Über Moduln und Codes

Ulrich Eidt

Bayreuth, den 14. August 1992

Vorgelegt als Diplomarbeit bei

Prof. Dr. A. Kerber
Lehrstuhl II für Mathematik
der Universität Bayreuth

Inhaltsverzeichnis

Vorwort

James [2] führte den $K\Gamma_r$-Modul $M^{\lambda,\omega}$ ein ($\omega = (1^r), \lambda \vdash r$) und bestimmte die Basen der darin enthaltenen Vektorräume $D^{\lambda^\#,\lambda,\omega}$ ($(\lambda^\#, \lambda)$ ein Paar von Partitionen für r).

Im ersten Kapitel verallgemeinern wir zunächst James' Definition von λ-Tabloiden auf λ-Tabloide vom Inhalt α, wobei α eine uneigentliche Partition von r in höchstens r Teile ist. Dann folgt eine Erweiterung seines Basissatzes auf die Vektorräume $D^{\lambda^\#,\lambda,\alpha} \subseteq M^{\lambda,\alpha}$.

Diese Räume wurden auf ihre Eigenschaften als Codes näher untersucht. Dabei sind Codes mit guten Eigenschaften gefunden worden, die im zweiten Kapitel, nach einer Codierungstheoretischen Einführung, näher beschrieben werden.
Zum Überblick von James-Gewichtsräumen als Codes sind im Anhang dieser Arbeit einige Tabellen.

Im dritten Kapitel werden trace-kompatible Körpererweiterungen endlicher Körper behandelt, die Scheerhorn [6] eingefürt hat. Es wird die Existenz trace-kompatibler Polynome gezeigt und die Implementierung endlicher Körper für das Algebrasystem SYMMETRICA vorgestellt.

An dieser Stelle möchte ich Herrn Dr. Karl-Heinz Zimmermann vom Lehrstuhl für Informatik danken, der mir mit Rat und Tat zur Seite stand, und mit dem es viel Spaß gemacht hat, im Dschungel der James-Gewichtsräume nach guten Codes zu suchen. Ich danke Herrn Prof. Dr. Adalbert Kerber für die Unterstützung und Förderung der Arbeit.
Außerdem danke ich Herrn Alfred Scheerhorn vom IBM Scientific Center in Heidelberg und Herrn Dr. Axel Kohnert, ohne die die Implementierung endlicher Körper für SYMMETRICA nicht möglich gewesen wäre.

Bayreuth, den 14. August 1992 Ulrich Eidt

Kapitel 1

Darstellungstheoretische Grundlagen

1.1 λ-Tabloide vom Inhalt α

Wir zeigen in diesem Unterkapitel die 1:1 Beziehung zwischen den λ-Tabloiden vom Inhalt α und den Γ_α-Orbits von λ-Sequenzen. Diese Beziehung nutzen wir dann für einen Algorithmus zur Konstruktion einer Transversale der λ-Tabloide vom Inhalt α.
Sei also r eine positive ganze Zahl, λ eine Partition von r und α eine uneigentliche Partition von r in höchstens r Teile für den Rest des Kapitels.

1.1.1 Definitionen:

a) Für $n \in \mathbb{N}$ bezeichnet $\underline{n}^{\underline{r}}$ die Menge aller Funktionen von $\underline{r} = \{1, \ldots, r\}$ nach $\underline{n} = \{1, \ldots, n\}$. Eine Funktion wird durch eine Folge $i = (i_1, \ldots, i_r)$ von Bilder beschrieben. Die Γ_r operiert von rechts auf $\underline{n}^{\underline{r}}$ vermöge Platzpermutation $i\pi = (i_{\pi(1)}, \ldots, i_{\pi(r)})$ für alle $i \in \underline{n}^{\underline{r}}$.

b) Sei $\mu = (\mu_1, \ldots, \mu_n) \models n$ in höchstens n Teile. $i = (i_1, \ldots, i_r)$ heißt μ-**Sequenz** genau dann, wenn jedes $j \in \underline{n}$ genau μ_j-mal in i als Bild vorkommt. Die Menge aller μ-Sequenzen wird mit $\mathrm{Seq}(\mu)$ bezeichnet.

•

1.1.2 Vereinbarung: Die monoton steigende μ-Sequenz wird bezeichnet mit

$$m_\mu = (\overbrace{1, \ldots, 1}^{\mu_1}, \overbrace{2, \ldots, 2}^{\mu_2}, \ldots, \overbrace{n, \ldots, n}^{\mu_n}).$$

Sei z.B. $\mu = (2, 2, 0, 1) \models 5$ dann ist $i = (1, 4, 2, 2, 1) \in \mathrm{Seq}(\mu)$ und $m_\mu = (1, 1, 2, 2, 4)$.

1.1.3 Definition: Das **Diagramm** $[\lambda]$ von λ wird durch eine Menge von Paaren

$$[\lambda] := \{(a, b) \in \mathbb{N} \times \mathbb{N} \mid 1 \leq a \wedge 1 \leq b \leq \lambda_i\}$$

definiert, wobei a die Zeile und b die Spalte angibt. •

1.1.4 Vereinbarung: Diagramme werden veranschaulicht, indem man für $(a,b) \in [\lambda]$ in einer gedachten Matrix an der Stelle (a,b) ein Kreuz macht.

Z.B. $\lambda = (3,2,2,1)$ dann ist $[\lambda] =$

$$\begin{array}{ccc} \times & \times & \times \\ \times & \times & \\ \times & \times & \\ \times & & \end{array}$$

1.1.5 Definition: Ein λ-**Tableau** ist eine Abbildung vom Diagramm $[\lambda]$ in eine nichtleere Menge. •

1.1.6 Vereinbarung: Ein solches Tableau stellen wir dar, indem wir im Diagramm $[\lambda]$ statt der Kreuze das der Abbildung entsprechende Mengenelement eintragen.
Wir fixieren das bijektive λ-Tableau $T : [\lambda] \to \underline{r}$,

$$T := \begin{array}{cccc} 1 & \dots & \dots & \lambda_1 \\ \lambda_1 + 1 & \dots & \lambda_1 + \lambda_2 & \\ \dots & \dots & & \end{array}$$

Für jedes $i \in \underline{n^r}$ ist auch iT wegen

$$[\lambda] \xrightarrow{T} \underline{r} \xrightarrow{i} \underline{n}$$

ein λ-Tableau und wird mit T_i bezeichnet.

1.1.7 Bemerkung: Die Abbildung von $\Gamma_r \times [\lambda] \to [\lambda]$

$$(\pi, (a,b)) \mapsto T^{-1}\pi T(a,b) =: \pi(a,b).$$

ist eine Operation.

Beweis:

a) $1(a,b) = T^{-1}1T(a,b) = (a,b)$

b) Seien $\pi, \rho \in \Gamma_r$. Dann ist

$$\pi(\rho(a,b)) = \pi(T^{-1}\rho T(a,b)) = T^{-1}\pi T T^{-1}\rho T(a,b) =$$
$$= T^{-1}\pi\rho T(a,b) = \pi\rho(a,b)$$

□

1.1.8 Definition:
Da die Γ_r auf dem Urbildraum von λ-Tableaux operiert, operiert sie auch auf der Menge der λ-Tableaux. Für jedes λ-Tableau t und jede Permutation π definieren wir πt wie folgt:

$$\pi t((a,b)) := t(\pi^{-1}(a,b)).$$

•

1.1.9 Beispiel: Sei $\lambda = (3,3,1)$, d.h.

$$T = \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & & \end{array}, \text{ außerdem sei } t = \begin{array}{ccc} 1 & 3 & 5 \\ 7 & 6 & 2 \\ 4 & & \end{array}$$

und $\sigma = (157) \in \Gamma_7$. Dann ist

$$\sigma t((1,1)) = t(\sigma^{-1}(1,1)) = t(T^{-1}\sigma^{-1}T(1,1)) = t(T^{-1}\sigma^{-1}1) = t(T^{-1}7) = t((3,1)) = 4,$$

$$\text{ebenso } \sigma t((2,2)) = 1 \text{ und } t((3,1)) = 6.$$

Alle anderen Bilder von t werden durch σ nicht beeinflußt. Wir haben also

$$\sigma t = \begin{array}{ccc} 4 & 3 & 2 \\ 7 & 1 & 5 \\ 6 & & \end{array}.$$

1.1.10 Bemerkung: Die Γ_r operiert also auf der Menge der λ-Tableaux durch *Platzpermutation*, wobei die Plätze von $[\lambda]$ wie in T numeriert sind.

1.1.11 Definitionen:

a) Es sei $\mu \models r$ in höchstens r Teile. Mit Γ_M^r für beliebiges $M \subset \underline{r}$ werde die Untergruppe der Γ_r bezeichnet, die alle Elemente aus $\underline{r} \backslash M$ unverändert läßt. Außerdem sei $\mu^i \subset \underline{r}$ für alle $i \in \underline{r}$ die Menge der Elemente aus dem i-ten μ-Block,

$$\mu^i := \{1 + \sum_{j=1}^{i-1} \mu_j, \ldots, \sum_{j=1}^{i} \mu_j\}.$$

Die kanonische **Younguntergruppe** zu μ wird dann mit

$$\Gamma_\mu := \Gamma_{\mu^1}^r \times \Gamma_{\mu^2}^r \times \ldots \times \Gamma_{\mu^r}^r$$

definiert.

b) Der **Zeilenstabilisator** R_T ist die zu λ gehörige kanonische Younguntergruppe der Γ_r.
Der **Spaltenstabilisator** wird definiert mit

$$C_T := \Gamma^r_{\{1,\alpha_1+1,\alpha_1+\alpha_2+1,\ldots\}} \times \Gamma^r_{\{2,\alpha_1+2,\ldots\}} \times \Gamma_{\{3,\alpha_1+3,\ldots\}} \times \ldots.$$

c) Ein λ-**Tabloid** $\{t\}$ ist die Äquivalenzklasse vom λ-Tableau t unter der Äquivalenzrelation

$$t_1 \sim t_2 :\Longleftrightarrow \text{Es existiert ein } \pi \in R_T : \pi t_1 = t_2.$$

•

1.1.12 Vereinbarung: $\{t\}$ wird dargestellt, indem bei t Linien zwischen die Zeilen und zusätzlich eine Linie darüber und eine darunter gezogen werden.

$$\text{Z.B. } \{T\} = \begin{array}{|ccc|}\hline 1 & \dots & \lambda_1 \\ \hline \lambda_1+1 & \dots & \lambda_1+\lambda_2 \\ \hline \dots \\ \cline{1-1}\end{array}\ .$$

1.1.13 Bemerkung: Sei μ eine uneigentliche Partition von r in höchstens r Teile. Für die monotone Sequenz m_μ sind die Elemente aus dem i-ten μ-Block μ^i alle gleich i. Γ_μ ist demnach genau die Untergruppe der Γ_r, die m_μ unverändert läßt.

1.1.14 Definition: $\{t\}$ heißt λ-**Tabloid vom Inhalt** α , genau dann, wenn jedes $j \in \underline{r}$ genau α_j-mal als Bild in t vorkommt.
Die Menge der λ-Tabloide vom Inhalt α wird mit $\mathrm{Tab}(\lambda, \alpha)$ bezeichnet. •

1.1.15 Beispiel: Sei $\lambda = (3,2),\ \alpha = (2,2,1)$

Dann ist $[\lambda] = \begin{array}{ccc} \times & \times & \times \\ \times & \times & \end{array}$. Ein λ-Tableau $t : [\lambda] \to \underline{5}$ ist z.B. $\begin{array}{ccc} 1 & 2 & 4 \\ 3 & 5 & \end{array}$.

$$R_T = \Gamma^5_{\{1,2,3\}} \times \Gamma^5_{\{4,5\}}$$

somit ist das λ-Tabloid $\{t\}$

$$\begin{array}{|ccc|}\hline 1&2&4\\ \hline 3&5&\\ \cline{1-2}\end{array} = \begin{array}{|ccc|}\hline 2&1&4\\ \hline 5&3&\\ \cline{1-2}\end{array} = \begin{array}{|ccc|}\hline 1&4&2\\ \hline 3&5&\\ \cline{1-2}\end{array} = \begin{array}{|ccc|}\hline 4&1&2\\ \hline 5&3&\\ \cline{1-2}\end{array} = \text{ usw.,}$$

d.h. die Reihenfolge in den Zeilen spielt für $\{t\}$ keine Rolle.
Die verschiedenen λ-Tabloide vom Inhalt α sind

$$\begin{array}{|ccc|}\hline 1&1&2\\ \hline 2&3&\\ \cline{1-2}\end{array},\ \begin{array}{|ccc|}\hline 1&1&3\\ \hline 2&2&\\ \cline{1-2}\end{array},\ \begin{array}{|ccc|}\hline 1&2&2\\ \hline 1&3&\\ \cline{1-2}\end{array},\ \begin{array}{|ccc|}\hline 1&2&3\\ \hline 1&2&\\ \cline{1-2}\end{array} \text{ und } \begin{array}{|ccc|}\hline 2&2&3\\ \hline 1&1&\\ \cline{1-2}\end{array}.$$

Die wesentlichen Informationen der λ-Tabloide liegen darin, welche Bilder von $\{t\}$ in welcher Zeile sind.

$$\text{Wir betrachten z.B. } \begin{array}{|ccc|}\hline 1&1&2\\ \hline 2&3&\\ \cline{1-2}\end{array}.$$

Entscheidend ist hier, daß die Bilder 1,1 und 2 in der ersten Zeile und 2 und 3 in der zweiten Zeile stehen. Die Menge von Tupeln $\{(1,1),(1,2),(1,2),(2,2),(2,3)\}$ enthält demnach alle nötigen Informationen von obigem Tabloid, wobei ein Tupel (a,b) angibt, daß das Bild b in der Zeile a liegt. Man kann also jeder Tupelmenge ein Tabloid und jedem Tabloid eine Tupelmenge zuordnen. Wichtig ist hierbei, daß eine Tupelmenge genau dann einem λ-Tabloid entspricht, wenn eine Folge, bestehend aus den ersten Koordinaten der Tupelmenge, eine λ-Sequenz ist. Darüberhinaus entspricht sie einem

λ-Tabloid vom Inhalt α, wenn zusätzlich eine Folge der zweiten Koordinaten eine α-Sequenz ist. Es entspricht zum Beispiel $\{(2,3),(1,5),(2,5),(3,1),(1,4),(1,2)\}$ wegen $(2,1,2,3,1,1) \in \mathrm{Seq}(3,2,1)$ und $(3,5,5,1,4,2) \in \mathrm{Seq}(1,1,1,1,2)$ einem $(3,2,1)$-Tabloid vom Inhalt $(1,1,1,1,2)$, nämlich

5	4	2
3	5	
1		

.

1.1.16 Definitionen:

a) Sei K ein Körper und n eine positive ganze Zahl.
$R_K(n) = K[\{c_{s,t} \mid s,t \in \underline{n}\}]$ bezeichnet die (kommutative) Algebra aller Polynome über K in den n^2 Unbekannten $c_{s,t}$, $s,t \in \underline{n}$. Für $r \in I\!N$ mit $r \leq n$ sei $R_k(n,r)$ der Unterraum von $R_K(n)$, der von den Monomen

$$c_{i,j} := c_{i_1,j_1} \cdot \ldots \cdot c_{i_r,j_r},\ i = (i_1,\ldots,i_r), j = (j_1,\ldots,j_r) \in \underline{r}^{\underline{r}},$$

vom Grad r aufgespannt wird.

b) Sei $i \in \mathrm{Seq}(\lambda)$ und $j \in \mathrm{Seq}(\alpha)$.
Ein Monom der Form $c_{m_\lambda,j}$ bzw. c_{i,m_α} heißt **in erster Normalform** (1.NF) bzw. **in zweiter Normalform** (2.NF).

•

1.1.17 Vereinbarung: Haben wir ein Monom in 2.NF, so wird es veranschaulicht mit

$$|i_1 \ldots i_{\alpha_1}|i_{\alpha_1+1} \ldots i_{\alpha_1+\alpha_2}|\ldots| := c_{i,m_\alpha}.$$

1.1.18 Korollar: Für alle $i,k \in \mathrm{Seq}(\lambda)$, $j,l \in \mathrm{Seq}(\alpha)$ gilt:

a) $c_{i,j} = c_{k,l} \Longleftrightarrow \exists \pi \in \Gamma_r : i\pi = k \wedge j\pi = l$.

b) $c_{i\pi^{-1},j} = c_{i,j\pi}$ für alle i$\pi \in \Gamma_r$.

c) Jedes Monom $c_{i,j}$ kann in 1.NF bzw. in 2.NF gebracht werden.

Beweis:

a) Folgt aus der Definition und der Kommutativität von $R_K(n)$.

b) Folgt unmittelbar aus a).

c) Es existieren $\sigma,\tau \in \Gamma_r$, so daß $i = m_\lambda\sigma$ und $j = m_\alpha\tau$, d.h. i und j entstehen aus den monotonen Sequenzen durch entsprechende (sortierende) Platzpermutationen. Mit $g = j\sigma^{-1}$ kann man $c_{i,j}$ in 1.NF $c_{i,j} = c_{m_\lambda\sigma,j} = c_{m_\lambda,g}$ (siehe b)) bringen.
Analog bringt man mit $h = i\tau^{-1}$ $c_{i,j}$ in 2.NF $c_{i,j} = c_{h,m_\alpha}$.

□

1.1.19 Bemerkung: $c_{i,j} \in R_K(n,r)$ entspricht der Menge $\{(i_1,j_1),\ldots,(i_r,j_r)\}$, d.h. einem λ-Tabloid vom Inhalt α genau dann, wenn $i \in \mathrm{Seq}(\lambda)$ und $j \in \mathrm{Seq}(\alpha)$. Es gibt somit eine Bijektion von der Menge der Monome $c_{i,j}$ $(i \in \lambda,\ j \in \alpha)$ auf die Menge der λ-Tabloide vom Inhalt α.
Diese Bijektion ist als solche besonders gut zu erkennen, wenn wir die Monome $c_{m_\lambda,j}$, $j \in \mathrm{Seq}(\alpha)$, in 1.NF vorliegen haben. Es entsprechen dann nämlich in der Folge j die Bilder $j_1,\ldots,j_{\lambda_1}$ genau den Bildern der 1.Reihe des Tabloids, die Bilder $j_{\lambda_1+1},\ldots,j_{\lambda_1+\lambda_2}$ genau den Bildern der 2.Reihe usw.

1.1.20 Beispiel: $\lambda = (3,2)$, $\alpha = (1,1,1,2)$

$$t := \overline{\begin{array}{\underline{ccc}} 4 & 2 & 3 \\ 1 & 4 \end{array}} \longleftrightarrow \{(1,4),(1,2),(1,3),(2,1),(2,4)\} \longleftrightarrow$$

$$c_{1,4}\cdot c_{1,2}\cdot c_{1,3}\cdot c_{2,1}\cdot c_{2,4} = c_{(1,1,1,2,2),(4,2,3,1,4)} =: \overline{t}$$

t legt $\overline{t}$ fest und umgekehrt. Ändert man bei $\overline{t}$ die Reihenfolge von $(4,2,3,1,4)$ innerhalb der λ-Blöcke, so muß $\overline{t}$ nach obiger Bemerkung immer noch t entsprechen.

$$\overline{t} = c_{(1,1,1,2,2),(3,4,2,4,1)} = c_{1,3}\cdot c_{1,4}\cdot c_{1,2}\cdot c_{2,4}\cdot c_{2,1} \longleftrightarrow$$

$$\{(1,3),(1,4),(1,2),(2,4),(2,1)\} \longleftrightarrow \overline{\begin{array}{\underline{ccc}} 3 & 4 & 2 \\ 4 & 1 \end{array}} = t$$

In 2.NF spielt die Reihenfolge innerhalb der α-Blöcke keine Rolle. Obiges Monom in 2.NF:

$$|2|1|1|12| = c_{(2,1,1,1,2),(1,2,3,4,4)} = c_{(2,1,1,2,1),(1,2,3,4,4)} = |2|1|1|21|$$

Um von einer Normalform in die andere zu gelangen, sortiert man einfach das Monom entweder nach der ersten oder nach der zweiten Sequenz, was wegen der Kommutativität von $R_K(n)$ möglich ist.

$$\left(\overline{\begin{array}{\underline{ccc}} 4 & 2 & 3 \\ 1 & 4 \end{array}} \leftrightarrow\right) c_{(1,1,1,2,2),(4,2,3,1,4)} = c_{(2,1,1,1,2),(1,2,3,4,4)} = |2|1|1|12| \ \ bzw.$$

$$|2|1|1|21| = c_{(2,1,1,2,1),(1,2,3,4,4)} = c_{(1,1,1,2,2),(2,3,4,1,4)} \left(\leftrightarrow \overline{\begin{array}{\underline{ccc}} 3 & 4 & 2 \\ 4 & 1 \end{array}}\right)$$

Wir wenden uns jetzt der Aufgabe zu, eine Transversale der λ-Tabloide vom Inhalt α zu konstruieren. Dafür ermitteln wir alle verschiedenen Monome $c_{i,j}$ mit $i \in \mathrm{Seq}(\lambda)$ und $j \in \mathrm{Seq}(\alpha)$.

1.1.21 Definition: Sei $i \in \underline{r}^{\underline{r}}$. Die Menge

$$\Gamma_\alpha(i) := \{i\pi \mid \pi \in \Gamma_\alpha\}$$

heißt Γ_α-**Bahn** von i. •

1.1.22 Korollar: Zwei Monome c_{i,m_α} und c_{k,m_α} in 2.NF sind genau dann gleich, wenn die λ-Sequenzen i,k in derselben Γ_α-Bahn liegen.

Beweis: Mit Korollar 1.1.18 gilt:

$$c_{i,m_\alpha} = c_{k,m_\alpha} \Leftrightarrow \exists \pi \in \Gamma_r : i\pi = k \wedge m_\alpha \pi = m_\alpha,$$

es folgt mit Bemerkung 1.1.13

$$c_{i,m_\alpha} = c_{k,m_\alpha} \Leftrightarrow \exists \pi \in \Gamma_\alpha : i\pi = k,$$

d.h. i und k liegen in derselben Γ_α-Bahn. □

1.1.23 Bemerkung: Zur Konstruktion aller verschiedenen Monome können wir uns nach Korollar 1.1.18 auf Monome in 2.NF beschränken, da sich jedes Monom auf 2.NF bringen läßt. Wir entnehmen also jeder Γ_α-Bahn von λ-Sequenzen einen Repräsentanten und interpretieren diese dann als Monome in 2.NF.

1.2 James Gewichtsräume

Im folgenden Kapitel wird der Basissatz für James Gewichtsräume bewiesen, der eine Verallgemeinerung des Basissatzes für Spechtmoduln ist, den James [2] durchgeführt hat. Der Beweis zu dieser Verallgemeinerung folgt in der Grundidee dem Beweis von Zimmermann [9].
Es sei für das ganze Kapitel n eine natürliche Zahl, λ eine Partition von n, α eine uneigentliche Partition von n in höchstens n Teile, $\omega = (1^n)$ die Partition von n in genau n Teile und K ein fest gewählter Körper.

1.2.1 Definitionen:

a) Es bezeichne
$$M^{\lambda,\alpha} := \bigoplus_{\{t\} \in \mathrm{Tab}(\lambda,\alpha)} K\ \{t\}$$
den K-Vektorraum, der durch die λ-Tabloide vom Inhalt α erzeugt wird.

b) Sei $n' \in \mathbb{N}$, mit $n' \leq n$. Sei $\lambda^{\#} \vdash n'$. Das Paar $(\lambda^{\#}, \lambda)$ heißt **Paar von Partitionen für n** , wenn gilt:
$$\lambda_i^{\#} \leq \lambda_i \text{ für alle } i \geq 1.$$

c) Sei $(\lambda^{\#}, \lambda)$ ein Paar von Partitionen für n. Mit $C_T(\lambda^{\#})$ wird die Untergruppe des Spaltenstabilisators C_T bezeichnet, die die Elemente außerhalb des Diagramms $[\lambda^{\#}]$ unverändert läßt.

d) Sei $t : [\lambda] \to \underline{n}$ ein λ-Tableau. Als $\lambda^{\#}, \lambda$-**Polytabloid zu t** wird
$$e_t^{\lambda^{\#},\lambda} := \sum_{\sigma \in C_T(\lambda^{\#})} \mathrm{sgn}(\sigma)\ \{\sigma t\}$$
definiert.

•

Für den Rest dieses Kapitels sei $(\lambda^{\#}, \lambda)$ immer ein Paar von Partitionen für n.

1.2.2 Beispiel: Sei $\lambda^{\#} = (2,2)$ und $\lambda = (3,2,2)$.

$$\text{Für } t = \begin{array}{ccc} 1 & 3 & 3 \\ 2 & 2 & \\ 1 & 4 & \end{array} \text{ ist } e_t^{\lambda^{\#},\lambda} \in M^{\lambda,(2,2,2,1)}, \text{ nämlich}$$

$$\{t\} - \{(14)t\} - \{(25)t\} + \{(14)(25)t\} = \underline{\begin{array}{ccc} 1 & 3 & 3 \\ 2 & 2 & \\ 1 & 4 & \end{array}} - \underline{\begin{array}{ccc} 2 & 3 & 3 \\ 1 & 2 & \\ 1 & 4 & \end{array}} - \underline{\begin{array}{ccc} 1 & 2 & 3 \\ 2 & 3 & \\ 1 & 4 & \end{array}} + \underline{\begin{array}{ccc} 2 & 2 & 3 \\ 1 & 3 & \\ 1 & 4 & \end{array}}.$$

1.2.3 Korollar: Stehen bei einem λ-Tableau $t : [\lambda] \to \underline{n}$ innerhalb von $[\lambda^\#]$ zwei gleiche Bilder in derselben Spalte, so gilt:

$$e_t^{\lambda^\#,\lambda} = 0.$$

Beweis: Es seien $(a,b),(a',b)$ die Stellen in t, an denen die gleichen Bilder in der selben Spalte b stehen. Es sei $j = T(a,b)$, $k = T(a',b)$. Es existiert eine Untergruppe H von $C_T(\lambda^\#)$, für die gilt:

$$C_T(\lambda^\#) = H \cup H(jk).$$

Wir haben

$$e_t^{\lambda^\#,\lambda} = \sum_{\sigma\in H} \operatorname{sgn}(\sigma)\{\sigma t\} - \sum_{\sigma\in H} \operatorname{sgn}(\sigma)\{\sigma(jk)t\} = 0,$$

da $t = (jk)t$. □

1.2.4 Definition: Wir definieren die Funktion $sub_\alpha : \underline{n} \to \underline{n}$ wie folgt:

$$sub_\alpha(s) = r :\Longleftrightarrow \sum_{k=1}^{r-1} \alpha_k < s \leq \sum_{k=1}^{r} \alpha_k \text{ für alle } s \in \underline{n}.$$

Wir erweitern die Definition von sub_α auf $\underline{n}^{\underline{n}}$ mit

$$sub_\alpha(i) = (sub_\alpha(i_1), \ldots, sub_\alpha(i_n)) \text{ für alle } i = (i_1,\ldots,i_n) \in \underline{n}^{\underline{n}}.$$

Analog wird sub_α auch auf λ-Tableaux definiert.

