,n^2-3740\,n+400)\,\mathrm{F}(n-1,\,k-3)\\
&+(-5+6\,n)\,(1485\,n^4-4950\,n^3+5337\,n^2-2020\,n+200)\,\mathrm{F}(n-1,\,k-4)\\
&-2\,n\,(3\,n-1)\,(3\,n-2)\,(45\,n^2-120\,n+79)\,\mathrm{F}(n,\,k-1)\\
&-n\,(3\,n-1)\,(3\,n-2)\,(27\,n^2-72\,n+47)\,\mathrm{F}(n,\,k-2)\\
&+(-5+6\,n)\,(81\,n^4-270\,n^3+285\,n^2-100\,n+8)\,\mathrm{F}(n-1,\,k)\\
&-n\,(3\,n-1)\,(3\,n-2)\,(27\,n^2-72\,n+47)\,\mathrm{F}(n,\,k)\\
&+2\,(-5+6\,n)\,(2673\,n^4-8910\,n^3+9669\,n^2-3740\,n+400)\,\mathrm{F}(n-1,\,k-2)\\
&-(3\,n-5)\,(n-1)\,(3\,n-4)\,(27\,n^2-18\,n+2)\,\mathrm{F}(n-2,\,k)\\
&+(-5+6\,n)\,(1485\,n^4-4950\,n^3+5337\,n^2-2020\,n+200)\,\mathrm{F}(n-1,\,k-1)\\
&+4\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(18\,n^2-12\,n+1)\,\mathrm{F}(n-2,\,k-1)\\
&+4\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(27\,n^2-18\,n+4)\,\mathrm{F}(n-2,\,k-2)\\
&+4\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(27\,n^2-18\,n+4)\,\mathrm{F}(n-2,\,k-6)\\
&+4\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(18\,n^2-12\,n+1)\,\mathrm{F}(n-2,\,k-7)\\
&-(3\,n-5)\,(n-1)\,(3\,n-4)\,(27\,n^2-18\,n+2)\,\mathrm{F}(n-2,\,k-8)\\
&-4\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(162\,n^2-108\,n+17)\,\mathrm{F}(n-2,\,k-5)\\
&+(-5+6\,n)\,(81\,n^4-270\,n^3+285\,n^2-100\,n+8)\,\mathrm{F}(n-1,\,k-5)\\
&-4\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(162\,n^2-108\,n+17)\,\mathrm{F}(n-2,\,k-3)\\
&+10\,(3\,n-5)\,(n-1)\,(3\,n-4)\,(99\,n^2-66\,n+10)\,\mathrm{F}(n-2,\,k-4)\;=0\}
\end{aligned}
$$

[3] alle Zeitangaben beziehen sich auf Berechnungen mit einem Computer mit einem 800 MHz schnellen Intel Pentium III-Prozessor und 128 MB Arbeitsspeicher.

Betrachten wir diese k-freie Rekursion genauer, so fällt auf, dass $a_{ij} \not\equiv 0$ für alle $(i,j) \in S_{\max}$ bzw. $a_{ij} \equiv 0$ für alle $(i,j) \in S_{2,8} \setminus S_{\max}$ gilt. Wir werden später sehen, dass dies kein Zufall ist. Verwenden wir die Prozedur `sum_rec`, die eine **k**-freie Rekursionsgleichung bzgl. allen Summationsvariablen summiert, so erhält man

```
> sum_rec(%);
```
$$\{8\,(6\,n-5)\,(6\,n-1)\,(2\,n-1)\,\mathrm{S}(n-1) - n\,(3\,n-1)\,(3\,n-2)\,\mathrm{S}(n) = 0\}$$

und mit dem Anfangswert $S(0) = 1$ die geschlossene Form

$$S(n) = \frac{64^n \left(\frac{1}{6}\right)_n \left(\frac{5}{6}\right)_n \left(\frac{1}{2}\right)_n}{n! \left(\frac{2}{3}\right)_n \left(\frac{1}{3}\right)_n}$$

für $n \in \mathbb{N}_0$. Dieses Beispiel dient nur zur Verdeutlichung, dass P-maximale Strukturmengen besser für die Ermittlung von Rekursionen geeignet sind als Standard-Strukturmengen. Möchte man tatsächlich die Summe $S(n)$ bestimmen, so empfiehlt sich zunächst die einfache Substitution $m = 3n$. Wir erhalten dann leicht die geschlossene Form $\binom{2m}{m}$ für die Summe von $F(m,k)$ und damit schließlich durch Rücksubstitution

$$S(n) = \binom{6n}{3n},$$

was dem obigen Ergebnis gleicht.

Wie konstruiert man nun zu einer gegebenen rechteckigen Strukturmenge $S_{I\mathbf{J}}$ die kleinste P-maximale Strukturmenge, die $S_{I\mathbf{J}}$ enthält? Antwort darauf gibt

Algorithmus 3.3 (P-maximale Strukturmengen)

1. **Eingabe**: ein zulässiger Term $F(n,\mathbf{k})$ mit $\mathbf{k} = (k_1,\ldots,k_r)$ eine Strukturmenge $S_{I\mathbf{J}}$.
2. Bestimme die Strukturfunktionen mit Algorithmus 3.1.
3. Bestimme die Randpunkte von allen $s_F \in \mathcal{S}_F$ bzgl. $S_{I\mathbf{J}}$ mit Algorithmus 3.2.
4. Bestimme die P-maximale Strukturmenge wie folgt
   - Sei $T := \{(i,\mathbf{j}) \in \mathbb{R}^{1+r} \mid s_F(i,\mathbf{j}) \le s_F(i^{s_F}_{S_{I\mathbf{J}}}, \mathbf{j}^{s_F}_{S_{I\mathbf{J}}})$ für alle $s_F \in \mathcal{S}_F\}$. Verwende den Simplex-Algorithmus, um
   $$i_{\max} = \left\lfloor \max_{(i,\mathbf{j})\in T} i \right\rfloor \quad \text{und} \quad j_{\max m} = \left\lfloor \max_{(i,\mathbf{j})\in T} j_m \right\rfloor$$
   sowie
   $$i_{\min} = \left\lceil \min_{(i,\mathbf{j})\in T} i \right\rceil \quad \text{und} \quad j_{\min m} = \left\lceil \min_{(i,\mathbf{j})\in T} j_m \right\rceil$$
   für alle $m \in [1..r]$ zu berechnen. Sei im Folgenden $S := \{(i,\mathbf{j}) \in \mathbb{Z}^{1+r} \mid i \in [i_{\min}..i_{\max}]$ und $\mathbf{j} \in [\mathbf{j}_{\min}..\mathbf{j}_{\max}]\}$. S ist somit die kleinste rechteckige Strukturmenge, die die zu bestimmende P-maximale Strukturmenge enthält.

- alternativ ohne Simplex-Algorithmus:
  Setze $S := \{(i, \mathbf{j}) \in \mathbb{Z}^{1+r} \mid i \in [-ext..ext] \text{ und } \mathbf{j} \in [-\mathbf{ext}..\mathbf{ext}]\}$ für hinreichend großes $ext \in \mathbb{N}$ und $\mathbf{ext} = (ext, \ldots, ext)$.

Setze $S_{\max} := \emptyset$. Überprüfe nun für jeden Punkt $(i, \mathbf{j}) \in S$, ob $s_F(i, \mathbf{j}) \leq s_F(i_{S_{IJ}}^{s_F}, \mathbf{j}_{S_{IJ}}^{s_F})$ für alle $s_F \in \mathcal{S}_F$ gilt. Ist das für einen Punkt $(i, \mathbf{j})$ der Fall, so wird dieser Punkt in die Menge $S_{\max}$ aufgenommen, andernfalls nicht.

5. **Ausgabe**: die P-maximale Strukturmenge $S_{\max}$.

Wenden wir Algorithmus auf den zulässigen Term $F(n, k) = \Gamma(n + k)$ und die Strukturmenge $S_{2,2}$ an, so stellt man fest, dass der Simplex-Algorithmus nicht durchführbar ist. Die von den beiden parallelen Strukturlinien $\{(i, j) \in \mathbb{R}^2 \mid j = -i\}$ und $\{(i, j) \in \mathbb{R}^2 \mid j = -i + 4\}$ eingegrenzte Menge T besitzt weder ein Minimum noch ein Maximum bzgl. i und j, d.h. die Menge $\mathbb{Z}^2 \cap T$ ist keine Strukturmenge mehr. Wir gehen in diesem Fall zu der alternativen Methode über und betrachten nur einen gewissen Ausschnitt, nämlich die Menge $S \cap T$, die wiederum endlich ist. Dabei ist zu beachten, dass der Parameter ext nicht zu groß gewählt wird. Im Falle einer endlichen Strukturmenge ist ext so zu wählen, dass $S_{\max}$ auf jeden Fall in S liegt.
Der Algorithmus 3.3 ist in der bereits erwähnten Funktion `Smax` implementiert. Für den Fall $r = 1$ oder $r = 2$ ist es möglich, sich mit der Prozedur `draw_Smax` einen Überblick über die Gestalt der P-maximalen Strukturmengen zu verschaffen. Die Zeile

```
> draw_Smax(binomial(3*n,k)^2,n,k,[2,2]);
```

liefert bspw. einen Plot, der der Abbildung von Seite 32 ähnelt. Ist $r = 2$, so ist es sinnvoll, in die Grafik hineinzuklicken und bei gedrückter Maustaste durch Bewegung der Maus das Koordinatensystem rotieren zu lassen.

Beispiel 3.2 Sei

$$F(n, i, j) = \binom{n}{j}\binom{j}{i} x^i y^{j-i} z^{n-j}.$$

Die kleinste P-maximale Strukturmenge $S_{\max}$ bzgl. F, die die Menge $S_{3,3,3}$ enthält, ist in Abbildung 3.1 dargestellt. Mit dem Aufruf

```
> F:=binomial(n,j)*binomial(j,i)*x^i*y^(j-i)*z^(n-j):
> draw_Smax(F,n,[i,j],[3,3,3]);
```

erhalten wir einen Plot für $S_{\max}$ mit einem ähnlichen Erscheinungsbild wie in Abbildung 3.1. Es werden allerdings zusätzlich noch die Struktur-Hyperebenen und die Randpunkte gezeichnet.

Betrachtet man eine Struktur S für einen irreduziblen Term F, die keine überflüssigen Punkte beinhaltet, so fällt auf, dass S oftmals einer P-maximalen Strukturmenge gleicht[4]. Allgemein ist in S ein nicht-trivialer Teil von jeder Strukturlinie enthalten, was folgender Satz lehrt

[4] siehe Beispiel 3.1

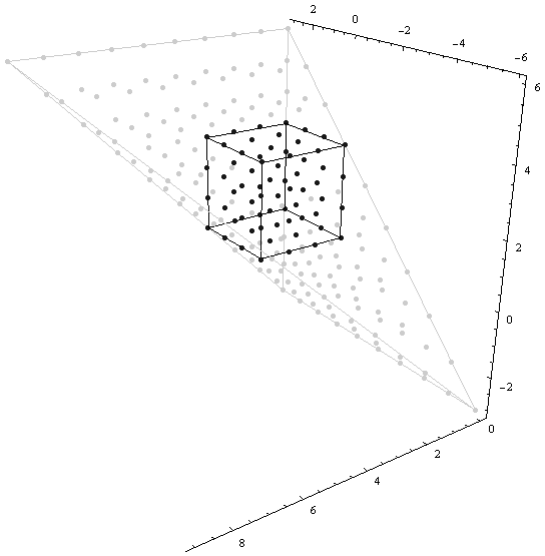


Abbildung 3.1: $S_{\max}$ und $S_{3,3,3}$ von F

Satz 3.3 *Sei F ein irreduzibler, zulässiger Term in n und k und S eine Strukturmenge für F. Es gelte außerdem $P_S^F(n,k) = \sum_{(i,j)\in S} a_{ij}(n)p_{ij}(n,k) = 0$ mit $a_{ij} \not\equiv 0$ für alle $(i,j) \in S$. Dann gilt $\left|S \cap H_S^{s_F}\right| \geq 2$ für eine beliebige Strukturlinie $H_S^{s_F}$.*

Beweis Korrespondiere s_F zu

1. einem Zähler- oder Nenner-Fakultätsterm $s = an+bk+c$ und sei $(i_s, j_s) \in H_S^{s_F}$ ein entsprechender Randpunkt. Da F irreduzibel ist, gilt für $s - ai_s - bj_s$ wegen (2.13)

$$\begin{aligned} s - ai_s - bj_s &\not| \, p_{ij} \qquad \forall\, (i,j) \in S \cap H_S^{s_F} \\ s - ai_s - bj_s &\mid p_{ij} \qquad \forall\, (i,j) \in S \setminus H_S^{s_F}. \end{aligned}$$

   Nach Definition 3.1 ist $b \neq 0$, so dass wir $k = -\frac{a}{b}(n - i_s) - \frac{c}{b} + j_s$ in $P_S^F(n,k) = 0$ einsetzen können. Es ergibt sich $s - ai_s - bj_s = 0$ und mit obigen Teilerbeziehungen

$$\sum_{(i,j)\in S\cap H_S^{s_F}} a_{ij}(n)p_{ij}(n, -\tfrac{a}{b}(n - i_s) - \tfrac{c}{b} + j_s) = 0. \tag{3.2}$$

   Folglich müssen mindestens zwei Elemente in $S \cap H_S^{s_F}$ existieren, da alle auftretenden Summanden nichtverschwindende Polynome sind.

2. der Fakultäts-Differenz. Anhand des Beweises von Satz 2.5 kann man erkennen, dass $\deg_k(p_{i'j'}) < \deg_k(p_{ij})$ für alle $(i',j') \in S \setminus H_S^{s_F}$ und $(i,j) \in S \cap H_S^{s_F}$ gilt. Aus diesem Grund vergleichen wir den höchsten Koeffizient des assoziierten Polynoms mit 0 und gelangen auf diese Weise auf die Gleichung

$$\sum_{(i,j)\in S\cap H_S^{s_F}} q_{ij}(n)a_{ij}(n) = 0.$$

Da nicht alle Polynome $q_{ij}(n)$ gleich 0 sind, gibt es wiederum mindestens zwei Elemente in $S \cap H_S^{s_F}$.

□

Wir können nun den Fasenmyer-Algorithmus auf P-maximale Strukturmengen $S_{\max}$ anwenden und erhalten somit

Algorithmus 3.4 (Fasenmyer-Algorithmus mit $S_{\max}$)

1. **Eingabe**: ein zulässiger Term $F(n, \mathbf{k})$ mit $\mathbf{k} = (k_1, \ldots, k_r)$
   eine Strukturmenge $S_{I\mathbf{J}}$ für F.

2. Bestimme mit Algorithmus 3.3 die kleinste P-maximale Strukturmenge $S_{\max}$, die $S_{I\mathbf{J}}$ enthält. Mache den Ansatz

$$\sum_{(i,\mathbf{j}) \in S_{\max}} a_{i\mathbf{j}}(n) F(n-i, \mathbf{k}-\mathbf{j}) = 0, \tag{3.3}$$

   wobei $a_{i\mathbf{j}}$ rationale Funktionen in n sind.

3. Bestimme das assoziierte Polynom $P^F_{S_{\max}}$ gemäß (2.13). Es gilt dann

$$P^F_{S_{\max}}(n, \mathbf{k}) = 0.$$

4. Betrachte $P^F_{S_{\max}}$ als Polynom in $\mathbf{k}$ und vergleiche die Koeffizienten der $k_1^{l_1} \cdots k_r^{l_r}$ mit 0, um ein homogenes lineares Gleichungssystem für die $a_{i\mathbf{j}}$'s über dem Körper der rationalen Funktionen in n zu erhalten.

5. Wenn nur die triviale Lösung $a_{i\mathbf{j}} \equiv 0$ für alle $(i, \mathbf{j}) \in S_{\max}$ existiert, dann gibt es keine $\mathbf{k}$-freie Rekursionsgleichung für F bzgl. $S_{\max}$.
   Vergrößere $S_{I\mathbf{J}}$ und wiederhole Schritte 2-5.
   Andernfalls setze die erhaltene Lösung in (3.3) ein und multipliziere mit einem gemeinsamen Nenner.
   Ausgabe: die durch den letzten Schritt entstandene, i. allg. nicht eindeutige $\mathbf{k}$-freie Rekursion für F bzgl. der Struktur $S_{\max}$.

Es stellt sich die Frage, wie man bei diesem Algorithmus die Strukturmengen wählt und vergrößert. Eine einfache Strategie wäre, ausgehend von der Strukturmenge $S_{1,\mathbf{0}}$, sukzessive die Komponenten des Nullvektors um eins zu erhöhen. Man erhält schließlich $S_{1,\mathbf{1}}$ und inkrementiert erneut Schritt für Schritt die Komponenten des Vektors $(1, \mathbf{1})$, dann die des Vektors $(2, \mathbf{2})$, u.s.w.. Dieses Verfahren, welches Wegschaider in seiner Mathematica-Implementation verwendet, ist allerdings sehr grob und berücksichtigt nicht die Gestalt der entsprechenden P-maximalen Strukturmengen. In vielen Fällen ist es sinnvoller, die endliche Menge

$$M_K = \{S_{\max} \text{ von } S_{I\mathbf{J}} \mid (I, \mathbf{J}) \in S_{K\mathbf{K}}\}$$

mit einer oberen Grenze $K \in \mathbb{N}$ und $\mathbf{K} = (K, K, \ldots, K)$ zu betrachten und aufsteigend nach der Größe der $S_{\max}$ zu sortieren. Gibt es keine Menge $S_{\max}$

aus M_K, die eine Struktur für F ist, so muss man K größer wählen. Der Vorteil dieser Strategie gegenüber der ersten ist, dass keine P-maximale Strukturmenge übersprungen und damit womöglich eine Lösung eines kleineren Gleichungssystems vernachlässigt wird. Es ist zu bemerken, dass es vorkommt, dass einige Strukturmengen aus M_K die gleiche Größe besitzen. Greifen wir das Beispiel 3.1 auf, so enthalten die P-maximalen Strukturmengen von $S_{1,5}$ bzw. $S_{0,8}$ genau soviele Elemente wie $S_{\max}$ von $S_{2,2}$. Diese Mengen unterscheiden sich von $S_{\max}$ von $S_{2,2}$ nur durch eine Verschiebung in i-Richtung um 1 bzw. 2[5] und liefern somit auch keine neuen Informationen. Deswegen betrachten wir unter allen gleich großen Strukturmengen aus M_K jeweils nur eine und fassen diese in der Menge $\overline{M}_K$ zusammen. Zwei verschiedene, gleich große P-maximale Strukturmengen S und S' besitzen sehr oft dieselbe Gestalt[6], d.h. es gilt $S = S' + \mathbf{l}$ für einen festen Vektor $\mathbf{l} = (l_0, l_1, \ldots, l_r) \in \mathbb{Z}^{1+r}$, da eine Änderung der zugrunde liegenden rechteckigen Strukturmengen einer Verschiebung der Strukturhyperebenen gleicht. Die Menge $\overline{M}_K$ lässt sich mit der Funktion `select_Smax` bestimmen. Den Algorithmus, auf den `select_Smax` beruht, fassen wir nun noch einmal zusammen.

Algorithmus 3.5 (Bestimmen von $\overline{M}_K$)

1. **Eingabe**: ein zulässiger Term $F(n, \mathbf{k})$ mit $\mathbf{k} = (k_1, \ldots, k_r)$ eine obere Grenze K.

2. Bestimme zunächst für alle $(I, J_1, \ldots, J_r)$ aus $S_{K\mathbf{K}}$, mittels Algorithmus 3.3, die kleinste P-maximale Strukturmenge, die $S_{I\mathbf{J}}$ enthält und fasse diese Mengen in der Menge M_K zusammen.

3. Sortiere die Mengen in M_K nach der Größe.

4. Bestimme $\overline{M}_K$ aus der sortierten Menge M_K wie folgt

$$
\begin{array}{l}
\overline{M}_K = \{M_{K1}\} \\
\texttt{for } i = 1 \texttt{ to } |M_K| - 1 \texttt{ do} \\
\quad \texttt{if } |M_{Ki+1}| \neq |M_{Ki}| \\
\quad\quad \texttt{then } \overline{M}_K = \overline{M}_K \cup \{M_{Ki+1}\} \\
\quad \texttt{end if} \\
\texttt{end for}
\end{array}
$$

5. **Ausgabe**: Die Menge $\overline{M}_K$.

