Masterarbeit

Theoretische und experimentelle Untersuchungen zu Normalbasen für Erweiterungen endlicher Körper

vorgelegt von

Stefan Hackenberg

am

Institut für Mathematik

der

Universität Augsburg

betreut durch

Prof. Dr. Dirk Hachenberger

Stand

26. Oktober 2014

Inhaltsverzeichnis

Kapitel 1

Grundbegriffe

In diesem Kapitel wollen wir ein paar grundlegende Resultate wiederholen, die dem Leser sicherlich bekannt sind. Daher werden wir die meisten Aussagen ohne Beweis lediglich zitieren.

1.1 | Ein wenig Gruppentheorie

Um später den Zerfall der Kreisteilungspolynome übe endlichen Körpern zu verstehen, wiederholen wir zunächst ein paar Aussagen über zyklische Gruppen.

[7, Theorem 1.15] fasst alle notwendigen Resultate zusammen.

Satz 1.1.

(1) Jede Untergruppe einer zyklischen Gruppe ist wieder zyklisch.

(2) Sei $\langle a \rangle$ eine zyklische Gruppe der Ordnung m, so erzeugt a^k eine Untergruppe der Ordnung $\frac{m}{\mathrm{ggT}(m,k)}$.

(3) Sei $\langle a \rangle$ eine zyklische Gruppe der Ordnung m und $d \mid m$, so enthält $\langle a \rangle$ genau eine Untergruppe der u Ordnung d.

(4) Sei f ein positiver Teiler der Gruppenordnung einer endlichen zyklischen Gruppe $\langle a \rangle$. Dann enthält $\langle a \rangle$ genau $\phi(f)$ Elemente der Ordnung f.

(5) Eine zyklische Gruppe der Ordnung m enthält genau $\varphi(m)$ (φ bezeichne die Eulersche Phifunktion) Erzeuger. Ist a ein Erzeuger, so sind alle Erzeuger der Form a^r mit $\mathrm{ggT}(r, m) = 1$.

Da wir später ein paar Eigenschaften der Eulerschen Phifunktion benötigen werden, wiederholen wir die wohlbekannte Definition der Eulerschen Phifunktion und geben wir dann die wichtigsten

Rechenregeln an.

Definition 1.2 (quadratfreier Teil).

Sei $n \in \mathbb{N}$ und $n = p_1^{r_1} \cdot \ldots \cdot p_l^{r_l}$ seine Primfaktorzerlegung. Dann heißt

$$\nu(n) := p_1 \cdot \ldots \cdot p_l$$

quadratfreier Teil von n.

Lemma 1.3 (Rechenregeln der Eulerschen Phifunktion). *Sei $a, b \in \mathbb{N}^*$, so gilt*

(1) $\varphi(ab) = \varphi(a)\varphi(b)$, *falls* $\mathrm{ggT}(a,b) = 1$,

(2) $\varphi(a) = \sum_{d|a} \varphi(d)$ *und*

(3) $\varphi(a) = \frac{a}{\nu(a)}\varphi(\nu(a))$.

1.2 | Automorphismen über endlichen Körpern

Satz 1.4.

Seien F ein endlicher Körper der Charakteristik $p \neq 0$ und $n \in \mathbb{N}_{>0}$. Dann ist

$$\begin{aligned} \sigma_n : \; F &\to F \\ a &\mapsto a^{p^n} \end{aligned}$$

ein Automorphismus auf F.

Beweis. [12, Corollary 3.18]. □

Bemerkung 1.5. Insbesondere gilt also für alle $a, b \in F$, F wie oben:

$$(a \pm b)^{p^n} = a^{p^n} \pm b^{p^n} .$$

Satz 1.6.

Sei q eine Primpotenz und $n \in \mathbb{N}_{>0}$. Der Automorphismus

$$\begin{aligned} \sigma : \; \mathbb{F}_{q^n} &\to \mathbb{F}_{q^n} \\ a &\mapsto a^q \end{aligned}$$

hält die Elemente von $\mathbb{F}_q$ fest, also

$$\sigma|_{\mathbb{F}_q} = \mathrm{id}_{\mathbb{F}_q} .$$

Ferner ist $\sigma^k \neq \mathrm{id}_{\mathbb{F}_{q^n}}$ für $k = 1, \ldots, n-1$, alle σ^ks sind paarweise verschiedene Automorphismen und $\sigma^n = \mathrm{id}_{\mathbb{F}_{q^n}}$.

Bezeichne $\mathrm{Gal}(E \mid F)$ die Galoisgruppe einer Galoiserweiterung E über F, so können wir das folgende zentrale Resultat zitieren:

Satz 1.7.

Es gilt

$$\mathrm{Gal}(\mathbb{F}_{q^n} \mid \mathbb{F}_q) = \langle \sigma \rangle .$$

Das bedeutet, dass es neben $\sigma, \ldots, \sigma^{n-1}$ keine weiteren Automorphismen von $\mathbb{F}_{q^n}$ gibt, die $\mathbb{F}_q$ fixieren.

Beweis. [12, Theorem 7.3]. □

Neben der Tatsache, dass der Frobenius alle Endomorphismen erzeugt, können wir auch zeigen, dass alle Potenzen des Frobenius von $\mathbb{F}_q$ über $\mathbb{F}_{q^n}$ linear unabhängig sind. Dies gilt sogar in einem größeren Kontext:

Satz 1.8 (Dedekindsches Lemma).

Seinen K, L zwei Körper, $n \in \mathbb{N}$ und $\tau_1, \ldots, \tau_n : K \to L$ injektive Körperhomorphismen. Dann ist für jedes $x \in K$

$$\{\tau_1(x), \ldots, \tau_n(x)\}$$

linear unabhängig über L.

Beweis. [5, Satz 27.2]. □

Mit Satz 1.7 wird klar, dass für ein irreduzibles Polynom $f(X) \in \mathbb{F}_q[X]$, das in $\mathbb{F}_{q^n}$ eine Nullstelle α besitzt, auch $\sigma^i(\alpha)$ für alle $i = 1, \ldots, n-1$ Nullstellen sind. Ferner kann man sich relativ einfach überlegen, dass für ein irreduzibles Polynom $f(X) \in \mathbb{F}_q[X]$ eine Nullstelle in $\mathbb{F}_{q^n}$ zu besitzen hinreichend ist, dass $f(X)$ vom Grad n ist. Beides fasst nachstehender Satz zusammen.

Satz 1.9.

Ist $f(X) \in \mathbb{F}_q[X]$ ein irreduzibles Polynom vom Grad n. Dann existiert eine Nullstelle α von $f(X)$ in $\mathbb{F}_{q^n}$, alle Nullstellen von $f(X)$ sind einfach und gegeben durch

$$\alpha, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}} \in \mathbb{F}_{q^n} .$$

Beweis. [7, Theorem 2.14]. □

Kapitel 2

Der Zerfall von $x^n - 1$ und die Kreisteilungspolynome $\Phi_D(x)$

Sei K ein beliebiger Körper der Charakteristik p und $\bar{K}$ ein fest gewählter algebraischer Abschluss. Wir wollen nun untersuchen, wie das Polynom $x^n - 1 \in K[x]$ über K zerfällt. Dazu orientieren wir uns an [7] und [12].

Definition 2.1 (Kreisteilungskörper, Einheitswurzeln).

Sei $n \in \mathbb{N}^*$. Der Zerfällungskörper von $X^n - 1 \in K[X]$ heißt der *n-te Kreisteilungskörper* und wird mit $K^{(n)}$ notiert. Die Nullstellen von $X^n - 1$ in $K^{(n)}$ heißen *n-ten Einheitswurzeln* und die Menge derer wird mit $E^{(n)}$ bezeichnet.

Satz 2.2.

Sei $n \in \mathbb{N}^*$.

(1) Sei $p \nmid n$. Dann ist $E^{(n)}$ eine zyklische Gruppe (bzgl. der Multiplikation in $K^{(n)}$) der Ordnung n.

(2) Ist $p \mid n$ und schreibt man $n = p^e m$ für positive ganze Zahlen m und e mit $p \nmid m$, so ist $K^{(n)} = K^{(m)}$ und $E^{(n)} = E^{(m)}$ und die Nullstellen von $X^n - 1 \in K[X]$ sind gerade die Elemente in $E^{(m)}$ jedoch jeweils mit Multiplizität p^e.

Beweis. [7, Theorem 2.42]. □

Definition 2.3 (primitive Einheitswurzeln).

Sei $n \in \mathbb{N}^*$ und $p \nmid n$. Dann heißen die Erzeuger von $E^{(n)}$ *primitive n-te Einheitswurzeln*. Die Untergruppe der primitiven Einheitswurzeln wird mit $C^{(n)}$ bezeichnet.

Definition 2.4 (Kreisteilungspolynom).

Seien $n \in \mathbb{N}^*$, $p \nmid n$. Das Polynom

$$\Phi_n(X) := \prod_{\zeta \in C^{(n)}} (X - \zeta)$$

heißt *d-tes Kreisteilungspolynom.*

Satz 2.5.

Seien K ein Körper der Charakteristik p und $n \in \mathbb{N}^*$ mit $p \nmid n$. Dann gilt:

(1) $X^n - 1 = \prod_{d|n} \Phi_d(X)$.

(2) $\Phi_n(X) \in P[X]$, *wobei $P \subset K$ den Unterkörper mit p Elementen notiert.*

Beweis. (1) Dies ist eine einfache Folgerung aus Satz 1.1.

(2) Lässt sich per Induktion recht einfach beweisen (vgl. [7, Theorem 2.45 (ii)]). □

Definition 2.6.

Für zwei teilerfremde natürliche Zahlen q, n größer Null sei

$$\mathrm{ord}_n(q) := \mathrm{ord}([q]_n)$$

die *multiplikative Ordnung von q modulo n*, wobei $[q]_n$ die Restklasse von q in $\mathbb{Z}_n$ bezeichnet und die Ordnung als Gruppenordnung in den Einheiten von $\mathbb{Z}_n$, notiert durch $\mathbb{Z}_n^\times$, zu lesen ist.

Lemma 2.7 (Rechenregeln der multiplikation Ordnung modulo n). *Seien $m, n, q \in \mathbb{N}^*$ mit* $\mathrm{ggT}(n, q) = 1$, $\mathrm{ggT}(m, q) = 1$ *und* $\mathrm{ggT}(m, n) = 1$, *so gilt*

(1) $\mathrm{ord}_n(q) \mid \varphi(n)$,

(2) $\mathrm{ord}_{mn}(q) = \mathrm{kgV}\{\mathrm{ord}_m(q), \mathrm{ord}_n(q)\}$.

Beweis. (1) Klar, da $[q]_n$ in $\mathbb{Z}_n^\times$ eine Untergruppe der Ordnung $\mathrm{ord}_n(q)$ erzeugt. Nach dem Satz von Lagrange teilt deren Ordnung die Gruppenordnung $|\mathbb{Z}_n^\times| = \varphi(n)$.

(2) Nach dem Chinesischen Restsatz (z.B. [2, Kapitel 2 Satz 12]) ist

$$\begin{array}{rccc} f: & \mathbb{Z}_{nm} & \xrightarrow{\cong} & \mathbb{Z}_n \times \mathbb{Z}_m\,, \\ & [x]_{nm} & \mapsto & ([x]_n, [x]_m) \end{array}$$

ein Isomorphismus von Ringen, da algebraisch $\mathbb{Z}_n$ ja nichts anderes ist, als $\mathbb{Z}/(n)$, wobei (n) das von n im Ring $\mathbb{Z}$ erzeugte Ideal meint. Dieser liefert einen Gruppenhomomorphismus auf den Einheiten:

$$f : \mathbb{Z}_{nm}^\times \to \mathbb{Z}_n^\times \times \mathbb{Z}_m^\times\,.$$

Nun ist per definitionem von $\mathrm{ord}_\bullet(q)$ die Behauptung klar. □

Damit können wir nun zu einem zentralen Resultat dieses Abschnittes kommen, das uns über die gesamte Arbeit hinweg begleiten wird.

Satz 2.8.

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$ mit $\mathrm{ggT}(q,n) = 1$. Dann zerfällt das n-te Kreisteilungspolynom $\Phi_n(X)$ über $\mathbb{F}_q$ in

$$\frac{\varphi(n)}{\mathrm{ord}_n(q)}$$

irreduzible paarweise teilerfremde Polynome von jeweils Grad $\mathrm{ord}_n(q)$.

Beweis. Sei $f(X) \mid \Phi_n(X)$ ein irreduzibler Teiler über $\mathbb{F}_q$. Ist dann $\zeta \in C^{(n)}$ eine Nullstelle von $f(X)$, so sind nach Satz 1.9 auch

$$\zeta^q, \zeta^{q^2}, \ldots, \zeta^{q^n}$$

Nullstellen von $f(X)$. Jedoch sind offenbar nur $\mathrm{ord}_n(q)$ dieser verschieden und da f als irreduzibles Polynom wieder nach Satz 1.9 nur einfache Nullstellen besitzt, können wir folgern, dass $\deg f = \mathrm{ord}_n(q)$. Da $f(X)$ als beliebiger irreduzibler Teiler von $\Phi_n(X)$ gewählt wurde, folgt sofort die Behauptung, wenn man sich überlegt, dass der Grad des n-ten Kreisteilungspolynoms per definitionem gerade $\varphi(n)$ ist. □

Lemma 2.9.

Im Beweis obigen Satzes haben wir gesehen, dass die Wirkung der Galoisgruppe $\mathrm{Gal}(\mathbb{F}_{q^n} \mid \mathbb{F}_q)$ auf der Menge der primitiven n-ten Einheitswurzeln $C^{(n)}$ (die Wirkung ist selbstredend durch Einsetzen gegeben) diese in Teilmengen der Mächtigkeit $\mathrm{ord}_n(q)$ zerlegt. Dies lässt sich natürlich auf $E^{(n)}$ übertragen, da ja gerade nach Satz 1.1 $E^{(n)} = \dot{\bigcup}_{d|n} C^{(d)}$. Dies motiviert nachstehende Definition.

Definition 2.10.

Für $m, q \in \mathbb{N}$ mit $\mathrm{ggT}(m,q) = 1$ und $j \in \mathbb{Z}_m$ definieren wir

$$M_q(j \bmod m) := \{j\, q^i \bmod m \mid i \in \mathbb{N}\} = \{j,\ jq,\ jq^2,\ jq^3, \ldots \bmod m\}$$

Ein vollständiges Repräsentantensystem von Nebenklassen von $M_q(1 \bmod m)$ in $\mathbb{Z}_m$ sei mit $R_q(m)$ bezeichnet.

Beispiel 2.11. Wollen wir den Zerfall von $X^{21} - 1$ über $\mathbb{F}_2$ untersuchen, so berechnen wir erst ein Vertretersystem von Restklassen mod21:

$l \in R_2(21)$	$M_2(l \bmod 21)$
0	0
1	1, 2, 4, 8, 11, 16
3	3, 6, 12
5	5, 10, 13, 17, 19, 20
7	7, 14
9	9, 15, 18

Nun wissen wir aus Satz 2.5, dass

$$X^{21} - 1 = \Phi_1(X) \cdot \Phi_3(X) \cdot \Phi_7(X) \cdot \Phi_{21}(X)\,.$$

Die Nullstellen von $\Phi_{21}(X)$ partitionieren sich gerade in diejenigen $M_2(l \bmod 21)$ für die $l = 1, 5$. Also haben wir

$$\begin{aligned}\Phi_{21}(X) &= (X^6 + X^4 + X^2 + X + 1) \cdot (X^6 + X^5 + X^4 + X^2 + 1)\\ &= (X-\zeta)(X-\zeta^2)(X-\zeta^4)(X-\zeta^8)(X-\zeta^{11})(X-\zeta^{16}) \cdot (X-\zeta^5)(X-\zeta^{10})(X-\zeta^{13})(X-\zeta^{17})(X-\zeta^{19})(X-\zeta^{20})\end{aligned}$$

falls wir $\zeta \in C^{(21)}$ als Nullstelle von $X^6 + X^4 + X^2 + X + 1$ setzen. Analog erhalten wir den Zerfall von $\Phi_7(X)$ durch Betrachtung der $M_2(l \bmod 21)$ für $l = 3, 9$.

$$\begin{aligned}\Phi_7(X) &= (X^3 + X + 1) \cdot (X^3 + X^2 + 1)\\ &= (X-\zeta^3)(X-\zeta^{3\cdot 2})(X-\zeta^{3\cdot 4}) \cdot (X-\zeta^{3\cdot 3})(X-\zeta^{3\cdot 5})(X-\zeta^{3\cdot 6})\end{aligned}$$

Sammeln wir den Rest auf, erhalten wir die Partitionierung für $\Phi_3(X)$ und den trivialen Fall $\Phi_1(X)$.

$$\begin{aligned}\Phi_3(X) &= X^2 + X + 1\\ &= (X-\zeta^7)(X-\zeta^{14})\,,\end{aligned}$$

$$\begin{aligned}\Phi_1(X) &= X - 1\\ &= X - \zeta^0\,.\end{aligned}$$

▲

Nun können wir uns überlegen, ob und wie unterschiedliche Kreisteilungspolynome zusammenhängen und kommen dabei auf die bekannten Resultate, die z.B. in [3, Proposition 10.6, 10.7] zu finden sind. Um diese anzugeben, benötigen wir jedoch noch eine Definitionen und zitieren einige Eigenschaften.

Definition 2.12.

Seien $r, n \in \mathbb{N}$, so definiere

$$\pi_r(n) := \max\{k \in \mathbb{N}^* : k \mid n,\ \nu(k) \mid \nu(r)\}\,.$$

Lemma 2.13. *Seien $q > 1$ eine ganze Zahl, $n \in \mathbb{N}^*$ und r ein Primteiler von $q - 1$. Dann gilt:*

(1) Ist $r \neq 2$ oder $q \equiv 1 \bmod 4$, so gilt

$$\pi_r(q^{r^n} - 1) = r^n\, \pi_r(q-1)\,.$$

(2) Ist $q \equiv 3 \bmod 4$, so gilt

$$\pi_2(q^{2^n} - 1) = 2^{n-1}\, \pi_2(q^2 - 1)\,.$$

Beweis. [3, Lemma 19.4]. □

Lemma 2.14. *Seien $q, m, k > 1$ ganze Zahlen mit $\nu(k) \mid \nu(m) \mid q - 1$. Dann gilt*

(1) Ist m ungerade oder $q \equiv 1 \bmod 4$ oder k ungerade, so gilt

$$\pi_m(q^k - 1) = k\, \pi_m(q-1)\,.$$

(2) Ist m gerade, $q \equiv 3 \bmod 4$ und k gerade, so ist

$$\pi_m(q^k - 1) \;=\; \tfrac{k}{2}\,\pi_m(q^2 - 1)\,.$$

Beweis. [3, Lemma 19.5]. □

Satz 2.15.

Seien $t, k \in \mathbb{N}^*$.

(1) Ist $\nu(t) \mid k$, so gilt

$$\Phi_k(X^t) \;=\; \Phi_{kt}(X) \;\in K[X]\,.$$

(2) Sind t und k teilerfremd, so gilt

$$\Phi_k(X^t) \;=\; \prod_{d \mid t} \Phi_{kd}(X) \;\in K[X]\,.$$

(3) Insbesondere gilt: Ist $q = p^r$ eine Primzahlpotenz, $t, k \in \mathbb{N}^*$ mit $p \nmid t, k$ und π eine Potenz von p. Sei ferner $t = \pi_k(t) \cdot \bar{t}$, so gilt

$$\Phi_k(X^{t\pi}) \;=\; \left(\prod_{d \mid \bar{t}} \Phi_{k\,d\,\pi_k(t)}(X)\right)^{\pi} \;\in \mathbb{F}_q[X]\,.$$

Beweis. Dass sich Potenzen von p aus dem Argument herausziehen lassen, ist klar, da $\mathrm{id}_P = (.)^\pi : P \to P$ für den Primkörper $P \subset K$ nach Satz 1.4 eine lineare Abbildung ist. Ferner haben nach Satz 2.5 die Kreisteilungspolynome nur Koeffizienten in P.

Der Kern des Beweises des Rests liegt in der Betrachtung des Gruppenhomomorphismus

$$\psi_n :\; \bar{K}^* \to \bar{K}^*,\; x \mapsto x^n$$

für $p \nmid n$. Denn nun ist offensichtlich, dass die Nullstellen von $\Phi_k(X^t)$ gerade alle Elemente in $\bar{K}^*$, deren t-te Potenz eine primitive k-te Einheitswurzel ist, sind, also $\psi_t^{-1}(C^{(k)})$. Ergo formulieren sich die Aussagen wie folgt um:

(1') Ist $\nu(t) \mid k$, so gilt $\psi_t^{-1}(C^{(k)}) \;=\; C^{(kt)}$.

(2') Ist $\mathrm{ggT}(t, k) = 1$, so gilt $\psi_t^{-1}(C^{(k)}) \;=\; \dot\bigcup_{d \mid t} C^{(kd)}$.

(3') Ist $k, t \in \mathbb{N}^*$ mit $p \nmid t, k$ und $k = \pi_k(t)\bar{t}$, so gilt

$$\psi_t^{-1}(C^{(k)}) \;=\; \dot\bigcup_{d \mid \bar{t}} C^{(k\,d\,\pi_k(t))}$$

Nun ist offensichtlich, dass es reicht (3') zu zeigen. Dazu notiere $t_0 := \pi_k(t)$ und seien $d \mid \bar{t}$ und $\zeta \in C^{(kdt_0)}$ beliebig. Dann ist

$$\mathrm{ord}(\zeta^t) = \mathrm{ord}((\zeta^{t_0 d})^{\frac{\bar{t}}{d}}) = k\,,$$

da per definitionem von $\pi_k(t)$ gerade $\mathrm{ggT}(\bar{t}, kt_0) = 1$. Also gilt $\psi_t(C^{(kdt_0)}) \subseteq C^{(k)}$ und damit

$$\dot{\bigcup_{d|\bar{t}}} C^{(kdt_0)} \subseteq \psi_t^{-1}\psi_t(\dot{\cup}_{d|\bar{t}} C^{(kdt_0)}) \subseteq \psi_t^{-1}(C^{(k)})$$

Die Gleichheit folgt mit einem Zählargument: Auf der einen Seite ist

$$\left|\dot{\bigcup_{d|\bar{t}}} C^{(kdt_0)}\right| = \sum_{d|\bar{t}} \varphi(kdt_0) = \varphi(kt_0) \sum_{d|\bar{t}} \varphi(d) = \varphi(kt_0) \cdot \bar{t} = \varphi(k)t\,,$$

wobei an Lemma 1.3 erinnert sei. Auf der anderen Seite haben wir

$$\left|\psi_t^{-1}(C^{(k)})\right| = t|C^{(k)}| = t\varphi(k)\,,$$

was den Beweis abschließt. □

Bevor wir den Zerfall der Kreisteilungspolynome noch genauer untersuchen, kann man als einfache Folgerung angeben, wann genau ein Binom $x^n - \beta \in \mathbb{F}_q[x]$ irreduzibel ist.

Satz 2.16.

Seien $\beta \in F_q^*$ und $n \in \mathbb{N}$. Es gilt: $x^n - \beta \in \mathbb{F}_q[x]$ ist genau dann irreduzibel, wenn

(1) $p := \mathrm{char}\,\mathbb{F}_q \nmid n$,

(2) $\nu(n) \mid e := \mathrm{ord}(\beta)$ *und*

(3) $\mathrm{ord}_{ne}(q) = n$.

Beweis. Zunächst ist klar, dass $p \nmid n$ erfüllt sein muss, da ansonsten $\beta' \in \mathbb{F}_q^*$ existiert mit $\beta'^p = \beta$ ($(\cdot)^p$ ist ein Automorphismus auf $\mathbb{F}_q$ nach Satz 1.4). Damit wäre $x^n - \beta = (x^{\frac{n}{p}} - \beta')^p$ eine Faktorisierung. Nun sei u eine Nullstelle von $x^n - \beta$, so lässt sich beobachten, dass (in Notation des Beweises von Satz 2.15) $u \in \psi_n^{-1}(C^{(e)})$. Damit gilt nach Satz 2.15 (3)

$$x^n - \beta = \prod_{d|\bar{n}} \mathrm{ggT}(x^n - \beta, \Phi_{en_0d}(x))$$

für $n = \pi_e(n)\bar{n}$ und diese Zerlegung ist, wie man sich analog zum Beweis von Satz 2.8 überlegen kann, nicht trivial (vgl. [8, Theorem ...]). Damit ist (2) der Behauptung klar, so dass $x^n - \beta \mid \Phi_{ne}(x)$. Ferner zerfällt nach Satz 2.8 $\Phi_{ne}(x)$ in $\frac{\varphi(ne)}{\mathrm{ord}_{ne}(q)}$ irreduzible Faktoren von jeweils Grad $\mathrm{ord}_{ne}(q)$. Damit wird auch (3) der Behauptung augenblicklich klar. □

Für die letzte Bedingung in obigem Satz existieren noch verschiedene weitere äquivalente Charakterisierungen, die nachstehend zu finden sind.

Satz 2.17.

Seien p eine Primzahl, q eine Potenz von p und $n, e \in \mathbb{N}^*$ mit $p \nmid n$ und $\nu(n) \mid e \mid q - 1$. Dann sind äquivalent:

(1) $\mathrm{ord}_{ne}(q) = n$,

(2) $\mathrm{ggT}(\frac{q-1}{e}, n) = 1$ *und* $q \equiv 1 \bmod 4$, *falls* $4 \mid n$, *und*

(3) $\pi_n(q-1) \mid e$ *und* $q \equiv 1 \bmod 4$, *falls* $4 \mid n$.

Beweis. [8, Theorem ...]. □

Wir haben nun erkannt, wann genau Binome über einem endlichen Körper irreduzibel sind. Doch wenn man dem Titel dieses Kapitels Glauben schenken mag, interessieren wir uns hier vorangig für den Zerfall der Kreisteilungspolynome über endlichen Körpern. Diese sind im Allgemeinen keine Binome, aber genau das Wissen über die Irreduzibiltät von Binomen lässt uns Bedingungen formulieren, die dazu führen, dass ein Kreisteilungspolynom über einem gegebenen endlichen Körper in irreduzible Binome zerfällt Später (Abschnitt 4.2) werden wir diese Bedingungen *stark regulär* (Definition 4.5) nennen und einsehen, dass sie eine wesentliche Rolle bei der expliziten Konstruktion von Normalbasen spielen.

Satz 2.18.

Seien $\mathbb{F}_q$ ein endlicher Körper von Charakteristik p und $m \in \mathbb{N}^*$. Es gelte $p \nmid m$, $\nu(m) \mid q-1$ und $4 \mid q-1$, falls $2 \mid m$. Setze $l := \pi_m(q-1)$, $a := \mathrm{ggT}(l, m)$ und $I_a := \{j \in \mathbb{N}^* : \ j \geq a, \ \mathrm{ggT}(j, a) = 1\}$. Ist $\zeta \in \mathbb{F}_q^*$ eine primitive a-te Einheitswurzel, so ist

$$\Phi_m(x) \ = \ \prod_{j \in I_a} \left(x^{\frac{m}{a}} - \zeta^j \right)$$

die vollständige Faktorisierung des m-ten Kreisteilungspolynoms über $\mathbb{F}_q$.