•

1.2.5 Bemerkung: Ist $i \in \mathrm{Seq}(\omega)$ dann ist $sub_\alpha(i) \in \mathrm{Seq}(\alpha)$.

Beweis: Sei $s \in \{1,\ldots,\alpha_1\}$. Dann gilt $0 < s \leq \alpha_1$, d.h. $sub_\alpha(s) = 1$. Desgleichen wird ein Bild $s \in \{\alpha_1+1,\ldots,\alpha_1+\alpha_2\}$ wegen $\alpha_1 < s \leq \alpha_1+\alpha_2$ auf die Zwei abgebildet, usw. Die Sequenz i enthält jedes Bild aus $\underline{n}$ genau einmal, so daß $sub_\alpha(i)$ α_1-mal die Eins, α_2-mal die Zwei enthält usw., d.h. $sub_\alpha(i) \in \mathrm{Seq}(\alpha)$. □

1.2.6 Beispiel:
Sei $\alpha = (3,2,1)$, dann ist

$$sub_\alpha(1) = sub_\alpha(2) = sub_\alpha(3) = 1,\ sub_\alpha(4) = sub_\alpha(5) = 2 \text{ und } sub_\alpha(6) = 3.$$

Mit $i = (1,6,4,3,2,5) \in \mathrm{Seq}(\omega)$ ist $sub_\alpha(i) = (1,3,2,1,1,2) \in \mathrm{Seq}(\alpha)$. Für

$$t = \begin{array}{ccc} 1 & 3 & 6 \\ 4 & 5 & 2 \end{array}, \text{ erhält man } sub_\alpha(t) = \begin{array}{ccc} 1 & 1 & 3 \\ 2 & 2 & 1 \end{array}.$$

1.2.7 Definitionen:

a) Wir definieren die Abbildung $\psi^{\lambda,\alpha} : M^{\lambda,\omega} \to M^{\lambda,\alpha}$. Sei $\{t\} \in \mathrm{Tab}(\lambda,\omega)$, dann ist

$$\psi^{\lambda,\alpha}(\{t\}) := \{sub_\alpha(t)\}.$$

b)

$$D^{\lambda^\#,\lambda,\omega} := \sum_{i \in \mathrm{Seq}(\omega)} e_{T_i}^{\lambda^\#,\lambda}$$

heißt $(\lambda^\#,\lambda,\omega)$-**James Gewichtsraum** .

c)

$$D^{\lambda^\#,\lambda,\alpha} := \psi^{\lambda,\alpha}(D^{\lambda^\#,\lambda,\omega})$$

heißt $(\lambda^\#,\lambda,\alpha)$-**James Gewichtsraum** .

•

1.2.8 Bemerkungen:

a) Die Abbildung $\psi^{\lambda,\alpha}$ ist ein K-Epimorphismus, denn zu jedem λ-Tabloid vom Inhalt α {t} existiert ein λ-Tableau t' mit $\{t\} = \{sub_\alpha(t')\}$.

b) $D^{\lambda^\#,\lambda,\omega}$ entspricht dem Vektorraum $S^{\lambda^\#,\lambda}$ bei James [2]. Darüberhinaus ist $D^{\lambda,\lambda,\omega}$ gleich dem Spechtmodul S^λ .

c) Es sei $t : [\lambda] \to \underline{n}$ ein λ-Tableau. Dann ist

$$\psi^{\lambda,\alpha}(e_t^{\lambda^\#,\lambda}) = e_{sub_\alpha(t)}^{\lambda^\#,\lambda},\ denn$$

$$\psi^{\lambda,\alpha}(e_t^{\lambda^\#,\lambda}) = \sum_{\sigma \in C_T(\lambda^\#)} \mathrm{sgn}(\sigma)\psi^{\lambda,\alpha}(\{\sigma t\}) = \sum_{\sigma \in C_T(\lambda^\#)} \mathrm{sgn}(\sigma)\{sub_\alpha(\sigma t)\}.$$

Dies entspricht dem $\lambda^\#,\lambda$ - Polytabloid zu $sub_\alpha(t)$, da $sub_\alpha(\sigma t) = \sigma sub_\alpha(t)$ für alle $\sigma \in \Gamma_n$.

1.2.9 Beispiel:

Sei $\lambda = (3,3),\ \alpha = (3,2,1)$ und $\omega = (1^6)$.

$$\text{Für } \{t\} = \overline{\begin{array}{ccc}1&2&3\\4&5&6\end{array}} \in M^{\lambda,\omega} \text{ ist } \psi^{\lambda,\alpha}(\{t\}) = \overline{\begin{array}{ccc}1&1&1\\2&2&3\end{array}} \in M^{\lambda,\alpha}.$$

Mit $\lambda^\# = (2,2)$ ist

$$e_t^{\lambda^\#,\lambda} = \overline{\begin{array}{ccc}1&2&3\\4&5&6\end{array}} - \overline{\begin{array}{ccc}4&2&3\\1&5&6\end{array}} - \overline{\begin{array}{ccc}1&5&3\\4&2&6\end{array}} + \overline{\begin{array}{ccc}4&5&3\\1&2&6\end{array}}.$$

$$\psi^{\lambda,\alpha}(e_t^{\lambda^\#,\lambda}) = \psi^{\lambda,\alpha}\left(\overline{\begin{array}{ccc}1&2&3\\4&5&6\end{array}} - \overline{\begin{array}{ccc}4&2&3\\1&5&6\end{array}} - \overline{\begin{array}{ccc}1&5&3\\4&2&6\end{array}} + \overline{\begin{array}{ccc}4&5&3\\1&2&6\end{array}}\right) =$$

$$\overline{\begin{array}{ccc}1&1&1\\2&2&3\end{array}} - \overline{\begin{array}{ccc}2&1&1\\1&2&3\end{array}} - \overline{\begin{array}{ccc}1&2&1\\2&1&3\end{array}} + \overline{\begin{array}{ccc}2&2&1\\1&1&3\end{array}} = e_{sub_\alpha(t)}^{\lambda^\#,\lambda}.$$

Unsere Aufgabe besteht darin, die Basis von $D^{\lambda^\#,\lambda,\alpha}$ zu bestimmen.
Die folgenden Definitionen in James [2] wurden zur Bestimmung der Basis von $D^{\lambda^\#,\lambda,\omega}$ gemacht. Von einer solchen Basis ausgehend, kann man ein Erzeugendensystem von $D^{\lambda^\#,\lambda,\alpha}$ bestimmen.

1.2.10 Definitionen:

a) Ein $\lambda - Tableau\ t\ :\ [\lambda] \to \underline{n}$ heißt **standard** , wenn die Bilder entlang der Zeilen monoton und entlang der Spalten streng monoton wachsen.

b) Sei $n \in \mathbb{N}$, λ eine Partition von n und $i \in \mathrm{Seq}(\lambda)$, dann wird die Qualität jedes einzelnen Bildes von i wie folgt definiert:

1. Alle 1'en sind **gut** .
2. Für beliebiges $k \geq 1$ ist ein auftretendes $k+1$ genau dann **gut** , wenn die Anzahl der vorherigen guten k's echt größer als die Anzahl der vorherigen $k+1$'en ist. Andernfalls ist das $k+1$ **schlecht** .

c) $\mathrm{Seq}(\lambda^\#, \lambda)$ ist die Menge derjenigen Sequenzen $i \in \mathrm{Seq}(\lambda)$, bei denen für jede positive Zahl k die Anzahl der guten in i auftretenden k's mindestens $\lambda_k^\#$ ist.

d) Durch folgende Konstruktion wird eine Abbildung von $\mathrm{Seq}(\lambda)$ in die Menge der λ-Tableaux definiert:
Sei $i \in \mathrm{Seq}(\lambda)$. $T \cdot i$ wird wie folgt konstruiert:

Für jedes $s = 1, \ldots, n$ (in dieser Reihenfolge) führe folgendes durch:
Ist i_s gut, dann setze s soweit wie möglich in der i_s-ten Reihe nach links. Ist i_s schlecht, dann soweit wie möglich nach rechts.

e) Wir definieren eine Totalordnung $<_{\mathrm{Tab}}$ auf der Menge $\mathrm{Tab}(\lambda, \alpha)$: Es seien $\{t\}$ und $\{t'\}$ zwei verschiedene λ-Tabloide vom Inhalt α, s und s' diejenigen Tableaux aus $\{t\}$ bzw. $\{t'\}$, die entlang der Zeilen monoton wachsend sind.
Es ist $\{t\} <_{Tab} \{t'\}$, wenn s in der ersten Zeile von unten, in der sich s und s' unterscheiden, lexikographisch kleiner ist als s'.

•

1.2.11 Beispiel: Bei der folgenden Sequenz wurden Bilder mit der Qualität 'schlecht' durch x gekennzeichnet:

$(1, 2, \overset{x}{2}, 3, \overset{x}{3}, 1, 2, 3, 1, 1) \in \mathrm{Seq}((4,2,2),(4,3,3))$, da 4 gute 1'er, 2 gute 2' er und 2 gute 3' er enthalten sind.

Sei $i = (2,1,3,1,2) \in \mathrm{Seq}((2,1),(2,2,1))$. Die Konstruktion von $T \cdot i$ sieht folgendermaßen aus:

$$(\overset{\downarrow}{2}, 1, 3, 1, 2) \longmapsto \begin{array}{cc} \times & \times \\ \times & 1 \\ \times & \end{array}$$

$$(2,\overset{\downarrow}{1},3,1,2) \longmapsto \begin{array}{cc} 2 & \times \\ \times & 1 \\ \times & \end{array}$$

$$(2,1,\overset{\downarrow}{3},1,2) \longmapsto \begin{array}{cc} 2 & \times \\ \times & 1 \\ 3 & \end{array}$$

$$(2,1,3,\overset{\downarrow}{1},2) \longmapsto \begin{array}{cc} 2 & 4 \\ \times & 1 \\ 3 & \end{array}$$

$$(2,1,3,1,\overset{\downarrow}{2}) \longmapsto \begin{array}{cc} 2 & 4 \\ 5 & 1 \\ 3 & \end{array}$$

Diese Konstruktion sorgt dafür, daß ein Bild eines guten Elementes r immer unter dem Bild eines guten Elementes steht, das in der Sequenz vor r auftritt. Somit gilt allgemein, daß mit $i \in \mathrm{Seq}(\lambda^{\#},\lambda)$ $T \cdot i$ innerhalb des $\lambda^{\#}$-Diagramms standard ist.
Sei $\lambda = (3,2)$ und $\alpha = (2,2,1)$. Die λ-Tabloide vom Inhalt α haben folgende Ordnung:

$$\underline{\begin{array}{ccc} 2 & 2 & 3 \\ 1 & 1 & \end{array}} <_{Tab} \underline{\begin{array}{ccc} 1 & 2 & 3 \\ 1 & 2 & \end{array}} <_{Tab} \underline{\begin{array}{ccc} 1 & 2 & 2 \\ 1 & 3 & \end{array}} <_{Tab} \underline{\begin{array}{ccc} 1 & 1 & 3 \\ 2 & 2 & \end{array}} <_{Tab} \underline{\begin{array}{ccc} 1 & 1 & 2 \\ 2 & 3 & \end{array}}.$$

1.2.12 Bemerkung: James beweist in seiner Arbeit, daß

$$\{e^{\lambda^{\#},\lambda}_{T\cdot i} \mid i \in \mathrm{Seq}(\lambda^{\#},\lambda)\}$$

eine K-Basis für $D^{\lambda^{\#},\lambda,\omega}$ ist.
Ein Erzeugendensystem für $D^{\lambda^{\#},\lambda,\alpha}$ ist also wegen $\psi^{\lambda,\alpha}(e^{\lambda^{\#},\lambda}_{t}) = e^{\lambda^{\#},\lambda}_{sub_\alpha(t)}$ (siehe Bemerkung 1.2.8) die Menge

$$\{e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot i)} \mid i \in \mathrm{Seq}(\lambda^{\#},\lambda)\}.$$

Dieses Erzeugendensystem gilt es zu minimieren.

1.2.13 Lemma: Sei $i \in \mathrm{Seq}(\lambda^{\#},\lambda)$, $\tau = (k,k+1)$, $(k \in \{1,\ldots,n-1\})$ eine Transposition aus $\Gamma_\alpha(i)$ mit $i_k < i_{k+1}$
Ist $i\tau \in \mathrm{Seq}(\lambda^{\#},\lambda)$, dann gilt

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot i)} = e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot(i\tau))},$$

falls $i\tau \notin \mathrm{Seq}(\lambda^{\#},\lambda)$, haben wir

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot i)} = 0.$$

Beweis: Wir betrachten die Abbildung $i \mapsto sub_\alpha(T \cdot i)$. Für die l-te Stelle von i wird bei $T \cdot i$ in die i_l-te Zeile die Zahl l eingetragen, die dann zu $sub_\alpha(l)$ wird. Das bedeutet, daß die ersten α_1 Stellen von i auf die 1, die darauffolgende α_2 Stellen von i auf die 2 abgebildet werden und so weiter.
Wir unterscheiden zwei Fälle:

i) Durch die Transposition τ ändert sich die Qualität des Bildes der Stelle $k+1$ nicht. Dies ist nur möglich, wenn auch $i\tau \in \mathrm{Seq}(\lambda^{\#}, \lambda)$. Da $\tau \in \Gamma_\alpha$, liegen die Stellen k und $k+1$ im selben α-Block, sie werden folglich unabhängig von τ auf dieselbe Zahl abgebildet, d.h. $sub_\alpha(T \cdot i) = sub_\alpha(T \cdot (i\tau))$.

ii) Die Transposition τ ändert die Qualität des Bildes der Stelle $k+1$.
Damit τ diese Wirkung hat müssen folgende Bedingungen gelten:

$i_k + 1 = i_{k+1}$.

Die Anzahl p der bis zur Stelle $k-1$ in i auftretenden 'guten' i_k's ist gleich der Anzahl der bis zur Stelle $k-1$ auftretenden 'guten' $i_k + 1$'en.

In i sind die Stellen k und $k+1$ 'gut'.

Ist nun $i\tau \in \mathrm{Seq}(\lambda^{\#}, \lambda)$, so stimmen innerhalb von $[\lambda^{\#}]$ $sub_\alpha(T \cdot i)$ und $sub_\alpha(T \cdot (i\tau))$ überein. Außerhalb von $[\lambda^{\#}]$ können sich diese Tableaux zwar unterscheiden, aber die Zeilen als Mengen sind gleich, da die Qualität eines Elementes in i bei der Konstruktion von $T \cdot i$ keinen Einfluß auf die Zeilen, sondern nur auf die Spalten hat.
Die $\lambda^{\#}, \lambda$-Polytabloide von $t = sub_\alpha(T \cdot i)$ und $t' = sub_\alpha(T \cdot (i\tau))$ sind gleich, da bei Tabloiden die Reihenfolge innerhalb der Zeilen keine Rolle spielt und t und t' innerhalb von $[\lambda^{\#}]$ übereinstimmen.
Ist $i\tau \notin \mathrm{Seq}(\lambda^{\#}, \lambda)$, so liegt das Bild der Stelle $k+1$ von i in $T \cdot i$ innerhalb von $[\lambda^{\#}]$. Sind wir also bei der Konstruktion von $T \cdot i$ an der Stelle $k-1$ angelangt, haben wir die Situation, daß in der Zeile i_k und in der Zeile $i_k + 1$ genau die ersten p Stellen belegt sind. Somit stehen k und $k+1$ in $T \cdot i$ in der Spalte p in den Zeilen i_k und $i_k + 1$. k und $k+1$ sind im gleichen α-Block von i, das bedeutet für $sub_\alpha(T \cdot i)$, daß innerhalb von $[\lambda^{\#}]$ zwei gleiche Bilder $sub_\alpha(k)$ stehen. Somit folgt mit Korollar 1.2.3:

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T \cdot i)} = 0.$$

□

1.2.14 Definition: $\mathrm{Seq}_\alpha(\lambda^{\#}, \lambda)$ wird definiert als die Menge von Sequenzen $i \in \mathrm{Seq}(\lambda^{\#}, \lambda)$, für die i entlang der α-Blöcke monoton fallend ist. •

1.2.15 Satz: Die Menge

$$\{e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i)} \mid i \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)\}$$

ist eine K-Basis von $D^{\lambda^\#,\lambda,\alpha}$.

Beweis:

i) Erzeugendensystem:
Es sei $i \in \mathrm{Seq}(\lambda',\lambda)$. Wir wählen eine Transposition $\tau_1 = (k, k+1) \in \Gamma_\alpha$, mit $i_k < i_{k+1}$. Ist $i\tau_1 \notin \mathrm{Seq}(\lambda',\lambda)$ so ist nach Lemma 1.2.13 $e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i)} = 0$.
Andernfalls ist $e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i)} = e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(i\tau_1))}$ und wir können für $j = i\tau_1$ ein $\tau_2 = (k', k'+1) \in \Gamma_\alpha$ wählen mit $j_{k'} < j_{k'+1}$ und das Verfahren wiederholen. Nach endlich vielen Wiederholungen erhalten wir entweder $i\pi \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$, wobei $\pi = \tau_1\tau_2\ldots$, oder $e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i)} = 0$.

ii) Lineare Unabhängigkeit:
Seien $i, j \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$, $i \neq j$. $j \notin \Gamma_\alpha(i)$, da i und j entlang der α-Blöcke absteigend sortiert sind.
Es sei x die erste Stelle, an der sich i und j unterscheiden. x sei im k-ten α-Block. Wir dürfen annehmen, daß $i_x > j_x$.
Sind wir bei der Konstruktion von $sub_\alpha(T\cdot i)$ an der Stelle x angelangt, tragen wir in der Zeile i_x $sub_\alpha(x) = k$ ein. Bei j stehen ab der Stelle x im k-ten α-Block nur Bilder, die kleiner als i_x sind, d.h. daß in $sub_\alpha(T\cdot i)$ in der Zeile i_x mindestens einmal mehr k steht als in $sub_\alpha(T\cdot j)$. Wir haben also:

$$i, j \in \mathrm{Seq}_\alpha(\lambda',\lambda),\ i \neq j \Longrightarrow \{sub_\alpha(T\cdot i)\} \neq \{sub_\alpha(T\cdot j)\}.$$

Es sei $t = sub_\alpha(T\cdot i)$. Im Träger von $e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i)}$ mit $i \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$ ist $\{t\}$ das größte Element in der Ordnung $<_{Tab}$, da t standard in $[\lambda^\#]$.
Es sei $\mid \mathrm{Seq}_\alpha(\lambda^\#,\lambda) \mid = m$. $i_1,\ldots,i_m$ seien die Sequenzen $\mathrm{Seq}_\alpha(\lambda^\#,\lambda)$.
Ohne Beschränkung der Allgemeinheit können wir annehmen, daß

$$\{sub_\alpha(T\cdot i_r)\} <_{Tab} \{sub_\alpha(T\cdot i_s)\},\ \text{falls } r < s.$$

Wir betrachten die Gleichung

$$k_1 e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i_1)} + \ldots + k_m e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot i_m)} = 0.$$

Dann ist $k_m = 0$, da $\{sub_\alpha(T\cdot i_m)\}$ im Träger von $e^{\lambda^\#}_{sub_\alpha(T\cdot i_m)}$ enthalten ist, aber in keinem Polytabloid sonst. Mit der gleichen Argumentation kann man zeigen, daß auch $k_{m-1} = k_{m-2} = \ldots = k_1 = 0$.

□

1.2.16 Korollar: Seien $n, n' \in \mathbb{N}$ mit $n' \leq n$, $\lambda^{\#} \vdash n'$, $(\lambda^{\#}, \lambda)$ ein Paar von Partitionen für n und α eine Partition von n in höchstens n Teile. μ sei eine Partition von $n'-1$, mit

$$\mu = (\lambda_1^{\#}, \ldots, \lambda_{i-1}^{\#}, \lambda_i^{\#} - 1, \lambda_{i+1}^{\#}, \ldots),\ \lambda_i^{\#} \geq 1.$$

Es gilt

a)
$$D^{\lambda^{\#},\lambda,\omega} \subseteq D^{\mu,\lambda,\omega},$$

b)
$$D^{\lambda^{\#},\lambda,\alpha} \subseteq D^{\mu,\lambda,\alpha}.$$

Beweis:

a) Wir zeigen, daß ein beliebiges Polytabloid aus $D^{\lambda^{\#},\lambda,\omega}$ in $D^{\mu,\lambda,\omega}$ enthalten ist. Sei $t = T_i$, $i \in \mathrm{Seq}(\omega)$ und $\{\pi_1, \ldots, \pi_k\}$ eine Transversale der Rechtsnebenklassen von $C_T(\mu)$ in $C_T(\lambda^{\#})$. Wir haben

$$e_t^{\lambda^{\#},\lambda} = \sum_{\sigma \in C_T(\lambda^{\#})} \mathrm{sgn}(\sigma)\{\sigma t\} =$$

$$= \sum_{\sigma \in C_T(\mu)} \mathrm{sgn}(\sigma\pi_1)\{\sigma\pi_1 t\} + \ldots + \sum_{\sigma \in C_T(\mu)} \mathrm{sgn}(\sigma\pi_k)\{\sigma\pi_k t\} =$$

$$= \mathrm{sgn}(\pi_1) e_{\pi_1 t}^{\mu,\lambda} + \ldots + \mathrm{sgn}(\pi_k) e_{\pi_k t}^{\mu,\lambda} \in D^{\mu,\lambda,\omega}.$$

b) Folgt aus a) wegen

$$D^{\lambda^{\#},\lambda,\alpha} = \psi^{\lambda,\alpha}(D^{\lambda^{\#},\lambda,\omega}) \subseteq \psi^{\lambda,\alpha}(D^{\mu,\lambda,\omega}) = D^{\mu,\lambda,\alpha}.$$

□

1.3 Allgorithmen zu James Gewichtsräumen

Zur Erzeugung einer Basis von $M^{\lambda,\alpha}$ geht man wie folgt vor: Man durchläuft die λ-Sequenzen in lexikographischer Reihenfolge und speichert nur diejenigen, die entlang der α-Blöcke monoton fallend sind. So erhält man aus jedem Γ_α-Orbit genau einen Repräsentanten. Nach Bemerkung 1.1.23 erhält man auf diese Weise alle verschiedenen Monome $c_{i,j}$ mit $i \in \mathrm{Seq}(\lambda)$, $j \in \mathrm{Seq}(\alpha)$, für die man leicht die zugehörigen λ-Tabloide vom Inhalt α ermitteln kann, die dann eine Basis von $M^{\lambda,\alpha}$ bilden (siehe Kapitel 1.1).

1.3.1 Beispiel: $\lambda = (3,2)$, $\alpha = (2,2,1)$

$$(|11|12|2| = m_1)$$

$$\begin{aligned}
m_1 &= |11|21|2| = c_{(1,1,2,1,2),(1,1,2,2,3)} \\
m_2 &= |11|22|1| = c_{(1,1,2,2,1),(1,1,2,2,3)} \\
&(|12|11|2| = m_3) \\
&(|12|12|1| = m_4) \\
&(|12|21|2| = m_4) \\
m_3 &= |21|11|2| = c_{(2,1,1,1,2),(1,1,2,2,3)} \\
&(|21|12|1| = m_4) \\
m_4 &= |21|21|1| = c_{(2,1,2,1,1),(1,1,2,2,3)} \\
m_5 &= |22|11|1| = c_{(2,2,1,1,1),(1,1,2,2,3)}
\end{aligned}$$

Dies sind alle verschiedenen Monome c_{i,m_α} mit $i \in \mathrm{Seq}(\lambda)$. Die Basis von $M^{\lambda,\alpha}$ ist also

$$c_{(1,1,2,1,2),(1,1,2,2,3)} = c_{(1,1,1,2,2),(1,1,2,2,3)} \mapsto \begin{array}{|ccc|}\hline 1 & 1 & 2 \\ \hline 2 & 3 \\ \cline{1-2}\end{array} =: b_1$$

$$c_{(1,1,2,2,1),(1,1,2,2,3)} = c_{(1,1,1,2,2),(1,1,3,2,2)} \mapsto \begin{array}{|ccc|}\hline 1 & 1 & 3 \\ \hline 2 & 2 \\ \cline{1-2}\end{array} =: b_2$$

$$c_{(2,1,1,1,2),(1,1,2,2,3)} = c_{(1,1,1,2,2),(1,2,2,1,3)} \mapsto \begin{array}{|ccc|}\hline 1 & 2 & 2 \\ \hline 1 & 3 \\ \cline{1-2}\end{array} =: b_3$$

$$c_{(2,1,2,1,1),(1,1,2,2,3)} = c_{(1,1,1,2,2),(1,2,3,1,2)} \mapsto \begin{array}{|ccc|}\hline 1 & 2 & 3 \\ \hline 1 & 2 \\ \cline{1-2}\end{array} =: b_4$$

$$c_{(2,2,1,1,1),(1,1,2,2,3)} = c_{(1,1,1,2,2),(2,2,3,1,1)} \mapsto \begin{array}{|ccc|}\hline 2 & 2 & 3 \\ \hline 1 & 1 \\ \cline{1-2}\end{array} =: b_5$$

Für die Erzeugung einer Basis von $D^{\lambda^\#,\lambda,\alpha}$ benötigen wir Sequenzen aus $\mathrm{Seq}_\alpha(\lambda^\#,\lambda)$. Man betrachtet also nur diejenigen Sequenzen aus $\mathrm{Seq}(\lambda)$, die entlang der α-Blöcke monoton fallend sind, und überprüft, ob diese in $\mathrm{Seq}(\lambda^\#,\lambda)$ und somit in $\mathrm{Seq}_\alpha(\lambda^\#,\lambda)$ enthalten sind.

1.3.2 Beispiel:

Es sei λ, α wie oben und $\lambda^\# = (2,1)$. Wir müssen nur die Monome in 2. Normalform m_1, m_2, m_3, m_4 und m_5 aus dem vorherigem Beispiel betrachten und diese als Sequenzen auffassen:

$$(1,1,2,1,2) \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$$
$$(1,1,2,2,1) \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$$
$$(2,1,1,1,2) \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$$
$$(2,1,2,1,1) \in \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$$
$$(2,2,1,1,1) \notin \mathrm{Seq}_\alpha(\lambda^\#,\lambda)$$

Die Basis von $D^{\lambda^\#,\lambda,\alpha}$ ist also

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(1,1,2,1,2))} = \begin{array}{|ccc|}\hline 1&1&2\\ \hline 2&3\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 2&1&2\\ \hline 1&3\\ \cline{1-2}\end{array}$$

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(1,1,2,2,1))} = \begin{array}{|ccc|}\hline 1&1&3\\ \hline 2&2\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 2&1&3\\ \hline 1&2\\ \cline{1-2}\end{array}$$

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(2,1,1,1,2))} = \begin{array}{|ccc|}\hline 1&2&2\\ \hline 3&1\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 3&2&2\\ \hline 1&1\\ \cline{1-2}\end{array}$$

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(2,1,2,1,1))} = \begin{array}{|ccc|}\hline 1&2&3\\ \hline 2&1\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 2&2&3\\ \hline 1&1\\ \cline{1-2}\end{array}$$

Unter Verwendung der 2. Normalform haben die den Polytabloiden entsprechenden Monomsummen die folgende Form:

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(1,1,2,1,2))} \mapsto |11|21|2| - |21|11|2|$$

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(1,1,2,2,1))} \mapsto |11|22|1| - |21|21|1|$$

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(2,1,1,1,2))} \mapsto |21|11|2| - |22|11|1|$$

$$e^{\lambda^\#,\lambda}_{sub_\alpha(T\cdot(2,1,2,1,1))} \mapsto |21|21|1| - |22|11|1|$$

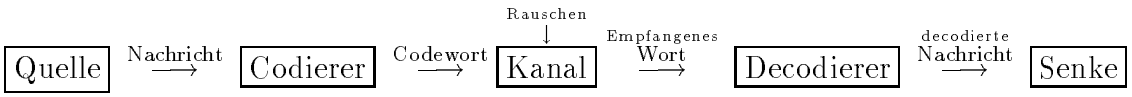
Die Algorithmen zur Ermittlung der Basen von $M^{\lambda,\alpha}$ und $D^{\lambda^\#,\lambda,\alpha}$, wurden in C implementiert und können jederzeit zur Verfügung gestellt werden.