In der Prozedur `find_rec` ist u.a. auch der Algorithmus 3.4 implementiert, wobei man mit dem vierten Parameter steuern kann, welche Strategie man verwenden möchte (0 entspricht der ersten Strategie und $K > 0$ entspricht der zweiten Strategie mit $\overline{M}_K$). Wir vergleichen beide Vorgehensweisen in dem

[5] siehe Abbildung auf Seite 32

[6] aber nicht immer: Betrachte $S_{\max}$ von $S_{1,1,0}$ und $S_{\max}$ von $S_{0,0,2}$ im Beispiel 3.3

Beispiel 3.3 Sei

$$F(n,i,j) = \binom{n+i}{j}\binom{j}{i}^3.$$

Die erste Strategie liefert

```
> F:=binomial(n+i,j)*binomial(j,i)^3:
> duration('order_rec(find_rec(F,n,[i,j],0,0,0))');

structureset: [1, 0, 0]  equations: 3    variables: 2
structureset: [1, 1, 0]  equations: 21   variables: 10
structureset: [1, 1, 1]  equations: 40   variables: 22
structureset: [2, 1, 1]  equations: 50   variables: 28
structureset: [2, 2, 1]  equations: 100  variables: 60

                        [93.103, [5, 6, 7]]
```

und somit (i,j)-freie Rekursionsgleichungen der Ordnung 5, 6 und 7 in ca. 93 Sekunden. Dabei ist ein 100×60-Gleichungssystem zu lösen. Anders bei der zweiten Strategie mit $\overline{M}_2$

```
> F:=binomial(n+i,j)*binomial(j,i)^3:
> duration('order_rec(find_rec(F,n,[i,j],2,0,0))');

structureset: [1, 0, 0]  equations: 3    variables: 2
structureset: [2, 0, 0]  equations: 6    variables: 3
structureset: [0, 0, 1]  equations: 5    variables: 4
structureset: [0, 1, 0]  equations: 15   variables: 7
structureset: [0, 0, 2]  equations: 23   variables: 10
structureset: [2, 1, 0]  equations: 28   variables: 13
structureset: [0, 1, 1]  equations: 31   variables: 16
structureset: [0, 2, 0]  equations: 45   variables: 22
structureset: [1, 2, 0]  equations: 55   variables: 28
structureset: [0, 1, 2]  equations: 61   variables: 30
structureset: [2, 2, 0]  equations: 66   variables: 34
structureset: [0, 2, 1]  equations: 73   variables: 40

                        [21.612, [5]]
```

Diese Variante benötigt nur ungefähr 21 Sekunden und liefert eine Rekursion der Ordnung 5. Das letztlich zu lösende 73×40-Gleichungssystem ist wesentlich kleiner als das 100×60-Gleichungssystem bei der groben Vorgehensweise. Dafür müssen bei der feineren Methode mehr Gleichungssysteme betrachtet werden.

Die Prozedur `ani_Smax` veranschaulicht die zweite Vorgehensweise in einer Animation. Bspw. ruft man `ani_Smax` für die Menge $\overline{M}_7$ der Summanden der Legendre-Polynome

$$P_n(x) = \sum_{k=0}^{n} \binom{n}{k}\binom{-n-1}{k}\left(\frac{1-x}{2}\right)^k$$

bzw.

$$P_n(x) = \frac{1}{2^n}\sum_{k=0}^{\lfloor n/2\rfloor} (-1)^k \binom{n}{k}\binom{2n-2k}{n} x^{n-2k}$$

mit der Befehlszeile

```
> ani_Smax(binomial(n,k)*binomial(-n-1,k)*((1-x)/2)^k,n,k,7);
```

bzw.

```
> ani_Smax((-1)^k*binomial(n,k)*binomial(2*n-2*k,n)*x^(n-2*k),
> n,k,7);
```

auf.

Mit Algorithmus 3.4 haben wir unseren effektivsten Algorithmus zur Bestimmung von **k**-freien Rekursionsgleichungen gefunden. Wir können aber viele Beispiele angeben, bei denen selbst dieser Algorithmus versagt, da die auftretenden linearen Gleichungssysteme viel mehr Gleichungen als Unbekannte besitzen und das Verhältnis von Gleichungen zu Unbekannten erst ab einer zu großen Standard-Strukturmenge umschlägt. Betrachte bspw.

```
> F:=binomial(i+j,i)^2*binomial(4*n-2*i-2*j,2*n-2*i):
> find_rec(F,n,[i,j],0,0,0);

structureset: [1, 0, 0]  equations: 9    variables: 2
structureset: [1, 1, 0]  equations: 36   variables: 5
structureset: [1, 1, 1]  equations: 81   variables: 14
structureset: [2, 1, 1]  equations: 141  variables: 20
structureset: [2, 2, 1]  equations: 216  variables: 34
structureset: [2, 2, 2]  equations: 309  variables: 59
Warning, computation interrupted
```

Aus diesem Grund ist es notwendig, sich eine andere Methode zu überlegen, mit der man mehrfache Summen über Terme, die dem obigen Beispiel gleichen, bestimmen kann. Die nächsten Abschnitte befassen sich mit diesem Sachverhalt.

3.2 Verallgemeinerungen des Fasenmyer-Algorithmus

Wir betrachten jetzt Verallgemeinerungen des Fasenmyer-Algorithmus, die bestimmte Rekursionen ermitteln, mit deren Hilfe wir auch wieder Identitäten mit mehrfachen Summen beweisen können. Dabei sind die Rekursionen nicht notwendig **k**-frei und sie lassen sich effizient berechnen. Wir werden sehen, dass sich jede **k**-freie Rekursionsgleichung in eine solche Rekursionsgleichung überführen lässt und die beiden neu gewonnenen Algorithmen, also beide Verallgemeinerungen des Fasenmyer-Algorithmus, angeben.

3.2.1 Zertifikats-Rekursionen

In diesem Abschnitt werden Rekursionsgleichungen betrachtet, die nicht notwendig **k**-frei sind. Wir verwenden die im ersten Kapitel eingeführten Operatoren L_k und $\Delta_k = 1 - L_k$.

Definition 3.4 *Sei $F : D \subseteq \mathbb{C}^{1+r} \to \mathbb{C}$ ein hypergeometrischer Term in n und $\mathbf{k} = (k_1, \ldots, k_r)$. Ferner sei $P_0 \in \mathbb{C}[n]\langle L_n\rangle$ und $P_m \in \mathbb{C}[n, \mathbf{k}]\langle L_n, \mathbf{L_k}\rangle$ für alle $m \in [1..r]$. Dann ist*

$$P_0(n, L_n) + \sum_{m=1}^{r} \Delta_{k_m} P_m(n, \mathbf{k}, L_n, \mathbf{L_k}) \tag{3.4}$$

ein **Zertifikats-Operator**[7]*. Die Gleichung*

$$\Big(P_0(n,L_n)+\sum_{m=1}^{r}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})\Big)F(n,\mathbf{k})=0 \tag{3.5}$$

nennen wir **Zertifikats-Rekursion** *oder* **Zertifikat von** F*. Der* **Hauptteil** *der Zertifikats-Rekursion ist gegeben durch* $P_0(n,L_n)F(n,\mathbf{k})$*. Außerdem bezeichnen wir das Zertifikat als* **trivial***, wenn der Hauptteil des Zertifikats 0 ist, andernfalls ist die Rekursion* **nicht-trivial***.*

Nicht-triviale Zertifikats-Rekursionen für einen hypergeometrischen Term F haben den Vorteil, dass man direkt aus der Rekursion ablesen kann, wie die holonome Rekursionsgleichung für die Summe von F aussieht. Allerdings müssen wir dabei voraussetzen, dass F einen endlichen Träger besitzt. Das ist genau dann der Fall, wenn die Menge

$$T_F(n)=\{\mathbf{k}\mid(n,\mathbf{k})\in D_F\cap(\mathbb{N}_0\times\mathbb{Z}^r)\text{ und }F(n,\mathbf{k})\neq 0\}$$

für alle $n\in\mathbb{N}_0$ endlich ist. Wir formulieren dazu folgenden Satz

Satz 3.4 *Sei* F *ein zulässiger Term in* n *und* $\mathbf{k}$ *mit einem endlichen Träger. Ferner gelte* $\mathbb{N}_0\times\mathbb{Z}^r\subseteq D_F$ *für die Menge der wohldefinierten Werte von* F*. Ist* $P_0(n,L_n)+\sum_{m=1}^{r}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})$ *ein nicht-trivialer Zertifikats-Operator und erfüllt* F *die Rekursionsgleichung*

$$\Big(P_0(n,L_n)+\sum_{m=1}^{r}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})\Big)F(n,\mathbf{k})=0,$$

dann gilt für die Summe $S(n)=\sum_{\mathbf{k}}F(n,\mathbf{k})$ *die holonome Rekursionsgleichung*

$$P_0(n,L_n)S(n)=0.$$

Beweis Es gilt

$$\begin{aligned}
\Big(P_0(n,L_n)+\sum_{m=1}^{r}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})\Big)F(n,\mathbf{k}) &= 0\\
\sum_{\mathbf{k}}\Big(P_0(n,L_n)+\sum_{m=1}^{r}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})\Big)F(n,\mathbf{k}) &= 0\\
\sum_{\mathbf{k}}P_0(n,L_n)F(n,\mathbf{k})+\sum_{m=1}^{r}\sum_{\mathbf{k}}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})F(n,\mathbf{k}) &= 0\\
P_0(n,L_n)\sum_{\mathbf{k}}F(n,\mathbf{k})+\sum_{m=1}^{r}\sum_{\mathbf{k}}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})F(n,\mathbf{k}) &= 0 \qquad (3.6)
\end{aligned}$$

Da F einen endlichen Träger besitzt, gilt für alle $m\in[1..r]$

$$\lim_{k_m\to\infty}F(n,\mathbf{k})=0\quad\text{und}\quad\lim_{k_m\to-\infty}F(n,\mathbf{k})=0$$

[7] sämtliche Operatoren, die aus (3.4) durch eine Indexverschiebung hervorgehen, bezeichnen wir ebenfalls als Zertifikats-Operator.

für alle $n \in \mathbb{N}_0$ und $(k_1, \ldots, k_{m-1}, k_{m+1}, \ldots, k_r) \in \mathbb{Z}^{r-1}$, so dass alle Teleskopsummen $\sum_{\mathbf{k}} \Delta_{k_m} P_m(n, \mathbf{k}, L_n, \mathbf{L_k}) F(n, \mathbf{k})$ verschwinden. Somit folgt die Behauptung mit (3.6). □

Eine Anwendung von Satz 3.4 zeigt

Beispiel 3.4 Sei

$$F(n,i,j) = \binom{r}{i}\binom{s}{j}\binom{t}{n-i-j}$$

mit zusätzlichen Parametern r, s und t. Dann ist

$$\begin{aligned} P(n,i,j,L_n,L_i,L_j) &= (n-r-s-t-1)L_nL_iL_j + nL_iL_j \\ &\quad + \Delta_i\,(iL_j) + \Delta_j\,(jL_i) \end{aligned} \tag{3.7}$$

ein Zertifikats-Operator für F[8]. Der zugrunde liegende Rekursions-Operator ist gegeben durch

$$(n-r-s-t-1)L_nL_iL_j + (n-i-j+2)L_iL_j + jL_i + iL_j. \tag{3.8}$$

Man erhält diesen nicht (i,j)-freien Rekursions-Operator durch „Ausmultiplizieren“ von (3.7). Mit Hilfe von (3.7) ist es einfach, eine Rekursion für die Summe $S(n)$ von F zu erhalten. Da F einen endlichen Träger besitzt, verschwinden nach Satz 3.4 die Delta-Terme bei der Summation durch Teleskopieren und wir bekommen

$$(n-r-s-t-1)S(n-1) + nS(n) = 0.$$

Es folgt durch Induktion mit $S(0) = 1$ und (1.11)

$$S(n) = \frac{(-1)^n}{n!}(-(r+s+t))_n = \binom{r+s+t}{n}.$$

Wir stellen uns nun die Frage, ob man jede $\mathbf{k}$-freie Rekursion in eine nichttriviale Zertifikats-Rekursion umwandeln kann. Eine Antwort liefert

Theorem 3.5 *Sei F ein hypergeometrischer Term in n und $\mathbf{k} = (k_1, \ldots, k_r)$ und $P \in \mathbb{C}[n]\langle L_n, \mathbf{L_k}\rangle$ ein Rekursions-Operator für F. Dann existiert eine nichttriviale Zertifikats-Rekursion für F.*

Beweis Es gelten die Voraussetzungen des Theorems. Außerdem seien o.B.d.A. alle Maxima der Exponenten der Operatoren $L_{k_1}, \ldots, L_{k_r}$ gleich 0. Wir beweisen das Theorem induktiv, indem wir zeigen, dass es für jedes $m \in [1..r+1]$ einen Operator $\tilde{P}_m$ der Form

$$P_0^m(n, L_n, L_{k_m}, \ldots, L_{k_r}) + \sum_{j=1}^{m-1} (1 - L_{k_j}) P_j(n, \mathbf{k}, L_n, L_{k_j}, \ldots, L_{k_r}) \tag{3.9}$$

für F gibt, wobei P_0^m nicht 0 sein soll.

[8] Multiplikation mit $L_i^{-1}L_j^{-1}$ liefert den Zertifikats-Operator der Form (3.4).

- IA: Sei $m = 1$. Dann gilt $\tilde{P}_1 = P_0^1 = P$. Also ist $\tilde{P}_1$ ein nicht-trivialer Rekursions-Operator für F.
- IV: Es gelte für ein $m \in [1..r+1]$, dass $\tilde{P}_m$ ein nicht-trivialer Rekursions-Operator für F ist.
- IS: Wir zeigen, dass $\tilde{P}_{m+1}$, der Operator, den wir durch Division von P_0^m von (3.9) durch $(1 - L_{k_m})$ aus $\tilde{P}_m$ erhalten, ein Rekursions-Operator für F mit nicht-trivialem P_0^{m+1} ist. Sei $S \in \mathbb{C}[n]\langle L_n, L_{k_m}, \ldots, L_{k_r}\rangle$ mit $P_0^m = (1 - L_{k_m})^i S$ und maximalem $i \in \mathbb{N}_0$. Ist $i > 0$, so multiplizieren wir P_0^m mit $(k_m)_i$ und erhalten mit $S' = (1 - L_{k_m})^{i-1} S$

$$\begin{aligned}(k_m)_i(1 - L_{k_m})S' &= ((k_m)_i - (k_m)_i L_{k_m})S' \\ &= ((k_m)_i - L_{k_m}(k_m+1)_i)S' \\ &= ((k_m)_i - (k_m+1)_i + (k_m+1)_i - L_{k_m}(k_m+1)_i)S' \\ &= (-i(k_m+1)_{i-1} + (1 - L_{k_m})(k_m+1)_i)S' \\ &= (-i + (1 - L_{k_m})(k_m+i))(k_m+1)_{i-1}S'. \qquad (3.10)\end{aligned}$$

Ist $i - 1 > 0$, so bekommen wir für $(k_m + 1)_{i-1}S'$ mit $S' = (1 - L_{k_m})S''$ analog zu (3.10)

$$(k_m+1)_{i-1}(1 - L_{k_m})S'' = (-(i-1) + (1 - L_{k_m})(k_m+i))(k_m+2)_{i-2}S''.$$

Fahren wir so fort, erhalten wir schließlich

$$(k_m + i - 1)(1 - L_{k_m})S = (-1 + (1 - L_{k_m})(k_m+i))S$$

und damit durch Rücksubstitution in (3.10)

$$(k_m)_i P_0^m = \Big(\prod_{j=1}^{i}(-j + (1 - L_{k_m})(k_m+i))\Big)S = (-1)^i i!S + (1 - L_{k_m})\tilde{S}$$

für ein geeignetes $\tilde{S} \in \mathbb{C}[n]\langle L_n, L_{k_m}, \ldots, L_{k_r}\rangle$ und für $i \geq 0$. Ferner existiert ein $\tilde{S}' \in \mathbb{C}[n]\langle L_n, L_{k_m}, \ldots, L_{k_r}\rangle$ und ein nicht-triviales $P_0^{m+1} \in \mathbb{C}[n]\langle L_n, L_{k_{m+1}}, \ldots, L_{k_r}\rangle$ mit

$$(-1)^i i!S = (1 - L_{k_m})\tilde{S}' + P_0^{m+1},$$

da $(1 - L_{k_m})$ nicht S teilt. Setzen wir $P_m := \tilde{S} + \tilde{S}'$, so erhalten wir mit Hilfe der letzten beiden Gleichungen und Multiplikation von $\tilde{P}_m$ mit $(k_m)_i$

$$\tilde{P}_{m+1} = (k_m)_i \tilde{P}_m = P_0^{m+1} + \Big(\sum_{j=1}^{m-1}(1 - L_{k_j})(k_m)_i P_j\Big) + (1 - L_{k_m})P_m. \quad (3.11)$$

Für $m = r$ folgt mit der letzten Identität (3.11) die Behauptung.

□

Es folgt

Korollar 3.6 *Für jeden zulässigen hypergeometrischen Term F existiert eine nicht-triviale Zertifikats-Rekursionsgleichung, die F erfüllt.*

Beweis Theorem 2.8 + Theorem 3.5. □

Nicht $\mathbf{k}$-freie Rekursionen kann man nicht immer in eine nicht-triviale Zertifikats-Rekursion transformieren. Haben wir für eine nicht $\mathbf{k}$-freie Rekursion eine entsprechende nicht-triviale Zertifikats-Rekursion gefunden, so können wir wieder die Rekursion für die Summe an dem Hauptteil ablesen. Die Transformation einer nicht notwendig $\mathbf{k}$-freien Rekursion in eine Zertifikats-Rekursion wird folgendermaßen durchgeführt

Algorithmus 3.6 (Zertifikats-Rekursion)

1. **Eingabe**: ein Rekursions-Operator
   $P(n,\mathbf{k},L_n,\mathbf{L_k}) = \sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L_k^j}$ mit $\mathbf{k}=(k_1,\dots,k_r)$.

2. Bestimme $\mathbf{j}_{\max} = (\max_{(i,\mathbf{j})\in S} j_1, \dots, \max_{(i,\mathbf{j})\in S} j_r)$.

3. Bestimme $P_1,\dots,P_r$ wie folgt

   `for` $m=1$ `to` r `do`
   $P_m = 0$
   `for every` $(i,\mathbf{j}) \in S$ `do`
   $\tilde{\mathbf{j}} = \mathbf{j}_{\max} - \mathbf{j}$
   $P_m = P_m + \sum_{l=0}^{\tilde{j}_m - 1} a_{i\mathbf{j}}(n, k_1 - \tilde{j}_1, \dots, k_m - l, \dots, k_r)$
   $L_n^i L_{k_1}^{j_{\max 1}} \cdots L_{k_{m-1}}^{j_{\max m-1}} L_{k_m}^{j_m+l} \cdots L_{k_r}^{j_r}$
   `end for`
   `end for`

4. Bestimme den Hauptteil P_0 wie folgt

   $$P_0 = \sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n,\mathbf{k}+\mathbf{j}-\mathbf{j}_{\max}) L_n^i \mathbf{L_k^{j_{\max}}}$$

5. Ist $P_0 = 0$, dann ist die entsprechende Zertifikats-Rekursion trivial.
   Ausgabe: „Es existiert kein nicht-trivialer Zertifikats-Operator.“
   Andernfalls existiert ein nicht-trivialer Zertifikats-Operator.

   Ausgabe: $P_0 + \sum_{m=1}^{r} \Delta_{k_m} P_m$. Möchte man als Ausgabe den entsprechenden Zertifikats-Operator in Normalform (3.4), so dividiert man den Operator zusätzlich durch $\mathbf{L_k^{j_{\max}}}$.