Beweis. Wir stellen fest, dass $\mathbb{F}_q$ in der Tat a-te Einheitswurzeln enthält, da $\mathrm{ord}_a(q) = 1$. Dies ist klar, da l per definitionem $q-1$ teilt und $a = \mathrm{ggT}(l, m)$. Nun wollen wir uns klar werden, dass beide Seiten obiger Gleichung auch identisch sind: Für $j \in I_a$ durchläuft ζ^j alle primitiven a-ten Einheitswurzeln und damit sind die Nullstellen der rechten Seite der Gleichung gerade alle primitiven m-ten Einheitswurzeln. Bleibt die Irreduzbilität von $x^{\frac{m}{a}} - \zeta^j$ zu zeigen, wobei wir ohne Einschränkung $j = 1$ wählen können: Klar ist, dass $p \nmid \frac{m}{a}$, da $p \nmid m$ nach Voraussetzungen. Ferner ist $\nu(l) = \nu(m)$, da wegen $\nu(m) \mid q-1$ gilt:

$$\pi_m(q-1) = \max\{k \in \mathbb{N}^* : \ k \mid q-1, \ \nu(k) = \nu(m)\}.$$

Also ist auch $\nu(a) = \nu(\mathrm{ggT}(l, m)) = \nu(m)$ und damit folgt $\nu(\frac{m}{a}) \mid \nu(m) = \nu(a) \mid a$, womit auch (2) in Satz 2.16 erfüllt wäre. Da $\nu(m) \mid q-1$ ist $\pi_{\frac{m}{a}}(q-1) \mid m$, also auch $\pi_{\frac{m}{a}}(q-1) \mid a$. Damit wäre durch die Bedingung $q \equiv 1 \bmod 4$, falls $2 \mid m$, auch (3) in Satz 2.17 erfüllt. □

Bemerkung 2.19. Man hätte obigen Beweis auch ohne das Wissen über irreduzible Binome führen können, in dem man sich Satz 2.8 bedient. So findet man dies auch in [3, Lemma 22.2].

Erinnert man sich nun erneut an Satz 2.8, so kann man sich die Frage stellen, ob man den Zusammenhang unterschiedlicher Kreisteilungspolynome aus Satz 2.15 in dem Sinne verfeinern kann, dass man sich nicht für das gesamte Kreisteilungspolynom interessiert, sondern lediglich für einen

irreduziblen Teiler. Diese Frage beantwortet nachstehender Satz.

Satz 2.20.

Seien $q = p^r$ eine Primzahlpotenz und $m, t \in \mathbb{N}$ mit $p \nmid m$, $p \nmid t$ und $\mathrm{ggT}(m,t) = 1$. Ist dann $f(x) \mid \Phi_m(x)$ ein über $\mathbb{F}_q$ irreduzibler monischer Teiler des m-ten Kreisteilungspolynoms. Definieren wir ferner

$$\Delta_q(m,d) \;:=\; \frac{\varphi(d)\,\mathrm{ord}_m(q)}{\mathrm{ord}_{md}(q)}\,,$$

so gilt:

$$f(x^t) \;=\; \prod_{d\mid t} \prod_{i=1}^{\Delta_q(m,d)} f_{d,i}(x)\,,$$

wobei für alle $i = 1, \ldots, \Delta_q(m,d)$

$$f_{d,i} \in \mathbb{F}_q[x] \text{ monisch, irreduzibel und } f_{d,i}(x) \mid \Phi_{md}(x)\,.$$

Ferner sind alle $f_{d,i}(x)$ paarweise teilerfremd.

Beweis. Wie schon im Beweis von Satz 2.15 betrachten wir den Gruppenhomomorphismus ψ_t, diesmal eingeschränkt auf $E^{(mt)}$:

$$\begin{aligned} \psi_t : \quad E^{(mt)} &\to E^{(m)}\,, \\ x &\mapsto x^t\,, \end{aligned}$$

was offenbar ein wohldefinierter Gruppenhomomorphismus bleibt. Offensichtlich ist $\ker \psi_t = E^{(t)}$. Da $\mathrm{ggT}(m,t) = 1$, also $E^{(mt)} = E^{(m)} \odot E^{(t)}$ als leichte Folgerung aus Satz 1.1, ist ψ auch surjektiv.

Ist nun $\alpha \in C^{(m)}$ eine Nullstelle von $f(x)$, so existiert – wiederum weil m und t teilerfremd sind – genau ein $\beta \in C^{(m)}$ mit $\beta^t = \alpha$. Damit ist also

$$\psi^{-1}(\{\alpha\}) \;=\; \beta\, E^{(t)} \;=\; \dot{\bigcup_{d\mid t}}\, \beta\, C^{(d)}\,.$$

Notiert wieder σ der Frobenius von $\mathbb{F}_q$, so sind nach Satz 1.9 $\sigma^j(\alpha)$, $j = 0, \ldots, \delta - 1$ für $\delta = \mathrm{ord}_q(m)$ die Nullstellen von $f(x)$. Da $p \nmid t$ bleibt die Menge der t-en Einheitswurzeln invariant unter σ und damit ist die Menge der Nullstellen von $f(x^t)$ gerade

$$\dot{\bigcup_{j=0}^{\delta-1}} \sigma^j(\beta)\, E^{(t)} \;=\; \dot{\bigcup_{j=0}^{\delta-1}}\, \dot{\bigcup_{d\mid t}}\, \beta^{q^j}\, C^{(d)} \;=\; \dot{\bigcup_{d\mid t}}\, \dot{\bigcup_{j=0}^{\delta-1}}\, \beta^{q^j}\, C^{(d)} \;=:\; \dot{\bigcup_{d\mid t}} N_d\,.$$

Wollen wir nun einsehen, wie $f(x^t)$ über $\mathbb{F}_q$ zerfällt, so müssen wir überlegen, wie obige Nullstellenmenge in σ-invariante Teilmengen zerfällt. Für jedes $d \mid t$ und jedes $j \in \{0, \ldots, \delta - 1\}$ ist $\zeta \in \beta^{q^j}\, C^{(d)}$ ein Element mit $\mathrm{ord}(\zeta) = md$, also Nullstelle von $\Phi_{md}(x)$. Ferner gilt offenbar $\forall d \mid t : \; |N_d| = \delta\varphi(d)$ und wir können folgern, dass N_d in genau

$$\frac{\delta\varphi(d)}{\mathrm{ord}_{md}(q)} \;=\; \frac{\mathrm{ord}_m(q)\,\varphi(d)}{\mathrm{ord}_{md}(q)} \;=\; \Delta_q(m,d)$$

σ-invariante Teilmengen zerfällt. $\Delta_q(m,d)$ ist in der Tat eine natürliche Zahl größer 0, da nach Lemma 2.7 (2)

$$\frac{\operatorname{ord}_m(q)\,\varphi(d)}{\operatorname{ord}_{md}(q)} = \frac{\varphi(d)\,\operatorname{ggT}(\operatorname{ord}_m(q),\operatorname{ord}_d(q))}{\operatorname{ord}_d(q)}$$

und $\operatorname{ord}_d(q) \mid \varphi(d)$ nach Lemma 2.7 (1). Damit ist alles gezeigt. □

Beispiel 2.21. Greifen wir noch einmal Beispiel 2.11 auf und betrachten einen irreduziblen Teiler $f(x)$ von $\Phi_7(x)$ über $\mathbb{F}_2$, sagen wir

$$f(x) := x^3 + x + 1\,.$$

Sei $t := 3$. Nun wissen wir nach Satz 2.20, dass $f(x^3)$ wie folgt über $\mathbb{F}_2$ zerfällt:

$$f(x^3) = \overbrace{\prod_{i=1}^{\Delta_2(7,1)} f_{1,i}(x)}^{d=1\mid 3} \cdot \overbrace{\prod_{i=1}^{\Delta_2(7,3)} f_{3,i}(x)}^{d=3\mid 3} = f_{1,1}(x)\cdot f_{3,1}(x)$$

da

$$\Delta_2(7,1) = \frac{\varphi(1)\operatorname{ord}_7(2)}{\operatorname{ord}_7(2)} = \frac{1\cdot 3}{3} = 1\,,$$
$$\Delta_2(7,3) = \frac{\varphi(3)\operatorname{ord}_7(2)}{\operatorname{ord}_{21}(2)} = \frac{2\cdot 3}{6} = 1\,.$$

Wir wollen nun herausfinden, welche Teiler $f_{1,1}(x)$ und $f_{3,1}(x)$ von $\varphi_7(x)$ und $\varphi_{21}(x)$ sind. Wir übernehmen den Zerfall der Kreisteilungspolynome aus Beispiel 2.11 und können einsehen, dass

$$f_{1,1}(x) = x^3 + x^2 + 1\,, \qquad f_{3,1}(x) = x^6 + x^5 + x^4 + x^2 + 1\,.$$ ▲

Beispiel 2.22. Als zweites Beispiel wollen wir uns einen Fall betrachten, in dem $\Delta_q(d,m)$ nicht immer 1 ist. Sei $p = q = 3$, $m = 5$ und $t := 4$. Also müssen wir ein Vertretersystem von Restklassen mod 20 betrachten:

$l \in R_3(20)$	$M_2(l \bmod 20)$
0	0
1	1, 3, 7, 9
2	2, 6, 14, 18
4	4, 8, 12, 16
5	5, 15
10	10
11	11, 13, 17, 19

Wir sehen, dass $\Phi_{20}(x)$ für $l = 1, 11$ in 2 Polynome von jeweils Grad 4 zerfällt:

$$\begin{aligned}\Phi_{20}(x) &= (x^4 + x^3 + 2x + 1) \cdot (x^4 + 2x^3 + x + 1)\\ &= (x-\zeta^{11})(x-\zeta^{13})(x-\zeta^{17})(x-\zeta^{19}) \cdot (x-\zeta)(x-\zeta^3)(x-\zeta^7)(x-\zeta^9)\,,\end{aligned}$$

wobei wir $\zeta \in C^{(20)}$ mit Minimalpolynom $x^4 + 2x^3 + x + 1$ gewählt haben. Nun können wir den Zerfall von $\Phi_5(x)$ und $\Phi_{10}(x)$ in Termen von ζ anhand der Restklassen mod 20 beschreiben:

$$\begin{aligned}\Phi_5(x) &= x^4 + x^3 + x^2 + x + 1\\ &= (x-\zeta^4)(x-\zeta^8)(x-\zeta^{12})(x-\zeta^{16})\,,\end{aligned}$$

$$\begin{aligned}\Phi_{10}(x) &= x^4 + 2x^3 + x^2 + 2x + 1\\ &= (x-\zeta^2)(x-\zeta^6)(x-\zeta^{14})(x-\zeta^{16})\,.\end{aligned}$$

Die Restklassen für $l = 0, 5, 10$ gehören zu den Kreisteilungspolynomen $\Phi_1(x), \Phi_4(x)$ und $\Phi_2(x)$, die wir für ein Beispiel zu Satz 2.20 nicht benötigen. Nun brauchen wir wieder einen irreduziblen monischen Teiler von $\Phi_m(x)$ und setzen daher $f(x) = \Phi_m(x)$. Wir berechnen wie oben

$$\begin{aligned}
\Delta_3(5,1) &= \frac{\varphi(1)\,\mathrm{ord}_5(3)}{\mathrm{ord}_5(3)} = \frac{1\cdot 4}{4} = 1\,,\\
\Delta_3(5,2) &= \frac{\varphi(1)\,\mathrm{ord}_5(3)}{\mathrm{ord}_{10}(3)} = \frac{1\cdot 4}{4} = 1\,,\\
\Delta_3(5,4) &= \frac{\varphi(4)\,\mathrm{ord}_5(3)}{\mathrm{ord}_{20}(3)} = \frac{2\cdot 4}{4} = 2\,.
\end{aligned}$$

Nun ist klar, wie $f(x^t)$ über $\mathbb{F}_3$ zerfällt:

$$f(x^3) = \underbrace{\left(x^4+x^3+x^2+x+1\right)}_{d=1\mid 4}\cdot\underbrace{\left(x^4+2x^3+x^2+2x+1\right)}_{d=2\mid 4}\cdot\underbrace{\left((x^4+x^3+2x+1)(x^4+2x^3+x+1)\right)}_{d=4\mid 4}$$

▲

Kapitel 3
Moduln

3.1 | Über Moduln über Hauptidealbereichen

Nähern wir uns der Situation von Normalbasen in möglichst allgemeiner Form, so beginnt die Reise bei der Betrachtung von Moduln über Hauptidealbereichen. Dazu wiederholen wir die wichtigsten Definitionen und Aussagen. Für eine intensivere Betrachtung sei auf Standardwerke der Algebra, z.B. [6] oder [4], verwiesen. Die Referenzen als Beweise seien ohne explizite Erwähnung immer als beispielhafte Angabe zu verstehen und es sei bemerkt, dass jene grundlegenden Resultate auch in anderen Werken zu finden sind.

Definition 3.1 (Integritätsbereich).

Sei R ein kommutativer Ring, so heißt R *Integritätsbereich*, falls R nullteilerfrei ist und $1 \neq 0$ in R.

Lemma 3.2. *Sei R ein Integritätsbereich. Dann ist $R[x]$, also der univariate Polynomring über R, ein Integritätsbereich und $R[x]^\times = R^\times$.*

Beweis. [5, Lemma 13.4]. □

Definition 3.3 (assoziierte Elemente).

Sei R ein Integritätsbereich. Zwei Elemente $r, s \in R$ heißen *assoziiert*, falls sie sich nur um eine Einheit unterscheiden, d.h. ein $u \in R^\times$ existiert mit $r = us$.

Bemerkung 3.4. Man sieht leicht ein, dass Assoziiertheit eine Äquivalenzrelation definiert.

Definition 3.5 (Hauptidealbereich).

Sei R ein Integritätsbereich, so heißt R *Hauptidealbereich* oder *Hauptidealring*, falls jedes Ideal $I \lhd R$ ein Hauptideal ist, d.h. ein $r \in R$ existiert mit $I = (r)$.

Lemma 3.6. *Sei K ein Körper, so ist K und $K[X]$ ein Hauptidealbereich.*

Beweis. [5, Satz 17.6]. □

Definition 3.7 (ggT und kgV).

Sei R ein Hauptidealbereich und $a, b \in R$. Dann heißt $t \in R$ mit

$$(a) + (b) \;=\; (a, b) \;=\; (t)$$

größter gemeinsamer Teiler von a und b; geschrieben $t = \mathrm{ggT}(a, b)$.
Ferner heißt $T \in R$ mit

$$(a) \cap (b) \;=\; (T)$$

kleinstes gemeinsames Vielfaches von a und b; geschrieben $T = \mathrm{kgV}(a, b)$.

Bemerkung 3.8. ggT und kgV sind nur bis auf Assoziiertheit gleich, wie man sich leicht überlegen kann.

Definition 3.9 (irreduzibles Element).

Sei R ein Integritätsbereich, so heißt $p \in R$ *irreduzibel*, falls $p \neq 0$, $p \notin R^\times$ und gilt:

$$\forall a, b \in R\colon\; p \mid ab \;\Rightarrow\; p \mid a \vee p \mid b\,.$$

Definition 3.10 (Faktorieller Ring).

Ein Integritätsbereich R heißt *faktorieller Ring*, falls jedes Element $0 \neq r \in R$ eine Zerlegung in irreduzible Faktoren besitzt, d.h.

$$r \;=\; e a_1 \dots a_n$$

mit $e \in R^\times$, $n \geq 0$ und $a_i \in R$ irreduzibel und diese Zerlegung eindeutig ist, d.h. sind

$$e a_1 \dots a_n \;=\; f b_1 \dots b_m$$

zwei Zerlegungen mit $e, f \in R^\times$, a_i, b_j irreduzibel, so folgt $n = m$ und $a_i = u_i b_{\pi(i)}$ für eine Permutation π von $\{1, \dots, n\}$ und Einheiten a_i für alle $i = 1, \dots, n$.

Satz 3.11.

Hauptidealbereiche sind faktorielle Ringe.

Beweis. [6, Theorem II.5.2]. □

Definition 3.12 (Modul).

Sei R ein kommutativer Ring, so ist ein *R-Modul* eine abelsche Gruppe $(M, +, 0)$ zusammen mit einer Abbildung

$$\cdot : R \times M \to M,\ (r, m) \mapsto r \cdot m\,,$$

sodass für alle $r, r' \in R$, m, m' im M gilt

(1) $r \cdot (r' \cdot m) = (rr') \cdot m$,

(2) $(r + r') \cdot m = r \cdot m + r' \cdot m$ und

(3) $r \cdot (m + m') = r \cdot m + r \cdot m'$.

Definition 3.13 (zyklischer Modul).

Ein R-Modul M heißt *zyklisch*, falls ein $x \in M$ existiert, so dass

$$M = xR\,.$$

Satz 3.14.

Untermoduln zyklischer Moduln sind wieder zyklisch.

Beweis. Die Aussage ist nicht anderes, als ein Spezialfall der Tatsache, dass Untermoduln freier Moduln wieder frei sind und die Dimension des Untermoduls nie größer ist als die des Moduls ([6, Theorem 7.1]). □

Definition 3.15 (Ordnung/Exponent eines Moduls).

Sei M ein R-Modul. Existiert ein $r \in R$ mit

$$rM = 0\,,$$

so heißt r *Ordnung von M* oder *Exponent von M*.

Nun können wir ein zentrales Resultat über Moduln mit Exponenten beweisen, das sich so auch beispielsweise in [4, Lemma 8.10] wiederfindet.

Satz 3.16 (Zerlegungssatz für Moduln mit Ordnung).

Seien R ein Hauptidealbereich und M ein nicht-trivialer R-Modul mit Exponent r, also $rM = 0$. Sei $r = ep_1^{\alpha_1} \dots p_k^{\alpha_k}$ eine Zerlegung in irreduzible Faktoren, sodass die p_i paarweise nicht assoziiert sind, so existieren eindeutig bestimmte R-Moduln $M_1, \dots, M_k$ mit $p_i^{\alpha_i} M_i = 0$ für alle $i = 1, \dots, k$, sodass

$$M = \bigoplus_{i=1}^{k} M_i\,.$$

Beweis. Zu Beginn stellen wir fest, dass die geforderte Zerlegung für r existiert und eindeutig ist, in dem wir (R ist faktoriell nach Satz 3.11) r in irreduzibele Elemente zerlegen und dann diejenigen, die zueinander assoziiert sind, zusammenfassen. Ferner notieren wir $d_i = \frac{r}{p_i^{\alpha_i}}$. Kümmern wir nun um die Eindeutigkeit. Sei also

$$M = M_1 \oplus \ldots \oplus M_k \quad \text{mit } p_i^{\alpha_i} M_i = 0$$

gegeben, so wollen wir zeigen, dass $M_i = d_i M$ und dadurch die Komponenten M_i eindeutig festgelegt sind. Es gilt offenbar für alle $i = 1, \ldots, k$

$$d_i M \subseteq d_i M_1 + \ldots + d_i M_k = d_i M_i \subseteq M_i\,,$$

denn $d_i M_j = 0$ für $i \neq j$ nach Voraussetzung. Wählen wir ein $i = 1, \ldots, k$ beliebig, so ist $(d_i, p_i^{\alpha_i}) = (1)$, d.h. es existieren $s, t \in R$ mit $sd_i + tp_i^{\alpha_i} = 1$. Für $m \in M_i$ folgt dann

$$m = 1m = (sd_i + tp_i^{\alpha_i})m = d_i(sm) \in d_i M\,.$$

Zusammen haben wir

$$M_i \subseteq d_i M \subseteq d_i M_i \subseteq M_i$$

und damit Gleichheit.

Um die Existenz zu zeigen, definieren wir einmal $M_i := d_i M$ und müssen nun die geforderten Eigenschaften nachprüfen. Zunächst ist klar, dass $p_i^{\alpha_i} M_i = 0$. Da $(d_1, \ldots, d_k) = (1)$ existeren $t_1, \ldots, t_k$ mit $\sum_{i=1}^k t_i d_i$ und für alle $x \in M$ folgt

$$x = 1x = \sum_{i=1}^{k} d_i(t_i x) \in \sum d_i M = \sum M_i\,.$$

Es fehlt nur noch zu zeigen, dass diese Summe auch direkt ist. Dazu sei wieder $i \in \{1, \ldots, k\}$ beliebig und $y \in \sum_{i \neq j} M_j$. Also ist $d_i y = 0$. Ist ferner zusätzlich $y \in M_i$, so ist $p_i^{\alpha_i} y = 0$. Wie oben existieren $s, t \in R$ mit $sd_i + tp_i^{\alpha_i} = 1$. Also

$$y = 1y = (s_d i + tp_i^{\alpha_i})y = 0\,,$$

was den Beweis abschließt. □

Definition 3.17.

Seien M ein R-Modul und $r \in R$, so definiere

$$V_r := \{a \in M : ra = 0\}\,.$$

Bemerkung 3.18. Es ist klar, dass V_r wieder zu einem R-Modul wird, da sich V_r auch lesen lässt, als der Kern des Modulhomomorphismus

$$M \to M,\ a \mapsto ra\,.$$

Satz 3.19.

Seien M ein Modul über einem Hauptidealbereich R und $r, s, t, T \in R$ mit $t = \mathrm{ggT}(r,s)$ und $T = \mathrm{kgV}(r,s)$. Dann gilt

(1) $V_r \cap V_s = V_t$.

(2) $V_r + V_s = V_T$.

Beweis. Zunächst ist klar, dass $V_r + V_s$ und $V_r \cap V_s$ wiederum R-Moduln sind.

(1) Sei $x \in V_t$, so ist $t \in \mathrm{Ann}_R\, x$ nach Definition des Annihilators. Dieser ist ein Ideal, also sind auch $s, t \in \mathrm{Ann}_R\, x$. Damit folgt sofort $x \in V_r \cap V_s$. Sei umgekehrt $x \in V_r \cap V_s$, also $rx = 0$ und $sx = 0$. Nach Definition des ggT existieren $r', s' \in R$ mit $t = r'r + s's$, also

$$tx = r'rx + s'sx = 0\,.$$

(2) Da r und s Teiler von T sind, ist klar, dass $V_r + V_s \subseteq V_T$. Sei umgekehrt $z \in V_T$. Schreibe nun $r = r't$, $s = s't$ und setze $x := s'z$, $y := r'z$. Dann ist

$$rx = rs'z = Tz = r'sz = sy$$

und wegen $Tz = 0$ folgt $x \in U_r$ und $y \in U_s$. Da nach Wahl nun $(r') + (s') = (1)$, existieren $\alpha, \beta \in R$ mit $\alpha r' + \beta s' = 1$ und wir folgern

$$z = \alpha r'z + \beta s'z = \alpha y + \beta x\,.$$

□

Definition 3.20 (Annihilator).

Sei M ein R-Modul. Für $S \subset M$ heißt

$$\mathrm{Ann}_R(S) := \{r \in R \mid sr = 0 \ \forall s \in S\}$$

der *Annihilator von S in R*. Für $S = \{x\}$ schreibe $\mathrm{Ann}_R(x) := \mathrm{Ann}_R(\{x\})$.

Bemerkung 3.21. In mancher Literatur wird auch die Schreibweise $M = (x)$ für einen zyklischen Modul benutzt, jedoch suggeriert (x) ein Ideal in R zu bezeichnen. Da dies im Allgemeinen nicht der Fall ist, verzichten wir auf diese Schreibweise.

Bemerkung 3.22. Man spricht in obiger Definition auch vom *Annihilator-Ideal*, da in der Tat $\mathrm{Ann}_R(S)$ ein Ideal in R ist. Insbesondere, falls $M = xR$ ein zyklischer R-Modul über einem Hauptidealbereich R ist, so ist

$$\mathrm{Ann}_R(x) = (r)$$

für ein $r \in R$.

Lemma 3.23. *Sei $M = xR$ ein zyklischer R-Modul. Dann gilt*

$$M \cong R/\mathrm{Ann}_R(x)$$

als R-Moduln und dieser Isomorphismus ist kanonisch.

Beweis. $\phi : R \to M,\ r \mapsto rx$ liefert einen surjektiven Homomorphismus von R-Moduln, dessen Kern gerade $\mathrm{Ann}_R(x)$ ist. Damit folgt die Behauptung sofort aus dem Homomorphiesatz für Moduln. □

Lemma 3.24. *Sei $Z = Rz$ ein zyklischer R-Modul von Ordnung p^α für ein Primelement $p \in R$. Dann sind die einzigen Teilmoduln von Z*

$$0 = Z_\alpha \subseteq Z_{\alpha-1} \subseteq \ldots \subseteq Z_0 = Z\,,$$

wobei $Z_\beta = p^\beta Z$.

Beweis. Nach Lemma 3.23 ist $Z \cong R/(p^\alpha)$. Damit stehen die Teilmoduln von Z in Bijektion zu den Teilmoduln von R (gelesen als R-Modul), die (p^α) enthalten. Solch ein Teilmoduln ist also gerade ein Ideal (r) mit $(p^\alpha) \subseteq (r)$. Da p prim ist, folgt $r = up^\beta$ für $0 \le \beta \le \alpha$ und $u \in R^\times$, wobei $u = 1$ oBdA angenommen werden kann. Damit sind die Z_β die einzigen Teilmoduln von Z. □

Lemma 3.25. *Sei $x \in M$ mit $\mathrm{Ann}_R(x) = (\lambda)$. Für $r \in R$ gilt*

$$\mathrm{Ann}_R(rx) = (\delta)$$

mit $\lambda = \delta t$ für $t = \mathrm{ggT}(r, \lambda)$.

Beweis. Schreibe $r = r't$, so ist $\delta rx = \delta r'tx = r'\lambda x = 0$. Also $\delta \in \mathrm{Ann}_R(rx)$. Für die andere Inklusion sei $s \in \mathrm{Ann}_R(rx)$, also $srx = 0$. Damit ist aber $sr \in \mathrm{Ann}_R(x)$ und $\lambda \mid sr$. Mit $\lambda = \delta t$ und $sr = sr't$ folgt für $t \neq 0$ (der Fall $t = 0$ ist ohnehin trivial) $\delta \mid sr'$. Nach Definition des ggT sind r' und δ teilerfremd, wodurch $\delta \mid s$. Also $s \in (\delta)$. □

Lemma 3.26. *Seien $x, y \in M$ mit $\mathrm{Ann}_R(x) = (a)$ und $\mathrm{Ann}_R(y) = (b)$. Sind a und b teilerfremd, so gilt $\mathrm{Ann}_R(x + y) = (ab)$.*

Beweis. Zunächst ist klar, dass $ab(x+y) = abx+aby = 0$, also $(ab) \subseteq \mathrm{Ann}_R(x+y)$. Ist nun $t(x+y) = 0$ für ein $t \in R$, so ist $z := tx = -ty \in (x) \cap (y)$. Ferner ist $(x) \cap (y) \subseteq V_a \cap V_b$ nach Voraussetzung. Nach Satz 3.19 (1) ist aber $V_a \cap V_b = (0)$. also auch $z = 0$. Damit ist

$$t \in \mathrm{Ann}_R(x) \cap \mathrm{Ann}_R(y) = (a) \cap (b) = = (\mathrm{kgV}(a, b)) = (ab)\,,$$

da a und b nach Voraussetzung teilerfremd sind. Also ist $\mathrm{Ann}_R(x + y) \subseteq (ab)$. □

3.2 | Vektorräume als Moduln

Definition 3.27 ((V, τ)).