Kapitel 2

Codierungstheoretische Anwendungen

2.1 Einführung in algebraische Codierungstheorie

In diesem Kapitel werden zunächst die allgemeinen Grundkonzepte der allgebraischen Codierungungstheorie vorgestellt, dabei werden weitgehend Definitionen von Hill [1] verwendet. Dann befassen wir uns mit linearen Codes über endlichen Körpern.
Ziel der Codierungstheorie ist es, Nachrichten möglichst fehlerfrei zu übertragen. Wir haben folgendes Model:



Die Aufgabe der Codierungstheorie besteht nun darin, die durch das Rauschen des Kanals entstehenden Übertragungsfehler zu erkennen und wenn möglich, zu korrigieren.

2.1.1 Definitionen:

a) Eine endliche nichtleere Menge

$$A = \{a_1, a_2, \ldots, a_k\}$$

heißt **Alphabet**, ihre Elemente sind **Buchstaben**.

b) Eine endliche Folge von Buchstaben

$$x = (x_1, x_2, \ldots, x_n)\ ;\ \ x_i \in A, \text{ für alle } i \in \underline{n}$$

heißt **Wort der Länge n**. Die Menge aller Wörter der Länge n wird mit A^n bezeichnet.

c) Ein **Code über A** ist eine nichtleere Teilmenge von A^n.

•

Um Übertragungsfehler näher untersuchen zu können, müssen wir sie meßbar machen.

2.1.2 Definitionen:

a) Die Abbildung $d : A^n \times A^n \to \mathbb{R}$,

$$d(x,y) = \mid \{i \in \underline{n} \mid x_i \neq y_i\} \mid ;\ x = (x_1, \ldots, x_n),\ y = (y_1, \ldots, y_n) \in A^n$$

heißt der **Hamming-Abstand** zwischen x und y.

b) Sei $C \subseteq A^n$ ein Code. Dann bezeichnet die Zahl

$$d_C \quad = \quad min\{d(u,v) \mid u, v \in C,\ u \neq v\}$$

die **Minimaldistanz von C** .

•

2.1.3 Lemma: Der Hamming-Abstand ist eine Metrik, d.h.

a) $d(x,y) = 0$, genau dann, wenn $x = y$.

b) $d(x,y) = d(y,x)$, für alle $x, y \in A^n$.

c) $d(x,y) \leq d(x,z) + d(z,y)$, für alle $x, y, z \in A^n$.

Beweis: a) und b) sind klar, zu c):
$d(x,y)$ gibt die kleinste Anzahl der zu ändernden Buchstaben an, wollte man x zu y machen. Man kann aber auch mit $d(x,z) + d(z,y)$ Änderungen zunächst x in z und dann z in y verwandeln. Die Zahl der Änderungen ist dann mindestens $d(x,y)$. □

Wenn ein Codewort x gesendet wurde und der Vektor y empfangen wurde, dann beschreibt $d(x,y)$ die Anzahl der aufgetretenen Fehler.

2.1.4 Satz: Sei $C \subseteq A^n$ ein Code.

a) C kann bis zu s Fehler erkennen, wenn $d_C \geq s + 1$.

b) C kann bis zu t Fehler korrigieren, wenn $d_C \geq 2t + 1$.

Beweis:

a) Sei $d_C \geq s + 1$ und x ein Codewort, bei dessen Übertragung höchstens s Fehler aufgetreten sind. Der empfangene Vektor kann kein von x verschiedenes Codewort sein und so können eventuelle Übertragungfehler erkannt werden.

b) Sei $d_C \geq 2t+1$, x das gesendete Codewort und y der empfangene Vektor, der sich von x in höchstens t Stellen unterscheidet, d.h. $d(x,y) \leq t$. Ist x' ein anderes Codewort als x, dann ist $d(x',y) \geq t+1$, da andernfalls $d(x',x) \leq d(x,y) + d(x',y) \leq 2t$ gelten würde, was ein Widerspruch zu $d_C \geq 2t+1$ ist.
Das bedeutet, daß x das Codewort ist, das zu y am nächsten ist. Decodiert man y jetzt als x, hat man eventuell aufgetretene Fehler korrigiert. □

2.1.5 Beispiel: Sei $A = \{0,1\}$, $n = 9$. Die Nachrichten seien Tripel (x_1,x_2,x_3); $x_i \in A$; $i = 1,2,3$. Die Codierung bestehe darin, die 3-Tupel dreimal zu wiederholen, d.h. das Codewort für die Nachricht (x_1,x_2,x_3) ist $(x_1,x_2,x_3,x_1,x_2,x_3,x_1,x_2,x_3)$. Die Minimaldistanz ist drei, der Code ist also in der Lage zwei Fehler zu erkennen und einen Fehler zu korrigieren.

$$\text{Nachricht} = (1,0,1) \longrightarrow \boxed{\overset{Codierer}{(1,0,1) \mapsto (1,0,1,1,0,1,1,0,1)}} \overset{\text{Rauschen}}{\longrightarrow} (1,0,1,1,0,\overset{\downarrow}{0},1,0,1) \longrightarrow$$

$$\longrightarrow \boxed{\overset{Decodierer}{(1,0,1,1,0,0,1,0,1) \approx (1,0,1,1,0,1,1,0,1) \mapsto (1,0,1)}} \overset{\text{empfangene}}{\longrightarrow} \text{Nachricht} = (1,0,1)$$

Der Decodierer kann, wenn höchstens ein Fehler auftritt, diesen korrigieren. Wäre der empfangene Vektor aber z.B. (1,0,0,1,0,0,1,0,1) gewesen, d.h. es wären zwei Fehler aufgetreten, so hätte der Decodierer zwar erkannt, daß die Übertragung fehlerhaft war, aber er hätte (1,0,0) als Nachricht weitergegeben.

Im Weiteren bestehe unser Alphabet A aus den Elementen eines endlichen Körpers mit q Elementen, bezeichnet mit F_q. Endliche Körper werden im 3. Kapitel ausführlich behandelt.

2.1.6 Definitionen: Sei $n \in \mathbb{N}$.

a) Ein **linearer Code** über F_q ist ein Unterraum eines F_q-Vektorraums.

b) Ein (linearer) **[n,k]-Code** über F_q ist ein k-dimensionaler Unterraum eines n-dimensionalen F_q-Vektorraums.

c) Ein (linearer) **[n,k,d]-Code** über F_q ist ein k-dimensionaler Unterraum eines n-dimensionalen F_q-Vektorraums mit Minimaldistanz d.

•

Im Beispiel handelte es sich also um einen linearen [9,3,3]-Code über F_2.
Im Allgemeinen wird es ein Ziel der Codierungstheorie sein, [n,k,d]-Codes mit kleinem n, möglichst großem k und möglichst großem d zu finden.

2.1.7 Definition: Eine $k \times n$-Matrix, deren Zeilenvektoren eine Basis für einen linearen $[n,k]$-Code darstellen, heißt **Generatormatrix** zu diesem Code. •

2.1.8 Beispiel: Eine Generatormatrix zum [9,3,3]-Code aus Beispiel 1 ist z.B.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Diese Matrix hat darüberhinaus die Eigenschaft, daß die zu übertragende Nachricht mittels Multiplikation codiert werden kann:

$$(x_1, x_2, x_3) \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = (x_1, x_2, x_3, x_1, x_2, x_3, x_1, x_2, x_3).$$

Allgemein werden in der Praxis Matrizen als Codierer linearer Codes verwendet.

2.1.9 Bemerkung: Ist G eine Generatormatrix vom Code C, dann auch jede Matrix G', die aus G mit Hilfe folgender Operationen erzeugt werden kann:

(Z1) Permutationen der Zeilen.

(Z2) Multiplikation einer Zeile mit einem Element $g \in F_q \backslash \{0\}$.

(Z3) Addition des g-fachen einer Zeile zu einer anderen,wobei $g \in F_q$.

Beweis: Nach jeder dieser Operationen spannen die Zeilenvektoren immer noch den Unterraum C auf. □

2.1.10 Definitionen:

a) Zwei lineare Codes C_1 und C_2 heißen **äquivalent**, wenn eine Generatormatrix von C_1 durch folgende Operationen in eine Generatormatrix von C_2 übergeführt werden kann:

(S1) Permutation der Spalten.

(S2) Multiplikation einer Spalte mit einem Element $g \in F_q \backslash \{0\}$.

b) Eine Generatormatrix G eines [n,k]-Codes heißt **in Standardform** , wenn

$$G = [I_k, A],$$

wobei I_k die $k \times k$-Einheitsmatrix ist und A eine $k \times n-k$-Matrix.

•

Die besondere Bedeutung der Standardform von Generatormatrizen wird deutlich, wenn man mit Hilfe dieser die Codierung von Nachrichten vornimmt. Sei C ein $[n,k]$-Code, $y = (y_1, \ldots, y_k)$, $y_1, \ldots, y_k \in F_q$ ein Vektor mit einer zu codierenden Nachricht, $G = [I_k, A]$ die Generatormatrix, mit der codiert wird. Dann enthält der zu y gehörige Codevektor y' wegen

$$y' = y[I_k, A]$$

in den ersten k Einträgen die Nachricht. Alle weiteren Einträge sind Prüfeinträge, mit deren Hilfe man, falls Fehler bei der Übertragung auftreten, diese aufspüren und gegebenenfalls korrigieren kann.

2.1.11 Bemerkungen:

a) Äquivalente Codes stimmen in ihren codierungstheoretischen Eigenschaften überein, d.h. sie haben die gleiche Dimension und die gleiche Minimaldistanz.

b) Jede Generatormatrix kann mit den Operationen (Z1), (Z2), (Z3), (S1) und (S2) in Standardform gebracht werden.

Beweis:

a) Es seien $z_1, \ldots, z_k$ die Zeilenvektoren von C. (S1) bedeutet für die Zeilenvektoren, daß ihre Einträge alle in der gleichen Weise permutiert werden. Bei einem beliebigen Codevektor x werden somit nur die Einträge permutiert, d.h. weder Dimension noch Minimaldistanz verändert.
(S2) ändert nicht den Rang von G, d.h. die Dimensionen äquivalenter Codes sind gleich. (S2) entspricht bei den Codevektoren einer Multiplikation eines Eintrages mit einem von 0 verschiedenem Element aus F_q. Der Hamming-Abstand beliebiger Codevektoren und somit die Minimaldistanz bleiben dadurch unbeeinflußt.

b) Man geht wie folgt vor:
Man permutiert die Spalten, bis in der ersten Zeile der erste Eintrag verschieden von 0 ist. Man teilt die erste Zeile durch diesen Eintrag und setzt ihn somit auf 1. Alle weiteren Einträge dieser Spalte werden auf 0 gebracht, indem man entsprechende Vielfache der ersten Zeile von den übrigen Zeilen abzieht.
Dieses Verfahren wiederholt man für die übrigen Zeilen $2, 3, \ldots, k$ und erhält so die gewünschte Form. □

Wir haben gesehen, wie mit Hilfe einer Generatormatrix Nachrichten codiert werden. Zur Angabe eines einfachen Decodierungsverfahrens sind folgende Definitionen nötig.

2.1.12 Definitionen:

a) Sei $n \in \mathbb{N}$, $x \in F_{q^n}$ ein Vektor. Das **Gewicht** $\mathrm{wgt}(x)$ wird definiert als die Anzahl der von 0 verschiedenen Einträge von x.

b) Sei $n \in \mathbb{N}$, C ein linearer $[n,k]$-Code über F_q. Eine **(Slepian) Standardmatrix** S_C zu C ist eine $q^{n-k} \times q^n$-Matrix, die wie folgt konstruiert wird:

   i) Man schreibt in die erste Zeile alle Codewörter aus C, wobei man mit dem Codewort 0 beginnt.

   ii) Man wähle einen Vektor $v_2 \in F_{q^n} \backslash C$, der minimales Gewicht hat.

   iii) Für jede Spalte trage man in die zweite Zeile $v_2 + w$, wobei w der Vektor aus der ersten Zeile ist.

   iv) Dies wiederhole man für die dritte Zeile mit $v_3 \in F_{q^n} \backslash (C \cup (v_2 + C))$ usw.

•

2.1.13 Bemerkung: Jeder Vektor aus F_{q^n} kommt in S_C genau einmal vor, da die Nebenklassen $a + C$ und $b + C$ für $b \notin a + C$ disjunkt sind und $F_{q^n} = (0 + C) \cup (v_1 + C) \cup (v_2 + C) + \ldots$.
Für die **Maximum-Likelihood Decodierung** geht man davon aus, das die Anzahl der Übertragungsfehler gering ist. Man geht mit einem empfangenen Vektor y wie folgt vor:

Ermittlung der Spalte i von S_C, in der y steht.

Den Vektor x ausgeben, der in S_C in der ersten Zeile in der Spalte i steht.

Es gilt:

$$d(y, x') \geq d(y, x) \text{ für alle } x' \in C$$

Beweis: y stehe in der j-ten Zeile von S_C, d.h. $y \in v_j + C$. Nach Konstruktion von S_C ist $x = y - v_j$. Um ein Codewort zu erhalten, muß man von y einen Vektor $v' \in v_j + C$ abziehen. v_j ist in $v_j + C$ ein Vektor von minimalem Gewicht, es gilt also

$$d(y, x') = wgt(y - x') \overset{y-x' \in v_j+C}{\geq} wgt(v_j) = d(y, x).$$

□

2.1.14 Beispiel: Wir betrachten den [6,2,3]-Code C über F_2 mit der Generatormatrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Die ersten drei Zeilen einer Standardmatrix zu C sind z.B.

$$\begin{array}{cccc} (0,0,0,0,0,0) & (1,0,1,0,1,0) & (0,1,0,1,0,1) & (1,1,1,1,1,1) \\ (1,0,0,0,0,0) & (0,0,1,0,1,0) & (1,1,0,1,0,1) & (0,1,1,1,1,1) \\ (1,1,0,0,0,0) & (0,1,1,0,1,0) & (1,0,0,1,0,1) & (0,0,1,1,1,1) \end{array}$$

Ist ein empfanger Vektor z.B. $y = (1,1,0,1,0,1)$, so ist $(0,1,0,1,0,1)$ das Codewort mit dem geringsten Abstand zu y.

2.2 James Gewichtsräume als Codes

Wir untersuchen die Eigenschaften von James Gewichtsräumen als Codes.
Es sei m eine natürliche Zahl, $(\lambda^{\#}, \lambda)$ ein Paar von Partitionen für m, α eine uneigentliche Partition von m in höchstens m Teile, $\omega = (1^m)$ die Partition von m in genau m Teile und $K = F_q$ ein endlicher Körper mit q Elementen. Darüberhinaus sei $n = \dim(M^{\lambda,\alpha})$ und $k = \dim(D^{\lambda^{\#},\lambda,\alpha})$.
Es werden $[n,k]$-Codes über F_q betrachtet, wobei $M^{\lambda,\alpha}$ der F_q-Vektorraum ist, der den linearen $[n,k]$-Code $D^{\lambda^{\#},\lambda,\alpha}$ enthält.

2.2.1 Beispiel: Sei $\lambda = (3,2)$, $\alpha = (2,2,1)$ und $\lambda^{\#} = (2,1)$ wie in Beispiel 1.3.1 und Beispiel 1.3.2. Dort hatten wir als Basis für $M^{\lambda,\alpha}$

$$b_1 = \begin{array}{|ccc|}\hline 1&1&3\\ \hline 2&2\\ \cline{1-2}\end{array},\quad b_2 = \begin{array}{|ccc|}\hline 1&1&3\\ \hline 2&2\\ \cline{1-2}\end{array},\quad b_3 = \begin{array}{|ccc|}\hline 1&2&2\\ \hline 1&3\\ \cline{1-2}\end{array},\quad b_4 = \begin{array}{|ccc|}\hline 1&2&3\\ \hline 1&2\\ \cline{1-2}\end{array} \text{ und } b_5 = \begin{array}{|ccc|}\hline 2&2&3\\ \hline 1&1\\ \cline{1-2}\end{array},$$

und als Basis für $D^{\lambda,\lambda^{\#},\alpha}$

$$p_1 = \begin{array}{|ccc|}\hline 1&1&2\\ \hline 2&3\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 1&2&2\\ \hline 1&3\\ \cline{1-2}\end{array},\quad p_2 = \begin{array}{|ccc|}\hline 1&1&3\\ \hline 2&2\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 1&2&3\\ \hline 1&2\\ \cline{1-2}\end{array},$$

$$p_3 = \begin{array}{|ccc|}\hline 1&2&2\\ \hline 3&1\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 2&2&3\\ \hline 1&1\\ \cline{1-2}\end{array} \text{ und } p_4 = \begin{array}{|ccc|}\hline 1&2&3\\ \hline 2&1\\ \cline{1-2}\end{array} - \begin{array}{|ccc|}\hline 2&2&3\\ \hline 1&1\\ \cline{1-2}\end{array}.$$

In dieser Nummerierung sieht die Generatormatrix dieses [5,4]-Codes wie folgt aus:

$$\begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Mit λ, α und $\lambda^{\#}$ haben wir 3 Parameter, die Einfluß auf n, k und die Minimaldistanz d haben. Im Anhang sind einige Tabellen zum Verhalten von n, k und d in Abhängigkeit von λ, α und $\lambda^{\#}$ zu finden. Bei den Untersuchungen der Codes konnten verschiedene Codeklassen bestimmt werden, die 'gute' Eigenschaften besitzen und die im Folgenden beschrieben werden.

2.2.1 MDS - Codes

2.2.2 Korollar: Für jeden $[n,k,d]$- Code C gilt die **Singleton-Schranke**

$$d \leq n-k+1.$$

Beweis: Da n, k und d äquivalenter Codes gleich sind (siehe Bemerkung 2.1.11a)) können wir o.B.d.A. davon ausgehen, das zu C eine Generatormatrix G in Standardform existiert. Die Zeilen von G sind eine Basis von C und somit Codevektoren. Für jede Zeile gilt, daß von den ersten k Einträgen genau einer verschieden von 0 ist. Ist also y ein beliebiger Zeilenvektor, so gilt

$$d = d_C \leq d(y,0) \leq n-k+1.$$

□

2.2.3 Definition: Ein **MDS - Code** (**M**aximum **D**istance **S**eperabel) ist ein $[n,k,d]$-Code, für den die Singleton-Schranke genau erfüllt ist, d.h. $d = n-k+1$ gilt. •

Seien $r,s \in \mathbb{N}$ mit $r \geq s$. Für dieses Unterkapitel sei

$$\lambda = (r,r), \lambda^{\#} = (s,s) \text{ und } \alpha = (r,r).$$

Wir befassen uns zunächst mit der Ermittlung von $n = \dim(M^{\lambda,\alpha})$. Dafür betrachten wir die λ-Tabloide vom Inhalt α:

$$\underline{\begin{array}{ccc}1 & \ldots & 1\\ 2 & \ldots & 2\end{array}},\ \underline{\begin{array}{cccc}1 & \ldots & 1 & 2\\ 1 & 2 & \ldots & 2\end{array}},\ \underline{\begin{array}{ccccc}1 & \ldots & 1 & 2 & 2\\ 1 & 1 & 2 & \ldots & 2\end{array}},\ \ldots,\ \underline{\begin{array}{ccc}2 & \ldots & 2\\ 1 & \ldots & 1\end{array}}.$$

Wie leicht zu erkennen ist, gibt es genau $r+1$ verschiedene λ-Tabloide vom Inhalt α. Diese wollen wir mit $b_0,\ldots,b_r$ in obiger Reihenfolge durchnumerieren, d.h. b_i ist das Tabloid mit i 1'en in der zweiten Zeile.
Um $k = \dim(D^{\lambda^{\#},\lambda,\alpha})$ zu bestimmen, müssen wir $\mid \mathrm{Seq}_{\alpha}(\lambda^{\#},\lambda) \mid$ ermitteln. Dafür werden λ-Sequenzen betrachtet, die entlang der α-Blöcke monoton fallend sind. Damit eine solche Sequenz aus $\mathrm{Seq}_{\alpha}(\lambda^{\#},\lambda)$ ist, muß sie mindestens s 'gute' 2'en im zweiten α-Block enthalten, da 2'en im ersten α-Block immer vor den 1'en stehen, also 'schlecht' sind. Jede 2 im zweiten α-Block ist gut, denn stehen t 2'en im zweiten α-Block, dann sind $r-t$ 1'en im zweiten und die restlichen t 1'en im ersten α-Block. Somit besteht $\mathrm{Seq}_{\alpha}(\lambda^{\#},\lambda)$ aus

$$j_0 = (\overbrace{1,\ldots,1}^{r},\overbrace{2,\ldots,2}^{r}),\ j_1 = (2,\overbrace{1,\ldots,1}^{r-1},\overbrace{2,\ldots,2}^{r-1},1),\ldots,$$

$$j_{r-s} = (\overbrace{2,\ldots,2}^{r-s},\overbrace{1,\ldots,1}^{s},\overbrace{2,\ldots,2}^{s},\overbrace{1,\ldots,1}^{r-s}).$$

Wir haben also einen $[r+1, r-s+1]$-Code über $M^{\lambda,\alpha}$.

Um die Generatormatrix dieses Codes aufstellen zu können, müssen wir

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot j_i)} \text{ für alle } i = 0,\ldots,r-s$$

bestimmen. Die ersten i Stellen von j_i sind mit 'schlechten' 2'en belegt, die laut Konstruktionsvorschrift in $sub_\alpha(T\cdot j_i)$ die letzten i Stellen der zweiten Zeile mit 1'en füllen. Alle anderen Elemente in j_i sind 'gut', so daß gilt:

$$sub_\alpha(T\cdot j_i) = \begin{array}{cc} \overbrace{1\ 1\ \ldots\ 1}^{r-i} & \overbrace{2\ 2\ \ldots\ 2}^{i} \\ \underbrace{2\ 2\ \ldots\ 2}_{r-i} & \underbrace{1\ 1\ \ldots\ 1}_{i} \end{array}.$$

Für $i = 0,\ldots,r-s$ stehen in $sub_\alpha(T\cdot j_i)$ in den ersten s Spalten jeweils in der ersten Zeile 1'en und in der zweiten Zeile 2'en. $C_T(\lambda^{\#}) =< (1,r+1),(2,r+2),\ldots,(s,r+s) >$ ist somit ein Erzeugnis von Transpositionen, die 1'en der ersten mit 2'en der zweiten Zeile vertauschen.
$\{sub_\alpha(T\cdot j_i)\}$ ist das λ-Tabloid vom Inhalt α, das i 1'en in der zweiten Spalte enthält, d.h.

$$\{sub_\alpha(T\cdot j_i)\} = b_i.$$

Jede aus l Transpositionen bestehende Permutation $\pi \in C_T(\lambda^{\#})$ bringt genau l 1'en in die zweite Zeile. Es ist somit $\{\pi sub_\alpha(T\cdot j_i)\} = b_{i+l}$. Es gibt $\left(\begin{array}{c} s \\ l \end{array}\right)$ Permutationen in $C_T(\lambda^{\#})$, die aus l Transpositionen bestehen, d.h.

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot j_i)} = \sum_{l=0}^{s}(-1)^l \left(\begin{array}{c} s \\ l \end{array}\right) b_{i+l}.$$

Die Generatormatrix G zu diesem Code hat bezüglich der Basis $(b_i)_{i=0,\ldots,r}$ die Form

$$G = (g_{i,j})_{\substack{i=0,\ldots,r-s \\ j=0,\ldots,r}} \quad \text{mit } g_{i,j} = (-1)^{j-i}\left(\begin{array}{c} s \\ j-i \end{array}\right).$$

2.2.4 Beispiel: Sei $r = 5$ und $s = 3$. Die Basis von $M^{\lambda,\alpha}$ ist

$$b_0 = \begin{array}{|ccccc|}\hline 1&1&1&1&1\\ 2&2&2&2&2\\ \hline\end{array},\ b_1 = \begin{array}{|ccccc|}\hline 1&1&1&1&2\\ 1&2&2&2&2\\ \hline\end{array},\ b_2 = \begin{array}{|ccccc|}\hline 1&1&1&2&2\\ 1&1&2&2&2\\ \hline\end{array},$$

$$b_3 = \begin{array}{|ccccc|}\hline 1&1&2&2&2\\ 1&1&1&2&2\\ \hline\end{array},\ b_4 = \begin{array}{|ccccc|}\hline 1&2&2&2&2\\ 1&1&1&1&2\\ \hline\end{array},\ b_5 = \begin{array}{|ccccc|}\hline 2&2&2&2&2\\ 1&1&1&1&1\\ \hline\end{array}.$$

$\mathrm{Seq}_{(5,5)}((3,3),(5,5))$ besteht aus den Sequenzen $j_1 = (1,1,1,1,1,2,2,2,2,2)$, $j_2 = (2,1,1,1,1,2,2,2,2,1)$ und $j_3 = (2,2,1,1,1,2,2,2,1,1)$.

$$e^{(3,3),(5,5)}_{sub_{(5,5)(T\cdot j_1)}} = \begin{array}{|ccccc|}\hline 1&1&1&1&1\\ 2&2&2&2&2\\ \hline\end{array} - \begin{array}{|ccccc|}\hline 2&1&1&1&1\\ 1&2&2&2&2\\ \hline\end{array} - \begin{array}{|ccccc|}\hline 1&2&1&1&1\\ 2&1&2&2&2\\ \hline\end{array} -$$

$$- \begin{array}{|ccccc|}\hline 1&1&2&1&1\\ 2&2&1&2&2\\ \hline\end{array} + \begin{array}{|ccccc|}\hline 2&2&1&1&1\\ 1&1&2&2&2\\ \hline\end{array} + \begin{array}{|ccccc|}\hline 2&1&2&1&1\\ 1&2&1&2&2\\ \hline\end{array} +$$

$$+ \begin{array}{|ccccc|}\hline 1&2&2&1&1\\ 2&1&1&2&2\\ \hline\end{array} - \begin{array}{|ccccc|}\hline 2&2&2&1&1\\ 1&1&1&2&2\\ \hline\end{array} = b_0 - 3b_1 + 3b_2 - b_3,$$

$$e^{(3,3),(5,5)}_{sub_{(5,5)(T\cdot j_2)}} = b_1 - 3b_2 + 3b_3 - b_4,\ e^{(3,3),(5,5)}_{sub_{(5,5)(T\cdot j_3)}} = b_2 - 3b_3 + 3b_4 - b_5.$$

Wir erhalten als Generatormatrix

$$G = \begin{bmatrix} 1 & -3 & 3 & -1 & 0 & 0 \\ 0 & 1 & -3 & 3 & -1 & 0 \\ 0 & 0 & 1 & -3 & 3 & -1 \end{bmatrix}.$$

2.2.5 Satz: Ein $[n,k]$-Code über F_q mit der Generatormatrix

$$G = (g_{i,j})_{\substack{i=1,\ldots,n-k \\ j=1,\ldots,n}} \quad \text{mit } g_{i,j} = (-1)^{j-i} \binom{n-k}{j-i},$$

hat die Minimaldistanz $d = n - k + 1$, falls die Charakteristik von F_q größer als $n-1$.

Beweis: Siehe Zimmermann [8]. □

Somit ist gezeigt, daß es sich bei den hier behandelten Codes um MDS - Codes handelt.

2.2.2 Majoritätslogik-decodierbare Codes

2.2.6 Definitionen:

a) Es sei C ein $[n,k]$-Code, dann ist der zugehörige Orthogonalraum ein $[n,n-k]$-Code und wird mit C^- bezeichnet.

b) Ist $c \in C^-$, dann heißt die Gleichung

$$\sum_{i=1}^{n} c_i x_i = 0$$

Parity-Check Gleichung zu C .