Beweis Wir betrachten einen Summanden $a_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L_k^j}$ von $P(n,\mathbf{k},L_n,\mathbf{L_k})$. Dieser Summand lässt sich mit $\mathbf{j}_{\max} = (\max_{(i,\mathbf{j})\in S} j_1, \dots, \max_{(i,\mathbf{j})\in S} j_r)$ und $\tilde{\mathbf{j}} = \mathbf{j}_{\max} - \mathbf{j}$ darstellen als

$$\Delta_{k_1} \left(\sum_{l=0}^{\tilde{j}_1 - 1} a_{i\mathbf{j}}(n, k_1 - l, k_2, \dots, k_r) L_n^i L_{k_1}^{j_1+l} \cdots L_{k_r}^{j_r} \right)$$
$$+ a_{i\mathbf{j}}(n, k_1 - \tilde{j}_1, \dots, k_r) L_n^i L_{k_1}^{j_{\max 1}} L_{k_2}^{j_2} \cdots L_{k_r}^{j_r}.$$

Der Operand von Δ_{k_1} ist somit ein Summand von P_1. Teilt man den zweiten Summanden durch Δ_{k_2}, den Rest durch Δ_{k_3} u.s.w., so erhält man nach r-maliger Division

$$\sum_{m=1}^{r} \Delta_{k_m} \left(\sum_{l=0}^{\tilde{j}_m-1} a_{i\mathbf{j}}(n, k_1 - \tilde{j}_1, \ldots, k_m - l, \ldots, k_r) L_n^i L_{k_1}^{j_{\max 1}} \cdots L_{k_m}^{j_m+l} \cdots L_{k_r}^{j_r} \right) + a_{i\tilde{\mathbf{j}}}(n, k_1 - \tilde{j}_1, \ldots, k_r - \tilde{j}_r) L_n^i L_{k_1}^{j_{\max 1}} L_{k_2}^{j_{\max 2}} \cdots L_{k_r}^{j_{\max r}}.$$

Der letzte Summand ist demzufolge ein Summand von dem Hauptteil P_0, was alles zeigt. □

Der Algorithmus ist in der Prozedur `to_cer` implementiert. Betrachten wir nun nochmal Beispiel 3.4, so können wir, unter Verwendung von `to_cer`, den Rekursions-Operator (3.8) in den Zertifikats-Operator (3.7) transformieren.

```
> rec:=(n-r-s-t-1)*F(n-1,i-1,j-1)+(n-i-j+2)*F(n,i-1,j-1)
>       +j*F(n,i-1,j)+i*F(n,i,j-1)=0:
> cer:=to_cer(rec);
```

$$cer := (n-r-s-t-1)\,\mathrm{F}(n-1,\, i-1,\, j-1) + n\,\mathrm{F}(n,\, i-1,\, j-1) + \Delta_i(i\,\mathrm{F}(n,\, i,\, j-1)) + \Delta_j(j\,\mathrm{F}(n,\, i-1,\, j)) = 0$$

```
> shift_cer(cer);
```

$$(n-r-s-t-1)\,\mathrm{F}(n-1,\, i,\, j) + n\,\mathrm{F}(n,\, i,\, j) + \Delta_i((i+1)\,\mathrm{F}(n,\, i+1,\, j)) + \Delta_j((j+1)\,\mathrm{F}(n,\, i,\, j+1)) = 0$$

Die Prozedur `shift_cer` führt die normalisierende Indexverschiebung durch. Die Funktionen `to_rec` und `shift_rec` verwendet man analog. Man beachte, dass man i. allg. durch Vertauschung der Divisionen auch andere Delta-Terme bekommt, wohingegen der Hauptteil immer derselbe bleibt.

3.2.2 1. Verallgemeinerung des Fasenmyer-Algorithmus

Wir kommen nun zu der 1. Verallgemeinerung des Fasenmyer-Algorithmus, bei der nicht ausschließlich $\mathbf{k}$-freie Rekursionen, sondern Zertifikats-Rekursionen bestimmt werden. Dabei möchten wir uns auf nicht-triviale Zertifikate beschränken, da nur diese eine Rekursion für die Summe eines hypergeometrischen Terms liefern. Wir beschäftigen uns zunächst mit der Frage, wann ein Rekursions-Operator einen nicht-trivialen Hauptteil besitzt und halten fest

Satz 3.7 *Sei* $P(n, \mathbf{k}, L_n, \mathbf{L_k}) = \sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n, \mathbf{k}) L_n^i \mathbf{L_k^j}$ *ein Rekursions-Operator,* $S(i) = \{\mathbf{j} \mid (i, \mathbf{j}) \in S\}$ *und* $\mathbf{j}_{\max} = (\max_{(i,\mathbf{j})\in S} j_1, \ldots, \max_{(i,\mathbf{j})\in S} j_r)$. *Falls*

$$\sum_{\mathbf{j}\in S(i)} a_{i\mathbf{j}}(n, \mathbf{k} + \mathbf{j} - \mathbf{j}_{\max}) \in \mathbb{C}[n] \tag{3.12}$$

für alle $i \in \mathbb{Z}$ *und*

$$\sum_{\mathbf{j}\in S(i)} a_{i\mathbf{j}}(n, \mathbf{k} + \mathbf{j} - \mathbf{j}_{\max}) \neq 0 \tag{3.13}$$

für mindestens ein $i \in \mathbb{Z}$ gilt, dann existiert ein nicht-trivialer Zertifikats-Operator mit Hauptteil

$$\sum_i \Big(\sum_{\mathbf{j} \in S(i)} a_{i\mathbf{j}}(n, \mathbf{k} + \mathbf{j} - \mathbf{j}_{\max}) \Big) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}_{\max}}. \tag{3.14}$$

Beweis folgt direkt aus dem Beweis von Algorithmus 3.6. □

Mit Hilfe von Satz 3.7 können wir den bislang effektivsten Algorithmus für die Bestimmung mehrfacher Summen angeben.

Algorithmus 3.7 (1. Verallgemeinerung)

1. **Eingabe**: ein zulässiger Term $F(n, \mathbf{k})$ mit $\mathbf{k} = (k_1, \ldots, k_r)$
   eine P-maximale Strukturmenge $S_{\max}$
   eine obere Grenze $M \in \mathbb{N}$.

2. Mache den Ansatz

$$\sum_{(i,\mathbf{j}) \in S_{\max}} \Big(\sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{M}} a_{i\mathbf{j}\mathbf{l}}(n) \mathbf{k}^{\mathbf{l}} \Big) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}} \tag{3.15}$$

   für den Rekursions-Operator, wobei $a_{i\mathbf{j}\mathbf{l}}$ Polynome in n sind und $\mathbf{M} = (M, M, \ldots, M)$ ist.

3. Bestimme $\mathbf{j}_{\max} = (\max_{(i,\mathbf{j}) \in S_{\max}} j_1, \ldots, \max_{(i,\mathbf{j}) \in S_{\max}} j_r)$ und damit

$$\sum_{\mathbf{j} \in S_{\max}(i)} \sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{M}} a_{i\mathbf{j}\mathbf{l}}(n) (\mathbf{k} + \mathbf{j} - \mathbf{j}_{\max})^{\mathbf{l}} \tag{3.16}$$

   für jedes i, wobei $S_{\max}(i) = \{\mathbf{j} \mid (i, \mathbf{j}) \in S_{\max}\}$ ist. Betrachte diesen Ausdruck als Polynom in $\mathbf{k}$ und vergleiche die Koeffizienten der nicht-trivialen $k_1^{l_1} \cdots k_r^{l_r}$ mit 0, um für jedes i ein homogenes lineares Gleichungssystem für die $a_{i\mathbf{j}\mathbf{l}}$'s über den ganzen Zahlen zu bekommen. Löse die erhaltenen **Reduktions-Systeme**.

4. Setze die Lösungen der Reduktions-Systeme in den Ansatz (3.15) ein. Den so vereinfachten Rekursions-Operator wende man nun auf den zulässigen Term $F(n, \mathbf{k})$ an und dividiere durch $F(n, \mathbf{k})$. Multipliziere, wie gewohnt, mit einem gemeinsamen Nenner, vergleiche die Koeffizienten der $k_1^{l_1} \cdots k_r^{l_r}$ mit 0 und löse das so erhaltene lineare Gleichungssystem.

5. Wenn nur die triviale Lösung existiert, dann gibt es keine Zertifikats-Rekursion für F bzgl. $S_{\max}$ und M.
   Ausgabe: „Es existiert keine Zertifikats-Rekursion für F bzgl. $S_{\max}$ und M.“
   Andernfalls erhalten wir eine Rekursionsgleichung, die man schließlich mit Hilfe von Algorithmus 3.6 in eine Zertifikats-Rekursion umwandelt. Ist dieses Zertifikat trivial, dann gib Folgendes aus

Ausgabe: „Es existiert keine nicht-triviale Zertifikats-Rekursion für F bzgl. $S_{\max}$ und M.“
Ansonsten bekommen wir das gewünschte Ergebnis.
Ausgabe: die erhaltene nicht-triviale Zertifikats-Rekursion.

Beweis Da wir eine Zertifikats-Rekursion bestimmen möchten, müssen wir fordern, dass (3.12) für alle $i \in \mathbb{Z}$ gilt. Diese Tatsache rechtfertigt die Vorgehensweise im dritten Schritt des Algorithmus. Lösen wir die Reduktions-Systeme, so kann man einige $a_{i\mathbf{jl}}$'s als ganzzahlige Linearkombination von den übrigen Unbekannten ausdrücken. Setzt man diese Linearkombinationen in den Ansatz (3.15) ein, so verringert man die Anzahl der Variablen und somit die Größe des zu lösenden Gleichungssystems. Wir haben die Forderung (3.13) nicht in den Algorithmus einfließen lassen, so dass wir als letzten Schritt überprüfen müssen, ob die Rekursionsgleichung in eine nicht-triviale Zertifikats-Rekursion transformiert werden kann oder nicht. □

Diesen Algorithmus wendet man mit den bereits für Algorithmus 3.4 beschriebenen Strategien an. In der Prozedur `find_rec`, die wir schon kennen gelernt haben, ist auch Algorithmus 3.7 implementiert. Wir betrachten jetzt wieder den binomischen Ausdruck von Seite 40, bei dem die Bestimmung (i, j)-freier Rekursionen fehlgeschlagen hat und erhalten diesmal[9]

```
> F:=binomial(i+j,i)^2*binomial(4*n-2*i-2*j,2*n-2*i):
> order_cer(find_rec(F,n,[i,j],1,2));

structureset: [0, 0, 1]  equations: 24  variables: 10
structureset: [1, 0, 1]  equations: 66  variables: 29
structureset: [0, 1, 1]  equations: 63  variables: 48
```

$$[1, 1, 1, 1, 1]$$

In wenigen Sekunden[10] liefert `find_rec` Zertifikats-Rekursionen der Ordnung 1. Bei weiteren Tests mit der ersten Verallgemeinerung sieht man, dass die Ordnungen der Zertifikats-Rekursionen sehr häufig kleiner sind als die Ordnungen entsprechender **k**-freier Rekursionen, sofern diese überhaupt existieren. Leider können wir das Argument, das wir im ersten Kapitel benutzt haben und das sich auf die Gestalt der Gleichungssysteme berief, nicht bei nicht **k**-freien Rekursionen anwenden. Demzufolge können wir keine Aussagen über die Existenz von nicht-trivialen Zertifikaten in Bezug auf Algorithmus 3.7 treffen.

3.2.3 2. Verallgemeinerung des Fasenmyer-Algorithmus

Die zweite Verallgemeinerung des Fasenmyer-Algorithmus liefert leider keine wesentliche Verbesserung gegenüber der ersten Verallgemeinerung. Trotzdem werden wir die Vorgehensweise bei dieser Verallgemeinerung erläutern. Der Unterschied zur ersten Verallgemeinerung ist, dass man bereits den Ansatz (3.15) in eine Form eines Zertifikats transformiert. Die Reduktions-Systeme verkleinern sich dadurch. Allerdings bringen diese Neuerungen keine Effizienzsteige-

[9] der fünfte Parameter von `find_rec` gibt die obere Grenze $M > 0$ an
[10] verwendet man die grobe Strategie, so muss ein 121×94-Gleichungssystem gelöst werden und man erhält eine Rekursion erst nach über einer Stunde!

rung, da die Reduktions-Systeme Gleichungssysteme über den ganzen Zahlen und ohnehin relativ schnell zu lösen sind. Wir führen zunächst den folgenden Begriff ein

Definition 3.5 *Seien* $\mathbf{k} = (k_1, \ldots, k_r)$, $b_{i\mathbf{j}} \in \mathbb{C}[n, \mathbf{k}]$ *und* $c_{mi\mathbf{j}} \in \mathbb{C}[n, \mathbf{k}]$ *für alle* $m \in [1..r]$. *Gilt* $S = S_0 \cup S_1 \cup \ldots \cup S_r$, *dann ist*

$$\sum_{(i,\mathbf{j})\in S_0} b_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}} + \sum_{m=1}^{r} \Delta_{k_m}\Big(\sum_{(i,\mathbf{j})\in S_m} c_{mi\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}}\Big)$$

ein S**-Zertifikats-Operator**.

Man beachte, dass ein S-Zertifikat i. allg. kein Zertifikat ist, aber eine Vorstufe desselbigen. Indem man $\sum_{(i,\mathbf{j})\in S_0} b_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}}$ sukzessive durch Δ_{k_m} dividiert, erhält man ein Zertifikat. Sei nun ein Rekursions-Operator

$$P(n, \mathbf{k}, L_n, \mathbf{L}_{\mathbf{k}}) = \sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}}$$

gegeben. Wir möchten diesen Rekursions-Operator in einen S-Zertifikats-Operator umwandeln und zwar so, dass die Mengen $S_0, S_1, \ldots, S_r$ so klein wie möglich sind. Wir betrachten dazu einen Summanden $a_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}}$. Ist $p_{mi\mathbf{j}} \in \mathbb{N}_0$ die kleinste Zahl, für die $(i, j_1, \ldots, j_m + p_{mi\mathbf{j}} + 1, \ldots, j_r) \notin S$ gilt, so kann man $a_{i\mathbf{j}}(n,\mathbf{k}) L_n^i \mathbf{L}_{\mathbf{k}}^{\mathbf{j}}$ darstellen als

$$\begin{aligned} & a_{i\mathbf{j}}(n, k_1, \ldots, k_m - p_{mi\mathbf{j}}, \ldots, k_r) L_n^i L_{k_1}^{j_1} \cdots L_{k_m}^{j_m + p_{mi\mathbf{j}}} \cdots L_{k_r}^{j_r} \qquad (3.17) \\ + \Delta_{k_m} & \left(\sum_{l=0}^{p_{mi\mathbf{j}}-1} a_{i\mathbf{j}}(n, k_1 \ldots, k_m - l, \ldots, k_r) L_n^i L_{k_1}^{j_1} \cdots L_{k_m}^{j_m + l} \cdots L_{k_r}^{j_r} \right), \end{aligned}$$

ohne dass man die Strukturmenge S vergrößern muss. Das Element $(i, \mathbf{j})$ ist dann ein Element aus S_m. Führt man diese Schritte für jeden Summanden bzgl. einem $m \in [1..r]$ durch, so erhält man ein S-Zertifikat mit $S_m \cap S_{m'} = \emptyset$ für alle $m \neq m'$. Da wir willkürlich ein m für jeden Summanden wählen, erhält man bei verschiedenen Wahlen von m auch unterschiedliche S-Zertifikate. Aus diesem Grund betrachten wir

Definition 3.6 *Sei* $S \subseteq \mathbb{Z}^{1+r}$ *eine Strukturmenge und* $m \in [1..r]$. *Wir nennen* $(i, j_1, \ldots, j_m, \ldots, j_r) \in S$ **reduzibel bzgl.** S *und* m, *wenn*

- $(i, j_1, \ldots, j_m + 1, \ldots, j_r) \in S$
- *kein* $l \in [1..r]$ *mit* $l < m$ *und* $(i, j_1, \ldots, j_l + 1, \ldots, j_r) \in S$ *existiert.*

Ist $(i, \mathbf{j}) \in S$ *bzgl. keinem* $m \in [1..r]$ *reduzibel, so bezeichnen wir* $(i, \mathbf{j})$ *als* **irreduzibel bzgl.** S.

Mit Hilfe dieser Definition können wir eine Rekursion in eindeutiger Weise in ein S-Zertifikat transformieren. Ist z.B. $(i, \mathbf{j}) \in S$ reduzibel bzgl. S und m, so gibt es bzgl. $l < m$ keine (3.17) entsprechende (nicht-triviale) Darstellung für

$a_{i\mathbf{j}}(n,\mathbf{k})L_n^i\mathbf{L}_\mathbf{k}^\mathbf{j}$ und wir wandeln den Summanden bzgl. m in den Ausdruck (3.17) um. Seien

$$S_0 = \{(i,\mathbf{j}) \in S \mid (i,\mathbf{j}) \text{ ist irreduzibel bzgl. } S\}$$
$$S_m = \{(i,\mathbf{j}) \in S \mid (i,\mathbf{j}) \text{ ist reduzibel bzgl. } S \text{ und } m\}$$

für alle $m \in [1..r]$, dann ist jedes Element aus S in genau einer der Mengen $S_0, S_1, \ldots, S_r$ zu finden. Folglich können wir den Rekursions-Operator schreiben als

$$\sum_{(i,\mathbf{j})\in S_0} b_{i\mathbf{j}}(n,\mathbf{k})L_n^i\mathbf{L}_\mathbf{k}^\mathbf{j} + \sum_{m=1}^{r} \Delta_{k_m}\Big(\sum_{(i,\mathbf{j})\in S_m} c_{mi\mathbf{j}}(n,\mathbf{k})L_n^i\mathbf{L}_\mathbf{k}^\mathbf{j}\Big) \tag{3.18}$$

mit geeigneten $b_{i\mathbf{j}} \in \mathbb{C}[n,\mathbf{k}]$ und $c_{mi\mathbf{j}} \in \mathbb{C}[n,\mathbf{k}]$ für alle $m \in [1..r]$.
Der folgende Algorithmus zeigt auf, wie man die Strukturmenge S in die Mengen $S_0, S_1, \ldots, S_r$ zerlegen kann

Algorithmus 3.8 (Bestimmen von $S_0, S_1, \ldots, S_r$)

1. **Eingabe**: eine Strukturmenge $S \subseteq \mathbb{Z}^{1+r}$.
2. Bestimme $S_0, S_1, \ldots, S_r$ wie folgt

```
S' = S
for m = 1 to r do
  S_m = ∅
  for every (i,j) ∈ S' do
    if(i, j_1, ..., j_m + 1, ..., j_r) ∈ S
      then S_m = S_m ∪ {(i,j)}
    end if
  end for
  S' = S' \ S_m
end for
S_0 = S'
```

3. **Ausgabe**: $S_0, S_1, \ldots, S_r$.

Dieser Algorithmus ist in der Prozedur `decompose_S` zu finden.

Beispiel 3.5 Wir betrachten den Rekursions-Operator (3.8) aus Beispiel 3.4. Mittels Algorithmus 3.8 erhalten wir die Zerlegung

$$S_0 = \{(0,1,1),(1,1,1)\},\ S_1 = \{(0,0,1)\},\ S_2 = \{(0,1,0)\}.$$

Für die reduziblen Elemente gilt somit

$$iL_j = (i-1)L_iL_j + \Delta_i(iL_j) \quad \text{und} \quad jL_i = (j-1)L_iL_j + \Delta_j(jL_i).$$

Setzt man diese Gleichungen in (3.8) ein, so bekommen wir wieder das Zertifikat (3.7). In diesem Fall stimmen also S-Zertifikat und Zertifikat überein.

Kommen wir nun zur zweiten Verallgemeinerung des Fasenmyer-Algorithmus.

Algorithmus 3.9 (2. Verallgemeinerung)

1. **Eingabe**: ein zulässiger Term $F(n,\mathbf{k})$ mit $\mathbf{k}=(k_1,\ldots,k_r)$
   eine P-maximale Strukturmenge $S_{\max}$
   eine obere Grenze $M\in\mathbb{N}$.

2. Bestimme $S_0,S_1,\ldots,S_r$ bzgl. $S_{\max}$ mit Algorithmus 3.8.

3. Sei $m\in[1..r]$. Mache den Ansatz (3.18) mit

$$b_{i\mathbf{j}}(n,\mathbf{k})=\sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{M}} b_{i\mathbf{jl}}(n)\mathbf{k}^{\mathbf{l}} \quad\text{und}\quad c_{mi\mathbf{j}}(n,\mathbf{k})=\sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{M}} c_{mi\mathbf{jl}}(n)\mathbf{k}^{\mathbf{l}} \tag{3.19}$$

   für den S-Zertifikats-Operator, wobei $b_{i\mathbf{jl}}$ sowie $c_{mi\mathbf{jl}}$ Polynome in n sind und $\mathbf{M}=(M,M,\ldots,M)$ ist.