Sei $\mathbb{K}$ ein Körper und V ein $\mathbb{K}$-Vektorraum und $\tau \in \mathrm{End}_\mathbb{K}(V)$, so können wir V als $\mathbb{K}[x]$-Modul auffassen:

$$f(x) \cdot v := f(\tau)(v)$$

für alle $f(x) \in \mathbb{K}[x]$ und $v \in V$. Nenne das Paar (V, τ) *$\mathbb{K}[x]$-Modul bzgl. τ*.

Notation 3.28. Sei $\tau \in \mathrm{End}_\mathbb{K}(V)$.

- Es bezeichne μ_τ das Minimalpolynom von τ, also das normierte Polynom kleinsten Grades $f(x) \in \mathbb{K}[x]$ mit $f(\tau) = 0$.
- Ferner schreibe χ_τ für das charakteristische Polynom von τ, also $\chi_\tau(x) := \det(x\,\mathrm{id}_V - \tau) \in \mathbb{K}[x]$. ▲

Bemerkung 3.29. Ist $\mathbb{K} = F := \mathbb{F}_q$ ein endlicher Körper, $V = E := \mathbb{F}_{q^n}$ eine Körpererweiterung von Grad n und

$$\begin{aligned} \tau = \sigma : \; E &\to E \\ v &\mapsto v^q \end{aligned}$$

der Frobenius von E, so ist

$$\mu_\tau(x) \;=\; \chi_\tau(x) \;=\; x^n - 1\,,$$

denn: Es ist klar, dass $n = \deg \chi_\tau$ und da nach dem Satz von Cayley-Hamilton ist σ Nullstelle von χ_τ. Daher teilt μ_τ das charakteristische Polynom. Jedoch kennen wir das Minimalpolynom von τ: Nach dem Dedekindschen Lemma (Satz 1.8) ist $\mathrm{id}_E, \sigma, \ldots, \sigma^{n-1}$ linear unabhänig über E, also insbesondere über F, und $\sigma^n = \mathrm{id}_E$.

Definition 3.30 (τ-Ordnung, Teilmodul).

Sei (V, τ) ein $\mathbb{K}[x]$-Modul. Zu jedem $v \in V$ betrachte den $\mathbb{K}[x]$-Modulhomomorphismus

$$\begin{aligned} \psi_v : \; \mathbb{K}[x] &\to V \\ f(x) &\mapsto f(x) \cdot v \end{aligned}$$

Sei ferner $\dim V < \infty$.

(1) Ist $\ker \psi_v = (g(x))$ für $g(x) \in \mathbb{K}[x]$ normiert, so heißt $g(x)$ *τ-Ordnung von v*. Ferner ist $g(x)$ eindeutig. Schreibe $\mathrm{Ord}_\tau(v) := g(x)$.

(2) $\mathbb{K}[\tau] \cdot v := \mathrm{im}\, \psi_v$ heißt der von v *erzeugte $\mathbb{K}[x]$-Teilmodul von V*.

Bemerkung 3.31. Die Eindeutigkeit der τ-Ordnung wird sofort klar, wenn man sich überlegt, dass für einen Hauptidealbereich R mit $(a) = (b) \subseteq R$ gilt: Schreibe $a = bb'$ und $b = aa'$ so gilt $a = aa'b'$, also $a'b' = 1$. Damit unterscheiden sich die Erzeuger zweier gleicher Hauptideale lediglich um eine Einheit. Wir haben jedoch $K[x]^\times = \mathbb{K}^\times$. Die Forderung nach Normiertheit klärt damit abschließend die Eindeutigkeit.

Bemerkung 3.32. In Notation von Abschnitt 3.1 gilt offensichtlich

$$\mathrm{Ann}_{\mathbb{K}[x]}(v) \;=\; \ker \psi_v\,.$$

Notation 3.33. Für $\mathbb{K} = \mathbb{F}_q$ einen endlichen Körper, $V = E \mid \mathbb{F}_q$ eine Körpererweiterung und $\tau = \sigma$ den Frobenius-Endomorphismus schreibe

$$\mathrm{Ord}_q := \mathrm{Ord}_\tau$$

und bezeichne Ord_q mit *q-Ordnung*. ▲

Lemma 3.34. *Sei (V,τ) ein $\mathbb{K}[x]$-Modul. Ferner seien $u,v \in V$ mit $g(x) := \mathrm{Ord}_\tau(u)$, $h(x) := \mathrm{Ord}_\tau(v)$ und $f(x) \in \mathbb{K}[x]$. Dann gilt*

(1) $\mathrm{Ord}_\tau(f(x)\cdot u) = \frac{g(x)}{\mathrm{ggT}(f(x),g(x))}$.

(2) $\mathrm{Ord}_\tau(u+v) = g(x)h(x)$, *falls* $\mathrm{ggT}(g,h) = 1$.

Beweis. (1) Dies ist lediglich eine Umformulierung von Lemma 3.25.

(2) In Lemma 3.26 haben wir gesehen, dass in diesem Fall $\mathrm{Ann}_{\mathbb{K}[x]}(u+v) = (g(x)h(x))$ gilt. □

Lemma 3.35. *Sei (V,τ) ein $\mathbb{K}[x]$-Modul. Sei $v \in V$. Dann gilt:*

$$\dim_{\mathbb{K}}(\mathbb{K}[x]\cdot v) \;=\; \deg(\mathrm{Ord}_\tau(v))\,.$$

Beweis. Nach dem Homomorphiesatz gilt: $\operatorname{im}\psi_v \cong \mathbb{K}[x]/\ker\psi_v$ als $\mathbb{K}[x]$-Moduln. □

Definition 3.36 (zyklischer $\mathbb{K}[x]$-Modul).

(V,τ) heißt *zyklischer $\mathbb{K}[x]$-Modul bzgl. w*, falls es ein $w \in \mathbb{K}$ gibt, sodass $\mathbb{K}[\tau]\cdot w = V$.

Satz 3.37.

Es gilt:

$$(V,\tau) \text{ ist ein zyklischer Modul} \quad\Leftrightarrow\quad \mu_\tau = \chi_\tau$$

Beweis. Fassen wir zunächst ein paar einfache Tatsachen zusammen: Ist $v \in V$, so haben wir

$$\dim(\mathbb{K}[x]\cdot v) = \deg(\mathrm{Ord}_\tau(v)) \;\le\; \deg\mu_\tau \;\le\; \deg\chi_\tau$$

und

$$\mathrm{Ord}_\tau(v) \;\mid\; \mu_\tau \;\mid\; \chi_\tau\,,$$

wobei die erste Teilbarkeitsrelation per definitionem erfüllt ist und die zweite gerade der Satz von Cayley-Hamilton ist. Damit kommen wir zum direkten Beweis:

„⇒“ Sei V also zyklisch bzgl. w, so ist dies nach obigem äquivalent zu $\deg(\mathrm{Ord}_\tau(w)) = n$. Daraus folgt aber sofort $\mu_t = \chi_\tau$, da beide normiert sind.

„⇐“ Zunächst sei behauptet, dass es stets ein $w \in V$ gibt mit $\mathrm{Ord}_\tau(w) = \mu_\tau$. Sei dazu $\mu_\tau(x) = \prod_{i=1}^r p_i(x)^{a_i}$ die Zerlegung in irreduzible Faktoren über $\mathbb{K}[x]$, so existieren $w_i \in V$ mit $\mathrm{Ord}_\tau(w_i) = p_i^{a_i}$. Andernfalls hätten wir einen Widerspruch zum Minimalpolynom von τ! Nach Satz 3.34 ist dann aber $w := \sum_{i=1}^r w_i$ ein Element in V mit τ-Ordnung μ_τ.

Ist dann also $\mu_\tau = \chi_\tau$, so hat obiges w genau τ-Ordnung χ_τ; erzeugt also V als $\mathbb{K}[x]$-Modul.

□

Satz 3.38.

Sei (V,τ) ein zyklischer Modul über einem endlich dimensionalem Vektorraum. Sei ferner $g(x) \in \mathbb{K}[x]$ normiert mit $g \mid \mu_\tau$. Dann gilt:

(1) V_g (siehe Definition 3.17) ist ein $\mathbb{K}[x]$-Teilmodul von V.

(2) Alle $\mathbb{K}[x]$-Teilmoduln von V sind von dieser Form.

(3) Die Erzeuger von V_g sind genau die Elemente $v \in V$ mit $\mathrm{Ord}_\tau(v) = g$, d.h. für diese gilt $\mathbb{K}[x] \cdot v = V_g$. Insbesondere sind die Erzeuger von V gerade die Elemente $u \in V$ mit $\mathrm{Ord}_\tau(u) = \mu_\tau$.

(4) V_g ist zyklisch bzgl. τ mit Minimalpolynom $g(x)$. Ferner ist $\dim(V_g) = \deg(g)$.

Beweis. (1) Klar: $0 \in V_g$. Weiter seien $f(x) \in \mathbb{K}[x]$ und $v \in V_g$ mit $h(x) := \mathrm{Ord}_\tau(v)$, so ist nach Lemma 3.34 $\mathrm{Ord}_\tau(f(x)\cdot v) = \frac{h(x)}{\mathrm{ggT}(f,h)} \mid g(x)$. Damit liegt auch $f(x)\cdot v$ in V_g.

(2) Dies ist lediglich eine Umformulierung von Satz 3.16 und Lemma 3.24.

(3) Sei $v \in V$ ein Erzeuger von V_g, so ist per definitionem von $g(x)\cdot v = 0$. Also $\mathrm{Ord}_\tau(v) \mid g(x)$. Schreibe $\mathrm{Ord}_\tau(v) =: h(x)$, so folgt

$$h(x)\cdot V_g \;=\; \mathbb{K}[x]\cdot(h(x)\cdot v) \;=\; 0 \quad\Leftrightarrow\quad g(x) \mid h(x)\,.$$

Ist andererseits $w \in V$ mit $\mathrm{Ord}_\tau(w) = g(x)$, so können wir zunächst festhalten, dass $\mathbb{K}[x]\cdot w \subseteq V_g$. Sei ferne $x \in V$ ein Erzeuger von V_g. Dann ist $w = f(x)\cdot v$ für ein $f(x) \in \mathbb{K}[x]$ und ferner $\mathrm{ggT}(f(x), g(x)) = 1$ nach Lemma 3.34. Also existieren $f'(x), g'(x) \in \mathbb{K}[x]$ mit $f'f + g'g = 1$. Wir folgern:

$$v \;=\; (f'(x)f(x) + g'(x)g(x))\cdot v \;=\; f'(x)\cdot w\,,$$

d.h. $v \in \mathbb{K}[x]\cdot w$; mithin $V_g \subseteq \mathbb{K}[x]\cdot w$.

(4) Klar nach (3) und Lemma 3.35. □

Bemerkung 3.39. Die Punkte (1) und (2) hätten bereits in Abschnitt 3.1 aufgeführt werden können, da die Zyklizität des Moduls hier ausreichend war. In [3, Theorem 7.10] ist dies auch so vorzufinden.

Korollar 3.40. *Sei $v \in V$ mit $\mathrm{Ord}_\tau(v) = g(x)$. Für $w \in V$ gilt dann:*

$$w \in V_g \quad\Leftrightarrow\quad w = f(x)\cdot v \text{ für ein } f(x) \in \mathbb{K}[x]_{<\deg g}$$

Beweis. Nach Satz 3.38 ist v Erzeuger von V_g, d.h. $V_g = \mathrm{im}\,\psi_v \cong \mathbb{K}[x]/(g(x))$, wobei letztere Isomorphie nach dem Homomorphiesatz gilt. Dies zeigt die Behauptung. □

Wir schließen dieses Kapitel mit einer wesentlichen Beobachtung, über den Zusammenhang von Zerlegungen im Ring $\mathbb{K}[x]$ und im Vektorraum V. Für den Spezialfall von zyklischen Galoiserweiterung findet man den Satz auch in [3, Theorem 8.6] wieder.

Definition 3.41 (Zerlegung).

Sei $f(x) \in \mathbb{K}[x]$ normiert mit $\deg f \geq 1$, so heißt $\Delta \subseteq \mathbb{K}[x]$ *Zerlegung von $f(x)$*, falls gilt: Alle $\delta \in \Delta$ sind normiert, vom Grad größer gleich 1, paarweise teilerfremd und es gilt: $f(x) = \prod_{\delta\in\Delta} \delta(x)$.

Satz 3.42.

Sei (V,τ) ein zyklischer Moduln von endlicher Dimension. Seien ferner $g(x) \in \mathbb{K}[x]$ normiert mit $g \mid \mu_\tau$ und Δ eine Zerlegung von g. Dann gilt:

(1) $V_g = \oplus_{\delta\in\Delta} V_\delta$ ist eine direkte Summe von zyklischen Moduln bzgl. τ.

(2) Jedes $w \in V_g$ lässt sich eindeutig schreiben als $w = \sum_{\delta\in\Delta} w_\delta$ mit $w_\delta \in V_\delta$. Ferner gilt

$$\mathrm{Ord}_\tau(w) \;=\; \prod_{\delta\in\Delta} \mathrm{Ord}_\tau(w_\delta)$$

und $\mathrm{Ord}_\tau(w)$ ist ein normierter Teiler von $g(x)$.

(3) w ist ein Erzeuger von V_g genau dann, wenn für alle $\delta \in \Delta$ auch w_δ Erzeuger von V_δ ist.

(4) Ist $V_g = \oplus_{i\in I} V_i$ eine Zerlegung in Teilmoduln, so existieren eine Zerlgung Δ von g und eine Bijektion $\pi : I \to \Delta$, so dass $V_i = V_{\pi(i)}$.

Beweis. (1) Betrachten wir V_g selbst als $\mathbb{K}[x]$-Modul, so ist diese Aussage lediglich eine Anwendung von Satz 3.16.

(2) Dass die geforderte Zerlegung in w_δ existiert und eindeutig ist, ist eine direkte Konsequenz aus (1). Da w von $g(x)$ annihiliert wird, ist klar, dass $\mathrm{Ord}_\tau(w) \mid g$. Analog gilt auch $\mathrm{Ord}_\tau(w_\delta) \mid \delta$ für alle $\delta \in \Delta$. Diese sind jedoch teilerfremd und wir erhalten die postulierte Gleichung aus Lemma 3.34 (2).

(3) Dies ist mit (2) und Satz 3.38 (3) sofort klar.

(4) Nach Satz 3.38 (2) sind alle $V_i = V_{\pi(i)}$ für einen normierterten Teiler $\pi(i) \mid g$. Da die Summe direkt ist, folgt die Behauptung mit Satz 3.19. □

Kapitel 4
Explizite Konstruktion von Normalbasen

Wir haben nun kennengelernt, wie man einen Vektorraum V über $\mathbb{K}$ zusammen mit einem Endomorphismus τ als $\mathbb{K}[x]$-Modul lesen kann. Analog dazu wollen wir nun eine Erweiterung endlicher Körper E über F in diesem Kontext verstehen: E ist ein Vektorraum über F und wird mit Hilfe des Frobenius $\sigma : \bar{F} \to \bar{F}$ zu einem $F[x]$-Modul im Sinne von Definition 3.27. Dies wird uns helfen, normale (und später vollständig normale) Elemente zu konstruieren.

4.1 | Grundlegende Ideen

Seien im Folgenden stets $F := \mathbb{F}_q$ ein endlicher Körper von Charakteristik p und $E := \mathbb{F}_{q^n} \mid F$ eine Körpererweiterung. Wir wiederholen kurz die Definition einer *Normalbasis*.

Definition 4.1 (normales Element, normales Polynom, Normalbasis).

Sei F ein Körper und $E \mid F$ eine endliche Galoiserweiterung von Grad n. Sei ferner $w \in E$ mit $F(w) = E$. w heißt *normal über* F, falls

$$\{\gamma(w) \mid \gamma \in \mathrm{Gal}(E \mid F)\}$$

eine F-Basis von E ist. $\{\gamma(w) \mid \gamma \in \mathrm{Gal}(E \mid F)\}$ heißt entsprechend *Normalbasis* und $g(x) \in F[x]$ mit

$$g(x) = \prod_{\gamma \in \mathrm{Gal}(E|F)} (x - \gamma(w))$$

heißt *normales Polynom*.

Wir wollen nun die Begriffe der normalen Elemente und der Erzeuger von zyklischen Moduln aus vorhergehendem Kapitel in Zusammenhang bringen. Dazu müssen wir uns Gedanken machen, wie sich Normalität im Kontext der Modulstruktur einer Körpererweiterung lesen lässt.

Satz 4.2.

Ein Element $u \in E$ ist genau dann normal über F, wenn

$$\mathrm{Ord}_q(u) \;=\; x^n - 1 \quad \in F[x]\,.$$

Beweis. Nach Bemerkung 3.29 ist das Minimalpolynom des Frobenius von F gerade $x^n - 1$. Ferner wissen wir nach Satz 1.7, dass $\mathrm{Gal}(E \mid F) = \langle \sigma \rangle$. Letztlich liefert Satz 3.38 (3), dass die Erzeuger von E als $F[x]$-Modul, also gerade die normalen Elemente, jene mit q-Ordnung $x^n - 1$ sind. □

Korollar 4.3. *Sei $x^n - 1 = \prod_{i=1}^{s} r_i(x)$ eine Zerlegung in paarweise teilerfremde Polynome. Seien ferner $u_i \in V_{r_i}$ Elemente mit $\mathrm{Ord}_q(u_i) = r_i(x)$ $\forall i = 1, \ldots, s$. Dann ist*

$$u \;=\; u_1 + u_2 + \ldots + u_s$$

normal in $E \mid F$.

Beweis. Satz 3.42. □

4.2 | Stark reguläre Erweiterungen

Beispiel 4.4. Wählen wir einmal $q = 7$ und $n = 9$. Also $F = \mathbb{F}_7$. Ferner wissen wir aus Kapitel 2, dass

$$x^n - 1 \;=\; x^9 - 1 \;=\; \Phi_1(x)\Phi_3(x)\Phi_9(x) \;=\; \big(x-1\big)\big((x+3)(x+5)\big)\big((x^3+3)(x^3+5)\big) \;\in \mathbb{F}_7[x]$$

die vollständige Faktorisierung von $x^n - 1$ über $\mathbb{F}_7$ ist. Da es sich hier bei den Faktoren lediglich um Binome handelt, können wir relativ einfach Elemente passender q-Ordnungen angeben: Beginnen wir mit $\Phi_9 = (x^3+3)(x^3+5)$. Gesucht ist nun ein Element u in einer passenden Erweiterung von F mit $\mathrm{Ord}_q(u) = \Phi_9$. Sei dazu $u \in \bar{F}$ eine primitive 27-te Einheitswurzel. Dann gilt:

$$\mathrm{ord}(u^{342}) \;=\; \mathrm{ord}(u^{18}) \;=\; \frac{27}{\mathrm{ggT}(27,18)} \;=\; 3$$

Wenn man sich die Frage stellt, warum an diesem Punkt gerade 342 eine interessante Zahl ist, so wird man diese sofort wiederfinden, wenn man versucht ein Element $w \in \bar{F}^*$ mit q-Ordnung $x^3 + 3$ (bzw. $x^3 + 5$) wiederzufinden:

$$(x^3+3)\cdot w \;=\; w^{q^3} + 3w \;=\; w(w^{7^3-1} + 3) \;=\; w(w^{342} + 3) \;\overset{!}{=}\; 0.$$

Da $w \neq 0$ und per definitionem der Kreisteilungspolynome $(-3), (-5) \in \mathbb{F}_7$ primitive 3-te Einheitswurzeln sind, brauchen wir ein w mit $\mathrm{ord}(w^{342}) = 3$; was obiges u gerade erfüllt! Damit ist also $u^{18} = u^{342}$ eine der beiden dritten Einheitswurzeln (-3) oder $(-5) \in \mathbb{F}_7$ und wir können mit Lemma 3.34 folgern:

$$\mathrm{Ord}_q(u + u^2) \;=\; \Phi_9$$

Auch die Suche nach einem Element mit q-Ordnung Φ_3 ist damit erledigt: Mit analoger Argumentation wie oben erhalten wir, dass

$$\mathrm{Ord}_q(u^3 + u^6) \;=\; \Phi_3.$$

Zusammengefasst ist also u ein normales Element von $\mathbb{F}_{7^9} \mid \mathbb{F}_7$. ▲

Ein entscheidender Vorteil in der Konstruktion eines normalen Elements in obigem Beispiel war der Zerfall der Kreisteilungspolynome in Binome. Aus Satz 2.18 wissen wir bereits, wann die Kreisteilungspolynome in Binome zerfallen. Damit können wir diese sehr einfache Möglichkeit Normalbasen explizit anzugeben, als Reihe von Aussagen formulieren:

Definition 4.5 (stark regulär).

Das Paar $(q,n) \in \mathbb{N}^2$ heißt *stark regulär*, falls

- $q = p^r$ mit p einer Primzahl und $r > 0$,
- $p \nmid n$,
- $\nu(n) \mid q-1$,
- $4 \mid q-1$, falls n gerade.

Schreibe $n \in \mathcal{S}_q$, falls (q,n) stark regulär. Eine Körpererweiterung $\mathbb{F}_{q^n} \mid \mathbb{F}_q$ heißt *stark regulär*, falls $n \in \mathcal{S}_q$.

Satz 4.6.

Sei $F = \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Seien $l := \pi_m(q-1)$, $a := \mathrm{ggT}(l,m)$ und $u \in \bar{\mathbb{F}}_q$ eine primitive (ml)-te Einheitswurzel. Dann gilt:

(1) $\mathbb{F}_q(u) = \mathbb{F}_{q^m} =: E$.

(2) $\mathrm{Ord}_q(u)$ *ist ein irreduziblerTeiler von* Φ_m *in* $\mathbb{F}_q[x]$.

(3) $v := \sum_{i \in I_a} u^i$ *hat q-Ordnung* Φ_m.

Beweis. (1) Aus Lemma 2.14 haben wir $\pi_m(q^m-1) = m\pi_m(q-1) = ml$ und damit $[F(u):F] = \mathrm{ord}_{ml}(q) = m$.

(2) Sei nun $\zeta \in F^*$ eine primitive a-te Einheitswurzel. Dann ist nach Satz 2.18

$$\Phi_m(x) = \prod_{i \in I_a} (x^{\frac{m}{a}} - \zeta^i).$$

Wie in Beispiel 4.4 betrachten wir für $i \in I_a$:

$$\begin{aligned} (x^{\frac{m}{a}} - \zeta^i) \cdot u &= \left(\sigma^{\frac{m}{a}} - \zeta^i \,\mathrm{id}_E\right) \\ &= u^{q^{\frac{m}{a}}} - \zeta^i u \\ &= u\left(u^{q^{\frac{m}{a}}-1} - \zeta^i\right), \qquad (*) \end{aligned}$$

wobei wie üblich $\sigma : E \to E, x \mapsto x^q$ den Frobenius von $E \mid F$ meint. Wir wollen nun zeigen, dass $\mathrm{Ord}_q(u) = (x^{\frac{m}{a}} - \zeta^i)$ für ein $i \in I_a$, explizit also, dass $(*) = 0$ für ein $i \in I_a$ gilt. Dazu müssen wir $q^{\frac{m}{a}} - 1$ genauer untersuchen: Wiederum nach Lemma 2.14 haben wir:

$$\pi_m(q^{\frac{m}{a}} - 1) = \frac{m}{a}\pi_m(q-1) = \frac{ml}{a} = \mathrm{kgV}(m,l) =: c\,.$$

Nun ist $q^{\frac{m}{a}} - 1 = ck$ für ein $k \in \mathbb{N}$ mit $\operatorname{ggT}(k, ml) = 1$, wie man sich anhand der Primfaktorzerlegungen leicht klar machen kann. Erinnern wir uns kurz, dass u eine primitive (ml)-te Einheitswurzel war, so folgern wir:

$$\operatorname{ord}(u^{ck}) \;=\; \operatorname{ord}(u^c) \;=\; \frac{ml}{\operatorname{ggT}(ml, c)} \;=\; \frac{ml}{c} \;=\; a\,.$$

Ergo gibt es $j \in I_a$ mit $u^{ck} = \zeta^j$ und für dieses j ist $(*)$ gerade 0, was zu zeigen war.

(3) Seien $j \in I_a$ mit $\operatorname{Ord}_q(u) = x^{\frac{m}{a}} - \zeta^j$ und $i \in I_a$ beliebig, so gilt

$$\begin{aligned}\left(x^{\frac{m}{a}} - \zeta^{ji}\right)u^i \;&=\; (u^{q^{\frac{m}{a}}})^i - (\zeta^j u)^i \\ &=\; (u^{q^{\frac{m}{a}}} - \zeta^j u) \sum_{k=0}^{i-1} u^{k\,q^{\frac{m}{a}}}\, \zeta^{(i-1-k)\,j}\, u^{i-1-k} \\ &=\; 0\,.\end{aligned}$$

Da i teilerfremd zu ml ist, haben wir ferner $\operatorname{ord}(u) = \operatorname{ord}(u^i)$ und können folgern, dass $\operatorname{Ord}_q(u^i) = x^{\frac{m}{a}} - \zeta^{ji}$. Packen wir nun alles zusammen und bedenken, dass $I_a \to I_a, i \mapsto ij$ eine Bijektion ist, können wir den letzten Schritt im Beweis führen:

$$\operatorname{Ord}_q\Big(\sum_{i \in I_a} u^i\Big) \;=\; \prod_{i \in I_a} \operatorname{Ord}_q(u^i) \;=\; \prod_{i \in I_a} \left(x^{\frac{m}{a}} - \zeta^{ji}\right) \;=\; \prod_{k \in K_a} \left(x^{\frac{m}{a}} - \zeta^k\right) \;=\; \Phi_m(x)\,.$$

□

Da wir nun die einzelnen Bausteine kennen, können wir auf die Faktorisierung und damit auf eine explizite Angabe eines normalen Elements zu schließen:

Korollar 4.7. *Für $F = \mathbb{F}_q$, $n \in \mathcal{S}_q$ sei $\lambda \in F^*$ eine primitive $\pi_n(q-1)$-te Einheitswurzel. Zu jedem $m \mid n$ seien*

- $a(q, m) := \operatorname{ggT}(m, \pi_m(q-1))$ *und*
- $I_a := \{i \le a : \operatorname{ggT}(i, a) = 1\}$.

Dann ist

$$x^n - 1 \;=\; \prod_{m \mid n} \prod_{i \in I_{a(q,m)}} \left(x^{\frac{m}{a(q,m)}} - \lambda^{\frac{\pi_n(q-1)}{a(q,m)} i}\right)$$

die vollständige Faktorisierung von $x^n - 1$ über $\mathbb{F}_q$.

Beweis. Da $\operatorname{ord}\left(\lambda^{\frac{\pi_n(q-1)}{a(q,m}}\right) = a(q, m)$ und für $m \mid n$ offensichtlich auch $m \in \mathcal{S}_q$, ist obige Aussage lediglich eine Anwendung von Satz 4.6. □

Satz 4.8.