•

2.2.7 Satz: Es sei C ein linearer $[n,k]$-Code, bei dem für alle Positionen $j = 1,\ldots,n$ gilt:
Es gibt m Parity-Check Gleichungen $(l = 1,\ldots,m)$ $\sum_{i=1}^{n} c_{i,l}^{j} x_i = 0$, $(c_{1,l}^{j},\ldots,c_{n,l}^{j}) \in C^-$ mit:

a) $c_{j,l}^{j} = 1$ für alle $l \in \underline{m}$.

b) Für alle j und alle $i \neq j$ sei aus der Koeffizientenmenge $\{c_{i,1}^{j},\ldots,c_{i,m}^{j}\}$ maximal ein Element verschieden von 0.

Für die Minimaldistanz von C gilt dann:

$$d_C \geq m+1.$$

Beweis: Es genügt zu zeigen, daß für alle Codevektoren das Gewicht echt größer als m ist, denn ist dies erfüllt, dann ist gilt für zwei beliebige Codevektoren $y,z \in C$ mit $y \neq z$

$$d(y,z) = wgt(z-y) \geq m+1, \text{ da } z-y \in C.$$

Sei nun $y \neq 0$ ein Codevektor mit $y_j \neq 0$. Für alle $l \in \underline{m}$ kann die Parity-Check Gleichung

$$\sum_{i=1}^{n} c_{i,l}^{j} y_i = 0$$

wegen $c_{j,l}^{j} y_j = y_j \neq 0$ nur dann erfüllt sein, wenn für mindestens ein $i \neq j$ das Produkt $c_{i,l}^{j} y_i \neq 0$.
Da nach b) jedes y_i in höchstens einer der m Parity-Check Gleichungen mit einem Koeffizienten verschieden von 0 versehen ist, müssen außer y_j mindestens m Einträge von y verschieden von 0 sein. □

2.2.8 Definition: Ein Code, der die Voraussetzungen von Satz 2.2.7 erfüllt, heißt **vollständig in einem Schritt orthogonalisierbar** . •

2.2.9 Bemerkung: Erfolgt die Codierung von Nachrichten eines $[n,k]$-Codes mit einer Generatormatrix in Standardform, dann müssen die Voraussetzungen von Satz 2.2.7 nur für $j = 1,\ldots,k$ erfüllt sein, da für jeden von 0 verschiedenen Codevektor mindestens eine der ersten k Stellen verschieden von 0 ist.
Ist nun C ein vollständig in einem Schritt orthogonalisierbarer $[n,k]$-Code und wird darüberhinaus die Codierung von Nachrichtenvektoren $z = (z_1,\ldots,z_k),\ z_1,\ldots,z_k \in F_q$ durch Multiplikation mit einer Generatormatrix in Standardform vorgenommen, dann enthalten die ersten k Stellen eines Codevektors die Nachricht und man kann folgendes Decodierungsschema angeben: Es sei x ein Codevektor. Löst man die Parity-Check Gleichungen für die erste Position nach x_1 auf, erhält man die Gleichungen

$$\begin{array}{lcccccc} x_1 & = & x_1 & & & & \\ x_1 & = & -c^1_{2,1}x_2 & - & \ldots & - & c^1_{n,1}x_n \\ & & \vdots & & & & \vdots \\ x_1 & = & -c^1_{2,m}x_2 & - & \ldots & - & c^1_{n,m}x_n \end{array},$$

wobei nach Vorraussetzung jedes x_i mit $i > 1$ in maximal einer Gleichung einen von 0 verschiedenen Koeffizienten hat.
Wird nun ein Vektor y empfangen, dann setzt man ihn für x ein und decodiert als erstes Element der Nachricht den Wert, der aufgrund dieser Gleichungen am häufigsten für y_1 ermittelt wird (d.h. den Wert, der die Majorität hat). Es sei m gerade. Geht man davon aus, daß maximal $\frac{m}{2}$ Fehler aufgetreten sind, dann erhält man als y_1 mindestens $\frac{m}{2}+1$-mal den korrekten Wert und hat mit dieser Decodierung eventuell aufgetreten Fehler korrigiert (ist m ungerade, kann man $\frac{m-1}{2}$ Fehler korrigieren). Das gleiche Verfahren wendet man auch auf die übrigen $k-1$ Nachrichteneinträge an.
C ist dann **majoritätslogik-decodierbar**.

2.2.10 Beispiel: Wir betrachten den [9,3,3]-Code C mit der Generatormatrix aus Beispiel 2.1.8

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Für die erste Position haben wir die Parity-Check Gleichungen

$$x_1 - x_4 = 0 \text{ und } x_1 - x_7 = 0,$$

da $c_1 = (1,0,0,-1,0,0,0,0,0)$ und $c_2 = (1,0,0,0,0,0,-1,0,0)$ wegen $Gc_1^T = Gc_2^T = 0$ aus dem Orthogonalraum von C sind. Derartige Gleichungen lassen sich natürlich auch für die zweite und die dritte Position finden.
Sei nun $y = (1,1,0,1,1,0,0,1,0)$ der empfangene Vektor. Da mit

$$y_1 = y_1 = 1,\ y_1 = y_4 = 1 \text{ und } y_1 = y_7 = 0$$

der Wert 1 die Majorität hat, wird dieser als erste Stelle des Nachrichtenvektors decodiert. Auf die gleiche Weise verfährt man mit der zweiten und dritten Position. Dieses Decodierungsprinzip ist aber schon wegen des Codierungsverfahrens, mit dem wir für dieses Beispiel ausgegangen sind, offensichtlich (siehe Beispiel 2.1.5).

Wong [7] zeigt, daß Spechtmoduln (d.h. $\lambda^{\#} = \lambda$ und $\alpha = \omega$) als Codes vollständig in einem Schritt orthogonalisierbar sind, falls λ eine Partition in zwei Teile ist. Der Beweis zu dieser Aussage liefert einen Algorithmus zur Berechnung der Parity-Check Gleichungen, der in C implementiert wurde.

2.2.11 Beispiel: Sei $\lambda = (2,2)$. Die Basis von $M^{(2,2),(1^4)}$ ist dann

$$b_1 = \overline{\underline{\begin{array}{cc}1&2\\3&4\end{array}}},\ b_2 = \overline{\underline{\begin{array}{cc}1&3\\2&4\end{array}}},\ b_3 = \overline{\underline{\begin{array}{cc}1&4\\2&3\end{array}}},\ b_4 = \overline{\underline{\begin{array}{cc}2&3\\1&4\end{array}}},\ b_5 = \overline{\underline{\begin{array}{cc}2&4\\1&3\end{array}}},\ b_6 = \overline{\underline{\begin{array}{cc}3&4\\1&2\end{array}}}.$$

Der Code $D^{(2,2),(2,2),(1^4)}$ hat die Basis

$$p_1 = \overline{\underline{\begin{array}{cc}1&2\\3&4\end{array}}} - \overline{\underline{\begin{array}{cc}3&2\\1&4\end{array}}} - \overline{\underline{\begin{array}{cc}1&4\\3&2\end{array}}} + \overline{\underline{\begin{array}{cc}3&4\\1&2\end{array}}},\ p_2 = \overline{\underline{\begin{array}{cc}1&3\\2&4\end{array}}} - \overline{\underline{\begin{array}{cc}2&3\\1&4\end{array}}} - \overline{\underline{\begin{array}{cc}1&4\\2&3\end{array}}} + \overline{\underline{\begin{array}{cc}2&4\\1&3\end{array}}}.$$

Als Generatormatrix erhalten wir somit

$$G = \begin{bmatrix} 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & 1 & -1 & -1 & 1 & 0 \end{bmatrix}.$$

Die Gleichungen

$$x_1 + x_2 + x_3 = 0,\ \ x_1 + x_4 + x_5 = 0 \ \text{ und } \ x_1 - x_6 = 0$$

sind Parity-Check Gleichungen für die erste Position, da

$$G\begin{pmatrix} 1\\1\\1\\0\\0\\0 \end{pmatrix} = G\begin{pmatrix} 1\\0\\0\\1\\1\\0 \end{pmatrix} = G\begin{pmatrix} 1\\0\\0\\0\\0\\-1 \end{pmatrix} = \begin{pmatrix} 0\\0 \end{pmatrix}$$

Die Parity-Check Gleichungen zur zweiten Position lauten

$$x_2 + x_1 + x_3 = 0,\ \ x_2 + x_4 + x_6 = 0 \ \text{ und } \ x_1 - x_5 = 0.$$

Wird die Codierung von Nachrichtenvektoren durch Multipliktion mit G vorgenommen, dann ist dieser Code Majoritäts-logik decodierbar. Da einerseits zu jeder Nachrichten-Position drei Parity-Check Gleichungen existieren und andererseits das Gewicht der Basisvektoren von $D^{(2,2),(2,2),(1^4)}$ gleich 4 ist, hat dieser Code die Minimaldistanz 4.

2.2.3 Binäre Reed-Muller Codes

Die in diesem Unterkapitel behandelten Codes sind immer Codes über F_2. Die bei Polytabloiden auftretenden Minuszeichen können also vernachlässigt werden.
Um zu zeigen, daß für bestimmte $\lambda^{\#}, \lambda$ und α die von uns untersuchten Vektorräume den binären Reed-Muller Codes entsprechen, werden diese zunächst definiert.

2.2.12 Definition: Seien $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n) \in (F_2)^n$. Wir definieren das Produkt

$$a * b := (a_1 b_1, \ldots, a_n b_n).$$

•

Im Weiteren sei $\mathbf{1} = (1, \ldots, 1)$ der Vektor der Länge n, der an jeder Stelle eine 1 hat und $m, r \in I\!N$ mit $r \leq m$.

2.2.13 Definition: Es sei $n = 2^m$. Wir stellen eine $m \times n$-Matrix auf, bei der die Spalten aus allen verschiedenen Vektoren aus $(F_2)^m$ bestehen. Die Zeilenvektoren dieser Matrix werden mit v_1 bis v_m durchnummeriert. Der binäre Reed-Muller Code der Ordnung r und der Länge $n = 2^m$ ist der Unterraum von $(F_2)^n$, der durch $\mathbf{1}$ und alle Vektoren die durch Produkte von maximal r Vektoren aus $\{v_1, \ldots, v_m\}$ entstehen, erzeugt wird. Dieser Code wird mit **RM(r,m)** bezeichnet. •

2.2.14 Beispiel: Wir betrachten RM(2,3). Die Matrix, deren Zeilenvektoren v_1, v_2 und v_3 festlegen, lautet

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Sei also $v_1 = (0,0,0,0,1,1,1,1)$, $v_2 = (0,0,1,1,0,0,1,1)$ und $v_3 = (0,1,0,1,0,1,0,1)$. RM(2,3) wird erzeugt von den Vektoren $\mathbf{1}$,$v_1, v_2, v_3, v_1 * v_2, v_1 * v_3$ und $v_2 * v_3$.

2.2.15 Satz: Der binäre Reed-Muller Code RM(r,m) ist ein $[2^m, \sum_{i=0}^{r} \binom{m}{i}, 2^{m-r}]$-Code über F_2.

Beweis: Siehe MacWilliams [5] Kapitel 13. □

Für den Rest dieses Kapitels sei $l \leq m$ eine natürliche Zahl, $\lambda^{\#} = (l, l)$, $\lambda = (m, m)$ und $\alpha = (m, 1^m)$. Wir wollen zeigen, daß der James-Gewichtsräum $D^{\lambda^{\#},\lambda,\alpha}$ über F_2 dem Reed-Muller Code RM(m-l,m) entspricht. Dazu wollen wir $M^{\lambda,\alpha}$ und $D^{\lambda^{\#},\lambda,\alpha}$ näher untersuchen.
Betrachtet man das Tabloid

1	1	...	1
2	3	...	$m+1$

aus $M^{\lambda,\alpha}$, gibt es offensichtlich $\left(\begin{array}{c} m \\ 1 \end{array} \right)$ verschiedene λ-Tabloide vom Inhalt α, bei denen genau ein von 1 verschiedenes Element in der ersten Zeile steht, $\left(\begin{array}{c} m \\ 2 \end{array} \right)$, bei denen zwei von 1 verschiedene Elemente in der ersten Zeile stehen usw. Wir erhalten also

$$\dim(M^{\lambda,\alpha}) = \sum_{i=0}^{m} \left(\begin{array}{c} m \\ i \end{array} \right) = 2^m.$$

Ferner ist jedes λ-Tabloid vom Inhalt α dadurch bestimmt, welche der Elemente aus $\{2, \ldots, m+1\}$ in der ersten Zeile stehen.
Die verschiedenen λ-Tabloide vom Inhalt α seien im Weiteren die **Standardbasis** von $M^{\lambda,\alpha}$.
Um die Dimension von $D^{\lambda^\#,\lambda,\alpha}$ zu bestimmen, betrachten wir die monotone Sequenz

$$m_\lambda = (\overbrace{1, \ldots, 1}^{\alpha_1}, \overbrace{2}^{\alpha_2}, \ldots, \overbrace{2}^{\alpha_{m+1}}) \in \mathrm{Seq}_\alpha(\lambda^\#, \lambda).$$

Wie bei den Maximum-Distance Codes untersuchen wir hier, welche der λ-Sequenzen, die entlang der α-Blöcke monoton fallend sind, mindestens l gute 2'en enthalten. Analog zu den Maximum-Distance Codes sind das genau diejenigen Sequenzen, die im ersten α-Block nicht mehr als $m - l$ 2'en stehen haben. Ausgehend von m_λ gibt es $\left(\begin{array}{c} m \\ i \end{array} \right)$ Möglichkeiten, i 2'en aus den Blöcken α_2 bis α_{m+1} in den ersten Block zu bringen. Folglich besteht $\mathrm{Seq}_\alpha(\lambda^\#, \lambda)$ aus genau $\sum_{i=0}^{m-l} \left(\begin{array}{c} m \\ i \end{array} \right)$ Sequenzen, so daß gilt

$$\dim(D^{\lambda^\#,\lambda,\alpha}) = \sum_{i=0}^{m-l} \left(\begin{array}{c} m \\ i \end{array} \right).$$

Somit ist gezeigt, daß $D^{\lambda^\#,\lambda,\alpha}$ ein $[2^m, \sum_{i=0}^{m-l} \left(\begin{array}{c} m \\ i \end{array} \right)]$-Code ist, also zumindest darin mit dem Reed-Muller Code RM(m-l,m) übereinstimmt. Die Gleicheit dieser Codes wird zunächst für zwei Spezialfälle gezeigt.

2.2.16 Satz:

a) Der James-Gewichtsraum $D^{\lambda,\lambda,\alpha}$ entspricht dem Reed-Muller Code RM(0,m).

b) Der James-Gewichtsraum $D^{(m-1,m-1),\lambda,\alpha}$ stimmt mit dem Reed-Muller Code RM(1,m) überein.

Beweis:

a) Es genügt nach Definition von RM(0,m), zu zeigen, daß $D^{\lambda,\lambda,\alpha}$ den Vektor **1** enthält. $\mathrm{Seq}_\alpha(\lambda, \lambda)$ besteht nur aus m_λ.

$$sub_\alpha(T \cdot m_\lambda) = \begin{array}{cccc} 1 & 1 & \ldots & 1 \\ 2 & 3 & \ldots & m+1 \end{array},$$

d.h. $C_T(\lambda) = C_T$ ist für dieses Tableau das Erzeugnis aller Transpositionen, die die von 1 verschiedenen Elemente aus der zweiten Zeile mit 1'en aus der ersten Zeile vertauschen. Somit steht jedes λ-Tabloide vom Inhalt α genau einmal in der Summe

$$j = \sum_{\pi \in C_T(\lambda)} \{\pi sub_\alpha(T \cdot m_\lambda)\},$$

d.h. $j = \mathbf{1}$ (wegen $K = F_2$ darf $\mathrm{sgn}(\pi)$ vernachlässigt werden).

b) Wir bezeichnen die Sequenz aus $\mathrm{Seq}_\alpha((m-1, m-1), \lambda)$, die im Block α_1 eine 2 und im Block α_{1+i} eine 1 stehen hat, mit s_i. Es gibt genau m solcher Sequenzen.

$$sub_\alpha(T \cdot s_i) = \begin{array}{cccc} 1 & \dots & 1 & i+1 \\ 2 & \dots & m+1 & 1 \end{array},$$

wobei in der letzten Spalte die 1 der zweiten Zeile nach Kontruktion von der 'schlechten' 2 an der ersten Stelle von s_i verursacht wird, und die $i+1$ der ersten Zeile dadurch, daß (wie gesehen im Lemma 1.2.13) die Elemente aus dem Block α_{i+1} auf $i+1$ abgebildet werden. Da $C_T(m-1, m-1)$, angewandt auf das Tableau $sub_\alpha(T \cdot s_i)$, alle Spalten außer der letzten permutiert, besteht für alle $i \in \underline{m}$ der Träger von

$$b_i := \sum_{\pi \in C_T(m-1, m-1)} \{\pi sub_\alpha(T \cdot s_i)\} \in D^{(m-1,m-1),\lambda,\alpha}$$

genau aus den Tabloiden, die in der ersten Zeile eine $i+1$ stehen haben.
Um zu zeigen, daß die b_i den Vektoren v_j aus der Definition der Reed-Muller Codes entsprechen, betrachten wir die Matrix A, die aus den Zeilenvektoren b_i, $i = 1, \dots, m$ besteht:
Wie wir oben bereits festgestellt haben, wird jedes λ-Tabloid vom Inhalt α durch die von 1 verschiedenen Elemente, die in der ersten Zeile stehen, eindeutig festgelegt. Es können daher zwei verschiedene λ-Tabloide vom Inhalt α nicht zu genau den gleichen Trägern der b_i gehören. Die 2^m Spalten von A sind demnach verschieden, d.h. es sind alle verschiedenen Vektoren aus $(F_2)^m$.
Es bleibt zu zeigen, daß $\mathbf{1} \in D^{(m-1,m-1),\lambda,\alpha}$.
$C_T((m-1, m-1))$, angewandt auf

$$sub_\alpha(T \cdot m_\lambda) = \begin{array}{cccc} 1 & 1 & \dots & 1 \\ 2 & 3 & \dots & m+1 \end{array},$$

ergibt alle λ-Tabloide vom Inhalt α, die in der zweiten Zeile eine $m+1$ stehen haben. Somit besteht der Träger von

$$j = s_m + \sum_{\pi \in C_T((m-1,m-1))} \{\pi sub_\alpha(T \cdot m_\lambda)\}$$

aus allen Tabloiden, die eine $m+1$ in der ersten oder in der zweiten Zeile stehen haben, d.h. $j = \mathbf{1}$. □

Diesen Satz können wir jetzt verallgemeinern.

2.2.17 Satz: Für alle $l = 0, \ldots, m$ stimmt der James-Gewichtsraum $D^{(l,l),\lambda,\alpha}$ mit dem Reed-Muller Code $\mathrm{RM}(m-l, m)$ überein.

Beweis: Es genügt zu zeigen, daß das Erzeugendensystem von Reed-Muller Codes in $D^{(l,l),\lambda,\alpha}$ enthalten ist, da die Dimensionen, wie bereits gezeigt, übereinstimmen. Der Menge $\{v_1, \ldots, v_m\}$ aus der Definition 2.2.13 entspricht die Menge $B^* := \{b_1, \ldots, b_m\}$ aus dem Beweis zu Satz 2.2.16 b). Es muß also gezeigt werden, daß alle Produkte von maximal $m-l$ Vektoren aus B^* in $D^{(l,l),\lambda,\alpha}$ sind.
Absteigende Induktion über l:

$l = m-1$: Es wurde im Beweis von Satz 2.2.16 gezeigt, daß die Vektoren aus B^* in $D^{(m-1,m-1),\lambda,\alpha}$ sind.

$l \to l-1$: $D^{(l,l),\lambda,\alpha}$ enthält nach Induktionsvorraussetzung alle Produkte von maximal $m-l$ Vektoren aus B^*, d.h. mit Korollar 1.2.16 sind alle diese Produkte auch in $D^{(l-1,l-1),\lambda,\alpha}$. Zu zeigen ist somit nur, daß Produkte von $m-l+1$ verschiedenen Vektoren aus B^* in $D^{(l-1,l-1),\lambda,\alpha}$ sind. Da wir Codes über F_2 betrachten, ist mit $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n) \in (F_2)^n$ $(n = 2^m)$ das Produkt $a * b$ der Vektor, der als Träger die Schnittmenge der Träger von a und b hat.

Wie im Beweis von Satz 2.2.16 gesehen, hat b_i als Träger alle λ-Tabloide vom Inhalt α, die in der ersten Zeile eine $i+1$ stehen haben. Sei

$$b = b_{i_1} * \ldots * b_{i_{m-l+1}},\ i_1, \ldots, i_{m-l+1} \in \underline{m} \text{ mit } i_r < i_s \text{ für } r < s.$$

b ist dann der Vektor, dessen Träger alle λ-Tabloide enthält, bei denen die Zahlen aus $I := \{i_1+1, i_2+1, \ldots, i_{m-l+1}+1\}$ in der ersten Zeile stehen.

Es sei $j \in \mathrm{Seq}_\alpha((l-1, l-1), \lambda)$ diejenige Sequenz, die im ersten α-Block $m-l+1$ 2'en, $l-1$ 1'en und in den Blöcken $\alpha_{i_1+1}, \ldots, \alpha_{i_{m-l+1}+1}$ 1'en als Bilder hat. Die übrigen α-Blöcke sind mit 2'en belegt.

Es ist

$$t = sub_\alpha(T \cdot j) = \begin{array}{cccccc} \overbrace{1 \ldots 1}^{l-1} & i_1+1 & i_2+1 & \ldots & i_{m-l+1}+1 \\ \underbrace{* \ldots *}_{l-1} & 1 & 1 & \ldots & 1 \end{array},$$

wobei alle Zahlen aus $\{2, \ldots, m+1\} \backslash I$, die in aufsteigender Reihenfolge in den ersten $l-1$ Spalten der zweiten Zeile stehen, mit $*$ gekennzeichnet wurden.

$C_T((l-1, l-1))$ permutiert jeweils die zwei Elemente aus den ersten $l-1$ Spalten, so daß

$$\sum_{\pi \in C_T((l-1,l-1))} \{\pi t\} = e^{(l-1,l-1),\lambda}_{sub_\alpha(T \cdot j)} \in D^{(l-1,l-1),\lambda,\alpha}$$

als Träger alle λ-Tabloide vom Inhalt α enthält, die in der ersten Zeile die Zahlen aus I stehen haben, so daß $b \in D^{(l-1,l-1),\lambda,\alpha}$. Also ist jedes Produkt von $m-l+1$ Vektoren aus B^* in $D^{(l-1,l-1),\lambda,\alpha}$.

□

Es gelten also alle Aussagen für Reed-Muller Codes auch für James-Gewichtsräume mit den Parametern $\lambda^{\#} = (l,l), \lambda = (m,m)$ und $\alpha = (m, 1^m)$. Die Minimaldistanz dieser James-Gewichtsräume ist folglich gleich 2^l (siehe Satz 2.2.15).

Reed-Muller Codes haben zudem Eigenschaften, die den Eigenschaften der in einem Schritt orthogonalisierbaren Codes aus dem letzten Unterkapitel gleichen. Reed-Muller Codes $\mathrm{RM}(m-l,m)$ und somit James-Gewichtsräume $D^{(l,l),(m,m),(m,1^m)}$ sind in l Schritten majoritäts-logikdecodierbar (siehe MacWilliams [5] Kapitel 13.7.). Diese Eigenschaft wird im abschließenden Beispiel erklärt.

2.2.18 Beispiel: Sei $\lambda^{\#} = (2,2)$, $\lambda = (3,3)$ und $\alpha = (3,1^3)$. Die Basis von $M^{\lambda,\alpha}$ ist:

$$b_1 = \overline{\begin{array}{ccc}1&1&1\\2&3&4\end{array}},\ b_2 = \overline{\begin{array}{ccc}2&1&1\\1&3&4\end{array}},\ b_3 = \overline{\begin{array}{ccc}1&3&1\\2&1&4\end{array}},\ b_4 = \overline{\begin{array}{ccc}1&1&4\\2&3&1\end{array}}$$

$$b_5 = \overline{\begin{array}{ccc}2&3&1\\1&1&4\end{array}},\ b_6 = \overline{\begin{array}{ccc}2&1&4\\1&3&1\end{array}},\ b_7 = \overline{\begin{array}{ccc}1&3&4\\2&1&1\end{array}},\ b_8 = \overline{\begin{array}{ccc}2&3&4\\1&1&1\end{array}}.$$

Als Basis von $D^{\lambda^{\#},\lambda,\alpha}$ erhalten wir

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot(1,1,1,2,2,2))} = \overline{\begin{array}{ccc}1&1&1\\2&3&4\end{array}} + \overline{\begin{array}{ccc}2&1&1\\1&3&4\end{array}} + \overline{\begin{array}{ccc}1&3&1\\2&1&4\end{array}} + \overline{\begin{array}{ccc}2&3&1\\1&1&4\end{array}} = b_1+b_2+b_3+b_5$$

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot(2,1,1,1,2,2))} = \overline{\begin{array}{ccc}1&1&2\\3&4&1\end{array}} + \overline{\begin{array}{ccc}3&1&2\\1&4&1\end{array}} + \overline{\begin{array}{ccc}1&4&2\\3&1&1\end{array}} + \overline{\begin{array}{ccc}3&4&2\\1&1&1\end{array}} = b_2+b_5+b_6+b_8$$

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot(2,1,1,2,1,2))} = \overline{\begin{array}{ccc}1&1&3\\2&4&1\end{array}} + \overline{\begin{array}{ccc}2&1&3\\1&4&1\end{array}} + \overline{\begin{array}{ccc}1&4&3\\2&1&1\end{array}} + \overline{\begin{array}{ccc}2&4&3\\1&1&1\end{array}} = b_3+b_5+b_7+b_8$$

$$e^{\lambda^{\#},\lambda}_{sub_\alpha(T\cdot(2,1,1,2,2,1))} = \overline{\begin{array}{ccc}1&1&4\\2&3&1\end{array}} + \overline{\begin{array}{ccc}2&1&4\\1&3&1\end{array}} + \overline{\begin{array}{ccc}1&3&4\\2&1&1\end{array}} + \overline{\begin{array}{ccc}2&3&4\\1&1&1\end{array}} = b_4+b_6+b_7+b_8$$

Als Generatormatrix haben wir somit

$$\begin{bmatrix} 1&1&1&0&1&0&0&0\\ 0&1&0&0&1&1&0&1\\ 0&0&1&0&1&0&1&1\\ 0&0&0&1&0&1&1&1 \end{bmatrix}.$$

Eine Generatormatrix in Standardform erhalten wir, indem wir die zweite und die dritte Zeile zu der ersten addieren:

$$G := \begin{bmatrix} 1&0&0&0&1&1&1&0\\ 0&1&0&0&1&1&0&1\\ 0&0&1&0&1&0&1&1\\ 0&0&0&1&0&1&1&1 \end{bmatrix}.$$

Die Vektoren

$p_1 = (1,1,1,0,1,0,0,0)$ $p_2 = (1,1,0,1,0,1,0,0)$ $p_3 = (1,1,0,0,0,0,1,1)$
$p_4 = (1,0,1,0,0,1,0,1)$ $p_5 = (1,0,1,1,0,0,1,0)$ $p_6 = (1,0,0,1,1,0,0,1)$

sind, wegen $Gp_i^T = 0$, für alle $i = 1,\ldots,6$ aus dem Orthogonalraum von $D^{\lambda^\#,\lambda,\alpha}$.
Für alle Codevektoren gelten also folgende Parity-Check Gleichungen:

$$\begin{array}{rcrcrcll} (x_1+x_2) & + & x_3 & + & x_5 & = & 0 & (p_1)\\ (x_1+x_2) & + & x_4 & + & x_6 & = & 0 & (p_2)\\ (x_1+x_2) & + & x_7 & + & x_8 & = & 0 & (p_3) \end{array}$$

$$\begin{array}{rcrcrcll} (x_1+x_3) & + & x_2 & + & x_5 & = & 0 & (p_1)\\ (x_1+x_3) & + & x_6 & + & x_8 & = & 0 & (p_4)\\ (x_1+x_3) & + & x_4 & + & x_7 & = & 0 & (p_5) \end{array}$$

$$\begin{array}{rcrcrcll} (x_1+x_4) & + & x_2 & + & x_6 & = & 0 & (p_2)\\ (x_1+x_4) & + & x_3 & + & x_7 & = & 0 & (p_5)\\ (x_1+x_4) & + & x_5 & + & x_8 & = & 0 & (p_6) \end{array}$$

Es sei y der empfangene Vektor. Für die Summen $z_1 := y_1 + y_2$, $z_1 := y_1 + y_3$ und $z_3 := y_1 + y_4$ haben wir wegen obiger Parity-Check Gleichungen je vier voneinander unabhängige Ausdrücke, nämlich

$$\begin{array}{rrcl} z_1: & y_1 & + & y_2\\ & y_3 & + & y_5\\ & y_4 & + & y_6\\ & y_7 & + & y_8 \end{array}$$

$$\begin{array}{rrcl} z_2: & y_1 & + & y_3\\ & y_2 & + & y_5\\ & y_6 & + & y_8\\ & y_4 & + & y_7 \end{array}$$

$$\begin{array}{rrcl} z_3: & y_1 & + & y_4\\ & y_2 & + & y_6\\ & y_3 & + & y_7\\ & y_5 & + & y_8 \end{array}$$

Im ersten Schritt zur Decodierung der ersten Koordinate des Nachrichtenvektors decodiert man die Summen z_1, z_2 und z_3 per Mayoritätslogik. Der zweiten Schritt besteht darin, (ebenfalls mit Mayoritätslogik) y_1 mit Hilfe der Ausdrücke

$$y_1,\ \ z_1 + y_2,\ \ z_2 + y_3 \text{ und } z_3 + y_4$$

zu decodieren. Falls nicht mehr als ein Fehler bei der Übertragung aufgetreten ist, wurde für y_1 der Wert ermittelt, der auch gesendet wurde. Analoge Gleichungen lassen sich auch für die anderen drei Nachrichtenvektoren finden, so daß $D^{\lambda^\#,\lambda,\alpha}$ in 2 Schritten mayoritätslogik-decodierbar ist.