4. Bestimme $\mathbf{j}_{\max}=(\max_{(i,\mathbf{j})\in S_0} j_1,\ldots,\max_{(i,\mathbf{j})\in S_0} j_r)$ und damit

$$\sum_{\mathbf{j}\in S_0(i)}\sum_{\mathbf{l}=\mathbf{0}}^{\mathbf{M}} b_{i\mathbf{jl}}(n)(\mathbf{k}+\mathbf{j}-\mathbf{j}_{\max})^{\mathbf{l}} \tag{3.20}$$

   für jedes i, wobei $S_0(i)=\{\mathbf{j}\mid(i,\mathbf{j})\in S_0\}$ ist. Betrachte diesen Ausdruck als Polynom in $\mathbf{k}$ und vergleiche die Koeffizienten der nicht-trivialen $k_1^{l_1}\cdots k_r^{l_r}$ mit 0, um für jedes i ein homogenes lineares Gleichungssystem für die $b_{i\mathbf{jl}}$'s über den ganzen Zahlen zu bekommen. Löse die erhaltenen Reduktions-Systeme.

5. Setze die Lösungen der Reduktions-Systeme in den Ansatz (3.18) ein. Den so vereinfachten S-Zertifikats-Operator wende man nun auf den zulässigen Term $F(n,\mathbf{k})$ an und dividiere durch $F(n,\mathbf{k})$. Multipliziere, wie gewohnt, mit einem gemeinsamen Nenner, vergleiche die Koeffizienten der $k_1^{l_1}\cdots k_r^{l_r}$ mit 0 und löse das so erhaltene lineare Gleichungssystem.

6. Wenn nur die triviale Lösung existiert, dann gibt es kein S-Zertifikat und somit keine Zertifikats-Rekursion für F bzgl. $S_{\max}$ und M.
   Ausgabe: „Es existiert keine Zertifikats-Rekursion für F bzgl. $S_{\max}$ und M."
   Andernfalls erhalten wir ein S-Zertifikat. Mit Hilfe von Algorithmus 3.6 wandeln wir $\sum_{(i,\mathbf{j})\in S_0} b_{i\mathbf{j}}(n,\mathbf{k})L_n^i\mathbf{L}_{\mathbf{k}}^{\mathbf{j}}$ in eine Zertifikats-Rekursion um. Ist dieses Zertifikat trivial, dann gib Folgendes aus
   Ausgabe: „Es existiert keine nicht-triviale Zertifikats-Rekursion für F bzgl. $S_{\max}$ und M."
   Ansonsten bekommen wir das gewünschte Ergebnis.
   Ausgabe: die erhaltene nicht-triviale Zertifikats-Rekursion.

Die Maple-Prozedur `find_cer` beinhaltet Algorithmus 3.9.

Beispiel 3.6 Wir betrachten den zulässigen Term von Beispiel 3.1 und verwenden diesmal die Funktion `find_cer`.

```
> find_cer(binomial(3*n,k)^2,n,k,1,1);

structureset: [0, 1]  equations: 3  variables: 3
structureset: [1, 0]  equations: 7  variables: 8
```

$$\{24\,(-5+6\,n)\,(6\,n-1)\,(2\,n-1)\,\mathrm{F}(n-1,\,k-3)-3\,(3\,n-2)\,(3\,n-1)\,n\,\mathrm{F}(n,\,k-3)$$
$$+\,\Delta_k(-3\,(3\,n-2)\,(3\,n-1)\,n\,\mathrm{F}(n,\,k-1)-3\,(3\,n-2)\,(3\,n-1)\,n\,\mathrm{F}(n,\,k-2)$$
$$+(6\,n-27\,n^2+27\,n^3-36\,n\,k+54\,n^2\,k+4\,k)\,\mathrm{F}(n-1,\,k)$$
$$+(468\,n-1026\,n^2+594\,n^3-120\,n\,k+180\,n^2\,k+16\,k-52)\,\mathrm{F}(n-1,\,k-1)$$
$$+(1539\,n^3-2403\,n^2+1050\,n-116-36\,n\,k+54\,n^2\,k+4\,k)\,\mathrm{F}(n-1,\,k-2)$$
$$-3\,(3\,n-2)\,(3\,n-1)\,n\,\mathrm{F}(n,\,k))=0\}$$

Es ergibt sich eine Zertifikats-Rekursion der Ordnung 1. Zur Erinnerung: Die k-freie Rekursion war eine Rekursion zweiter Ordnung.

3.3 Automatisches Filtern

Axel Riese beschreibt in [Rie01], wie man den Zeilberger-Algorithmus mit Hilfe der Methode des automatischen Filterns beschleunigen kann. Wir werden diese Methode nun auf den Fasenmyer-Algorithmus anwenden. Sei dazu $F(n,\mathbf{k})$ ein zulässiger Term, der mindestens einen $\mathbf{k}$-freien Faktor besitzt. Aufgrund des Faktors $\Gamma(n+1)$ ist z.B. der Binomialkoeffizient $\binom{n}{k}$ ein solcher Term. Machen wir den Ansatz

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n)F(n-i,\mathbf{k}-\mathbf{j})=0 \tag{3.21}$$

für eine Rekursion von F, so erhalten wir nach Division durch $F(n,\mathbf{k})$

$$\sum_{(i,\mathbf{j})\in S} a_{i\mathbf{j}}(n)R_{i\mathbf{j}}(n,\mathbf{k})Q_{i\mathbf{j}}(n)=0, \tag{3.22}$$

wobei die $R_{i\mathbf{j}}$'s rationale Funktionen in n und $\mathbf{k}$ und die $Q_{i\mathbf{j}}$'s rationale Funktionen in n sein sollen. Anstatt das aus (3.22) entstehende Gleichungssystem zu lösen, betrachten wir das Gleichungssystem, welches aus

$$\sum_{(i,\mathbf{j})\in S} \overline{a}_{i\mathbf{j}}(n)R_{i\mathbf{j}}(n,\mathbf{k})=0$$

resultiert. Die Koeffizienten dieses linearen Gleichungssystems sind kleiner als bei (3.22) und demzufolge findet Maple auch schneller eine Lösung. Wir erhalten schließlich eine Rekursion für F durch Einsetzen von $a_{i\mathbf{j}}(n)=\overline{a}_{i\mathbf{j}}(n)/Q_{i\mathbf{j}}(n)$ in den Ansatz (3.21) und Multiplikation mit dem kleinsten gemeinsamen Vielfachen der Nennerpolynome der $a_{i\mathbf{j}}$. Dass diese Vorgehensweise effizienter sein kann, zeigt

Beispiel 3.7 Sei

$$F(n,k)=(-1)^k\binom{2n}{k}^3.$$

Wir verwenden die Prozedur `find_krec` einmal mit Filtern und einmal ohne Filtern. Es ergibt sich

```
> F:=(-1)^k*binomial(2*n,k)^3:
> _EnvFilter:=1:
> duration('find_krec(F,n,k)')[1];
```

```
                                216.902
> _EnvFilter:=0:
> duration('find_krec(F,n,k)')[1];
                               1221.063
```

Die Bestimmung k-freier Rekursionen dauert also ohne automatisches Filtern über 16 Minuten länger[11]. Mit Hilfe der erhaltenen Rekursionsgleichung können wir im Übrigen auf einfache Art und Weise die Identität von Dixon

$$\sum_k (-1)^k \binom{2n}{k}^3 = (-1)^n \frac{(3n)!}{n!^3}$$

beweisen.

Filtert man bei den Verallgemeinerungen des Fasenmyer-Algorithmus, so konstatiert man bei zulässigen Termen, deren **k**-freier Anteil groß ist, ebenfalls eine erhebliche Beschleunigung

Beispiel 3.8 Sei

$$F(n,i,j) = \frac{(i-j)^2}{n} \binom{n}{i}^3 \binom{n}{j}^2.$$

Anwendung von `find_rec` bzgl. der kleinsten P-maximalen Strukturmenge, die die Menge $S_{1,2,2}$ enthält, liefert

```
> F:=(i-j)^2/n*binomial(n,i)^3*binomial(n,j)^2:
> _EnvFilter:=1:
> duration('find_rec(F,n,[i,j],[1,2,2],1)')[1];
                                962.315
> _EnvFilter:=0:
> duration('find_rec(F,n,[i,j],[1,2,2],1)')[1];
                               3004.421
```

Mit automatischem Filtern erzielen wir demnach einen Zeitgewinn von über einer halben Stunde.

[11] verwenden wir P-maximale Strukturmengen, so bekommen wir nach etwa 8 Sekunden bzw. 18 Sekunden eine k-freie Rekursion.

Kapitel 4

Das Maple-Package `multsum`

In diesem Kapitel werden wir die Prozeduren des Maple-Packages `multsum`, die wir bereits an vielen Stellen der letzten zwei Kapitel verwendet haben, detailliert beschreiben. Dazu werden wir den Verwendungszweck, die erwartete Eingabe und die Ausgabe-Struktur der jeweiligen Prozedur aufführen. Das Package `multsum` verwendet einige Funktionen, die in dem Package `hsum` implementiert sind. `hsum` ist ein von Wolfram Koepf geschriebenes Package, das auf dem Buch „Hypergeometric Summation“ ([Koe98]) basiert und viele verschiedene Algorithmen zur hypergeometrischen Summation bereitstellt. U.a. beinhaltet `hsum` den Gosper-Algorithmus, den Zeilberger-Algorithmus und den Petkovšek-Algorithmus. Das Package `multsum` installiert man wie folgt. Die Dateien *hsum6.mpl*, *multsum.mpl* und *multsum.dll* extrahiert man in einen Ordner. Wir nennen diesen Ordner *multsum* und er soll direkt auf dem Laufwerk C: liegen. Man lädt das Package durch folgende Schritte

```
> restart;
> libname:=libname,"C:/multsum":
> read "C:/multsum/hsum6.mpl";
```

Package “Hypergeometric Summation“, Maple V – Maple 8
Copyright 1998 – 2002, Wolfram Koepf, University of Kassel

```
> read "C:/multsum/multsum.mpl";
```

Package “Multiple Summation”, Maple 8
Copyright 2003 – 2004, Torsten Sprenger, University of Kassel

Zunächst ändert man also die Pfadangabe mit der Variablen `libname`, so dass der Pfad auf den Ordner zielt, der obige Dateien enthält. Dabei ist darauf zu achten, dass der neu eingegebene String der letzte String in `libname` ist, falls `libname` eine Folge von Strings darstellt. Alternativ kann man, sofern `libname` auf den *lib*-Ordner von Maple eingestellt ist, die Datei *multsum.dll* in diesen Ordner verschieben und `libname` unverändert lassen. Danach lädt man die beiden Packages `hsum` und `multsum` mit dem `read`-Befehl. Jetzt stehen uns alle Prozeduren der beiden Packages zur Verfügung.
Einige Methoden des Packages sind sowohl in der Maple-internen Programmiersprache als auch in C geschrieben. Die C-Funktionen befinden sich in

der Bibliothek *multsum.dll* und dienen zur effizienteren Bestimmung von P-maximalen Strukturmengen. Zu jeder C-Prozedur existiert eine äquivalente Maple-Prozedur, so dass man sämtliche Berechnungen auch ohne C-Bibliothek durchführen kann.

4.1 Grundlagen

An dieser Stelle werden die Standardeingaben und die Umgebungsvariablen näher erläutert. Da wir meist einen zulässigen Term F in n und $\mathbf{k} = (k_1, \ldots, k_r)$ und/oder eine Strukturmenge S als Eingabe erwarten, werden wir im Folgenden aufführen, wie diese Parameter der Prozeduren auszusehen haben.

- F muss ein zulässiger Term in n und $\mathbf{k}$ sein. Der Ausdruck F sollte aus diesem Grund (multiplikativ) aus folgenden Konstrukten aufgebaut sein
  - `binomial`(), `factorial`() oder !, `GAMMA`(), `pochhammer`() mit (in n und $\mathbf{k}$) ganzzahlig-linearen Argumenten
  - Polynome oder rationale Funktionen in n und $\mathbf{k}$
  - Potenzen, wobei die Basis weder n noch $\mathbf{k}$ enthalten darf und der Exponent ein ganzzahlig-lineares Polynom in n und $\mathbf{k}$ sein kann.

  Ist F nicht hypergeometrisch bzw. zulässig, so wird eine entsprechende Fehlermeldung ausgegeben.
- n ist die Hauptvariable und kann in Maple ein beliebiges Wort bzw. Symbol sein, welches allerdings nicht bereits vergeben sein darf.
- $\mathbf{k}$ ist im Falle von $r > 1$ eine Liste mit Summationsvariablen. Ist $r = 1$, so gibt man für $\mathbf{k}$ lediglich eine Variable ein (ohne Listenklammer).
- S wird eingegeben als
  - Menge von Listen der Länge $r + 1$, falls S eine allgemeine Strukturmenge ist.
  - Liste von Zahlen $I, J_1, \ldots, J_r$, falls $S{=}S_{I\mathbf{J}}$ eine Standard-Strukturmenge mit $\mathbf{J} = (J_1, \ldots, J_r)$ ist.
- F_{var}, S_{var} bzw. a_{var} sollen Variablen für einen hypergeometrischen Term, für eine Summe bzw. für Koeffizienten sein. Es ist sinnvoll, diese optionalen Variablen einzugeben, damit man mit den Ausgaben der entsprechenden Prozeduren weiterarbeiten kann.

Wir beschreiben nun die Umgebungsvariablen von `multsum`.

- `_EnvC`
  Mit Hilfe dieser Umgebungsvariable kann man steuern, ob die C-Funktionen oder die Maple-Funktionen verwendet werden sollen. Der voreingestellte Wert beträgt 1. Bei `_EnvC` $= 1$ läuft die Bestimmung von P-maximalen Strukturmengen über die C-Funktionen. Setzt man `_EnvC` auf 0, so werden ausschließlich die Maple-Prozeduren verwendet. Die Namen der Maple-Prozeduren fangen mit `m_` an, während die Namen der C-Prozeduren, die die entsprechenden Methoden von *multsum.dll* aufrufen, mit `c_` beginnen[1].

[1] man betrachte *multsum.mpl* in einem Editor

- `_EnvP`
  Mit `_EnvP` kann man das Anzeigen zusätzlicher Informationen, wie z.B. die betrachtete Strukturmenge und die Größe der linearen Gleichungssysteme, unterbinden. Dazu ist `_EnvP` auf 0 zu setzen. Die Umgebungsvariable wird mit 1 initialisiert.

- `_EnvSmax`
  Diese Umgebungsvariable ist wichtig zur effektiven Ermittlung von P-maximalen Strukturmengen und entspricht dem Wert der oberen Grenze ext von Algorithmus 3.3. Der Initialisierungswert beträgt 10. Ist `_EnvC` $= 0$, so wird der Simplex-Algorithmus verwendet und `_EnvSmax` wird im Normalfall (endliche Strukturmengen) nicht benötigt. Ist allerdings die Strukturmenge nicht endlich oder `_EnvC` $= 1$, so wird die P-maximale Strukturmenge $S_{\max}$, ausgehend von dem Quadrat bzw. Würfel der Länge 2`_EnvSmax`$+1$, ermittelt, indem man überprüft, welche Elemente des Quadrats bzw. Würfels in $S_{\max}$ liegen müssen. Es ist zu beachten, dass bei einer zu kleinen Wahl von `_EnvSmax` in der Regel nicht mehr alle Elemente der P-maximalen Strukturmenge erfasst werden.

- `_EnvSolve`
  Das Lösen der linearen Gleichungssysteme ist der zeitintensivste Teil des Fasenmyer-Algorithmus, so dass wir eine effiziente Lösungsmethode verwenden müssen. `_EnvSolve` legt deshalb den „Gleichungssystemlöser“ fest. Dabei übergibt man der Umgebungsvariablen lediglich den Namen der Prozedur. Die Ein- und Ausgabe der verwendeten Methode muss dabei mit der Ein- und Ausgabe von `solve` übereinstimmen. Die voreingestellte (und in Bezug auf Maple schnellste) Prozedur ist `SolveTools[Linear]`, kann aber jederzeit durch eine effizientere Prozedur, die bspw. eine entsprechende C-Bibliothek verwendet, ausgetauscht werden.

- `_EnvFilter`
  Die Variable steuert, ob „gefiltert“ werden soll (1) oder nicht (0). Siehe dazu Abschnitt 3.3. `_EnvFilter` ist anfangs 0.

4.2 Das kontext-sensitive Menü

Um Standardfunktionen schnell aufrufen zu können, bietet sich das kontext-sensitive Menü von `multsum` an. Es umfasst einen Großteil der in den folgenden zwei Abschnitten aufgelisteten Prozeduren und ist einfach zu bedienen. Allerdings ruft es die Funktionen meist ohne optionale Parameter auf, so dass bei speziellen Funktionsaufrufen eine manuelle Eingabe der Befehle zu bevorzugen ist. Das Menü wird mit einem Klick der rechten Maustaste auf einen in Frage kommenden zulässigen Term, eine Rekursionsgleichung (bzw. Mengen von Rekursionen) oder auf eine Zertifikats-Rekursion (bzw. Mengen von Zertifikaten) geöffnet[2]. Je nach Art des Ausdrucks erscheinen dann die zugehörigen Prozeduren von `multsum`. Es ist zu beachten, dass die Hauptvariable mit n oder m bezeichnet werden muss und die Summationsvariablen mit i, j und/oder k.

[2] die Ausdrücke müssen sich in der Ausgabezeile befinden!

4.3 Die Haupt-Prozeduren

In diesem Abschnitt werden die wichtigsten Prozeduren von `multsum` vorgestellt. Die Prozedur `sum_recursion` ist dabei die Prozedur, die wir in dem nächsten Kapitel am häufigsten verwenden werden und die bisher noch nicht zum Einsatz gekommen ist. Sie beinhaltet die effizientesten Prozeduren, wie z.B. `find_rec` und `sum_cer`, und ist daher unser bestes und benutzerfreundlichstes Hilfsmittel zur Ermittlung von holonomen Rekursionen bzw. geschlossenen Formen mehrfacher Summen. Für jede Funktion wird im Folgenden eine Tabelle aufgeführt, die Informationen über die Eingabe, den verwendeten Algorithmus (falls in dieser Arbeit beschrieben), die Funktionsweise und die Ausgabe bereitstellt. Daher eignet sich dieser und der nächste Abschnitt auch als Hilfestellung und Nachschlagewerk für den Anwender. Optionale Parameter werden mit (opt.) gekennzeichnet.

`find_krec`

EINGABE: $F, n, \mathbf{k}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- (opt.) eine Strukturmenge S oder eine Standard-Strukturmenge $S_{I\mathbf{J}}$
- (opt.) Termvariable F_{var}
- (opt.) Koeffizientenvariable a_{var}

ALGO: Algorithmus 2.1 + Algorithmus 2.2

INFO: Die Prozedur bestimmt zu einem zulässigen Term F in n und $\mathbf{k}$ $\mathbf{k}$-freie Rekursionsgleichungen. Bei Eingabe einer rechteckigen Strukturmenge $S_{I\mathbf{J}}$ oder einer allgemeinen Strukturmenge S, werden nur bzgl. dieser Strukturmenge $\mathbf{k}$-freie Rekursionen gesucht. Andernfalls werden, beginnend mit einer kleinen Strukturmenge, die Strukturmengen solange vergrößert, bis $\mathbf{k}$-freie Rekursionsgleichungen gefunden werden.

AUSGABE: Eine Menge von $\mathbf{k}$-freien Rekursionsgleichungen. Die Eingabe der Variablen F_{var} hat zur Folge, dass der zulässige Term als $F_{var}(n, \mathbf{k})$ in der Ausgabe erscheint. Gibt man zusätzlich zu F_{var} eine Variable a_{var} ein, so bekommt man schließlich eine allgemeine $\mathbf{k}$-freie Rekursionsgleichung mit $a_{var\,i\mathbf{j}}$ als Koeffizienten.