Seien $F = \mathbb{F}_q$ ein endlicher Körper, $n \in \mathcal{S}_q$ und $L := \pi_n(q-1)$. Ferner sei $u \in \bar{F}$ eine primitive (nL)-te Einheitswurzel. Dann ist mit Notation aus Korollar 4.7

$$w \;:=\; \sum_{m \mid n} \sum_{i \in I_a} u^{\frac{n\,L}{m\,\pi_m(q-1)} i}$$

normal in $E := \mathbb{F}_{q^n}$ über F.

Beweis. Im Grunde haben wir bereits alles gezeigt. Daher reicht ein kurzer Kommentar, warum wir Satz 4.6 anwenden können aus. Es ist trivialerweise

$$\operatorname{ord}\left(u^{\frac{nL}{m\pi_m(q-1)}}\right) \;=\; m\,\pi_m(q-1)$$

und damit sind alle Voraussetzungen erfüllt. □

Bemerkung 4.9. Wie wir in Beispiel 4.4 und Satz 4.6 gesehen haben, sind die (ml)-ten Einheitswurzeln die Elemente von kleinster multiplikativer Ordnung, deren q-Ordnung ein irreduzibler Teiler von $\Phi_m(x)$ wird. Natürlich können wir dieses Konzept auch erweitern und uns überlegen, welche primitiven Einheitswurzeln die selbe Eigenschaft erfüllen. Darüber hinaus können wir die Elemente deren q-Ordnung ein irreduzibler Teiler von $\Phi_m(x)$ über $F[x]$ ist, auch durch die Modulstruktur selbst beschreiben. Diese beiden Überlegungen wollen wir in den nächsten beiden Lemmas beweisen.

Lemma 4.10. *Seien die Voraussetzungen wie in Satz 4.6, also $F = \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Setze ferner $l := \pi_m(q-1)$, $a := \operatorname{ggT}(l,m)$. Ist nun θ eine primitive (nf)-te Einheitswurzel für $l \mid f \mid q-1$, so ist $\operatorname{Ord}_q(\theta)$ ein irreduzibler Teiler von $\Phi_m(x)$ über $F[x]$.*

Beweis. Sei $f = le$ mit $\operatorname{ggT}(e,l) = 1$. Diese Zerlegung ist möglich, da l per definitionem der größte Teiler von $q-1$ ist, dessen Primfaktoren allesamt in m vorkommen, d.h. jeder Primfaktor von $q-1$ der l teilt, kommt dort bereits in maximaler Potenz vor. Damit reicht es – wenn wir wir den Beweis von Satz 4.6 noch einmal nachvollziehen – zu zeigen, dass $\operatorname{ggT}(mle, ck) = ce$ für $c := \operatorname{kgV}(m,l)$ und $q^{\frac{m}{a}} - 1 = ck$ für ein k mit $\operatorname{ggT}(k, ml) = 1$. Da $\operatorname{ggT}(e,l) = 1$, ist per definitionem von l auch $\operatorname{ggT}(e,m) = 1$ und damit auch $\operatorname{ggT}(e,c) = 1$. Da $e \mid q-1 \mid q^{\frac{m}{a}-1}$ zerfällt k in $k = \bar{k}k_e$ mit $e \mid k_e$ und $\operatorname{ggT}(\bar{k}, e) = 1$. Damit folgt $\operatorname{ggT}(mle, ck) = ce$ und wir haben $\operatorname{ord}(\theta^{ck}) = \frac{mle}{\operatorname{ggT}(mle,ck)} = \frac{mle}{ce} = a$. □

Als Korollar dieses Lemmas können wir einen Satz von Semaev 1989 in [11] beweisen, welcher erneut von Blake, Gao und Mullin 1997 in [1] bewiesen wurde:

Satz 4.11 ([1, Theorem 2.7], [11]).

Sei $q = p^r$ eine Primzahlpotenz. Sei $n \in \mathbb{N}$ mit $\operatorname{ggT}(n,q) = 1$. Ist $x^n - a \in \mathbb{F}_q[x]$ irreduzibel mit Wurzel $\theta \in \mathbb{F}_{q^n}$, so gilt

$$\mathbb{F}_{q^n} \;=\; \bigoplus_{l \in R_q(n)} \langle \theta^l \rangle_q \,,$$

wobei $\langle \theta^l \rangle_q := \operatorname{span}_{\mathbb{F}_q}\{\theta^i \mid i \in M_q(l \bmod n)\}$.

Beweis. Aus Satz 2.16 und Satz 2.17 wissen wir, dass für $a \in \mathbb{F}_q^*$ das Polynom $x^n - a \in \mathbb{F}_q[x]$ genau dann irreduzibel ist, wenn $p \nmid n$, $\nu(n) \mid f := \operatorname{ord}(a)$, $l := \pi_n(q-1) \mid f$ und $q \equiv 1 \bmod 4$, falls $4 \mid n$. Ist bereits $q \equiv 1 \bmod 4$, falls n gerade, so sind wir genau in der Situation, dass n stark regulär ist. Damit ist θ eine primitive (nf)-te Einheitswurzel und wir wissen nach Lemma 4.10, dass $\operatorname{Ord}_q(\theta)$ ein irreduzibler Teiler von $\Phi_n(x)$ über $\mathbb{F}_q[x]$ ist, womit alles gezeigt wäre.

Es bleibt ein Wort zur Situation $q \equiv 3 \bmod 4$, falls n gerade, zu verlieren. Dieser Fall wird von den bisherigen Resultaten nicht erfasst. Mit etwas mehr Aufwand ist es jedoch möglich, die explizite Konstruktion von Normalbasen mit primitiven Einheitswurzeln zu erweitern, wie Hachenberger in [3, Section 22] zeigt. □

Bevor wir die Ideen der stark regulären Körpererweiterungen verallgemeinern wollen, betrachten wir noch ein Lemma, das die irreduziblen Teilmodule genauer beschreibt und in ganz ähnlicher Form in [3, Theorem 22.5] wiederzufinden ist.

Lemma 4.12. *Seien die Voraussetzungen wie in Satz 4.6, also $F = \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Setze ferner $l := \pi_m(q-1)$, $a := \mathrm{ggT}(l,m)$. Ist dann u eine primitive (ml)-te Einheitswurzel mit $\mathrm{Ord}_q(u) = f(x)$ für $f(x)$ einen irreduziblen monischen Teiler von $\Phi_m(x)$, so gilt für $v \in E := \mathbb{F}_{q^m}$:*

$$\mathrm{Ord}_q(v) = f(x) \quad \Leftrightarrow \quad v = g(x)\cdot u \quad \text{für ein } 0 \neq g \in F[x]_{<\frac{m}{a}}$$

Beweis. In Korollar 3.40 haben wir gezeigt, dass für $v \in E$ gilt

$$v \in E_f \quad \Leftrightarrow \quad v = g(x)\cdot u \text{ für ein } g(x) \in F[x]_{<\deg f},\ \mathrm{ggT}(f,g) = 1$$

Da f irreduzibel von Grad $\frac{m}{a}$ ist, sind alle $v \in E_f \setminus \{0\}$ von q-Ordnung f und es folgt die Behauptung. □

Bemerkung 4.13. Obiges Lemma gilt nicht nur für primitive (ml)-te Einheitswurzeln, sondern – wie man offensichtlich erkennen kann – für jedes Element mit gleicher q-Ordnung!

4.3 | Reguläre Erweiterungen

Die Erkenntnisse über stark reguläre Erweiterungen wollen wir nutzen, um Normalbasen auch in einem allgemeineren Kontext angeben zu können. Ist nämlich $m \in \mathbb{N}$ ungerade mit $p \nmid m$ und setzen wir $s = \mathrm{ord}_{\nu(m)}(q)$, so erkennen wir, dass $\nu(m) \mid q^s - 1$ per definitionem von $\mathrm{ord}_{\nu(m)}(q)$. Mit anderen Worten: $m \in \mathcal{S}_{q^s}$!

Definition 4.14 (regulär).

Sei $q = p^r$ eine Primzahlpotenz und $n \in \mathbb{N}_{>0}$. Setze $n = n'p^c$ mit $p \nmid n'$. Das Paar (q,n) heißt *regulär*, falls $\mathrm{ggT}(n, \mathrm{ord}_{\nu(n')}(q)) = 1$. Ist $F = \mathbb{F}_q$ und $E = \mathbb{F}_{q^n}$, so nenne die Erweiterung $E \mid F$ *regulär*.

Satz 4.15.

Seien $F := \mathbb{F}_q$ ein endlicher Körper für eine Primzahlpotenz $q = p^r$ und $m \in \mathbb{N}$ ungerade mit $p \nmid m$. Setze $s := \mathrm{ord}_{\nu(m)}(q)$, $l := \pi_m(q^s - 1)$, $b := b(q,m) = \mathrm{ggT}(l,m)$, $E := \mathbb{F}_{q^m}$ und $E' = \mathbb{F}_{q^{sm}}$. Ist dann u eine primitive (ml)-te Einheitswurzel, so gilt

(1) $F(u) = E'$,

(2) $\mathrm{Ord}_q(u) = f(x^s)$ *für einen monischen irreduziblen Teiler* $f(x) \in F[x]$ *von* $\Phi_m(x)$ *und*

(3) $v := \sum_{\substack{j \in R_q(m) \\ \mathrm{ggT}(j,m)=1}} u^j$ *hat q-Ordnung* $\Phi_m(x^s)$.

Beweis. **TODO**

Nun haben wir also Elemente mit q-Ordnung $\Phi_m(x^s)$ gefunden. Es stellt sich jedoch die Frage, wir wir daraus Elemente mit q-Ordnung $\Phi_m(x)$ „basteln“ können. Dies zeigt uns der folgende Satz:

Satz 4.16.

Seien alle Voraussetzungen und Notationen wie in Satz 4.15. Sei jedoch zusätzlich $\mathrm{ggT}(m,s) = 1$. *Bezeichne ferner* $\sigma_E : E' \to E',\ x \mapsto x^{q^m}$ *den Frobenius von* E' *auf* E. *Ist dann* u *eine primitive* (ml)*-te Einheitswurzel und* $v = \sum_{j \in R_q(m)} u^j$, *so gilt:*

(1) $H(\sigma_E)(v)$ *hat q-Ordnung* $\Phi_m(x)$ *für* $H(x) := \frac{\Phi_m(x^s)}{\Phi_m(x)}$.

(2) $\mathrm{Tr}_{E'|E}(v)$ *hat q-Ordnung* $\Phi_m(x)$.

(3) $\mathrm{Ord}_q\left(\mathrm{Tr}_{E'|E}(u)\right)$ *ist ein irreduzibler monischer Teiler von* $\Phi_m(x)$.

Beweis. Zerlegen wir $s = \bar{s}p^\beta$ mit $p \nmid s'$, so ist nach Voraussetzungen $\mathrm{ggT}(\bar{s}, m) = 1$ und wir sind in der Situation von Satz 2.15. Damit gilt

$$\Phi_m(x^s) \;=\; \Phi_m(x^{\bar{s}})^{p^\beta} \;=\; \prod_{d|\bar{s}} \Phi_{md}(x)^{p^\beta}\,.$$

Mit konsequenter Anwendung von Lemma 3.34 folgern wir die Behauptungen:

$$\mathrm{Ord}_q(H(\sigma_E)v) = \frac{\Phi_m(x^s)}{\mathrm{ggT}(\Phi_m(x^s), H(x))} = \Phi_m(x)\,.$$

Für (2) und (3) überlegen wir uns, dass für $a \in E'$

$$\mathrm{Tr}_{E'|E}(a) = \sum_{i=0}^{s-1} a^{q^{im}} = \left[\frac{x^{sm}-1}{x^m-1}\right](\sigma_E)(a)$$

und

$$\frac{x^{sm}-1}{x^m-1} \;=\; \frac{\prod_{d|\bar{s}m} \Phi_d(x)^{p^\beta}}{\prod_{d|m} \Phi_d(x)} \;=\; \Phi_m(x)^{p^\beta-1} \prod_{\substack{l|m\\ l\neq m}} \Phi_l(x)^{p^\beta-1} \prod_{\substack{d|\bar{s}\\ d\neq 1}} \Phi_{md}(x)^{p^\beta}\,.$$

Ergo ist

$$\mathrm{Ord}_q(\mathrm{Tr}_{E'|E}(v)) \;=\; \frac{\Phi_m(x^s)}{\mathrm{ggT}(\Phi_m(x^s), \frac{x^{sm}-1}{x^m-1})} \;=\; \Phi_m(x)$$

und

$$\mathrm{Ord}_q(\mathrm{Tr}_{E'|E}(u)) \;=\; \frac{f(x^s)}{\mathrm{ggT}(f(x^s), \frac{x^{sm}-1}{x^m-1})} \;=\; f_1(x)\,,$$

wobei wir uns hierfür noch einmal an **??** erinnern müssen, wo wir gezeigt haben, dass in genau dieser Situation $f(x^s) = \prod_{i=1}^{?} f_i(x) \in F[x]$ mit f_i irreduzibler monischer Teiler von $\Phi_{md}(x)$ für $d \mid \bar{s}$, wobei wir oBdA $f_1 \mid \Phi_m(x)$ für den einzigen Teiler von $\Phi_m(x)$ wählen. □

Wir können sogar noch in gewisser Weise eine Verschärfung dieses Satzes angeben, die genauer charakterisiert, warum die Spurfunktion in diesem Fall gute Dienste leistet.

Satz 4.17.

Seien die Voraussetzungen wie in Satz 4.16. Ist dann wiederum u eine primitive (nl)-te Einheitswurzel und $k(x) = \sum_{i=0}^{j} a_i x^i \in F[x]$, $j \in \mathbb{N}$, so dass $\mathrm{Ord}_q(u) = \mathrm{Ord}_q(u^i) = f(x^s)$ für einen irreduziblen monischen Teiler $f(x)$ von $\Phi_m(x)$ über F für alle $i = 0, \ldots, j$. Gilt letztlich $[F(k(u)) : F] = n$, so gilt: $\mathrm{Ord}_q(k(u))$ ist ein irreduzibler monischen Teiler von $\Phi_m(x)$ über F.

Beweis. Sammeln wir einmal den Wissensstand für gegebene Situation zusammen:

- $\mathrm{Ord}_q(u) = \mathrm{Ord}_q(u^i) = f(x^s)$ teilt $\Phi_m(x^s)$.
- Für $s = \bar{s}p^\beta$ mit $p \nmid \bar{s}$ gilt nach Satz 2.15
$$\Phi_m(x^s) = \prod_{d|\bar{s}} \Phi_{md}(x)^{p^\beta}$$
- Als leichte Konsequenz aus **??** haben wir
$$\mathrm{Ord}_q(k(u)) \mid \mathrm{kgV}\{\mathrm{Ord}_q(a_i u^i) : i = 1, \ldots, j\} = f(x^s)$$
- Da $k(u) \in E$ und die q-Ordnung immer das Minimalpolynom des zugehörigen Frobenius teilt, folgt
$$\mathrm{Ord}_q(k(u)) \mid x^m - 1 = \prod_{d|m} \Phi_d(x)$$

Zusammengefasst teilt also $\mathrm{Ord}_q(k(u))$ nur den Anteil von $f(x^s)$, der auch in $\Phi_m(x)$ liegt. Damit folgt nach **??** die Behauptung. □

Bemerkung 4.18. Auf diese Weise kann man auch den Beweis von Satz 4.16 führen, wenn man sich überlegt, dass für $\mathrm{Ord}_q(u) = f(x^s)$ auch $\mathrm{Ord}_q(u^{q^m}) = f(x^s)$ (m und s sind teilerfremd!) gilt und gerade $\mathrm{Tr}_{E'|E}(u) \in E$ gilt.

Auch für reguläre Erweiterungen wollen wir ein Analogon von Lemma 4.10 beweisen:

Lemma 4.19. *Seien die Voraussetzungen wie in Satz 4.15, also also $F := \mathbb{F}_q$ ein endlicher Körper für eine Primzahlpotenz $q = p^r$ und $m \in \mathbb{N}$ ungerade mit $p \nmid m$. Setze $s := \mathrm{ord}_{\nu(m)}(q)$, $l := \pi_m(q^s - 1)$, $b := b(q, m) := \mathrm{ggT}(l, m)$, $E := \mathbb{F}_{q^m}$ und $E' = \mathbb{F}_{q^{sm}}$. Ist nun θ eine primitive (nf)-te Einheitswurzel für $l \mid f \mid q^s - 1$, so ist $\mathrm{Ord}_q(\theta) = f(x^s)$ für f einen irreduzibler Teiler von $\Phi_m(x)$ über $F[x]$.*

Beweis. Lemma 4.10 mit dem Beweis von Satz 4.15. □

4.4 | Normalbasen mit Dickson-Polynomen

Scheerhorn zeigt in [9, 10], dass sich Dickson-Polynome eignen, um Normalbasen zu beschreiben. Insbesondere zeigt er die beiden folgenden Resultate.

Satz 4.20 ([10, Theorem 2]).

Seien $n \geq 3$ ein Produkt ungerader Primzahlen von $(q+1)$ und $a, b \in \mathbb{F}_q =: F$, so dass $D_n(x,a)-b \in F[x]$ irreduzibel ist. Sei $\gamma \in E := \mathbb{F}_{q^n}$ eine Wurzel von $D_n(x,a)$. Dann ist

$$E \;=\; \langle 1\rangle_q \oplus \bigoplus_{l \in R_q(n)\setminus\{0\}} \langle D_l(\gamma,a)\rangle_q \,,$$

eine Zerlegung von E in irreduzible $F[x]$-Teilmoduln, wobei

$$\langle D_l(\gamma,a)\rangle_q \;:=\; \operatorname{span}_F\{D_i(\gamma,a) \mid i \in M_q(l \bmod n)\}\,,$$

so dass $\{D_i(\gamma,a) \mid i \in M_q(l \bmod n)\}$ eine F-Basis von $\langle D_l(\gamma,a)\rangle_q$ ist.

Satz 4.21 ([10, Theorem 3]).

Seien $n \geq 3$ ein Produkt ungerader Primzahlen von $(q-1)$ und $a, b \in \mathbb{F}_q =: F$, so dass $D_n(x,a)-b \in F[x]$ irreduzibel ist. Seien ferner $x^2+bx+a^n = (x-\beta)(x-a^n\beta^{-1}) \in \mathbb{F}_q$ und $\theta \in \mathbb{F}_{q^n}$ eine Wurzel von $x^n-\beta$. Ist schießlich $\gamma \in \mathbb{F}_{q^n}$ eine Wurzel von $D_n(x,a)-b$, so gilt für $l \in R_q(n)\setminus\{0\}$

$$\langle \theta^l\rangle_q \oplus \langle \theta^{n-l}\rangle_q \;=\; \langle D_l(\gamma,a)\rangle_q$$

Wir wollen nun im Folgenden zeigen, dass dies relativ einfach aus der allgemeinen Theorie aus dem vorherigen Abschnitt folgt. Zunächst brauchen wir jedoch eine wesentliche Eigenschaft der Dickson-Polynome.

Lemma 4.22 (??). *Es gilt $D_n(x+ax^{-1},a) = x^n + a^n x^{-n}$.*

Nun können wir obige Sätze beweisen.

Beweis (von Satz 4.20). Betrachten wir die vorliegende Situation, so sehen wir, dass $s := \operatorname{ord}_{\nu(n)}(q) = 2$ und damit $n \in \mathcal{S}_{q^2}$. Nach **??** ist $x^2+bx+a^n \in F[x]$ irreduzibel. Sei $x^2+bx+a^n = (x-\beta)(x-a^n\beta^{-1})$ über $K := \mathbb{F}_{q^2}$. Dann ist $x^n-\beta \in \mathbb{F}_{q^2}[x]$ irreduzibel. Ist dann $\theta \in \mathbb{F}_{q^{2n}} =: E'$ eine Wurzel, so wissen wir nach Satz 4.11 und Lemma 4.19, dass $\operatorname{Ord}_q(\theta) = f(x^s)$ für einen irreduziblen Teiler $f(x)$ von $\Phi_n(x)$ über F. Nach Satz 4.16 ist

$$f_1(x) \;:=\; \operatorname{Ord}_q(\operatorname{Tr}_{E'|E}(\theta)) \;=\; \operatorname{Ord}_q(\theta+\theta^{q^n})$$

ein irreduzibler Teiler von $\Phi_n(x)$ in $F[x]$.

Es ist nun $\theta^{q^n} = \theta^{-1}$: Da $\theta \in E'$, gilt $\theta^{q^{2n}} = \theta$, also

$$\theta^{q^{2n}-1} \;=\; 1 \;=\; \theta^{(q^n+1)(q^n-1)}\,.$$

wäre $\theta^{q^n-1} = 1$, so läge θ bereits in $\mathbb{F}_{q^n}$, im Widerspruch zur Definition von θ. Also haben wir nach Satz 4.17 gerade

$$\mathrm{Ord}_q(\theta + a\theta^{-1}) = f_1(x)\,,$$

da mit Lemma 4.22

$$D_n(\theta + a\theta^{-1}, a) - b \;=\; \theta^n + a^n\theta^{-n} - b \;=\; \beta + a^n\beta^{-1} - b \;=\; b - b \;=\; 0\,,$$

also $\theta + a\theta^{-1}$ Grad n über F hat. Da wir mit θ^l für $l \in R_q(n)$ alle irreduziblen Teilmoduln [1] von $x^n - 1$ erzeugen können, also $\mathrm{Ord}_q(\theta^i) = g(x^s)$ für einen irreduziblen monischen Teiler $g(x)$ von $x^n - 1$ über F, ist alles gezeigt, da analog zu oben

$$\mathrm{Ord}_q(\theta^l + a^l\theta^{-l}) \;\mid\; x^n - 1$$

für $l \in R_q(n)$ ein irreduzibler Teiler von $x^n - 1$ ist (für $l = 0$ hat $\theta^l + a^l\theta^{-l}$ natürlich Grad 0 über F). $\gamma := \theta + a\theta^{-1}$ und $D_l(\gamma, a) = \theta^l + a^l\theta^{-l}$ schließt den Beweis ab. □

[1] hier vielleicht noch ein Satz/Lemma/Bemerkung im vorherigen Abschnitt über die Erzeugung *aller* irreduziblen Teilmodule

Kapitel 5

Vollständige Normalbasen

In den vorherigen Kapiteln haben wir einige Resultate zu Normalbasen kennen gelernt. Es stellt sich jedoch ganz natürlich die Frage, ob dieser Begriff nicht erweitert werden kann: Für eine Körpererweiterung E über F existieren im Allgemeinen Zwischenkörper $F \mid K \mid E$, so wollen wir untersuchen, ob ein Element $w \in E$, welches normal über F ist, auch normal über allen Zwischenkörpern bleibt. Diese Eigenschaft verdient einen eigenen Namen:

Definition 5.1 (vollständig normal).

Sei $E \mid F$ eine Körpererweiterung endlicher Körper E über F. $w \in E$ heißt *vollständig normal*, falls w normal über jedem Zwischenkörper $E \mid K \mid F$ ist.

Ungeklärt ist in diesem Moment, ob überhaupt vollständig normale Elemente existieren. Greifen wir auf das zurück, was wir über normale Elemente wissen, so treffen wir folgende Definition und erkennen

Definition 5.2 (einfach).

Eine Körpererweiterung $E \mid F$ endlicher Körper F und E heißt *einfach*, falls jedes normale Element von E über F bereits vollständig normal ist.

Satz 5.3.

Sei $\mathbb{F}_{q^m} \mid \mathbb{F}_q$ eine Erweiterung endlicher Körper. Dann ist äquivalent:

(1) $\mathbb{F}_{q^m} \mid \mathbb{F}_q$ ist einfach.

(2) Für jeden Primteiler $r \mid m$ ist jedes normale Element in $\mathbb{F}_{q^m}$ über $\mathbb{F}_q$ auch normal in $\mathbb{F}_{q^m}$ über $\mathbb{F}_{q^r}$.

(3) Für jeden Primteiler $r \mid m$ teilt r nicht $\operatorname{ord}_{(\frac{m}{r})'}(q)$.

Definition 5.4 (verallgemeinertes Kreisteilungspolynom).

Sei F ein endlicher Körper. Seien $k, t \geq 1$ natürliche Zahlen und k teilerfremd zu $\operatorname{char} F$, so heißt

$$\Phi_{k,t}(x) \;:=\; \Phi_k(x^t) \;\in F[x]$$

verallgemeinertes Kreisteilungspolynom.

Definition 5.5 (verallgemeinerter Kreisteilungsmodul, Modulcharakter).

Sei $\Phi_{k,t}$ ein verallgemeinertes Kreisteilungspolynom über einem endlichen Körper F. Notiere ferner $\sigma : \bar{F} \to \bar{F}$ den Frobenius von F,so heißt

$$\mathcal{C}_{k,t} \;:=\; \{w \in \bar{F} :\; \Phi_{k,t}(\sigma)(w) = 0\}$$

verallgemeinerter Kreisteilungsmodul.

Der *Modulcharakter* von $\mathcal{C}_{k,t}$ ist $\frac{kt}{\nu(k)}$.

Definition 5.6 (vollständiger Erzeuger).

Sei $\mathcal{C}_{k,t}$ ein verallgemeinerter Kreisteilungsmodul über $\mathbb{F}_q$. $w \in \bar{F}$ heißt *vollständiger Erzeuger von* $\mathcal{C}_{k,t}$, falls w ein Erzeuger von $\mathcal{C}_{k,t}$ als $\mathbb{F}_{q^d}[x]$-Modul für alle Teiler d des Modulcharakters $\frac{kt}{\nu(k)}$ ist.

Definition 5.7 (Zerlegung in verallgemeinerte Kreisteilungsmoduln).

Sei $\Phi_{k,t}$ ein verallgemeinertes Kreisteilungspolynom über F. $\Delta \subseteq F[x]$ heißt eine *Zerlegung von* $\Phi_{k,t}$ *in verallgemeinerte Kreisteilungspolynome*, falls Δ nur verallgemeinerte Kreisteilungspolynome enthält, diese paarweise teilerfremd sind und

$$\Phi_{k,t}(x) \;=\; \prod_{\Psi \in \Delta} \Psi(x)\,.$$

Definiere ferner

$$i(\Delta) \;:=\; \{(l,s) \in \mathbb{N}^2 :\; \Phi_{l,s} \in \Delta\}\,.$$

Definition 5.8 (verträgliche Zerlegung).

Sei Δ eine Zerlegung von $\Phi_{k,t}$ in verallgemeinerte Kreisteilungspolynome über F. Dann heißt Δ *verträgliche Zerlegung* falls gilt: Sei für jedes $(l,s) \in i(\Delta)$ $w_{l,s} \in \bar{F}$ ein vollständiger Erzeuger von $\mathcal{C}_{l,s}$ über F, so ist

$$w = \sum_{(l,s) \in i(\Delta)} w_{l,s}$$

ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$ über F.

Satz 5.9 (Zerlegungssatz für verallgemeinerte Kreisteilungsmoduln).

Sei $\Phi_{k,t}$ ein verallgemeinertes Kreisteilungspolynom über einem endlichen Körper $\mathbb{F}_q$ mit Charakteristik p. Sei r eine Primzahl mit

- $r \mid t$,
- $r \neq p$,
- $r \nmid k$.