Kapitel 3

Endliche Körper

3.1 Theoretische Grundlagen für endliche Körper in Normalbasenrepräsentation

In diesem Kapitel wird die Existenz von trace-kompatiblen Elementen und trace-kompatiblen Polynomen gezeigt. Es wird ferner beschrieben, wie mit Hilfe dieser Polynome die Einbettung endlicher Körper in Oberkörper implementiert werden kann.
Sei F_q ein endlicher Körper mit q Elementen (siehe Lidl/Niederreiter [4] Kapitel 2.1) und $\overline{F_q}$ sein algebraischer Abschluß.

3.1.1 Definitionen:

a) $\alpha \in \overline{F_q}$ heißt **normal in** F_{q^n} **über** F_q , wenn die Konjugierten $\{\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}}\}$ von α eine Basis von F_{q^n} über F_q darstellen. Diese Basis heißt **Normalbasis** .

b) $f \in F_q[X]$ vom Grad m heißt **normal über** F_q, wenn eine Wurzel $\alpha \in F_{q^n}$ von f normal in F_{q^n} über F_q ist.

•

Im folgenden bezeichne $I \subseteq I\!N$ eine unter Teilbarkeit abgeschlossene Menge, d.h. mit $n \in I$ sind auch alle Teiler von n in I enthalten.

3.1.2 Definitionen:

a) Seien $m, d \in I\!N$, d ein Teiler von m und $\alpha \in F_{q^n}$. Die **Trace-Funktion** $T_{m:d}$: $F_{q^m} \to F_{q^d}$ ist definiert durch

$$T_{m:d}(\alpha) := \alpha + \alpha^{q^d} + \alpha^{q^{2d}} + \ldots + \alpha^{q^{m-d}} = \sum_{i=0}^{r-1} \alpha^{q^{id}}, \; wobei \; r = \frac{m}{d}.$$

b) Eine Folge $(\alpha_n)_{n\in I}$ von Elementen $\alpha_n \in \overline{F}_q$ heißt **trace-kompatibel über** F_q, wenn für alle $n \in I$ gilt :

1. α_n ist normal in F_{q^n} über F_q und
2. $T_{n:d}(\alpha_n) = \alpha_d$ für alle Teiler d von n.

c) Eine Folge $(f_n)_{n\in I}$ von Polynomen $f_n \in F_q[X]$, $deg(f_n) = n$, heißt **trace-kompatibel über** F_q,wenn für alle $n \in I$ gilt :

1. f_n ist normal über F_q und
2. es existiert eine Wurzel $\alpha \in F_{q^n}$ von f_n, so daß $T_{n:d}(\alpha)$ eine Wurzel von f_d ist, für alle Teiler d von n.

•

3.1.3 Bemerkungen:

a) In Definition 2 a) gilt $T_{m:d}(\alpha) \in F_{q^d}$, da

$$(\sum_{i=0}^{r-1} \alpha^{q^{id}})^{q^d} = \sum_{i=0}^{r-1} \alpha^{q^{id}q^d} = \alpha^{q^d} + \alpha^{q^{2d}} + \ldots + \alpha^{q^m} = \quad (wegen\ \alpha^{q^m} = \alpha)$$

$$= \alpha + \alpha^{q^d} + \alpha^{q^{2d}} + \ldots + \alpha^{q^{m-d}} = \sum_{i=0}^{r-1} \alpha^{q^{id}},$$

d.h. $T_{m:d}(\alpha)$ ist Wurzel von $X^{q^d} - X$, und somit aus F_{q^d}.

b) Es gilt für $\alpha, \beta \in F_{q^n}$ wegen $\sum_{i=0}^{\frac{m}{d}-1} (\alpha+\beta)^{q^i} = \sum_{i=0}^{\frac{m}{d}-1} \alpha^{q^i} + \beta^{q^i}$:

$$T_{m:d}(\alpha + \beta) = T_{m:d}(\alpha) + T_{m:d}(\beta)$$

c) Die Trace-Funktion ist transitiv, d.h. mit $d \mid m \mid n$, $\alpha \in F_{q^n}$ gilt:

$$T_{n:d}(\alpha) \;=\; T_{m:d}(T_{n:m}(\alpha))$$

$$\begin{aligned} \textit{Beweis: } T_{n:d}(\alpha) = \; & \alpha + \alpha^{q^d} + \ldots + \alpha^{q^{m-d}} + \\ & + \alpha^{q^m} + \alpha^{q^{m+d}} + \ldots + \alpha^{q^{2m-d}} + \\ & \vdots \qquad\quad \vdots \qquad\qquad \vdots \\ & + \alpha^{q^{n-m}} + \alpha^{q^{n-m+d}} + \ldots + \alpha^{q^{n-d}} = \quad (zeilenweise) \end{aligned}$$

$$= T_{m:d}(\alpha) + T_{m:d}(\alpha^{q^m}) + \ldots + T_{m:d}(\alpha^{q^{n-m}}) = \quad (nach\ \text{b}))$$

$$= T_{m:d}(\alpha + \alpha^{q^m} + \ldots + \alpha^{q^{n-m}}) = T_{m:d}(T_{n:m}(\alpha)).$$

□

Nun gilt es, die Existenz von trace-kompatiblen Elementen bzw. Polynomen zu beweisen. Mit den folgenden Definitionen kann man (nach Alfred Scheerhorn [6] Kapitel 2) den Existenzbeweis auf ein Reste-Problem von Polynomen zurückführen.

3.1.4 Definitionen:

a) Die Abbildung $\diamond : F_q[X] \times \overline{F}_q \to \overline{F}_q$

$$\sum_{i=0}^{n} a_i X^i \quad \diamond \quad \alpha \quad := \quad \sum_{i=0}^{n} a_i \alpha^{q^i}$$

heißt **(additiver) Exponent** .

b) Seien $f \in F_q[X]$, $\alpha, \beta \in F_{q^n}$ und $f \diamond \alpha = \beta$. Der **(additive) Logarithmus von β zur Basis α** wird definiert durch

$$\mathrm{Log}_\alpha(\beta) \quad := \quad f \bmod (X^n - 1).$$

c) Für beliebiges $\alpha \in \overline{F}_q$ bezeichne $\mathrm{Ord}(\alpha)$ das normierte Polynom $f \in F_q[X]$ kleinsten Grades mit $f \diamond \alpha = 0$.

•

3.1.5 Bemerkung: Die Existenz von $\mathrm{Ord}(\alpha)$ für $\alpha \in F_{q^n}$ ist klar wegen

$$X^n - 1 \diamond \alpha = \alpha^{q^n} - \alpha = 0.$$

3.1.6 Korollar: Es gilt mit $f, g \in F_q[X]$, $\alpha, \beta \in F_{q^n}$:

a)
$$f \diamond (g \diamond \alpha) = (fg) \diamond \alpha$$

b)
$$\mathrm{Log}_\alpha(f \diamond \beta) = f \, \mathrm{Log}_\alpha(\beta) \bmod (X^n - 1)$$

c)
$$\alpha \text{ normal in } F_{q^n} \text{ über } F_q \iff \mathrm{Ord}(\alpha) = X^n - 1$$

Beweis:

a) Sei $f = \sum_{i=0}^{n} a_i X^i$ und $g = \sum_{j=0}^{m} b_j X^j$ mit $a_i, b_j \in F_q$, dann ist

$$f \diamond (g \diamond \alpha) = f \diamond \sum_{j=0}^{m} b_j \alpha^{q^j} = \sum_{i=0}^{n} a_i \left(\sum_{j=0}^{m} b_j \alpha^{q^j} \right)^{q^i} \overset{b_j \in F_q}{=} \sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j \alpha^{q^{i+j}} =$$

$$= \left(\sum_{i=0}^{n} \sum_{j=0}^{m} a_i b_j X^{i+j} \right) \diamond \alpha = \left(\left(\sum_{i=0}^{n} a_i X^i \right) \left(\sum_{j=0}^{m} b_j X^j \right) \right) \diamond \alpha = (fg) \diamond \alpha.$$

b) Sei $\beta = g \diamond \alpha$, d.h. $\mathrm{Log}_\alpha(\beta) = g \bmod (X^n - 1)$, damit ist

$$\mathrm{Log}_\alpha(f \diamond \beta) \stackrel{a)}{=} \mathrm{Log}_\alpha((fg) \diamond \alpha)) = fg \bmod (X^n - 1) = f \,\mathrm{Log}_\alpha(\beta) \bmod (X^n - 1).$$

c) $\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}}$ sind genau dann linear unabhängig, wenn die Gleichung

$$\sum_{i=0}^{n-1} a_i \alpha^{q^i} = 0, \; mit \; a_i \in F_q$$

nur die triviale Lösung besitzt, d.h. es kein Polynom $f \in F_q[X]$, $Grad(f) < n$, mit $f \diamond \alpha = 0$ gibt.

□

3.1.7 Korollar: Sei $\gamma \in F_{q^n}$ und α normal in F_{q^n} über F_q. Sei weiterhin m ein Teiler von n, $\beta \in F_{q^m}$ und $f \in F_q[X]$. Es gilt :

a)
$$\frac{X^n - 1}{X^m - 1} \mid \mathrm{Log}_\alpha(\beta).$$

b)
$$\beta = T_{n:m}(\gamma) \Longleftrightarrow \mathrm{Log}_\alpha(\gamma)\frac{X^n - 1}{X^m - 1} \equiv \mathrm{Log}_\alpha(\beta) \bmod (X^n - 1).$$

c) $f \diamond \alpha$ genau dann normal in F_{q^m} über F_q, wenn gilt : $\mathrm{ggT}(f, X^m - 1) = \frac{X^n - 1}{X^m - 1}$.

Beweis:

a) Da $\beta \in F_{q^m}$ haben wir $\beta^{q^m} = \beta$. Das bedeutet für die Darstellung von β bezüglich der Normalbasis $\{\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}}\}$:

$$\begin{array}{rcccccccccc} \beta & = & a_0\alpha & + & \ldots & + & a_m\alpha^{q^m} & + & \ldots & + & a_{n-1}\alpha^{q^{n-1}} & = \\ \beta^{q^m} & = & a_0\alpha^{q^m} & + & \ldots & + & a_m\alpha^{q^{2m}} & + & \ldots & + & a_{n-1}\alpha^{q^{n-1+m}} & = \\ & = & a_{n-m}\alpha & + & \ldots & + & a_0\alpha^{q^m} & + & \ldots & + & a_{n-m-1}\alpha^{q^{n-1}} & \end{array}$$

(m-fache Rechtsrotation der Summenelemente).

Per Koeffizientenvergleich erhält man $a_0 = a_m = a_{2m} = \ldots = a_{n-m}$, ebenso $a_1 = a_{m+1} = a_{2m+1} = \ldots = a_{n-m+1}$ usw., d.h.

$$\beta = a_0\alpha + \ldots + a_{m-1}\alpha^{q^{m-1}} + \ldots\ldots + a_0\alpha^{q^{n-m}} + \ldots + a_{m-1}\alpha^{q^{n-1}}.$$

Somit ist

$$\mathrm{Log}_\alpha(\beta) = a_0 + \ldots + a_{m-1}X^{m-1} + \ldots\ldots + a_0X^{n-m} + \ldots + a_{m-1}X^{n-1}.$$

Mit

$$\frac{X^n - 1}{X^m - 1}(a_0 + \ldots + a_{m-1}X^{m-1}) = (1 + X^m + \ldots + X^{n-m})(a_0 + \ldots + a_{m-1}X^{m-1})$$

folgt die Behauptung.

b) "$\Longrightarrow$":

$$\beta = T_{n:m}(\gamma) = \sum_{i=0}^{\frac{n}{m}-1} \gamma^{q^{im}} = \left(\sum_{i=0}^{\frac{n}{m}-1} X^{im} \right) \diamond \gamma = \frac{X^n - 1}{X^m - 1} \diamond \gamma$$

Den Logarithmus zur Basis α darauf angewant ergibt:

$$\mathrm{Log}_\alpha(\beta) = \mathrm{Log}_\alpha(\frac{X^n - 1}{X^m - 1} \diamond \gamma) \overset{Kor.3.1.6\ b)}{=} \mathrm{Log}_\alpha(\gamma)\frac{X^n - 1}{X^m - 1} \bmod (X^n - 1)$$

"$\Longleftarrow$":

$$\mathrm{Log}_\alpha(\gamma)\frac{X^n - 1}{X^m - 1} \equiv \mathrm{Log}_\alpha(\beta) \bmod (X^n - 1) \overset{Kor.3.1.6\ b)}{\Longrightarrow} \mathrm{Log}_\alpha(\frac{X^n - 1}{X^m - 1} \diamond \gamma) = \mathrm{Log}_\alpha(\beta)$$

$$\Longrightarrow \mathrm{Log}_\alpha(\frac{X^n - 1}{X^m - 1} \diamond \gamma) \diamond \alpha = \mathrm{Log}_\alpha(\beta) \diamond \alpha \overset{Def.3.1.4b)}{\Longrightarrow} \beta = \frac{X^n - 1}{X^m - 1} \diamond \gamma = T_{n:m}(\gamma).$$

c) Nach Korollar 3.1.6 c) reicht es zu zeigen:

$$\mathrm{Ord}(f \diamond \alpha) = X^m - 1 \Longleftrightarrow \mathrm{ggT}(f, X^n - 1) = \frac{X^n - 1}{X^m - 1}.$$

Sei $g = \mathrm{Ord}(f \diamond \alpha)$, d.h. $g \diamond (f \diamond \alpha) = gf \diamond \alpha = 0$.
$\mathrm{Ord}(\alpha) = X^n - 1$ somit gilt für jedes Polynom h, $h \diamond \alpha = 0 \Leftrightarrow X^n - 1 \mid h$ (siehe Lidl/Niederreiter [4] Kapitel 3.4). Da g also das Polynom kleinsten Grades mit $X^n - 1 \mid gf$ ist, haben wir:

$$g = \frac{X^n - 1}{\mathrm{ggT}(f, X^n - 1)}.$$

Folglich gilt $g = X^m - 1 \Leftrightarrow \mathrm{ggT}(f, X^n - 1) = \frac{X^n - 1}{X^m - 1}$

□

3.1.8 Satz: Trace-kompatible Elementfolgen existieren.

Beweis: Sei $I_m \in I\!N$ eine unter Teilbarkeit abgeschlossene Menge, die alle Teiler von m, nicht aber m selbst enthält. Wir setzen jetzt voraus, daß $(\alpha_n)_{n \in I_m}$ eine trace-kompatible Folge von Elementen über F_q ist. Können wir nun die Existenz eines α_m zeigen, mit dem $(\alpha_n)_{n \in I_m \cup \{m\}}$ trace-kompatibel bleibt, dann ist der Satz bewiesen.

α_m muß zwei Bedingungen genügen:

1. α_m ist normal in F_{q^m} über F_q.
2. $T_{m:d}(\alpha_m) = \alpha_d$ für alle Teiler d von m.

Seien $m = \prod_{i=1}^{r} p_i^{e_i}$, $r \geq 1$, $e_i \geq 1$ für alle $i \in \underline{r}$, die Primfaktorzerlegung von m in $\mathbb{N}$, $m_i = m/p_i$ die maximalen Teiler von m und $m_{i,j} = \mathrm{ggT}(m_i, m_j)$ für alle $1 \leq i, j \leq r$. Sei außerdem α normal in F_{q^m} über F_q. Da $T_{m:d}$ transitiv ist, reicht es, wenn $T_{m:m_i}(\alpha_m) = \alpha_{m_i}$ für alle $i \in \underline{r}$ gilt. Wir haben also (mit Korollar 3.1.7) die Existenz eines Polynoms $f(X)$ zu zeigen, das Folgendes erfüllt:

$$\mathrm{ggT}(f(X), X^m - 1) = 1 \tag{3.1.9}$$

$$f(X)\frac{X^m - 1}{X^{m_i} - 1} \equiv \mathrm{Log}_\alpha(\alpha_{m_i}) \bmod (X^m - 1) \text{ für alle } 1 \leq i \leq r. \tag{3.1.10}$$

Wir befassen uns zunächst mit (3.1.10):
$(X^m - 1)/(X^{m_i} - 1)$ teilt $\mathrm{Log}_\alpha(\alpha_{m_i})$, da $\alpha_{m_i} \in F_{q^{m_i}}$ (siehe Korollar 3.1.7 a)). Wir erhalten somit folgendes äquivalentes Kongruenzensystem:

$$f(X) \equiv \mathrm{Log}_\alpha(\alpha_{m_i})\frac{X^{m_i} - 1}{X^m - 1} \bmod (X^{m_i} - 1) \text{ für alle } 1 \leq i \leq r. \tag{3.1.11}$$

Nach dem Chinesischem Restesatz (vgl. Satz 3.12, W.J.LeVeque [3]) ist dies dann und nur dann lösbar, wenn für alle $1 \leq i, j \leq r$ gilt:

$$\mathrm{ggT}(X^{m_i} - 1, X^{m_j} - 1) \;\Big|\; \Big(\frac{X^{m_i} - 1}{X^m - 1}\mathrm{Log}_\alpha(\alpha_{m_i}) - \frac{X^{m_j} - 1}{X^m - 1}\mathrm{Log}_\alpha(\alpha_{m_j})\Big) \ .$$

Da $(\alpha_n)_{n \in I_m}$ trace-kompatibel ist, haben wir

$$T_{m_i:m_{i,j}}(\alpha_{m_i}) = \alpha_{m_{i,j}} = T_{m_j:m_{i,j}}(\alpha_{m_j}),$$

d.h. wegen Korollar 3.1.7 b)

$$\frac{X^{m_i} - 1}{X^{m_{i,j}} - 1}\mathrm{Log}_\alpha(\alpha_{m_i}) \equiv \frac{X^{m_j} - 1}{X^{m_{i,j}} - 1}\mathrm{Log}_\alpha(\alpha_{m_j}) \bmod (X^m - 1).$$

Damit ist

$$\frac{X^{m_i} - 1}{X^m - 1}\mathrm{Log}_\alpha(\alpha_{m_i}) \equiv \frac{X^{m_j} - 1}{X^m - 1}\mathrm{Log}_\alpha(\alpha_{m_j}) \bmod (X^{m_{i,j}} - 1),$$

da $(X^m - 1)/(X^{m_{i,j}} - 1) \mid \mathrm{Log}_\alpha(a_{m_{i,j}})$ (nach Korollar 3.1.7 a)).

Es ist $\mathrm{ggT}(X^{m_i} - 1, X^{m_j} - 1) = X^{m_{i,j}} - 1$ (vgl. Korollar 3.7 S.77,Lidl/Niederreiter [4]), die Bedingungen für den Chinesischen Restesatz sind also erfüllt, und (3.1.11) somit lösbar.
Sei $h(X)$ eine Lösung dieses Systems. Die Menge aller Lösungen ist dann

$$L = \{h(X) + c(X)l(X) \mid c(X) \in F_q[X]\},$$

wobei $l(X) = kgV\{(X^{m_i} - 1) \mid i \in \underline{r}\}$. Da α_{m_i} normal in $F_{q^{m_i}}$ über F_q, haben wir für alle $i \in \underline{r}$:

$$\mathrm{ggT}(\mathrm{Log}_\alpha(\alpha_{m_i}), X^m - 1) = \frac{X^m - 1}{X^{m_i} - 1} \Longrightarrow \mathrm{ggT}(\mathrm{Log}_\alpha(\alpha_{m_i})\frac{X^{m_i} - 1}{X^m - 1}, X^{m_i} - 1) = 1$$

$h(X)$ ist also (wegen (3.1.11)) teilerfremd zu $X^{m_i}-1$ für alle $i \in \underline{r}$, somit auch zu $l(X)$. Können wir nun beweisen, daß es ein $f(X) \in L$ gibt mit $\mathrm{ggT}(f(X), X^m-1)=1$, dann erfüllt $f(X)$ (3.1.9) und (3.1.10), und der Satz ist bewiesen.

Zu zeigen: Es gibt $c(X)$ mit $\mathrm{ggT}(h(X)+c(X)l(X), X^m-1)=1$.
Es sei $p(X)$ ein irreduzibler Faktor von X^m-1. Ist $p(X)$ ein Teiler von $l(X)$, so kann $p(X)$ kein Teiler von $h(X)+c(X)l(X)$ sein, da $h(X)$ und $l(X)$ teilerfremd sind.
Es seien jetzt $p_i(X),\ i=1,\ldots,s$ die irreduziblen Faktoren von X^m-1, die nicht gleichzeitig Teiler von $l(X)$ sind.
Für $c(X)$ mit

(3.1.12) $$c(X)l(X) \equiv 1-h(X) \bmod (p_i(X)), \text{ für alle } i \in \underline{s}$$

gilt, daß kein irreduzibler Faktor von X^m-1 das Polynom $h(X)+c(X)l(X)$ teilt. Wir müssen also nur zeigen, daß (3.1.12) als lineares Kongruenzensystem lösbar ist.

Es sei für $i \in \underline{s}$ $c_i(X)$ die Lösung der i-ten Kongruenz von (3.1.12). Diese Lösung existiert, da $\mathrm{ggT}(l(X), p_i(X))=1$ für alle $i \in \underline{s}$. Darüberhinaus sind alle Lösungen der Kongruenz

$$\overline{c}(X) \equiv c_i(X) \bmod (p_i(X)),$$

Lösungen der i-ten Kongruenz von (3.1.12) (siehe Satz 3.10 W.J.LeVeque [3]).

Wir erhalten ein Kongruenzensystem

$$\overline{c}(X) \equiv c_i(X) \bmod (p_i(X)),\ i \in \underline{s},$$

das nun seinerseits lösbar ist wegen $\mathrm{ggT}(p_i(X), p_k(X)) = 1$, für $i \neq j$ (Chinesischer Restesatz). D.h. wir haben mit eine Lösung von (3.1.12) und so ein Polynom $c(X)$ mit

$$\mathrm{ggT}(h(X)+c(X)l(X), X^m-1)=1.$$

□

3.1.13 Satz: Trace-kompatible Polynomfolgen existieren.

Beweis: Die Folge von Minimalpolynomen einer trace-kompatiblen Elementfolge ist definitionsgemäß trace-kompatibel. □

3.1.14 Lemma: Sei $(f_n)_{n\in I}$ eine trace-kompatible Folge von Polynomen über F_q. Ferner sei d ein Teiler von $m \in I$, α eine Wurzel von f_m und $\beta = T_{m:d}(\alpha)$, d.h. nach Definition eine Wurzel von f_d. Dann gilt für jedes $a \in F_{q^d}$:

$$\begin{aligned} a = a_0\beta + \ldots + a_{d-1}\beta^{q^{d-1}} = {} & a_0\alpha + \ldots + a_{d-1}\alpha^{q^{d-1}} + \\ & + a_0\alpha^{q^d} + \ldots + a_{d-1}\alpha^{q^{2d-1}} + \\ & \vdots \qquad\qquad \vdots \\ & + a_0\alpha^{q^{m-d}} + \ldots + a_{d-1}\alpha^{q^{m-1}}, \end{aligned}$$

mit $a_0, \ldots, a_{d-1} \in F_q$.

Beweis: Nach Definition von $T_{m:d}$ gilt :

$$\beta = T_{m:d}(\alpha) = \alpha + \alpha^{q^d} + \alpha^{q^{2d}} + \ldots + \alpha^{q^{m-d}}.$$

Daraus folgt für alle $i = 0, 1, \ldots, d-1$:

$$\beta^{q^i} = \alpha^{q^i} + (\alpha^{q^d})^{q^i} + (\alpha^{q^{2d}})^{q^i} + \ldots + (\alpha^{q^{m-d}})^{q^i} = \alpha^{q^i} + \alpha^{q^{d+i}} + \alpha^{q^{2d+i}} + \ldots + \alpha^{q^{m-d+i}},$$

womit die Gleichung (spaltenweise) bewiesen ist. □

3.1.15 Bemerkung: Die Einbettung eines Elements a aus F_{q^d} in F_{q^m} führt man nun durch, indem man den Koordinatenvektor $(a_0, a_1, \ldots, a_{d-1})$ für a bezüglich der Normalbasis $\{\beta, \beta^q, \ldots, \beta^{q^{d-1}}\}$ von F_{q^d} über F_q $\frac{m}{d}$-mal wiederholt und diesen dann als Koordinatenvektor bezüglich der Normalbasis $\{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$ von F_{q^m} über F_q auffaßt.