`find_rec`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}|K, M$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- Standard-Strukturmenge $S_{I\mathbf{J}}$ oder Strategie K
- obere Grenze M
- (opt.) Zertifikatsprädikat Z
- (opt.) Termvariable F_{var}
- (opt.) Koeffizientenvariable a_{var}

ALGO: Algorithmus 3.4 + Algorithmus 3.7

INFO: Die Prozedur bestimmt zu einem zulässigen Term F in n und $\mathbf{k}$ Rekursionsgleichungen. Ist $M = 0$, so wird Algorithmus 3.4 verwendet

und demzufolge werden nur **k**-freie Rekursionen bestimmt. Die erste Verallgemeinerung findet Anwendung bei $M > 0$. In diesem Fall ist M die obere Grenze für die allgemeinen Koeffizientenpolynome in n und **k**. Bei Eingabe einer rechteckigen Strukturmenge $S_{I\mathbf{J}}$, wird nur bzgl. der P-maximalen Strukturmenge von $S_{I\mathbf{J}}$ eine Rekursion gesucht. Andernfalls wird ein K eingegeben. Ist $K = 0$, so werden die Strukturmengen nach der groben Strategie vergrößert. Für ein $K > 0$ wird die Menge $\overline{M}_k$ bestimmt[3] und es wird versucht bzgl. der in $\overline{M}_k$ enthaltenen P-maximalen Strukturmengen Rekursionsgleichungen zu finden.

AUSGABE: Eine Menge von nicht-trivialen Zertifikats-Rekursionsgleichungen. Ist $Z = 0$, so wird eine Menge von Rekursionsgleichungen ausgegeben, für die nicht notwendigerweise entsprechende nicht-triviale Zertifikate existieren müssen. Der voreingestellte Wert ist $Z = 1$. Die Eingabe der Variablen F_{var} hat zur Folge, dass der zulässige Term als $F_{var}(n, \mathbf{k})$ in der Ausgabe erscheint. Gibt man zusätzlich zu F_{var} eine Variable a_{var} ein, so bekommt man schließlich eine allgemeine (Zertifikats)-Rekursionsgleichung mit $a_{var\,i\mathbf{j}}$ als Koeffizienten.

`find_cer`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}} | K, M$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) **k**
- Standard-Strukturmenge $S_{I\mathbf{J}}$ oder Strategie K
- obere Grenze M
- (opt.) Zertifikatsprädikat Z
- (opt.) Termvariable F_{var}
- (opt.) Koeffizientenvariable a_{var}

ALGO: Algorithmus 3.9

INFO: Die Prozedur bestimmt zu einem zulässigen Term F in n und **k** Zertifikats-Rekursionsgleichungen. M ist die obere Grenze für die allgemeinen Koeffizientenpolynome in n und **k**. Bei Eingabe einer rechteckigen Strukturmenge $S_{I\mathbf{J}}$, wird nur bzgl. der P-maximalen Strukturmenge von $S_{I\mathbf{J}}$ ein Zertifikat gesucht. Andernfalls wird ein K eingegeben. Ist $K = 0$, so werden die Strukturmengen nach der groben Strategie vergrößert. Für ein $K > 0$ wird die Menge $\overline{M}_k$ bestimmt und es wird versucht bzgl. der in $\overline{M}_k$ enthaltenen P-maximalen Strukturmengen Rekursionsgleichungen zu finden.

AUSGABE: Eine Menge von nicht-trivialen Zertifikats-Rekursionsgleichungen. Ist $Z = 0$, so wird eine Menge von Rekursionsgleichungen ausgegeben, für die nicht notwendigerweise entsprechende nicht-triviale Zertifikate existieren müssen. Der voreingestellte Wert ist $Z = 1$. Die Eingabe der Variablen F_{var} hat zur Folge, dass der zulässige Term als $F_{var}(n, \mathbf{k})$ in der Ausgabe erscheint. Gibt man zusätzlich

[3] siehe Seite 38

zu F_{var} eine Variable a_{var} ein, so bekommt man schließlich eine allgemeine (Zertifikats)-Rekursionsgleichung mit $a_{var\,i\mathbf{j}}$ als Koeffizienten.

`sum_rec`

EINGABE: *rec*|*rec_set*

- eine Rekursionsgleichung *rec* oder
  eine Menge von Rekursionsgleichungen *rec_set*
- (opt.) Summenvariable S_{var}
- (opt.) Prädikat *all*

INFO: Die Prozedur summiert die gegebene(n) Rekursionsgleichung(en) über alle Summationsvariablen und bestimmt eine holonome Rekursionsgleichung minimaler Ordnung in der Hauptvariablen für die Summe. Dabei wird auch versucht, Rekursionen gleicher Ordnung in eine Rekursion kleinerer Ordnung zu transformieren.
Man beachte: Ein endlicher Träger wird vorausgesetzt.

AUSGABE: Es wird eine Rekursionsgleichung minimaler Ordnung für die Summe ausgegeben. Bei Eingabe einer zusätzlichen Variable S_{var} erscheint in der Ausgabe S_{var} für die Summe. Ist $all = 1$, dann werden alle Rekursionsgleichungen, die berechnet werden, in einer Menge ausgegeben. Möchte man beide optionale Parameter verwenden, so gibt man zuerst S_{var} und dann die 1 ein.

`sum_cer`

EINGABE: *cer*|*cer_set*

- eine Zertifikats-Rekursionsgleichung *cer* oder
  eine Menge von Zertifikats-Rekursionsgleichungen *cer_set*
- (opt.) Summenvariable S_{var}
- (opt.) Prädikat *all*

INFO: Die Prozedur summiert die gegebene(n) Zertifikats-Rekursionsgleichung(en) über alle Summationsvariablen und bestimmt eine holonome Rekursionsgleichung minimaler Ordnung in der Hauptvariablen für die Summe. Dabei wird auch versucht, Rekursionen gleicher Ordnung in eine Rekursion kleinerer Ordnung zu transformieren.
Man beachte: Ein endlicher Träger wird vorausgesetzt.

AUSGABE: Es wird eine Rekursionsgleichung minimaler Ordnung für die Summe ausgegeben. Bei Eingabe einer zusätzlichen Variable S_{var} erscheint in der Ausgabe S_{var} für die Summe. Ist $all = 1$, dann werden alle Rekursionsgleichungen, die berechnet werden, in einer Menge ausgegeben. Möchte man beide optionale Parameter verwenden, so gibt man zuerst S_{var} und dann die 1 ein.

`closed_form`

EINGABE: $F, n, \mathbf{k}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- (opt.) Strategie K
- (opt.) obere Grenze M

INFO: Die Prozedur bestimmt eine geschlossene Form für die Summe von F, falls `find_rec` eine Rekursion erster Ordnung findet. Bei Eingabe eines Parameters M, wird die obere Grenze des Grades der Koeffizienten-Polynome auf den Wert von M gesetzt. Der Standardwert von M beträgt 1. Gibt man zwei Parameter K und M ein, so wird zunächst eine Rekursion bzgl. $\overline{M}_K$ gesucht. Falls keine Rekursion gefunden wird, so wird K sukzessive erhöht. Voreingestellt ist $K = 1$. Ist $K = 0$, so wird die grobe Strategie verwendet.
Man beachte: Ein endlicher Träger für F wird vorausgesetzt.

AUSGABE: Die geschlossene Form der Summe von F. Wird keine geschlossene Form gefunden, so wird eine Fehlermeldung ausgegeben.

BEISPIEL: Wir können Beispiel 2.2 wie folgt abhandeln

```
> F:=binomial(n,j)*binomial(j,i)*x^i*y^(j-i)*z^(n-j):
> closed_form(F,n,[i,j]);
```

$$S(0)(x+y+z)^n$$

`sum_recursion`

EINGABE: $F, n, \mathbf{k}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- (opt.) Summenvariable S_{var}
- (opt.) Strategie K
- (opt.) obere Grenze M

INFO: Die Prozedur bestimmt eine Rekursionsgleichung minimaler Ordnung[4] für die Summe von F. Bei Eingabe eines Parameters M, wird die obere Grenze des Grades der Koeffizienten-Polynome auf den Wert von M gesetzt. Der Standardwert von M beträgt 1. Gibt man zwei Parameter K und M ein, so wird zunächst eine Rekursion bzgl. $\overline{M}_K$ gesucht. Falls keine Rekursion gefunden wird, so wird K sukzessive erhöht. Voreingestellt ist $K = 1$. Ist $K = 0$, so wird die grobe Strategie verwendet.
Man beachte: Ein endlicher Träger für F wird vorausgesetzt.

AUSGABE: Eine Rekursionsgleichung minimaler Ordnung für die Summe von F. Bei Eingabe von S_{var}, erscheint in der Ausgabe S_{var} als Platzhalter für die Summe. Dabei ist S_{var}, bei Eingabe mehrerer optionaler Parameter, vor allen anderen zusätzlichen Parametern einzugeben.

[4] minimal bzgl. allen Zertifikaten, die gefunden werden. Es können also Rekursionen kleinerer Ordnung existieren!

`to_rec`

EINGABE: *cer*|*cer_set*

- eine Zertifikats-Rekursionsgleichung *cer* oder
  eine Menge von Zertifikats-Rekursionsgleichungen *cer_set*

INFO: Die Prozedur bestimmt zu der gegebenen Zertifikats-Rekursionsgleichung *cer* die zugrunde liegende Rekursionsgleichung bzw. zu *cer_set* eine Menge von Rekursionsgleichungen.

AUSGABE: Die entsprechende(n) Rekursionsgleichung(en).

`to_cer`

EINGABE: *rec*|*rec_set*

- eine Rekursionsgleichung *rec* oder
  eine Menge von Rekursionsgleichungen *rec_set*
- (opt.) Hauptteilwert *H*

ALGO: Algorithmus 3.6

INFO: Die Prozedur bestimmt zu der gegebenen Rekursionsgleichung *rec* die entsprechende Zertifikats-Rekursionsgleichung bzw. zu *rec_set* eine Menge von Zertifikaten.

AUSGABE: Die entsprechende(n) Zertifikats-Rekursionsgleichung(en). Gibt man ein weiteres, beliebiges Argument *H* ein, so wird nur der Hauptteil der Zertifikate ausgegeben.

`shift_rec`

EINGABE: *rec*|*rec_set*

- eine Rekursionsgleichung *rec* oder
  eine Menge von Rekursionsgleichungen *rec_set*

INFO: Die Prozedur führt eine Indexverschiebung der Rekursion *rec* bzw. der Rekursionen von *rec_set* durch.

AUSGABE: Die normalisierte(n) Rekursionsgleichung(en). Bei diesen Rekursionen treten bzgl. den auftretenden Variablen nur negative Shifts auf.

`shift_cer`

EINGABE: *cer*|*cer_set*

- eine Zertifikats-Rekursionsgleichung *cer* oder
  eine Menge von Zertifikats-Rekursionsgleichungen *cer_set*

INFO: Die Prozedur führt eine Indexverschiebung des Zertifikats *cer* bzw. der Zertifikate von *cer_set* durch.

AUSGABE: Die normalisierte(n) Zertifikats-Rekursion(en), d.h. die zugrunde liegenden Zertifikats-Operatoren besitzen nun die Gestalt (3.4).

`check_rec`

EINGABE: $rec|rec\_set, F$

- eine Rekursionsgleichung rec oder
  eine Menge von Rekursionsgleichungen rec_set
- zulässiger Term F

INFO: Die Prozedur überprüft, ob der zulässige Term F die gegebene Rekursionsgleichung rec bzw. die Rekursionen von rec_set erfüllt.

AUSGABE: Genügt F einer Rekursionsgleichung, so wird `true` ausgegeben, andernfalls `false`. Bei einer Eingabe von rec_set liefert `check_rec` eine Liste mit booleschen Werten.

`check_cer`

EINGABE: $cer|cer\_set, F$

- eine Zertifikats-Rekursionsgleichung cer oder
  eine Menge von Zertifikats-Rekursionsgleichungen cer_set
- zulässiger Term F

INFO: Die Prozedur überprüft, ob der zulässige Term F die gegebene Zertifikats-Rekursionsgleichung cer bzw. die Zertifikate von cer_set erfüllt.

AUSGABE: Genügt F einer Zertifikats-Rekursionsgleichung, so wird `true` ausgegeben, andernfalls `false`. Bei einer Eingabe von cer_set liefert `check_cer` eine Liste mit booleschen Werten.

`order_rec`

EINGABE: $rec|rec\_set$

- eine Rekursionsgleichung rec oder
  eine Menge von Rekursionsgleichungen rec_set
- (opt.) Summationsvariable k

INFO: Die Prozedur bestimmt die Ordnungen der gegebenen Rekursionsgleichungen rec_set bzw. die Ordnung der gegebenen Rekursionsgleichung rec in der Hauptvariablen. Wird zusätzlich noch eine Summationsvariable k eingegeben, so wird die Ordnung in k bestimmt.

AUSGABE: Bei Eingabe von rec_set wird eine Liste der Ordnungen ausgegeben und bei Eingabe von rec die Ordnung von rec.

`order_cer`

EINGABE: $cer|cer\_set$

- eine Zertifikats-Rekursionsgleichung cer oder
  eine Menge von Zertifikats-Rekursionsgleichungen cer_set
- (opt.) Summationsvariable k

INFO: Die Prozedur bestimmt die Ordnungen der gegebenen Zertifikats-Rekursionsgleichungen *cer_set* bzw. die Ordnung der gegebenen Zertifikats-Rekursionsgleichung *cer* in der Hauptvariablen. Wird zusätzlich noch eine Summationsvariable k eingegeben, so wird die Ordnung in k bestimmt.

AUSGABE: Bei Eingabe von *cer_set* wird eine Liste der Ordnungen ausgegeben und bei Eingabe von *cer* die Ordnung von *cer*.

`add_rec`

EINGABE: rec_1, rec_2

- Rekursionsgleichung rec_1
- Rekursionsgleichung rec_2

INFO: Die Prozedur addiert zwei Rekursionsgleichungen und zwar so, dass ein Summand eliminiert wird. Stimmen die Ordnungen der Rekursionen rec_1 und rec_2 überein, so entsteht entweder eine Rekursionsgleichung kleinerer Ordnung oder die triviale Gleichung $0 = 0$. Zwei Rekursionsgleichungen kann man übrigens auch ganz normal mit dem Operator + addieren.

AUSGABE: Die aus der Addition entstandene Rekursionsgleichung.

`add_cer`

EINGABE: cer_1, cer_2

- Zertifikats-Rekursion cer_1
- Zertifikats-Rekursion cer_2
- (opt.) Additionsparameter *add*

INFO: Die Prozedur addiert zwei Zertifikats-Rekursionsgleichungen und zwar so, dass ein Summand des Hauptteils eliminiert wird. Bei Eingabe von *add* wird keine Elimination durchgeführt. Ist *add* positiv, werden die beiden Zertifikate addiert und bei negativem Parameter *add* wird cer_2 von cer_1 abgezogen.

AUSGABE: Das aus der Addition bzw. Subtraktion entstandene Zertifikat.

`solve_rec`

EINGABE: *rec*

- eine Rekursionsgleichung *rec*
- (opt.) Gleichung $S(i) = a$

INFO: Die Prozedur löst eine Rekursionsgleichung *rec* erster Ordnung.

AUSGABE: Ist *rec* eine Rekursionsgleichung erster Ordnung, so wird die entsprechende Lösung ausgegeben. Andernfalls wird ein Fehler gemeldet. Bei Eingabe einer weiteren Gleichung der Form $S(i) = a$ für ein $i \in \mathbb{N}_0$, wird a als Anfangswert genutzt und es wird daraus die vollständige Lösung berechnet. Ohne $S(i) = a$ erscheint in der Ausgabe zusätzlich ein Platzhalter $S(i)$ für den Startwert.

`Smax`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- Standard-Strukturmenge $S_{I\mathbf{J}}$
- (opt.) Grenze *ext*

ALGO: Algorithmus 3.3

INFO: Die Prozedur bestimmt zu einer Standard-Strukturmenge $S_{I\mathbf{J}}$, die kleinste P-maximale Strukturmenge $S_{\max}$, die $S_{I\mathbf{J}}$ enthält.

AUSGABE: Die entsprechende P-maximale Strukturmenge $S_{\max}$. Falls die Umgebungsvariable `_EnvSmax` so klein gewählt wurde, dass ein Element von $S_{\max}$ auf dem Rand von der $S_{\max}$ umfassenden Strukturmenge S liegt, dann wird ein Warnhinweis ausgegeben. In diesem Fall ist es möglich, dass $S_{\max}$ nicht vollständig ist. Mit dem optionalen Parameter *ext* kann man die Größe von S beeinflussen. Dabei ist der voreingestellte Wert von *ext* der Wert von `_EnvSmax`. Ist `_EnvC` $= 0$, so spielen *ext* und `_EnvSmax` nur bei unendlichen Strukturmengen eine Rolle, da dann die kleinste rechteckige Strukturmenge S, die $S_{\max}$ enthält, mit Hilfe des Simplex-Algorithmus ermittelt wird.

`draw_Smax`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- Standard-Strukturmenge $S_{I\mathbf{J}}$
- (opt.) Grenze *ext_view*

INFO: Die Prozedur zeichnet die kleinste P-maximale Strukturmenge, die $S_{I\mathbf{J}}$ enthält, sofern $r = 1$ oder $r = 2$ ist. Mit dem Wert *ext_view* kann man den Zeichenbereich festlegen. Dabei wird der Sichtbereich aller Achsen in beide Richtungen um den Betrag von *ext_view* erweitert. Der voreingestellte Wert beträgt *ext_view* $= 3$. Für negatives *ext_view* werden keine Struktur-Hyperebenen gezeichnet.

AUSGABE: 2D bzw. 3D-Plot der P-maximalen Strukturmenge. Die schwarzen Punkte gehören zu $S_{I\mathbf{J}}$. Die restlichen Punkte gehören zusammen mit den schwarzen Punkten zu der P-maximalen Strukturmenge und die Randpunkte werden umrandet dargestellt.

`draw_support`

EINGABE: $F, n, \mathbf{k}, bd$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- obere Schranke *bd*
- (opt.) Wert der Hauptvariablen $\overline{n}$

INFO: Die Prozedur veranschaulicht einen Ausschnitt des Trägers eines zulässigen Terms in einem 2D bzw. 3D-Plot.

AUSGABE: Alle Punkte $(n, \mathbf{k})$, für die der zulässige Term $F(n, \mathbf{k})$ nicht verschwindet, werden in ein Koordinatensystem eingezeichnet. Dabei ist bd die obere Schranke (und somit $-bd$ die untere Schranke) des Zeichenbereichs bzgl. den Summationsvariablen. Der Plotbereich von n liegt zwischen 0 und bd. Wird zusätzlich $\overline{n}$ eingegeben, so wird im Fall $\mathbf{k} = (k_1, k_2)$ ein zwei-dimensionaler Plot bzgl. $F(\overline{n}, \mathbf{k})$ erzeugt. Es ist zu beachten, dass bei Eingabe eines binomischen Ausdrucks, ein Ausschnitt des Trägers dieses Terms gezeichnet wird. Möchte man den Träger des zugehörigen zulässigen Terms, so verwendet man zuerst die Funktion `simpcomb` (vereinfacht hypergeometrische Ausdrücke) aus `hsum` und wendet darauf `draw_support` an. Werte, für die der zulässige Term nicht wohldefiniert ist, erscheinen grau.

4.4 Einige Hilfs-Prozeduren

An dieser Stelle werden einige Hilfs-Prozeduren vorgestellt, die teilweise Verwendung in den Haupt-Prozeduren finden. Diese Prozeduren können aber auch separat aufgerufen werden.

`check_hyper`

EINGABE: $F, n, \mathbf{k}$

- Term F
- Summationsvariable(n) $\mathbf{k}$
- Hauptvariable n

INFO: Die Prozedur überprüft, ob der eingegebene Term F hypergeometrisch in n und $\mathbf{k}$ ist oder nicht.

AUSGABE: Ist F hypergeometrisch, so wird nichts zurückgegeben, andernfalls wird eine Fehlermeldung ausgegeben.

`check_proper`

EINGABE: $F, n, \mathbf{k}$

- Term F
- Summationsvariable(n) $\mathbf{k}$
- Hauptvariable n

INFO: Die Prozedur überprüft, ob der eingegebene Term F zulässig ist oder nicht. Dabei wird u.a. mittels `check_hyper` getestet, ob F überhaupt hypergeometrisch ist.

AUSGABE: Ist F zulässig, so wird das Polynom und der aus den Gamma-Funktionen bestehende Term in einer Liste ausgegeben. Andernfalls ist F entweder hypergeometrisch, aber nicht zulässig oder nicht hypergeometrisch und ein entsprechender Fehler wird gemeldet.

`properterm`

EINGABE: $F, n, \mathbf{k}$

- Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$

INFO: Die Prozedur zerlegt, sofern F zulässig ist, F in seine Bestandteile. Dabei wird u.a. die Prozedur `check_proper` verwendet.

