Universität Regensburg
Naturwissenschaftliche Fakultät I
- Mathematik -

Algebraische Probleme in der Theorie der Warteschlangenmodelle mit Kontrolle

Diplomarbeit von
Arndt Wills

vorgelegt bei
PD Dr. Martin Kreuzer

Regensburg, im Oktober 2002

Inhaltsverzeichnis

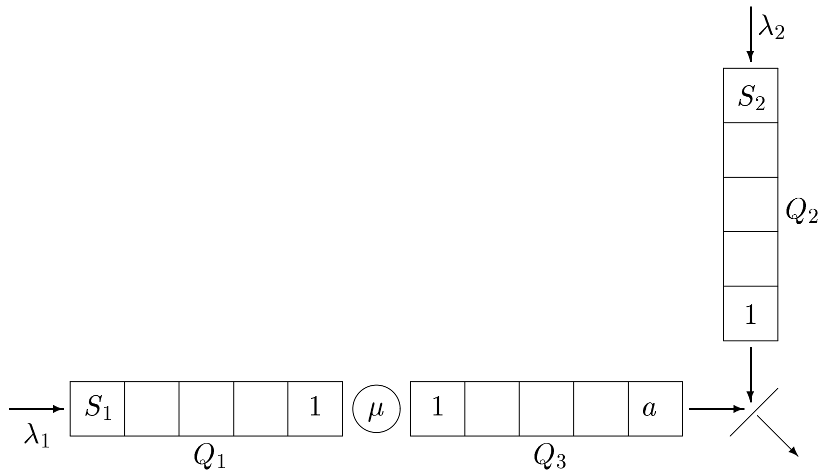
1 Einleitung

Warteschlangenmodelle mit Kontrolle spielen eine sehr wichtige Rolle in Telekomunikationssystemen und Fertigungssystemen. Man benötigt sie, um die Auswirkungen von Ausfällen, Wartungsarbeiten oder Urlaubszeiten zu untersuchen.
Diese Arbeit befasst sich mit einer Warteschlange, die von einem Nachfrageprozess kontrolliert wird. Es wird davon ausgegangen, dass die Werksstücke im Zulauf gemäß eines Poisson-Prozesses ankommen und die weitere Verarbeitung durch ein markovsches Einzelbediensystem erfolgt. Fertige Teile werden in einem endlichen Speicher abgelegt. Ist der Speicher voll, so stoppt das System bis wieder Platz frei wird. Die ankommende Nachfrage wird sofort befriedigt, falls der Speicher nicht leer ist, andernfalls wird sie in eine Warteschlange gestellt.
Der genaue Aufbau dieses Systems wird im zweiten Abschnitt dargestellt. Außerdem wird ein mathematische Modell dieses markovschen Einzelbediensystems entwickelt, aus dem dann das grundlegende Gleichungssystem hergeleitet werden kann, welches die Zustandswahrscheinlichkeiten des Systems beschreibt.
Der dritte Abschnitt beschäftigt sich mit der Übersetzung des Gleichungssystems in die Sprache der Computeralgebra. Es entsteht dabei ein lineares Gleichungssystem aus $(s+1)^2$ Gleichungen mit ebenso vielen Unbekannten über einem Polynomring P in 3 Unbestimmten. Unter anderem wird nun gezeigt, dass der Lösungsraum über dem Quotientenköper von P eindimensional ist. Durch eine zusätzliche Normierung erhält man eine eindeutige Lösung des Systems. Verzichtet man auf diese Normierung, so kann stets eine Lösung über dem Polynomring gefunden werden. Der Rest des Abschnittes beschäftigt sich mit der aus der Cramerschen Regel folgenden Lösung. Das zentrale Ergebnis des theoretischen Teils dieser Arbeit ist eine allgemeine Formel für die Leitterme und Leitkoeffizienten der Polynome des Lösungsvektors bezüglich der graduiert-umgekehrt-lexikographischen Termordnung `DegRevLex`.
Der zweite Teil der Arbeit beschäftigt sich mit der effektiven Lösung des Gleichungssystems. Schon H. Gold hat sich mit diesem Problem befasst (vgl. [G]). Nach seinen Angaben war eine Lösungsfindung für $s > 4$ mit herkömlichen Mathematikprogrammen nicht mehr möglich. In dieser Arbeit wird nun versucht, dieses Problem mit Hilfe der Computeralgebra zu lösen. Dazu werden im vierten Abschnitt einige Lösungsalgorithmen vorgestellt. Hierbei handelt es sich um einen leicht modifizierten Gauss-Algorithmus, zwei Bareiss-Verfahren und einen auf Syzygienberechnung basierenden Algorithmus. Letzterer verwendet eine in CoCoA eingebaute Funktion, wodurch die volle Macht dieses Computeralgebrasystems ausgenutzt werden kann. Es wird gezeigt, wie man das Gleichungssystem in eine kompaktere Form überführen kann, um den Wirkungsgrad dieser Algorithmen zu erhöhen. Eine

Dehomogenisierung des Problems dient im weiteren Verlauf dem gleichen Zweck. Die Krönung dieses Abschnitts bildet eine Methode, die die Ergebnisse von Berechnungen über endlichen Körpern mit Hilfe des chinesischen Restsatzes in eine Lösung über $\mathbb{Z}$ überführt. Ziel dieses Abschnittes ist, einen möglichst effizienten und schnellen Algorithmus zu finden.
Der fünfte Abschnitt behandelt die Implementierung der im vorangehenden Abschnitt vorgestellten Algorithmen mit dem Computeralgebrasystem CoCoA, welches kostenlos über `http://cocoa.dima.unige.it/` bezogen werden kann. Es werden alle wichtigen Funktionen der beiliegenden CoCoA-Programme beschrieben. Der Schwerpunkt liegt dabei nicht in der Programmiertechnik, sondern darin, mit welchen Argumenten und zu welchem Zweck der Anwender die Funktionen benutzen kann. Insbesondere wird auch das Paket `chinese.pkg` genau dokumentiert, welches die modulare Methode des vorangegangenen Abschnitts implementiert. Dieses Paket wurde so flexibel gestaltet, dass es sich auch für ganz andere Problemstellungen als nützlich erweisen kann.
Im letzten Abschnitt werden die implementierten Algorithmen dann angewendet. Dabei wird die Bearbeitungsdauer der unterschiedlichen Methoden verglichen und bewertet. Ausgehend von der langsamsten Methode wird versucht, alle Schritte der Beschleunigung nachvollziehbar darzustellen und den quantitativen Nutzen jeder Modifikation mit Zahlen zu belegen. Dazu werden die Lösungen für verschiedene Fälle des Gleichungssystems effektiv berechnet. Es zeigt sich, dass der auf Syzygienberechnung basierende Algorithmus im Polynomring in 2 Unbestimmten, mit der Modularen Methode ausgeführt, bei der Berechnung der komprimierten Matrix am effektivsten ist. Auf diese Weise kann man die Lösung für den Fall $s = 10$ bereits in etwas über 5 Stunden berechnen. Da die Bearbeitungszeit für größere s im Vergleich zum nächstkleineren Fall stets etwa $4,5$-mal so lang war, ist zu erwarten, dass eine effektive Berechnung auf einem handelsüblichen Computer noch bis zum Fall $s = 12$ in etwa einer Woche durchführbar ist. In der Praxis sind die Fälle bis $s = 25$ relevant. Dieser Arbeit liegen Dateien mit den Lösungen für die Fälle s bis einschließlich 10 bei. Da der Lösungsvektor für $s = 10$ schon eine Textdatei mit fast 23MB Größe darstellt, wird auf eine Berechnung weiterer Fälle verzichtet. Wie weit die Berechnung mit dem besten der vorgestellten Algorithmen durch das Computeralgebrasystem CoCoA tatsächlich durchführbar ist, konnte mangels Rechenkapazität leider nicht ermittelt werden. Gerade die modulare Methode eignet sich ausgezeichnet für eine parallele Berechnung, man kann also mit Hilfe von Rechnerclustern oder Großrechnern eventuell sogar die meisten der verbleibenden relevanten Fälle berechnen.

2 Vorstellung des behandelten Problems

Im Folgenden soll ein markovsches Einzelbediensystem betrachtet werden. Es besteht aus einem Input-Bereich, einem Speicherbereich und einem Nachfragebereich. Der Inputbereich sei eine Warteschlange Q_1 mit S_1 Plätzen, die durch einen Poisson-Prozess der Rate λ_1 befüllt wird. Ein Server bearbeitet den Input aus Q_1 und befüllt den endlichen Speicher Q_3 der Länge a. Auch dieser Vorgang sei ein Poisson-Prozess der Rate μ. Der Nachfragebereich ist ebenfalls eine Warteschlange Q_2 der Länge S_2, die durch einen Poisson-Prozess der Rate λ_2 befüllt wird.



Das System sei so gesteuert, dass der Server nur solange einen Abarbeitungsschritt startet, wie ein Platz in Q_3 frei ist. Weiter werde eine Nachfrage sofort erfüllt, solange Q_3 nicht leer ist, andernfalls werde sie in Q_2 gestellt. Das bedeutet, dass zu jedem Zeitpunkt mindestens eine der Warteschlangen Q_2 oder Q_3 leer ist. Interpretiert man den Speicher Q_3 als "negative Nachfrage", so kann man das System Q_2,Q_3 durch eine Warteschlange Q_2' ersetzen, welche die Länge $S_2' := S_2 + a$ hat, zu Beginn auf a gefüllt ist, durch die Nachfrage befüllt und durch den Server geleert wird.
Im Folgenden beschränken wir uns auf den symmetrischen Fall:

$$S_1 = S_2' = S_2 + a =: s$$

Der Zustand des Systems ist durch die Füllmenge in Q_1 und Q_2' eindeutig bestimmt. Man kann ihn also durch ein Paar (x, y) beschreiben, wobei

$$x = \begin{cases} \#(\text{Jobs in } Q_1) + 1 & \text{falls der Server im Betrieb ist} \\ 0 & \text{sonst} \end{cases}$$

$$y = \begin{cases} \#(\text{Jobs in } Q_2) + a & \text{falls } Q_2 \text{ nicht leer} \\ \#(\text{freien Plätze in } Q_3) & \text{sonst} \end{cases}$$

Das folgende Diagramm beschreibt nun das Fließgleichgewicht im stationären Zustand:

$$\begin{tikzcd}
(0,s) \arrow[r] & (1,s) \arrow[r] \arrow[dl] & \cdots \arrow[r] & (s,s) \arrow[dl] \\
\vdots \arrow[u] & \vdots \arrow[u] \arrow[dl] & \iddots \arrow[dl] & \vdots \arrow[u] \\
(0,1) \arrow[u] \arrow[r] & (1,1) \arrow[u] \arrow[r] \arrow[dl, "\mu"] & \cdots \arrow[r] \arrow[dl] & (s,1) \arrow[u] \arrow[dl] \\
(0,0) \arrow[u, "\lambda_2"] \arrow[r, "\lambda_1"'] & (1,0) \arrow[u] \arrow[r] & \cdots \arrow[r] & (s,0) \arrow[u]
\end{tikzcd}$$

Im Gleichgewichtszustand muss an jedem Knoten die Summe der Zuflüsse gleich der Summe der Abflüsse sein. Bezeichnet man für $i, j = 1, \ldots, s$ mit $p(i,j)$ die Wahrscheinlichkeit, im Zustand (i,j) zu sein, so ergibt sich folgende Gleichung:

$$\begin{aligned} p(i,j)(\chi_{i<s}\lambda_1 + \chi_{j<s}\lambda_2 + \chi_{i>0}\chi_{j>0}\mu) = {} & p(i+1,j+1)\chi_{i+1\leq s}\chi_{j+1\leq s}\mu \\ & +p(i-1,j)\chi_{i-1\geq 0}\lambda_1 \\ & +p(i,j-1)\chi_{j-1\geq 0}\lambda_2 \end{aligned} \tag{1}$$

Hierbei sei $\chi_A = \begin{cases} 0 & A \text{ unwahr} \\ 1 & A \text{ wahr} \end{cases}$

Daraus ergeben sich 9 Fälle, nämlich die 4 Ecken, die 4 Ränder und das Innere.

3 Algebraische Untersuchung des Problems

Im Folgenden sei nun $x := \lambda_1$, $y := \lambda_2$ und $z := \mu$.
Sei K ein Körper und $P := K[x,y,z]$. Weiter sei $Q := P[x_{i,j}]_{i,j=0,\ldots,s}$ der Polynomring in $(s+1)^2$ Unbestimmten über P. Identifiziere $x_{i,j} = p(i,j)$ für $i,j = 0,\ldots,s$. Man erhält $(s+1)^2$ Gleichungen $f_{i,j}$ in ebenso vielen Unbestimmten über P aus (1).
Die 9 Fälle der Grundgleichung (1) sehen dann wie folgt aus:

$$\begin{array}{llllll}
f_{0,0} = x_{0,0} \cdot & (x+y) & - x_{1,1} \cdot z & & & = 0 \\
f_{i,0} = x_{i,0} \cdot & (x+y) & - x_{i+1,1} \cdot z & - x_{i-1,0} \cdot x & & = 0 \\
f_{s,0} = x_{s,0} \cdot & y & & - x_{s-1,0} \cdot x & & = 0 \\
f_{0,j} = x_{0,j} \cdot & (x+y) & - x_{1,j+1} \cdot z & & - x_{0,j-1} \cdot y & = 0 \\
f_{i,j} = x_{i,j} \cdot & (x+y+z) & - x_{i+1,j+1} \cdot z & - x_{i-1,j} \cdot x & - x_{i,j-1} \cdot y & = 0 \\
f_{s,j} = x_{s,j} \cdot & (y+z) & & - x_{s-1,j} \cdot x & - x_{s,j-1} \cdot y & = 0 \\
f_{0,s} = x_{0,s} \cdot & x & & & - x_{0,s-1} \cdot y & = 0 \\
f_{i,s} = x_{i,s} \cdot & (x+z) & & - x_{i-1,s} \cdot x & - x_{i,s-1} \cdot y & = 0 \\
f_{s,s} = x_{s,s} \cdot & z & & - x_{s-1,s} \cdot x & - x_{s,s-1} \cdot y & = 0
\end{array} \tag{2}$$

Hierbei sei $0 < i < s$ und $0 < j < s$.

Um das Gleichungssystem weiterzubehandeln, soll es im Folgenden als Matrix geschrieben werden. Für die theoretische Behandlung ist es zunächst zweckmäßig die Anordnung der Unbestimmten zu ändern. Die dazu benötigte Abbildung liefert der folgende Satz:

Satz 3.1. Sei $I_1 := \{(i,j)|i = 0,\ldots,s \text{ und } j = 0,\ldots,s\}$ sowie $I_2 := \{(k,l_k)|k = 0,\ldots,s \text{ und } l_k = 0,\ldots,2k\}$ und $I_3 := \{0,\ldots,s^2+2s\}$.

a) Die Abbildung $\varphi : I_1 \longrightarrow I_2$, definiert durch

$$\varphi(i,j) := (s - \min(i,j), s - \min(i,j) + i - j)$$

ist bijektiv mit Umkehrabbildung $\varphi^{-1} : I_2 \longrightarrow I_1$, definiert durch

$$\varphi^{-1}(k,l_k) := (s - \min(k,l_k) + l_k - k, s - \min(k,l_k)).$$

b) Die Abbildung $\psi : I_2 \longrightarrow I_3$, definiert durch

$$\psi(k,l_k) := k^2 + l_k$$

ist bijektiv.

Beweis. zu a)
Um die Wohldefiniertheit von φ bzw. φ^{-1} zu zeigen, muss nachgeprüft werden, ob die Funktionen für alle Werte der Definitionsmenge Ergebnisse in der Zielmenge liefern.
Sei im Folgenden $(i,j) \in I_1$, also $0 \le i \le s$ und $0 \le j \le s$. Bezeichne mit $\varphi_1(i,j)$ die erste Komponente von $\varphi(i,j)$ und mit $\varphi_2(i,j)$ die zweite Komponente. Da $i,j \in \{0,\ldots,s\}$, ist auch $\min(i,j) \in \{0,\ldots,s\}$ und somit

$$\varphi_1(i,j) = s - \min(i,j) \in \{0,\ldots,s\}.$$

Aus $s - j \ge 0$ und $i - \min(i,j) \ge 0$ folgt

$$\varphi_2(i,j) = s - \min(i,j) + i - j \ge 0.$$

Aus $s - i \ge 0$ und $j - \min(i,j) \ge 0$ folgt $s - \min(i,j) \ge i - j$ und damit

$$\varphi_2(i,j) = s - \min(i,j) + i - j \le 2 \cdot (s - \min(i,j)) = 2 \cdot \varphi_1(i,j).$$

Insgesamt folgt also

$$\varphi(i,j) \in I_2 \text{ für alle } (i,j) \in I_1.$$

Sei nun $(k,l_k) \in I_2$, also $0 \le k \le s$ und $0 \le l_k \le 2k$. Bezeichne mit $\varphi_1^{-1}(k,l_k)$ die erste Komponente von $\varphi^{-1}(k,l_k)$ und mit $\varphi_2^{-1}(k,l_k)$ die zweite Komponente.

Da $0 \leq k \leq s$ und $l_k \geq 0$, ist $0 \leq \min(k, l_k) \leq s$ und somit

$$\varphi_2^{-1}(k, l_k) = s - \min(k, l_k) \in \{0, \ldots, s\}.$$

Aus $l_k - \min(k, l_k) \geq 0$ und $s - k \geq 0$ folgt

$$\varphi_1^{-1}(k, l_k) = s - \min(k, l_k) + l_k - k \geq 0.$$

Aus $l_k \leq 2k$ folgt $l_k - k \leq k$, andererseits gilt auch $l_k - k \leq l_k$, also $l_k - k \leq \min(k, l_k)$. Hieraus ergibt sich

$$\varphi_1^{-1}(k, l_k) = s - \min(k, l_k) + l_k - k \leq s.$$

Insgesamt gilt also

$$\varphi^{-1}(k, l_k) \in I_1 \text{ für alle } (k, l_k) \in I_2.$$

Für den Nachweis der Bijektivität genügt es nun nachzurechnen, dass $\varphi^{-1}(\varphi(i,j)) = (i,j)$ und $\varphi(\varphi^{-1}(k, l_k)) = (k, l_k)$ ist für alle $(i,j) \in I_1$ und $(k, l_k) \in I_2$.
1.Fall: $j \geq i$, also $i = \min(i,j)$

$$\varphi(i,j) = (s-i, s-j)$$

In diesem Fall ist $s - j \leq s - i$, d.h. $\min(s-i, s-j) = s - j$ und somit

$$\varphi^{-1}(s-i, s-j) = (s - (s-j) + (s-j) - (s-i), s - (s-j)) = (i,j).$$

2.Fall: $j < i$, also $j = \min(i,j)$

$$\varphi(i,j) = (s-j, s+i-2j)$$

Aus $0 < i - j$ folgt $s - j < s + i - 2j$, d.h. $\min(s-j, s+i-2j) = s - j$. Damit ergibt sich

$$\varphi^{-1}(s-j, s+i-2j) = (s - (s-j) + (s+i-2j) - (s-j), s - (s-j)) = (i,j).$$

Insgesamt gilt also

$$\varphi^{-1}(\varphi(i,j)) = (i,j) \text{ für alle } (i,j) \in I_1.$$

Analog rechnet man $\varphi(\varphi^{-1}(k, l_k)) = (k, l_k)$ für alle $(k, l_k) \in I_2$ nach.
zu b)
Für jedes $k = 0, \ldots, s$ und $l_k = 0, \ldots, 2k$ gilt:

$$k^2 \leq k^2 + l_k \leq k^2 + 2k < k^2 + 2k + 1 = (k+1)^2$$

Damit ist gezeigt, dass die Abbildung ψ injektiv ist. Die Bijektivität folgt sofort aus $\#I_2 = \#I_1 = (s+1)^2 = \#I_3$. □

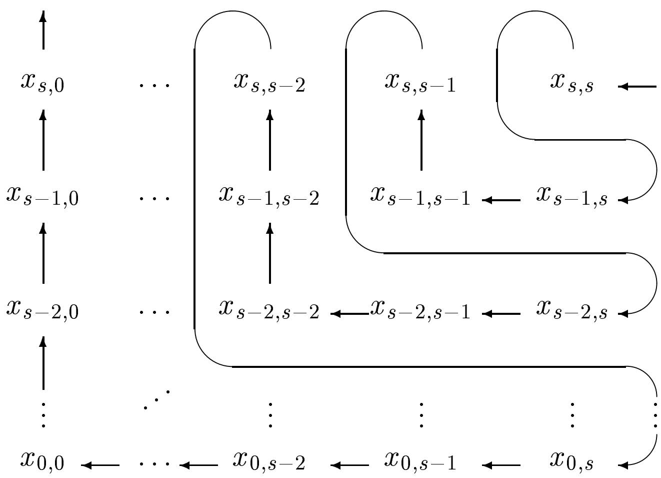
Definition 3.2. Unter den Voraussetzungen von Satz 3.1 setze für alle $i, j = 0, \dots, s$ $x_{i,j}$ an die $\psi(\varphi(i,j))$-te Position eines Vektors $V \in Q^{(s+1)^2}$. Diese Anordnung sei im Folgenden L-Form genannt.
Definiere weiter für $i, j = 0, \dots, s$

$$y_{\varphi(i,j)} := x_{i,j}$$

und für $k = 0, \dots, s$ sowie für $l_k = 0, \dots, 2k$

$$g_{k,l_k} := f_{\varphi^{-1}(k,l_k)}$$

Das folgende Diagramm soll die neue Anordnung veranschaulichen. Folgt man den Pfeilen, so durchläuft man die Unbestimmten genau in L-Form.