3.2 Die Implementierung für SYMMETRICA

3.2.1 Dokumentation der bereitgestellten Funktionen

Die Funktionen für SYMMETRICA sind in der Text-Datei `UE_doc.txt` beschrieben :

```
Die folgenden Funktionen stellen eine Arithmetik fuer endliche Koerper in
Normalbasenrepraesentation dar. Sie wurde mit Hilfe trace-kompatibler
Polynome implementiert, die Alfreed Scheerhorn vom IBM Scientific Center in
Heidelberg bereitgestellt hat.
Um mit diesen Funktionen arbeiten zu koennen werden die Dateien

       trace_02.pol       trace_03.pol       trace_05.pol
       trace_07.pol       trace_11.pol       trace_13.pol

benoetigt die fuer die jeweilige Charakteristik die trace-kompatiblen
Polynome vom Grad 1-100 enthalten.




   Die Funktion erzmulttafel berechnet die Multiplikationstafel fuer die
   Normalbasis. erzmulttafel wird automatisch durchgefuehrt, falls Addition oder
   Multiplikation zweier Koerperelemente dies erfordern. Sie muss also nur dann
   angesprochen werden, wenn explizit ein bestimmter Erweiterungsgrad gewuenscht
   wird, d.h. wenn z.B. Koerperelemente zufaellig erzeugt werden sollen, die
   einen bestimmten Erweiterungsgrad haben.
   erzmulttafel gibt 1 zurueck, falls die Erzeugung der Multiplikationstafel
   erfolgreich abgeschlossen wurde und 0 sonst.

INT erzmulttafel(Erweiterungsgrad)
INT Erweiterungsgrad;




   Die Funktion UE_Platz stellt ein undefiniertes Koerperelement bereit.

UE_Platz(Koerperzeiger)
INT **Koerperzeiger;



   Mit der Funktion UE_scan werden Koerperelemente eingelesen.

UE_scan(Koerperzeiger)
INT **Koerperzeiger;
```

```
   Die Funktion UE_Zeige gibt ein Koerperelement auf dem Bildschirm aus.

INT UE_Zeige(Koerperzeiger)
INT **Koerperzeiger;



  Die Funktion UE_fZeige gibt ein Koerperelement auf das File f aus.

INT UE_fZeige(f,Koerperzeiger)
FILE *f;
INT **Koerperzeiger;



   Die Funktion minimalErw bettet ein gegebenes Element in den Koerper kleinsten
   Erweiterungsgrades ein, in dem es enthalten ist.

minimalErw(Element)
INT **Element;



   Die Funktion UE_add berechnet die Summe von Summ1zeig und Summ2zeig und
   gibt das Ergebnis dieser Summe auf Ergebzeig aus.
   Der Zeiger Ergebzeig darf dabei nicht auf dasselbe Koerperelement zeigen,
   wie Summ1zeig oder Summ2zeig.

UE_add(Summ1zeig,Summ2zeig,Ergebzeig)
INT **Summ1zeig;
INT **Summ2zeig;
INT **Ergebzeig;



   Die Funktion UE_mult berechnet das Produkt von Fakt1zeig und Fakt2zeig und
   gibt das Ergebnis dieses Produktes auf Ergebzeig aus.
   Der Zeiger Ergebnzeig darf dabei nicht auf dasselbe Koerperelement zeigen,
   wie Fakt1zeig oder Fakt2zeig.

UE_mult(Fakt1zeig,Fakt2zeig,Ergebzeig)
INT **Fakt1zeig;
INT **Fakt2zeig;
INT **Ergebzeig;
```

```
   Die Funktion negativ berechnet das additive Inverse von Element und gibt
   -Element auf Ergebnis aus.
   Der Zeiger Ergebnis darf dabei nicht auf dasselbe Koerperelement zeigen,
   wie Element.

negativ(Element,Ergebnis)
INT **Element,**Ergebnis;
```

```
   Die Funktion UE_hoch berechnet die m-te Potenz von Element und gibt
   Element**m auf Ergebnis.
   Der Zeiger Ergebnis darf dabei nicht auf dasselbe Koerperelement zeigen,
   wie Element.

UE_hoch(Element,m,Ergebnis)
INT **Element;
INT m;
INT **Ergebnis;
```

```
   Die Funktion invers berechnet das multiplikative Inverse zu Element und gibt
   1/Element auf Ergebnis aus.
   Der Zeiger Ergebnis darf dabei nicht auf dasselbe Koerperelement zeigen,
   wie Element.

UE_invers(Element,Ergebnis)
INT **Element,**Ergebnis;
```

```
   Die Funktion ist_null ueberprueft, ob Element dem neutralem Element der
   additiven Gruppe des aktuellen Koerpers entspricht. ist_null gibt 1 zurueck,
   falls dies gilt und 0 sonst.

INT ist_null(Element)
INT **Element;
```

```
   Die Funktion ist_eins ueberprueft, ob Element dem neutralem Element der
   multiplikativen Gruppe des aktuellen Koerpers entspricht. ist_eins gibt 1
   zurueck falls dies gilt und 0 sonst.

INT ist_eins(Element)
INT **Element;
```

```
   Die Funktion Int_Aequivalent berechnet die a-fache Summe des 1-Elementes im
   Grundkoerper und belegt das Koerperelement Ergebnis mit dieser Summe.

Int_Aequivalent(a,Ergebnis)
INT a;
INT **Ergebnis;
```

```
   Die Funktion ist_gleich vergleicht die Koerperelemente Element1 und
   Element2. Dazu werden die Koeffizienten von Element1 und Element2 bezueglich
   der Normalbasis ihres gemeinsammen Erweiterungskoerpers von links nach rechts
   verglichen. Die erste Stelle in der sie sich unterscheiden ist
   ausschlaggebend.
   ist_gleich gibt  1 zurueck, falls Element1 > Element2,
                   -1        , falls Element1 < Element2, und
                    0        , falls Element1 = Element2.

INT ist_gleich(Element1,Element2)
INT **Element1;
INT **Element2;
```

```
   Die Funktion Order berechnet die kleinste Zahl m, fuer die Element**m = 1
   gilt. Order gibt diese Zahl m zurueck.

int Order(Element)
int **Element;
```

```
   Die Funktion UE_Random erzeugt ein zufaelliges Koerperelement aus dem Koerper
   mit der aktuellen Koerpererweiterung. Das Ergebnis wird auf Element
   ausgegeben.

UE_Random(Element)
INT **Element;
```