AUSGABE: Ist F zulässig, so wird eine Liste mit drei Elementen ausgegeben. Das erste Element ist der Polynom-Teil von F. Das zweite Element besteht aus einer Liste mit sämtlichen Koeffizienten der Fakultätsterme und sieht wie folgt aus

$$\Big[\ [a_1,\ldots,a_{pp}],\big[[b_{11},\ldots,b_{1pp}],\ldots,[b_{r1},\ldots,b_{rpp}]\big],[c_1,\ldots,c_{pp}],$$
$$[u_1,\ldots,u_{qq}],\big[[v_{11},\ldots,v_{1qq}],\ldots,[v_{r1},\ldots,v_{rqq}]\big],[w_1,\ldots,w_{qq}]\ \Big].$$

Das dritte Element ist schließlich der Potenz-Teil. Ist F kein zulässiger Term, so wird eine Fehlermeldung ausgegeben.

`fac_diff`

EINGABE: $F, n, \mathbf{k}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$

ALGO: siehe (2.12)

INFO: Die Prozedur bestimmt die Fakultäts-Differenz von F.

AUSGABE: Die Fakultäts-Differenz von F als Liste.

`structure_fct`

EINGABE: $F, n, \mathbf{k}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$

ALGO: Algorithmus 3.1

INFO: Die Prozedur bestimmt die Strukturfunktionen s_F von F.

AUSGABE: Eine Liste bestehend aus weiteren Listen, die die Strukturfunktionen repräsentieren. Dabei werden in den Listen lediglich die Koeffizienten der Strukturfunktionen s_F gespeichert und ausgegeben.

`boundpts_Sstd`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- Standard-Strukturmenge $S_{I\mathbf{J}}$

ALGO: Algorithmus 3.2

INFO: Die Prozedur bestimmt die Randpunkte von allen s_F bzgl. $S_{I\mathbf{J}}$. Dabei wird u.a. die Prozedur `structure_fct` verwendet.

AUSGABE: Eine Liste bestehend aus zwei weiteren Listen. Die erste Liste beinhaltet die Koeffizienten der Strukturfunktionen von F und die zweite Liste die Randpunkte von allen s_F bzgl. $S_{I\mathbf{J}}$.

`boundpts_S`

EINGABE: $F, n, \mathbf{k}, S$

- zulässiger Term F
- Summationsvariable(n) $\mathbf{k}$
- Hauptvariable n
- Strukturmenge S

INFO: Die Prozedur bestimmt die Randpunkte von allen s_F bzgl. S. Dabei wird u.a. die Prozedur `structure_fct` verwendet.

AUSGABE: Eine Liste bestehend aus zwei weiteren Listen. Die erste Liste beinhaltet die Koeffizienten der Strukturfunktionen von F und die zweite Liste die Randpunkte von allen s_F bzgl. S.

`decompose_S`

EINGABE: S

- Strukturmenge S

ALGO: Algorithmus 3.8

INFO: Die Prozedur zerlegt die Strukturmenge S in $S_0, \ldots, S_r$ gemäß Algorithmus 3.8.

AUSGABE: Eine Liste bestehend aus den Mengen $S_0, \ldots, S_r$.

`inequat`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}, x$

- zulässiger Term F
- Standard-Strukturmenge $S_{I\mathbf{J}}$
- Hauptvariable n
- Variable x
- Summationsvariable(n) $\mathbf{k}$

INFO: Die Prozedur bestimmt die Ungleichungen, die die P-maximale Strukturmenge von $S_{I\mathbf{J}}$ eingrenzen.

AUSGABE: Eine Menge bestehend aus den Ungleichungen in der Variablen x.

`Scov`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}$

- zulässiger Term F
- Standard-Strukturmenge $S_{I\mathbf{J}}$
- Hauptvariable n
- (opt.) Grenze ext
- Summationsvariable(n) $\mathbf{k}$

INFO: Die Prozedur bestimmt die kleinste rechteckige Strukturmenge S (siehe Algorithmus 3.3), die die P-maximale Strukturmenge $S_{\max}$

von $S_{I\mathbf{J}}$ enthält. Ist $S_{\max}$ nicht endlich, so wird mit ext eine Strukturmenge erzeugt, die bzgl. des Nullpunkts zentriert ist und die die Gestalt eines Quadrats bzw. Würfels der Kantenlänge $2ext + 1$ besitzt. Wird ext nicht eingegeben und ist $S_{\max}$ nicht endlich, so wird ext auf den Wert der Umgebungsvariablen `_EnvSmax` gesetzt.

AUSGABE: Die ermittelte Strukturmenge wird, ähnlich wie bei Standard-Strukturmengen, wie folgt ausgegegeben

$$\big[[i_{\min}, i_{\max}], [j_{\min 1}, j_{\max 1}], \ldots, [j_{\min r}, j_{\max r}]\big],$$

wobei die auftretenden Werte gemäß Algorithmus 3.3 bestimmt werden. Im Falle einer nicht endlichen Strukturmenge lautet die Ausgabe

$$\big[[-ext, ext], [-ext, ext], \ldots, [-ext, ext]\big].$$

`Sstd`

EINGABE: $S_{I\mathbf{J}}$

- Standard-Strukturmenge $S_{I\mathbf{J}}$

INFO: Die Prozedur gibt die Standard-Strukturmenge $S_{I\mathbf{J}}$ in der Schreibweise allgemeiner Strukturmengen aus.

AUSGABE: Die entsprechende allgemeine Strukturmenge.

`select_Smax`

EINGABE: $F, n, \mathbf{k}, K$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- Strategie K
- (opt.) Grenze ext

ALGO: Algorithmus 3.5

INFO: Die Prozedur bestimmt die Menge $\overline{M}_K$.

AUSGABE: Es wird eine Liste ausgegeben, deren Elemente Listen sind, die zwei Strukturmengen beinhalten. Die jeweils zweite Menge ist die P-maximale Strukturmenge $S_{\max}$ aus $\overline{M}_K$, die aus der ersten Menge, der zugrunde liegenden rechteckigen Strukturmenge, entsteht. Ist die Umgebungsvariable `_EnvC` 1, so kann man mit dem optionalen Argument ext (siehe Algorithmus 3.3) die Größe der Strukturmenge beeinflussen, die die P-maximalen Strukturmengen enthalten soll.

`select_Sstd`

EINGABE: $F, n, \mathbf{k}, K$

- zulässiger Term F
- Hauptvariable n
- Summationsvariable(n) $\mathbf{k}$
- Strategie K

INFO: Die Prozedur bestimmt die zugrunde liegenden Standard-Strukturmengen der P-maximalen Strukturmengen aus der Menge $\overline{M}_K$.

AUSGABE: Es wird eine Liste mit den Standard-Strukturmengen ausgegeben.

`is_finite`

EINGABE: $F, n, \mathbf{k}, S_{I\mathbf{J}}$

- zulässiger Term F
- Summationsvariable(n) $\mathbf{k}$
- Hauptvariable n
- Standard-Strukturmenge $S_{I\mathbf{J}}$

INFO: Die Prozedur überprüft, ob die kleinste P-maximale Strukturmenge $S_{\max}$, die die Standard-Strukturmenge $S_{I\mathbf{J}}$ enthält, endlich ist.

AUSGABE: Ist $S_{\max}$ endlich, so werden die Grenzen der kleinsten $S_{\max}$ umfassenden, rechteckigen Strukturmenge in einer Liste ausgegeben. Andernfalls `false`.

`max_order`

EINGABE: $F, n, \mathbf{k}$

- zulässiger Term F
- Summationsvariable(n) $\mathbf{k}$
- Hauptvariable n

INFO: Die Prozedur bestimmt eine obere Schranke für die Ordnung der $\mathbf{k}$-freien Rekursion für F, die nach dem Existenzsatz existieren muss.

AUSGABE: Die obere Schranke (2.17).

`ani_Smax`

EINGABE: $F, n, \mathbf{k}, K$

- zulässiger Term F
- Summationsvariable(n) $\mathbf{k}$
- Hauptvariable n
- Strategie K

INFO: Die Prozedur veranschaulicht die Vergrößerungsstrategie der Strukturmengen mit $\overline{M}_K$ und zeichnet die P-maximalen Strukturmengen von $\overline{M}_K$ in einer Animation, wobei diese Strukturmengen $S_{\max}$ noch geeignet verschoben werden, so dass das Minimum bzgl. jeder Komponente der Elemente der $S_{\max}$ 0 ist.

AUSGABE: Eine Animation der Vergrößerungsstrategie mit $\overline{M}_K$.

`duration`

EINGABE: '*proc*'

- eine Prozedur *proc*

INFO: `duration` misst die Zeit, wie lange die Ausführung der Prozedur *proc* dauert. Die Eingabe ist dabei in ' ' zu setzen.

AUSGABE: Die benötigte Zeit in Sekunden.

4.5 Vergleich zwischen Maple und C

Wir betrachten an dieser Stelle einige Beispiele, aus denen hervorgeht, dass die C-Funktionen den Maple-Prozeduren vorzuziehen sind. Wir untersuchen dazu die Dauer der Ausführung der Prozedur `Smax` für verschiedene Eingaben. Zunächst sei $r = 2$ mit

$$F(n,i,j) = \binom{n}{j}\binom{j}{i} x^i y^{j-i} z^{n-j}.$$

Wir tragen die Zeiten von `Smax` in Sekunden in eine Tabelle ein, wobei in der Kopfzeile die Werte von I der zugrunde liegenden Strukturmengen $S_{I,I,I}$ abgetragen werden und in der Leitspalte die Werte von `_EnvSmax`. Es werden dabei ausschließlich C-Prozeduren verwendet, wohingegen die Zeiten der letzten Zeile auf die Maple-Prozeduren zurückzuführen sind.

	1	2	3	4	5	6	7	8	9	10
3	0.000									
6	0.000	0.010								
9	0.000	0.010	0.010							
12	0.010	0.010	0.100	0.040						
15	0.010	0.030	0.021	0.030	0.040					
18	0.030	0.030	0.020	0.040	0.041	0.060				
21	0.040	0.040	0.041	0.050	0.060	0.070	0.080			
24	0.060	0.050	0.060	0.070	0.070	0.080	0.100	0.131		
27	0.070	0.080	0.080	0.081	0.100	0.100	0.120	0.150	0.170	
30	0.110	0.100	0.110	0.111	0.140	0.130	0.150	0.160	0.191	0.330
33	0.140	0.130	0.131	0.150	0.150	0.170	0.181	0.200	0.220	0.231
	0.881	0.160	0.361	0.641	1.011	1.823	3.004	4.847	7.851	12.448

Es treten nur die Einträge auf, bei denen `Smax` die vollständige P-maximale Strukturmenge bestimmt. Für $r = 1$ ergibt sich ein ähnliches Bild. Betrachten wir nun $r = 3$ mit

$$F(n,i,j,k) = \binom{i+j}{i}\binom{j+k}{j}\binom{n-i-j}{k}\binom{n-j-k}{n-i-j-k},$$

so erhalten wir folgende Tabelle

	1	2	3	4	5	6	7
3	0.091						
6	0.030	0.040					
9	0.100	0.100	0.161				
12	0.220	0.240	0.321	0.310			
15	0.491	0.580	0.541	0.651	0.731		
18	0.961	1.042	1.041	1.152	1.241	1.503	
21	1.713	1.762	1.893	1.903	2.093	2.333	2.834
24	2.945	2.924	2.994	3.165	3.264	3.596	4.115
	0.942	0.591	1.922	6.109	20.189	61.920	175.630

Man beachte, dass die kleinste P-maximale Strukturmenge $S_{\max}$, die $S_{7,7,7,7}$ enthält, bereits 27000 Elemente besitzt. Man kann $S_{\max}$ mit der C-Prozedur innerhalb drei Sekunden (mit `_EnvSmax` $= 21$) bestimmen, während die Maple-Prozedur fast drei Minuten benötigt. Diese großen Strukturmengen spielen bei uns jedoch keine Rolle, da man die daraus entstehenden Gleichungssysteme aufgrund ihrer Größe nicht mehr lösen kann. In beiden Beispielen kann man jede von der Maple-Prozedur erzielten Zeit durch eine geeignete Wahl von `_EnvSmax` mit der C-Funktion unterbieten. Der Nachteil ist, dass man vorher nicht weiß, wie groß man `_EnvSmax` wählen muss. Der voreingestellte Wert 10 reicht allerdings für Betrachtungen zulässiger Terme in n und $\mathbf{k}$, deren Koeffizienten in n und $\mathbf{k}$ klein sind, fast immer aus, da wir in vielen Fällen schon für die Mengen $\overline{M}_1$ und $\overline{M}_2$ eine Rekursion bekommen. Sind die Koeffizienten der hypergeometrischen Variablen n und $\mathbf{k}$ groß, so muss man die Umgebungsvariable `_EnvSmax` erhöhen, um keinen Warnhinweis und somit mit Sicherheit die komplette P-maximale Strukturmenge zu erhalten. Alternativ kann man dann auch durch eine Änderung von `_EnvC` die Maple-Prozeduren verwenden. Ein simples Beispiel, für das die obere Schranke 10 nicht ausreicht, ist

$$F(n,k) = \binom{10n}{k}$$

und $S_{2,2}$. Um die kleinste P-maximale Strukturmenge zu erhalten, die $S_{2,2}$ enthält, ist `_EnvSmax` auf 22 zu setzen. Dann wird immer noch ein Warnhinweis ausgegeben, der schließlich bei dem Wert 23 ausbleibt. Betrachten wir nun

$$F(n,i,j) = \binom{i+j}{i}\binom{n-i}{j}\binom{n-j}{n-i-j}$$

und die Prozedur `select_Smax`, die Verwendung in `find_rec` und `find_cer` findet. Die Prozedur `Smax` wird innerhalb von `select_Smax` mehrmals aufgerufen, so dass sich wieder der Unterschied zwischen C und Maple bemerkbar macht

```
> F:=binomial(i+j,i)*binomial(n-i,j)*binomial(n-j,n-i-j):
> _EnvSmax:=11:
> _EnvC:=1:
> seq(duration('select_Smax(F,n,[i,j],k)')[1],k=1..5);
```

0.180, 0.281, 0.591, 1.512, 4.086

```
> _EnvC:=0:
> seq(duration('select_Smax(F,n,[i,j],k)')[1],k=1..5);
```

1.372, 2.694, 6.590, 14.671, 32.337

Folglich sollte man den C-Prozeduren den Vorzug geben.

Kapitel 5

Anwendungsbeispiele der Algorithmen

In diesem Kapitel werden wir zeigen, wie man mehrfache Summen mit Hilfe der uns nun zur Verfügung stehenden Prozeduren des Maple-Packages `multsum` bestimmen können. Besitzt ein zulässiger Term F in n und $\mathbf{k}$ einen endlichen Träger und genügend wohldefinierte Werte außerhalb des Trägers, so berechnet man zunächst ein Zertifikat für F und erhält durch Summation eine holonome Rekursion für die Summe $S(n)$. Mittels dieser Rekursion ermittelt man dann eine geschlossene oder vereinfachte Form für $S(n)$. Einige Beispiele dafür haben wir bereits kennen gelernt. Allerdings ist es praktisch nicht immer möglich, eine holonome Rekursion für $S(n)$ zu bestimmen, nämlich dann nicht, wenn F keinen endlichen Träger besitzt. In diesem Fall kann man wenigstens eine inhomogene Rekursionsgleichung aufstellen, die schließlich auch zur Vereinfachung einer mehrfachen Summe dienen kann.

5.1 Grundlagen zum Beweisen binomischer Identitäten mit `multsum`

Da wir in diesem Kapitel binomische Identitäten, wie z.B. die simple Gleichung

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n \,, \ \forall\, n \in \mathbb{N}_0 \tag{5.1}$$

mit dem Maple-Package `multsum` beweisen möchten, weisen wir an dieser Stelle auf den Unterschied zwischen Binomialkoeffizient und dessen Darstellung als zulässiger Term hin. Für den Binomialkoeffizienten gilt, wie wir bereits wissen,

$$\binom{n}{k} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)} \tag{5.2}$$

für alle $n \notin -\mathbb{N}$. Mit der Prozedur `find_rec` erhalten wir eine Rekursion für die rechte Seite von (5.2). Da wir $n \in \mathbb{N}_0$ vorausgesetzt haben, gilt diese Rekursion auch für den Binomialkoeffizienten und wir können durch Summation (5.1) folgern. Diese Argumentation lässt sich aber oftmals nicht anwenden, da das obere

Argument eines Binomialkoeffizienten eine Summationsvariable enthalten und damit auch negativ werden kann. In diesem Fall ist der Binomialkoeffizient definiert, der zugehörige zulässige Term aber nicht. Möchten wir also eine Rekursion für die Summe eines Terms mit Binomialkoeffizienten bestimmen, so berechnen wir mit Hilfe der Prozeduren aus `multsum` zunächst eine Rekursion für den zugehörigen zulässigen Term. Dann müssen wir zeigen, dass diese Rekursion auch für den binomischen Ausdruck gilt. Das kann man durch geeignete Grenzübergänge in der Regel einfach beweisen, so dass wir diesen Aspekt in den folgenden Beispielen vernachlässigen. In den Fällen, wo eine Unterscheidung zwischen Binomialkoeffizient und dessen zulässigen Term notwendig ist, werden wir an entsprechender Stelle darauf hinweisen. Andernfalls werden wir, wie gewohnt, keinen Unterschied zwischen den beiden Begrifflichkeiten machen.

5.2 Summen mit natürlichen Grenzen

Wir betrachten nun ausschließlich zulässige Terme F in n und $\mathbf{k}$, die einen endlichen Träger besitzen. Wie wir bereits aus Satz 3.4 wissen, gilt für die Summe $S(n)$ dieser Terme dann

$$S(n) = \sum_{\mathbf{k}\in\mathbb{Z}^r} F(n,\mathbf{k}) = \sum_{\mathbf{k}\in T_F(n)} F(n,\mathbf{k})$$

mit

$$T_F(n) = \{\mathbf{k} \mid (n,\mathbf{k}) \in D_F \cap (\mathbb{N}_0 \times \mathbb{Z}^r) \text{ und } F(n,\mathbf{k}) \neq 0\}\,,$$

sofern $\mathbb{N}_0 \times \mathbb{Z}^r \subseteq D_F$ ist. In diesem Fall ist $T_F(n)$ die kleinste Menge, über die summiert werden kann, ohne den Wert der Summe zu verändern. Untersuchen wir eine Summe, deren Summationsbereich gleich $T_F(n)$ ist, so reden wir von einer Summe mit natürlichen Grenzen. Alle Beispiele, die wir in den letzten Kapiteln behandelt haben, sind Beispiele, die zu dieser Kategorie von Summen zählen. Manchmal sind zulässige Terme F, die einen endlichen Träger besitzen, nicht für alle $(n,\mathbf{k}) \in \mathbb{N}_0\times\mathbb{Z}^r$ wohldefiniert, so dass man Satz 3.4 nicht anwenden kann. Dann erhält man immer noch eine holonome Rekursion, wenn gewährleistet ist, dass außerhalb von $T_F(n)$ hinreichend viele ganzzahlige $\mathbf{k}$ existieren, für die $F(n,\mathbf{k})$ wohldefiniert ist. Dazu betrachten wir den folgenden Begriff

Definition 5.1 *Sei F ein zulässiger Term in n und $\mathbf{k}$ und $P \in \mathbb{C}[n,\mathbf{k}]\langle L_n, \mathbf{L_k}\rangle$. Ferner seien $I \in \mathbb{N}_0$ und $\mathbf{J} \in \mathbb{N}_0^r$ die Ordnungen des Rekursions-Operators $P(n,\mathbf{k},L_n,\mathbf{L_k})$. Dann ist für jedes $n \in \mathbb{N}_0 + I$*

$$\mathcal{R}^F_{I\mathbf{J}}(n) := \{\mathbf{k} \mid (n-i,\mathbf{k}-\mathbf{j}) \in D_F \cap (\mathbb{N}_0 \times \mathbb{Z}^r) \text{ für alle } i \in [0..I] \text{ und } \mathbf{j} \in [\mathbf{0}..\mathbf{J}]\}$$

der **Summationsbereich von F bzgl. P.**

$\mathcal{R}^F_{I\mathbf{J}}(n)$ ist also der größte Bereich, über den man die Zertifikats-Rekursion $P(n,\mathbf{k},L_n,\mathbf{L_k})F(n,\mathbf{k})$ summieren kann. Mit Hilfe von $\mathcal{R}^F_{I\mathbf{J}}(n)$ können wir Satz 3.4 verallgemeinern und erhalten