Wie man leicht nachrechnet, hat das Gleichungssystem (2) in y_{k,l_k} ausgedrückt ($k = 0, \dots, s$, $l_k = 0, \dots, 2k$) nun folgende Form:

$$\begin{array}{llllll}
g_{0,0} = y_{0,0} \cdot & z & & - y_{1,0} \cdot x & - y_{1,2} \cdot y & = 0 \\
g_{k,0} = y_{k,0} \cdot & (x+z) & & - y_{k+1,0} \cdot x & - y_{k,1} \cdot y & = 0 \\
g_{k,l_k} = y_{k,l_k} \cdot & (x+y+z) & - y_{k-1,l_k-1} \cdot z & - y_{k+1,l_k} \cdot x & - y_{k,l_k+1} \cdot y & = 0 \\
g_{k,k} = y_{k,k} \cdot & (x+y+z) & - y_{k-1,k-1} \cdot z & - y_{k+1,k} \cdot x & - y_{k+1,k+2} \cdot y & = 0 \\
g_{k,l'_k} = y_{k,l'_k} \cdot & (x+y+z) & - y_{k-1,l'_k-1} \cdot z & - y_{k,l'_k-1} \cdot x & - y_{k+1,l'_k+2} \cdot y & = 0 \\
g_{k,2k} = y_{k,2k} \cdot & (y+z) & & - y_{k,2k-1} \cdot x & - y_{k+1,2k+2} \cdot y & = 0 \\
g_{s,0} = y_{s,0} \cdot & x & & & - y_{s,1} \cdot y & = 0 \\
g_{s,l_s} = y_{s,l_s} \cdot & (x+y) & - y_{s-1,l_s-1} \cdot z & & - y_{s,l_s+1} \cdot y & = 0 \\
g_{s,s} = y_{s,s} \cdot & (x+y) & - y_{s-1,s-1} \cdot z & & & = 0 \\
g_{s,l'_s} = y_{s,l'_s} \cdot & (x+y) & - y_{s-1,l'_s-1} \cdot z & - y_{s,l'_s-1} \cdot x & & = 0 \\
g_{s,2s} = y_{s,2s} \cdot & y & & - y_{s,2s-1} \cdot x & & = 0
\end{array} \quad (3)$$

Hierbei sei $0 < k < s$, $0 < l_k < k$, $k < l'_k < 2k$, $0 < l_s < s$ und $s < l'_s < 2s$.

Um die zu diesem Gleichungssystem gehörige Matrix beschreiben zu können, brauchen wir ein paar “Bausteine”, die uns die folgende Definition liefert:

Definition 3.3. Sei K ein Körper und $P := K[x,y,z]$. Für $i \in \mathbb{N}_+$ bezeichne $E_i \in M(i \times i, P)$ die Einheitsmatrix.
Setze $A^{(0)} := \left(z \right) \in M(1 \times 1, P)$, $B^{(0)} := \left(-x \mid 0 \mid -y \right) \in M(1 \times 3, P)$ und $C^{(0)} := \left(\begin{array}{c} 0 \\ \hline -z \\ \hline 0 \end{array} \right) \in M(3 \times 1, P)$.
Sei $A^{(i)} \in M((2i+1) \times (2i+1), P)$ definiert durch:

$$
\begin{aligned}
A^{(i)} := z \cdot E_{2i+1} &+ \left(\begin{array}{c|c} & 0 \\ x \cdot E_{2i} & \vdots \\ & 0 \\ \hline 0 \dots 0 & 0 \end{array} \right) + \left(\begin{array}{cc|c} 0 \quad \dots & 0 & 0 \\ \hline 0 \cdot E_i & 0 & 0 \\ \hline 0 & -x \cdot E_i & \vdots \\ & & 0 \end{array} \right) \\
&+ \left(\begin{array}{c|c} 0 & 0 \dots 0 \\ \hline 0 & \\ \vdots & y \cdot E_{2i} \\ 0 & \end{array} \right) + \left(\begin{array}{c|cc} 0 & -y \cdot E_i & 0 \\ \vdots & & \\ \hline 0 & 0 & 0 \cdot E_i \\ \hline 0 & 0 \quad \dots & 0 \end{array} \right)
\end{aligned}
$$

Setze $B^{(i)} \in M((2i+1) \times (2i+3), P)$ wie folgt:

$$
B^{(i)} := \left(\begin{array}{cc|c|c} -x \cdot E_{i+1} & 0 & 0 & 0 \\ & & \vdots & \vdots \\ \hline 0 & 0 \cdot E_i & 0 & 0 \end{array} \right) + \left(\begin{array}{c|c|c|c} 0 & 0 & 0 \cdot E_i & 0 \\ \vdots & \vdots & & \\ \hline 0 & 0 & 0 & -y \cdot E_{i+1} \end{array} \right)
$$

Schließlich sei

$$
C^{(i)} := \left(\begin{array}{c} 0 \quad \dots \quad 0 \\ \hline \\ -z \cdot E_{2i+1} \\ \\ \hline 0 \quad \dots \quad 0 \end{array} \right) \in M((2i+3) \times (2i+1), P)
$$

und $D^{(i)} := A^{(i)} - z \cdot E_{2i+1} \in M((2i+1) \times (2i+1), P)$

Proposition 3.4. Die Matrix

$$
N^{(s)} := \begin{pmatrix} A^{(0)} & B^{(0)} & & 0 \\ C^{(0)} & \ddots & \ddots & \\ & \ddots & A^{(s-1)} & B^{(s-1)} \\ 0 & & C^{(s-1)} & D^{(s)} \end{pmatrix} \in M((s+1)^2 \times (s+1)^2, P)
$$

beschreibt das Gleichungssystem (3), wenn man die Zeilen und Spalten wie in Definition 3.2 beschrieben L-förmig anordnet.

Beweis. Man prüft komponentenweise leicht nach, dass für $k = 0, \dots, s-1$
$A^{(k)}_{i,j}$ der Koeffizient von $y_{k,j}$ in $g_{k,i}$ ($i = 0, \dots, 2k$, $j = 0, \dots, 2k$),
$B^{(k)}_{i,j}$ der Koeffizient von $y_{k+1,j}$ in $g_{k,i}$ ($i = 0, \dots, 2k$, $j = 0, \dots, 2k+2$),
$C^{(k)}_{i,j}$ der Koeffizient von $y_{k,j}$ in $g_{k+1,i}$ ($i = 0, \dots, 2k+2, j = 0, \dots, 2k$) und
$D^{(s)}_{i,j}$ der Koeffizient von $y_{s,j}$ in $g_{s,i}$ ($i = 0, \dots, 2s$, $j = 0, \dots, 2s$) ist. □

Zur Veranschaulichung sei für den Fall $s = 2$ das Gleichungssystem und die zugehörige Matrix angegeben:

Beispiel 3.5. Im Fall $s = 2$ hat das Gleichungssystem (3) die folgende Gestalt:

$$
\begin{array}{lllllll}
g_{0,0} = y_{0,0} \cdot & z & & - y_{1,0} \cdot x & - y_{1,2} \cdot y & = 0 \\
g_{1,0} = y_{1,0} \cdot & (x+z) & & - y_{2,0} \cdot x & - y_{1,1} \cdot y & = 0 \\
g_{1,1} = y_{1,1} \cdot & (x+y+z) & - y_{0,0} \cdot z & - y_{2,1} \cdot x & - y_{2,3} \cdot y & = 0 \\
g_{1,2} = y_{1,2} \cdot & (y+z) & & - y_{1,1} \cdot x & - y_{2,4} \cdot y & = 0 \\
g_{2,0} = y_{2,0} \cdot & x & & & - y_{2,1} \cdot y & = 0 \\
g_{2,1} = y_{2,1} \cdot & (x+y) & - y_{1,0} \cdot z & & - y_{2,2} \cdot y & = 0 \\
g_{2,2} = y_{2,2} \cdot & (x+y) & - y_{1,1} \cdot z & & & = 0 \\
g_{2,3} = y_{2,3} \cdot & (x+y) & - y_{1,2} \cdot z & - y_{2,2} \cdot x & & = 0 \\
g_{2,4} = y_{2,4} \cdot & y & & - y_{2,3} \cdot x & & = 0
\end{array}
$$

Die zugehörige Matrix sieht dann wie folgt aus:

$$
\left(\begin{array}{c|ccc|ccccc}
z & -x & 0 & -y & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & x+z & -y & 0 & -x & 0 & 0 & 0 & 0 \\
-z & 0 & x+y+z & 0 & 0 & -x & 0 & -y & 0 \\
0 & 0 & -x & y+z & 0 & 0 & 0 & 0 & -y \\
\hline
0 & 0 & 0 & 0 & x & -y & 0 & 0 & 0 \\
0 & -z & 0 & 0 & 0 & x+y & -y & 0 & 0 \\
0 & 0 & -z & 0 & 0 & 0 & x+y & 0 & 0 \\
0 & 0 & 0 & -z & 0 & 0 & -x & x+y & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -x & y
\end{array}\right)
$$

Durch die eingezogenen Linien erkennt man sehr schön die Teilmatrizen $A^{(0)}, B^{(0)}, C^{(0)}, A^{(1)}, B^{(1)}, C^{(1)}$ und $D^{(2)}$.

Da im weiteren Verlauf die Determinanten der Matrizen $D^{(i)}$ für $i = \in \mathbb{N}$ benötigt werden, sollen diese zunächst bestimmt werden.

Hilfssatz 3.6. Für $i \in \mathbb{N}_+$ gilt:

$$\det D^{(i)} = xy(x+y)^{2i-1}$$

Beweis. Trivialerweise gilt:

$$\det D^{(1)} = \det \begin{pmatrix} x & -y & 0 \\ 0 & x+y & 0 \\ 0 & -x & y \end{pmatrix} = x \cdot (x+y) \cdot y$$

Für $i \geq 2$ kann aus der Definition gefolgert werden:

$$\begin{aligned} D^{(i)} = {} & \left(\begin{array}{c|c} x \cdot E_{2i} & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \ldots 0 & 0 \end{array}\right) + \left(\begin{array}{c|c|c} \multicolumn{2}{c|}{0 \quad \ldots \quad 0} & 0 \\ \hline 0 \cdot E_i & 0 & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \cline{1-2} 0 & -x \cdot E_i & \end{array}\right) \\ & + \left(\begin{array}{c|c} 0 & 0 \ldots 0 \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & y \cdot E_{2i} \end{array}\right) + \left(\begin{array}{c|c|c} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & -y \cdot E_i & 0 \\ \cline{2-3} & 0 & 0 \cdot E_i \\ \hline 0 & \multicolumn{2}{c}{0 \quad \ldots \quad 0} \end{array}\right) \end{aligned}$$

$$= \left(\begin{array}{c|c} \left(\begin{array}{c|c} x & 0 \quad \ldots \quad 0 \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & (x+y)E_{i-1} \end{array}\right) - \left(\begin{array}{c|c} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & y \cdot E_{i-1} \\ \hline 0 & 0 \ \ldots \ 0 \end{array}\right) & - \left(\begin{array}{c|c} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & 0 \cdot E_{i-1} \\ \hline y & 0 \ \ldots \ 0 \end{array}\right) \\ \hline 0 & \left(\begin{array}{c|c} (x+y)E_i & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \ \ldots \ 0 & y \end{array}\right) - \left(\begin{array}{c|c} 0 \ldots 0 & 0 \\ \hline x \cdot E_i & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \end{array}\right) \end{array}\right)$$

Man kann die Determinate von $D^{(i)}$ also erhalten, indem man die Determinaten der beiden Teilmatrizen auf der Diagonalen miteinander multipliziert.

$$\begin{aligned} \det D^{(i)} &= \det \left(\left(\begin{array}{c|c} x & 0 \quad \ldots \quad 0 \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & (x+y)E_{i-1} \end{array}\right) - \left(\begin{array}{c|c} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & y \cdot E_{i-1} \\ \hline 0 & 0 \ \ldots \ 0 \end{array}\right) \right) \\ & \quad \cdot \det \left(\left(\begin{array}{c|c} (x+y)E_i & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \ \ldots \ 0 & y \end{array}\right) - \left(\begin{array}{c|c} 0 \ldots 0 & 0 \\ \hline x \cdot E_i & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \end{array}\right) \right) \\ &= \det \left(\begin{array}{c|c} x & 0 \quad \ldots \quad 0 \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & (x+y)E_{i-1} \end{array}\right) \cdot \det \left(\begin{array}{c|c} (x+y)E_i & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \ \ldots \ 0 & y \end{array}\right) \\ &= x(x+y)^{i-1} \cdot (x+y)^i y = xy(x+y)^{2i-1} \end{aligned}$$

$\square$

Die folgenden Matrizen werden später noch eine wichtige Rolle spielen. Da eine von ihnen für den nächsten Beweis bereits benötigt wird, werden sie schon jetzt definiert.

Definition 3.7. Sei $O^{(s)} \in M(((s+1)^2-1) \times ((s+1)^2-1), P)$ die Matrix, die durch Streichen der ersten Zeile und Spalte aus $N^{(s)}$ entsteht, also

$$O^{(s)} := \begin{pmatrix} A^{(1)} & B^{(1)} & & 0 \\ C^{(1)} & \ddots & \ddots & \\ & \ddots & A^{(s-1)} & B^{(s-1)} \\ 0 & & C^{(s-1)} & D^{(s)} \end{pmatrix}.$$

Weiter sei $O^{(s)}_{0,0} := O^{(s)}$ und für $k = 1, \ldots, s$ sowie $l_k = 0, \ldots, 2k$ sei $O^{(s)}_{k,l_k}$ die Matrix die aus $O^{(s)}$ entsteht, wenn die $(k^2 + l_k)$-te Spalte durch

$$b := \begin{pmatrix} 0 \\ z \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in P^{(s+1)^2-1}$$

ersetzt wird.

Nun kann für die in Proposition 3.4 definierte Matrix $N^{(s)}$ der Rang bestimmt werden:

Satz 3.8. $\mathrm{Rang}(N^{(s)}) = (s+1)^2 - 1$

Beweis. Zunächst ist klar, dass $\mathrm{Rang}(N^{(s)}) \leq (s+1)^2 - 1$ gilt, da die Summe aller Zeilen von $N^{(s)}$ Null ergibt, wie man leicht nachprüfen kann.
Da $O^{(s)}$ eine Teilmatrix von $N^{(s)}$ mit $(s+1)^2 - 1$ Zeilen und Spalten ist, genügt es zu zeigen, dass sie vollen Rang besitzt.
Setze in $O^{(s)}$ die Unbestimmte $z := 0$. Es ensteht die Matrix

$$O^{(s)\prime} = \begin{pmatrix} D^{(1)} & B^{(1)} & & 0 \\ & \ddots & \ddots & \\ & & D^{(s-1)} & B^{(s-1)} \\ 0 & & & D^{(s)} \end{pmatrix} \in M(((s+1)^2-1) \times ((s+1)^2-1), P).$$

Offenbar gilt $\mathrm{Rang}(O^{(s)\prime}) \leq \mathrm{Rang}(O^{(s)})$. Die Determinante kann wieder aus den Determinanten der Teilmatrizen bestimmt werden:

$$\det(O^{(s)\prime}) = \prod_{i=1}^{s} \det D^{(i)} = \prod_{i=1}^{s} xy(x+y)^{2i-1} = x^s y^s (x+y)^{s^2}$$

Da offensichtlich $\det(O^{(s)\prime}) \neq 0$ ist, hat $O^{(s)\prime}$ vollen Rang $(s+1)^2 - 1$ und damit auch $O^{(s)}$. □

Bemerkung 3.9. Aus Satz 3.8 folgt sofort, dass das Gleichungssystem (2), aufgefasst über dem Quotientenkörper von P, einen 1-dimensionalen Lösungsraum besitzt. Durch die zusätzliche Normierung

$$\sum_{i,j=0}^{s} x_{ij} = 1$$

(die Summe der Wahrscheinlichkeiten aller möglichen Zustände ergibt 1) erhält man eine eindeutige Lösung für das Problem. Multipliziert man diese mit dem Hauptnenner durch, so erhält man eine Lösung in P. Im Folgenden genügt es aber, das Gleichungssystem über P zu lösen, da anschließend stets normiert werden kann.

Satz 3.10. Sei $l = (l_i)_{i=1,\ldots,(s+1)^2} \in P^{(s+1)^2}$ eine Lösung des Gleichungssystems (2) bzw. (3) mit $\mathrm{ggT}((l_i)_{i=1,\ldots,(s+1)^2}) = 1$. Dann gilt für jede weitere Lösung $l' = (l'_i)_{i=1,\ldots,(s+1)^2} \in P^{(s+1)^2}$

$$l' = f \cdot l$$

mit $f \in P$. Insbesondere gibt es eine bis auf eine Einheit eindeutig bestimmbare Lösung kleinsten Grades im Polynomring P.

Beweis. Die beiden Lösungen l und l' über dem Polynomring können auch als Lösungen über dem Quotientenkörper aufgefasst werden. Dort ist eine Normierung auf die eindeutige Lösung des Systems möglich.
Für jedes $i = 1, \ldots, (s+1)^2$ gilt dann

$$\frac{l_i}{\sum_{j=1}^{(s+1)^2} l_j} = \frac{l'_i}{\sum_{j=1}^{(s+1)^2} l'_j}$$

Schreibe

$$\frac{\sum_{j=1}^{(s+1)^2} l'_j}{\sum_{j=1}^{(s+1)^2} l_j} = \frac{f}{g}$$

mit zwei Polynomen $f, g \in P$, für die $\mathrm{ggT}(f,g) = 1$. Dann gilt für alle $i = 1, \ldots, (s+1)^2$:

$$l'_i = \frac{f}{g} l_i$$

Da l'_i stets ein Polynom ist und $\mathrm{ggT}(f,g) = 1$, muss g ein Teiler von l_i sein, für alle $i = 1, \ldots, (s+1)^2$. Da aber $\mathrm{ggT}((l_i)_{i=1,\ldots,(s+1)^2}) = 1$ ist, folgt $g = 1$ und damit ergibt sich die Behauptung. □

Mit Hilfe der Cramerschen Regel kann nun endlich eine Lösung des Gleichungssystems (3) gefunden werden. Ab jetzt bezeichne $M^{\mathtt{T}}$ die Transponierte der Matrix M.

Satz 3.11. Für $k = 0, \ldots, s$ sowie $l_k = 0, \ldots, 2k$ sei $L_{k^2+l_k} := \det O^{(s)}_{k,l_k}$. Dann gilt für

$$\begin{aligned} L &:= (L_0, \ldots, L_{s^2+2s})^{\mathtt{T}} \\ &= \left(\det O^{(s)}_{0,0}, \det O^{(s)}_{1,0}, \det O^{(s)}_{1,1}, \ldots, \det O^{(s)}_{s,2s}\right)^{\mathtt{T}} \in P^{(s+1)^2} \end{aligned}$$

die Gleichung

$$N^{(s)} \cdot L = 0.$$

Bezogen auf das Gleichungssystem (3) bedeutet diese Aussage, dass die

$$y_{k,l_k} := \det O^{(s)}_{k,l_k}$$

eine Lösung des Systems darstellen.