Dann ist

$$\Delta_r := \{\Phi_{k,\frac{t}{r}},\ \Phi_{kr,\frac{t}{r}}\}$$

eine Zerlegung von $\Phi_{k,t}$ in verallgemeinerte Kreisteilungspolynome und diese ist verträglich genau dann, wenn

$$r^a \nmid \operatorname{ord}_{\nu(kt')}(q)$$

mit $a = \max\{b \in \mathbb{N} : r^b \mid t\}$.

Definition 5.10 (regulär).

Ein verallgemeinerter Kreisteilungsmodul $\mathcal{C}_{k,t}$ mit $\operatorname{ggT}(k,t) = 1$ heißt *regulär*, falls $\operatorname{ord}_{\nu(k\,t')}(q)$ und $k\,t$ teilerfremd sind.

Definition 5.11 (ausfallend).

Sei $\mathcal{C}_{k,p^b}$ ein regulärer verallgemeinerter Kreisteilungsmodul über $\mathbb{F}_q$ mit $\operatorname{char}\mathbb{F}_q = p$. Schreibe $k = 2^c \cdot \bar{k}$ mit $\bar{k}$ ungerade. Dann heißt $\mathcal{C}_{k,p^b}$ *ausfallend*, falls gilt:

- $q \equiv 3 \bmod 4$,
- $c \geq 3$ und
- $\operatorname{ord}_{2^c}(q) = 2$.

Definition 5.12 (τ-Teiler).

Sei $\mathcal{C}_{k,p^b}$ ein regulärer verallgemeinerter Kreisteilungsmodul über $\mathbb{F}_q$. Schreibe

$$\operatorname{ord}_k(q) = \operatorname{ord}_{\nu(k)}(q) \prod_{r \in \pi(k)} r^{\alpha_r},$$

wobei $\pi(k)$ die Primteiler von k bezeichnen. Dann heißt

$$\tau := \tau(q,k) := \prod_{r \in \pi(k)} r^{\lfloor \frac{\alpha_r}{2} \rfloor}$$

der *τ-Teiler von $\mathcal{C}_{k,p^b}$*.

Satz 5.13 (Über reguläre Erweiterungen).

Sei $\mathbb{F}_q$ ein endlicher Körper von Charakteristik p. Seien k eine positive ganze Zahl teilerfremd zu q und $\mathcal{C}_{k,p^b}$ ein regulärer verallgemeinerter Kreisteilungsmodul. Dann gilt:

(1) Ist $\mathcal{C}_{k,p^b}$ nicht ausfallend, so ist $u \in \bar{F}$ genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k,p^b}$, falls

$$\mathrm{Ord}_{q^\tau}(u) = \Phi_{\frac{k}{\tau},p^b}\,.$$

(2) Ist $\mathcal{C}_{k,p^b}$ ausfallend, so ist $u \in \bar{F}$ genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k,p^b}$, falls

$$\mathrm{Ord}_{q^\tau}(u) = \Phi_{\frac{k}{\tau},p^b} \quad \textit{und} \quad \mathrm{Ord}_{q^{2\tau}}(u) = \Phi_{\frac{k}{2\tau},p^b}\,.$$

Kapitel 6

Enumeration primitiver und vollständig normaler Elemente

Grundsätzlich wurde zur konkreten Suche und Enumeration primitiver und vollständig normaler Elemente das Computeralgebrasystem `Sage` verwendet.

6.1 | Implementierung endlicher Körper und Körpererweiterungen

`Sage` bietet ja bereits die Möglichkeit in endlichen Körpern zu rechnen. Jedoch hat sich herausgestellt, dass die zugrunde liegenden `C`-Bibliotheken (im Allgemeinen Fall ist dies das `Pari C library`[1]) zu langsam sind. Dies ist sicherlich auf die Allgemeinheit ihrer Anwendungsgebiete zurückzuführen. Beispielsweise arbeitet die `Pari`-Bibliothek stets mit Ganzzahlen beliebiger Größe. Deren Arithmetik ist selbstredend aufwendiger und langsamer, als maschineninterne `Integer`-Arithmetik. Daher haben wir uns entschlossen eigene `C`-Bibliotheken anzulegen, die auf einfacher (jedoch begrenzter) `Integer`-Arithmetik basieren.

6.1.1 | Beschreibung von Elementen endlicher Körper

Die Implementierung von Primkörpern ist freilich kanonisch. Daher brauchen wir an dieser Stelle nicht viele Worte verlieren, da wir auf der Suche nach primitiv und vollständig normalen Elementen ohnehin nur in Erweiterungen von Graden größer 1 zu rechnen haben.

Sei also $\mathbb{F}_q$ ein endlicher Körper von Charakteristik p und $q = p^r$ für $r > 1$. Wie auch in `Sage` üblich, haben wir uns entschieden bei der programmatischen Beschreibung die Isomorphie

$$\mathbb{F}_q \cong \mathbb{F}_p[x]/(f(x))$$

mit $f(x) \in F_p[x]$ irreduzibel von Grad r zu nutzen. Also wird ein Element $w \in \mathbb{F}_q$ als Array der Länge $r+1$ beschrieben, wobei die nullte Stelle des Arrays auch den Koeffizienten von x^0 meint, und alle Berechnungen (insb. Multiplikation) modulo $f(x)$ ausgeführt.

[1] vgl. `http://www.sagemath.org/doc/reference/rings_standard/sage/rings/finite_rings/constructor.html`

Es hat sich herausgestellt, dass es von Vorteil ist, neben dem Koeffizienten tragenden Array ein weiteres Array mitzuführen, welches die Indizes speichert, deren zugehörige Koeffizienten nicht verschwinden. Letztlich fehlt noch, wie es in `C` üblich und notwendig ist, die Länge des Indexarrays zu speichern und wir erhalten den Datentyp `struct FFElem`.

Listing 6.1: Aus `../Sage/enumeratePCNs.c`

```
15 /**
16  * Finite Field Element.
17  *
18  * !! idcs must be in desc order !!
19  *
20  * Uses int arrays, i.e. you must not consider
21  * PrimeFields of order p with (p-1)*(p-1) > INT_MAX
22  */
23 struct FFElem{
24     int *el;
25     int *idcs;
26     int len;
27 };
```

`len` gibt immer die Länge von `idcs` an. Zusätzlich fordern wir noch folgende Eigenschaften, die den Umgang mit `struct FFElem` erleichtern.

Invariante 6.1.

Für das Indexarray `idcs` eines `struct FFElem` sei sichergestellt, dass die Werte stets in absteigender Reihenfolge sortiert sind.

Invariante 6.2.

Bei der Benutzung von `struct FFElem` sei sichergestellt, dass die Länge aller auftretenden Arrays dem Grade der Körpererweiterung über dem jeweiligen Primkörper entspricht

Satz 6.1 erleichtert den Zugriff auf den Grad des Elements (also seinen Grad als Polynom in $\mathbb{F}_p[x]/(f(x))$). Letztere Invariante stellt sicher, dass durch Veränderung eines `struct FFElem` (beispielsweise Arithmetik) kein Speicherzugriffsfehler auftritt.

Beispiel 6.3. Wollen wir das Element

$$w := x^8 + 2 * x^6 + x^2 + 2 \in \mathbb{F}_3[x]$$

des endlichen Körpers $\mathbb{F}_{3^{10}}$ (wir verzichten auf Angabe eines Minimalpolynoms, da es hier keine Rolle spielt) in obiger Darstellung beschreiben, so müssen wir `C`-üblich Speicher allokieren und die Arrays in passender Länge anlegen:

```
struct FFElem *w = malloc(sizeof(struct FFElem));
w->el = (int[]) {2, 0, 1, 0, 0, 0, 2, 0, 1, 0};
w->idcs = (int[]) {9, 7, 2, 0, 0, 0, 0, 0, 0, 0};
w->len = 4;
```

Der besseren Lesbarkeit zu Gute haben wir die ungenutzten Indizes und die verschwindenden Koeffizienten mit `0` aufgefüllt. Man überlege sich jedoch, dass lediglich eine einzige `0` notwendig ist und alle anderen beliebig ersetzt werden könnten. Beispielsweise ist

```
struct FFElem *w = malloc(sizeof(struct FFElem));
w->el = (int[]) {2, -10, 1, 100, -2, -3, 2, -4, 1, -8};
w->idcs = (int[]) {9, 7, 2, 0, -3, -2, -5, -1, -1, -1};
w->len = 4;
```

mit obiger Beschreibung identisch. ▲

Hilfsfunktionen zum Anlegen und Löschen

Da `C` ohne *Garbage-Collection* auskommt, muss man selbst für die entsprechende Speicherverwaltung sorgen. Dies erleichtern die Funktionen `mallocFFElem` und `freeFFElem`.

Listing 6.2: Aus `../Sage/enumeratePCNs.c`

```
30 inline struct FFElem *mallocFFElem(int m){
31     struct FFElem *ff = malloc(sizeof(struct FFElem));
32     ff->el = malloc(m*sizeof(int));
33     ff->idcs = malloc(m*sizeof(int));
34     ff->len = 0;
35     return ff;
36 }
```

Listing 6.3: Aus `../Sage/enumeratePCNs.c`

```
37 inline void freeFFElem(struct FFElem *ff){
38     free(ff->el);
39     free(ff->idcs);
40     free(ff);
41 }
```

Schließlich führen wir noch eine Funktion ein, die den Inhalt eines `struct FFElem`s in ein neues kopiert. Dieses muss aber bereits allokiert sein!

Listing 6.4: Aus `../Sage/enumeratePCNs.c`

```
50 /**
51  * Copies the content of ff1 into ff2
52  *
53  * !! ff2 must be malloced!
54  */
55 inline void copyFFElem(struct FFElem *ff1, struct FFElem *ff2){
56     if(ff1 == ff2) return;
57     int i;
58     for(i=0;i < ff1->len;i++){
59         ff2->idcs[i] = ff1->idcs[i];
60         ff2->el[ ff1->idcs[i] ] = ff1->el[ ff1->idcs[i] ];
61     }
62     ff2->len = ff1->len;
63 }
```

6.1.2 Arithmetik in endlichen Körpern

Additions- und Multiplikationstabellen

Will man Arithmetik mit `struct FFElems` betreiben, so stellt sich sicherlich am Anfang die Frage, wie die Arithmetik im Primkörper $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$ aussehen möge. Da die `FFElems` auf `int`-Arrays basieren liegt es nahe, die Addition bzw. Multiplikation zweier Elemente $a, b \in \mathbb{F}_p$ durch die integrierten Funktionen `(a+b) % p` und `(a*b) % p` zu implementieren. Es hat sich jedoch herausgestellt, dass dies vergleisweise langsam ist. Insbesondere bei kleinen Primzahlen hat sich das Anlegen einer Additions- und einer Multiplikationstabelle bewährt. Diese sind `int`-Arrays, sodass die `(a+b)`-te Stelle der Additions- und die `(a*b)`-te Stelle der Multiplikationstabelle gerade das Ergebnis der jeweiligen Rechnung in $\mathbb{F}_p$ liefert.

Bemerkung 6.4. Um sich nicht um vorzeichenbehaftete Werte kümmern zu müssen, überdecken die Tabellen auch negative Bereiche und daher ist eine Additionstabelle in $\mathbb{F}_p$ stets von Länge $4(p-1)+1$ und eine Multiplikationstabelle von Länge $2(p-1)^2+1$.

Beispiel 6.5. Betreiben wir Arithmetik in $\mathbb{F}_3$, so legen wir eine Additions- bzw. Multiplikationstabelle wie folgt an und stellen durch eine Verschiebung des Pointers sicher, dass auch vorzeichenbehaftete Rechnungen richtig erfasst werden können.

```
int addTableRaw[] = {2, 0, 1, 2, 0, 1, 2, 0, 1};
int initialAddShift = 4;
int *addTable = addTableRaw+initialAddShift;
int multTableRaw[] = {2, 0, 1, 2, 0, 1, 2, 0, 1};
int initialMultShift = 4;
int *multTable = multTableRaw+initialMultShift;
```

Führen wir nun Rechnungen durch können wir diese nutzen:

```
addTable[ 2+1 ] // == 0
addTable[ 0-2 ] // == 1
multTable[ 2*2 ] // == 1
```

Addition

Aufgrund der effizienteren Darstellung der Elemente endlicher Körper durch Speicherung ihrer Indices, ist die Addition nicht lediglich gegeben durch komponentenweise Betrachtung, sondern erfordert etwas mehr Aufwand.

Listing 6.5: Aus `../Sage/enumeratePCNs.c`

```
226 /**
227  * Adds two FFElems.
228  *
229  * !! ff1 may be same as ret !!
230  * !! ff2 must not be same as ret !!
231  */
232 inline void addFFElem(struct FFElem *ff1, struct FFElem *ff2,
233         struct FFElem *ret,
```

```
234         int *tmp,
235         int *multTable, int *addTable){
236     int i=0,j=0,k=0, i2;
237     bool end = false;
238     //handle trivial cases
239     if(ff1->len == 0){
240         copyFFElem(ff2,ret);
241         return;
242     }
243     if(ff2->len == 0){
244         copyFFElem(ff1,ret);
245         return;
246     }
247     copyArray(ff1->idcs,tmp,ff1->len);
248     while( end == false ){
249         while( tmp[i] != ff2->idcs[j] ){
250             if( tmp[i] > ff2->idcs[j] ){
251                 ret->el[ tmp[i] ] = ff1->el[ tmp[i] ];
252                 ret->idcs[k] = tmp[i];
253                 i++; k++;
254             }else if( tmp[i] < ff2->idcs[j] ){
255                 ret->el[ ff2->idcs[j] ] = ff2->el[ ff2->idcs[j] ];
256                 ret->idcs[k] = ff2->idcs[j];
257                 j++; k++;
258             }
259             if(i == ff1->len || j == ff2->len){
260                 end = true;
261                 break;
262             }
263 
264         }
265         if(end == true) break;
266         //tmp[i] == ff2->idcs[j]
267         i2 = tmp[i];
268         ret->el[i2] = addTable[ ff1->el[i2] + ff2->el[i2] ];
269         if(ret->el[i2] != 0){
270             ret->idcs[k] = i2;
271             k++;
272         }
273         i++; j++;
274         if(i == ff1->len || j == ff2->len) end = true;
275     }
276     //add rest of ff1 or ff2
277     if(i != ff1->len ){
278         while(i<ff1->len){
279             ret->el[ tmp[i] ] = ff1->el[ tmp[i] ];
280             ret->idcs[k] = tmp[i];
281             i++; k++;
282         }
283     }else if(j != ff2->len){
284         while(j<ff2->len){
285             ret->el[ ff2->idcs[j] ] = ff2->el[ ff2->idcs[j] ];
286             ret->idcs[k] = ff2->idcs[j];
287             j++; k++;
```

```
288         }
289     }
290     ret->len = k;
291 }
```

Wie später aus der Beschreibung anderer Algorithmen hervorgeht, ist es von Vorteil, wenn das Ergebnis einer Addition bereits eines der beiden addierten Elemente ist. Auf diese Weise spart man das Anlegen unnötiger Hilfs-`FFElem`s. Wie man schnell einsieht, werden jeweils nur die beiden Indexarrays durchlaufen und lediglich wenn diese gleich sind, muss eine Addition ausgeführt werden; ansonsten reicht es den jeweiligen Koeffizienten zu übernehmen.

Multiplikation

Wir haben uns entschieden, keine speziellen Multiplikationsalgorithmen (wie Karatsuba oder FFT-basierte Algorithmen) zu implementieren, da die hier betrachteten Erweiterungen nicht von Graden sind, in denen jene Algorithmen ihre Vorteile ausspielen könnten.

Listing 6.6: Aus `../Sage/enumeratePCNs.c`

```
302 /**
303  * Multiplies two FFElems and reduces the result by mipo.
304  *
305  * !! tmp must have at least length m!
306  * !! ret must be malloced!
307  */
308 inline void multiplyFFElem(struct FFElem *ff1, struct FFElem *ff2,
309         struct FFElem *ret,
310         struct FFElem *mipo, int *tmp, int m,
311         int *multTable, int *addTable){
312     /*
313      * catch trivial cases
314      */
315     if(ff1->len == 0 || ff2->len == 0){
316         ret->len = 0;
317         return;
318     }
319     if(ff1->len == 1 && ff1->idcs[0] == 0 && ff1->el[0] == 1){
320         copyFFElem(ff2, ret);
321         return;
322     }
323     if(ff2->len == 1 && ff2->idcs[0] == 0 && ff2->el[0] == 1){
324         copyFFElem(ff1, ret);
325         return;
326     }
327
328     /*
329      * Do multiplication
330      */
331     int maxlen = ff1->idcs[0] + ff2->idcs[0] + 1;
332     int i,j,i2,j2,k;
333     int max2 = maxlen;
334     if( maxlen > m ){
```

```
335         max2 = m;
336         initPoly(tmp,maxlen-m);
337     }
338     initPoly(ret->el,max2);
339     //multiply
340     for(i=0;i<(ff1->len);i++){
341         for(j=0;j<(ff2->len);j++){
342             i2 = ff1->idcs[i];
343             j2 = ff2->idcs[j];
344             k = i2+j2;
345             if(k<m){
346                 ret->el[k] = addTable[ ret->el[k] +
347                     multTable[ ff1->el[i2] * ff2->el[j2] ] ];
348             }else{
349                 tmp[k-m] = addTable[ tmp[k-m] +
350                     multTable[ ff1->el[i2] * ff2->el[j2] ] ];
351             }
352         }
353     }
354 
355     /*
356      * Reduce mod mipo
357      */
358     if(maxlen > m){
359         int quo;
360         for(i=maxlen-m-1;i>=0;i--){
361             quo = tmp[i];
362             if(quo == 0) continue;
363             for(j=0;j<(mipo->len); j++){
364                 j2 = mipo->idcs[j];
365                 k = i+j2;
366                 if(k>=m){
367                     tmp[k-m] = addTable[ tmp[k-m] -
368                         multTable[ mipo->el[j2]*quo ] ];
369                 }else{
370                     ret->el[k] = addTable[ ret->el[k] -
371                         multTable[ mipo->el[j2]*quo ] ];
372                 }
373             }
374         }
375     }
376 
377     /*
378      * Recalc indices
379      */
380     i2 = 0;
381     for(i=max2-1;i>=0;i--){
382         if(ret->el[i] != 0){
383             ret->idcs[i2] = i;
384             i2++;
385         }
386     }
387     ret->len = i2;
388 }
```

Außer den beiden zu multiplizierenden `FFElem`s muss man natürlich das Minimalpolynom des zu Grunde liegenden Körpers und dessen Grad über dem Primkörper – hier mit `int m` bezeichnet – mit übergeben. Leider war es an dieser Stelle im Gegensatz zur Addition nicht möglich, die Indizes des Produkts direkt zu berechnen, da es sich bei den Koeffizienten des Produkts ja Summen von Produkten von Koeffizienten der beiden Faktoren handelt. Daher muss nach der Reduktion modulo Minimalpolynoms eine Neuberechnung der Indizes erfolgen.

Quadratur

Im Hinblick auf das Testen von `struct FFElem`s auf Primitivität und dem damit verbundenen Potenzieren, existiert eine separate Funktion zur Quadrierung eines `FFElem`s. Es ist klar, dass beim Quadrieren weniger Produkte und Summen berechnet werden müssen als bei einer allgemeinen Multiplikation.

Listing 6.7: Aus `../Sage/enumeratePCNs.c`

```
404 /**
405  * Squares an FFElem
406  *
407  * !! ff is not modified !!
408  * !! tmp must have at least length m !!
409  */
410 inline void squareFFElem(struct FFElem *ff, struct FFElem *mipo,
411         struct FFElem *ret, int *tmp, int m,
412         int *multTable, int *addTable){
413     /*
414      * catch trivial cases
415      */
416     if(ff->len == 0){
417         copyFFElem(ff,ret);
418         return;
419     }
420     if(ff->len == 1 && ff->idcs[0] == 0 && ff->el[0] == 1){
421         copyFFElem(ff,ret);
422         return;
423     }
424
425     /*
426      * Do multiplication
427      */
428     int maxlen = 2*ff->idcs[0] + 1;
429     int i,j,i2,j2,k;
430     int max2 = maxlen;
431     if( maxlen > m ){
432         max2 = m;
433         initPoly(tmp,maxlen-m);
434     }
435     initPoly(ret->el,max2);
436     for(i=0;i<(ff->len);i++){
437         // same index must be squared
```

```
438         i2 = ff->idcs[i];
439         k = 2*i2;
440         if(k<m){
441             ret->el[k] = addTable[ ret->el[k] +
442                 multTable[ ff->el[i2]*ff->el[i2] ] ];
443         }else{
444             tmp[k-m] = addTable[ tmp[k-m] +
445                 multTable[ ff->el[i2]*ff->el[i2] ] ];
446         }
447         // other indices only multipied and doubled
448         for(j=i+1;j<(ff->len);j++){
449             i2 = ff->idcs[i];
450             j2 = ff->idcs[j];
451             k = i2+j2;
452             if(k<m){
453                 ret->el[k] = addTable[ ret->el[k] +
454                     multTable[ 2 * multTable[ ff->el[i2] * ff->el[j2] ] ] ];
455             }else{
456                 tmp[k-m] = addTable[ tmp[k-m] +
457                     multTable[ 2 * multTable[ ff->el[i2] * ff->el[j2] ] ]];
458             }
459         }
460     }
461     /*
462      * Reduce mod mipo
463      */
464     if(maxlen > m){
465         int quo;
466         for(i=maxlen-m-1;i>=0;i--){
467             quo = tmp[i];
468             if(quo == 0) continue;
469             for(j=0;j<(mipo->len); j++){
470                 j2 = mipo->idcs[j];
471                 k = i+j2;
472                 if(k>=m){
473                     tmp[k-m] = addTable[ tmp[k-m] -
474                         multTable[ mipo->el[j2]*quo ] ];
475                 }else{
476                     ret->el[k] = addTable[ ret->el[k] -
477                         multTable[ mipo->el[j2]*quo ] ];
478                 }
479             }
480         }
481     }
482
483     /*
484      * Recalc indices
485      */
486     i2 = 0;
487     for(i=max2-1;i>=0;i--){
488         if(ret->el[i] != 0){
489             ret->idcs[i2] = i;
490             i2++;
491         }
```

```
492     }
493     ret->len = i2;
494 }
```

6.1.3 Matrizen und Polynome über endlichen Körpern

Matrizen und Matrixmultiplikation

Nach **??** ist das Potenzieren mit der Charakteristik in endlichen Körpern eine lineare Abbildung. Dies wollen wir Nutzen und haben daher als Darstellung von Matrizen über endlichen Körpern naheliegenderweise ein Array aus `FFElems` gewählt.

Listing 6.8: Aus `../Sage/enumeratePCNs.c`

```
515 /**
516  * Matrix multiplication
517  *
518  * !! tmp must have at least length m!
519  */
520 inline void matmul(struct FFElem **mat, struct FFElem *ff,
521         struct FFElem *ret,
522         int m, int *multTable, int *addTable){
523     int i,j,i2, row;
524     bool end;
525     for(row=0;row<m;row++){
526         ret->el[row] = 0;
527         i=0; j=0;
528         end = false;
529         while(end == false){
530             while(ff->idcs[i] != mat[row]->idcs[j]){
531                 if(ff->idcs[i] > mat[row]->idcs[j]) i++;
532                 else if(ff->idcs[i] < mat[row]->idcs[j]) j++;
533                 if(i == ff->len || j == mat[row]->len){
534                     end = true;
535                     break;
536                 }
537             }
538             if(end == true) break;
539             i2 = ff->idcs[i]; // == mat[row]->idcs[j]
540             ret->el[row] = addTable[ ret->el[row]
541                 + multTable[ mat[row]->el[i2]*ff->el[i2] ] ];
542             i++;
543             j++;
544             if(i==ff->len || j==mat[row]->len) end = true;
545         }
546     }
547     i2 = 0;
548     for(i=m-1;i>=0;i--){
549         if(ret->el[i] != 0){
550             ret->idcs[i2] = i;
551             i2++;
552         }
```

```
553     }
554     ret->len = i2;
555 }
```

Hier wird – anders als bei der Addition – nur nach den gemeinsamen Indizes gesucht (alle anderen Produkte sind schließlich 0). Invariante 6.1 stellt dabei wiederum sicher, dass das hier aufgeführte Verfahren funktioniert.

Ferner existiert eine Funktion, die das Freigeben von Matrizen erleichtert.

Listing 6.9: Aus `../Sage/enumeratePCNs.c`

```
43 inline void freeFFElemMatrix(struct FFElem **mat, int len){
44     if(mat==0) return;
45     int i;
46     for(i=0;i<len;i++) freeFFElem(mat[i]);
47     free(mat);
48 }
```

Polynome

Im Hinblick auf das Testen von `FFElems` auf vollständige Normalität (bzw. vollständige Erzeuger-Eigenschaft) müssen wir einen Weg wählen, Polynome über endlichen Körpern darzustellen; also Polynome deren Koeffizienten `FFElems` sind. Dazu führen wir ein eigenes **`struct`** ein.

Listing 6.10: Aus `../Sage/enumeratePCNs.c`

```
144 struct FFPoly{
145     struct FFElem **poly;
146     int lenPoly;
147 };
```

Listing 6.11: Aus `../Sage/enumeratePCNs.c`

```
149 inline struct FFPoly *mallocFFPoly(int m, int lenPoly){
150     struct FFPoly *poly = malloc(lenPoly*sizeof(struct FFElem*));
151     poly->lenPoly = lenPoly;
152     int i;
153     for(i=0;i<lenPoly;i++) poly->poly[i] = mallocFFElem(m);
154     return poly;
155 }
```

Listing 6.12: Aus `../Sage/enumeratePCNs.c`

```
157 inline void freeFFPoly(struct FFPoly *poly){
158     int i;
159     for(i=0;i<poly->lenPoly;i++) freeFFElem(poly->poly[i]);
160     free(poly->poly);
161     free(poly);
162 }
```

6.2 | Potenzieren und Primitivitätstest

6.2.1 | Potenzieren

Für das Potenzieren von `FFElems` wurde stets ein Square-and-Multiply-Ansatz verwendet. Da in endlichen Körpern jedoch das Potenzieren mit der Charakteristik eine lineare Abbildung darstellt, ist es a priori nicht unklug eine p-adische Square-and-Multiply-Variante zu wählen. Es hat sich jedoch herausgestellt, dass in den meisten Fällen normales Square-and-Multiply schneller ist als sein p-adisches Pendant. Dies veranschaulicht auch nachstehendes Beispiel.