Satz 5.1 *Sei F ein zulässiger Term in n und $\mathbf{k}$ mit einem endlichen Träger. Ferner gelte*

$$T_F(n-i)+\mathbf{j} \subseteq \mathcal{R}^F_{I\mathbf{J}}(n) \tag{5.3}$$

für alle $n \in \mathbb{N}_0+I$, $i \in [0..I]$ und $\mathbf{j} \in [\mathbf{0}..\mathbf{J}]$, wobei $P(n,\mathbf{k},L_n,\mathbf{L_k}) = P_0(n,L_n)+\sum_{m=1}^{r}\Delta_{k_m}P_m(n,\mathbf{k},L_n,\mathbf{L_k})$ ein nicht-trivialer Zertifikats-Operator der Ordnung $(I,\mathbf{J})$ ist. Erfüllt F die Rekursionsgleichung

$$P(n,\mathbf{k},L_n,\mathbf{L_k})F(n,\mathbf{k}) = 0,$$

dann gilt für die Summe $S(n) = \sum_{\mathbf{k}\in T_F(n)} F(n,\mathbf{k})$ die holonome Rekursionsgleichung

$$P_0(n,L_n)S(n) = 0.$$

Beweis Sei

$$P(n,\mathbf{k},L_n,\mathbf{L_k}) = \sum_{i=0}^{I}\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}} a_{i\mathbf{j}}(n,\mathbf{k})L_n^i\mathbf{L}_\mathbf{k}^\mathbf{j}$$

der Zertifikats-Operator in Normalform mit Polynomen $a_{i\mathbf{j}} \in \mathbb{C}[n,\mathbf{k}]$. Der Hauptteil von P hat nach (3.14) die Gestalt

$$P_0(n,L_n) = \sum_{i=0}^{I}\Big(\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}} a_{i\mathbf{j}}(n,\mathbf{k}+\mathbf{j}-\mathbf{J})\Big)L_n^i\mathbf{L}_\mathbf{k}^\mathbf{J}$$

und hängt nicht von $\mathbf{k}$ ab. Aus diesem Grund gilt

$$\begin{aligned}
\sum_{\mathbf{k}\in\mathcal{R}^F_{I\mathbf{J}}(n)}\sum_{i=0}^{I}\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}} a_{i\mathbf{j}}(n,\mathbf{k})F(n-i,\mathbf{k}-\mathbf{j}) &= 0\\
\sum_{i=0}^{I}\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}}\sum_{\mathbf{k}\in\mathcal{R}^F_{I\mathbf{J}}(n)} a_{i\mathbf{j}}(n,\mathbf{k})F(n-i,\mathbf{k}-\mathbf{j}) &= 0\\
\sum_{i=0}^{I}\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}}\sum_{\mathbf{k}\in T_F(n-i)+\mathbf{j}} a_{i\mathbf{j}}(n,\mathbf{k})F(n-i,\mathbf{k}-\mathbf{j}) &= 0\\
\sum_{i=0}^{I}\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}}\sum_{\mathbf{k}\in T_F(n-i)} a_{i\mathbf{j}}(n,\mathbf{k}+\mathbf{j})F(n-i,\mathbf{k}) &= 0\\
\sum_{i=0}^{I}\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}}\sum_{\mathbf{k}\in T_F(n-i)+\mathbf{J}} a_{i\mathbf{j}}(n,\mathbf{k}+\mathbf{j}-\mathbf{J})F(n-i,\mathbf{k}-\mathbf{J}) &= 0\\
\sum_{i=0}^{I}\Big(\sum_{\mathbf{j}=\mathbf{0}}^{\mathbf{J}} a_{i\mathbf{j}}(n,\mathbf{k}+\mathbf{j}-\mathbf{J})\Big)\sum_{\mathbf{k}\in T_F(n-i)} F(n-i,\mathbf{k}) &= 0
\end{aligned}$$

Aus der letzten Gleichung folgt die Behauptung. □

Untersuchen wir nun einige Beispiele

Beispiel 5.1 Wir möchten mit Hilfe von `multsum` zeigen, dass die Gleichung

$$\sum_i \sum_j \binom{n}{i}\binom{n+i}{i}\binom{i}{j}^3 = \sum_k \binom{n}{k}^2\binom{n+k}{k}^2$$

für alle $n \in \mathbb{N}_0$ gilt. Die Zahlen auf der rechten Seite der Gleichung nennt man die Apéry-Zahlen, weil sie bei Apérys Beweis der Irrationalität von $\zeta(3)$ eine wesentliche Rolle spielen. Die Identität wurde von Strehl in [Str94] auf sechs verschiedene Art und Weisen bewiesen. Da der Summand der Doppelsumme offensichtlich einen endlichen Träger besitzt, verwenden wir die Prozedur `sum_recursion` und erhalten

```
> F:=binomial(n,i)*binomial(n+i,i)*binomial(i,j)^3:
> sum_recursion(F,n,[i,j],2);

structureset: [1, 0, 0]  equations: 2   variables: 2
structureset: [0, 0, 1]  equations: 30  variables: 21
```

$$-n^3\,\mathrm{S}(n) + (2\,n-1)\,(17\,n^2 - 17\,n + 5)\,\mathrm{S}(n-1) - (n-1)^3\,\mathrm{S}(n-2) = 0$$

```
> G:=binomial(n,k)^2*binomial(n+k,k)^2:
> sum_recursion(G,n,k,2);

structureset: [1, 0]  equations: 3  variables: 2
structureset: [0, 1]  equations: 5  variables: 6
```

$$-n^3\,\mathrm{S}(n) + (2\,n-1)\,(17\,n^2 - 17\,n + 5)\,\mathrm{S}(n-1) - (n-1)^3\,\mathrm{S}(n-2) = 0$$

Es ergeben sich folglich für beide Summen die gleiche Rekursionsgleichung zweiter Ordnung. Vergleichen wir die beiden Summen für $n = 0$ bzw. $n = 1$, so erhalten wir ebenfalls die gleichen Werte 1 bzw. 5. Damit ist obige Identität bewiesen.

Beispiel 5.2 Mit Hilfe von `sum_recursion` können wir auf einfache Art und Weise die Gültigkeit der Identität

$$\sum_{i=0}^{n}\sum_{j=0}^{i}\binom{n}{i}\binom{i}{j}^3 = \sum_{k=0}^{n}\binom{n}{k}^2\binom{2k}{k} \tag{5.4}$$

überprüfen. Alle Grenzen sind natürlich und Summation eines Zertifikats liefert wieder eine holonome Rekursion für die Summe. Es ergibt sich

```
> F:=binomial(n,i)*binomial(i,j)^3:
> sum_recursion(F,n,[i,j]);

structureset: [1, 0, 0]  equations: 2   variables: 2
structureset: [0, 0, 1]  equations: 19  variables: 10
structureset: [1, 0, 1]  equations: 24  variables: 19
structureset: [0, 1, 1]  equations: 38  variables: 31
```

$$-\mathrm{S}(n)\,n^2 + (10\,n^2 - 10\,n + 3)\,\mathrm{S}(n-1) - 9\,(n-1)^2\,\mathrm{S}(n-2) = 0$$

```
> G:=binomial(n,k)^2*binomial(2*k,k):
> sum_recursion(G,n,k);

structureset: [1, 0]  equations: 3  variables: 2
structureset: [0, 1]  equations: 5  variables: 4
structureset: [1, 1]  equations: 7  variables: 7
```

$$-\mathrm{S}(n)\,n^2+(10\,n^2-10\,n+3)\,\mathrm{S}(n-1)-9\,(n-1)^2\,\mathrm{S}(n-2)=0$$

Die Startwerte betragen bei beiden Summen $S(0)=1$ und $S(1)=3$.

Bei beiden Beispielen ist es nicht möglich, durch eine einfache Anwendung des Zeilberger-Algorithmus, eine geschlossene Form für die innere Summe zu bekommen, so dass man die Gleichheit der Summen nicht iterativ beweisen kann.

Beispiel 5.3 Wir möchten nun die auf Carlitz zurückgehende Identität

$$\sum_i \sum_j \binom{i+j}{i}\binom{n-i}{j}\binom{n-j}{n-i-j}=\sum_{k=0}^{n}\binom{2k}{k}, \quad \forall\, n\in\mathbb{N}_0$$

beweisen. Dazu betrachten wir den Träger von dem Summanden F der linken Summe. Es gilt

$$T_F(n)=\left\{(i,j)\in\mathbb{Z}^2 \mid 0\le i\le n \text{ und } 0\le j\le n-i\right\}$$

und somit besitzt F wieder einen endlichen Träger[1].

```
> F:=binomial(i+j,i)*binomial(n-i,j)*binomial(n-j,n-i-j):
> sum_recursion(F,n,[i,j],S);
```

$$(2-4\,n)\,\mathrm{S}(n-2)-\mathrm{S}(n)\,n+(5\,n-2)\,\mathrm{S}(n-1)=0$$

Für die Summe der rechten Seite der zu beweisenden Identität gilt offenbar für alle $m\in[1..n]$

$$S(n)-S(n-m)=\sum_{k=n-m+1}^{n}\binom{2k}{k},$$

so dass mit der Zeile

```
> simpcomb(subs({seq(S(n-m)=S(n)-sum(binomial(2*k,k),k=n-m+1..n),
> m=1..2)},%));
```

$$0=0$$

und mit dem Vergleich der Anfangswerte ($S(0)=1$ und $S(1)=3$) die Behauptung folgt.

Beispiel 5.4 Sei

$$S(n)=\sum_i\sum_j \binom{j-i}{n-i}\binom{t-u-j+i}{r-u-n+i}\binom{u}{i}\binom{t-u}{j-i}\binom{v-t}{s-j}$$

mit $n,s\in\mathbb{N}_0$. Der Darstellung

$$\frac{\Gamma(u+1)\Gamma(t-u+1)\Gamma(v-t+1)}{\Gamma(j-n+1)\Gamma(n-i+1)\Gamma(t+n-j-r+1)\Gamma(r+i-u-n+1)\Gamma(u-i+1)\Gamma(i+1)\Gamma(v+j-s-t+1)\Gamma(s-j+1)}$$

des Summanden F kann man entnehmen, dass

$$T_F(n)=\{(i,j)\in\mathbb{Z}^2 \mid 0\le i\le n \text{ und } n\le j\le s\}$$

[1] man verschaffe sich einen Überblick mit `draw_support`

gilt. Wir verwenden wieder Maple und bekommen[2]

```
> F:=binomial(j-i,n-i)*binomial(t-u-j+i,r-u-n+i)*binomial(u,i)*
> binomial(t-u,j-i)*binomial(v-t,s-j):
> sum_recursion(F,n,[i,j]);
```

$$n\,(n+v-r-s)\,\mathrm{S}(n)+(-1-s+n)\,(r-n+1)\,\mathrm{S}(n-1)=0$$

```
> check_rec(%,binomial(r,n)*binomial(v-r,s-n)*binomial(t-u,r-u));
```

$$\mathit{true}$$

Wir berechnen den Anfangswert $S(0)=\sum_{j=0}^{s}\binom{t-u-j}{r-u}\binom{t-u}{j}\binom{v-t}{s-j}$ mit der Prozedur `closed_form`

```
> G:=binomial(t-u-j,r-u)*binomial(t-u,j)*binomial(v-t,s-j):
> subs(S(0)=binomial(t-u,r-u),closed_form(G,s,j));
```

$$\frac{\mathrm{binomial}(t-u,\,r-u)\,\mathrm{pochhammer}(-v+r,\,s)\,(-1)^s}{s!}$$

Somit haben wir

$$S(n)=\binom{r}{n}\binom{v-r}{s-n}\binom{t-u}{r-u}$$

bewiesen.

Beispiel 5.5 Wir zeigen nun die Linearisierungsformel

$$H_m(x)H_n(x)=\sum_{i=0}^{\min(n,m)}\binom{m}{i}\binom{n}{i}2^i i!H_{m+n-2i}(x) \tag{5.5}$$

für das Produkt zweier Hermitepolynome $H_m(x)$ und $H_n(x)$, die in [And99] zu finden ist. Wir verwenden dafür die Darstellung

$$H_n(x)=n!\sum_{j=0}^{n}\frac{(-1)^j}{j!(n-2j)!}(2x)^{n-2j}$$

aus [Abr64] und bestimmen zuerst eine Rekursion für die Doppelsumme der rechten Seite von (5.5). Maple liefert

```
> F:=n!*(-1)^j/(j!*(n-2*j)!)*(2*x)^(n-2*j):
> sum_recursion(binomial(m,i)*binomial(n,i)*2^i*i!*
>                 subs(n=m+n-2*i,F),n,[i,j]);
```

$$-2\,x\,\mathrm{S}(n-1)+(2\,n-2)\,\mathrm{S}(n-2)+\mathrm{S}(n)=0$$

Mit der nächsten Befehlszeile erhalten wir eine Rekursion für $H_n(x)$. Da $H_m(x)$ nicht von n abhängt, gilt diese Rekursion auch für das Produkt $H_m(x)H_n(x)$.

```
> sum_recursion(F,n,j);
```

$$-2\,x\,\mathrm{S}(n-1)+(2\,n-2)\,\mathrm{S}(n-2)+\mathrm{S}(n)=0$$

[2] `find_rec` findet keine Rekursion erster Ordnung! Zwei unabhängige Rekursionen werden zu einer Rekursion erster Ordnung zusammengefügt.

Die Rekursionen stimmen überein. Ferner bekommen wir jeweils den gleichen Anfangswert $S(0) = H_m(x)$. Für $S(1)$ erhalten wir einerseits $2xH_m(x)$ und andererseits $2mH_{m-1}(x) + H_{m+1}(x)$. Die Ausdrücke sind ebenfalls identisch, was man zusammen mit obiger Rekursion der Hermitepolynome wie folgt einsieht

```
> shift_rec(2*x*H(m)=2*m*H(m-1)+H(m+1));
```

$$-2\,x\,\mathrm{H}(m-1) + (2\,m-2)\,\mathrm{H}(m-2) + \mathrm{H}(m) = 0\,.$$

5.3 Summen mit nicht-natürlichen Grenzen

An den folgenden Beispielen kann man erkennen, dass man, durch Anwendung einiger Tricks, auch Identitäten für Summen, die keine natürlichen Grenzen besitzen und deren Summand keinen endlichen Träger hat, zeigen kann. Man findet das nächste Beispiel in [Lyo02].

Beispiel 5.6 Sei

$$S(n) = \sum_{i\geq 1}\sum_{j}\binom{2i-1}{j}\frac{1}{(2i-1)2^{2i-1}(j-i-n+1/2)}\,, \quad \forall\, n\in\mathbb{N}.$$

Wir möchten beweisen, dass die Gleichung

$$S(n) = -\frac{2}{n}\sum_{k=1}^{n}\frac{1}{2k-1}$$

für alle $n \in \mathbb{N}$ gilt. Zunächst ist zu bemerken, dass der Summand F der Doppelsumme keinen endlichen Träger besitzt und die untere Grenze des Summationsindex i bzgl. der Summe $S(n)$ keine natürliche Grenze ist. Aus diesem Grund betrachten wir den Term

$$F_\delta(n,i,j) = \frac{\Gamma(2i-1+\delta)}{\Gamma(j+1)\Gamma(2i-j)2^{2i-1}(j-i-n+1/2)}$$

mit $\delta > 0$, für den

$$S(n) = \lim_{\delta\to 0}\sum_{i\geq 1}\sum_{j}F_\delta(n,i,j)$$

für hinreichend kleines δ gilt[3]. $F_\delta(n,i,j)$ verschwindet für alle $i \in -\mathbb{N}_0$, so dass die untere Grenze der äußeren Summe von $\sum_{i\geq 1}\sum_j F_\delta(n,i,j)$ natürlich ist. Es gilt jetzt, dass $F_\delta(n,i,j) = 0$ für alle $j < 0$, für alle $j > 2i-1$ und für alle $i \leq 0$ ist. Da der Summationsindex i nach oben unbegrenzt ist, besitzt F_δ immer noch keinen endlichen Träger, so dass die Delta-Terme einer Zertifikats-Rekursion für F_δ bei Summation nicht automatisch verschwinden. Summieren wir ein Zertifikat für F_δ über $i \leq I$ und j, so erhalten wir vorerst eine inhomogene Rekursionsgleichung. Betrachten wir diese Rekursion genauer, so fällt auf, dass für $I \to \infty$ alle auftretenden Ausdrücke (Produkte aus Polynomen $p_\delta(n,I,j)$ und $F_\delta(n-a, I-b, j-c)$), aufgrund des Terms $2^{2(I-b)-1}$,

[3] siehe auch Beweis von Satz 5.2

verschwinden, so dass wir wieder eine holonome Rekursion für $S(n)$ erhalten. Diese Tatsache rechtfertigt die folgende Vorgehensweise

```
> F[delta]:=GAMMA(2*i-1+delta)/
>           (GAMMA(2*i-j)*GAMMA(j+1)*2^(2*i-1)*(j-n-i+1/2));
```

$$F_\delta := \frac{\Gamma(2\,i-1+\delta)}{\Gamma(2\,i-j)\,\Gamma(j+1)\,2^{(2\,i-1)}\,(j-n-i+\frac{1}{2})}$$

```
> subs(delta=0,sum_recursion(F[delta],n,[i,j],S));
```

$$2\,n\,(2\,n-1)\,\mathrm{S}(n)+(-3+2\,n)\,(2\,n-4)\,\mathrm{S}(n-2)+(-8+16\,n-8\,n^2)\,\mathrm{S}(n-1)=0$$

```
> simplify(subs({seq(S(n-m)=n/(n-m)*S(n)+(2/(n-m))*
> sum(1/(2*k-1),k=n-m+1..n),m=1..2)},%));
```

$$0=0$$

In der letzten Zeile verifizieren wir, dass die Summe $-\frac{2}{n}\sum_{k=1}^{n}\frac{1}{2k-1}$, die mit Hilfe von der Prozedur `sum_recursion` erhaltene Rekursion erfüllt. Es bleibt zu zeigen, dass beide Summen auch für $n=1$ und $n=2$ gleich sind. Betrachten wir zunächst $S(1)$. Wegen[4]

$$\begin{aligned} G(n,i)=\sum_j F(n,i,j) &= \frac{1}{2}\sum_j F(n,i,j)+F(n,i,2i-j-1) \\ &= \frac{1}{2}\sum_j F(n,i,j)\left(1+\frac{j-i-n+1/2}{i-j-n-1/2}\right) \end{aligned}$$

erhalten wir für $R(i)=G(1,i)$ mit $R(1)=-4/3$

```
> F:=binomial(2*i-1,j)/((2*i-1)*(2^(2*i-1))*(j-i-n+1/2)):
> symmF:=subs(n=1,(1/2)*F*(1+(j-i-n+1/2)/(i-j-n-1/2))):
> sum_recursion(symmF,i,j,R);
```

$$(8\,i-12)\,\mathrm{R}(i-1)+(-8\,i-4)\,\mathrm{R}(i)=0$$

```
> R(i):=simplify(solve_rec(%,R(1)=-4/3));
```

$$\mathrm{R}(i) := -\frac{4}{4\,i^2-1}$$

```
> S(1):=sum(R(i),i=1..infinity);
```

$$\mathrm{S}(1) := -2$$

und wegen $-\frac{2}{1}\frac{1}{2\cdot1-1}=-2$ stimmen die beiden Summen für $n=1$ überein. Schließlich zeigt man $S(2)=-\frac{4}{3}=-\frac{2}{2}(\frac{1}{2\cdot1-1}+\frac{1}{2\cdot2-1})$ analog. Allerdings muss man zur Bestimmung der inneren Summe diesmal den Zeilberger-Algorithmus bemühen, da der Fasenmyer-Algorithmus lediglich eine Rekursionsgleichung zweiter Ordnung findet.

Allgemein kann man durch Einführung einer zusätzlichen Variablen δ erreichen, dass die Grenzen einer Summe natürlich werden. Wenn wir dann eine Rekursion für diese Summe gefunden haben, so lassen wir δ gegen 0 streben. Dazu betrachten wir

[4] diese Technik, „creative symmetrizing“ genannt, liefert in unserem Fall eine Rekursion erster Ordnung

Satz 5.2 *Sei F ein zulässiger Term in n und $\mathbf{k}$ und*

$$B(n) = \{\mathbf{k} \in \mathbb{Z}^r \mid x_i n + \mathbf{y}_i \cdot \mathbf{k} + z_i \geq 0 \text{ für } i \in [1..R]\}$$

der Summatonsbereich der Summe $S(n) = \sum_{\mathbf{k}\in B(n)} F(n,\mathbf{k})$ mit $x_i \in \mathbb{Z}, \mathbf{y}_i \in \mathbb{Z}^r, z_i \in \mathbb{C}$ und $R \in \mathbb{N}$. Dann erfüllt $S(n)$ eine homogene Rekursionsgleichung mit polynomialen Koeffizienten.