Beweis. Sei $-b$ der Vektor, der entsteht, wenn man die erste Komponente der ersten Spalte von $N^{(s)}$ streicht.
Nach der Cramerschen Regel ist der Vektor

$$L' = \left(\frac{\det O^{(s)}_{1,0}}{\det O^{(s)}}, \frac{\det O^{(s)}_{1,1}}{\det O^{(s)}}, \ldots, \frac{\det O^{(s)}_{s,2s}}{\det O^{(s)}}\right)^{\mathtt{T}} \in P^{(s+1)^2-1}$$

die Lösung der Gleichung $O^{(s)} \cdot L' = b$.
Setzt man $L'' := \left(1, (L')^{\mathtt{T}}\right)^{\mathtt{T}}$, so kann man die Gleichung umformen in $\left(-b \mid O^{(s)}\right) \cdot L'' = 0$. Da die Summe aller Zeilen von $N^{(s)}$ Null ergibt und $\left(-b \mid O^{(s)}\right)$ durch Streichen der ersten Zeile aus $N^{(s)}$ hervorgeht, gilt auch $N^{(s)} \cdot L'' = 0$. Multipliziert man L'' mit $\det O^{(s)}$ durch, erhält man die Behauptung. $\square$

Bemerkung 3.12. Im Allgemeinen ist die in Satz 3.11 bestimmte Lösung nicht die aus Satz 3.10 folgende Lösung kleinsten Grades. Schon im Fall $s = 2$ besitzen die Komponenten des so bestimmten Lösungspolynoms einen gemeinsamen Teiler. Obwohl bei den für diese Arbeit durchgeführten Berechnungen für $s \geq 3$ keine gemeinsamen Teiler mehr gefunden wurden, kann an dieser Stelle nur vermutet werden, dass für $s \geq 3$ die Lösung aus Satz 3.11 auch die Lösung kleinsten Grades in P ist. In jedem Fall ist es aber interessant sie weiter zu untersuchen, da man die Lösung kleinsten Grades in P aus ihr erhalten kann, indem man den größten gemeinsamen Teiler der Komponenten findet und anschließend herausteilt.
Eine weitere interessante Eigenschaft der Lösung aus Satz 3.11 besteht darin, dass alle Koeffizienten ganze Zahlen sind, weil alle Koeffizienten in den Matrizen, deren Determinanten gebildet werden, schon ganzzahlig waren.

Der Rest dieses Abschnitts beschäftigt sich jetzt mit der Lösung aus Satz 3.11. Es sollen sowohl die Leitterme als auch die Leitkoeffizienten ihres Lösungsvektors bestimmt werden. Hierfür benötigen wir eine leichte Abwandlung der Matrizen aus Definition 3.7.

Definition 3.13. Sei $\tilde{O}^{(1)} := D^{(1)}$.
Für $s \in \mathbb{N}$, $s > 1$ setze

$$\tilde{O}^{(s)} := \begin{pmatrix} A^{(1)} & \tilde{B}^{(1)} & & 0 \\ \tilde{C}^{(1)} & \ddots & \ddots & \\ & \ddots & A^{(s-1)} & \tilde{B}^{(s-1)} \\ 0 & & \tilde{C}^{(s-1)} & D^{(s)} \end{pmatrix}$$

mit $\tilde{C}^{(i)} := \frac{1}{z}C^{(i)}$ und $\tilde{B}^{(i)} := zB^{(i)}$ für $i = 1, \ldots, s$.
Weiter sei $\tilde{O}^{(s)}_{0,1} := \tilde{O}^{(s)}$ und für $k = 1, \ldots, s$ sowie $l_k = 0, \ldots, 2k$ sei $\tilde{O}^{(s)}_{k,l_k}$ die Matrix die aus $\tilde{O}^{(s)}$ entsteht, wenn die $(k^2 + l_k)$-te Spalte durch

$$b := \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in P^{(s+1)^2-1}$$

ersetzt wird.

Lemma 3.14. Für $i, j = 0, \ldots, n$ seien $l_i \in \mathbb{N}_+$, $A_{i,j} \in M(l_i \times l_j, P)$ Matrizen und $z \in P$. Setze

$$\mathcal{A}_1^{(n)} := \begin{pmatrix} A_{0,0} & A_{0,1} & \ldots & A_{0,n} \\ z \cdot A_{1,0} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & A_{n-1,n} \\ z^n \cdot A_{n,0} & \ldots & z \cdot A_{n,n-1} & A_{n,n} \end{pmatrix}$$

und

$$\mathcal{A}_2^{(n)} := \begin{pmatrix} A_{0,0} & z \cdot A_{0,1} & \ldots & z^n \cdot A_{0,n} \\ A_{1,0} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & z \cdot A_{n-1,n} \\ A_{n,0} & \ldots & A_{n,n-1} & A_{n,n} \end{pmatrix}$$

Dann gilt

$$\det \mathcal{A}_1^{(n)} = \det \mathcal{A}_2^{(n)}$$

Beweis. Für $n = 1$ gilt:

$$\det\left(\begin{array}{c|c} A_{0,0} & A_{0,1} \\ \hline z\cdot A_{1,0} & A_{1,1}\end{array}\right) = \left(\frac{1}{z}\right)^{l_1} \det\left(\begin{array}{c|c} A_{0,0} & z\cdot A_{0,1} \\ \hline z\cdot A_{1,0} & z\cdot A_{1,1}\end{array}\right) = \det\left(\begin{array}{c|c} A_{0,0} & z\cdot A_{0,1} \\ \hline A_{1,0} & A_{1,1}\end{array}\right)$$

Sei nun die Behauptung für $n-1$ bereits gezeigt.
Für $i, j = 0, \ldots, n-2$ setze $B_{i,j} := A_{i,j}$, $B_{i,n-1} := \left(A_{i,n-1} \mid A_{i,n}\right)$,
$B_{n-1,j} := \left(\begin{array}{c} A_{n-1,j} \\ \hline z\cdot A_{n,j}\end{array}\right)$ und $B_{n-1,n-1} := \left(\begin{array}{c|c} A_{n-1,n-1} & A_{n-1,n} \\ \hline A_{n,n-1} & A_{n,n}\end{array}\right)$.
Weiter definiere $\mathcal{B}_1^{(n-1)}$ und $\mathcal{B}_2^{(n-1)}$ analog zu $\mathcal{A}_1^{(n-1)}$ bzw. $\mathcal{A}_2^{(n-1)}$.
Nach Induktionsvoraussetzung gilt

$$\det \mathcal{A}_1^{(n)} = \det \mathcal{B}_1^{(n-1)} = \det \mathcal{B}_2^{(n-1)} = \det \left(\begin{array}{c|c} & z^{n-1}\cdot A_{0,n} \\ \mathcal{A}_2^{(n-1)} & \vdots \\ & z\cdot A_{n-2,n} \\ & A_{n-1,n} \\ \hline z\cdot\left(A_{n,0} \ldots A_{n,n-1}\right) & A_{n,n}\end{array}\right)$$

Wendet man nun den Fall $n = 1$ an, folgt die Behauptung. □

Korollar 3.15. Lemma 3.14 liefert für $k = 1, \ldots, s$ und $l_k = 0, \ldots, 2k$

$$z^k \det \tilde{O}_{k,l_k}^{(s)} = \det O_{k,l_k}^{(s)}$$

Beweis. Folgt direkt aus Definition 3.7 und Definition 3.13 □

Im Folgenden sei stets $\sigma :=$ DegRevLex und $\mathrm{LM}_\sigma(f)$, $\mathrm{LT}_\sigma(f)$ sowie $\mathrm{LC}_\sigma(f)$ bezeichnen Leitmonom, Leitterm beziehungsweise Leitkoeffizient eines Polynoms $f \in P$ bezüglich der Termordnung σ.
Der nächste Satz liefert einen Zusammenhang zwischen den Leitmonomen der Lösung zweier aufeinander folgender Fälle s und $s+1$.

Satz 3.16. Falls für alle $k = 1, \ldots, s$ und $l_k = 0, \ldots, 2k$ gilt:

a) $\mathrm{LM}_\sigma(\det \tilde{O}_{k,l_k}^{(s)})$ ist nicht durch z teilbar und

b) $\deg \mathrm{LM}_\sigma(\det \tilde{O}_{k,l_k}^{(s)}) = (s+1)^2 - 1 - k$

Dann gilt für $k = 1, \ldots, s$ und $l_k = 0, \ldots, 2k$

a) $\mathrm{LM}_\sigma(\det O_{k,l_k}^{(s+1)}) = x^{2s+2} y \,\mathrm{LM}_\sigma(\det O_{k,l_k}^{(s)})$,

b) $\mathrm{LM}_\sigma(\det \tilde{O}_{k,l_k}^{(s+1)})$ ist nicht durch z teilbar und

c) $\deg \mathrm{LM}_\sigma(\det \tilde{O}_{k,l_k}^{(s+1)}) = (s+2)^2 - 1 - k$

Beweis. In $\tilde{O}^{(s)}_{k,l_k}$ setze $z := 0$. Dies ergibt die Matrix $\tilde{O}^{(s)}_{k,l_k}{}'$, die durch Ersetzen der $(k^2 + l_k)$-ten Spalte in

$$\tilde{O}(s)' := \begin{pmatrix} D^{(1)} & & & 0 \\ \tilde{C}^{(1)} & \ddots & & \\ & \ddots & D^{(s-1)} & \\ 0 & & \tilde{C}^{(s-1)} & D^{(s)} \end{pmatrix} \text{ mit } b := \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in P^{(s+1)^2-1} \text{ entsteht.}$$

Offenbar gilt

$$\det \tilde{O}^{(s+1)\prime}_{k,l_k} = \det \tilde{O}^{(s)}_{k,l_k}{}' \cdot \det D^{(s+1)} = \tilde{O}^{(s)}_{k,l_k}{}' \cdot xy(x+y)^{2s+1}$$

und damit auch

$$\mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)\prime}_{k,l_k}) = x^{2s+2}y\,\mathrm{LM}_\sigma(\det \tilde{O}^{(s)}_{k,l_k}{}'). \qquad (*)$$

Aus Voraussetzung a) folgt sofort

$$\mathrm{LM}_\sigma(\det \tilde{O}^{(s)}_{k,l_k}{}') = \mathrm{LM}_\sigma(\det \tilde{O}^{(s)}_{k,l_k}).$$

Voraussetzung b) liefert

$$(s+1)^2 - 1 - k = \deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s)}_{k,l_k}) = \deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s)}_{k,l_k}{}').$$

Mit $(*)$ folgt dann

$$\deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)\prime}_{k,l_k}) = (s+1)^2 - 1 - k + 2s + 2 + 1 = (s+2)^2 - 1 - k.$$

$O^{(s+1)}_{k,l_k}$ besitzt nur lineare polynomiale Einträge. Also ist $\det O^{(s+1)}_{k,l_k}$ ein Polynom vom Grad kleiner oder gleich $(s+2)^2 - 1$. Nach Korollar 3.15 gilt dann $\deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)}_{k,l_k}) \leq (s+2)^2 - 1 - k$.
Wegen $\deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)}_{k,l_k}) \geq \deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)\prime}_{k,l_k})$ folgt Behauptung c).
Wäre $\mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)}_{k,l_k})$ durch z teilbar, so müssten alle anderen Terme von $\det \tilde{O}^{(s+1)}_{k,l_k}$ einen kleineren Grad als $(s+2)^2 - 1 - k$ haben oder auch durch z teilbar sein. Setzt man in dieser Determinante $z = 0$, kommt $\det \tilde{O}^{(s+1)\prime}_{k,l_k}$ heraus und alle durch z teilbaren Terme verschwinden. Wegen $\deg \mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)\prime}_{k,l_k}) = (s+2)^2 - 1 - k$ haben wir einen Widerspruch und es folgt daraus Behauptung b). Aus ihr schließt man sofort

$$\mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)}_{k,l_k}) = \mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)\prime}_{k,l_k}).$$

Insgesamt gilt nun

$$\mathrm{LM}_\sigma(\det \tilde{O}^{(s+1)}_{k,l_k}) = x^{2s+2}y\,\mathrm{LM}_\sigma(\det \tilde{O}^{(s)}_{k,l_k})$$

und mit Korollar 3.15 ergibt sich Behauptung a). □

Der letzte Satz reicht noch nicht aus, um alle Leitmonome zu bestimmen, jedoch gibt es noch Zusammenhänge untereinander, mit denen man sich die fehlenden Leitmonome erschließen kann.

Satz 3.17.

a) $x\,\mathrm{LM}_\sigma(\det O_{s,0}^{(s)}) = y\,\mathrm{LM}_\sigma(\det O_{s,1}^{(s)})$

b) Für $l_s = 1, \ldots, s-1$ gilt
$x\,\mathrm{LM}_\sigma(\det O_{s,l_s}^{(s)}) = \mathrm{LM}_\sigma(z \det O_{s-1,l_s-1}^{(s)} + y \det O_{s,l_s+1}^{(s)})$

c) $x\,\mathrm{LM}_\sigma(\det O_{s,s}^{(s)}) = z\,\mathrm{LM}_\sigma(\det O_{s-1,s-1}^{(s)})$

d) Für $l_s = s+1, \ldots, 2s-1$ gilt
$x\,\mathrm{LM}_\sigma(\det O_{s,l_s}^{(s)}) = \mathrm{LM}_\sigma(z \det O_{s-1,l_s-1}^{(s)} + x \det O_{s,l_s-1}^{(s)})$

e) $y\,\mathrm{LM}_\sigma(\det O_{s,2s}^{(s)}) = x\,\mathrm{LM}_\sigma(\det O_{s,2s-1}^{(s)})$

Beweis. Nach Satz 3.11 erfüllen für $k = 0, \ldots, s$ und $l_k = 0, \ldots, 2k$ die $y_{k,l_k} := \det O_{k,l_k}^{(s)}$ das Gleichungssystem (3). Dies gilt insbesondere auch dann, wenn man auf beiden Seiten der Gleichungen nur die Leitmonome betrachtet. Die letzten 5 Fälle von (3) ergeben dann a) bis e). □

Jetzt kennen wir alle nötigen Fakten, um eine allgemeine Formel für die Leitterme und Leitkoeffizienten der Lösung aus Satz 3.11 herleiten zu können:

Satz 3.18. Für $s > 0$, $k = 0, \ldots, s$ und $l_k = 0, \ldots, 2k$ gilt:

a) $\mathrm{LT}_\sigma(\det O_{k,l_k}^{(s)}) = x^{s^2+s+l_k-2k} y^{s+k-l_k} z^k$

b) $\mathrm{LC}_\sigma(\det O_{k,l_k}^{(s)}) = \begin{cases} \binom{2k-l_k}{k}\frac{l+1}{k+1} & \text{für } k \geq l_k \\ 1 & \text{für } k < l_k \end{cases}$

Beweis. Für $k = 0, \ldots, s+1$ und $l_k = 0, \ldots, 2k$ definiere

$$c_{k,l_k} := \begin{cases} \binom{2k-l_k}{k}\frac{l+1}{k+1} & \text{für } k \geq l_k \\ 1 & \text{für } k < l_k \end{cases}$$

Beweis durch Induktion:
Induktionsbeginn: $s = 1$
Man errechnet leicht:
$\det O_{0,0}^{(1)} = xy(x+y)$, also $\mathrm{LM}_\sigma(\det O_{0,0}^{(1)}) = x^2 y$,
$\mathrm{LM}_\sigma(\det O_{1,0}^{(1)}) = \det O_{1,0}^{(1)} = y^2 z$,
$\mathrm{LM}_\sigma(\det O_{1,1}^{(1)}) = \det O_{1,1}^{(1)} = xyz$ und
$\mathrm{LM}_\sigma(\det O_{1,2}^{(1)}) = \det O_{1,2}^{(1)} = x^2 z$
Sowohl a) als auch b) sind erfüllt.

Induktionsschritt: $s \to s+1$
Für den Fall s seien a) und b) bereits gezeigt. Nach Anwendung von Korollar 3.15 auf a) sieht man, dass alle Voraussetzungen von Satz 3.16 erfüllt sind. Für $k = 0, \dots, s$, $l_k = 0, \dots, 2k$ liefert dieser:

$$\begin{aligned}
\mathrm{LM}_\sigma(\det O^{(s+1)}_{k,l_k}) &= x^{2s+2}y \cdot \mathrm{LM}_\sigma(\det O^{(s)}_{k,l_k}) \\
&= x^{2s+2}y \cdot c_{k,l_k} x^{s^2+s+l_k-2k} y^{s+k-l_k} z^k \\
&= c_{k,l_k} x^{(s+1)^2+(s+1)+l_k-2k} y^{(s+1)+k-l_k} z^k
\end{aligned}$$

Aussage a) und b) sind also erfüllt. Im Folgenden sei stets $k = s+1$.
Für $l = s+1$ liefert Satz 3.17 c):

$$\begin{aligned}
\mathrm{LM}_\sigma(\det O^{(s+1)}_{s+1,s+1}) &= x^{-1}z \cdot \mathrm{LM}_\sigma(\det O^{(s+1)}_{s,s}) \\
&= x^{-1}z \cdot c_{s,s} x^{(s+1)^2+(s+1)+s-2s} y^{(s+1)+s-s} z^s \\
&= c_{s,s} x^{(s+1)^2+s+1+s+1-2(s+1)} y^{s+1+(s+1)-(s+1)} z^{s+1}
\end{aligned}$$

Aussage a) ist also erfüllt und wegen $c_{s,s} = 1 = c_{s+1,s+1}$ stimmt auch b).
Sei nun $l = 1, \dots, s$. Weiter seien a) und b) für $l+1$ bereits bewiesen.

$$\begin{aligned}
z \cdot \mathrm{LT}_\sigma(\det O^{(s+1)}_{s,l-1}) &= z \cdot x^{(s+1)^2+s+1+l-1-2s} y^{s+1+s-(l-1)} z^s \\
&= x^{(s+1)^2+s+1+l-1-2s} y^{s+1+s-l+1} z^{s+1} \\
&= x^{(s+1)^2+s+1+l+1-2(s+1)} y^{s+1+s+1-(l+1)+1} z^{s+1} \\
&= y \cdot x^{(s+1)^2+s+1+l+1-2(s+1)} y^{s+1+s+1-(l+1)} z^{s+1} \\
&= y \cdot \mathrm{LT}_\sigma(\det O^{(s+1)}_{s+1,l+1})
\end{aligned}$$

Daher folgt mit Satz 3.17 b):

$$\begin{aligned}
&\mathrm{LM}_\sigma(\det O^{(s+1)}_{s+1,l}) = x^{-1} \cdot \mathrm{LM}_\sigma(z \cdot \det O^{(s+1)}_{s,l-1} + y \cdot \det O^{(s+1)}_{s+1,l+1}) \\
&\quad = x^{-1} \cdot \mathrm{LM}_\sigma(z \cdot \mathrm{LM}_\sigma(\det O^{(s+1)}_{s,l-1}) + y \cdot \mathrm{LM}_\sigma(\det O^{(s+1)}_{s+1,l+1})) \\
&\quad = (c_{s,l-1} + c_{s+1,l+1}) \cdot x^{-1}z \cdot \mathrm{LT}_\sigma(\det O^{(s+1)}_{s,l-1}) \\
&\quad = (c_{s,l-1} + c_{s+1,l+1}) \cdot x^{(s+1)^2+s+1+l-2(s+1)} y^{s+1+s+1-l} z^{s+1}
\end{aligned}$$

Aussage a) ist also erfüllt und Aussage b) folgt aus:

$$\begin{aligned}
c_{s,l-1} + c_{s+1,l+1} &= \binom{2s-l+1}{s} \cdot \frac{l}{s+1} + \binom{2s-l+1}{s+1} \cdot \frac{l+2}{s+2} \\
&= \frac{(2s-l+1)!}{s!(s-l+1)!} \cdot \frac{l}{s+1} + \frac{(2s-l+1)!}{(s+1)!(s-l)!} \cdot \frac{l+2}{s+2} \\
&= \frac{(2s-l+1)!}{(s+1)!(s-l+1)!} \cdot \frac{l(s+2)+(l+2)(s-l+1)}{s+2} \\
&= \frac{(2s-l+1)!}{(s+1)!(s-l+1)!} \cdot \frac{(2s-l+2)(l+1)}{s+2} \\
&= \binom{2s+2-l}{s+1} \cdot \frac{(l+1)}{s+2} = c_{s+1,l}
\end{aligned}$$