Beispiel 6.6. Sei $u \in E := \mathbb{F}_{3^4}$ und zu berechnen sei u^{16}, so stellen wir zunächst 16 binär und 3-adisch da:

$$16 = 10000_2 = 121_3 .$$

Damit gilt

$$u^{16} = ((u^2)^2)^2)^2 = (u^3 \cdot u \cdot u)^3 \cdot u .$$

In einer Implementierung sehen wir also, dass die binäre Exponentiation 4 Quadrierungen „kostet“, die 3-adische Version hingegen 2 Matrixmultiplikationen und 3 Multiplikationen. Da in der Regel allgemeine Multiplikationen teuer sind, wäre in diesem Fall die binäre Variante wohl die bessere Wahl.
Wollen wir u^{10} berechnen, so sehen wir aus

$$10 = 1010_2 = 101_3 ,$$

dass in diesem Fall die binäre Exponentiation 4 Quadierungen und eine allgemeine Multiplikation erfordert, die 3-adische Variante jedoch nur 2 Matrixmultiplikationen und 1 allgemeine Multiplikation. Letzteres lässt sich sogar auf eine Matrixmultiplikation reduzieren, berechnet man die Darstellungsmatrix der linearen Abbildung $E \to E$, $x \mapsto x^9$ bereits vorher! Die beiden Varianten der Berechnung würden in diesem Fall also wie folgt von Statten gehen:

$$u^{10} = ((u^2)^2 \cdot u)^2 = u^9 \cdot u .$$ ▲

Nachstehend werden nun die beiden Varianten der Implementierung der Potenzierung aufgeführt. Wir beginnen mit p-adischem Square-and-Multiply. Zu bemerken ist, dass die Potenz bereits in p-adischer Darstellung als `int`-Array übergeben werden muss. Zudem werden vermeidbare Matrixmultiplikationen (vgl. obiges Beispiel) nicht durchgeführt und es ist sicherzustellen, dass `struct FFElem **matCharac` als `struct FFElem*`-Array von Länge $(l+1)m$ ist, wobei l die Länge des maximal auftretenden 0-Intervalls in der p-adischen Darstellung meint (in obigem Beispiel bei u^{10} wäre $l = 1$).

Listing 6.13: Aus `../Sage/enumeratePCNs.c`

```
562 /**
563  * Square and multiply in charac
564  * mat is powering by charac
565  *
566  * !! ff is modified !!
567  */
568 inline void powerFFElem(struct FFElem *ff, struct FFElem *mipo,
```

```
569         struct FFElem *ret,
570         int m, int *power, int powerLen,
571         struct FFElem **matCharac, int *tmp, struct FFElem *ffTmp,
572         int *multTable, int *addTable){
573     int i,j,k;
574     int lenCurGap = 0;
575     struct FFElem *ffSwitch = 0;
576     struct FFElem *ffRetInt = ret;
577     // init ret to 1
578     ffRetInt->el[0] = 1; ffRetInt->idcs[0] = 0; ffRetInt->len = 1;
579     for(j=powerLen-1;j>=0;j--){
580         for(k=0;k<power[j];k++){
581             multiplyFFElem(ffRetInt,ff,ffTmp, mipo,tmp,m,multTable,addTable);
582             ffSwitch = ffRetInt; ffRetInt = ffTmp; ffTmp = ffSwitch;
583         }
584         if(j==0 || power[j-1] == 0){
585             lenCurGap++;
586             continue;
587         }
588         matmul(matCharac+lenCurGap*m, ff, ffTmp, m, multTable,addTable);
589         ffSwitch = ff; ff = ffTmp; ffTmp = ffSwitch;
590         lenCurGap = 0;
591     }
592     copyFFElem(ffRetInt,ret);
593 }
```

Als nächstes folgt die standardmäßige binäre Exponentiation. Auch hier wird die Potenz bereits in Binärdarstellung erwartet.

Listing 6.14: Aus `../Sage/enumeratePCNs.c`

```
602 /**
603  * Square and multiply
604  *
605  * !! ff is modified !!
606  */
607 inline void powerFFElemSqM(struct FFElem *ff, struct FFElem *mipo,
608         struct FFElem *ret,
609         int m, int *power, int powerLen,
610         int *tmp, struct FFElem *ffTmp,
611         int *multTable, int *addTable){
612     int i,j,k;
613     int lenCurGap = 0;
614     struct FFElem *ffSwitch = 0;
615     struct FFElem *ffRetInt = ret;
616     // init ret to 1
617     ffRetInt->el[0] = 1; ffRetInt->idcs[0] = 0; ffRetInt->len = 1;
618     for(j=powerLen-1;j>=0;j--){
619         if(power[j] == 1){
620             multiplyFFElem(ffRetInt,ff,ffTmp, mipo,tmp,m,multTable,addTable);
621             //switch ffTmp and ffRetInt
622             ffSwitch = ffRetInt; ffRetInt = ffTmp; ffTmp = ffSwitch;
623         }
624         if(j>0){
```

```
625             squareFFElem(ff,mipo,ffTmp,tmp,m,multTable,addTable);
626             //switch ffTmp and ff
627             ffSwitch = ff; ff = ffTmp; ffTmp = ffSwitch;
628         }
629     }
630     copyFFElem(ffRetInt,ret);
631 }
```

6.2.2 | Primitivitätstest

Beim Testen eines Elements eines endlichen Körpers auf Primitivität bedienen wir uns des wohlbekannten Satz von Lagrange aus der Gruppentheorie und geben zunächst ein kleines Lemma an, auf dem der dann folgende Algorithmus basiert.

Lemma 6.7. *Sei $u \in \mathbb{F}_q$ und*

$$q-1 \;=\; p_1^{\nu_1} \cdot \ldots \cdot p_r^{\nu_r}$$

die Primfaktorzerlegung von $q-1$. Definiere für alle $i = 1, \ldots, r$

$$\bar{n}_i \;:=\; \frac{q-1}{p_i} \;=\; p_1^{\nu_1} \cdot \ldots \cdot p_{i-1}^{\nu_{i-1}} \cdot p_i^{\nu_i - 1} \cdot p_{i+1}^{\nu_{i+1}} \cdot \ldots \cdot p_r^{\nu_r}\,.$$

Dann gilt: u ist primitiv genau dann, wenn

$$u^{\bar{n}_i} \;\neq\; 1 \quad \forall i = 1, \ldots, r\,.$$

Beweis. Per definitionem der Primitivität klar. □

Es bleibt jedoch immer noch offen diese r Potenzierungen möglichst gut zu organisieren. Nehmen wir an, die Primzahlen sind in der Primfaktorzerlegung

$$q-1 \;=\; p_1^{\nu_1} \cdot \ldots \cdot p_r^{\nu_r}$$

aufsteigend sortiert, also $p_1 < p_2 < \ldots < p_r$, so hat sich als besonders hilfreich erwiesen, die Potenzen $\bar{n}_i$ auf Basis der Potenzen

- $d := \mathrm{ggT}\{\bar{n}_i : \; i = 1, \ldots, r\}$ und
- $d' := \mathrm{ggT}\{\frac{\bar{n}_i}{d} : \; i = 1, \ldots, r-1\}$

durchzuführen und die bereits berechneten Potenzen zu nutzen, wie nachstehendes Beispiel veranschaulicht.

Beispiel 6.8. Sei $u \in \mathbb{F}_{3^{10}}$. Da

$$3^{10} - 1 = \; 2^{\,3} \cdot 11^2 \cdot 61\,,$$

sind folgende Potenzen von u zu berechnen:

$$\begin{aligned}
\bar{n}_1 &= 2^2 \cdot 11^2 \cdot 61 \;=\; 29524\,,\\
\bar{n}_2 &= 2^3 \cdot 11 \;\cdot 61 \;=\; 5368\,,\\
\bar{n}_3 &= 2^3 \cdot 11^2 \;\;\;\;\;\;\;\; =\; 968\,.
\end{aligned}$$

Wir sehen jedoch dass die Potenzen

$$d := \mathrm{ggT}\{\bar{n}_1, \bar{n}_2, \bar{n}_3\} = 2^2 \cdot 11 = 44$$

und

$$d' := \mathrm{ggT}\{\tfrac{\bar{n}_1}{d}, \tfrac{\bar{n}_2}{d}\} = 61$$

uns die Arbeit erheblich erleichtern können: Wir berechnen $v := u^d = u^{44}$ und $w := v^{d'} = v^{61}$ separat, so schreiben sich die restlichen Potenzen wie folgt:

$$\begin{aligned} u^{\bar{n}_3} &= w^{11}\,, \\ u^{\bar{n}_2} &= v^2\,, \\ u^{\bar{n}_1} &= v^2 \cdot v^9\,. \end{aligned}$$

Selbstverständlich kann man den Test auf Primitivität bereits abbrechen, falls $v = 1$ oder $w = 1$. Es ist klar, dass in diesem Beispiel obiges Vorgehen eine erhebliche Verkleinerung der zu berechnenden Potenzen liefert, die jedoch nicht in allen Fällen erwartet werden kann. ▲

In nachstehender Implementierung sind die separat aufgelisteten Potenzen d mit `commonBarFactor` und d' mit `commonBiggestBarFactor` bezeichnet und werden in p-adischer bzw. binärer Darstellung erwartet. Ferner werden die restlichen $\bar{n}_i$s in `barFactors` bereits in p-adischer bzw. binärer Darstellung als ein einziges `int`-Array übergeben, wobei die jeweilige Länge der einzelnen Faktoren in dem `int`-Array `lenBarFactors` zu hinterlegen ist. Wie im Quelltext bemerkt, wird die Exponentiation p-adisch durchgeführt (siehe Listing 6.13), falls `matCharac` ungleich `0` ist, ansonsten binär (siehe Listing 6.14), wobei natürlich sicherzustellen ist, dass die Potenzen in passender Darstellung vorliegen.

Listing 6.15: Aus `../Sage/enumeratePCNs.c`

```
666 /**
667  * Test if element is primitive.
668  *
669  * !! if matCharac is Zero, all powers are assumed as binary arrays !!
670  *
671  * !! fff,ffTmp,ffTmp2,ffTmp3,ffRet must be malloced !!
672  * !! x is NOT modified !!
673  */
674 inline bool isPrimitive(struct FFElem *ff, struct FFElem *mipo,
675         int m,
676         int *barFactors, int *lenBarFactors, int countBarFactors,
677         int *commonBarFactor, int lenCommonBarFactor,
678         int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
679         struct FFElem **matCharac,
680         struct FFElem *fff, struct FFElem *ffff, struct FFElem *ffTmp,
681         struct FFElem *ffTmp2, struct FFElem *ffRet,
682         int *tmp, int *multTable, int *addTable){
683     int i;
684     int curPos = 0;
685     bool binarySqM = (matCharac == 0);
686     struct FFElem *ffSwitch = 0;
687
688     copyFFElem(ff,fff);
689     // all barFactors are power of commonBarFactor
```

```
690     if(binarySqM)
691         powerFFElemSqM(fff,mipo,ffTmp,
692                 m,commonBarFactor,lenCommonBarFactor,
693                 tmp,ffTmp2,
694                 multTable,addTable);
695     else
696         powerFFElem(fff,mipo,ffTmp,
697                 m,commonBarFactor,lenCommonBarFactor,
698                 matCharac,tmp,ffTmp2,
699                 multTable,addTable);
700     if(isOne(ffTmp)) return false;
701     //switch ffTmp and fff
702     ffSwitch = fff; fff = ffTmp; ffTmp = ffSwitch;
703     copyFFElem(fff,ffff);
704     //test first barFactor
705     if(binarySqM)
706         powerFFElemSqM(ffff,mipo,ffTmp,
707                 m,barFactors,lenBarFactors[0],
708                 tmp,ffTmp2,
709                 multTable,addTable);
710     else
711         powerFFElem(ffff,mipo,ffTmp,
712                 m,barFactors,lenBarFactors[0],
713                 matCharac,tmp,ffTmp2,
714                 multTable,addTable);
715     if(isOne(ffTmp)) return false;
716     curPos += lenBarFactors[0];
717     //test further factors which are powers of commonBiggestBarFactor
718     //so first, calc y^commonBiggestBarFactor
719     copyFFElem(fff, ffff);
720     if(binarySqM)
721         powerFFElemSqM(ffff,mipo,ffTmp,
722                 m,commonBiggestBarFactor,lenCommonBiggestBarFactor,
723                 tmp,ffTmp2,
724                 multTable,addTable);
725     else
726         powerFFElem(ffff,mipo,ffTmp,
727                 m,commonBiggestBarFactor,lenCommonBiggestBarFactor,
728                 matCharac,tmp,ffTmp2,
729                 multTable,addTable);
730     if(isOne(ffTmp)) return false;
731     ffSwitch = fff; fff = ffTmp; ffTmp = ffSwitch;
732     for(i=1;i<countBarFactors;i++){
733         // copy z (fff) to ffff
734         copyFFElem(fff,ffff);
735         // *** ffff == fff == y^commonBiggestBarFactor
736         if(binarySqM)
737             powerFFElemSqM(ffff, mipo, ffTmp,
738                     m,barFactors+curPos, lenBarFactors[i],
739                     tmp,ffTmp2,
740                     multTable,addTable);
741         else
742             powerFFElem(ffff, mipo, ffTmp,
743                     m,barFactors+curPos, lenBarFactors[i],
```

```
744                     matCharac,tmp,ffTmp2,
745                     multTable,addTable);
746
747         if(i>1){
748             multiplyFFElem(ffRet,ffTmp,ffTmp2,mipo,
749                     tmp,m,multTable,addTable);
750         }else{
751             ffSwitch = ffTmp2; ffTmp2 = ffTmp; ffTmp = ffSwitch;
752         }
753         if(isOne(ffTmp2)) return false;
754         curPos += lenBarFactors[i];
755         ffSwitch = ffRet; ffRet = ffTmp2; ffTmp2 = ffSwitch;
756     }
757     return true;
758 }
```

6.3 | Frobenius-Auswertung und Test auf vollständige Erzeuger-Eigenschaft

6.3.1 | Frobenius-Auswertung

Sei wie immer $F = \mathbb{F}_q$ ein endlicher Körper und $E = \mathbb{F}_{q^m}$ eine Körpererweiterung. Sei $\mathcal{C}_{k,t}$ ein verallgemeinerter Kreisteilungsmodul über F (vgl. Definition 5.5) und $u \in E$ ein Element, das wir als vollständigen Erzeuger in Betracht ziehen (vgl. Definition 5.6). Nach **??** (3) ist u genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$, wenn

$$\mathrm{Ord}_{q^d}(u) \;=\; \Phi_{\nu(k),\,\frac{kt}{\nu(k)d}} \qquad \forall d \mid \tfrac{kt}{\nu(k)}\,.$$

Folglich müssen wir, um q-Ordnungen berechnen zu können, in der Lage sein, für beliebige Zwischenkörper $F \mid K \mid E$ von Grad d über F und beliebige $f(x) \in K[x]$

$$f(\sigma^d)(u) \in E$$

auswerten zu können, wobei wieder $\sigma : \bar{F} \to \bar{F}, x \mapsto x^q$ den Frobenius über F bezeichne. Bleibt die Frage, wie die verschiedenen Zwischenkörper mit Hilfe der `FFElems` gelesen werden können. Da wir E jedoch stets als Erweiterung über dem zu Grunde liegenden Primkörper betrachten (vgl. Unterabschnitt 6.1.1) ist dies völlig unklar. Daher umgehen wir dieses Problem und betrachten eine beliebige Einbettung von K in E. Die Frage, ob man damit immer noch q-Ordnungen berechnen kann, beantwortet nachstehendes Lemma.

Lemma 6.9. *Seien $F \mid K \mid E$ ein Turm endlicher Körper mit $[K : F] = d$ und $\sigma : \bar{F} \to \bar{F}$ der Frobenius von F. Sei $f(x) \in K[x]$ ein Polynom und $u \in E$. Für je zwei injektive Körperhomomorphismen $g, h : K \to E$ ist entweder*

$$h(f)(\sigma^d)(u) \;=\; 0 \quad \textit{und} \quad g(f)(\sigma^d)(u) \;=\; 0$$

oder

$$h(f)(\sigma^d)(u) \;\neq\; 0 \quad \textit{und} \quad g(f)(\sigma^d)(u) \;\neq\; 0\,,$$

wobei $h(f) \in E[x]$ koeffizientenweise zu lesen ist.
Mit anderen Worten hängt also die Frage, ob eine Frobenius-Auswertung 0 ist oder nicht, nicht von der Wahl der konkreten Einbettung ab.

Müsste man hier ausführlicher argumentieren?

Beweis. Aufgrund der Eindeutigkeit endlicher Körper (z.B. **??**) unterscheiden sich zwei Einbettungen $g, h : K \to E$ lediglich um einen Automorphismus $a : E \to E$, also $h = a \circ g$. Dies beweist aber bereits die Behauptung. □

Damit können wir uns erstmal davon ausgehen, dass die zu betrachtenden Polynome bereits in $E[x]$ liegen; also vom Typ `FFPoly` sind. Analog zu Listing 6.13 wird auch hier das Potenzieren durch Matrixmultiplikation beschrieben, wobei sicherzustellen ist, dass die maximal auftretende Matrixpotenz vorhanden ist, d.h. übergibt man ein Polynom `poly` vom Grad k, so muss `mats` als Array bestehend aus `FFElem*` von Länge $m \cdot k$ sein, wobei m wiederum den Grad der Erweiterung von E über dem Primkörper meint. Das bedeutet insbesondere, dass die erste Matrix in `mats` die Darstellungsmatrix zu σ^1 ist und der Fall $\sigma^0 = \mathrm{id}$ separat betrachtet werden muss (vgl. Zeile `791` in Listing 6.16).

Im Hinblick auf das Berechnen von q-Ordnungen, wo ein Körperelement meist mehr als einmal einer Frobenius-Auswertung unterzogen werden muss, haben wir die Möglichkeit bereitgestellt, bereits durchgeführte Matrixmultiplikationen in `matmulCache` zu speichern. Das Array `matmulCacheCalced` gibt dabei an, welche Stellen in `matmulCache` bereits berechnet wurden. Selbstredend wird dieser Zwischenspeicher durch die Ausführung von `applyFrob` fortwährend aktualisiert.

Listing 6.16: Aus `../Sage/enumeratePCNs.c`

```
773 /*
774  * calculates g(sigma^frobPower)(x) where g is a polynomial
775  * and sigma the frobenius
776  * application of frobenius is given by mats
777  */
778 inline void applyFrob(struct FFElem *ff, struct FFElem *mipo,
779         struct FFPoly *poly,
780         struct FFElem **mats,
781         int frobPower, struct FFElem *ret,
782         int m, int *tmp, struct FFElem *ffTmp, struct FFElem *ffTmp2,
783         struct FFElem **matmulCache, bool *matmulCacheCalced,
784         int *multTable, int *addTable){
785     int i,j;
786
787     ret->len = 0;
788     for(i=0;i<poly->lenPoly;i++){
789         if(poly->poly[i]->len == 0) continue;
790         j = i*frobPower-1;
791         if(i>0 && matmulCacheCalced[j] == true){
792             multiplyFFElem(matmulCache[j],poly->poly[i],
793                     ffTmp, mipo,
794                     tmp,m,multTable,addTable);
795             addFFElem(ret,ffTmp,ret,tmp,multTable,addTable);
796         }else{
797             if(i>0){
798                 matmul(mats+j*m, ff, ffTmp, m, multTable,addTable);
799                 //update matmulCache
```

```
800                 copyFFElem(ffTmp, matmulCache[j]);
801                 matmulCacheCalced[j] = true;
802             }else{
803                 copyFFElem(ff,ffTmp);
804             }
805             //go on and multiply ffTmp with current coefficient
806             multiplyFFElem(ffTmp, poly->poly[i],
807                     ffTmp2, mipo,
808                     tmp,m,multTable,addTable);
809             addFFElem(ret,ffTmp2,ret,tmp,multTable,addTable);
810         }
811     }
812 }
```

Falls bereits klar ist, dass für ein gegebenes Element nur eine Frobenius-Auswertung vollzogen wird, so ist der `matmulCache` überflüssig und führt zur Variante `applyFrob_noCache`, die ansonsten identisch zu obigem ist.

Listing 6.17: Aus `../Sage/enumeratePCNs.c`

```
827 /*
828  * calculates g(sigma^frobPower)(x) where g is a polynomial
829  * and sigma the frobenius
830  * application of frobenius is given by mats
831  */
832 inline void applyFrob_noCache(struct FFElem *ff, struct FFElem *mipo,
833         struct FFPoly *poly,
834         struct FFElem **mats,
835         int frobPower, struct FFElem *ret,
836         int m, int *tmp, struct FFElem *ffTmp, struct FFElem *ffTmp2,
837         int *multTable, int *addTable){
838     int i,j;
839     ret->len = 0;
840 
841     for(i=0;i<poly->lenPoly;i++){
842         if(poly->poly[i]->len == 0) continue;
843         if(i>0){
844             j = i*frobPower-1;
845             matmul(mats+j*m, ff, ffTmp, m, multTable,addTable);
846         }else{
847             copyFFElem(ff,ffTmp);
848         }
849         multiplyFFElem(ffTmp, poly->poly[i],
850                 ffTmp2, mipo,
851                 tmp,m,multTable,addTable);
852         addFFElem(ret,ffTmp2,ret,tmp,multTable,addTable);
853     }
854 }
```

6.3.2 Testen von vollständigen Erzeugern

Wie bereits erwähnt ist $u \in \mathbb{F}_{q^m}$ über $\mathbb{F}_q$ genau dann ein vollständiger Erzeuger eines verallgemeinerten Kreisteilungsmoduls $\mathcal{C}_{k,t}$, wenn

$$\mathrm{Ord}_{q^d}(u) \;=\; \Phi_{\nu(k),\,\frac{kt}{\nu(k)d}} \qquad \forall d \mid \tfrac{kt}{\nu(k)}\,.$$

Analog zum Primitivitätstest reicht es, lediglich maximale Kofaktoren des jeweiligen verallgemeinerten Kreisteilungspolynoms zu testen, wie nachstehendes Lemma beschreibt.

Lemma 6.10. *Seien $u \in \mathbb{F}_{q^m}$ und $\Phi_{k,t}(x) \in \mathbb{F}_q[x]$ ein verallgemeinertes Kreisteilungspolynom. Sei ferner*

$$\Phi_{k,t}(x) \;=\; f_1(x)^{\nu_1} \cdot \ldots \cdot f_r(x)^{\nu_r} \quad \in \mathbb{F}_q[x]$$

die vollständige Faktorisierung von $\Phi_{k,t}$ über $\mathbb{F}_q$ und bezeichne $F_i(x) := \frac{\Phi_{k,t}(x)}{f_i(x)}$ den jeweiligen maximalen Kofaktor von f_i in $\Phi_{k,t}$ für alle $i = 1, \ldots, r$. Seien zuletzt $h : \mathbb{F}_q \to \mathbb{F}_{q^m}$ ein injektiver Körperhomomorphismus und $\sigma : \bar{\mathbb{F}}_q \to \bar{\mathbb{F}}_q, x \mapsto x^q$ der Frobenius von $\mathbb{F}_q$, so ist $\mathrm{Ord}_q(u) = \Phi_{k,t}$ genau dann, wenn

$$h(\Phi_{k,t})(\sigma)(u) \;=\; 0 \qquad \textit{und} \qquad h(F_i)(\sigma)(u) \;\neq\; 0 \quad \forall i = 1, \ldots, r\,.$$

Beweis. Klar per definitionem der q-Ordnung und Lemma 6.9. □

Nun können wir auf diese Weise leicht eine Implementierung eines Tests auf vollständige Erzeuger-Eigenschaft angeben, wenn wir davon ausgehen, dass die Berechnung der maximalen Kofaktoren bereits geschehen ist. In Listing 6.18 ist also sicherzustellen, dass in dem Array `polys` sowohl das verallgemeinerte Kreisteilungspolynom, als auch alle maximalen Kofaktoren auftauchen. Das Array `evalToZero` gibt dabei an, ob bei Vorliegen eines vollständigen Erzeugers die Auswertung am jeweiligen Polynom 0 ergibt (`true`) oder nicht (`false`). Der Rückgabewert der Funktion ist selbstredend ein `bool` mit der Information, ob das getestete Element `ff` ein vollständiger Erzeuger dieses Kreisteilungsmoduls ist (`true`) oder nicht (`false`).

Listing 6.18: Aus `../Sage/enumeratePCNs.c`

```
912 inline bool testSubmod(struct FFElem *ff, struct FFElem *mipo,
913         struct FFPoly **polys,
914         int polysCount, bool *evalToZero,
915         struct FFElem **mats, int *frobPowers,
916         int m, int *tmp,
917         struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
918         struct FFElem **matmulCache, bool *matmulCacheCalced,
919         int *multTable, int *addTable){
920     int i;
921     int goodCounter = 0;
922     for(i=0;i<polysCount;i++){
923         applyFrob(ff,mipo,
924                 polys[i],
925                 mats,frobPowers[i], ffTmp,
926                 m,tmp,ffTmp2,ffTmp3,
927                 matmulCache,matmulCacheCalced,
928                 multTable,addTable);
929         if( isZero(ffTmp) == evalToZero[i] ){
930             goodCounter++;
```

```
931         }else{
932             return false;
933         }
934     }
935     if(goodCounter == polysCount){
936         return true;
937     }
938     return false;
939 }
```

Ferner bieten wir die Möglichkeit ein Element auf vollständige Erzeuger-Eigenschaft für mehrere verallgemeinerte Kreisteilungsmoduln zu testen, wie Listing 6.19 zeigt. `decompCount` ist dabei die Anzahl der zu testenden verallgemeinerten Kreisteilungsmoduln und das Array `polysCountPerDecomp` gibt die Anzahl der Polynome für den jeweiligen Kreisteilungsmodul an. Das Array `bool * toTestIndicator` legt fest, welche Kreisteilungsmodule getestet werden. Der Rückgabewert – anders als in Listing 6.18 – ist ein `int`, der die Werte -1, falls `ff` kein vollständiger Erzeuger der getesteten Kreisteilungsmoduln ist, oder i, falls `ff` gerade vollständiger Erzeuger des i-ten getesteten Kreisteilungsmoduls ist, annimmt. Ferner bricht die Funktion ab, falls `ff` ein vollständiger Erzeuger ist, da es klar sein sollte, dass diese Eigenschaft lediglich für *einen* verallgemeinerten Kreisteilungsmodul zutreffen kann.

Listing 6.19: Aus `../Sage/enumeratePCNs.c`

```
861 inline int testAllSubmods(struct FFElem *ff, struct FFElem *mipo,
862         int decompCount, struct FFPoly **polys,
863         int *polysCountPerDecomp, bool *evalToZero,
864         struct FFElem **mats, int *frobPowers, bool *toTestIndicator,
865         int m, int *tmp,
866         struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
867         struct FFElem **matmulCache, bool *matmulCacheCalced,
868         int *multTable, int *addTable){
869     if(ff->len == 0) return -1;
870     int i,j,k;
871     int goodCounter = 0;
872     int curDecompPosition = 0;
873     for(i=0;i<decompCount;i++){
874         if(toTestIndicator != 0 && toTestIndicator[i] == false){
875             curDecompPosition += polysCountPerDecomp[i];
876             continue;
877         }
878         goodCounter = 0;
879         for(j=0;j<polysCountPerDecomp[i];j++){
880             applyFrob(ff,mipo,
881                     polys[curDecompPosition+j],
882                     mats,frobPowers[curDecompPosition+j], ffTmp,
883                     m,tmp,ffTmp2,ffTmp3,
884                     matmulCache,matmulCacheCalced,
885                     multTable,addTable);
886             if( isZero(ffTmp) == evalToZero[curDecompPosition+j] ){
887                 goodCounter++;
888             }else break;
889         }
890         if(goodCounter == polysCountPerDecomp[i]){
```

```
891             return i;
892         }
893         curDecompPosition += polysCountPerDecomp[i];
894     }
895     return -1;
896 }
```

6.4 Implementierung der gezielten Enumeration

6.4.1 Enumeration eines verallgemeinerten Kreisteilungsmoduls

Sei $E := \mathbb{F}_{q^m}$ über $F := \mathbb{F}_q$ eine Körpererweiterung endlicher Körper. Die Frage nach einer Enumeration aller vollständig normaler Elemente dieser Erweiterung lässt sich nach dem Zerlegungssatz (Satz 5.9) auf die separate Enumeration von verallgemeinerten Kreisteilungsmoduln zurückführen. Daher starten wir mit einem verallgemeinerten Kreisteilungsmodul $\mathcal{C}_{k,t}$ über $\mathbb{F}_q$. Sicherlich könnte man alle q^m Elemente von E testen, ob sie vollständige Erzeuger von $\mathcal{C}_{k,t}$ sind, was jedoch einen unnötig großen Aufwand darstellen würde. Sei nämlich $u \in E$ ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$, so erhalten wir alle weiteren Elemente dieses Kreisteilungsmoduls durch Anwendung von Korollar 3.40, was wir hier in passender Notation noch einmal formulieren möchten.