3.2.2 Quellcode

Die C-Routinen der Implementierung stehen im File `UE_ffields.c` :

```
#define maxgrad 100
typedef long INT;
INT Charakteristik;
INT UE_Erw_Grad=0;
INT *UE_Platz_Mult;
INT **Mult_Tafel;

/*****************************************************************************/
/*        Hilfsfunktionen                                                    */
/*                                                                           */
/*                                              Verfasser : Ulrich Eidt      */
/*                      Stand     : 04.11.91                                 */
/*****************************************************************************/

/*****************************************************************************/
/* Funktion kgv berechnet das kleinste gemeinsamme Vielfache von zwei        */
/*              Integerzahlen.                                               */
/*****************************************************************************/
INT kgv(a,b)
INT a,b;
{
INT c,d;
c = a;
d = b;
while(c && d)
    {
    c = c % d;
    if(c)
        d = d % c;
    }
if(c)
    return(a*b/c);
else
    return(a*b/d);
}



/*****************************************************************************/
/* Funktion sqrt berechnet die abgerundete integer-Wurzel einer Integerzahl.  */
/*****************************************************************************/
INT sqrt(x)
INT x;
    {
    INT i;
    if (x==0)
        return(0);
```

```
    for (i=1;i<x;i++)
        if (i*i > x)
            return(i-1);
    return(1);
    }



/*****************************************************************************/
/* Funktion power berechnet die y-te Potenz von x                            */
/*****************************************************************************/
INT power(x,y)
INT x,y;
{
INT i, s = 1;
    for (i=0;i<y;i++)
        s *= x;
    return(s);
}



/*****************************************************************************/
/* Funktion prim prueft, ob p eine Primzahl ist.                             */
/* Rueckgabewert: 1, falls p Primzahl, 0 sonst.                              */
/*****************************************************************************/
INT prim(p)
INT p;
    {
    INT sqrt();
    INT i, s;

    s = sqrt(p);
    if (p < 2)
        return(0);
    if (p > 2 && ((p % 2) == 0))
        return(0);
    else
        {
        for (i=3;i<=s;i+=2)
            {
            if ((p % i) == 0)
                return(0);
            }
        }
    return(1);
    }



/*****************************************************************************/
/*        Endliche Koerper    :   Grundkoerperarithmetik                     */
```

```
/*                                                                          */
/*                      Verfasser : Ulrich Eidt                             */
/*                      Stand     : 04.11.91                                */
/****************************************************************************/

INT addg(a,b) /* Summation : a,b sind die Summanden (Integerzahlen) */
    {
    if (!b)
        return(a);
    if (!a)
        return(b);
    if (a+b>=Charakteristik)
        return(a+b-Charakteristik);
    return(a+b);
    }

INT subg (a,b) /* Subtraktion : a,b sind Integerzahlen */
    {
    if (!b)
        return(a);
    if (!a)
        return(Charakteristik-b);
    if (a>=b)
        return(a-b);
    return(Charakteristik+a-b);
    }

INT multg(a,b) /* Multiplikation : a,b sind Faktoren (Integerzahlen) */
    {
    if (a && b)
        return((a*b) % Charakteristik);
    return(0);
    }

INT divg(a,b) /* Division : a,b sind Integerzahlen */
    {
    INT j,i,s;
    if (!b)
        {
        printf("Division durch 0\n");
        return(-1);
        }
    if (!a || b==1)
        return(a);
    s = a;
    j = b;
    i = Charakteristik-2;
    while(i>0)
        {
        while(!i % 2)
            {
            i /= 2;
```

```
            j =(j*j) % Charakteristik;
            if (j==1)
                return(s);
            }
        i--;
        s = (s*j) % Charakteristik;
        }
    return(s);
    }


/*****************************************************************************/
/* Gaussalgorithmen : Algorithmen fuer Gleichungssystemloesung               */
/*                                                                           */
/*                                          Verfasser : Ulrich Eidt          */
/*                                          Stand     : 07.11.91             */
/*****************************************************************************/

/*****************************************************************************/
/* Funktion gausszerlegu fuehrt die Dreieckszerlegung PA = LR durch          */
/* Uebergabeparameter : Mat  : ist die Matrix                                */
/*                      n    : Dimension der Matrix                          */
/*                      P    : Platz fuer Permutationsvektor                 */
/*                                                                           */
/* Rueckgabeparameter : 0 falls Matrix singulaer                             */
/*                      1 falls Zerlegung erfolgreich durchgefuehrt          */
/*                                                                           */
/* Bemerkungen : - Die Eintraege der Matrix Mat sind Zeile fuer Zeile von oben*/
/*                  nach unten in Mat eingetragen.                           */
/*               - Die Funktionen add,sub,mult,div werden verwendet.         */
/*                  Diese stehen in EndlArithmet.c                           */
/*****************************************************************************/
INT gausszerlegu(Mat,n,P)
INT **Mat;
INT n;
INT *P;
    {
    INT i,j,k=0,StellePermutation,Zwischenspeicher;

    for (k=0;k<n;k++)
        {

    /* Sollte ein Eintrag = 1 sein, dann sind wir optimal */
    /*  ansonsten nehmen wir den ersten Eintrag <> 0      */
        StellePermutation = -1;
        for (i=k;i<n;i++)
            if (Mat[i][k] == 1)
                {
                StellePermutation = i;
                i = n+1;
                }
        if (StellePermutation < 0)
```

```
            for (i=k;i<n;i++)
                if (Mat[i][k] > 0)
                    {
                    StellePermutation = i;
                    i = n+1;
                    }

    /* Falls kein Eintrag <> 0 : Abbruch da Matrix singulaer */
        if (StellePermutation < 0)
            {
            free((char *) Mat);
            return(0);
            }

    /* Permutation speichern */
        P[k] = StellePermutation;

    /* Vertauschung falls notwendig */
        if (StellePermutation > k)
            for (j=0;j<n;j++)
                {
                Zwischenspeicher = Mat[k][j];
                Mat[k][j] = Mat[StellePermutation][j];
                Mat[StellePermutation][j] = Zwischenspeicher;
                }

    /* Berechnung von L */
        Zwischenspeicher = Mat[k][k];
        if (Zwischenspeicher != 1)
            {
            Zwischenspeicher = divg(1,Zwischenspeicher);
            for (i=k+1;i<n;i++)
                Mat[i][k] = multg(Mat[i][k],Zwischenspeicher);
            }

    /* Spaltenweise Berechnung der Untermatrix */
        for (j=k+1;j<n;j++)
            {
            Zwischenspeicher = Mat[k][j];
            for (i=k+1;i<n;i++)
                Mat[i][j] = subg(Mat[i][j],multg(Mat[i][k],Mat[k][j]));
            }
        }
    if (!Mat[n-1,n-1])
        return(0);

    return(1);
    }


/*****************************************************************************/
```

```
/* Funktion gaussaufloes berechnet die Loesung des Gleichungssystems :        */
/*                         LRx = Pb                                           */
/* Uebergabeparameter : Mat  : ist die Matrix mit den Informationen zu L und R*/
/*                      n    : Dimension der Matrix                           */
/*                      b    : rechte Seite des Permutationsvektors           */
/*                      P    : Permutationsvektor                             */
/*                                                                            */
/* Bemerkungen : - Die Funktion ist ohne Rueckgabeparameter.                  */
/*               - Der Vektor b enthaelt nach der Aktion das Ergebnis         */
/*               - Die Funktionen addg,subg,multg,divg werden verwendet.      */
/******************************************************************************/
INT gaussaufloes(Mat,n,b,P)
INT **Mat;
INT n;
INT *b;
INT *P;
    {
    INT i,k=0,Zwischenspeicher;

/* b := Pb */
    for (k=0;k<n-1;k++)
        if (k<P[k])
            {
            Zwischenspeicher = b[k];
            b[k] = b[P[k]];
            b[P[k]] = Zwischenspeicher;
            }

/* b := L**-1b */
    for (k=0;k<n-1;k++)
        for (i=k+1;i<n;i++)
            b[i] = subg(b[i],multg(Mat[i][k],b[k]));

/* b := R**-1b */
    for (k=n-1;k>=0;k--)
        {
        b[k] = divg(b[k],Mat[k][k]);
        for (i=0;i<=k-1;i++)
            b[i] = subg(b[i],multg(Mat[i][k],b[k]));
        }
    return;
    }


/******************************************************************************/
/* Funktion reduzierpoly fuehrt die Reduktion eines Polynoms modulo eines     */
/* anderen durch. Funktion nur fuer endliche Koerper verwendbar.              */
/*                                                                            */
/* Bemerkungen : - Funktion ohne Rueckgabeparameter.                          */
/*               - Polynome werden in Potenzaufsteigender Reihenfolge         */
/*                  sortiert, beginnend bei 0. ZB. Pol = x**2 + x + 3  wird   */
/*                  uebergeben als : Pol[0] = 3, Pol[1] = 1, Pol[3] = 1.      */
```

```
/*               - ReduzierPolynom wird als normiert vorrausgesetzt, d.h. in  */
/*                  der Funktion wird der Koeffizient der hoechsten Potenz    */
/*                  automatisch als 1 angenommen.                             */
/*               - Polynom wird in der Funktion geaendert und enthaelt        */
/*                  beim Abschluss der Funktion das reduzierte Polynom.       */
/*               - die Funktionen multg,addg,subg werden verwendet.           */
/*                                                                            */
/*                  Autor : Ulrich Eidt                                       */
/*                  Stand : 04.11.91                                          */
/******************************************************************************/
reduzierpoly(Polynom,Grad,ReduzierPolynom,GradReduzierPol)

INT *Polynom;         /* enthaelt das Polynom vor Reduktion         */
INT Grad;             /* Grad des Polynoms vorher                   */
INT *ReduzierPolynom; /* Polynom nach dem reduziert wird            */
INT GradReduzierPol;  /* Grad des Polynoms nach dem reduziert wird  */

    {
    INT Polynomzeiger, i, Stelle, Faktor;

/* falls Grad < GradReduzierPol ohne Aenderung zurueck   */
    if (Grad < GradReduzierPol)
        {
        return;
        }
    Polynomzeiger = Grad;

/* Reduzieren, solange der Grad reduzierten Polynoms < Grad ReduzierPolynoms */
    while (Polynomzeiger >= GradReduzierPol)
        {
        Faktor = Polynom[Polynomzeiger];

    /* Reduzierung nur noetig, falls der Koeffizient <> 0            */
        if (Faktor)
            {
            Polynom[Polynomzeiger] = 0;
            for (i=0;i<GradReduzierPol;i++)
                {
                Stelle = Polynomzeiger+i-GradReduzierPol;

            /* Hier findet die Reduktion statt    */
                Polynom[Stelle] = subg(Polynom[Stelle],
                     multg(Faktor,ReduzierPolynom[i]));
                }
            }
        Polynomzeiger--;
        }
    return;
    }
```

```
/*****************************************************************************/
/*   Funktion lies_tracepol liesst das tracecompatible Polynom zu gegebener   */
/*   Charakteristik und gegebenen Erweiterungsgrad ein.                       */
/*                                                                            */
/*   Es muss zur Charakteristik ** eine Datei mit dem Namen trace_**.pol      */
/*   existieren, die in der i-ten Zeile die Koeffizienten des Polynoms i-ten  */
/*   Grades in aufsteigender Reihenfolge enthaelt.                            */
/*   Pro Koeffizient ist nur eine Stelle vorgesehen, d.h. fuer Koeeffizienten */
/*   >9 ist 10 durch A, 11 durch B, usw. repraesentiert.                      */
/*                                                                            */
/*   Rueckgabewert : 0 , falls das Polynom nicht initialisiert werden konnte, */
/*           1 , sonst.                                                       */
/*                                                                            */
/*                                    Verfasser: Ulrich Eidt                  */
/*                   Stand    : 04.11.91                                     */
/*****************************************************************************/
INT liestracepol(Grad,tracePolynom)
INT Grad;            /* Gibt den gewuenschten Erweiterungsgrad an.            */
INT tracePolynom[];  /* Vektor mit Platz fuer die Eintraege des ermittelten   */
                     /* Polynoms (die i-te Stelle von tracePolynom gibt den   */
                     /* Koeffizienten zu x**i an der hoechste Koeffizient wird*/
                     /* (da normiert) nicht belegt)).                         */
    {
    INT gelesen=1;      /* Anzahl der gelesenen Zeichen, 0 falls Dateiende*/
    INT Pufferzeiger=0; /* Zeiger fuer Bearbeitung des Puffer             */
    INT Dateizeile=1;   /* Zeile der Datei auf deren Anfang man steht     */
                        /* und somit Grad des folgenden Polynoms          */
    INT Datei;          /* Dateiidentifikationsnummer                     */
    INT Polynomzeiger=0;/* Zeiger fuer Position von tracePolynom          */
    char *Puffer;       /* Einlesepuffer                                  */
    char *Dateiname = "trace_00.pol";/* Name der Datei                    */
    char Zeichen;       /* Platz fuer ein Zeichen wird benutzt beim       */
                        /*  Einlesen des Polynoms                         */

/* Pufferplatz freimachen  */
    if (!(Puffer = malloc(512*sizeof(char))))
        {
            printf("Nicht genug Speicherplatz vorhanden!\n");
            printf("Polynom nicht belegbar.\n");
        return(0);
        } /* */

/* Datei oeffnen und erstes Zeichen lesen     */
    Dateiname[6] = 48+(Charakteristik / 10);
    Dateiname[7] = 48+(Charakteristik % 10);
    Datei = open(Dateiname,0);
    if (Datei == -1)
        {
        printf("Die Datei %s ist nicht verfuegbar!\n",Dateiname);
        return(0);
        }
    gelesen = read(Datei,Puffer,1);
```

```
/* Suchen des Anfangs des Polynoms  */
    while(Dateizeile<Grad && gelesen)
        {
        gelesen = read(Datei,Puffer,512);
        Pufferzeiger = 0;
        while (Dateizeile<Grad && Pufferzeiger<512)
            if (Puffer[Pufferzeiger++] == 10)
                Dateizeile++;
        }

/* Initialisierung des Polynoms     */
    if (!gelesen && Dateizeile<Grad) /* d.h. Dateiende erreicht ohne Zeile*/
                                     /* <Grad> zu finden                  */
        {
        printf("Kann das Polynom nicht finden\n");
        close(Datei);
        return(0);
        }

/* Einlesen des Polynoms            */
    while (Polynomzeiger < Grad)  /* Solange nicht mindestes <Grad>      */
                                  /* Koefizienten gelesen worden sind    */
        {

    /* Leerzeichen ignorieren          */
        while (Puffer[Pufferzeiger] == 32 && Pufferzeiger <512)
            Pufferzeiger++;

    /* falls Puffer zuende naechste Zeichen einlesen */
        if (Pufferzeiger >= 512)
            {
            gelesen = read(Datei,Puffer,512);
            Pufferzeiger = 0;
            }

    /* Leerzeichen ignorieren          */
        while (Puffer[Pufferzeiger] == 32 && Pufferzeiger <512)
            Pufferzeiger++;

    /* Zeichen einlesen und Polynom entsprechend initialisieren */
        Zeichen = Puffer[Pufferzeiger++];

    /* Buchstaben */
        if (Zeichen < 91 && Zeichen > 64)
            tracePolynom[Polynomzeiger] = Zeichen - 55;

    /* Ziffern */
        if (Zeichen < 58 && Zeichen > 47)
            tracePolynom[Polynomzeiger] = Zeichen - 48;

    /* Weder noch also nicht defineirt */
```

```
        if (Zeichen < 48 || Zeichen > 90 || (Zeichen>57 && Zeichen<65))
            {
            printf("Undefiniertes Zeichen beim Polynomeinlesen!\n");
            return(0);
            }

    /* Zum naechsten Zeichen */
        Polynomzeiger++;
        }

    return(1);
    }
```

```
/*****************************************************************************/
/* Funktion erzmulttafel berechnet die Multiplikationstafel fuer die         */
/* Normalbasis.                                                              */
/*                                                                           */
/* Uebergabeparameter :                                                      */
/*  - Erweiterungsgrad :  Gibt den gewuenschten Erweiterungsgrad an          */
/*                                                                           */
/* Rueckgabeparameter : 0 falls Tafel nicht erzeugt werden konnte.           */
/*                      1 sonst.                                             */
/*                                                                           */
/* Bemerkungen:                                                              */
/*               - Funktionen, die benutzt werden (also eingebunden sein     */
/*                  muessen) :                                               */
/*                    liestracepol     : aufgerufen von erzmulttafel()       */
/*                                       zu finden in LiesTracePol.c         */
/*                    reduzierpoly     : aufgerufen von erzmulttafel()       */
/*                                       zu finden in ReduzierPoly.c         */
/*                    gausszerlegu     : aufgerufen in erzmulttafel()        */
/*                                       zu finden in GaussAlgorit.c         */
/*                    gaussaufloes     : aufgerufen in erzmulttafel()        */
/*                                       zu finden in GaussAlgorit.c         */
/*                addg,subg,multg,divg : aufgerufen von reduzierpoly(),       */
/*                                        gausszerlegu()                     */
/*                                       zu finden in EndlArithmet.c         */
/*                                                                           */
/* Zur Implementierung:                                                      */
/*  Zunaechst wird das Tracecompatible Polynom eingelesen, und falls dies    */
/*   erfolgreich war die noetigen Speicherbelegungen durchgefuehrt.          */
/*                                                                           */
/*  Dann wird die Normalbasis in der Polynomialen Basis ermittelt und sowohl */
/*   in Tafel als auch in Gaussmatrix abgespeichert. Dafuer wird immer in    */
/*   Grosspolynom das letzte Basiselement**Erweiterungsgrad (= Frobeniushom.) */
/*   eingelesen und nach Tracepolynom reduziert.                             */
/*                                                                           */
/*  Mit Gausszerlegu() wird die Basistransformation (Gleichungssystem)       */
/*   Polynomialbasis -> Normalbasis ermoeglicht.                             */
/*                                                                           */
```

```
/*  Diese Transformation wird auf die gewuenschten x - Potenzen in Polynomial-*/
/*   basis angewendet, die man erhaelt, indem die in Tafel abgespeicherten    */
/*   Polynome mit x multipliziert, und dann reduziert.                        */
/*                                                                            */
/*                                        Autoer : Ulrich Eidt                */
/*                                        Stand  : 05.11.91                   */
/******************************************************************************/
INT erzmulttafel(Erweiterungsgrad)
INT Erweiterungsgrad;
    {
    INT *Gaussmatrix;  /* Matrix fuer Basistransformation                 */
    INT **Gau;         /* Zeigersystem fuer Gaussmatrix                   */
    INT *Grosspolynom; /* Platz fuer Polynome mit Grad > Erweiterungsgrad */
    INT Grosspolzeiger;/* Hilfszeiger fuer Grosspolynom                   */
    INT Gradgrosspol;  /* Gibt den Grad von Grosspolynom an               */
    INT *Tracepolynom; /* Tracecompatibles Polynom                        */
    INT *Permutation;  /* Permutationsvektor bei Gleichungssystemloesung  */
    INT i,j,k;           /* Laufvariable                                  */

    if (UE_Erw_Grad)
        if (kgv(UE_Erw_Grad,Erweiterungsgrad)<=100)
            Erweiterungsgrad = kgv(UE_Erw_Grad,Erweiterungsgrad);
/* Tracecompatibles Polynom wird eingelesen, falls nicht moeglich Zurueck */
/* falls moeglich Speicher freimachen                                     */
    Tracepolynom = (INT *) calloc(Erweiterungsgrad,sizeof(INT));
    if (!liestracepol(Erweiterungsgrad,Tracepolynom))
        {
        printf("Tracecompatibles Polynom nicht beschaffbar!\n");
        free((char *) Tracepolynom);
        return(0);
        }

/* Bestehende Tafeln freimachen */
    if (UE_Erw_Grad)
        {
        free((char *) Mult_Tafel);
        free((char *) UE_Platz_Mult);
        }
    UE_Platz_Mult = (INT *) malloc(Erweiterungsgrad*Erweiterungsgrad*
                                sizeof(INT));

/* Abspeichern der jeweiligen Zeilenanfaenge der Matrix, um Multiplikationen */
/*  bei der Addressierung zu vermeiden */
    Mult_Tafel = (INT **) malloc(Erweiterungsgrad*sizeof(INT*));
    k=0;
    for (i=0;i<Erweiterungsgrad;i++)
        {
        Mult_Tafel[i] = &UE_Platz_Mult[k];
        k += Erweiterungsgrad;
        }
        if (Erweiterungsgrad == 1)
            {
```

```
            free((char *) Tracepolynom);
            Mult_Tafel[0][0] = 1;
            UE_Erw_Grad = 1;
            return(1);
            }

    Grosspolynom = (INT *) calloc(Erweiterungsgrad*Charakteristik,sizeof(INT));
    Gaussmatrix = (INT *) malloc(Erweiterungsgrad*Erweiterungsgrad*sizeof(INT));
    Permutation = (INT *) malloc(Erweiterungsgrad*sizeof(INT));

/* Abspeichern der jeweiligen Zeilenanfaenge der Matrix, um Multiplikationen */
/*  bei der Addressierung zu vermeiden */
    Gau = (INT **) malloc(Erweiterungsgrad*sizeof(INT*));
    k=0;
    for (i=0;i<Erweiterungsgrad;i++)
        {
        Gau[i] = &Gaussmatrix[k];
        k += Erweiterungsgrad;
        }

/* Initialisierung der Gaussmatrix und Zwischenspeichern der Potenzen */
    for (i=0;i<Erweiterungsgrad;i++)
        {
        Gau[i][0] = 0;
        Mult_Tafel[0][i] = 0;
        }
    Gau[1][0] = 1;
    Mult_Tafel[0][1] = 1;
    for (i=1;i<Erweiterungsgrad;i++)
        {
        for (j=0;j<Erweiterungsgrad;j++)
            Grosspolynom[j] = 0;
        Grosspolzeiger = 0;
        for (j=0;j<Erweiterungsgrad;j++)
            {

        /* Grosspolynom = letztes Polynom ** Charakteristik(Frobenius)*/
            if (Gau[j][i-1])
                {
                Grosspolynom[Grosspolzeiger] = Gau[j][i-1];
                Gradgrosspol = Grosspolzeiger;
                }
            Grosspolzeiger += Charakteristik;
            }

    /* Grosspolynom wird nach Tracepolynom reduziert und abgespeichert */
        reduzierpoly(Grosspolynom,Gradgrosspol,Tracepolynom,Erweiterungsgrad);
        for (j=0;j<Erweiterungsgrad;j++)
            {
            Gau[j][i] = Grosspolynom[j];
            Mult_Tafel[i][j] = Grosspolynom[j];
            }
```

```
        }

/* Gaussdreieckszerlegung */
    if(!gausszerlegu(Gau,Erweiterungsgrad,Permutation))
        {
        printf("Matrix war singulaer\n");
        free((char *) Gaussmatrix);
        free((char *) Grosspolynom);
        free((char *) Tracepolynom);
        free((char *) Permutation);
        free((char *) Gau);
        return(0);
        }

/* Berechnung der Tafeleintraege von x*x**(p**i) in der Normalbasis */
    for (i=0;i<Erweiterungsgrad;i++)
        {

    /* Grosspolynom wird belegt mit x*x**(p**i) in Polynomialbasis */
        Grosspolynom[0] = 0;
        for (j=0;j<Erweiterungsgrad;j++)
            Grosspolynom[j+1] = Mult_Tafel[i][j];

    /* Grosspolynom wird reduziert und der Basiswechsel durchgefuehrt */
        reduzierpoly(Grospolynom,Erweiterungsgrad,Tracepolynom,Erweiterungsgrad);
        gaussaufloes(Gau,Erweiterungsgrad,Grosspolynom,Permutation);

    /* Abschliessend wird das Ergebnis gespeichert */
        for (j=0;j<Erweiterungsgrad;j++)
            Mult_Tafel[i][j] = Grosspolynom[j];
        }

/* Speicher wieder freigeben */
    free((char *) Gaussmatrix);
    free((char *) Grosspolynom);
    free((char *) Tracepolynom);
    free((char *) Permutation);
    free((char *) Gau);
    UE_Erw_Grad = Erweiterungsgrad;
    return(1);
    }



/*****************************************************************************/
/*                       FUNKTIONEN FUER ENDLICHE KOERPER                    */
/* Die folgenden Funktionen stellen eine Arithmetik fuer endliche Koerper in  */
/* Normalbasenrepraesentation dar. Ein Koerperelement wird wie folgt          */
/* abgespeichert:                                                            */
/*   Element[0] enthaelt den Erweiterungsgrad, die weiteren <Element[0]>      */
/*   Stellen die Eintraege des Elements bezueglich der entsprechenden         */
/*   Normalbasis.  Element[0] = 0 ist gleichbedeutend mit 'nicht definiert'.  */
```

```
/*                                                                          */
/*                      Verfasser : Ulrich Eidt                             */
/*                      Stand     : 04.11.91                                */
/****************************************************************************/

/****************************************************************************/
/* Funktion minimalErw stellt den Koerper mit dem kleinsten Erweiterungsgrad  */
/* fest, in dem ein Element enthalten ist. Ist dieser kleiner als Element[0]  */
/* so wird die Speicherung des Elementes auf den kleinsten moeglichen Erwei-  */
/* terungsgrad angepasst.                                                     */
/* Ein Element ist genau dann aus einem Unterkoerper, wenn fuer einen Teiler  */
/* m des Erweiterungsgrades sich die Eintraege immer nach m Eintraegen wieder-*/
/* holen.                                                                     */
/****************************************************************************/
minimalErw(Element)
INT **Element;
    {
    INT i,j,maximum=(*Element)[0]/2,Grad=(*Element)[0],minGrad=0;
        INT erfolgreich;
/* falls nicht definiert keine Aenderungen */
    if ((*Element)[0] == 0)
        return;
/* Teiler suchen nur bis zum maximalen Teiler  */
    while(minGrad<=maximum)
        {
        minGrad++;
/* Nur Teiler muessen ueberprueft werden */
        if (!(Grad % minGrad))
            {
            erfolgreich = 1;
            for (i=minGrad;i<Grad;i+=minGrad)
                for (j=1;j<=minGrad;j++)
                    if ((*Element)[j] != (*Element)[i+j])
                        {
/* Keine Wiederholung -> minGrad nicht der minimale Erweiterungsgrad */
                        erfolgreich = 0;
                        j = minGrad+1;
                        i = Grad;
                        }
            if (erfolgreich)
                {
                **Element = minGrad;
                return;
                }
            }
        }
    return;
    }



/****************************************************************************/
```

```
/* Funktion UE_scan uebernimmt das Einlesen der Koerperelemente               */
/*                                                                            */
/* Struktur : 0-te Stelle enthaelt den Grad der Koerpererweiterung            */
/*            Die darauffolgenden <Grad> Stellen enthalten das Element        */
/****************************************************************************/
UE_scan(Koerperzeiger)
INT **Koerperzeiger;
    {
    INT i,j=0,ZeichenZeiger = 1,*Koerperelement;
    char *Zeichen;
    Koerperelement = *Koerperzeiger;
    Zeichen = calloc(500,sizeof(char));
    printf("Eingabe eines Koerperelements\nGrad der Koerpererweiterung : ");
    scanf("%d",&i);
    if (i<=0)
        {
        Koerperelement[0] = UE_Erw_Grad;
        for (i=1;i<=UE_Erw_Grad;i++)
            Koerperelement[i] = 0;
        return;
        }
    if (i>UE_Erw_Grad)
        {
        if (UE_Erw_Grad > 0)
            free((char *) Koerperelement);
        Koerperelement = (INT *) malloc((i+1)*sizeof(INT));
        *Koerperzeiger = Koerperelement;
        }
    for (j=0;j<=i;j++)
        Koerperelement[j] = 0;
    printf("Eingabe der %d Stellen, bitte mit Kommas getrennt eingeben",i);
    printf("\nNicht eingegebene Stellen werden 0 gesetzt\n");
    scanf("%s",Zeichen);
    j = 1;
    ZeichenZeiger = 0;
    while (j<=i)
        {
/* 44 = 'Komma' */
        while (Zeichen[ZeichenZeiger]!=44 && Zeichen[ZeichenZeiger]!=0)
            {
            Koerperelement[j] = Koerperelement[j] * 10 +
                                        Zeichen[ZeichenZeiger] - 48;
            ZeichenZeiger++;
            }
        ZeichenZeiger++;
        j++;
        }
    for (j=1;j<=i;j++)
        Koerperelement[j] %= Charakteristik;
        Koerperelement[0] = i;
    minimalErw(Koerperzeiger);
    }
```

```
/*****************************************************************************/
/* Funktion UE_Platz tellt ein undefiniertes Koerperelement bereit.          */
/*****************************************************************************/
UE_Platz(Koerperzeiger)
INT **Koerperzeiger;
    {
    *Koerperzeiger = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
    (*Koerperzeiger)[0] = 0;
    return;
    }



/*****************************************************************************/
/* Funktion UE_Zeige gibt ein Koerperelement auf dem Bildschirm aus.         */
/*****************************************************************************/
UE_Zeige(Koerperzeiger)
INT **Koerperzeiger;
    {
    INT i,*Koerperelement;
    if ((*Koerperzeiger)[0] == 0)
        {
        printf("nicht definiert\n");
        return;
        }
    minimalErw(Koerperzeiger);
    Koerperelement = *Koerperzeiger;
    for (i=1;i<Koerperelement[0];i++)
        printf("%d,",Koerperelement[i]);
    printf("%d\n",Koerperelement[Koerperelement[0]]);
    return;
    }

/*****************************************************************************/
/* Funktion UE_fZeige gibt ein Koerperelement auf f aus.                     */
/*****************************************************************************/
UE_fZeige(f,Koerperzeiger)
INT **Koerperzeiger; char *f;
    {
    INT i,*Koerperelement;
    if ((*Koerperzeiger)[0] == 0)
        {
        fprintf(f,"nicht definiert\n");
        return;
        }
    minimalErw(Koerperzeiger);
    Koerperelement = *Koerperzeiger;
    for (i=1;i<Koerperelement[0];i++)
        fprintf(f,"%d,",Koerperelement[i]);
```

```
    fprintf(f,"%d\n",Koerperelement[Koerperelement[0]]);
    return;
    }



/*****************************************************************************/
/* Funktion UE_add belegt Ergebzeig mit Summ1zeig+Summ2zeig                  */
/*****************************************************************************/
UE_add(Summ1zeig,Summ2zeig,Ergebzeig)
INT **Summ1zeig;
INT **Summ2zeig;
INT **Ergebzeig;
    {
    INT i,j,k,*Summ1hilf,*Summ2hilf,*Summand1,*Summand2,*Ergebnis;
    Summand1 = *Summ1zeig;
    Summand2 = *Summ2zeig;
    Ergebnis = *Ergebzeig;


/* Falls der Grad einer der Summanden nicht Den Erweiterungsgrad teilt, */
/* Wird eine Koerpererweiterung noetig */
    if (!UE_Erw_Grad || (UE_Erw_Grad % Summand1[0]+UE_Erw_Grad%Summand2[0]))
        {
        minimalErw(Summ1zeig);
        minimalErw(Summ2zeig);
        }
    if (!UE_Erw_Grad || (UE_Erw_Grad % Summand1[0]+UE_Erw_Grad%Summand2[0]))
        if (!erzmulttafel(kgv(Summand1[0],Summand2[0])))
            {
            printf("Kann die gewuenschte Addition nicht ");
                        printf("durchfuehren\n");
            return;
            }

/* ist der Grad einer der Summanden m nicht gleich dem aktuellen Grad, so */
/* muss eine Einbettung vorgenommen werden, indem man die Koeffizienten   */
/* (<aktueller Grad> / m)-mal wiederholt.                                 */
    if (Summand1[0] != UE_Erw_Grad)
        {
        k=1;
        Summ1hilf = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        for (i=0;i<UE_Erw_Grad/Summand1[0];i++)
            for (j=1;j<=Summand1[0];j++)
                Summ1hilf[k++] = Summand1[j];
        }
    else
        Summ1hilf = Summand1;

    if (Summand2[0] != UE_Erw_Grad)
        {
```

```
        k=1;
        Summ2hilf = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        for (i=0;i<UE_Erw_Grad/Summand2[0];i++)
            for (j=1;j<=Summand2[0];j++)
                Summ2hilf[k++] = Summand2[j];
        }
    else
        Summ2hilf = Summand2;

/* Der Grad des Ergebniselementes muss korrekt sein */
    if (Ergebnis[0]!=UE_Erw_Grad)
        {
        if (Ergebnis[0] != 0)
            free((char *) *Ergebnis);
        Ergebnis = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        Ergebnis[0] = UE_Erw_Grad;
        *Ergebzeig = Ergebnis;
        }

/* Addition */
    for (i=1;i<=UE_Erw_Grad;i++)
        Ergebnis[i] = addg(Summ1hilf[i],Summ2hilf[i]);
    return;
    }




/*****************************************************************************/
/* Funktion UE_mult belegt Ergebzeig mit Fakt1zeig+Fakt2zeig                 */
/*****************************************************************************/
UE_mult(Fakt1zeig,Fakt2zeig,Ergebzeig)
INT **Fakt1zeig;
INT **Fakt2zeig;
INT **Ergebzeig;
    {
    INT i,j,k,l,m,*Fakt1hilf,*Fakt2hilf,*Faktor1,*Faktor2,*Ergebnis,zwisch;
    Faktor1 = (INT *) malloc(((*Fakt1zeig)[0]+1)*sizeof(INT));
    Faktor1[0] = (*Fakt1zeig)[0];
    for (i=1;i<=Faktor1[0];i++)
        Faktor1[i] = (*Fakt1zeig)[i];
    Faktor2 = (INT *) malloc(((*Fakt2zeig)[0]+1)*sizeof(INT));
    Faktor2[0] = (*Fakt2zeig)[0];
    for (i=1;i<=Faktor2[0];i++)
        Faktor2[i] = (*Fakt2zeig)[i];
    Ergebnis = *Ergebzeig;


/* Falls der Grad einer der Faktoren nicht Den Erweiterungsgrad teilt, */
/* Wird eine Koerpererweiterung noetig */
    if (!UE_Erw_Grad || (UE_Erw_Grad%Faktor1[0] + UE_Erw_Grad%Faktor2[0]))
        {
```

```
        minimalErw(Fakt1zeig);
        minimalErw(Fakt2zeig);
        }
    if (!UE_Erw_Grad || (UE_Erw_Grad%Faktor1[0] + UE_Erw_Grad%Faktor2[0]))
        if (!erzmulttafel(kgv(Faktor1[0],Faktor2[0])))
            {
            printf("Kann die gewuenschte Multiplikation ");
                        printf("nicht durchfuehren\n");
            return;
            }

/* ist der Grad einer der Faktoren m nicht gleich dem aktuellen Grad, so  */
/* muss eine Einbettung vorgenommen werden, indem man die Koeffizienten   */
/* (<aktueller Grad> / m)-mal wiederholt.                                 */
    if (Faktor1[0] != UE_Erw_Grad)
        {
        k=1;
        Fakt1hilf = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        for (i=0;i<UE_Erw_Grad/Faktor1[0];i++)
            for (j=1;j<=Faktor1[0];j++)
                Fakt1hilf[k++] = Faktor1[j];
        }
    else
        Fakt1hilf = Faktor1;

    if (Faktor2[0] != UE_Erw_Grad)
        {
        k=1;
        Fakt2hilf = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        for (i=0;i<UE_Erw_Grad/Faktor2[0];i++)
            for (j=1;j<=Faktor2[0];j++)
                Fakt2hilf[k++] = Faktor2[j];
        }
    else
        Fakt2hilf = Faktor2;

/* der Grad des Ergebnisselementes muss korrekt sein */
    if (Ergebnis[0]!=UE_Erw_Grad)
        {
        if (Ergebnis[0] != 0)
            free((char *) Ergebnis);
        Ergebnis = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        Ergebnis[0] = UE_Erw_Grad;
        *Ergebzeig = Ergebnis;
        }

/* Ergebniselement mit 0 vobelegen */
    for (i=1;i<=UE_Erw_Grad;i++)
        Ergebnis[i] = 0;
/* Multiplikation mittels Multiplikationstafel */
    for (i=1;i<=UE_Erw_Grad;i++)
        {
```

```
    if(Fakt1hilf[i])
        {
        l=0;
        for (j=UE_Erw_Grad-i+1;j<=UE_Erw_Grad-1;j++)
            {
            l++;
            if (Fakt2hilf[l])
               {
               zwisch=multg(Fakt1hilf[i],Fakt2hilf[l]);
               m=0;
               for (k=UE_Erw_Grad-i+1;k<=UE_Erw_Grad-1;k++)
                {
                m++;
                if (Mult_Tafel[j][k])
                Ergebnis[m]=addg(Ergebnis[m],
                multg(zwisch,Mult_Tafel[j][k]));
                }
               for (k=0;k<=UE_Erw_Grad-i;k++)
                {
                m++;
                if (Mult_Tafel[j][k])
                Ergebnis[m] = addg(Ergebnis[m],
                multg(zwisch,Mult_Tafel[j][k]));
                }
               }
            }
        for (j=0;j<=UE_Erw_Grad-i;j++)
            {
            l++;
            if (Fakt2hilf[l])
               {
               zwisch = multg(Fakt1hilf[i],Fakt2hilf[l]);
               m=0;
               for (k=UE_Erw_Grad-i+1;k<=UE_Erw_Grad-1;k++)
                {
                m++;
                if (Mult_Tafel[j][k])
                Ergebnis[m] = addg(Ergebnis[m],
                multg(zwisch,Mult_Tafel[j][k]));
                }
               for (k=0;k<=UE_Erw_Grad-i;k++)
                {
                m++;
                if (Mult_Tafel[j][k])
                Ergebnis[m] = addg(Ergebnis[m],
                multg(zwisch,Mult_Tafel[j][k]));
                }
               }
            }
        }
    }
free ((char *) Faktor1);
```

```
    free ((char *) Faktor2);
    if (Faktor1!=Fakt1hilf)
        free ((char *) Fakt1hilf);
    if (Faktor2!=Fakt2hilf)
        free ((char *) Fakt2hilf);
    return;
    }



/*****************************************************************************/
/* Funktion negativ belegt Ergebnis mit -Element                             */
/*****************************************************************************/
negativ(Element,Ergebnis)
INT **Element,**Ergebnis;
    {
    INT *Elemhelp,i;
    minimalErw(Element);
/* 0 Element */
    if ((*Element)[0] == 1 && (*Element)[1] == 0)
        {
        (*Ergebnis)[0] = 1;
        (*Ergebnis)[1] = 0;
        return;
        }
/* Ergebniselement muss korrekte Erweiterung haben */
    if ((*Ergebnis)[0] < (*Element)[0])
        {
        free((char*) (*Ergebnis));
        *Ergebnis = (INT *) malloc(((*Element)[0] +1)*sizeof(INT));
        }

    (*Ergebnis)[0] = (*Element)[0];
    for (i=1;i<=(*Element)[0];i++)
        if ((*Element)[i])
/* Negativbildung modulo Charakteristik */
            (*Ergebnis)[i] = Charakteristik-(*Element)[i];
        else
            (*Ergebnis)[i] = 0;
    return;
    }


/*****************************************************************************/
/* Funktion UE_hoch berechnet Element**m und gibt das Ergebnis in Ergebzeigaus*/
/*****************************************************************************/
UE_hoch(Element,m,Ergebnis)
INT **Element;
INT m;
INT **Ergebnis;
    {
    INT *Elemhelp,i;
```

```
    minimalErw(Element);
    if ((*Element)[0] == 1 && (*Element)[1] == 0)
        {
        (*Ergebnis)[0] = 1;
        (*Ergebnis)[1] = 0;
        return;
        }

/* falls Einselement */
    if ((*Element)[0] == 1 && (*Element)[1] == 1)
        {
        (*Ergebnis)[0] = 1;
        (*Ergebnis)[1] = 1;
        return;
        }

/* Erzeugung eines Hilfselementes  */
    Elemhelp = (int *) malloc(((*Element)[0] +1)*sizeof(int));
    for (i=0;i<=(*Element)[0];i++)
        Elemhelp[i] = (*Element)[i];

/* falls Ergebniselement nicht den benoetigten Erweiterungsgrad hat */
    if ((*Ergebnis)[0] < Elemhelp[0])
        {
        free((char*) (*Ergebnis));
        *Ergebnis = (int *) malloc(((*Element)[0] +1)*sizeof(int));
        }

    for (i=0;i<=(*Element)[0];i++)
        (*Ergebnis)[i] = (*Element)[i];
    i = m-1;
    while(i>0)
        {
        while(!(i % 2))
            {
            i /= 2;
            UE_mult(&Elemhelp,&Elemhelp,&Elemhelp);
            }
        i--;
        UE_mult(Ergebnis,&Elemhelp,Ergebnis);
        }
    free((char *) Elemhelp);
    return;
    }



/*****************************************************************************/
/* Funktion UE_invers berechnet 1/Element und gibt dies in Ergebzeig aus.    */
/*****************************************************************************/
UE_invers(Element,Ergebnis)
INT **Element,**Ergebnis;
```

```
    {
    INT i;
    if ((*Element)[0] == 1 && (*Element)[1] == 0)
        {
        printf("Division durch 0\n");
        (*Ergebnis)[0] = 0;
        return;
        }

/* 1/Element = Element**(Charakteristik**(Erweiterungsgrad des Elem.)-2) */
    i = power(Charakteristik,(*Element)[0])-2;
    UE_hoch(Element,i,Ergebnis);
    return;
    }



/*****************************************************************************/
/* Funktion ist_null testet, ob Element = 0                                  */
/* Rueckgabewert : 1 falls Element = 0, 0 sonst                              */
/*****************************************************************************/
INT ist_null(Element)
INT **Element;
    {
    minimalErw(Element);
    if((*Element)[0] == 1 && (*Element)[1] == 0)
        return(1);
    return(0);
    }



/*****************************************************************************/
/* Funktion ist_eins testet, ob Element = 1                                  */
/* Rueckgabewert : 1 falls Element = 1, 0 sonst                              */
/*****************************************************************************/
INT ist_eins(Element)
INT **Element;
    {
    minimalErw(Element);
    if((*Element)[0] == 1 && (*Element)[1] == 1)
        return(1);
    return(0);
    }



/*****************************************************************************/
/* Funktion Int_Aequivalent berechnet zu einer Integerzahl a das zugehoerige */
/* Grundkoerperelement und belegt Ergebnis damit.                            */
/*****************************************************************************/
Int_Aequivalent(a,Ergebnis)
```

```
INT a;
INT **Ergebnis;
    {
/* falls Ergebniselement nicht den benoetigten Erweiterungsgrad hat */
    if ((*Ergebnis)[0] < 1)
        {
        free((char*) (*Ergebnis));
        *Ergebnis = (INT *) malloc(2*sizeof(INT));
        }
    (*Ergebnis)[0] = 1;
    (*Ergebnis)[1] = a % Charakteristik;
    return;
    }



/*****************************************************************************/
/* Funktion ist_gleich vergleicht Element1 mit Element2                      */
/* Rueckgabewert :  1  falls Element1 > Element2,                            */
/*                 -1  falls Element1 < Element2,                            */
/*                  0  falls Element1 = Element2.                            */
/*****************************************************************************/
INT ist_gleich(Element1,Element2)
INT **Element1;
INT **Element2;
    {
    INT i,j,k,*Elem1hilf,*Elem2hilf;
    INT gemein_grad;

/* falls die Elemente verschiedene Erweiterungsgrade ueber dem Grundkoerper */
/* haben, muessen sie in den gemeinsammen Erweiterungskoerper eingebettet   */
/* werden.                                                                   */
    if ((*Element1)[0] != (*Element2)[0])
        {
        gemein_grad = kgv((*Element1)[0],(*Element2)[0]);
        if ((*Element1)[0] != gemein_grad)
            {
            k=1;
            Elem1hilf = (int *) malloc((gemein_grad+1)*sizeof(int));
            for (i=0;i<gemein_grad/(*Element1)[0];i++)
                for (j=1;j<=(*Element1)[0];j++)
                    Elem1hilf[k++] = (*Element1)[j];
            }
        else
            Elem1hilf = *Element1;

        if ((*Element2)[0] != gemein_grad)
            {
            k=1;
            Elem2hilf = (int *) malloc((gemein_grad+1)*sizeof(int));
            for (i=0;i<gemein_grad/(*Element2)[0];i++)
                for (j=1;j<=(*Element2)[0];j++)
```

```
                    Elem2hilf[k++] = (*Element2)[j];
            }
        else
            Elem2hilf = *Element2;
        }
    else
        {
        Elem1hilf = *Element1;
        Elem2hilf = *Element2;
        }

/* Vergleich */
    k = 0;
    for (i=1;i<=gemein_grad;i++)
        if (Elem1hilf[i] != Elem2hilf[i])
            {
            k = i;
            i = gemein_grad+1;
            }

    if (k > 0 && Elem1hilf[k] < Elem2hilf[k])
        k = -1;

    if (k > 0 && Elem1hilf[k] > Elem2hilf[k])
        k = 1;

/* Speicher, fallsbenoetigt freigeben */
    if (*Element1!=Elem1hilf)
        free ((char *) Elem1hilf);
    if (*Element2!=Elem2hilf)
        free ((char *) Elem2hilf);

    return(k);
    }



/*****************************************************************************/
/* Funktion Order berechnet das kleinste m mit Element**m = 1                */
/* Rueckgabewert : die berechnete Ordnung.                                   */
/*****************************************************************************/
int Order(Element)
int **Element;
    {
    INT maxord = power(Charakteristik,(*Element)[0])-1;
    INT maxteiler = maxord/2;
    INT i,j,*Ergebnis;
    for (i=1;i<=maxteiler;i++)
        if (!(maxord % i))
            {
            UE_Platz(&Ergebnis);
            UE_hoch(Element,i,&Ergebnis);
```

```
            if (ist_eins(&Ergebnis))
                return(i);
            }
    return(maxord);
    }



/*****************************************************************************/
/* Funktion UE_Random erzeugt ein zufaelliges Element im Koerper des aktuellen*/
/* Grades.                                                                   */
/*****************************************************************************/
UE_Random(Element)
INT **Element;
    {
    INT i;
/* Der Erweiterungsgrad des Elementes muss mit dem aktuellen uebereinstimmen */
    if ((*Element)[0] != UE_Erw_Grad)
        {
        *Element = (INT *) malloc((UE_Erw_Grad+1)*sizeof(INT));
        (*Element)[0] = UE_Erw_Grad;
        }
/* Zufallserzeugung */
    for (i=1;i<=UE_Erw_Grad;i++)
        (*Element)[i] = rand() % Charakteristik;
    minimalErw(Element);
    return;
    }
```

Anhang A

Tabellen mit James-Gewichtsräumen $D^{\lambda^\#,\lambda,\alpha}$

Wie im 2.Kapitel gesehen, kann man James-Gewichtsräume als $[n,k,d]$-Codes auffassen. Die Tabellen enthalten Überblicke zum Verhalten von n und k in Abhängigkeit von $\lambda^\#, \lambda$ und α. Für die Minimaldistanz d wird jeweils nur eine obere Schranke angegeben.
Die Tupel in den Tabellen geben n und k an. Spalten und Zeilen mit $k \leq 1$ werden weggelassen.