Für $l = 0$ liefert Satz 3.17 a):

$$\begin{aligned}
\mathrm{LM}_\sigma(\det O^{(s+1)}_{s+1,0}) &= x^{-1}y \cdot \mathrm{LM}_\sigma(\det O^{(s+1)}_{s+1,1}) \\
&= c_{s+1,1} \cdot x^{(s+1)^2+s+1-2(s+1)} y^{s+1+s+1} z^{s+1}
\end{aligned}$$

Aussage a) ist also erfüllt und Aussage b) gilt wegen

$$\begin{aligned} c_{s+1,0} &= \binom{2(s+1)}{s+1} \cdot \frac{1}{s+2} = \frac{(2(s+1))!}{(s+1)!(s+1)!(s+2)} = \frac{(2(s+1)-1)!\cdot 2}{(s+1)!s!(s+2)} \\ &= \binom{2(s+1)-1}{s+1} \cdot \frac{2}{s+2} = c_{s+1,1} \end{aligned}$$

Sei nun $l = s+2, \dots, 2s+1$. Weiter seien a) und b) für $l-1$ bereits bewiesen.

$$\begin{aligned} z \cdot \mathrm{LT}_\sigma(\det O_{s,l-1}^{(s+1)}) &= x^{(s+1)^2+s+1+l-1-2s} y^{s+1+s-(l-1)} z^{s+1} \\ &>_\sigma x^{(s+1)^2+s+1+l-1-2(s+1)+1} y^{s+1+s+1-(l-1)} z^{s+1} \\ &= x \cdot \mathrm{LT}_\sigma(\det O_{s+1,l-1}^{(s+1)}) \end{aligned}$$

Daher folgt mit Satz 3.17 d):

$$\begin{aligned} \mathrm{LM}_\sigma(\det O_{s+1,l}^{(s+1)}) &= x^{-1} \cdot \mathrm{LM}_\sigma(z \cdot \det O_{s,l-1}^{(s+1)} + x \cdot \det O_{s+1,l-1}^{(s+1)}) \\ &= x^{-1} z \cdot \mathrm{LM}_\sigma(\det O_{s,l-1}^{(s+1)}) \\ &= c_{s,l-1} \cdot x^{(s+1)^2+s+1+l-2(s+1)} y^{s+1+s+1-l} z^{s+1} \end{aligned}$$

Aussage a) ist also erfüllt und wegen $c_{s+1,l} = 1 = c_{s,l-1}$ stimmt auch b). Für $l = 2s+2$ liefert Satz 3.17 e):

$$\begin{aligned} &\mathrm{LM}_\sigma(\det O_{s+1,2s+2}^{(s+1)}) = y^{-1} x \cdot \mathrm{LM}_\sigma(\det O_{s+1,2s+1}^{(s+1)}) \\ &\quad = c_{s+1,2s+1} \cdot x^{(s+1)^2+s+1+2s+1-2(s+1)+1} y^{s+1+s+1-(2s+1)-1} z^{s+1} \\ &\quad = c_{s+1,2s+1} \cdot x^{(s+1)^2+s+1+2s+2-2(s+1)} y^{s+1+s+1-(2s+2)} z^{s+1} \end{aligned}$$

Aussage a) ist also erfüllt und wegen $c_{s+1,2s+1} = 1 = c_{s,2s+2}$ stimmt auch Aussage b). □

Wechselt man von der L-förmigen Anordnung der Unbestimmten wieder zur lexikographischen Anordnung zurück, so erhält man aus dem gerade bewiesenen Satz sofort das folgende abschließende Theorem:

Theorem 3.19. Für $s > 0$ und $i, j = 0, \dots, s$ gilt:

a) $\mathrm{LT}_\sigma(x_{i,j}) = x^{s^2+\min(i,j)+i-j} y^{s+j-i} z^{s-\min(i,j)}$

b) $\mathrm{LC}_\sigma(x_{i,j}) = \begin{cases} \binom{s-\min(i,j)+j-i}{j-i} \frac{s-\min(i,j)+i-j+1}{s-\min(i,j)+1} & \text{für } j \geq i \\ 1 & \text{für } j < i \end{cases}$

falls $(x_{i,j})_{i,j=1,\dots,s}$ die Lösung des Gleichungssystems (2) nach der Cramerschen Regel darstellt.

Beweis. Folgt direkt aus Satz 3.18 wenn man Satz 3.11 und Satz 3.1 anwendet. □

Bemerkung 3.20. Nach Satz 3.10 geht die untersuchte Lösung aus Satz 3.11 durch Multiplikation mit einem Polynom f aus einer Lösung kleinsten

Grades über P wie in Satz 3.10 hervor. Insbesondere gilt dies auch, wenn man nur die Leitterme betrachtet. Für eine der Lösungen kleinsten Grades über P bedeutet die Aussage von Theorem 3.19 also, dass ihre Leitkoeffizienten genau wie in b) sind. Unabhängig von der Lösungsmethode sind die Leitkoeffizienten aller $x_{i,i}$ für $i = 0, \dots, s$ gleich, es soll also stets durch den Leitkoeffizienten von etwa $x_{s,s}$ geteilt werden. Dann kann man auch von **der** Lösung kleinsten Grades über P sprechen. Die Leitterme ihrer Komponenten teilen die entsprechenden aus a) derart, dass stets derselbe Term entsteht. Ab hier sind also modulo eines gemeinsamen Teilers auch die Leitterme und Leitkoeffizienten der Lösung kleinsten Grades über P bestimmt.

4 Algorithmen und Lösungsansätze

In diesem Abschnitt sollen Methoden zur Lösung des Gleichungssystems (2) behandelt werden. Hierbei soll jedoch noch nicht auf die Implementierung in CoCoA eingegangen werden, auch wenn einige Optimierungsschritte durch Ergebnisse bei der Anwendung oder durch die Eigenheiten von CoCoA motiviert sind.

4.1 Grundlegende Lösungsalgorithmen

Eine naheliegende Methode zur Lösung eines linearen Gleichungssystems ist der Gauss-Algorithmus. Fasst man das Gleichungssystem (2) über dem Quotientenkörper von P auf, so könnte man ihn direkt anwenden. Dabei würden allerdings rationale Funktionen auftreten, die die Berechnung in CoCoA gewaltig verlangsamen (da ständig versucht wird zu kürzen). Es gilt also den Gauss-Algorithmus so zu modifizieren, dass der Polynomring nicht verlassen wird.

Satz 4.1. Sei $M = (M_{i,j})$ eine Matrix mit n Zeilen und m Spalten, $m \geq n$. Für $i = 1, \dots, n$ und $j = 1, \dots, m$ seien die Einträge $M_{i,j}$ Polynome.
Betrachte die folgenden Anweisungen:

(1) Setze $i := 1$ und $j := 1$

(2) Falls $M_{i,j} = 0$, vertausche die i-te Zeile mit einer Zeile $k > i$, für die $M_{k,j} \neq 0$. Sollte es keine solche Zeile geben und $j < m$ sein, erhöhe j um Eins und wiederhole (2). Ansonsten mache nichts.

(3) Falls $i < n$ ist, setze $k := i + 1$, sonst gehe zu (8)

(4) Falls $M_{k,j} \neq 0$ subtrahiere das $M_{k,j}$-fache der i-ten Zeile vom $M_{i,j}$-fachen der k-ten Zeile und setze das Ergebnis als neue k-te Zeile.

(5) Teile die k-te Zeile durch den größten gemeinsamen Teiler ihrer Komponenten.

(6) Falls $k < n$, erhöhe k um Eins und fahre bei (4) fort.

(7) Falls $i < n$ und $j < m$, erhöhe i und j um Eins und fahre bei (2) fort.

(8) Gib die (modifizierte) Matrix M zurück.

Dies ist ein Algorithmus, der die ihm übergebene Matrix in Zeilenstufenform überführt und dabei den Polynomring nicht verlässt. Insbesondere ist das Ergebnis eine obere Dreiecksmatrix und die letzte Zeile eine Nullzeile, falls der Rang der Matrix $n - 1$ und $n = m$ ist.

Beweis. Bei jedem Schritt, der zu einem Vorausgehenden zurückspringt, wird i und j bzw. k erhöht, aber diese Variablen werden nie verringert. Da alle diese Variablen nach oben durch n bzw. m beschränkt sind, kann ein Zurückspringen nur endlich oft vorkommen. Es handelt sich deswegen um einen Algorithmus.
Alle auftretenden Umformungen sind elementare Zeilenumformungen, der Lösungsraum ändert sich also nicht. Dabei wird nur in Schritt (5) dividiert, allerdings dort durch ein Polynom, welches auch Teiler ist. Der Polynomring wird also nicht verlassen.
In den Schritten (3) bis (6) werden alle Einträge der j-ten Spalte unterhalb der i-ten Zeile zu Null gemacht. Eine Spalte wird in Schritt (2) nur dann übersprungen, wenn eben diese Einträge schon Null waren.
Nach jedem vollen Durchlauf von (2)-(7) ist die j-te Spalte unterhalb der i-ten Zeile Null. Da im jeweils darauffolgenden Durchlauf nur die Zeilen unterhalb der i-ten Zeile verwendet werden, bleiben diese Einträge auch weiterhin Null. Nach Beendigung des Algorithmus entspricht das Resultat dann genau der Zeilenstufenform.
Ist speziell der Rang der Matrix $n - 1$ und $n = m$, so ist klar, dass die Matrix in Zeilenstufenform am Ende genau eine Nullzeile besitzen muss. Da $n = m$ ist, konnte im gesamten Algorithmus keine Spalte übersprungen worden sein, folglich handelt es sich wirklich um eine obere Dreiecksmatrix mit Einträgen ungleich Null in der Hauptdiagonalen (außer in der letzten Zeile natürlich). □

Bemerkung 4.2. In Satz 4.1 ist der Schritt (5) nicht zwingend notwendig. Er verhindert jedoch, dass die Grade explodieren, was zu einer gewaltigen Verlangsamung führen würde. Leider benötigt dieser Schritt gerade bei Polynomen höheren Grades viel Zeit, wodurch der Algorithmus ausgebremst wird. Die Verlangsamung durch die explodierenden Grade ist aber noch größer, so dass bei der Implementierung auf Schritt (5) nicht verzichtet wird.

Mit Hilfe des Algorithmus aus Satz 4.1 kann man nun den ersten Lösungsalgorithmus formulieren. Dabei muss zusätzlich zur Trigonalisierung die Matrix ausgeräumt werden, um einen Lösungsvektor ablesen zu können.

Satz 4.3. Lösung mit Hilfe des Gauss-Algorithmus
Sei M eine Matrix mit $n \in \mathbb{N}$ Zeilen und Spalten und dem Rang $n-1$. Betrachte die folgenden Anweisungen:

(1) Überführe M mit Hilfe von Satz 4.1 in eine obere Dreiecksmatrix.

(2) Setze $k := n-1$ und $i := 1$. Falls $i \geq k$ gehe zu Schritt (5).

(3) Falls $M_{i,k} \neq 0$ subtrahiere das $M_{i,k}$-fache der k-ten Zeile vom $M_{k,k}$-fachen der i-ten Zeile und setze das Ergebnis als neue i-te Zeile.

(4) Teile die i-te Zeile durch den größten gemeinsamen Teiler ihrer Komponenten.

(5) Falls $i < k-1$ erhöhe i um 1 und gehe zu Schritt (3)

(6) Falls $k > 2$ vermindere k um 1, setze $i := 1$ und gehe zu Schritt (3)

(7) Finde das kleinste gemeinsame Vielfache $V \in P$ von $(M_{i,i})_{i=1,\ldots,n-1}$ und multipliziere für $i = 1, \ldots, n-1$ die i-te Zeile von M mit $\frac{V}{M_{i,i}}$

(8) Gib den Vektor $l := (-M_{1,n}, \ldots, -M_{n-1,n}, V) \in P^n$ zurück.

Dies ist ein Algorithmus, der ein $l \in P^n$ mit $M \cdot l = 0$ liefert.

Beweis. In Schritt (1) wird M in Zeilenstufenform überführt. Da M genauso viele Zeilen wie Spalten hat, und ihr Rang $n-1$ ist, entsteht eine obere Dreiecksmatrix, deren letzte Zeile eine Nullzeile ist. Der Lösungsraum ändert sich dabei nicht.
In Schritt (3) wird der Eintrag M_{ik} zu Null gemacht, falls er es nicht schon war. Aufgrund der Gestalt der k-ten Zeile ändert sich dabei sonst nur die letzte Spalte. Der vierte Schritt dient wieder dazu die Grade klein zu halten. Die Schritte (3)-(5) werden ausgeführt bis die ganze Spalte oberhalb der Eintrags $M_{k,k}$ zu Null geworden ist. Schritt (6) wechselt dann in die nächstkleinere Spalte. Da es nur endlich viele Zeilen und Spalten gibt, endet der Algorithmus.
Wenn Schritt (7) erreicht ist, hat die Matrix nur noch in der Hauptdiagonalen und der letzten Spalte Einträge ungleich Null. Die hier beschriebene Operation sorgt dafür, dass in der Hauptdiagonalen stets dasselbe Polynom steht. Nun kann man das l mit der gewünschten Eigenschaft direkt ablesen, was in Schritt (8) getan wird. Da alle auftretenden Operationen nur Zeilenumformungen sind, die den Lösungsraum unverändert lassen, gilt die Eigenschaft auch für das ursprüngliche M. □

Um die Theorie der Computeralgebra für die Lösung dieses linearen Gleichungssystems ausnutzen zu können, müssen wir uns erst einmal verdeutlichen, was Lösungen eigentlich für Eigenschaften besitzen. Sie sind Vektoren, die multipliziert mit der zugehörigen Matrix den Nullvektor ergeben.

Fasst man die Spalten der Matrix als Erzeugende eines Moduls auf, entsprechen diese Lösungsvektoren genau den Syzygien der Spalten (vgl. [KR]). CoCoA besitzt eingebaute Methoden, um solche Syzygien mit Hilfe von Gröbnerbasen zu berechnen. Dies führt uns zum zweiten Lösungsalgorithmus:

Satz 4.4. Lösung mit Hilfe von Syzygien
Gegeben sei eine Matrix M über dem Polynomring P mit n Spalten und $m \geq n-1$ Zeilen. Weiter seit $\text{Rang}(M) = n-1$. Betrachte die folgenden Anweisungen:

(1) Definiere N als den von den Spalten von M erzeugten Modul.

(2) Berechne den zu N gehörigen Syzygienmodul S

(3) Teile den ersten Erzeugenden von S durch den Leitkoeffizienten seiner letzten Komponente und gib das Ergebnis zurück.

Dies ist ein Algorithmus, der ein $l \in P^n$ mit $M \cdot l = 0$ liefert.

Beweis. Eine Syzygie des von den Spalten von M aufgespannten Moduls N hat genau die gewünschte Eigenschaft. Im ersten Schritt wird nur dieser Modul erzeugt. Der Syzygienmodul S von N kann effektiv berechnet werden (siehe [KR]). Jedes Element von S ist eine Syzygie, insbesondere auch der erste Erzeuger. Da der Rang von M genau um 1 kleiner als die Spaltenzahl ist, muss der Lösungsraum des zugehörigen Gleichungssystems über dem Quotientenkörper von P eindimensional sein. Das Herausteilen des Leitkoeffizienten der letzten Komponente liefert also wieder eine Lösung des zugehörigen Gleichungssystems. □

Eine weitere Methode besteht darin, das Gleichungssystem mit Hilfe von Bareiss-Verfahren zu lösen. Genaueres hierzu kann in [K] nachgelesen werden. Von den dort getesteten Algorithmen soll hier lediglich das Forward-Backup-Verfahren "B2DMDMDFB" sowie ein Dichotomie-Verfahren zu Vergleichszwecken verwendet werden. In den beiden zugehörigen Algorithmen kann man sich auf den Aufruf dieser Funktionen und das Umformen des Ergebnisses in einen Vektor beschränken.

Satz 4.5. Lösung mit Hilfe von Bareiss-Verfahren
Die Matrix M mit n Spalten und $n-1$ Zeilen habe den Rang $n-1$. Betrachte die folgenden Anweisungen:

(1) Bringe M mit Hilfe von `B2DMDMDFB` oder `MalDichHM` in Diagonalform.

(2) Berechne den größten gemeinsamen Teiler $g \in P$ der Komponenten des Vektors $l' := (-M_{1,n}, \ldots, -M_{n-1,n}, M_{1,1}) \in P^n$.

(3) Gib $\frac{l'}{g} \in P^n$ zurück.

Dies ist ein Algorithmus, der ein $l \in P^n$ mit $M \cdot l = 0$ liefert.

Beweis. Nach [K] ist das Ergebnis beider Funktionen eine Matrix, die nur in der Hauptdiagonalen und in der letzten Spalte Polynome ungleich Null beinhaltet. Dabei sind alle Einträge der Hauptdiagonalen gleich. Aus einer Matrix dieser Gestalt kann man eine Lösung wie in Schritt (2) sofort ablesen. Unglücklicherweise scheinen die Komponenten der Lösung manchmal nicht teilerfremd zu sein, also wird der größte gemeinsame Teiler bestimmt und herausdividiert. Da der Lösungsraum (über dem Quotientenkörper von P) eindimensional war, bleibt das Ergebnis hierbei eine Lösung des zugehörigen Gleichungssystems, die sogar weiter in P liegt. □

Bemerkung 4.6. Alle beschriebenen Algorithmen liefern die normierte Lösung kleinsten Grades nach Bemerkung 3.20, da durch die jeweils angewandten Normierungen sichergestellt ist, dass die letzte Komponente den Leitkoeffizienten 1 hat. Wo es nötig war, wurde zusätzlich durch den größten gemeinsamen Teiler der Komponenten geteilt.

4.2 Kompression

Alle vorgestellten Algorithmen geht nicht auf die spezielle Struktur der zum Gleichungssystem (2) gehörigen Matrix ein. Daher soll diese nun genauer betrachtet werden.

Definition 4.7. Sei K ein Körper und $P := K[x, y, z]$
Sei $E_s \in M(s \times s, P)$ die Einheitsmatrix. Setze für jedes $s \in \mathbb{N}_+$

$$A_{s+1} := \left(\begin{array}{c|c} y \cdot E_s & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 0 \; \dots \; 0 & 0 \end{array}\right) + \left(\begin{array}{c|c} 0 \; \dots \; 0 & 0 \\ \hline -y \cdot E_s & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \end{array}\right) + x \cdot E_{s+1}$$

$$B_{s+1} := \left(\begin{array}{c|c} \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & -z \cdot E_s \\ \hline 0 & 0 \; \dots \; 0 \end{array}\right) \qquad C_{s+1} := \left(\begin{array}{c|c} 0 & 0 \; \dots \; 0 \\ \hline \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} & z \cdot E_s \end{array}\right) + A_{s+1}$$

$$D_{s+1} := C_{s+1} - x \cdot E_{s+1}$$

Hierbei sind $A_{s+1}, B_{s+1}, C_{s+1}, D_{s+1} \in M((s+1) \times (s+1), P)$.

Mit Hilfe dieser "Bausteine" ist es nun möglich die zum Gleichungssystem (2) gehörige Matrix zu beschreiben.

Proposition 4.8. Die Matrix

$$M_s := \begin{pmatrix} A_{s+1} & B_{s+1} & & & 0 \\ -x \cdot E_{s+1} & C_{s+1} & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & C_{s+1} & B_{s+1} \\ 0 & & & -x \cdot E_{s+1} & D_{s+1} \end{pmatrix} \in M((s+1)^2 \times (s+1)^2, P)$$

beschreibt das Gleichungssystem (2), falls die Zeilen und Spalten lexikographisch anordnet werden.