Lemma 6.11. *Sei $u \in E$ ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$ über F. Dann gilt*

$$\mathcal{C}_{k,t} \;=\; \left\{ f(\sigma)(u) :\; f(x) \in F[x]_{<\varphi(k)\,t},\; \operatorname{ggT}(f, \Phi_{k,t}) = 1 \right\},$$

wobei wiederum σ den Frobenius von F und φ die Eulersche φ-Funktion notieren.

Beweis. Korollar 3.40 mit der Erkenntnis, dass $\deg(\Phi_{k,t}) = \varphi(k)\,t$. □

Nun ist klar, wie wir ausgehend von einem Erzeuger alle weiteren generieren können: Sei $u \in E$ ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$ über F, so berechnen wir iterativ $v := f(\sigma)(u)$ für alle $f \in F[x]_{<\varphi(k)t}$ mit `applyFrob_noCache` (Listing 6.17) und testen anschließend v auf vollständige Erzeuger-Eigenschaft mit `testSubmod` (Listing 6.18). Auf eine Berechnung von $\operatorname{ggT}(f, \Phi_{k,t})$ verzichten wir, da wir `testSubmod` ohnehin ausführen müssen und damit durch die Kenntnis des ggT keinen Vorteil erlangen würden. Die Generierung der fs erfolgt direkt in Listing 6.20, wobei die Elemente aus F wieder mittels eines injektiven Körperhomomorphismus $F \to E$ als `FFElem*`-Array namens `elementsF` übergeben werden.

Müsste man hier ausführlicher argumentieren?

Bemerkung 6.12. Wiederum überzeuge man sich kurz mit der gleichen Argumentation wie in Lemma 6.9, dass ein Endomorphismus $E \to E$ lediglich die Erzeuger eines verallgemeinerten Kreisteilungsmoduls permutiert.

Wie erwähnt müssen wir dieses Verfahren natürlich mit einem vollständigen Erzeuger starten. Es ist sicherzustellen, dass sich dieser am aktuellen Knoten der Liste `struct Node *root` befindet.

Listing 6.20: Aus ../Sage/enumeratePCNs.c

```
953 inline void calcSubmoduleElements(struct Node *root,
954         struct FFElem *mipo,
955         int maxLenPoly,
956         int *genCounts, int curGen,
957         struct FFPoly **polys, int polysCount, bool *evalToZero,
958         struct FFElem **mats, int matLen, int *frobPowers,
959         struct FFElem **elementsF,
960         int m, int q, int *tmp,
961         struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
962         struct FFElem *ffTmp4,
963         struct FFElem **matmulCache, bool *matmulCacheCalced,
964         int *multTable, int *addTable){
965     int i,j;
966     struct Node *curRoot = root;
967     struct FFElem *ff = root->ff;
968     int *curPoly = malloc( maxLenPoly*sizeof(int) );
969     struct FFPoly *curFPoly = malloc( sizeof(struct FFPoly) );
970     curFPoly->poly = malloc( maxLenPoly*sizeof(struct FFElem*) );
971     curFPoly->lenPoly = 0;
972 
973     initPoly(curPoly,maxLenPoly);
974     curPoly[0] = 2;
975     int curLenPoly = 1;
976     if( q == 2 && maxLenPoly > 1){
977         curLenPoly = 2;
978         curPoly[0] = 0;
979         curPoly[1] = 1;
980     }
981     if(q != 2 || maxLenPoly > 1){
982         while(true){
983             //setup curFPoly
984             for(i=0;i<curLenPoly;i++)
985                 curFPoly->poly[i] = elementsF[curPoly[i]];
986             curFPoly->lenPoly = curLenPoly;
987             //apply Frobenius
988             applyFrob_noCache(ff,mipo,
989                     curFPoly,
990                     mats,1, ffTmp, //return value
991                     m,tmp,ffTmp2,ffTmp3,
992                     multTable,addTable);
993             //test generated element
994             for(i=0;i<matLen;i++) matmulCacheCalced[i] = false;
995             if(testSubmod(ffTmp, mipo,
996                     polys,polysCount,evalToZero,
997                     mats,frobPowers,m,tmp,
998                     ffTmp2,ffTmp3,ffTmp4,
999                     matmulCache,matmulCacheCalced, multTable,addTable)){
1000                 curRoot = appendToEnd(curRoot,ffTmp,m);
1001                 genCounts[curGen]++;
1002             }
1003             //generate next element
1004             curPoly[0] += 1;
1005             if( curPoly[0] == q ){
```

```
1006                for(i=0;i<maxLenPoly-1 && curPoly[i]==q;i++){
1007                    curPoly[i] = 0;
1008                    curPoly[i+1] += 1;
1009                }
1010                if(i+1>curLenPoly)
1011                    curLenPoly = i+1;
1012                if( curPoly[maxLenPoly-1]==q){
1013                    break;
1014                }
1015            }
1016        }
1017    }
1018    free(curPoly);
1019    free(curFPoly->poly);
1020    free(curFPoly);
1021 }
```

Wie man in obigem Listing erkennt, startet die Erzeugung der Polynome aus $F[x]_{\varphi(k)t}$ beim Polynom $2 \in F[x]$ (falls es die Charakteristik zulässt), da $1 \in F[x]$ ja wieder $(1)(\sigma)(u) = \mathrm{id}(u) = u$ liefert. `maxLenPoly` gibt dabei die maximale Länge der zu betrachtenden Polynome an (in hiesiger Notation also `maxLenPoly`$= \varphi(k)t + 1$). Die Polynome selbst werden in zwei Schritten erzeugt: Sei $l :=$`maxLenPoly`, so durchläuft das `int`-Array `curPoly` alle Elemente aus $\mathbb{Z}_q^l$. Das korrekte Polynom in $E[x]$ wird dann durch einsetzen jeder Stelle dieses Tupels aus $\mathbb{Z}_q^l$ in die `elementsF` erzeugt und in `curFPoly` gespeichert.

Die Anzahl der berechneten Erzeuger werden im `int`-Array `genCounts` an der Stelle `curGen` gespeichert und da unsere Suche auf vollständig normale Elemente abzielt, werden die konkreten Erzeuger durch `appendToEnd` an die verkettete Liste `struct Node *root` angehängt und damit für späteres Zusammensetzen gespeichert.

Verkettete Listen zum Speichern berechneter vollständiger Erzeuger

Die verkettete Liste ist dabei wie folgt aufgebaut.

Listing 6.21: Aus `../Sage/enumeratePCNs.c`

```
175 struct Node {
176     struct FFElem *ff;
177     struct Node *next;
178 };
```

Ebenfalls ist das Anheften eines Elements ans Ende der Liste wie man es erwartet, wobei zu bemerken gilt, dass der neue Endknoten zurückgegeben wird. Auf diese Weise muss nicht bei jedem Anheften die komplette Liste durchlaufen werden. Das Element `struct FFElem *element` wird dabei kopiert, so dass es anschließend weiterverwendet werden kann und die Liste unverändert bleibt (vgl. Anwendung in Listing 6.20).

Listing 6.22: Aus `../Sage/enumeratePCNs.c`

```
180 /**
181  * appends element to end of root, where element is copied to new FFElem.
182  */
```

```
183 inline struct Node *appendToEnd(struct Node *root, struct FFElem *element,int m){
184     struct Node *nextNode = root;
185     if( nextNode != 0){
186         while(nextNode->next != 0){
187             nextNode = nextNode->next;
188         }
189         if( nextNode->ff != 0){
190             nextNode->next = malloc( sizeof(struct Node) );
191             nextNode = nextNode->next;
192         }
193         if( nextNode != 0){
194             nextNode->next = 0;
195             nextNode->ff = mallocFFElem(m);
196             copyFFElem(element,nextNode->ff);
197             return nextNode;
198         }
199     }
200     return NULL;
201 }
```

Wie üblich in C, ist es hilfreich das Freigeben von Speicher in eine eigene Funktion zu setzen.

Listing 6.23: Aus `../Sage/enumeratePCNs.c`

```
203 inline void freeNode(struct Node* head){
204     struct Node *next_n = NULL;
205     struct Node *tmp_n = NULL;
206     for(tmp_n=head; tmp_n !=NULL; ){
207         next_n = tmp_n->next;
208         freeFFElem(tmp_n->ff);
209         free(tmp_n);
210         tmp_n = next_n;
211     }
212     head = 0;
213 }
```

6.4.2 | Dynamische Enumeration des größten Kreisteilungsmoduls

Da der Zerlegungssatz (Satz 5.9) nicht immer eine echte Zerlegung liefert (sich also alle vollständig normalen Elemente auf einen einzigen Modul konzentrieren) und in vielen Zerlegungen ein verallgemeinerter Kreisteilungsmodul vorkommt, der verglichen mit den anderen Moduln dieser Zerlegung, besonders viele Elemente enthält, hat sich die Speicherung *aller* Erzeuger als schlecht erwiesen. Daher sind wir dazu übergegangen, den größten Kreisteilungsmodul dynamisch zu enumerieren. Das bedeutet, dass alle anderen verallgemeinerten Kreisteilungsmoduln vorab durch `calcSubmoduleElements` (Listing 6.20) behandelt werden. Bei der Enumeration des größten nutzen wir dann diese Informationen und setzen die gefundenen Erzeuger zu einem vollständig normalen Element zusammen. Dies können wir dann auf Primitivität durch `isPrimitive` (Listing 6.15) testen und abschließend verwerfen, da es uns ja nur auf eine Enumeration und nicht auf die konkrete Angabe der vollständig normalen und primitiven Elemente ankommt.

Die bereits berechneten vollständigen Erzeuger werden durch das Array von Listen `struct Node **roots` übergeben. `decompCount` gibt dabei die Anzahl aller (also inklusive des größten) verallgemeinerten Kreisteilungsmoduln an. Alle anderen Variablen wurden bereits in den vorherigen Funktionen erklärt, wobei noch bemerkt werden sollte, dass diesmal die Erzeugung der Polynome bei $1 \in F[x]$ startet, da der bereits gefundene Erzeuger des größten Kreisteilungsmoduls auch Teil einer gültigen Kombination zu einem vollständig normalen Element ist. Dieser „Fehler" in der Berechnung der Anzahl `genCounts` wird in Zeile `1156` am Ende der Funktion korrigiert. Der Erzeuger selbst befindet sich wieder am aktuellen Knoten der der letzten Liste des Arrays `roots`, da die Datenstrukturen so aufgebaut werden, dass dieser größte verallgemeinerte Kreisteilungsmodul der letzte ist (siehe Zeilen `1062` und `1064`).

Listing 6.24: Aus `../Sage/enumeratePCNs.c`

```
1030 /**
1031  * Processes last submodule as others before, but does not save generated
1032  * elements. Cycles through already generated elements and tests for
1033  * primitivity:
1034  *
1035  * !! all temporary variables are generated inside !!
1036  */
1037 unsigned long long processLastSubmoduleAndTestPrimitivity(struct Node **roots,
1038         struct FFElem *mipo, int decompCount,
1039         int maxLenPoly,
1040         int *genCounts,
1041         struct FFPoly **polys, int polysCount, bool *evalToZero,
1042         struct FFElem **mats, int matLen, int *frobPowers,
1043         struct FFElem **elementsF,
1044         int m, int q,
1045         int *barFactors, int *lenBarFactors, int countBarFactors,
1046         int *commonBarFactor, int lenCommonBarFactor,
1047         int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
1048         struct FFElem **matCharac,
1049         struct FFElem **matmulCache, bool *matmulCacheCalced,
1050         int *multTable, int *addTable){
1051     //generate temporary variables
1052     struct FFElem *fff = mallocFFElem(m);
1053     struct FFElem *ffff = mallocFFElem(m);
1054     struct FFElem *ffTmp = mallocFFElem(m);
1055     struct FFElem *ffTmp2 = mallocFFElem(m);
1056     struct FFElem *ffTmp3 = mallocFFElem(m);
1057     struct FFElem *ffTmp4 = mallocFFElem(m);
1058     struct FFElem *ffTmp5 = mallocFFElem(m);
1059     int *tmp = malloc(m*sizeof(int));
1060 
1061     int i,j;
1062     int curGen = decompCount-1;
1063     struct Node **curRoots = malloc( decompCount*sizeof(struct Node*) );
1064     struct FFElem *ff = roots[curGen]->ff;
1065     int *curPoly = malloc( maxLenPoly*sizeof(int) );
1066     struct FFPoly *curFPoly = malloc( sizeof(struct FFPoly) );
1067     curFPoly->poly = malloc( maxLenPoly*sizeof(struct FFElem*) );
1068     curFPoly->lenPoly = 0;
1069 
1070     initPoly(curPoly,maxLenPoly);
```

```
1071    curPoly[0] = 1;
1072    int curLenPoly = 1;
1073
1074    unsigned long long pcn = 0;
1075    while(true){
1076        //setup curFPoly
1077        for(i=0;i<curLenPoly;i++)
1078            curFPoly->poly[i] = elementsF[curPoly[i]];
1079        curFPoly->lenPoly = curLenPoly;
1080        //apply Frobenius
1081        applyFrob_noCache(ff,mipo,
1082                curFPoly,
1083                mats,1, fff, //return value
1084                m,tmp,ffTmp,ffTmp2,
1085                multTable,addTable);
1086        //test generated element
1087        for(i=0;i<matLen;i++) matmulCacheCalced[i] = false;
1088        if(testSubmod(fff, mipo,
1089                polys,polysCount,evalToZero,
1090                mats,frobPowers,m,tmp,
1091                ffTmp,ffTmp2,ffTmp3,
1092                matmulCache,matmulCacheCalced, multTable,addTable)){
1093            genCounts[curGen]++;
1094            // build element as sum of already calced Nodes and ffTmp
1095            for(i=0;i<decompCount;i++) curRoots[i] = roots[i];
1096            // cycle through Nodes, build element and test primitivity
1097            while(true){
1098                copyFFElem(fff,ffff);
1099                //build element
1100                for(i=0;i<decompCount-1;i++){
1101                    addFFElem(ffff, curRoots[i]->ff, ffff, tmp,
1102                            multTable,addTable);
1103                }
1104                //test primitivity
1105                if(countBarFactors > 0){
1106                    if(isPrimitive(ffff, mipo,m,
1107                                barFactors,lenBarFactors,countBarFactors,
1108                                commonBarFactor,lenCommonBarFactor,
1109                                commonBiggestBarFactor,lenCommonBiggestBarFactor,
1110                                matCharac,
1111                                ffTmp,ffTmp2,ffTmp3,ffTmp4,ffTmp5,
1112                                tmp,multTable,addTable)){
1113                        pcn++;
1114                    }
1115                }
1116
1117                //next element
1118                curRoots[0] = curRoots[0]->next;
1119                if( curRoots[0] == 0 ){
1120                   for(i=0;i<decompCount-1 && curRoots[i]==0;i++){
1121                       curRoots[i] = roots[i];
1122                       curRoots[i+1] = curRoots[i+1]->next;
1123                   }
1124                }
```

```
1125                if( curRoots[decompCount-1] == 0){
1126                   break;
1127                }
1128            }
1129        }
1130        //generate next element
1131        curPoly[0] += 1;
1132        if( curPoly[0] == q ){
1133            for(i=0;i<maxLenPoly-1 && curPoly[i]==q;i++){
1134                curPoly[i] = 0;
1135                curPoly[i+1] += 1;
1136            }
1137            if(i+1>curLenPoly)
1138                curLenPoly = i+1;
1139            if( curPoly[maxLenPoly-1]==q){
1140                break;
1141            }
1142        }
1143    }
1144    free(curPoly);
1145    free(curFPoly->poly);
1146    free(curFPoly);
1147    freeFFElem(fff);
1148    freeFFElem(ffff);
1149    freeFFElem(ffTmp);
1150    freeFFElem(ffTmp2);
1151    freeFFElem(ffTmp3);
1152    freeFFElem(ffTmp4);
1153    freeFFElem(ffTmp5);
1154
1155    //we added first element twice
1156    genCounts[curGen]--;
1157    return pcn;
1158 }
```

Bemerkung 6.13. Wie Zeile `1105` zu erkennen gibt, kann man durch das Setzen von `countBarFactors = 0` den Test auf Primitivität überspringen. Dies ist sinnvoll, wenn man nur an der Anzahl der vollständig normalen Elemente interessiert ist.

Vorbereiten der Enumeration auf Auffinden vollständiger Erzeuger

Alle bisher betrachteten Verfahren basierten immer auf der Annahme, dass bereits ein vollständiger Erzeuger eines Kreisteilungsmoduls bereits gefunden ist. Es ist klar, dass man diese irgendwann suchen muss, was die Funktion `processFiniteField` bewerkstelligt. Gleichzeitig bildet sie den Wrapper, der von `Sage` aufgerufen wird und als Rückgabewert **`unsigned long long`** die Anzahl der primitiven vollständig normalen Elemente trägt. Alle zu übergebenen Parameter werden in `Sage` erzeugt und wurden bereits erklärt.

Listing 6.25: Aus `../Sage/enumeratePCNs.c`

```
1173 unsigned long long processFiniteField(struct FFElem *mipo, int decompCount,
1174         struct FFPoly **polys, int *polysCountPerDecomp,
```

```
1175        bool *evalToZero,
1176        struct FFElem **mats, int matLen, int *frobPowers,
1177        int *genCounts, int m, int charac, int q,
1178        int *barFactors, int *lenBarFactors, int countBarFactors,
1179        int *commonBarFactor, int lenCommonBarFactor,
1180        int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
1181        struct FFElem **matCharac, struct FFElem **elementsF,
1182        int *multTable, int *addTable){
1183    time_t TIME = time(NULL);
1184    int i,j;
1185
1186    //setup temporary variables ------------------------------------------
1187    int *tmp = malloc(m*sizeof(int));
1188    struct FFElem *ff = mallocFFElem(m);
1189    initPoly(ff->el,m);
1190    struct FFElem *ffRet = mallocFFElem(m);
1191    struct FFElem *ffTmp = mallocFFElem(m);
1192    struct FFElem *ffTmp2 = mallocFFElem(m);
1193    struct FFElem *ffTmp3 = mallocFFElem(m);
1194    struct FFElem *ffTmp4 = mallocFFElem(m);
1195
1196    struct FFElem **matmulCache = malloc(matLen*sizeof(struct FFElem));
1197    for(i=0;i<matLen;i++) matmulCache[i] = mallocFFElem(m);
1198    bool *matmulCacheCalced = malloc(matLen*sizeof(bool));
1199
1200    bool *toTestIndicator = malloc(decompCount*sizeof(bool));
1201    struct Node **roots = malloc( decompCount*sizeof(struct Node) );
1202    struct Node **curRoots = malloc(decompCount*sizeof(struct Node*));
1203    for(i=0;i<decompCount;i++){
1204        roots[i] = malloc( sizeof(struct Node) );
1205        roots[i]->ff = 0;
1206        roots[i]->next = 0;
1207        curRoots[i] = roots[i];
1208        toTestIndicator[i] = true;
1209    }
1210    //------------------------------------------------------------------------
1211
1212    int foundCounter = 0;
1213    initPoly(genCounts,decompCount);
1214
1215    // chase for elements ------------------------------------------------
1216    while(true){
1217        for(i=0;i<matLen;i++) matmulCacheCalced[i] = 0;
1218        int curGen = testAllSubmods(ff,mipo,decompCount,
1219                polys,polysCountPerDecomp,evalToZero,
1220                mats,frobPowers,toTestIndicator,
1221                m,tmp,ffTmp,ffTmp2,ffTmp3,
1222                matmulCache,matmulCacheCalced,
1223                multTable,addTable);
1224        if( curGen != -1 ){
1225            if(toTestIndicator[curGen] == true){
1226                genCounts[curGen]++;
1227                appendToEnd(roots[curGen], ff, m);
1228                foundCounter++;
```

```
1229                toTestIndicator[curGen] = false;
1230            }
1231            if(foundCounter == decompCount) break;
1232        }
1233        //generate next element
1234        // (for sure there is a more efficient method)
1235        ff->el[0] += 1;
1236        if( ff->el[0] == charac ){
1237            for(i=0; i<m-1 && ff->el[i]==charac; i++){
1238                ff->el[i] = 0;
1239                ff->el[i+1] += 1;
1240            }
1241            if( ff->el[m-1] == charac )
1242                break;
1243        }
1244        updateFFElem(ff,m);
1245    }
1246    if( foundCounter != decompCount ){
1247        printf("BAAAD␣ERROR!!!␣foundCounter=%i␣<␣decompCount=%i\n",
1248                foundCounter,decompCount);
1249        exit(0);
1250    }
1251    printf("finding␣time:␣%.2f\n", (double)(time(NULL)-TIME));
1252    //-------------------------------------------------------------------------
1253
1254
1255
1256    // Process found elements -----------------------------------------------
1257    int curDecompPosition = 0;
1258    for(i=0;i<decompCount-1;i++){
1259        calcSubmoduleElements(roots[i], mipo,
1260                polys[curDecompPosition]->lenPoly-1, // *** == maxLenPoly
1261                genCounts,i, // *** i == curGen
1262                polys+curDecompPosition, polysCountPerDecomp[i],
1263                evalToZero+curDecompPosition,
1264                mats, matLen, frobPowers+curDecompPosition,
1265                elementsF,
1266                m,q,tmp,
1267                ffTmp,ffTmp2,ffTmp3,ffTmp4,
1268                matmulCache,matmulCacheCalced,
1269                multTable,addTable);
1270        curDecompPosition += polysCountPerDecomp[i];
1271    }
1272    printf("all␣not␣last␣time:␣%.2f\n", (double)(time(NULL)-TIME));
1273    //-------------------------------------------------------------------------
1274
1275    // Process last Decomposition -------------------------------------------
1276    int curGen = decompCount-1;
1277    unsigned long long pcn =
1278        processLastSubmoduleAndTestPrimitivity(roots,mipo,decompCount,
1279            polys[curDecompPosition]->lenPoly-1, // *** == maxLenPoly
1280            genCounts,
1281            polys+curDecompPosition,polysCountPerDecomp[curGen],
1282            evalToZero+curDecompPosition,
```

```
1283            mats,matLen,frobPowers+curDecompPosition,
1284            elementsF,
1285            m,q,
1286            barFactors,lenBarFactors,countBarFactors,
1287            commonBarFactor,lenCommonBarFactor,
1288            commonBiggestBarFactor,lenCommonBiggestBarFactor,
1289            matCharac,
1290            matmulCache,matmulCacheCalced,
1291            multTable,addTable);
1292    //------------------------------------------------------------------------
1293
1294    //free variables
1295    for(i=0;i<decompCount;i++)
1296        freeNode(roots[i]);
1297    free(roots); free(curRoots);
1298
1299    //free temporary variables
1300    free(tmp);
1301    freeFFElem(ff);
1302    freeFFElem(ffRet);
1303    freeFFElem(ffTmp);
1304    freeFFElem(ffTmp2);
1305    freeFFElem(ffTmp3);
1306    freeFFElem(ffTmp4);
1307    for(i=0;i<matLen;i++) freeFFElem(matmulCache[i]);
1308    free(matmulCache);
1309    free(matmulCacheCalced);
1310    free(toTestIndicator);
1311
1312
1313    printf("total␣time:␣%.2f\n", (double)(time(NULL)-TIME));
1314    return pcn;
1315 }
```

Wie zu erkennen ist, erfolgt die Suche nach vollständigen Erzeugern zunächst durch iterative Enumeration aller Elemente. Wurde ein vollständiger Erzeuger gefunden, so wird die jeweilige Stelle des `toTestIndicators` umgeschaltet, wodurch der zugehörige verallgemeinerte Kreisteilungsmodul in `testAllSubmods` nicht mehr berücksichtigt wird. Ist für jeden Kreisteilungsmodul ein vollständiger Erzeuger gefunden werden wie oben beschrieben durch `calcSubmoduleElements` (Listing 6.20) alle, bis auf den letzten, verarbeitet. Dieser wird abschließend separat in Listing 6.24 betrachtet und liefert die Anzahl der primitiven vollständig normalen Elemente.

6.4.3 | Top-Level-Implementierung in Sage

Eingangs wurde zwar erwähnt, dass `Sage` nicht ausreichend performant ist, um die hier angestrebten Ziele zu erreichen, doch wollen wir nicht gänzlich auf die hochsprachlichen Funktionen dieses Computeralgebrasystems verzichten. Insbesondere eignet sich `Sage` hier, die Daten für `processFiniteField` (Listing 6.25) bereitzustellen.

Anwendung des Zerlegungssatzes

Es ist klar, dass am Anfang der Berechnung von primitiv vollständig normalen Elementen einer Erweiterung endlicher Körper stets die Anwendung des Zerlegungssatzes (Satz 5.9) steht.

Listing 6.26: Aus `../Sage/enumeratePCNs.spyx`

```
544 # Application of the Decomposition Theorem (Section 19)
545 # for x^n-1 over F_p^e
546 def decompose(p,e, n):
547     pi = largestDiv(p,n)
548     return decompose_cycl_module(p,e, 1, n/pi, pi)
```

Listing 6.27: Aus `../Sage/enumeratePCNs.spyx`

```
571 # internal application of the Decomposition Theorem
572 # for Phi_k(x^(t*pi)) over F_p^e
573 def decompose_cycl_module(p,e, k,t,pi):
574     if p.divides(k*t): print "ERROR p | kt"
575     #test all prime divisors, start with largest one
576     flag = False
577     for r,l in reversed(factor(t)):
578         if not (r**l).divides(ordn(squarefree(k*t),p**e)):
579             R = largestDiv(r,t)
580             return decompose_cycl_module(p,e, k, t/r, pi) \
581                    + decompose_cycl_module(p,e, k*R, t/R, pi)
582     return [(k,t,pi)]
```

Beispiel 6.14. Wollen wir einmal den Zerlegungssatz auf $E := \mathbb{F}_{3^{20}}$ über $F := \mathbb{F}_3$ anwenden, so rufen wir `decompose(3,1,20)` auf und erhalten

```
[(1, 1, 1), (2, 1, 1), (4, 1, 1), (5, 4, 1)].
```

Umformuliert bedeutet das, dass

$$x^{20} - 1 \;=\; \Phi_1(x)\,\Phi_2(x)\,\Phi_4(x)\,\Phi_5(x^4) \quad \in \mathbb{F}_3[x]\,,$$

eine verträgliche Zerlegung ist. Oder in Termen der erweiterten Kreisteilungsmoduln ist

$$\mathcal{C}_{1,20} \;=\; \mathcal{C}_{1,1} \oplus \mathcal{C}_{2,1} \oplus \mathcal{C}_{4,1} \oplus \mathcal{C}_{5,4}$$

eine verträgliche Zerlegung über $\mathbb{F}_3$. ▲

Die benutzten Funktionen `largestDiv`, `ordn` und `squarefree` sind dabei wie folgt gegeben.