Beweis Sei $\overline{F}_\delta$ der zulässige Term, der entsteht, wenn man alle Zähler-Fakultätsterme $a_p n + \mathbf{b}_p \cdot \mathbf{k} + c_p$ von F, die für bestimmte $(n,\mathbf{k})$ nichtpositiv sind, durch $a_p n + \mathbf{b}_p \cdot \mathbf{k} + c_p + \delta$ ersetzt. Dann ist $\overline{F}_\delta(n,\mathbf{k})$ für hinreichend kleines δ und für alle $n \in \mathbb{N}_0$ und $\mathbf{k} \in \mathbb{Z}^r$ wohldefiniert. Sei

$$F_\delta(n,\mathbf{k}) = \overline{F}_\delta(n,\mathbf{k}) \prod_{i=1}^{R} \frac{\Gamma(x_i n + \mathbf{y}_i \cdot \mathbf{k} + z_i + 1 + \delta)}{\Gamma(x_i n + \mathbf{y}_i \cdot \mathbf{k} + z_i + 1)}.$$

Dann gilt

$$S(n) = \sum_{\mathbf{k}\in B(n)} F(n,\mathbf{k}) = \lim_{\delta\to 0} \sum_{\mathbf{k}\in B(n)} F_\delta(n,\mathbf{k}) = \lim_{\delta\to 0} \sum_{\mathbf{k}\in \mathbb{Z}^r} F_\delta(n,\mathbf{k}) \tag{5.6}$$

und die mittlere Summe von (5.6) besitzt somit natürliche Grenzen. Aus (5.6) und Korollar 3.6 folgt nun mit Summation über alle $\mathbf{k} \in \mathbb{Z}^r$ die Behauptung. □

Man beachte, dass die Hinzunahme von δ die Laufzeit des Algorithmus von Fasenmyer erheblich erhöht. Betrachten wir ein weiteres Beispiel

Beispiel 5.7 Wir möchten eine geschlossene Form für die Summe

$$S(n) = \sum_{i=0}^{n} \sum_{j=0}^{i} \binom{n}{j}$$

bestimmen. Man sieht, dass die untere Grenze der inneren Summe natürlich ist, die obere Grenze aber nicht. Multiplizieren wir den Binomialkoeffizienten mit dem Bruch

$$\frac{\Gamma(i-j+1+\delta)}{\Gamma(i-j+1)},$$

so verschwindet der Summand für $j > i$. Die untere Grenze von i ist damit auch natürlich geworden, da der Summand für negatives i ebenfalls 0 wird (für $j \geq 0$ und $i < 0$ gilt nämlich $i - j + 1 \leq 0$). Es bleibt die obere Grenze von i übrig. Wir multiplizieren den Summanden nochmals mit $\Gamma(n-i+1+\delta)/\Gamma(n-i+1)$ und erhalten

$$F_\delta(n,i,j) = \frac{\Gamma(n+1)\Gamma(i-j+1+\delta)\Gamma(n-i+1+\delta)}{\Gamma(j+1)\Gamma(n-j+1)\Gamma(i-j+1)\Gamma(n-i+1)}.$$

Wie gewohnt verwenden wir `multsum`

```
> F[delta]:=GAMMA(n+1)*GAMMA(i-j+1+delta)*GAMMA(n-i+1+delta)/
>           (GAMMA(j+1)*GAMMA(n-j+1)*GAMMA(i-j+1)*GAMMA(n-i+1));
```

$$F_\delta := \frac{\Gamma(n+1)\,\Gamma(i-j+1+\delta)\,\Gamma(n-i+1+\delta)}{\Gamma(j+1)\,\Gamma(n-j+1)\,\Gamma(i-j+1)\,\Gamma(n-i+1)}$$

```
> subs(delta=0,sum_recursion(F[delta],n,[i,j]));
```

$$-3\,\mathrm{S}(n-1)\,n+\mathrm{S}(n)\,n+(2\,n-2)\,\mathrm{S}(n-2)=0$$

```
> check_rec(%,(n+2)*2^(n-1));
```

true

Aus der letzten Zeile können wir entnehmen, dass $S(n) = (n+2)2^{n-1}$ ist, denn auch die Anfangswerte stimmen überein.

Untersuchen wir auf gleichem Wege die nächste Summe

Beispiel 5.8 Sei

$$S(n) = \sum_i \sum_{j\le i} \frac{i-j}{n+1}\binom{n+1}{i}\binom{n+1}{j}$$

für alle $n \in \mathbb{N}_0$. Wir möchten zeigen, dass

$$S(n) = \binom{2n+1}{n}$$

gilt. Wegen $j \le i$ multiplizieren wir den Summanden mit $\Gamma(i-j+1+\delta)/\Gamma(i-j+1)$ und erhalten

```
> F[delta]:=(i-j)/(n+1)*binomial(n+1,i)*binomial(n+1,j)*
>           GAMMA(i-j+1+delta)/GAMMA(i-j+1);
```

$$F_\delta := \frac{(i-j)\,\mathrm{binomial}(n+1,\,i)\,\mathrm{binomial}(n+1,\,j)\,\Gamma(i-j+1+\delta)}{(n+1)\,\Gamma(i-j+1)}$$

```
> subs(delta=0,sum_recursion(F[delta],n,[i,j]));
```

$$(-2\,n+8\,n^2)\,\mathrm{S}(n-1)-8\,(n-1)\,(-1+2\,n)\,\mathrm{S}(n-2)-(n+1)\,n\,\mathrm{S}(n)=0$$

```
> check_rec(%,binomial(2*n+1,n));
```

true

Die Anfangswerte $S(0) = 1$ und $S(1) = 3$ komplettieren den Beweis.

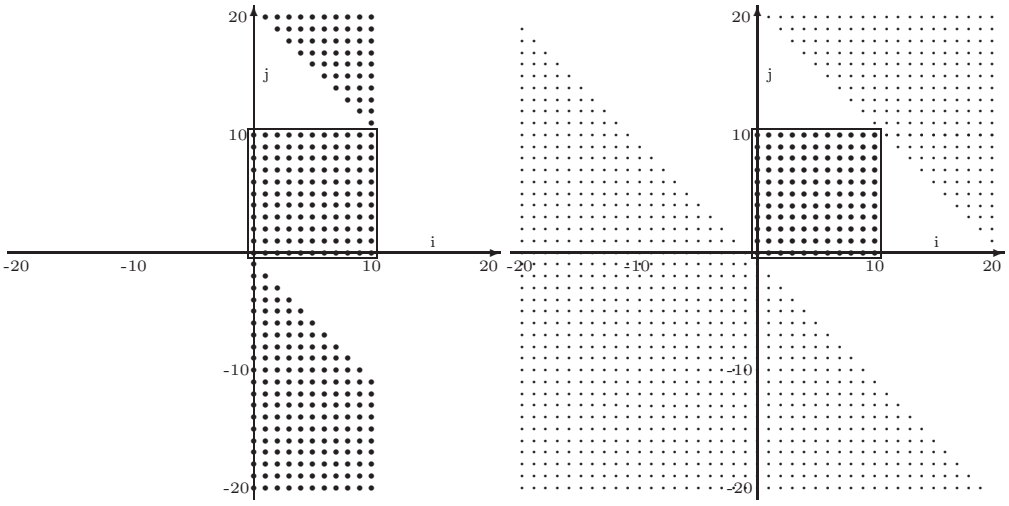
Beispiel 5.9 Die folgende Identität wurde von Andrews und Paule bewiesen

$$\sum_{i=0}^{n}\sum_{j=0}^{n}\binom{i+j}{j}^2\binom{4n-2i-2j}{2n-2i} = (2n+1)\binom{2n}{n}^2, \quad \forall\, n \in \mathbb{N}_0.$$

Verschaffen wir uns zunächst einen Überblick über den Träger des Summanden F der Doppelsumme $S(n)$. Dazu wählen wir $n = 10$ und verwenden die Prozedur `draw_support`, die einige (i,j) aus $T_F(n) = \{(i,j) \in \mathbb{Z}^2 \mid F(n,i,j) \neq 0\}$ als Punktmenge veranschaulicht. Den Summationsbereich der Variablen i und j legen wir auf das ganzzahlige Intervall $[-20..20]$ fest. In einem weiteren Plot stellen wir einen Teil des Trägers des zugehörigen zulässigen Terms dar und erhalten[5]

[5] die Plots von `draw_support` sind den abgebildeten Grafiken ähnlich. Die dünn eingezeichneten Punkte, sind die Paare, für die der zulässige Term nicht wohldefiniert ist. Der gewünschte Summationsbereich liegt innerhalb des Quadrates.

```
> F:=binomial(i+j,i)^2*binomial(4*n-2*i-2*j,2*n-2*i):
> draw_support(F,n,[i,j],20,10);
> draw_support(simpcomb(F),n,[i,j],20,10);
```



Die linke Grafik lässt vermuten, dass F keinen endlichen Träger besitzt. Ein scharfer Blick auf den Ausdruck bestätigt dies: Es gilt z.B.

$$\binom{n+j}{n}^2 \binom{2n-2j}{0} \neq 0$$

für festes n, $i = n$ und für alle $j > n$. Der zulässige Term hat hingegen einen endlichen Träger, aber es existiert kein hinreichend großer Bereich von Nullen um den Träger, so dass man zu keiner holonomen Rekursion für die Summe gelangt. Mittels Einführung von

$$F_\delta(n,i,j) = \frac{\Gamma(i+j+1+\delta)^2\Gamma(4n-2i-2j+1+\delta)}{\Gamma(i+1)^2\Gamma(j+1)^2\Gamma(2n-2i+1)\Gamma(2n-2j+1)}$$

würden wir dieses Problem in den Griff bekommen und natürliche Grenzen erhalten. Allerdings finden unsere Prozeduren zu diesem Term leider keine Rekursion, so dass wir einen anderen Weg einschlagen müssen. Eine weitere Möglichkeit wäre die folgende: Man verwendet `find_rec` und bestimmt eine Rekursionsgleichung für $F(n,i,j)$. Aus dieser bildet man eine inhomogene Rekursion für die Summe und versucht die dabei entstehenden Randwerte zu ermitteln. Wiederum schlägt diese Methode fehl, da die Zertifikate von F zu komplex sind, um die Grenzwerte, die auch Summen darstellen, zu berechnen. Wir skizzieren einen erfolgreichen Lösungsweg: Sei

$$R(n,m) = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \sum_{j=0}^{\lfloor \frac{m}{2} \rfloor} \binom{i+j}{i}^2 \binom{m+n-2i-2j}{n-2i}.$$

Wir zeigen, dass

$$R(n,m) = \frac{\Gamma(\lfloor \frac{m+n+1}{2} \rfloor + 1)\Gamma(\lfloor \frac{m+n+2}{2} \rfloor + 1)}{\Gamma(\lfloor \frac{m}{2} \rfloor + 1)\Gamma(\lfloor \frac{m+1}{2} \rfloor + 1)\Gamma(\lfloor \frac{n}{2} \rfloor + 1)\Gamma(\lfloor \frac{n+1}{2} \rfloor + 1)} \tag{5.7}$$

ist. Vorerst bestimmen wir Zertifikate für den Summanden der Doppelsumme, den wir G nennen.

```
> cer:=find_rec(binomial(i+j,i)^2*binomial(n+m-2*i-2*j,n-2*i),
> n,[i,j],1,1,G);
```

$$
\begin{aligned}
cer := \{&(n-1)\,\mathrm{G}(n-1,\,i-1,\,j-1)+(-n-m)\,\mathrm{G}(n-2,\,i-1,\,j-1)\\
&+\Delta_i((n-1)\,\mathrm{G}(n-1,\,i,\,j-1)+(-1-n+i+2\,j-m)\,\mathrm{G}(n-2,\,i,\,j-1)\\
&+i\,\mathrm{G}(n,\,i,\,j)-i\,\mathrm{G}(n,\,i,\,j-1))\\
&+\Delta_j((-2\,j-i)\,\mathrm{G}(n-2,\,i-1,\,j)+(i-1)\,\mathrm{G}(n,\,i-1,\,j))=0,\\
&(2\,n+1+m)\,\mathrm{G}(n-1,\,i-1,\,j-1)+(-n-1)\,\mathrm{G}(n,\,i-1,\,j-1)\\
&+(-n-m)\,\mathrm{G}(n-2,\,i-1,\,j-1)\\
&+\Delta_i((2\,n+1+m)\,\mathrm{G}(n-1,\,i,\,j-1)+(-1-m-n+j)\,\mathrm{G}(n-2,\,i,\,j-1)\\
&+(-n-j)\,\mathrm{G}(n,\,i,\,j-1)+j\,\mathrm{G}(n,\,i,\,j))\\
&+\Delta_j(-j\,\mathrm{G}(n-2,\,i-1,\,j)+j\,\mathrm{G}(n,\,i-1,\,j))=0\}
\end{aligned}
$$

Als nächstes addieren wir die Differenz der beiden Zertifikate zur ersten Zertifikatsrekursion mit `add_rec`. Es ergibt sich

```
> add_cer(add_cer(cer[1],cer[2],-1),cer[1]);
```

$$
\begin{aligned}
&(m+1)\,\mathrm{G}(n-1,\,i,\,j)\\
&+\Delta_i((n+1)\,(1+i)\,\mathrm{G}(n+1,\,1+i,\,j)-(n+1)\,(1+i)\,\mathrm{G}(n+1,\,1+i,\,j+1)\\
&+(-i+m-2\,j\,n-2\,j-1-i\,n-2\,n)\,\mathrm{G}(n-1,\,1+i,\,j)+n\,(-1+j-i)\\
&\mathrm{G}(n,\,1+i,\,j)-n\,(-i+j)\,\mathrm{G}(n,\,1+i,\,j+1)+n\,(2+i+j)\,\mathrm{G}(n-2,\,1+i,\,j))\\
&+\Delta_j((n+1)\,(3+i+2\,j)\,\mathrm{G}(n-1,\,i,\,j+1)-i\,(n+1)\,\mathrm{G}(n+1,\,i,\,j+1)\\
&-n\,(2+i+j)\,\mathrm{G}(n-2,\,i,\,j+1)-n\,(j-i+1)\,\mathrm{G}(n,\,i,\,j+1))=0
\end{aligned}
$$

Wir erhalten also eine diskrete Stammfunktion für $G(n-1,i,j)$. Durch Summation von $-\infty$ bis ∞ bzgl. i und von 0 bis $\lfloor\frac{m}{2}\rfloor$ bzgl. j bekommen wir eine inhomogene Rekursion für $R(n,m)$. Sie lautet $(m+1)R(n-1,m)=\sum_i r_2-\sum_i r_1$ mit

$$
\begin{aligned}
r_2 = &-(n+1)(3+i+2\lfloor\tfrac{m}{2}\rfloor)G(n-1,i,\lfloor\tfrac{m}{2}\rfloor+1)+(n+1)iG(n+1,i,\lfloor\tfrac{m}{2}\rfloor+1)\\
&+n(2+i+\lfloor\tfrac{m}{2}\rfloor)G(n-2,i,\lfloor\tfrac{m}{2}\rfloor+1)+n(\lfloor\tfrac{m}{2}\rfloor-i+1)G(n,i,\lfloor\tfrac{m}{2}\rfloor+1)
\end{aligned}
$$

und

$$
\begin{aligned}
r_1 = &-(n+1)(i+1)G(n-1,i,0)+(n+1)iG(n+1,i,0)\\
&+n(i+1)G(n-2,i,0)-niG(n,i,0).
\end{aligned}
$$

Aus den folgenden Zeilen kann man entnehmen, dass $(n+1)(i+1)G(n-1,i,0)=(n+1)(i+1)G(n+1,i+1,0)$ und $n(i+1)G(n-2,i,0)=n(i+1)G(n,i+1,0)$ gilt.

```
> simpcomb(subs({n=n-1,j=0},G)/subs({n=n+1,i=i+1,j=0},G));
```

$$1$$

```
> simpcomb(subs({n=n-2,j=0},G)/subs({i=i+1,j=0},G));
```

$$1$$

Folglich verschwindet der Randwert $\sum_i r_1$. Untersuchen wir nun $\sum_i r_2$. Es sind vier Fälle zu unterscheiden: jede Kombinationsmöglichkeit von n gerade/ungerade und m gerade/ungerade. Wir führen die Berechnung von $\sum_i r_2$ exemplarisch für n gerade und m gerade aus. Man sieht leicht ein, dass

$$G(n-k,i,\lfloor\tfrac{m}{2}\rfloor+1)=\binom{i+\lfloor\frac{m}{2}\rfloor+1}{i}^2\binom{n-k-2i-2}{n-k-2i}$$

nur dann ungleich 0 ist, wenn $n-k-2i=0$ oder $n-k-2i=1$ ist. D.h. die Summe $\sum_i r_2$ besteht aus genau vier nichtverschwindenden Summanden, da n gerade ist.

```
> r2[1]:=subs(i=(n-2)/2,-(n+1)*(3+i+m)*subs({n=n-1,j=m/2+1},G)):
> r2[2]:=subs(i=n/2,(n+1)*i*subs({n=n+1,j=m/2+1},G)):
> r2[3]:=subs(i=(n-2)/2,n*(2+i+m/2)*subs({n=n-2,j=m/2+1},G)):
> r2[4]:=subs(i=n/2,n*(m/2-i+1)*subs(j=m/2+1,G)):
> simpcomb(expand((1/(m+1))*sum(r2[k],k=1..4)));
```

$$\frac{1}{2}\,\frac{n\,\Gamma\left(\frac{n}{2}+\frac{m}{2}+1\right)^2}{\Gamma\left(\frac{n}{2}+1\right)^2\,\Gamma\left(\frac{m}{2}+1\right)^2}$$

Die Ausgabe entspricht der geschlossenen Form von $R(n-1,m)$ für gerades n und m. Alle anderen Fälle arbeitet man analog ab und erhält zusammenfassend

$$R(n,m)=\frac{\Gamma(\lfloor\frac{m+n+1}{2}\rfloor+1)\Gamma(\lceil\frac{m+n+1}{2}\rceil+1)}{\Gamma(\lfloor\frac{m}{2}\rfloor+1)\Gamma(\lceil\frac{m}{2}\rceil+1)\Gamma(\lfloor\frac{n}{2}\rfloor+1)\Gamma(\lceil\frac{n}{2}\rceil+1)},$$

was dem Ausdruck (5.7) gleicht. Die Identität $S(n)=R(2n,2n)$ liefert letztendlich den Beweis für die Summe $S(n)$ von Andrews-Paule.

Literaturverzeichnis

[Abr64] M. Abramowitz, I. A. Stegun - *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* - National Bureau of Standards, Applied Mathematics Series 55, 775, Juni 1964 76

[And99] G. E. Andrews, R. Askey, R. Roy - *Special Functions - Encyclopedia of Mathematics and Its Applications* - The University Press, Cambridge, 318, 1999 76

[Koe98] W. Koepf - *Hypergeometric Summation - An Algorithmic Approach to Summation and Special Function Identities* - Vieweg, Advanced Lectures in Mathematics, April 1998 53

[Lyo02] R. Lyons, P. Paule, A. Riese - *A Computer Proof of a Series Evaluation in Terms of Harmonic Numbers* - April 2002 77

[Rie01] A. Riese - *Fine-Tuning Zeilberger's Algorithm: The Method of Automatic Filtering and Creative Substituting* - „Symbolic Computation, Number Theory, Special Functions and Combinatorics", Development in Mathematics, Vol. 4, 243-254, 2001 51

[Str94] V. Strehl - *Binomial Identities - Combinatorial and Algorithmic Aspects* - University of Erlangen-Nürnberg, 1994 74

[Weg97] K. Wegschaider - *Computer Generated Proofs of Binomial Multi-Sum Identities* - Diplomarbeit, J. Kepler Universität Linz, Mai 1997 5

[Wil92] H. S. Wilf, D. Zeilberger - *An algorithmic proof theory for hypergeometric multisum/integral identities* - Journal of Computational and Applied Math., 32, 321-368, 1990 5

Ich versichere hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Torsten Sprenger