Beweis. Die Matrix M_s zerlegt sich auf natürlicher Weise in $s+1$ Zeilen bzw. Spalten aus $(s+1) \times (s+1)$-Matrizen. Betrachte zunächst die erste Blockzeile, sie wird gebildet aus den Gleichungen $f_{0,j}$ mit $j = 0, \ldots, s$. Die Koeffizienten der $x_{0,j}$ entsprechen der Hauptdiagonalen, die Koeffizienten der $x_{0,j-1}$ der ersten unteren Nebendiagonalen von A_{s+1}. Die Koeffizienten der $x_{1,j+1}$ entsprechen der ersten oberen Nebendiagonalen von B_{s+1}.
Analog wird die letzte Blockzeile aus den Gleichungen $f_{s,j}$ gebildet. Die Koeffizienten der $x_{s,j}$ entsprechen der Hauptdiagonalen, die Koeffizienten der $x_{s,j-1}$ der ersten unteren Nebendiagonalen von D_{s+1}. Die Koeffizienten der $x_{s-1,j}$ entsprechen der Hauptdiagonalen von $-x \cdot E_{s+1}$.
Die verbleibenden Blockzeilen werden aus den Gleichungen $f_{i,j}$ mit $i = 1, \ldots, s-1$ gebildet. Die Koeffizienten der $x_{i,j}$ entsprechen der Hauptdiagonalen, die Koeffizienten der $x_{i,j-1}$ der ersten unteren Nebendiagonalen von C_{s+1}. Die Koeffizienten der $x_{i-1,j}$ entsprechen der Hauptdiagonalen von $-x \cdot E_{s+1}$. Die Koeffizienten der $x_{i+1,j+1}$ entsprechen der ersten oberen Nebendiagonalen von B_{s+1}.
Vergleicht man nach diesem Muster die Matrix M_s mit dem Gleichungssystem (2), stellt man Übereinstimmung fest. □

Zur Veranschaulichung sei als Beispiel wieder der Fall $s = 2$ gegeben:

Beispiel 4.9. Im Fall $s = 2$ hat das Gleichungssystem (2) die folgende Gestalt:

$$\begin{array}{rcll}
f_{0,0} &=& x_{0,0} \cdot (x+y) - x_{1,1} \cdot z &= 0 \\
f_{0,1} &=& x_{0,1} \cdot (x+y) - x_{1,2} \cdot z - x_{0,0} \cdot y &= 0 \\
f_{0,2} &=& x_{0,2} \cdot x - x_{0,1} \cdot y &= 0 \\
f_{1,0} &=& x_{1,0} \cdot (x+y) - x_{2,1} \cdot z - x_{0,0} \cdot x &= 0 \\
f_{1,1} &=& x_{1,1} \cdot (x+y+z) - x_{2,2} \cdot z - x_{0,1} \cdot x - x_{1,0} \cdot y &= 0 \\
f_{1,2} &=& x_{1,2} \cdot (x+z) - x_{0,2} \cdot x - x_{1,1} \cdot y &= 0 \\
f_{2,0} &=& x_{2,0} \cdot y - x_{1,0} \cdot x &= 0 \\
f_{2,1} &=& x_{2,1} \cdot (y+z) - x_{1,1} \cdot x - x_{2,1} \cdot y &= 0 \\
f_{2,2} &=& x_{2,2} \cdot z - x_{1,2} \cdot x - x_{2,1} \cdot y &= 0
\end{array}$$

Die zugehörige Matrix M_2 sieht dann wie folgt aus:

$$\left(\begin{array}{ccc|ccc|ccc}
x+y & 0 & 0 & 0 & -z & 0 & 0 & 0 & 0 \\
-y & x+y & 0 & 0 & 0 & -z & 0 & 0 & 0 \\
0 & -y & x & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
-x & 0 & 0 & x+y & 0 & 0 & 0 & -z & 0 \\
0 & -x & 0 & -y & x+y+z & 0 & 0 & 0 & -z \\
0 & 0 & -x & 0 & -y & x+z & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & -x & 0 & 0 & y & 0 & 0 \\
0 & 0 & 0 & 0 & -x & 0 & -y & y+z & 0 \\
0 & 0 & 0 & 0 & 0 & -x & 0 & -y & z
\end{array}\right)$$

Durch die eingezogenen Linien erkennt man sehr schön die Teilmatrizen A_3, B_3, C_3 und D_3 sowie $-x \cdot E_3$.

Der nächste Satz folgt eigentlich schon aus Satz 3.8, da M_s durch Zeilen und Spaltenvertauschungen aus $N^{(s)}$ hervorgeht. Es soll jedoch hier im Hinblick auf den nächsten Unterabschnitt noch ein alternativer Beweis gezeigt werden.

Satz 4.10. $\mathrm{Rang}(M_s) = (s+1)^2 - 1$

Beweis. M_s hat $(s+1)^2$ Zeilen. Die Summe aller Zeilen ergibt eine Nullzeile. Folglich gilt

$$\mathrm{Rang}(M_s) \leq (s+1)^2 - 1.$$

Durch Einsetzen fester Werte für x,y und z kann der Rang nur abnehmen. Setzt man $x := 1$, $y := 1$ und $z := 0$, so ist die Matrix M_s bereits in unterer Dreiecksgestalt. Der durch Streichen der letzten Zeile und Spalte aus ihr hervorgehende Minor hat den Rang $(s+1)^2 - 1$, da in der Hauptdiagonalen nur positive Einträge sind, und damit seine Determinante > 0 ist. Es folgt

$$\mathrm{Rang}(M_s) \geq (s+1)^2 - 1.$$

Insgesamt folgt die Gleichheit. □

Betrachtet man die zum Gleichungssystem (2) gehörige Matrix M_s, so stellt man fest, dass durch Setzen der ersten Blockzeile (bzw. der ersten $s+1$ Zeilen) an das Ende der Matrix der obere Teil schon trigonalisiert ist. Weiter kann man durch einfaches Aufsummieren aller Zeilen eine Nullzeile erschaffen ohne den Lösungsraum zu verändern.
Insbesondere der in Satz 4.1 vorgestellte Gauss-Algorithmus hätte dann weniger Reduktionsschritte durchzuführen, was seine Bearbeitungszeit verringern würde.

Definition 4.11. Ersetze in M_s die $s+1$-te Zeile durch die Summe aller Zeilen, also die Nullzeile. Dabei ändert sich der Lösungsraum des zugehörigen Gleichungssystems nicht. Setze dann die ersten $s+1$ Zeilen ans Ende der Matrix und erhalte M'_s. Mit

$$A'_{s+1} := \left(\begin{array}{ccc|c} & & & 0 \\ & (x+y)\cdot E_s & & \vdots \\ & & & 0 \\ \hline 0 & \dots & 0 & 0 \end{array}\right) + \left(\begin{array}{ccc|c|c} 0 & \dots & 0 & 0 & 0 \\ \hline & & & 0 & 0 \\ & -y\cdot E_{s-1} & & \vdots & \vdots \\ & & & 0 & 0 \\ \hline 0 & \dots & 0 & 0 & 0 \end{array}\right) \in M((s+1)\times(s+1), P)$$

sowie $B_{s+1}, C_{s+1}, D_{s+1}, E_{s+1} \in M((s+1)\times(s+1), P)$ aus Definition 4.7 hat $M'_s \in M((s+1)^2 \times (s+1)^2, P)$ dann folgende Gestalt:

$$M'_s = \begin{pmatrix} -x\cdot E_{s+1} & C_{s+1} & B_{s+1} & & 0 \\ & \ddots & \ddots & \ddots & \\ & & -x\cdot E_{s+1} & C_{s+1} & B_{s+1} \\ 0 & & & -x\cdot E_{s+1} & D_{s+1} \\ A'_{s+1} & B_{s+1} & 0 & \dots & 0 \end{pmatrix}$$

Fasst man M'_s als Blockmatrix aus $(s+1)\times(s+1)$ Matrizen auf, so kann man eine dem Gauss-Algorithmus verwandte Umformung durchführen:

Satz 4.12. Kompression
Seien die Matrizen $A'_{s+1}, B_{s+1}, C_{s+1}, D_{s+1}, E_{s+1} \in M((s+1)\times(s+1), P)$ wie in Definition 4.11 definiert. Setze $S^{(s+1)} := E_{s+1}$ und $S^{(s)} := D_{s+1}$. Für $i = 1, \dots, s-1$ definiere rekursiv

$$S^{(s-i)} := C_{s+1}\cdot S^{(s-i+1)} + x\cdot B_{s+1}\cdot S^{(s-i+2)}$$

und setze

$$S_s := A'_{s+1}\cdot S^{(1)} + x\cdot B_{s+1}\cdot S^{(2)}.$$

Dann gelten die folgenden Aussagen:

a) Die Matrix

$$M' := \begin{pmatrix} -x^s\cdot E_{s+1} & & & & 0 & S^{(1)} \\ & \ddots & & & & \vdots \\ & & -x^i\cdot E_{s+1} & & & S^{(s+1-i)} \\ & & & \ddots & & \vdots \\ & & & & -x^1\cdot E_{s+1} & S^{(s)} \\ 0 & & & & & S_s \end{pmatrix}$$

hat denselben Lösungsraum wie M'_s bzw. M_s.

b) Die letzte Zeile von S_s ist eine Nullzeile und $\mathrm{Rang}(S_s) = s$

Beweis. zu a) Für $i = 1, \ldots, s$ denke man sich die i-te Blockzeile von M'_s mit x^{s-i} und die $(s+1)$-te Blockzeile mit x^s multipliziert. Offensichtlich geht M' aus dieser Matrix hervor, indem mittels Matrizenoperationen zunächst der obere Teil der Matrix diagonalisiert und dann die letzte Zeile ausgeräumt wird. Diese Matrizenoperationen entsprechen Zeilenumformungen, bei denen der Lösungsraum unverändert bleibt. Die rekursive Definition der $S^{(i)}$ folgt genau diesem Vorgang.
zu b) Da M' durch Zeilenumformungen aus M_s hervorgeht, haben beide Matrizen den gleichen Rang $(s+1)^2 - 1$. Aus der Darstellung in a) sieht man sofort, dass die ersten $s \cdot (s+1)$ Zeilen von M' linear unabhängig sind. Von den letzten $s+1$ Zeilen können also nur noch s linear unabhängig sein. Da außer in S_s keine Einträge in diesen Zeilen vorhanden sind, hat S_s den Rang s. Bei den Umformungen bleibt sogar die Nullzeile in der letzten Zeile erhalten. □

Satz 4.13. Expansion
Unter den Voraussetzungen von Satz 4.12 gilt für ein $l \in P^{s+1}$ mit $S_s \cdot l = 0$

$$M_s \cdot \begin{pmatrix} x^0 \cdot S^{(1)} \cdot l \\ \hline \vdots \\ \hline x^s \cdot S^{(s+1)} \cdot l \end{pmatrix} = 0$$

Beweis. Kann unmittelbar aus der Form in Satz 4.12 a) abgelesen werden. □

Im Folgenden kann also anstelle von M_s bzw. M'_s auch die Matrix S_s zur Lösungsfindung verwendet werden. Sie hat zwar keine einfache Struktur mehr, besitzt jedoch deutlich weniger Zeilen und Spalten, wodurch alle Algorithmen schneller ablaufen.

4.3 Einsparung einer Unbestimmten

Theoretisch sollten alle Algorithmen für homogene Polynome in 3 Unbestimmten genauso schnell sein wie für inhomogene Polynome in 2 Unbestimmten. In der Praxis ist allerdings der Verwaltungsaufwand für die erste Methode größer als für die zweite. Auch müssten alle Algorithmen auf den homogenen Fall optimiert werden. Daher empfiehlt es sich erst zu dehomogenisieren um dadurch eine Unbestimmte einzusparen und dann mit Algorithmen zu arbeiten, die nicht auf homogene Systeme optimiert sind.
Alle Gleichungen im Gleichungssystem (2) sind homogen vom Grad 1. Teilt man sie durch x und substituiert anschließend $v := \frac{y}{x}$ sowie $w := \frac{z}{x}$, so erhält

man das folgende Gleichungssystem über dem Polynomring $P' := K[v,w]$.

$$\begin{array}{lllllll}
h_{0,0} = x_{0,0} \cdot & (v+1) & - x_{1,1} & \cdot w & & & = 0 \\
h_{0,j} = x_{0,j} \cdot & (v+1) & - x_{1,j+1} & \cdot w & & - x_{0,j-1} \cdot v & = 0 \\
h_{0,s} = x_{0,s} & & & & & - x_{0,s-1} \cdot v & = 0 \\
h_{i,0} = x_{i,0} \cdot & (v+1) & - x_{i+1,1} & \cdot w & - x_{i-1,0} & & = 0 \\
h_{i,j} = x_{i,j} \cdot & (v+w+1) & - x_{i+1,j+1} & \cdot w & - x_{i-1,j} & - x_{i,j-1} \cdot v & = 0 \\
h_{i,s} = x_{i,s} \cdot & (w+1) & & & - x_{i-1,s} & - x_{i,s-1} \cdot v & = 0 \\
h_{s,0} = x_{s,0} \cdot & v & & & - x_{s-1,0} & & = 0 \\
h_{s,j} = x_{s,j} \cdot & (v+w) & & & - x_{s-1,j} & - x_{s,j-1} \cdot v & = 0 \\
h_{s,s} = x_{s,s} \cdot & w & & & - x_{s-1,s} & - x_{s,s-1} \cdot v & = 0
\end{array} \qquad (4)$$

Hierbei sei $0 < i < s$ und $0 < j < s$.
Beachte das für $i,j = 0,\ldots,s$ nach Resubstitution die $h_{i,j}$ in natürlicher Weise auch als Funktionen im Quotientenkörper $K(x,y,z)$ von P aufgefasst werden können.
Aus dem nun folgenden Satz kann man leicht einen Algorithmus herleiten, mit dem eine Lösung aus dem Polynomring in 2 Unbestimmten in eine Lösung aus dem Polynomring in 3 Unbestimmten überführt werden kann. Im Wesentlichen handelt es sich dabei um eine Homogenisierung.

Satz 4.14. Sei das Tupel $(l_{i,j})_{i,j=0,\ldots,s} \in P'^{(s+1)^2}$ eine Lösung des Gleichungssystems (4), also für $k,l = 0,\ldots,s$ sei stets $h_{k,l}((l_{i,j})_{i,j=0,\ldots,s}) = 0$. Weiter sei $d := \max\{\deg(l_{i,j}) | i,j = 0,\ldots,s\}$.
Dann ist das Tupel $(L_{i,j})_{i,j=0,\ldots,s} \in P^{(s+1)^2}$ mit $L_{i,j} := x^d \cdot l_{i,j}(\frac{y}{x}, \frac{z}{x}) \in P$ eine Lösung des Gleichungssystems (2), also $f_{k,l}((L_{i,j})_{i,j=0,\ldots,s}) = 0$ für alle $k,l = 0,\ldots,s$.

Beweis. Nach Voraussetzung ist $h_{k,l}((l_{i,j})_{i,j=0,\ldots,s}) = 0$ für alle $k,l = 0,\ldots,s$. Dies gilt natürlich auch, wenn man nach Resubstitution von $v = \frac{y}{x}$ und $w = \frac{z}{x}$ in $h_{k,l}$ bzw. $l_{i,j}$ die Gleichungen im Quotientenkörper $K(x,y,z)$ von P auffasst. Dort gilt daher für alle $k,l = 0,\ldots,s$:

$$f_{k,l}((l_{i,j}(\tfrac{y}{x}, \tfrac{z}{x}))_{i,j=0,\ldots,s}) = x \cdot h_{k,l}((l_{i,j}(\tfrac{y}{x}, \tfrac{z}{x}))_{i,j=0,\ldots,s}) = 0$$

Das Tupel $(l_{i,j}(\frac{y}{x}, \frac{z}{x}))_{i,j=0,\ldots,s}$ ist also eine Lösung des Gleichungssystems (2) über dem Quotientenkörper $K(x,y,z)$ von P. Wie bereits in Satz 3.8 bewiesen, ist dieser Lösungsraum linear, folglich ist auch das Tupel $(L_{i,j})_{i,j=0,\ldots,s} \in P^{(s+1)^2}$ mit $L_{i,j} := x^d \cdot l_{i,j}(\frac{y}{x}, \frac{z}{x})$ eine Lösung von (2). Aus der Definition von d folgt sofort $L_{i,j} \in P$, da in den Nennern der $l_{i,j}$ maximal x^d auftritt. Insgesamt ist also $(L_{i,j})_{i,j=0,\ldots,s}$ auch eine Lösung von (2) aufgefasst über P. $\square$

Bemerkung 4.15. Die Einsparung einer Umbestimmten ist mit den Algorithmen aus den vorangegangenen Unterabschnitten verträglich. Formal erhält man das Gleichungssystem (4) aus dem System (2), indem man

$(x, y, z) := (1, v, w)$ einsetzt. Dieselbe Ersetzung kann nun in allen Definitionen und Sätzen von 4.2 gemacht werden, dabei geht P in P' über. Man sieht sofort, dass sich die Aussagen der Sätze ebenfalls erhalten.

Der Satz über die Kompression und Expansion soll für den Fall $P' = K[v, w]$ noch einmal formuliert und um ein paar Aussagen erweitert werden.

Satz 4.16. (Kompression)
Die Matrizen $A'_{s+1}, B_{s+1}, C_{s+1}, D_{s+1}, E_{s+1} \in M((s+1) \times (s+1), P')$ seien wie in Definition 4.11 definiert. Dabei sei die formale Substitution $(x, y, z) := (1, v, w)$ bereits vorgenommen. Zum leichteren Verständnis behalten jedoch alle Matrizen ihren Namen.
Setze $S^{(s+1)} := E_{s+1}$ und $S^{(s)} := D_{s+1}$. Für $i = 1, \dots, s-1$ definiere rekursiv

$$S^{(s-i)} := C_{s+1} \cdot S^{(s-i+1)} + B_{s+1} \cdot S^{(s-i+2)}$$

und setze

$$S_s := A'_{s+1} \cdot S^{(1)} + B_{s+1} \cdot S^{(2)}.$$

Dann gelten die folgenden Aussagen:

a) Die Matrix

$$M' := \begin{pmatrix} -E_{s+1} & 0 & S^{(1)} \\ & \ddots & \vdots \\ & -E_{s+1} & S^{(s)} \\ 0 & & S_s \end{pmatrix}$$

hat denselben Lösungsraum wie M'_s bzw. M_s (über P' definiert).

b) Die letzte Zeile der Matrix S_s ist eine Nullzeile und $\mathrm{Rang}(S_s) = s$

c) Jedes $L \in P'^{(s+1)^2}$ mit $M_s \cdot L = 0$ ist von der Gestalt

$$L = \begin{pmatrix} S^{(1)} \cdot l \\ \hline \vdots \\ \hline S^{(s+1)} \cdot l \end{pmatrix},$$

wobei $l \in P'^{s+1}$ die letzten $s+1$ Komponenten von L seien.

d) Für $i = 1, \dots, s+1$ sind die Grade der Polynome in $S^{(i)}$ höchstens $s + 1 - i$.

e) Die Grade der Polynome in S_s sind höchstens $s + 1$.

Beweis. Aussagen a) und b) sind nur die Umformulierungen von Satz 4.12, es ist also nichts zu zeigen.

Für die letzten $s+1$ Komponenten l eines $L \in P'^{(s+1)^2}$ mit $M_s \cdot L = 0$ muss nach a) $S_s \cdot l = 0$ gelten. Mit Hilfe von Satz 4.17 folgt dann c).
In A'_{s+1}, B_{s+1}, C_{s+1} und D_{s+1} kommen nur Polynome vom Grad kleiner oder gleich 1 vor. Für $S^{(s+1)}$ und $S^{(s)}$ ist d) nach Definition richtig. Aus der Rekursionsformel ergibt sich, dass bei jedem Rekursionsschritt die Grade der Polynome in $S^{(s-i)}$ höchstens um 1 gegenüber den Graden der Polynome in $S^{(s-i+1)}$ zunehmen. Damit ergibt sich d) und aus der Definition von S_s folgt schießlich e). □

Satz 4.17. **(Expansion)** Unter den Voraussetzungen von Satz 4.16 gilt für ein $l \in P'^{s+1}$ mit $S_s \cdot l = 0$

$$M_s \cdot \begin{pmatrix} S^{(1)} \cdot l \\ \hline \vdots \\ \hline S^{(s+1)} \cdot l \end{pmatrix} = 0$$

Beweis. Kann direkt aus der Form in Satz 4.16 a) abgelesen werden. □

Um sicherzustellen, dass die Normierungen in den Algorithmen aus 4.1 (nach Expansion) auch die Lösung kleinsten Grades über P gemäß Bemerkung 3.20 liefern, muss noch gezeigt werden, dass der Leitkoeffizient (bezüglich DegRevLex) der letzten Komponente 1 ist, wenn sie durch Einsetzen von $(x, y, z) := (1, v, w)$ in eine Lösung über P' überführt wird.