A.1 λ und α Partitionen von 5

$p_1 = (3,2)$
$p_2 = (3,1,1)$
$p_3 = (2,2,1)$
$p_4 = (2,1,1,1)$
$p_5 = (1,1,1,1,1)$

$\lambda^\# = (2,2)$, $d \leq 4$

α

λ		p_3	p_4	p_5
	p_1	$5,2$	$7,3$	$10,5$
	p_3	$11,3$	$18,5$	$30,10$

$\lambda^\# = (2,2,1)$, $d \leq 12$

α

λ		p_4	p_5
	p_3	$18,2$	$30,5$

A.2 λ und α Partitionen von 6

$p_1 = (4,2)$
$p_2 = (4,1,1)$
$p_3 = (3,3)$
$p_4 = (3,2,1)$
$p_5 = (3,1,1,1)$
$p_6 = (2,2,2)$
$p_7 = (2,2,1,1)$
$p_8 = (2,1,1,1,1)$
$p_9 = (1,1,1,1,1,1)$

$\lambda^\# = (2,2)$, $d \leq 4$

α

λ	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	$3,1$	$5,2$	$7,3$	$6,3$	$8,4$	$11,6$	$15,9$
p_3	$4,2$	$6,3$	$8,4$	$7,4$	$10,6$	$14,9$	$20,14$
p_4	$6,2$	$12,4$	$19,6$	$15,6$	$24,10$	$38,17$	$60,30$
p_6	$7,1$	$15,3$	$24,5$	$21,6$	$33,9$	$54,16$	$90,30$
p_7	$10,2$	$24,5$	$42,8$	$33,9$	$58,16$	$102,30$	$180,60$

$\lambda^\# = (3,3)$, $d \leq 8$

α

λ	p_7	p_8	p_9
p_3	$10,2$	$14,3$	$20,5$

$\lambda^\# = (2,2,1)$, $d \leq 12$

α

λ	p_5	p_6	p_7	p_8	p_9
p_4	$19,2$	$15,2$	$24,4$	$38,8$	$60,16$
p_6	$24,2$	$21,3$	$33,5$	$54,10$	$90,21$
p_7	$42,2$	$33,3$	$58,6$	$102,13$	$180,30$

A.3 λ und α Partitionen von 7

$p_1 = (5,2)$
$p_2 = (5,1,1)$
$p_3 = (4,3)$
$p_4 = (4,2,1)$
$p_5 = (4,1,1,1)$
$p_6 = (3,3,1)$
$p_7 = (3,2,2)$
$p_8 = (3,2,1,1)$
$p_9 = (3,1,1,1,1)$
$p_{10} = (2,2,2,1)$
$p_{11} = (2,2,1,1,1)$
$p_{12} = (2,1,1,1,1,1)$
$p_{13} = (1,1,1,1,1,1,1)$

$\lambda^\# = (2,2)$, $d \leq 4$

α

λ	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}
p_1	3,1	5,2	7,3	5,2	6,3	8,4	11,6	9,5	12,7	16,10	21,14
p_3	4,2	6,3	8,4	7,4	8,5	11,7	15,10	13,9	18,3	25,19	35,28
p_4	6,2	12,4	19,6	13,5	16,7	25,11	39,18	30,15	46,24	70,39	105,63
p_6	7,3	13,5	20,7	16,8	19,10	30,16	46,25	27,22	58,36	90,59	140,98
p_7	8,2	16,4	25,6	19,6	25,9	39,14	62,23	51,21	81,35	130,60	210,105
p_8	11,3	25,6	43,9	30,10	39,14	67,24	114,41	87,36	148,64	250,115	420,210
p_{10}	13,2	30,5	51,8	37,7	51,12	87,20	150,36	120,33	207,59	360,110	639,210
p_{11}	18,3	46,7	84,11	58,12	81,19	148,34	270,62	207,57	378,108	690,210	1260,420

$\lambda^\# = (3,3)$, $d \leq 8$

α

λ	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}
p_3	7,2	8,2	11,3	15,4	13,4	18,6	25,9	35,14
p_6	16,3	19,3	30,5	46,7	37,7	58,12	90,20	140,35

$\lambda^\# = (2,2,1)$, $d \leq 12$

α

λ	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}
p_4	19,2	13,1	16,2	25,4	39,8	30,6	46,11	70,20	105,35
p_6	20,2	16,2	19,3	30,6	46,11	37,9	58,17	90,31	140,56
p_7	25,2	19,2	25,4	39,7	62,13	51,12	81,22	130,41	210,77
p_8	43,2	30,2	39,4	67,8	114,16	87,14	148,28	250,56	420,112

A.4 λ und α Partitionen von 8

$p_1 = (6,2)$
$p_2 = (6,1,1)$
$p_3 = (5,3)$
$p_4 = (5,2,1)$
$p_5 = (5,1,1,1)$
$p_6 = (4,4)$
$p_7 = (4,3,1)$
$p_8 = (4,2,2)$
$p_9 = (4,2,1,1)$
$p_{10} = (4,1,1,1,1)$
$p_{11} = (3,3,2)$
$p_{12} = (3,3,1,1)$
$p_{13} = (3,2,2,1)$
$p_{14} = (3,2,1,1,1)$
$p_{15} = (3,1,1,1,1,1)$
$p_{16} = (2,2,2,2)$
$p_{17} = (2,2,2,1,1)$
$p_{18} = (2,2,1,1,1,1)$
$p_{19} = (2,1,1,1,1,1,1)$
$p_{20} = (1,1,1,1,1,1,1,1)$

$\lambda^\# = (2,2)$, $d \leq 4$

α

λ	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
p_1	3,1	5,2	7,3	3,1	5,2	6,3	8,4	11,6	6,3	8,4
p_3	4,2	6,3	8,4	4,2	7,4	8,5	11,7	15,10	9,6	12,8
p_4	6,2	12,4	19,6	6,2	13,5	16,7	25,11	39,18	17,8	26,12
p_6	4,2	6,3	8,4	5,3	8,5	9,6	12,8	16,11	10,7	14,10
p_7	7,3	13,5	20,7	8,4	17,9	20,11	31,17	47,26	23,14	36,22
p_8	8,2	16,4	25,6	9,3	20,7	26,10	40,15	63,24	29,12	45,19
p_9	11,3	25,6	43,9	12,4	31,11	40,15	68,25	115,42	23,14	76,32
p_{10}	9,3	17,5	26,7	10,4	23,10	29,13	45,20	70,31	35,18	54,28
p_{11}	12,4	26,7	44,10	14,6	36,16	45,20	76,33	126,53	45,19	92,48
p_{13}	14,3	31,6	52,9	16,4	43,12	57,17	96,28	162,47	69,24	116,40
p_{14}	19,4	47,8	85,12	22,6	66,19	89,26	160,45	286,77	108,38	194,68
p_{16}	16,2	36,5	60,8	19,3	52,9	72,15	120,24	204,42	88,19	148,32
p_{17}	22,3	54,7	96,11	26,4	80,14	111,22	198,38	354,68	139,31	248,54
p_{18}	30,4	80,9	150,14	36,6	122,22	173,33	324,59	606,106	220,50	412,92

α

λ	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}
p_1	9,5	12,7	16,10	10,6	13,8	17,11	22,15	28,20
p_3	14,10	19,14	26,20	16,12	22,17	30,24	41,34	56,48
p_4	31,16	47,25	71,40	36,20	54,31	80,48	117,74	168,112
p_6	16,12	22,17	30,24	19,15	26,21	36,30	50,43	70,62
p_7	43,28	66,44	100,69	52,36	80,57	122,90	185,142	280,224
p_8	57,26	89,42	140,69	72,36	111,57	173,93	270,153	420,252
p_9	96,44	160,75	265,129	120,60	198,102	324,174	525,297	840,504
p_{10}	69,38	108,61	170,99	88,52	139,85	220,140	350,233	560,392
p_{11}	116,64	194,109	320,184	148,88	248,152	412,262	680,452	1120,784
p_{13}	154,58	260,100	440,175	204,84	345,147	584,260	990,465	1680,840
p_{14}	260,98	463,177	820,320	348,144	618,264	1092,486	1920,900	3360,1680
p_{16}	204,50	348,88	600,160	282,78	480,135	828,244	1440,450	2520,840
p_{17}	345,85	618,155	1110,290	480,132	861,243	1548,456	2790,870	5040,1680
p_{18}	584,144	1092,272	2040,520	828,228	1548,438	2892,852	5400,1680	10080,3360

$\lambda^\# = (3,3)$, $d \leq 8$

α

λ	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
p_3	4, 1	7, 2	8, 2	11, 3	15, 4	9, 3	12, 4	14, 5	19, 7
p_6	5, 2	8, 3	9, 3	12, 4	16, 5	10, 4	14, 6	16, 7	22, 10
p_7	8, 2	17, 4	20, 4	31, 6	47, 8	23, 6	36, 10	43, 12	66, 19
p_{11}	10, 1	23, 3	29, 3	45, 5	70, 7	35, 6	54, 9	69, 12	108, 19
p_{12}	14, 2	36, 5	45, 5	76, 8	126, 11	54, 9	92, 16	116, 20	194, 34

α

λ	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}
p_3	26, 10	16, 6	22, 9	30, 13	41, 19	56, 28
p_6	30, 14	19, 9	26, 13	36, 19	50, 28	70, 42
p_7	100, 29	52, 16	80, 26	122, 42	185, 68	280, 112
p_{11}	170, 30	88, 16	139, 28	220, 47	350, 80	560, 140
p_{12}	320, 55	148, 28	248, 50	412, 88	680, 155	1120, 280

$\lambda^\# = (2,2,1)$, $d \leq 12$

α

λ	p_5	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
p_4	19, 2	13, 1	16, 2	25, 4	39, 8	17, 2	26, 4	31, 6	47, 11
p_7	20, 2	17, 2	20, 3	31, 6	47, 11	23, 4	36, 8	43, 11	66, 20
p_8	25, 2	20, 2	26, 4	40, 7	63, 13	29, 5	45, 9	57, 14	89, 25
p_9	43, 2	31, 2	40, 4	68, 8	115, 16	45, 5	76, 10	96, 16	160, 31
p_{11}	26, 2	23, 3	29, 5	45, 9	70, 16	35, 8	54, 14	69, 21	108, 37
p_{12}	44, 2	36, 3	45, 5	76, 10	126, 19	54, 8	92, 16	116, 24	194, 46
p_{13}	52, 2	43, 3	57, 6	96, 11	162, 21	69, 10	116, 18	154, 30	260, 56
p_{14}	85, 2	66, 3	89, 6	160, 12	286, 24	108, 10	194, 20	260, 34	463, 68

α

λ	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}
p_4	71, 20	36, 8	54, 14	80, 24	117, 40	168, 64
p_7	100, 35	52, 15	80, 27	122, 47	185, 80	280, 134
p_8	140, 45	72, 21	111, 36	173, 63	270, 110	420, 190
p_9	265, 60	120, 24	198, 45	324, 84	525, 155	840, 280
p_{11}	170, 65	88, 31	139, 55	220, 97	350, 171	560, 302
p_{12}	320, 86	148, 36	248, 69	412, 130	680, 242	1120, 448
p_{13}	440, 106	204, 48	345, 90	584, 170	990, 323	1680, 616
p_{14}	820, 136	348, 56	618, 112	1092, 224	1920, 448	3360, 896

A.5 λ und α Partitionen von 9

$p_1 = (7,2)$
$p_2 = (7,1,1)$
$p_3 = (6,3)$
$p_4 = (6,2,1)$
$p_5 = (6,1,1,1)$
$p_6 = (5,4)$
$p_7 = (5,3,1)$
$p_8 = (5,2,2)$
$p_9 = (5,2,1,1)$
$p_{10} = (5,1,1,1,1)$
$p_{11} = (4,4,1)$
$p_{12} = (4,3,2)$
$p_{13} = (4,3,1,1)$
$p_{14} = (4,2,2,1)$
$p_{15} = (4,2,1,1,1)$
$p_{16} = (4,1,1,1,1,1)$
$p_{17} = (3,3,3)$
$p_{18} = (3,3,2,1)$
$p_{19} = (3,3,1,1,1)$
$p_{20} = (3,2,2,2)$
$p_{21} = (3,2,2,1,1)$
$p_{22} = (3,2,1,1,1,1)$
$p_{23} = (3,1,1,1,1,1,1)$
$p_{24} = (2,2,2,2,1)$
$p_{25} = (2,2,2,1,1,1)$
$p_{26} = (2,2,1,1,1,1,1)$
$p_{27} = (2,1,1,1,1,1,1,1)$
$p_{28} = (1,1,1,1,1,1,1,1,1)$

$\lambda^\# = (2,2)$, $d \leq 4$

α

λ	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
p_1	3,1	5,2	7,3	3,1	5,2	6,3	8,4	11,6	5,2	6,3
p_3	4,2	6,3	8,4	4,2	7,4	8,5	11,7	15,10	7,4	9,6
p_4	6,2	12,4	19,6	6,2	13,5	16,7	25,11	39,18	13,5	17,8
p_6	4,2	6,3	8,4	5,3	8,5	9,6	12,8	16,11	9,6	11,8
p_7	7,3	13,5	20,7	8,4	17,9	20,11	31,17	47,26	18,10	24,15
p_8	8,2	16,4	25,6	9,3	20,7	26,10	40,15	63,24	21,8	30,13
p_9	11,3	25,6	43,9	12,4	31,11	40,15	68,25	115,42	32,12	46,20
p_{11}	7,3	13,5	20,7	9,5	18,10	21,12	32,18	48,27	21,13	27,18
p_{12}	9,3	17,5	26,7	11,5	24,11	30,14	46,21	71,32	27,14	39,22
p_{13}	12,4	26,7	44,10	15,7	37,17	46,21	77,34	127,54	42,22	60,34
p_{14}	14,3	31,6	52,9	17,5	44,13	58,18	97,29	163,48	49,17	75,29
p_{15}	19,4	47,8	85,12	23,7	67,20	90,27	161,46	287,78	74,26	116,45
p_{17}	10,3	18,5	27,7	12,4	27,10	33,13	51,20	78,31	30,12	45,21
p_{18}	15,4	32,7	53,10	19,7	49,18	63,23	105,37	174,59	56,24	87,41
p_{19}	20,5	48,9	86,13	26,10	74,27	97,34	172,57	302,93	86,38	135,64
p_{20}	17,3	37,6	61,9	22,5	58,14	78,20	129,32	216,53	67,18	109,34
p_{21}	23,4	55,8	97,12	30,7	88,21	119,29	210,49	370,83	102,28	169,53
p_{22}	31,5	81,10	151,15	41,10	132,31	183,42	339,73	626,125	154,44	263,83
p_{24}	26,3	62,7	108,11	35,5	104,16	144,25	252,42	444,74	123,21	212,43
p_{25}	35,4	90,9	165,14	48,7	156,24	219,36	402,63	738,112	186,32	330,67
p_{26}	47,5	129,11	246,17	66,10	232,35	333,51	634,91	1206,162	280,50	515,105

α

λ	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}
p_1	8, 4	9, 5	12, 7	16, 10	6, 3	9, 5	12, 7	10, 6	13, 8
p_3	12, 8	14, 10	19, 14	26, 20	10, 7	15, 11	20, 15	17, 13	23, 18
p_4	26, 12	31, 16	47, 25	71, 40	18, 9	32, 17	48, 26	37, 21	55, 32
p_6	15, 11	17, 13	23, 18	31, 25	12, 9	19, 15	26, 21	22, 18	30, 25
p_7	37, 23	44, 29	67, 45	101, 70	27, 18	49, 34	74, 52	58, 42	88, 65
p_8	46, 20	58, 27	90, 43	141, 70	33, 15	63, 31	97, 49	78, 41	119, 64
p_9	77, 33	97, 45	161, 76	266, 130	51, 24	105, 52	172, 86	129, 68	210, 113
p_{11}	42, 28	49, 34	74, 52	110, 79	30, 21	56, 41	86, 64	67, 51	102, 79
p_{12}	60, 34	75, 44	116, 69	180, 109	45, 27	87, 55	135, 87	109, 72	169, 114
p_{13}	101, 57	125, 73	206, 121	335, 199	69, 42	145, 92	240, 154	181, 120	298, 201
p_{14}	125, 48	163, 66	272, 111	455, 189	87, 36	188, 83	313, 141	243, 114	403, 193
p_{15}	206, 79	272, 109	479, 192	840, 339	135, 57	313, 139	548, 246	408, 192	710, 340
p_{17}	69, 32	87, 43	135, 68	210, 109	55, 28	105, 56	162, 88	132, 74	207, 119
p_{18}	145, 68	188, 91	313, 152	520, 254	105, 54	227, 122	378, 206	295, 166	493, 283
p_{19}	240, 113	313, 150	548, 261	950, 449	162, 84	378, 204	664, 362	496, 280	868, 498
p_{20}	181, 56	243, 80	408, 136	690, 235	132, 45	295, 107	496, 183	396, 153	666, 263
p_{21}	298, 92	403, 131	710, 231	1250, 410	207, 72	493, 180	868, 320	666, 258	1173, 463
p_{22}	486, 151	669, 214	1229, 389	2250, 705	324, 114	822, 302	1508, 558	1128, 438	2064, 814
p_{24}	372, 74	516, 113	912, 202	1620, 370	264, 57	644, 151	1140, 272	894, 228	1584, 411
p_{25}	606, 120	852, 183	1566, 337	2880, 630	417, 93	1077, 255	1980, 472	1515, 387	2787, 723
p_{26}	980, 195	1405, 295	2670, 555	5070, 1050	660, 150	1800, 430	3420, 820	2580, 660	4900, 1270

α

λ	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}
p_1	17, 11	22, 15	14, 9	18, 12	23, 16	29, 21	36, 27
p_3	31, 25	42, 35	26, 21	35, 29	47, 40	63, 55	84, 75
p_4	81, 49	118, 75	62, 38	90, 57	129, 85	182, 125	252, 180
p_6	41, 35	56, 49	35, 30	48, 42	66, 59	91, 83	126, 117
p_7	132, 100	197, 154	104, 80	156, 123	232, 188	343, 286	504, 432
p_8	183, 102	282, 164	144, 82	219, 129	333, 204	504, 322	756, 504
p_9	339, 188	543, 314	252, 144	402, 237	634, 388	987, 630	1512, 1008
p_{11}	154, 122	230, 187	123, 99	186, 153	280, 236	420, 363	630, 558
p_{12}	263, 182	410, 292	212, 150	330, 240	515, 386	805, 623	1260, 1008
p_{13}	486, 335	785, 556	372, 264	606, 441	980, 734	1575, 1218	2520, 2016
p_{14}	669, 330	1110, 567	516, 264	852, 450	1405, 771	2310, 1323	3780, 2268
p_{15}	1229, 603	2115, 1071	912, 468	1566, 828	2670, 1464	4515, 2583	7560, 4536
p_{17}	324, 192	510, 313	264, 160	417, 261	660, 428	1050, 707	1680, 1176
p_{18}	822, 483	1370, 827	644, 392	1077, 675	1800, 1166	3010, 2023	5040, 3528
p_{19}	1508, 882	2600, 1556	1140, 696	1980, 1242	3420, 2214	5880, 3948	10080, 7056
p_{20}	1128, 459	1920, 810	894, 378	1515, 663	2580, 1175	4410, 2100	7560, 3780
p_{21}	2064, 835	3630, 1515	1584, 672	2787, 1221	4900, 2230	8610, 4095	15120, 7560
p_{22}	3764, 1514	6840, 2820	2820, 1200	5130, 2250	9300, 4230	16800, 7980	30240, 15120
p_{24}	2820, 756	5040, 1410	2202, 618	3924, 1137	7020, 2120	12600, 3990	22680, 7560
p_{25}	5130, 1366	9450, 2610	3924, 1104	7227, 2097	13320, 4020	24570, 7770	45360, 15120
p_{26}	9300, 2460	17640, 4800	7020, 1980	13320, 3870	25260, 7620	47880, 15120	90720, 30240

$\lambda^\# = (3,3)$, $d \leq 8$

α

λ	p_5	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
p_4	19, 2	13, 1	16, 2	25, 4	39, 8	13, 1	17, 2	26, 4	31, 6
p_7	20, 2	17, 2	20, 3	31, 6	47, 11	18, 2	24, 4	37, 8	44, 11
p_8	25, 2	20, 2	26, 4	40, 7	63, 13	21, 2	30, 5	46, 9	58, 14
p_9	43, 2	31, 2	40, 4	68, 8	115, 16	32, 2	46, 5	77, 10	97, 16
p_{11}	20, 2	18, 2	21, 3	32, 6	48, 11	21, 3	27, 5	42, 10	49, 13
p_{12}	26, 2	24, 3	30, 5	46, 9	71, 16	27, 4	39, 9	60, 16	75, 23
p_{13}	44, 2	37, 3	46, 5	77, 10	127, 19	42, 4	60, 9	101, 18	125, 26
p_{14}	52, 2	44, 3	58, 6	97, 11	163, 21	49, 4	75, 11	125, 20	163, 32
p_{15}	85, 2	67, 3	90, 6	161, 12	287, 24	74, 4	116, 11	206, 22	272, 36
p_{17}	27, 2	27, 3	33, 5	51, 9	78, 16	30, 4	45, 10	69, 17	87, 25
p_{18}	53, 2	49, 4	63, 7	105, 13	174, 24	56, 6	87, 16	145, 29	188, 44
p_{19}	86, 2	74, 4	97, 7	172, 14	302, 27	86, 6	135, 16	240, 32	313, 49
p_{20}	61, 2	58, 3	78, 6	129, 11	216, 21	67, 4	109, 12	181, 22	243, 36
p_{21}	97, 2	88, 4	119, 8	210, 15	370, 29	102, 6	169, 19	298, 35	403, 58
p_{22}	151, 2	132, 4	183, 8	339, 16	626, 32	154, 6	263, 19	486, 38	669, 64

α

λ	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}
p_4	47, 11	71, 20	18, 2	32, 6	48, 11	37, 8	55, 14
p_7	67, 20	101, 35	27, 5	49, 13	74, 23	58, 17	88, 30
p_8	90, 25	141, 45	33, 6	63, 16	97, 28	78, 23	119, 39
p_9	161, 31	266, 60	51, 6	105, 18	172, 34	129, 26	210, 48
p_{11}	74, 23	110, 39	30, 6	56, 16	86, 29	67, 21	102, 37
p_{12}	116, 40	180, 69	45, 12	87, 30	135, 52	109, 42	169, 72
p_{13}	206, 49	335, 90	69, 12	145, 34	240, 64	181, 48	298, 89
p_{14}	272, 59	455, 110	87, 15	188, 42	313, 77	243, 63	403, 114
p_{15}	479, 71	840, 140	135, 15	313, 47	548, 92	408, 72	710, 139
p_{17}	135, 43	210, 74	55, 15	105, 35	162, 59	132, 49	207, 84
p_{18}	313, 80	520, 145	105, 24	227, 64	378, 116	295, 94	493, 171
p_{19}	548, 95	950, 181	162, 24	378, 72	664, 140	496, 108	868, 209
p_{20}	408, 67	690, 126	132, 18	295, 52	496, 96	396, 81	666, 149
p_{21}	710, 110	1250, 211	207, 30	493, 88	868, 166	666, 138	1173, 262
p_{22}	1229, 128	2250, 256	324, 30	822, 98	1508, 196	1128, 158	2064, 316

α

λ	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}
p_4	$81,24$	$118,40$	$62,17$	$90,28$	$129,45$	$182,70$	$252,105$
p_7	$132,51$	$197,85$	$104,38$	$156,64$	$232,105$	$343,169$	$504,267$
p_8	$183,67$	$282,115$	$144,52$	$219,87$	$333,145$	$504,239$	$756,387$
p_9	$339,88$	$543,160$	$252,64$	$402,114$	$634,200$	$987,344$	$1512,576$
p_{11}	$154,63$	$230,104$	$123,48$	$186,81$	$280,134$	$420,218$	$630,351$
p_{12}	$263,123$	$410,209$	$212,99$	$330,168$	$515,284$	$805,478$	$1260,801$
p_{13}	$486,162$	$785,290$	$372,123$	$606,222$	$980,395$	$1575,694$	$2520,1206$
p_{14}	$669,208$	$1110,380$	$516,165$	$852,297$	$1405,535$	$2310,960$	$3780,1710$
p_{15}	$1229,268$	$2115,515$	$912,204$	$1566,387$	$2670,730$	$4515,1365$	$7560,2520$
p_{17}	$324,143$	$510,244$	$264,118$	$417,202$	$660,345$	$1050,590$	$1680,1011$
p_{18}	$822,310$	$1370,561$	$644,251$	$1077,456$	$1800,827$	$3010,1499$	$5040,2718$
p_{19}	$1508,400$	$2600,758$	$1140,312$	$1980,597$	$3420,1134$	$5880,2142$	$10080,4032$
p_{20}	$1128,277$	$1920,518$	$894,228$	$1515,423$	$2580,789$	$4410,1477$	$7560,2772$
p_{21}	$2064,500$	$3630,959$	$1584,408$	$2787,780$	$4900,1496$	$8610,2877$	$15120,5544$
p_{22}	$3764,632$	$6840,1264$	$2820,504$	$5130,1008$	$9300,2016$	$16800,4032$	$30240,8064$

$\lambda^\# = (2,2,1)$, $d \leq 12$

α

λ	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}
p_3	$4,1$	$7,2$	$8,2$	$11,3$	$15,4$	$7,2$	$9,3$	$12,4$	$14,5$
p_6	$5,2$	$8,3$	$9,3$	$12,4$	$16,5$	$9,4$	$11,5$	$15,7$	$17,8$
p_7	$8,2$	$17,4$	$20,4$	$31,6$	$47,8$	$18,5$	$24,7$	$37,11$	$44,13$
p_{11}	$9,3$	$18,5$	$21,5$	$32,7$	$48,9$	$21,8$	$27,10$	$42,16$	$49,18$
p_{12}	$11,2$	$24,4$	$30,4$	$46,6$	$71,8$	$27,6$	$39,9$	$60,14$	$75,17$
p_{13}	$15,3$	$37,6$	$46,6$	$77,9$	$127,12$	$42,10$	$60,14$	$101,24$	$125,28$
p_{17}	$12,1$	$27,3$	$33,3$	$51,5$	$78,7$	$30,3$	$45,6$	$69,9$	$87,12$
p_{18}	$19,2$	$49,5$	$63,5$	$105,8$	$174,11$	$56,7$	$87,12$	$145,20$	$188,25$
p_{19}	$26,3$	$74,7$	$97,7$	$172,11$	$302,15$	$86,12$	$135,19$	$240,34$	$313,41$

α

λ	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}
p_3	$19,7$	$26,10$	$10,4$	$15,6$	$20,8$	$17,7$	$23,10$
p_6	$23,11$	$31,15$	$12,6$	$19,10$	$26,14$	$22,12$	$30,17$
p_7	$67,20$	$101,30$	$27,9$	$49,17$	$74,26$	$58,21$	$88,33$
p_{11}	$74,27$	$110,39$	$30,12$	$56,24$	$86,38$	$67,30$	$102,47$
p_{12}	$116,26$	$180,39$	$45,12$	$87,24$	$135,38$	$109,31$	$169,50$
p_{13}	$206,45$	$335,69$	$69,18$	$145,40$	$240,68$	$181,52$	$298,88$
p_{17}	$135,19$	$210,30$	$55,10$	$105,18$	$162,27$	$132,23$	$207,38$
p_{18}	$313,41$	$520,65$	$105,18$	$227,39$	$378,65$	$295,52$	$493,90$
p_{19}	$548,69$	$950,110$	$162,27$	$378,65$	$664,116$	$496,88$	$868,158$

α

λ	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}
p_3	$31,14$	$42,20$	$26,12$	$35,17$	$47,24$	$63,34$	$84,48$
p_6	$41,24$	$56,34$	$35,21$	$48,30$	$66,43$	$91,62$	$126,90$
p_7	$132,51$	$197,79$	$104,42$	$156,66$	$232,103$	$343,161$	$504,252$
p_{11}	$154,73$	$230,112$	$123,61$	$186,96$	$280,151$	$420,238$	$630,378$
p_{12}	$263,80$	$410,128$	$212,68$	$330,111$	$515,182$	$805,301$	$1260,504$
p_{13}	$486,147$	$785,242$	$372,120$	$606,204$	$980,346$	$1575,588$	$2520,1008$
p_{17}	$324,61$	$510,100$	$264,52$	$417,87$	$660,145$	$1050,245$	$1680,420$
p_{18}	$822,153$	$1370,260$	$644,128$	$1077,225$	$1800,395$	$3010,700$	$5040,1260$
p_{19}	$1508,279$	$2600,485$	$1140,228$	$1980,414$	$3420,750$	$5880,1365$	$10080,2520$

Literaturverzeichnis

[1] Raymond Hill. *A First Course in Coding Theory.* Oxford Applied Mathematics and Computing Science Series. Clarendon Press, 1986.

[2] Gorden D. James. *The Representation Theory of the Symmetric Groups.* Lecture Notes of Mathematics. Springer Verlag, 1978.

[3] W.J. LeVeque. *Topics in Number Theory.* Addison-Wesley Publishing Company, 1956.

[4] R. Lidl. *Finite Fields.* Encyclopedia of Mathematics and its Applications. Addison-Wesley, 1986.

[5] F.J. MacWilliams. *The Theory of Error-Correcting Codes.* north-holand publishing company, 1978.

[6] Alfred Scheerhorn. *Trace- and Norm-Compatible Extensions of Finite Fields.* Doktorarbeit (pre version). Lehrstuhl Informatik, Univ. Erlangen, 1992.

[7] A.W. Wong. *Modules as Codes.* Thesis, Univ. Colorado, Fort Collins, 1977.

[8] K.-H. Zimmermann. *Über die Automorphismengruppen von MDS-Codes.* Preprint, 1992.

[9] K.-H. Zimmermann. *On Weight Spaces of Polynomial Representations of the General Linear Group as Linear Codes.* erscheint im Journal of Combinatorical Theory, Series A., 1992.

Urhebervermerk

Ich versichere, daß ich diese Arbeit selbständig angefertigt habe.
Es wurden nur die angegebenen Hilfsmittel und Quellen verwendet.

Bayreuth, den 14. August 1992

. . . . . .

Ulrich Eidt