Satz 4.18. Sei die Lösung von Satz 3.11 durch Einsetzen von $(x, y, z) := (1, v, w)$ in eine Lösung über P' überführt. Dann gilt:

a) $\mathrm{LT}_\sigma(x_{s,s}) = v^{s \cdot (s+1)}$

b) $\mathrm{LC}_\sigma(x_{s,s}) = 1$

Beweis. In Satz 3.11 wird $x_{s,s}$ mit Hilfe der Determinante von $O^{(s)}_{0,0}$ berechnet. Macht man die L-förmige Sortierung rückgängig, so erkennt man, dass $O^{(s)}_{0,0}$ der Matrix $\tilde{M}_s$ entspricht, wobei $\tilde{M}_s$ aus M_s hervorgeht, indem man die letze Zeile und Spalte streicht. Da M'_s durch Vertauschen und Summieren einiger Zeilen aus M_s hervorgegangen ist, kann man anstelle von $\tilde{M}_s$ auch $\tilde{M}'_s$ verwenden. Die Zeilen und Spaltenumformungen, die in Satz 4.12 zu M' geführt haben, kann man auch mit $\tilde{M}'_s$ durchführen, ohne dabei ihre Determinante zu verändern. Insgesamt erkennt man, dass sich die Determinante von $O^{(s)}_{0,0}$ auch durch Berechnen der Determinante von $\tilde{M}'$ bestimmen lässt. Offensichtlich ist aber $\det \tilde{M}' = x^{s \cdot (s+1)} \cdot \det \tilde{S}_s$, da man $\det \tilde{M}'$ nach und nach spaltenweise entwickeln kann und in der Hauptdiagonalen nur geradzahlig $(s \cdot (s+1))$ viele $-x$ stehen.

Setzt man $(x,y,z) := (1,v,w)$ ein, so stehen nach Satz 4.16 e) in $\tilde{S}_s$ über P' nur Polynome vom Grad kleiner oder gleich $s+1$. Die Determinante der $s \times s$ Matrix $\tilde{S}_s$ über P' ist also ein Polynom f vom Grad kleiner oder gleich $s \cdot (s+1)$.
Formal geht das Gleichungssystem (4) aus dem Gleichungssystem (2) auch hervor, wenn man für $i,j = 1,\ldots,s$ die Unbestimmten $x_{i,j}$ mit $x_{j,i}$ vertauscht und $(x,y,z) := (v,1,w)$ einsetzt. Eine Lösung des Gleichungssystems (2) kann durch diese formale Operation also auch in eine Lösung des Systems (4) überführt werden, insbesondere auch die Lösung aus Satz 3.11. Dabei ist es für alle Diagonalelemente $x_{i,i}$ egal, ob man $(1,v,w)$ oder $(v,1,w)$ einsetzt, es entsteht in beiden Fällen die gleiche Lösung in P'.
Das Leitmonom (bezüglich DegRevLex) von $x_{s,s}(x,y,z) \in P$ ist nach Theorem 3.19 $x^{s^2+s}y^s$. Er geht in das Monom v^{s^2+s} über, wenn man $(v,1,w)$ einsetzt. Also gibt es in f ein Monom v^{s^2+s}. Dieses muss aufgrund seines Grades das Leitmonom sein. $\square$

Man kann also alle Algorithmen aus 4.1 auch nach Einsparung einer Unbestimmten anwenden. Überführt man das Ergebnis mit Hilfe von Satz 4.14 nach P, so erhält man genau den gleichen Vektor, der sich ergäbe, wenn man gleich in P rechnen würde.

4.4 Modulare Methode

Über endlichen Körpern sind die meisten Algorithmen deutlich schneller. Dies liegt hauptsächlich an der Art der Implementierung. Multiplikationen können in endlichen Körpern mit Hilfe von Multiplikationstabellen ausgeführt werden, was die Bearbeitungsdauer für Multiplikationen in den Bereich von Additionen bringt. Auch in CoCoA sind Berechnungen über endlichen Körpern optimiert implementiert. Dies soll im Folgenden ausgenutzt werden.

Definition 4.19. Zu jedem $x \in \mathbb{Z}_p$ existiert genau ein $y \in \mathbb{Z}$ mit $y \in]-\frac{p}{2}, \frac{p}{2}]$ und $x \equiv y \bmod p$. Definiere ZPQ : $\mathbb{Z}_p \to \mathbb{Q}$ durch ZPQ$(x) := y$.
Analog sei ZPQ : $\mathbb{Z}_p[v,w] \to \mathbb{Q}[v,w]$ bzw. ZPQ : $\mathbb{Z}_p[x,y,z] \to \mathbb{Q}[x,y,z]$ koeffizientenweise definiert. ZPQ von einem Vektor bilde diesen komponentenweise ab.

Satz 4.20. Seien $p,q \in \mathbb{Z}$ mit $\mathrm{ggT}(p,q) = 1$ sowie $f^{(p)}$ ein Polynom über $\mathbb{Z}_p$ und $f^{(q)}$ ein Polynom über $\mathbb{Z}_q$. Betrachte die folgenden Anweisungen:

(1) Finde mit Hilfe des erweiterten Euklidschen Algorithmus zwei Zahlen $c,d \in \mathbb{Z}$ mit $cp \equiv 1 \bmod q$ und $dq \equiv 1 \bmod p$.

(2) Bestimme den gemeinsamen Träger $T \in \mathbb{T}^t$ von $f^{(p)}$ und $f^{(q)}$, mit einem geeigneten $t \in \mathbb{N}$.

(3) Wähle $C^{(p)} \in \mathbb{Z}_p^t$ und $C^{(q)} \in \mathbb{Z}_q^t$ derart, dass $C^{(p)} \cdot T = f^{(p)}$ und $C^{(q)} \cdot T = f^{(q)}$.

(4) Für $i = 1, \dots, t$ setze $C_i^{(pq)} := c \cdot p \cdot C_i^{(q)} + d \cdot q \cdot C_i^{(p)} \bmod pq$.

(5) Gib das Polynom $C^{(pq)} \cdot T$ über $\mathbb{Z}_{pq}$ zurück.

Dies ist ein Algorithmus, welcher eine Abbildung definiert, die zwei Polynomen $f^{(p)}$ und $f^{(q)}$ ein Polynom $f^{(pq)}$ aus dem Polynomring über $\mathbb{Z}_{pq}$ zuordnet, welches koeffizientenweise modulo p bzw. q genommen $f^{(p)}$ bzw. $f^{(q)}$ ergibt.
Vektoren sollen komponentenweise abgebildet werden, der erste Schritt muss hierbei natürlich nur einmal erfolgen.

Beweis. Folgt koeffizientenweise aus dem chinesischen Restsatz. □

Satz 4.21. Sei der Körper K nun $\mathbb{Q}$ und M die Matrix M_s, M_s' oder S_s über P bzw. P' definiert. Betrachte die folgenden Anweisungen:

(1) Setze $N := \emptyset$

(2) Wähle eine Primzahl $p \in \mathbb{Z} \setminus N$ und füge sie zu N hinzu

(3) Bilde aus M komponentenweise $M^{(p)}$, indem alle auftretenden Koeffizienten modulo p genommen werden.

(4) Löse das zu $M^{(p)}$ gehörige Gleichungssystem im Polynomring über $\mathbb{Z}_p$ und erhalte die Lösung $l^{(p)}$.

(5) Teile $l^{(p)}$ durch den Leitkoeffizienten der letzten Komponente.

(6) Setze $n := p$ und $l^{(n)} := l^{(p)}$

(7) Führe die Schritte (8)-(9) durch, bis $M \cdot \mathrm{ZPQ}(l^{(n)}) = 0$

(8) Führe die Schritte (2)-(5) aus.

(9) Rekombiniere $l^{(n)}$ und $l^{(p)}$ mit Hilfe von Satz 4.20. Ersetze $l^{(n)}$ durch das Ergebnis und n durch $n \cdot p$.

(10) Gib $l := \mathrm{ZPQ}(l^{(n)})$ zurück.

Dies ist ein Algorithmus, der einen Vektor l liefert mit $M \cdot l = 0$.

Beweis. Die Korrektheit ist klar, da die Eigenschaft des Ergebnisses genau der Abbruchbedingung in Schritt (7) entspricht. Zu jedem Zeitpunkt gilt $\mathrm{ggT}(p, n) = 1$, da jede Primzahl genau einmal verwendet wird. Satz 4.20 ist also stets anwendbar. Zu jedem Zeitpunkt gilt auch komponentenweise

$$M^{(n)} \cdot l^{(n)} \equiv 0 \bmod n.$$

Über P gibt es nach Satz 3.11 eine Lösung von M, die nur Koeffizienten aus $\mathbb{Z}$ besitzt und deren letzte Komponente den Leitkoeffizienten 1 hat. Auch für die über P' definierten Matrizen folgt dies mit Satz 4.18. Die Lösung kleinsten Grades müsste diese nach Satz 3.10 komponentenweise teilen, insbesondere hätte sie auch nur Koeffizienten in $\mathbb{Z}$ und 1 als den Leitkoeffizienten der letzten Komponente.
Ein solcher Lösungsvektor l mit $M \cdot l = 0$ liefert modulo n auch stets einen Vektor $l^{(n)}$ mit $M^{(n)} \cdot l^{(n)} = 0$. Dabei bleibt der Leitkoeffizient 1 der letzen Komponente erhalten. Ist $\frac{n}{2}$ größer als der größte auftretende Koeffizient in l, so ändert sich beim Bilden der Restklassen modulo n keiner der Koeffizienten, oder anders ausgedrückt: $\mathrm{ZPQ}(l^{(n)}) = l$. Da es eine Lösung l gibt und n im Algorithmus ständig wächst, wird irgendwann dieser Punkt erreicht und der Algorithmus endet.
Schritt (5) stellt sicher, dass beim Rekombinieren stets die Restklassen der angepeilten Lösung l verwendet werden. Sie haben die Eigenschaft, dass der Leitkoeffizient der letzten Komponente 1 ist. Auch das Ergebnis von Satz 4.20 hat diese Eigenschaft.
Schritt (2) kann ausgeführt werden, weil in M_s und M'_s nur Koeffizienten aus $\mathbb{Z}$ vorkommen. Beim Bilden von S_s aus M'_s wird nie dividiert, also besitzt auch S_s nur ganzzahlige Koeffizienten. □

Bei diesem Algorithmus dauert das Rekombinieren in Schritt (9) relativ lange. Wenn man erwartet modulo vieler Primzahlen rechnen zu müssen, bis n groß genug ist, ist es sinnvoll, gleich mehrere modulare Lösungen auf einmal zu rekombinieren, bevor man in kleinen Schritten das n weiter vergrößert. Diese Verbesserung erfordert eine Erweiterung von Satz 4.20:

Satz 4.22. Für $i = 1, \ldots, k$ seien $p_i \in \mathbb{Z}$ verschiedene Primzahlen und $f^{(p_i)}$ Polynome über $\mathbb{Z}_{p_i}$. Betrachte die folgenden Anweisungen:

(1) Setze $n := p_1 \cdot \ldots \cdot p_k$ und für $i = 1, \ldots, k$ setze $q_i := \frac{n}{p_i}$.

(2) Finde mit Hilfe des erweiterten Euklidschen Algorithmus Zahlen $c_i \in \mathbb{Z}$ mit $c_i \cdot q_i \equiv 1 \bmod p_i$ für $i = 1, \ldots, k$.

(3) Bestimme den gemeinsamen Träger $T \in \mathbb{T}^t$ von $f^{(p_i)}$ mit einem geeigneten $t \in \mathbb{N}$.

(4) Wähle $C^{(p_i)} \in \mathbb{Z}_{p_i}^t$ derart, dass $C^{(p_i)} \cdot T = f^{(p_i)}$ für $i = 1, \ldots, k$.

(5) Für $j = 1, \ldots, t$ setze $C_j^{(n)} := \sum_{i=1}^k c_i \cdot q_i \cdot C_j^{(p_i)} \bmod n \in \mathbb{Z}_n$.

(6) Gib das Polynom $C^{(n)} \cdot T$ über $\mathbb{Z}_n$ zurück.

Dies ist ein Algorithmus, der eine Abbildung definiert, die den Polynomen $f^{(p_i)}$ ein Polynom $f^{(n)}$ aus dem Polynomring über $\mathbb{Z}_n$ zuordnet, welches koeffizientenweise modulo p_i genommen $f^{(p_i)}$ ergibt.

Vektoren soll komponentenweise abgebildet werden, die Schritte (1) und (2) müssen hierbei natürlich nur einmal erfolgen.

Beweis. Folgt koeffizientenweise aus dem chinesischen Restsatz. □

Nun kann man den Algorithmus aus Satz 4.21 wie folgt modifizieren:

Satz 4.23. Sei der Körper K nun $\mathbb{Q}$ und M die Matrix M_s, M'_s oder S_s über P bzw. P' definiert. Für ein $k \in \mathbb{N}$ betrachte die folgenden Anweisungen:

(1) Setze $N := \emptyset$

(2) Für $i = 1, \ldots, k$ wähle verschiedene Primzahlen $p_i \in \mathbb{Z}$ und füge sie zu N hinzu.

(3) Bilde aus M komponentenweise $M^{(p_i)}$, indem alle auftretenden Koeffizienten modulo p_i genommen werden.

(4) Löse die zu $M^{(p_i)}$ gehörigen Gleichungssysteme im Polynomring über $\mathbb{Z}_{p_i}$ und erhalte die Lösungen $l^{(p_i)}$.

(5) Teile alle $l^{(p_i)}$ durch die Leitkoeffizienten ihrer letzten Komponente.

(6) Setze $n := p_1 \cdot \ldots \cdot p_k$. Rekombiniere die $l^{(p_i)}$ mit Hilfe von Satz 4.22 und nenne das Ergebnis $l^{(n)}$.

(7) Führe die Schritte (8)-(12) durch, bis $M \cdot \mathrm{ZPQ}(l^{(n)}) = 0$

(8) Wähle eine Primzahl $p \in \mathbb{Z} \setminus N$ und füge sie zu N hinzu

(9) Bilde aus M komponentenweise $M^{(p)}$, indem alle auftretenden Koeffizienten modulo p genommen werden.

(10) Löse das zu $M^{(p)}$ gehörige Gleichungssystem und erhalte die Lösung $l^{(p)}$.

(11) Teile $l^{(p)}$ durch den Leitkoeffizienten der letzten Komponente.

(12) Wende Satz 4.20 auf $l^{(n)}$ und $l^{(p)}$ an. Ersetze $l^{(n)}$ durch das Ergebnis und n durch $n \cdot p$.

(13) Gib $l := \mathrm{ZPQ}(l^{(n)})$ zurück.

Dies ist ein Algorithmus, der einen Vektor l liefert mit $M \cdot l = 0$.

Beweis. Dieser Algorithmus unterscheidet sich vom Algorithmus aus Satz 4.21 nur darin, dass zuerst für k Primzahlen modulare Lösungen gefunden werden, die dann auf einmal rekombiniert werden. An der Endlichkeit und Korrektheit ändert sich hierbei nichts, der Beweis würde analog zu dem vom Satz 4.21 gehen. □

Bemerkung 4.24. Da Satz 4.21 ein Spezialfall von Satz 4.23 ist, wird in der Praxis nur der letztere Algorithmus benötigt. Man könnte natürlich auch in Schritt (12) Satz 4.22 statt Satz 4.20 anwenden und die Schritte (8)-(11) immer für mehrere Primzahlen auf einmal ausführen. Dadurch würde aber nur die Schrittweite bei der Annäherung an ein genügend großes n erhöht, was in der Praxis von geringer Bedeutung ist, da man ggf. ein größeres k für den ersten Rekombinierungsdurchlauf wählen kann.
Eine (grobe) Abschätzung für k erhält man zum Beispiel, wenn man den Fall s schon berechnet hat dadurch, dass der Fall $s+1$ auf jeden Fall ein größeres k benötigt.

5 Implementierung in CoCoA

In diesem Kapitel werden Grundkenntnisse der zum Computeralgebraprogramm CoCoA gehörigen Programmiersprache CoCoAL vorausgesetzt. In Anlehnung an die CoCoA-Hilfe werden im Folgenden Variablen stets in der Form

```
Name : Typ
```

beschrieben, sowie Funktionen in folgender Form deklariert:

```
Funktionsname(Arg#1, ..., Arg#n) : Typ des Rückgabewertes
```

Falls eine Funktion keinen Rückgabewert besitzt, wird kein Typ angegeben. Im Folgenden soll zunächst die Funktionsweise der CoCoA-Programme in der Datei `solve.coc` beschrieben werden. Eine genauere Dokumentation des ebenfalls zu dieser Arbeit gehörigen Pakets `chinese.pkg` findet sich am Ende dieses Abschnitts. Auf eine Beschreibung der Bareiss-Algorithmen wird ganz verzichtet, da dies bereits Teil der Diplomarbeit [K] war. Im Unterverzeichnis `code/bareiss` findet sich jedoch außer dem Programmcode auch eine genaue Beschreibung.
Alle Funktionen der CoCoA-Datei `solve.coc` und `chinese.pkg` wurden für CoCoA 4.1 geschrieben und auch nur mit dieser Version getestet.

5.1 Grundlegendes und Initialisierung

Der Benutzer sollte zu Beginn die globale Variable `MEMORY.BASEDIR` auf das Basisverzeichnis der beiliegenden Programme setzen. Soll zum Beispiel alles von der CD gestartet werden, so wäre dies mit dem Befehl

```
MEMORY.BASEDIR:="/cdrom/code/";
```

zu erreichen. Wird diese Variable nicht gesetzt, so wird sie beim Einlesen von `solve.coc` auf einen Leerstring gesetzt. Dies hat zur Folge, dass CoCoA alle Dateien ausgehend von seinem Arbeitsverzeichnis sucht. Wird CoCoA aus dem Unterverzeichnis `code` heraus gestartet, so sollte dies kein Problem sein. Anschließend muss mit dem Befehl

```
<< MEMORY.BASEDIR+"solve.coc";
```

das Hauptprogramm geladen werden.

Da die meisten Algorithmen vom Parameter s abhängen, wird dieser in der globalen Variablen `MEMORY.S:INT` gehalten. Sollte sie nicht bereits definiert sein, wird sie beim Laden von `solve.coc` standardmäßig auf 1 gesetzt.

Als nächstes muss der gewünschte Ring generiert und in ihn gewechselt werden. Dies geschieht mit einem der folgenden Befehle:

```
Use RingXYZ::=Q[x,y,z];
Use RingVW::=Q[v,w];
Use RingXYZ::=Z/(32003)[x,y,z];
Use RingVW::=Z/(32003)[v,w];
```

Natürlich kann anstelle von 32003 auch jede andere Primzahl innerhalb der Grenzen von CoCoA verwendet werden.

Anschließend muss die Koeffizientenmatrix für ein s initialisiert werden. Hierfür stehen die internen Routinen

```
InitKMatrixXYZ()
```

beziehungsweise

```
InitKMatrixVW()
```

zur Verfügung. Sie generieren abhängig von `MEMORY.S` die Koeffizientenmatrix für das Gleichungssystem (2) bzw. (4). Dabei werden die Hilfsroutinen

```
F(I:INT,J:INT) : LIST
```

und

```
H(I:INT,J:INT) : LIST
```

verwendet, die die zu $f_{i,j}$ bzw. $h_{i,j}$ gehörige Zeile der Koeffizientenmatrix erzeugen. Die entstandene Koeffizientenmatrix entspricht dem M_s aus Proposition 4.8 und wird in die globale Variable `MEMORY.KMatrix` gespeichert.

```
InitLMatrix()
```

erzeugt die in Definition 4.11 beschriebene Matrix M'_s aus `MEMORY.KMatrix` und speichert sie in die globale Variable `MEMORY.LMatrix`.

```
InitBMatrix()
```

teilt `MEMORY.LMatrix` zur weiteren Bearbeitung in $s+1$ mal $s+1$ Blöcke aus $(s+1)\times(s+1)$-Matrizen auf und speichert sie in der globalen Variablen `MEMORY.BMatrix`.

Die Funktionen

```
InitMMatrixXYZ()
InitMMatrixVW()
```

wandeln `MEMORY.BMatrix` in die Matrix M' aus Satz 4.12 a) bzw. Satz 4.16 a) um und speichern das Ergebnis in die globale Variable `MEMORY.MMatrix`.

Schließlich speichert

```
InitSMatrix()
```

die in der letzten Blockzeile und -spalte von `MEMORY.MMatrix` stehende $(s+1)\times(s+1)$-Matrix in die globale Variable `MEMORY.SMatrix`.

Zur Vereinfachung wurden folgende Funktionen definiert, die alle vorangegangenen Matrizen auf einmal erzeugen:

```
Init(S:INT)
InitVW(S:INT)
InitXYZ(S:INT)
```

Dabei ruft `Init(S)` abhängig vom jeweils aktuellen Ring die entsprechende Initialisierungsroutine `InitXYZ(S)` oder `InitVW(S)` auf. Diese setzen beide zunächst `MEMORY.S:=S` und führen anschließend in dem ihnen zugehörigen Ring alle oben genannten Routinen zur Matrizenerzeugung aus. Insgesamt erhält man also durch die Initialisierung folgende globale Objekte:

```
MEMORY.S : INT
MEMORY.KMatrix : MAT
MEMORY.LMatrix : MAT
MEMORY.BMatrix : MAT
MEMORY.MMatrix : MAT
MEMORY.SMatrix : MAT
```

5.2 Lösungsalgorithmen

Der in Satz 4.1 vorgestellte leicht modifizierter Gauss Algorithmus, der eine gegebene Matrix in Zeilenstufenform überführt und dabei rationale Funktionen vermeidet, wird durch die Funktion

```
Gauss(M:MAT) : MAT
```

implementiert. Sie wird vom Benutzer eigentlich nicht direkt aufgerufen, sondern von der Funktion

```
GaussSolve(M:MAT) : LIST
```

verwendet, die den Algorithmus aus Satz 4.3 darstellt. Als Argumente sind `MEMORY.KMatrix`, `MEMORY.LMatrix` und `MEMORY.SMatrix` sinnvoll. Wird das Argument weggelassen, verwendet sie `MEMORY.SMatrix` als Vorgabewert.

Die Implementierung des Algorithmus aus Satz 4.4 ist durch die Funktion

```
SyzSolve(M:MAT) : LIST
```

vorhanden. Die gleichen Matrizen wie für `GaussSolve(M)` sind als Argumente sinnvoll. Auch hier ist der Standardwert `MEMORY.SMatrix`, falls nichts anderes angegeben wird.

Die beiden Funktionen

```
BarSolve(M:MAT) : LIST
MalDichSolve(M:MAT) : LIST
```

implementieren Satz 4.5. Dabei rufen sie aus dem Paket `linalg.pkg` die zugehörigen Funktionen

```
B2DMDMDFB(M:MAT) : MAT
MalDichHM(M:MAT) : MAT
```

auf und wandeln das Ergebnis, eine diagonalisierte Matrix, in einen Lösungsvektor um und geben diesen zurück. Näheres zu den aufgerufenen Funktionen findet man in [K]. `M` muss vollen Rang besitzen, da `B2DMDMDFB` und

`MaDichHM` als Argument nur solche Matrizen erwarten. Die drei bereits genannten Matrizen können daher nicht direkt eingesetzt werden. Da bei `MEMORY.KMatrix` die Summe aller Zeilen eine Nullzeile ergibt, kann etwa die letzte weggelassen werden. Bei `MEMORY.LMatrix` und `MEMORY.SMatrix` ist die letzte Zeile schon eine Nullzeile, sie kann also ignoriert werden. Das Abschneiden der letzten Zeile kann in CoCoA mit dem Aufruf

```
Mat([M[I]|I In 1..(Len(M)-1)]);
```

erfolgen. Hierbei ist `M` eine Matrix. Sinnvolle Argumente sind also wieder `MEMORY.KMatrix`, `MEMORY.LMatrix` und `MEMORY.SMatrix`, von denen mit dem obigen Befehl die letzte Zeile abgeschnitten wurde. Wird keine Matrix angegeben, so ist der Vorgabewert `MEMORY.SMatrix` mit gestrichener letzten Zeile.

Mit Hilfe des Pakets `chinese.pkg` kann man alle bereits beschriebenen Algorithmen modular ausführen und ihre Ergebnisse rekombinieren.

```
ChinGaussSolve(M:MAT, N:INT) : LIST
ChinSyzSolve(M:MAT, N:INT) : LIST
ChinBarSolve(M:MAT, N:INT) : LIST
ChinMalDichSolve(M:MAT, N:INT) : LIST
```

rufen die entsprechenden Funktionen für die Matrix `M` mit Hilfe des Pakets `chinese.pkg` auf. Für die Bareiss-Verfahren muss `M` vollen Rang besitzen. Wird die Matrix nicht angegeben, so wird standardmäßig `MEMORY.SMatrix` (ggf. ohne letzte Zeile) verwendet. Der optionale Parameter `N` gibt dabei die Anzahl der endlichen Körper an, in denen vor dem ersten Rekombinierungsschritt eine Lösung berechnet werden soll.
Das Paket `chinese.pkg` benötigt zusätzlich eine Testfunktion, mit der entschieden wird, ob schon eine Lösung gefunden wurde. Die Funktionen

```
GaussCheck(A:MAT) : BOOL
SyzCheck(A:MAT) : BOOL
BarCheck(A:MAT) : BOOL
MalDichCheck(A:MAT) : BOOL
```

erfüllen diesen Zweck. Ihnen wird stets die Matrix als Argument übergeben, welche zur Lösungsfindung der zugehörigen `ChinSolve`-Funktion verwendet wurde. Es wird das Produkt dieser Matrix mit der momentanen Lösung der modularen Berechnung gebildet. Kommt dabei ein Nullvektor heraus, geben diese Funktionen `TRUE` zurück, ansonsten `FALSE`.

5.3 Sonstige Algorithmen

```
ExpandSolution(L:LIST) : LIST
ExpandSolutionXYZ(L:LIST) : LIST
ExpandSolutionVW(L:LIST) : LIST
```

Die erste Funktion ruft abhängig vom aktuellen Ring eine der beiden anderen auf. Diese erwarten als Argument eine Lösung von `MEMORY.SMatrix` und erzeugen eine Lösung von `MEMORY.KMatrix`, wie in Satz 4.13 bzw. Satz 4.17 beschrieben.

```
VW2XYZ(L:LIST) : LIST
```

überführt eine Lösung für `MEMORY.KMatrix` oder `MEMORY.SMatrix` aus dem Polynomring `RingVW` in eine Lösung aus `RingXYZ`, wie in Satz 4.14 beschrieben. Dabei hat der Benutzer darauf zu achten, dass der Zielring `RingXYZ` auch existiert und den gleichen Grundkörper besitzt wie `RingVW`. `L` muss ein Objekt aus `RingVW` sein, das Ergebnis lebt in `RingXYZ`.

Die Funktionen

```
BlockPrint() : STRING
BlockLatexPrint() : STRING
```

geben `MEMORY.KMatrix` als einen übersichtlich ausgerichteten String bzw. als LaTeX-Quellcode aus.

Schließlich existieren noch die die beiden Funktionen

```
Save(L:LIST,D:STRING)
Load(D:STRING) : LIST
```

um eine Liste `L` von Polynomen (die Lösung) in eine Datei mit dem Namen `D` zu speichern. Der Dateiname ist optional, wird er weggelassen, so wird ein Standardname verwendet, der sich aus "lsg", dem aktuellen `MEMORY.S` und ".coc" zusammensetzt. Die Datei mit dem Standardnamen wird in das Unterverzeichnis `loesung` des in `MEMORY.BASEDIR` angegebenen Verzeichnisses gespeichert bzw. von dort geladen. Selbstverständlich ist darauf zu achten, dass man dort auch Lese- bzw. Schreibrechte besitzt. Insbesondere kann auf die CD natürlich nicht geschrieben werden. In der Datei steht nicht einfach der Lösungsvektor, CoCoA hätte beim Einlesen von Polynomen mit zu großem Träger Probleme. Man umgeht dieses Problem, indem man die Polynome der Liste zuerst in Listen aus Monomen umwandelt und diese geschachtelte Liste dann speichert. Beim erneuten Einlesen müssen dann die Listen jeweils summiert werden, um wieder in Polynome verwandelt zu werden. Auch diesen Vorgang übernehmen die beiden Funktionen, daher kann es schon eine Weile dauern bis ein Polynom eingelesen wird. Weiter sei vor dem Speicherhunger von CoCoA gewarnt, auf dem Testsystem mit 512MB ist beim Einlesen der Lösung für den Fall $s = 8$ der Speicher schon recht knapp geworden.

5.4 Das Paket `chinese.pkg`

Für die modulare Methode nach 4.4 wurde auf eine möglichst flexible Implementierung Wert gelegt. Normalerweise wird aus dem Paket `chinese.pkg` nur die Funktion

```
Run(FName:STRING,FArg:LIST,CFName:STRING,CFArg:LIST,K:INT):OBJECT
```

benutzt. Sie implementiert Satz 4.23 für den Lösungsalgorithmus `FName`, der im Polynomring über $\mathbb{Q}$ mit den Argumenten `FArg` aufgerufen würde. `K` gibt die Anzahl der Körper an, in denen gerechnet werden soll, bevor zum ersten

Mal rekombiniert wird. `CFName` ist der Name der Prüfroutine und `CFArg` die Liste ihrer Argumente. Sie wird unter `Init` genauer beschrieben.

Das Paket kann jedoch auch interaktiv verwendet werden. Dazu wird zuerst mit

```
Init(FName:STRING,FArg:LIST,CFName:STRING,CFArg:LIST,K:INT):OBJECT
```

`chinese.pkg` für den Lösungsalgorithmus `FName` initialisiert, der im Polynomring über $\mathbb{Q}$ mit den Argumenten `FArg` aufgerufen würde. Diese Funktion startet mit `Execute(FName,FArg,P)` die Berechnung für `K` Primzahlen `P` und rekombiniert die Ergebisse. Das Resultat wird zurückgegeben.
`CFName` ist dabei eine Prüfroutine die `TRUE` zurückgibt, wenn in keinem weiteren endlichen Körper gerechnet werden soll und `FALSE` wenn in weiteren endlichen Körpern gerechnet werden soll. In ihr kann mit `PKG.Chinese.Result` auf das aktuelle Ergebnis zurückgegriffen werden, welches im Abbruchfall zurückgegeben würde. `CFArg` ist die Liste der Argumente, die für die Prüfroutine zusätzlich benötigt wird.

Nachdem das Paket initialisiert ist und der erste Rekombinierungsschritt ausgeführt wurde existiert in `PKG.Chinese.Result` ein Zwischenergebnis. Mit

```
Check() : BOOL
```

wird die an `Init` übergebene Prüfroutine mit der Liste ihrer Argumente aufgerufen und ihr Ergebnis zurückgegeben.

```
Next() : OBJECT
```

führt die Berechnung in einem weiteren Körper aus und gibt das rekombinierte Ergebnis zurück. Sie kann beliebig oft ausgeführt werden, es wird dabei stets eine neue Primzahl ausgewählt.

```
About()
Functions() : LIST
Help(FunktionName:STRING)
```

sind Funktionen, die Informationen über das Paket und seine Funktionen ausgeben. Insbesondere kann mit `Help(FunktionName)` eine Online-Hilfe der meisten Funktionen ausgegeben werden.

Die nachfolgenden Funktionen sind Arbeitsroutinen und werden normalerweise nicht vom Benutzer aufgerufen. Sie werden nur aus Gründen der Vollständigkeit aufgeführt.

```
InitPrimes()
```

Erzeugt die Liste aller in CoCoA für endliche Körper verwendbaren Primzahlen in der globalen Variablen `PKG.Chinese.Primes` und setzt die aktuelle Primzahlposition `PKG.Chinese.PrimePos` auf die letzte Komponente dieser Liste.

```
GetFirstPrime() : INT
GetNextPrime() : INT
```

Die erste Funktion gibt die erste Primzahl aus `PKG.Chinese.Primes` zurück und setzt die Primzahlposition `PKG.Chinese.PrimePos` auf das letzte Element dieser Liste. Die zweite Funtion vermindert die Primzahlposition um Eins und gibt die Primzahl dieser Position aus `PKG.Chinese.Primes` zurück. Wenn die erste Komponente dieser Liste schon erreicht war, wirft sie einen Fehler aus. Da es innerhalb der Grenzen von CoCoA 3512 verschiedene Primzahlen gibt, ist es eher unwahrscheinlich, dass diese Grenze jemals erreicht wird.

```
MCombine([O_1:OBJECT,P_1:INT],...,[O_n:OBJECT,P_n:INT]) : OBJECT
MCombine(LIST of [O:OBJECT,P:INT]) : OBJECT
```

rekombiniert die CoCoA-Objekt `O_i` wie in Satz 4.22 beschrieben. Zulässige Datentypen sind `POLY`, `INT` sowie Listen und Matrizen aus diesen Typen. Dabei leben die Objekte in $\mathbb{Z}$, `P_i` gibt an, in welchem endlichen Körper die Koeffizienten aufgefasst werden sollen. Die Paare `[O_i,P_i]` können einzeln, oder in der zweiten Version als Liste übergeben werden. Diese Funktion ruft rekursiv die interne Funktion

```
_MCombine(A:LIST,C:LIST,L:INT) : OBJECT
```

auf, welche die eigentliche Arbeit macht. Hierbei ist `A` die Liste der Argumente von `MCombine`, `C` die Liste der Ergebnisse des erweiterten Euklidschen Algorithmus und `L` die Länge der beiden Listen.

```
Combine([FP:OBJECT,P:INT],[FQ:OBJECT,Q:INT]) : OBJECT
```

rekombiniert die CoCoA-Objekt `FP` und `FQ` wie in Satz 4.20 beschrieben. Zulässige Datentypen sind `POLY`, `INT` sowie Listen und Matrizen aus diesen Typen. Dabei leben die Objekte in $\mathbb{Z}$, `P` bzw. `Q` gibt an, in welchem endlichen Körper die Koeffizienten aufgefasst werden sollen. Diese Funktion ruft rekursiv die interne Funktion

```
_Combine([FP:OBJECT,P:INT],[FQ:OBJECT,Q:INT],C:INT,D:INT) : OBJECT
```

auf, welche die eigentliche Arbeit macht. Hierbei sind `C` und `D` die Ergebnisse des erweiterten Euklidschen Algorithmus wie im ersten Schritt von Satz 4.20 beschrieben.

```
ExtEuklid(A:INT,B:INT) : LIST
```

implementiert den erweiterten Euklidschen Algorithmus und gibt eine Liste `[C:INT,D:INT,E:INT]` zurück, bei der `E` der größte gemeinsame Teiler von `A` und `B` ist sowie $\mathtt{A} \cdot \mathtt{C} + \mathtt{B} \cdot \mathtt{D} = \mathtt{E}$ gilt.

```
Execute(FName:STRING,FArg:LIST,P:INT) : OBJECT
```

führt den Lösungsalgorithmus `FName` mit den Argumenten `FArg` im Polynomring über $\mathbb{Z}/(P)$ aus und gibt das Ergebnis von `FName` zurück. Das Ergebnis lebt jedoch im Polynomring über $\mathbb{Z}$, was durch die interne Funktion `_ZPQ` erreicht wird.

```
IndetString(RingName:STRING) : STRING
```

gibt einen String aus der die Unbestimmten des Rings mit dem Namen `RingName` beschreibt. Wird kein Ring angegeben, so wird der aktuelle Ring verwendet.

```
_ZPQ(A:OBJECT) : OBJECT
_QZP(A:OBJECT) : OBJECT
```

überführen das Objekt `A` von $\mathbb{Z}$ nach $\mathbb{Z}/(p)$ bzw. von $\mathbb{Z}/(p)$ nach $\mathbb{Z}$. Sie entsprechen den internen CoCoA-Funktionen `QZP` und `ZPQ`, lassen jedoch einige Sicherheitsprüfungen weg, die automatisch erfüllt sind. Sie stellen gewissermaßen eine optimierte Version dieser Funtionen dar.

```
Initialize()
```

ist die Initialisierungsroutine des Pakets. Sie wird automatisch ausgeführt, bevor eine Funktion aus `chinese.pkg` aufgerufen wird. Dabei wird der globale Paketspeicher `PKG.Chinese` angelegt bzw. gelöscht und die Primzahlliste `PKG.Chinese.Primes` mit `InitPrimes()` generiert.

Bei der Benutzung von `chinese.pkg` über die Funktion `Run` wird während der Ausführung für jeden endlichen Körper, in dem vor dem ersten Rekombinierungsschritt gerechnet wird, ein "`*`" und für jeden weiteren endlichen Körper ein "`.`" ausgegeben. Auf diese Weise erhält man einen Überblick, in wie vielen Körpern gerechnet wurde und in wie weit eine Beschleunigung durch einen anderen Wert für den ersten Schritt möglich ist.

6 Vergleich der Algorithmen

In diesem Abschnitt sollen die verschiedenen implementierten Algorithmen verglichen werden. Das verwendete Testsystem war ein AMD AthlonXP 1800 mit 512MB RAM und 2GB Swap unter SuSE-Linux 8.0. Alle gemessenen Werte beziehen sich auf CoCoA 4.1, welches auf einem System dieser Leistungsklasse installiert ist. Da CoCoA in der vorliegenden Version immer langsamer zu werden scheint, je länger man damit arbeitet, wurde das Programm vor jeder Berechnung neu gestartet, um aussagekräftige Ergebnisse zu erhalten. Alle Messwerte sind auf ganze Sekunden gerundet, falls nichts Anderes angegeben wird. Bearbeitungszeiten unter einer Sekunde werden nicht aufgeführt.
Um es dem Leser zu erleichtern, eigene Tests durchzuführen, werden die jeweils durchgeführten CoCoA-Befehle für die einzelnen Messserien angegeben. Dabei wird davon ausgegangen, dass `MEMORY.BASEDIR` bereits gesetzt und die Datei `solve.coc` geladen wurde. Die weitere Initialisierung benötigt erst im Fall $s = 9$ mehr als 1 Sekunde. Da die hierfür benötigte Zeit auch für größeres s nur leicht ansteigt, kann sie vernachlässigt werden und wird daher nicht gemessen.

6.1 Berechnung in 3 Unbestimmten mit `MEMORY.KMatrix`

In der ersten Vergleichsserie soll `MEMORY.KMatrix` in $\mathbb{Q}[x,y,z]$ mit den Basisalgorithmen gelöst werden. Da die beiden Bareiss-Verfahren Matrizen von vollem Rang erwarten, wird bei ihnen die letzte Zeile weggelassen. Für `GaussSolve` und `SyzSolve` verändert sich die Bearbeitungszeit nur unwesentlich, wenn man die letzte Zeile streicht, es wird also dort mit der vollen Matrix gerechnet.
Die Initialisierung erfolgt durch

```
Use RingXYZ::=Q[x,y,z];
Init(2);
M:=Mat([MEMORY.KMatrix[I]|I In 1..(Len(MEMORY.KMatrix)-1)]);
```

Anschließend wird einer der folgenden Befehle verwendet:

```
Time T:=GaussSolve(MEMORY.KMatrix);
Time T:=SyzSolve(MEMORY.KMatrix);
Time T:=BarSolve(M);
Time T:=MalDichSolve(M);
```

Tabelle 1: Berechnung von `MEMORY.KMatrix` in $\mathbb{Q}[x,y,z]$

S	1-2	3	4	5	6	7	8
`GaussSolve`	<1	71	11217				
`SyzSolve`	<1	<1	7	154	3452		
`BarSolve`	<1	5	67	1010	9140		
`MalDichSolve`	<1	3	45	613	6078		

Die gleichen Berechnungen werden nun über einem endlichen Körper ausgeführt. Für das hier verwendete Beispiel $\mathbb{Z}/(32003)$ wird bei der Initialisierung daher

```
Use RingXYZ::=Z/(32003)[x,y,z];
```

anstelle des bisherigen `RingXYZ` definiert. Für die Berechnung wird einer der oben genannten Befehle aufgerufen.

Tabelle 2: Berechnung von `MEMORY.KMatrix` in $\mathbb{Z}/(32003)[x,y,z]$

S	1-2	3	4	5	6	7	8
`GaussSolve`	<1	7	120	2252			
`SyzSolve`	<1	<1	1	8	63	353	1804
`BarSolve`	<1	3	23	221	1865	12794	
`MalDichSolve`	<1	2	14	123	1101	7701	

Man erkennt, das über endlichen Körpern alle Berechnungen deutlich schneller ablaufen. Wie bereits in 4.4 beschrieben, kann man die Algorithmen auch modular ausführen und mit Hilfe des chinesischen Restsatzes die Ergebnisse zu einer Lösung über $\mathbb{Z}$ rekombinieren. Die Hoffnung dabei ist, dass trotz mehrfacher Berechnung über endlichen Körpern und Rekombinierung der Ergebnisse die Gesammtbearbeitungsdauer noch unterhalb der in $\mathbb{Q}[x, y, z]$ benötigten Zeit liegt.
Mit Hilfe des Pakets `chinese.coc` wurden vier weitere Algorithmen definiert, die genau auf diese Weise modular rechnen. Die Initialisierung erfolgt wieder mit

```
Use RingXYZ::=Q[x,y,z];
Init(2);
M:=Mat([MEMORY.KMatrix[I]|I In 1..(Len(MEMORY.KMatrix)-1)]);
```

Für die Berechnungen wird einer der folgenden Befehle benutzt:

```
Time T:=ChinGaussSolve(MEMORY.KMatrix);
Time T:=ChinSyzSolve(MEMORY.KMatrix);
Time T:=ChinBarSolve(M);
Time T:=ChinMalDichSolve(M);
```

Tabelle 3: Modulare Berechnung von `MEMORY.KMatrix` in $\mathbb{Q}[x, y, z]$

S	1-2	3	4	5	6	7	8
`ChinGaussSolve`	< 1	7	271	4410			
`ChinSyzSolve`	< 1	1	6	33	297	2691	
`ChinBarSolve`	< 1	3	50	449	5717		
`ChinMalDichSolve`	< 1	2	34	291	3562		

Offenbar war die Hoffnung berechtigt, alle modularen Algorithmen laufen schneller als ihre nichtmodularen Versionen. Um die Laufzeiten besser einschätzen zu können, interessiert es, in wie vielen endlichen Körpern die Berechnung ausgeführt werden muss, bis man beim Rekombinieren eine Lösung über $\mathbb{Z}$ erhält. Diese Anzahl ist für alle Berechnungsarten und alle Algorithmen gleich, da sie nur vom größten auftretenden Koeffizienten im Lösungsvektor abhängt.

Tabelle 4: Anzahl der benötigten endlichen Körper

S	1-3	4-5	6	7	8	9	10
Anzahl der Körper	1	2	3	5	6	7	9

Man erkennt, dass die Berechnungen mit der modularen Methode etwas länger dauern, als die Berechnung über $\mathbb{Z}/(32003)$ multipliziert mit der Zahl der nötigen Berechnungen. Der Unterschied ist auf das Rekombinieren zurückzuführen, welches nach jeder Berechnung durchgeführt wird.

6.2 Berechnung in 2 Unbestimmten mit `MEMORY.KMatrix`

Als nächstes soll wie in 4.3 beschrieben in $\mathbb{Q}[v, w]$ gerechnet werden. Da zu erwarten ist, dass die Berechnung über endlichen Körpern im gleichen Maß schneller als über $\mathbb{Q}$ ist wie im Polynomring in 3 Unbestimmten, überspringen wir diese und untersuchen die Algorithmen nur über $\mathbb{Q}[v, w]$.
Die Initialisierung unterscheidet sich von der aus dem vorangegangen Abschnitt nur dadurch, dass

```
Use RingVW::=Q[v,w];
```

anstelle von `RingXYZ` verwendet wird. Zur Berechnung werden die gleichen Befehle wie im vorangegangen Unterabschnitt benutzt.

Tabelle 5: Berechnung von `MEMORY.KMatrix` in $\mathbb{Q}[v, w]$

S	1-2	3	4	5	6	7	8
`GaussSolve`	< 1	23	811	12281			
`SyzSolve`	< 1	< 1	5	123	2278		
`BarSolve`	< 1	5	73	1009	8884		
`MalDichSolve`	< 1	3	43	611	6063		

Tabelle 6: Modulare Berechnung von `MEMORY.KMatrix` in $\mathbb{Q}[v, w]$

S	1-2	3	4	5	6	7	8
`ChinGaussSolve`	< 1	10	361	6431			
`ChinSyzSolve`	< 1	< 1	4	25	211	1891	12319
`ChinBarSolve`	< 1	< 1	50	445	5654		
`ChinMalDichSolve`	< 1	2	31	267	3428		

Man beobachtet, dass gerade `SyzSolve` und `ChinSyzSolve` von der Einsparung der Unbestimmten profitieren. Bei den Bareiss-Verfahren zeigt sich kein nennenswerter Unterschied zur Berechnung in $\mathbb{Q}[x, y, z]$. Interessanterweise wird `ChinGaussSolve` sogar langsamer. Vermutlich liegt dies daran, dass der dominante Teil des Algorithmus, die interne CoCoA-Funktion `GCD` über endlichen Körpern die Homogenität der auftretenden Polynome ausnutzt.

Es erstaunt jedoch, dass im Gegensatz dazu `GaussSolve` über $\mathbb{Q}$ deutlich schneller wird. Hier könnte `GCD` wohl noch besser optimiert werden.
Das Ergebnis kann mit den folgenden Befehlen wieder homogenisiert werden, um es mit den Resultaten aus dem vorangegangen Abschnitt vergleichen zu können. Die hierfür benötigte Zeit muss streng genommen zu den Messwerten addiert werden.

```
Use RingXYZ::=Q[x,y,z];
Time TT:=VW2XYZ(T);
```

Tabelle 7: Übersetzung der Lösung von $\mathbb{Q}[v,w]$ nach $\mathbb{Q}[x,y,z]$

S	1-2	3	4	5	6	7	8
`VW2XYZ`	< 1	1	1	6	23	108	417

6.3 Berechnungen mit `MEMORY.LMatrix`

Weiter ist es interessant, ob die Umformung zu `MEMORY.LMatrix` den Algorithmen einen Vorteil verschafft. Zu diesem Zweck sollen die Berechnungen noch einmal mit dieser Matrix (ggf. ohne die letzte Nullzeile für die Bareiss-Verfahren) durchgeführt werden. Die Initialisierung erfolgt entsprechend den vorangegangenen Unterabschnitten, jedoch wird

```
M:=Mat([MEMORY.LMatrix[I]|I In 1..(Len(MEMORY.LMatrix)-1)]);
```

gesetzt und bei der Berechnung `MEMORY.LMatrix` statt `MEMORY.KMatrix` verwendet.

Tabelle 8: Berechnung von `MEMORY.LMatrix` in $\mathbb{Q}[x,y,z]$

S	1-2	3	4	5	6	7	8
`GaussSolve`	< 1	12	231				
`SyzSolve`	< 1	< 1	10	224	4816		
`BarSolve`	< 1	2	7	27	147	742	3351
`MalDichSolve`	< 1	1	6	25	141	717	3234

Tabelle 9: Berechnung von `MEMORY.LMatrix` in $\mathbb{Z}/(32003)[x,y,z]$

S	1-2	3	4	5	6	7	8
`GaussSolve`	< 1	4	35	327	2689		
`SyzSolve`	< 1	< 1	1	16	132	724	3155
`BarSolve`	< 1	1	4	15	50	195	771
`MalDichSolve`	< 1	< 1	3	11	39	173	727

Tabelle 10: Modulare Berechnung von `MEMORY.LMatrix` in $\mathbb{Q}[x, y, z]$

S	1-2	3	4	5	6	7	8
`ChinGaussSolve`	< 1	4	82	713	9358		
`ChinSyzSolve`	< 1	< 1	6	48	506	4597	
`ChinBarSolve`	< 1	1	13	46	281	2119	11303
`ChinMalDichSolve`	< 1	1	10	37	228	1874	10972

`SyzSolve` und `ChinSyzSolve` sind mit `MEMORY.LMatrix` langsamer als mit `MEMORY.KMatrix`. Bei alle übrigen Algorithmen ist jedoch eine deutliche Beschleunigung zu verzeichnen. Dies führt sogar so weit, dass die auf Bareiss-Verfahren gestützten Algorithmen `BarSolve` und `MalDichSolve` schneller sind als ihre modularen Varianten. Offensichtlich reicht nun die Zeitersparnis bei der Berechnung über endlichen Körpern nicht mehr aus, um die mehrfache Ausführung auszugleichen.
Rechnet man mit nur 2 Unbestimmten, bestätigt sich das Bild. Allerdings fällt die Verlangsamung von `SyzSolve` und `ChinSyzSolve` nicht mehr so stark ins Gewicht.

Tabelle 11: Berechnung von `MEMORY.LMatrix` in $\mathbb{Q}[v, w]$

S	1-2	3	4	5	6	7	8
`GaussSolve`	< 1	7	100	1076			
`SyzSolve`	< 1	< 1	5	133	2574		
`BarSolve`	< 1	1	4	19	125	708	3096
`MalDichSolve`	< 1	1	1	18	87	497	2286

Tabelle 12: Modulare Berechnung von `MEMORY.LMatrix` in $\mathbb{Q}[v, w]$

S	1-2	3	4	5	6	7	8
`ChinGaussSolve`	< 1	6	91	861	11549		
`ChinSyzSolve`	< 1	< 1	4	24	218	2135	
`ChinBarSolve`	< 1	1	8	33	223	1857	10191
`ChinMalDichSolve`	< 1	1	6	30	167	1528	8612

6.4 Komprimierte Berechnungen mit MEMORY.SMatrix

Wendet man die Kompression gemäß Satz 4.12 an, so erhält man die Matrix MEMORY.SMatrix. Als Nächstes sollen also alle bisherigen Untersuchungen auch mit dieser Matrix anstelle von MEMORY.KMatrix durchgeführt werden. Da sie viel weniger Zeilen und Spalten besitzt, ist eine weitere Beschleunigung zu erwarten. Die Vorgabewerte aller Algorithmen sind für diesen Fall gewählt. Die Initialisierung beschränkt sich also auf die Definition des Rings mit einem der Befehle

```
Use RingXYZ::=Q[x,y,z];
Use RingXYZ::=Z/(32003)[x,y,z];
```

Die Matrizen werden anschließend mit

```
Init(2)
```

erzeugt. Für die Berechnung wird dann einer der folgenden Befehle aufgerufen:

```
GaussSolve();
SyzSolve();
BarSolve();
MalDichSolve();
```

Tabelle 13: Berechnung von MEMORY.SMatrix in $\mathbb{Q}[x, y, z]$

S	1-3	4	5	6	7	8	9
GaussSolve	< 1	33	> 13h				
SyzSolve	< 1	2	35	579			
BarSolve	< 1	2	10	88	443	2097	7804
MalDichSolve	< 1	1	12	97	466	2028	7763

Tabelle 14: Berechnung von MEMORY.SMatrix in $\mathbb{Z}/(32003)[x, y, z]$

S	1-3	4	5	6	7	8	9
GaussSolve	< 1	2	38	308	2473		
SyzSolve	< 1	< 1	1	5	28	118	441
BarSolve	< 1	< 1	3	15	88	431	1663
MalDichSolve	< 1	< 1	3	15	91	423	1534

Wie erwartet kann man bei allen Algorithmen eine deutliche Beschleunigung gegenüber der Berechnung mit MEMORY.KMatrix bzw. MEMORY.LMatrix beobachten. Man merkt jedoch auch, dass GaussSolve für die Fälle $s > 4$ über $\mathbb{Q}$ nicht mehr zu gebrauchen ist.

Über $\mathbb{Q}$ können auch noch die modularen Befehle

```
ChinGaussSolve();
ChinSyzSolve();
ChinBarSolve();
ChinMalDichSolve();
```

verwendet werden.

Tabelle 15: Modulare Berechnung von `MEMORY.SMatrix` in $\mathbb{Q}[x, y, z]$

S	1-3	4	5	6	7	8	9
`ChinGaussSolve`	< 1	7	115	1030			
`ChinSyzSolve`	< 1	1	6	40	334	1511	5874
`ChinBarSolve`	< 1	3	14	95	771	3827	
`ChinMalDichSolve`	< 1	1	9	75	738	3622	

Auch hier sind alle Berechnungen schneller als mit den größeren Matrizen `MEMORY.KMatrix` und `MEMORY.LMatrix`. Der bereits bei `MEMORY.LMatrix` aufgetretene Effekt, dass die nichtmodularen Bareiss-Algorithmen schneller sind als ihre modularen Varianten, tritt auch in diesem Fall auf. Die Berechnungen in 2 Unbestimmten zeigen das gleiche Bild, wie die folgenden Tabellen zeigen:

Tabelle 16: Berechnung von `MEMORY.SMatrix` in $\mathbb{Q}[v, w]$

S	1-3	4	5	6	7	8	9
`GaussSolve`	< 1	15	181	1403			
`SyzSolve`	< 1	1	28	408	5422		
`BarSolve`	< 1	1	10	77	451	2051	7783
`MalDichSolve`	< 1	1	10	81	463	2018	7721

Tabelle 17: Berechnung von `MEMORY.SMatrix` in $\mathbb{Z}/(32003)[v, w]$

S	1-3	4	5	6	7	8	9
`GaussSolve`	< 1	2	45	358	2778		
`SyzSolve`	< 1	< 1	1	3	15	63	247
`BarSolve`	< 1	1	3	15	86	418	1536
`MalDichSolve`	< 1	1	3	14	90	423	1496

Tabelle 18: Modulare Berechnung von MEMORY.SMatrix in $\mathbb{Q}[v,w]$							
S	1-3	4	5	6	7	8	9
ChinGaussSolve	< 1	7	113	1170	16658		
ChinSyzSolve	< 1	1	6	33	253	1173	4553
ChinBarSolve	< 1	2	10	77	704	3822	
ChinMalDichSolve	< 1	1	9	75	700	3556	

Die Modulare Berechnung von MEMORY.SMatrix in $\mathbb{Q}[v,w]$ verwendet alle in dieser Arbeit behandelten Optimierungen. Tatsächlich findet sich hier auch die schnellste Berechnungsmethode, ChinSyzSolve. Für sie soll nun noch die letzte Optimierung durchgeführt werden, die in 4.4 angesprochen wurde, der modifizierte erste Schritt.

Tabelle 19: Modulare Berechnung von MEMORY.SMatrix in $\mathbb{Q}[v,w]$							
S	1-4	5	6	7	8	9	10
ChinSyzSolve	< 1	6	33	253	1173	4553	18972
ChinSyzSolve(3)	–	–	24	231	1098	4461	18876
ChinSyzSolve(5)	–	–	–	190	1040	4253	18668
ChinSyzSolve(6)	–	–	–	–	1032	3766	18572
ChinSyzSolve(7)	–	–	–	–	–	3710	18489
ChinSyzSolve(9)	–	–	–	–	–	–	18336

Man erkennt eine leichte Verbesserung, je mehr modulare Lösungen auf einmal rekombiniert werden. Diese Optimierung verliert allerdings bei größeren s immer mehr an Bedeutung, da hier die einzelnen Berechnungen schon so aufwendig sind.

Die Lösung über $\mathbb{Q}[v,w]$ muss noch nach $\mathbb{Q}[x,y,z]$ übertragen werden. Es bietet sich an, dies vor der Expansion durchzuführen. Die Zeit, die hierfür benötigt wird, muss streng genommen zu den Berechnungen in $\mathbb{Q}[v,w]$ hinzugerechnet werden. Bei allen Berechnungen mit MEMORY.SMatrix muss noch die Dauer der Expansion addiert werden. Die folgenden Messungen geben diese Zeiten an.

Tabelle 20: Homogenisierung und Expansion in $\mathbb{Q}[x, y, z]$

S	1-4	5	6	7	8	9	10
`VW2XYZ`	< 1	1	5	18	61	183	410
`ExpandSolution`	< 1	1	5	25	80	228	596

Schließlich soll noch ein Überblick über die Dauer des Speicherns und Einlesens der Lösung für `MEMORY.KMatrix` gegeben werden. Man beachte, dass in den Fällen $S > 7$ kurzzeitig eine Menge Arbeitsspeicher benötigt wird.

Tabelle 21: Laden und Speichern der Lösung in $\mathbb{Q}[x, y, z]$

S	1-4	5	6	7	8	9	10
`Save`	< 1	1	2	7	14	31	57
`Load`	< 1	3	14	70	293	1174	3771

6.5 Zusammenfassung

Eine Verwendung von `GaussSolve` mit `MEMORY.LMatrix` war schon im Fall $s = 5$ problematisch. Auch steigt die Berechnungsdauer derartig stark an, dass spätestens im Fall $s = 6$ Schluss sein dürfte. Die Bareiss-Verfahren halten sich bei allen Berechnungen in einem erträglichen Zeitrahmen. Allerdings wurden sie als einzige Algorithmen im Test langsamer, wenn man sie modular ausführte. Der Übergang vom Polynomring in 3 Unbestimmten zum Polynomring in 2 Unbestimmten zeigte bei keinem der getesteten Algorithmen Nachteile. Die Bareiss-Verfahren konnten aus dieser Optimierung jedoch nur wenig Nutzen ziehen. Für alle Algorithmen war die Kompression des Problems eine deutliche Verbesserung. Der in den vorliegenden Tests erfolgreichste Algorithmus war `ChinSyzSolve`. Waren für `MEMORY.LMatrix` die Bareiss-Verfahren noch schneller, so konnte er nach der Komprimierung alle übrigen Algorithmen schlagen. Besonders nach der Einsparung einer Unbestimmten zeigte sich seine Überlegenheit. Es war schon in etwa 5 Stunden möglich das Gleichungssystem für den Fall $s = 10$ zu lösen. Die Berechnung des Falles $s = 12$ auf einem handelsüblichen Computer sollte in etwa einer Woche durchführbar sein. Schließlich sei darauf hingewiesen, dass dieser Algorithmus sich ausgezeichnet für eine Parallelisierung eignet. So könnten die Lösungen in den einzelnen endlichen Körpern durchaus gleichzeitig auf verschiedenen Systemen berechnet werden. Die Ergebnisse würden dann anschließend auf einem zentralen Rechner rekombiniert. Die Durchführung dieses technischen Tricks ist jedoch nicht mehr Teil dieser Arbeit.

Anhang: Inhalt der beiliegenden CD

Die CD hat folgende Verzeichnisstruktur:

```
+- cocoa41/
|  +- linux/
|  +- windows/
|
+- code/
   +- bareiss/
   |  +- manual/
   |
   +- loesung/
```

Im Unterverzeichnis `cocoa41` findet sich CoCoA 4.1 für Microsoft Windows und für Linux. Neuere Versionen von CoCoA und Versionen für andere Betriebssysteme können unter

`http://cocoa.dima.unige.it/`

kostenlos heruntergeladen werden.
Im Unterverzeichnis `code` finden sich alle CoCoA-Programme, die in dieser Arbeit verwendet wurden. Die Datei `chinese.pkg` beinhaltet das im Unterabschnitt 5.4 beschriebene Paket zur modularen Berechnung. In der Datei `solve.coc` befinden sich alle übrigen im Abschnitt 5 vorgestellten Algorithmen. Im Unterverzeichnis `bareiss` liegen die zu [K] gehörigen Dateien für die Bareiss-Verfahren und eine Beschreibung dieser Algorithmen als DVI- und als Textdatei.
Im Unterverzeichnis `loesung` befinden sich für die Fälle $s = 1, \dots, 10$ die Lösungen des Gleichungssystems (2). Die Dateien bestehen jeweils aus einer Liste von Listen aus Monomen. Jede Liste aus Monomen entspricht einer Komponente des Lösungsvektors. Zur besseren Lesbarkeit wurde nach jedem Monom ein Zeilenumbruch eingefügt. Um aus einer dieser Dateien wieder einen Lösungsvektor zu erhalten, muss sie mit der Funktion `Load` in CoCoA eingelesen werden.
Schließlich ist im Wurzelverzeichnis der CD noch diese Arbeit im DVI-, Postscript- und PDF-Format beigefügt.

Literatur

[C] A. Capani, G. Niesi, L. Robbiano, CoCoA, a system for doing Computations in Commutative Algebra, Available via anonymous ftp from: cocoa.dima.unige.it

[KR] M. Kreuzer, L. Robbiano, Computational Commutative Algebra 1, Springer, Berlin, 2000

[G] H.Gold, A Markovian single server with upstream job and downstream demand arrival system, Queueing Systems 30 (1998), 435-455

[K] B. Krammer, Effektive lineare Algebra für Polynommatrizen, Diplomarbeit, Universität Regensburg 2000

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit mit dem Thema

Algebraische Probleme in der Theorie der Warteschlangenmodelle mit Kontrolle

selbst verfasst und neben den angegebenen Quellen und Hilfsmitteln keine weiteren verwendet habe.

Regensburg, im Oktober 2002