Listing 6.28: Aus `../Sage/enumeratePCNs.spyx`

```
586 # returns the largest power of p dividing n
587 def largestDiv(p,n):
588     l = 0
589     while (p**l).divides(n):
590         l = l+1
591     return p**(l-1);
```

Listing 6.29: Aus ../Sage/enumeratePCNs.spyx

```
538 # computes ordn m(q) = min{ k: q ** k = 1 mod m }
539 def ordn(m,q):
540     if m == 1: return 1
541     for i in range(1,m+1):
542         if (q ** i)%m == 1: return i;
```

Listing 6.30: Aus ../Sage/enumeratePCNs.spyx

```
534 # computes the quadratic free part of an integer
535 def squarefree(n):
536     return prod(map(lambda x: x[0], factor(Integer(n))))
```

Ausnutzen einfacher Zerlegungen

Zunächst müsste man für jeden erweiterten Kreisteilungsmodul nun *alle* Teiler des Modulcharakters testen, um vollständige Erzeuger zu finden. Jedoch garantiert Satz 5.3, dass dies in manchen Fällen überflüssig ist, da bei einer einfachen Erweiterung ein Erzeuger eines Kreisteilungsmoduls $\mathcal{C}_{k,t}$ über $\mathbb{F}_q$ bereits ein vollständiger Erzeuger ist. Ist eine Erweiterung nicht einfach, so sollte man die Hoffnung nach einer Vereinfachung der Suche nach vollständigen Erzeugern nicht aufgeben, sondern sich überlegen, dass es einen Teiler $d \mid n$ geben kann, für den die Erweiterung $\mathbb{F}_{q^n}$ über $\mathbb{F}_{q^d}$ einfach ist. Dann müssten keine Teiler von Modulcharaktern getestet werden, die größer oder gleich d wären, da – wie man sich sehr leicht überlegt – falls $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ einfach ist, auch für alle Teiler $d \mid n$ $\mathbb{F}_{q^n}$ über $\mathbb{F}_{q^d}$ einfach ist.

Dies wollen wir nutzen in nachstehender Funktion, die gerade zu betrachtenden Teiler einer Erweiterung liefert (und deren Benennung vielleicht etwas kontraintuitiv gewählt wurde).

Listing 6.31: Aus ../Sage/enumeratePCNs.spyx

```
594 # returns the NOT completely basic divisors of an
595 # extension n over GF(p^e)
596 def get_completely_basic_divisors(p,e,n):
597     n = Integer(n)
598     q = Integer(p**e)
599     divs = []
600     for d in divisors(n):
601         isComplBasic = True
602         for r in prime_divisors(n/d):
603             if r.divides(ordn(p_free_part(n/d/r,p),q**d)):
604                 isComplBasic = False
605                 break
606         divs += [d]
607         if isComplBasic: return divs
608     return divs
```

`p_free_part` gibt wie in Satz 5.3 (3) zu sehen ist, gerade den größten Teiler des ersten Arguments an, der nicht mehr durch p, dem zweiten Argument, teilbar ist. Es wird dabei nicht überprüft, ob das zweite Argument eine Primzahl ist.

Listing 6.32: Aus ../Sage/enumeratePCNs.spyx

```
612 # p-free part of t
613 def p_free_part(t,p):
614     while p.divides(t):
615         t /= p
616     return t
```

Ausnutzen regulärer Kreisteilungsmoduln

Sicherlich wollen wir auch Regularität (Definition 5.10) nicht unbeachtet lassen, um uns die Suche nach vollständig normalen Elementen zu erleichtern. Also haben wir auch einen Test auf Regularität nach Sage übersetzt.

Listing 6.33: Aus ../Sage/enumeratePCNs.spyx

```
431 # tests if cyclotomic module C_k,t is regular over F_p^e
432 def isRegular(p,e, k,t,pi):
433     return gcd( ordn( squarefree(k*p_free_part(t,p)), p**e ), k*t*pi) == 1
```

Ist ein Kreisteilungsmodul regulär, so ist ein Test auf vollständige Erzeuger-Eigenschaft durch Satz 5.13 gegeben. Da Regularität lediglich die Anzahl der Teiler des Erweiterungsgrades, deren zugehörige Kreisteilungsmoduln auf vollständige Erzeuger getestet werden müssen, reduziert, wird Satz 5.13 durch Rückgabe der Teiler τ bzw. τ und 2τ (in Notation dieses Satzes) im ausfallenden Fall realisiert, wie die Funktion get_tau_divisors zeigt. Die zu übergebenden Parameter bestehen wieder aus $q = p^e$ und der Daten (k, t, π) des zu betrachtenden Kreisteilungsmoduls $\mathcal{C}_{k,tpi}$ über $\mathbb{F}_q$.

Listing 6.34: Aus ../Sage/enumeratePCNs.spyx

```
436 # returns tau-divisors for complete generator test of
437 # the cyclotomic module C_k,t*pi over F_p^e
438 def get_tau_divisors(p,e, k,t,pi):
439     if t != 1:
440         print "ERROR␣get_tau_divisors:␣t␣!=␣1␣for␣p=",p,"␣e=",e\
441                 ,"␣k=",k,"␣t=",t,"␣pi=",pi
442         raise Exception("Error␣t!=1")
443     q = p**e
444     tau = ordn(k,q) / ordn(squarefree(k),q)
445     tau = prod(map(lambda ra: ra[0]**floor(ra[1]/2), factor(tau)))
446     if isExceptional(p,e, k):
447         return [ tau, 2*tau ]
448     else:
449         return [ tau ]
```

Wie in ?? gezeigt, ist die kanonische Zerlegung im regulären Fall verträglich. Daher tritt ein Fehler auf, wird obiger Funktion ein Kreisteilungsmodul $\mathcal{C}_{l,m}$ übergeben mit $m \neq p^b$ für ein $b \geq 0$.

Es bleibt natürlich noch ein Test anzugeben, der überprüft, ob die Parameter (p, e, k) ausfallend sind (vgl. Definition 5.11).

Listing 6.35: Aus ../Sage/enumeratePCNs.spyx

```
452 # tests if n is exceptional over F_p^e
453 def isExceptional(p,e, n):
454     q = p**e
455     c = 0
456     nbar = n
457     while Integer(2).divides(nbar):
458         c += 1
459         nbar /= 2
460     if (q).mod(4) == 3 and c >= 3 and ordn(q, 2**c) == 2:
461         return True
462     return False
```

Die zentrale Sage-Funktion countCompleteSubmoduleGenerators

Als übergeordnete Funktion, die die Anzahl aller (primitiven) vollständig normale Elemente und aller vollständigen Erzeuger im Sinne des Zerlegungssatzes liefert, stellen wir countCompleteSubmoduleGenerators bereit. Als Argumente sind selbstredend ein endlicher Körper zu übergeben und der Grad der zu betrachtenden Erweiterung. Ferner gibt es die Möglichkeit durch das optionale Argument binaryPowers=False den Test auf Primitivität durch p-adische Exponentiation durchführen zu lassen wie in dem Absatz vor Listing 6.15 erwähnt wurde (vgl. auch Unterabschnitt 6.2.2). Der Test auf Primitivität lässt sich auch vollständig deaktivieren durch die Übergabe von testPrimitivity =False (vgl. Bemerkung 6.13).

Der Rückgabewert der Funktion enthält die Anzahl aller vollsändig normalen Elemente der Erweiterung, die Anzahl aller primitiv vollständig normalen (oder 0, falls der Test auf Primitivität deaktiviert wurde), die Anzahl der jeweiligen vollständigen Erzeuger der Zerlegung in verallgemeinerte Kreisteilungsmodule nach Satz 5.9 und abschließend die Dauer der Berechnung.

Im Gegensatz zu den bisherigen Listings werden wir countCompleteSubmoduleGenerators in mehrere Teile aufspalten, um ein besseres Verständnis zu gewährleisten. Wir beginnen mit den ersten Zeilen, die in offensichtlicher Weise die Datenstrukturen der Zerlegung bereitstellen, wie sie in testAllSubmods (Listing 6.19) bzw. testSubmod (Listing 6.18) benötigt werden.

Listing 6.36: Aus ../Sage/enumeratePCNs.spyx

```
88 def countCompleteSubmoduleGenerators(F,n, binaryPowers=True, \
89         testPrimitivity=True):
90     TIME = time.time()
91     p = F.characteristic()
92     q = F.order();
93     e = q.log(p)
94     E = F.extension(Integer(n),'a');
95     P = E.prime_subfield()
96     #generate factors
97     polys = []
98     polysCount = []
99     evalToZero = []
100    frobPowers = []
101    notComplBasicDivisors = get_completely_basic_divisors(p,e,n)
102    decomposition = decompose(p,e,n)
```

```
103     for decomp in decomposition:
104         k,t,pi = decomp
105         divs = divisors(get_module_character(*decomp))
106         divs = filter(lambda x: x in notComplBasicDivisors, divs)
107         countPolysForThisDecomp = 0
108         for d in divs:
109             G = F.extension(Integer(d), 'c');
110             Gx = PolynomialRing(G,'x');
111             h = Hom(G,E)[0]
112             cycl = Gx.cyclotomic_polynomial(squarefree(k))\
113                    (Gx.gen()**(k*t*pi/squarefree(k)/d))
114             polys += [map(lambda x: x.polynomial().list(),
115                 cycl.map_coefficients(h).list())]
116             frobPowers += [d]
117             evalToZero += [1]
118             countPolysForThisDecomp += 1
119             # add Co-Factors
120             for f,mult in cycl.factor():
121                 g = cycl.quo_rem(f)[0]
122                 gE = g.map_coefficients(h)
123                 polys += [map(lambda x: x.polynomial().list(), gE.list())]
124                 frobPowers += [d]
125                 evalToZero += [0]
126                 countPolysForThisDecomp +=1
127         polysCount += [countPolysForThisDecomp]
```

Wie man gut erkennen kann, werden einfache Zerlegungen in den Zeilen `101` und `106` ausgenutzt.

Anschließend berechnen wir, falls `testPrimitivity=True`, die Kofaktoren, wie sie beim Test auf Primitivität in `isPrimitive` (Listing 6.15) verwendet werden. Dabei ist zu unterscheiden, ob die Faktoren in binärer (ab Zeile `151`) oder p-adischer Form (ab Zeile `156`) genutzt werden sollen. Bei p-adischer Darstellung muss, wie in dem Absatz vor `powerFFElem` (Listing 6.13) erwähnt, die Länge des maximal auftretenden 0-Intervalls berechnet werden (ab Zeile `164`).

Listing 6.37: `countCompleteSubmoduleGenerators` Fortsetzung (I)

```
128     charac = int(E.characteristic())
129         #mipo
130     mipo = E.modulus().list()
131     m = len(mipo)-1
132
133     #calc prime factors of order
134     barFactors = []
135     primitiveOrder = E.order()-1
136     if testPrimitivity:
137         factors = reversed(factor(primitiveOrder))
138         for r,k in factors:
139             barFactors += [primitiveOrder/r]
140         countBarFactors = len(barFactors)
141         commonBarFactor = gcd(barFactors)
142         commonBiggestBarFactor = max(gcd(barFactors[1:]) / commonBarFactor,1)
143         barFactors = map(lambda b: b/commonBarFactor, barFactors)
144         curF = 0
```

```
145         barFactors_tmp = [barFactors[0]]
146         for b in barFactors[1:]:
147             barFactors_tmp += [ b/commonBiggestBarFactor - curF]
148             curF = b/commonBiggestBarFactor
149         barFactors = barFactors_tmp
150 
151         if binaryPowers:
152             barFactors = map(lambda b: get_padic_representation(b,2),barFactors)
153             commonBarFactor = get_padic_representation(commonBarFactor,2)
154             commonBiggestBarFactor = \
155                     get_padic_representation(commonBiggestBarFactor,2)
156         else:
157             barFactors = map(lambda b: get_padic_representation(b,p),barFactors)
158             commonBarFactor = get_padic_representation(commonBarFactor,p)
159             commonBiggestBarFactor = \
160                     get_padic_representation(commonBiggestBarFactor,p)
161         lenCommonBarFactor = len(commonBarFactor)
162         lenCommonBiggestBarFactor = len(commonBiggestBarFactor)
163 
164         lenBiggestZeroGap = 0
165         if not binaryPowers:
166             #find biggest gap (i.e. zero-interval)
167             lenCurGap = 0
168             for b in barFactors+[commonBarFactor]+[commonBiggestBarFactor]:
169                 i = 0
170                 while i < len(b):
171                     lenCurGap = 0
172                     while i<len(b) and b[i] == 0:
173                         lenCurGap+= 1
174                         i += 1
175                     lenBiggestZeroGap = max(lenBiggestZeroGap, lenCurGap)
176                     i += 1
177     else:
178         countBarFactors = 0
179         barFactors = []
180         commonBarFactor = []
181         commonBiggestBarFactor = []
182         lenBiggestZeroGap = 0
```

Im letzten Teil der reinen Sage-Aufbereitung, liften wir die Elemente des Grundkörpers mittels eines injektiven Körperhomomorphismus in den Erweiterungskörper, wie sie in `calcSubmoduleElements` (Listing 6.20) bzw. `processLastSubmoduleAndTestPrimitivity` (Listing 6.24) benötigt werden. Ferner stellen wir die Additions- und Multiplikationstabellen nach Unterabschnitt 6.1.2 auf.

Listing 6.38: `countCompleteSubmoduleGenerators` Fortsetzung (II)

```
183         #generate F elements in E
184     elementsF = []
185     if e == 1:
186         elementsF = map(lambda e: [e], list(F))
187     else:
188         h = Hom(F,E)[0]
189         for e in itertools.product(xrange(p),repeat=e):
190             elementsF += [h( F(list(reversed(e))) ).polynomial().list()]
```

```
191
192         #calculate addition and multiplication tables
193     ps = range(p)
194     addTable = ps[P(-2*(p-1)):] + ps*2 + ps[:Integer(P(2*(p-1)))+1]
195     multTable = ps[P(-(p-1)**2):] + ps*(2*(p-2)) + ps[:Integer(P((p-1)**2))+1]
```

Nun sind wir bereit alle Daten nach `C` zu transferieren. Dies ist ein notwendiges Übel, da die interne Repräsentation von `Sage`-Objekten nicht mit denen in `C` vereinbar sind. Beispielsweise sind Listen von Ganzzahlen keineswegs Arrays, jedoch existiert gerade für diesen Fall die Möglichkeit die komfortable Syntax von `numpy`-Arrays zu nutzen, die direkt auf `C`-Arrays basieren.[2]

Andere Datenstrukturen, wie die selbst erstellten `struct FFElems`, müssen händisch übersetzt werden. Da `Cython` das (etwas merkwürdig wirkende) Mischen von `Python` und `C` erlaubt, schieben wir die hierfür erstellten Funktionen der Übersetzung von `Python`-Listen in die jeweilige `C`-Datenstruktur kurz ein.

Listing 6.39: Aus `../Sage/enumeratePCNs.spyx`

```
55 cdef FFElem *pyList2FFElem(element,int m):
56     cdef FFElem *ff = mallocFFElem(<int>m)
57     initPoly(ff.el,m)
58     for i,e in enumerate(element):
59         ff.el[i] = e
60     updateFFElem(ff,m)
61     return ff
```

Listing 6.40: Aus `../Sage/enumeratePCNs.spyx`

```
63 cdef FFElem **pyList2PointFFElem(pyList, int m):
64     lenList = len(pyList)
65     cdef FFElem **ffs = <FFElem**>malloc(lenList*sizeof(FFElem*))
66     for i,e in enumerate(pyList):
67         ffs[i] = pyList2FFElem(e,m)
68     return ffs
```

Listing 6.41: Aus `../Sage/enumeratePCNs.spyx`

```
70 cdef FFPoly *pyList2FFPoly(listPoly, int m):
71     lenPoly = len(listPoly)
72     cdef FFPoly *poly = <FFPoly*>malloc(sizeof(FFPoly))
73     poly.poly = <FFElem**>malloc(lenPoly*sizeof(FFElem*))
74     poly.lenPoly = lenPoly
75     for i,e in enumerate(listPoly):
76         poly.poly[i] = pyList2FFElem(e,m)
77     return poly
```

Listing 6.42: Aus `../Sage/enumeratePCNs.spyx`

```
79 cdef FFPoly **pyList2PointFFPoly(listPolys, int m):
80     countPolys = len(listPolys)
```

[2] Siehe z.B. `http://www.sagemath.org/doc/numerical_sage/numpy.html` für die Benutzung von `numpy`-Arrays in `Sage`.

```
81      cdef FFPoly **polys = <FFPoly**>malloc(countPolys*sizeof(FFPoly*))
82      for i,e in enumerate(listPolys):
83          polys[i] = pyList2FFPoly(e,m)
84      return polys
```

Nun können wir die Beschreibung von `countCompleteSubmoduleGenerators` fortsetzen und erkennen sofort die gerade vorgestellten Funktionen der Übersetznug sowie die Benutzung der `numpy`-Arrays.

Listing 6.43: `countCompleteSubmoduleGenerators` Fortsetzung (III)

```
196     # SETUP C DATA ============================================================
197     maxMatPower = max(map(lambda d: euler_phi(d[0])*d[1]*d[2], decomposition))
198         # multiplication and addition table
199     cdef np.ndarray[int,ndim=1,mode="c"] multTableRawC\
200         = np.array(multTable, dtype=np.int32)
201     cdef np.ndarray[int,ndim=1,mode="c"] addTableRawC\
202         = np.array(addTable, dtype=np.int32)
203     cdef int* multTableC = <int*>multTableRawC.data + <int>((p-1)**2)
204     cdef int* addTableC = <int*>addTableRawC.data + <int>(2*(p-1))
205         #setup mipo
206     cdef FFElem *mipoC = pyList2FFElem(mipo,m+1)
207        #setup matrices
208     cdef FFElem **matsC = genFrobMats(mipoC,m,maxMatPower,q,
209            multTableC, addTableC)
210         # mat charac
211     cdef FFElem **matCharacC
212     if binaryPowers:
213         matCharacC = <FFElem**>0
214     else:
215         matCharacC = genFrobMats(mipoC,m,lenBiggestZeroGap+1,
216                p, multTableC, addTableC)
217     #setup polynomials, polyLength, frobPowers, evaltoZero
218     decompCount = int(len(polysCount))
219         #evalToZeroC
220     cdef np.ndarray[char,ndim=1,mode="c",cast=True] evalToZeroC\
221            = np.array(evalToZero, dtype=np.uint8)
222         #frobPowersC
223     cdef np.ndarray[int,ndim=1,mode="c"] frobPowersC\
224            = np.array(frobPowers, dtype=np.int32)
225         #polysCountC
226     cdef np.ndarray[int,ndim=1,mode="c"] polysCountC\
227            = np.array(polysCount, dtype=np.int32)
228     cdef FFPoly **polysC = pyList2PointFFPoly(polys,m)
229         # bar Factors
230     cdef np.ndarray[int,ndim=1,mode="c"] barFactorsC \
231         = np.array(list(itertools.chain(*barFactors)), dtype=np.int32)
232     cdef np.ndarray[int,ndim=1,mode="c"] lenBarFactorsC \
233         = np.array(map(len,barFactors), dtype=np.int32)
234     cdef np.ndarray[int,ndim=1,mode="c"] commonBarFactorC \
235         = np.array(commonBarFactor, dtype=np.int32)
236     cdef np.ndarray[int,ndim=1,mode="c"] commonBiggestBarFactorC \
237         = np.array(commonBiggestBarFactor, dtype=np.int32)
238         # F elements in E
```

```
239     cdef FFElem **elementsFC = pyList2PointFFElem(elementsF,m)
240     #========================================================================
```

Es gilt anzumerken, dass die Erzeugung der Darstellungsmatrizen des Frobenius in `C` durch die Funktion `genFrobMats` (die in `../Sage/enumeratePCNs.c` zu finden ist und hier nicht näher erläutert wird, da sie weder vom mathematischen Standpunkt her besonders spannend ist, noch besonderes programmiertechnisch besondere Aufmerksamkeit verdient) geschieht, wobei die maximal zu berechnende Matrixpotenz gerade durch den Grad des größten auftretenden Polynoms der Zerlegung gegeben ist. Wir wissen jedoch genau, wie der Grad eines verallgemeinerten Kreisteilungspolynoms zu berechnen ist, wie Zeile `197` erkennen lässt.

In einem letzten Schritt können wir (nun endlich) die bereitgestellte `C`-Funktion `processFiniteField` (Listing 6.25) aufrufen und die Rückgabewerte verwalten. Hier gilt es anzumerken, dass die Anzahl der vollständigen Erzeuger direkt in das Array `genCountsC` geschrieben wird und nicht als expliziter Rückgabewert erkennbar ist.

Listing 6.44: `countCompleteSubmoduleGenerators` Fortsetzung (IV)

```
241     #setup return values
242     cdef np.ndarray[int,ndim=1,mode="c"] genCountsC
243     genCountsC = np.zeros(decompCount, dtype=np.int32)
244
245     cdef unsigned long long pcn = \
246             processFiniteField(mipoC, decompCount,
247                     polysC,<int*>polysCountC.data,
248                     <char*>evalToZeroC.data,
249                     matsC,maxMatPower,<int*>frobPowersC.data,
250                     <int*>genCountsC.data, m, p, q,
251                     <int*>barFactorsC.data, <int*>lenBarFactorsC.data,
252                     countBarFactors,
253                     <int*>commonBarFactorC.data,lenCommonBarFactor,
254                     <int*>commonBiggestBarFactorC.data,lenCommonBiggestBarFactor,
255                     matCharacC,elementsFC,
256                     multTableC,addTableC)
257
258     genCounts = dict()
259     for i,d in enumerate(decomposition):
260         genCounts[d] = Integer(genCountsC[i])
261
262     # Free all malloced variables at the end ================================
263     freeFFElem(mipoC)
264     freeFFElemMatrix(matsC,m*maxMatPower)
265     for i in range(len(polys)):
266         freeFFPoly(polysC[i])
267     free(polysC)
268     freeFFElemMatrix(matCharacC,m*(lenBiggestZeroGap+1))
269     freeFFElemMatrix(elementsFC,len(elementsF))
270     #========================================================================
271     return prod(genCounts.values()), Integer(pcn), genCounts,\
272             strfdelta(datetime.timedelta(seconds=(time.time()-TIME)))
```

6.4.4 | Ein ausführliches Beispiel

Wir wollen nun einmal das gesamte Verfahren zur Berechnung Anzahl der primitiv vollständig normalen Elemente einer Erweiterung endlicher Körper anhand eines Beispiels nachvollziehen. Dazu wählen wir $F := \mathbb{F}_2$ und $n := 6$, also $E := \mathbb{F}_{2^6}$. Die Wahl des Minimalpolynoms dieser Erweiterung überlassen wir `Sage` und erhalten

$$E = \mathbb{F}_2[a]/(a^6 + a^4 + a^3 + a + 1)\,.$$

Gehen wir erneut den Code von `countCompleteSubmoduleGenerators` Zeile für Zeile durch, so beginnen wir mit der Festlegung der grundlegenden Parameter:

$$p := 2, \qquad q := 2, \qquad e := 1, \qquad P := \mathbb{F}_2\,.$$

Berechnung der nicht einfachen Teiler Im nächsten Schritt berechnen wir die nicht einfachen Teiler mithilfe `get_completely_basic_divisors` (Listing 6.31). Dazu gehen wir alle Teiler von $n = 6$ durch und überprüfen, ob die jeweiligen Erweiterungen einfach sind, d.h. für jeden Teiler $d \mid n$ testen wir für jeden Primteiler $r \mid \frac{n}{d}$, ob $r \nmid \operatorname{ord}_{(\frac{n}{dr})'}(q^d)$ (vgl. Satz 5.3). Wir brechen jeweils ab, falls ein r die Teilbarkeitsbedingung nicht erfüllt.

d	$\frac{n}{d}$	r	$(\frac{n}{dr})'$	$\operatorname{ord}_{(\frac{n}{dr})'}(q^d)$	$r \nmid \operatorname{ord}_{(\frac{n}{dr})'}(q^d)$
1	6	2	3	2	↯ $\rightsquigarrow$ nicht einfach
2	3	3	1	1	✓ $\rightsquigarrow$ einfach

Damit sind alle zu betrachtenden Teiler von n gegeben durch

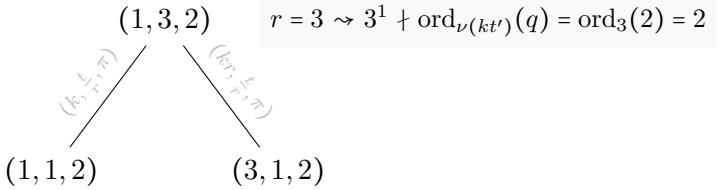
$$\texttt{notComplBasicDivisors} := [1, 2]\,.$$

Wie man erkennt, wollen wir in diesem Beispiel alle auftretenden Listen in der `Python`/`Sage`-üblichen Notation [, ,...] angeben.

Anwendung des Zerlegungssatzes Anschließend folgt die Berechnung der Zerlegung in Kreisteilungsmoduln durch den Zerlegungssatz. Da wir in der konkreten Implementierung stets drei Parameter für die Angabe von Kreisteilungsmoduln verwenden, d.h. Potenzen der Charakteristik immer „ausklammern", wollen wir dies auch hier so notieren. Der zu $x^n - 1 = x^6 - 1$ über $\mathbb{F}_2$ gehörige Kreisteilungsmodul ist offenbar

$$\mathcal{C}_{1,6} \;=\; \mathcal{C}_{1,3\cdot 2}$$

und wir erhalten damit das Parametertripel $(k, t, \pi) := (1, 3, 2)$. Hier startet der Zerlegungssatz rekursiv und wie in `decompose_cycl_module` (Listing 6.27) erkennbar, durchlaufen wir die Primteiler von t in der Größe nach absteigend sortierter Reihenfolge.



Da an den beiden Blättern $t = 1$ gilt, endet hier die Möglichkeit einer weiteren Rekursionsstufe und wir fassen zusammen, dass

$$x^6 - 1 \;=\; \Phi_1(x)^2\ \Phi_3(x)^2$$

die feinste verträgliche Zerlegung des Kreisteilungsmoduls $\mathcal{C}_{1,6}$ über $\mathbb{F}_2$ ist.

Polynome aufstellen Nun sind wir in der Lage, die Polynome zu berechnen, die wir für den Test von vermeindlichen vollständigen Erzeugen benötigen werden.

(1) Wir starten beim ersten erweiterten Kreisteilungspolynom

$$\Phi_{1,1}^2 \;=\; x^2 + 1 \qquad \in \mathbb{F}_2[x]\,.$$

Es ist nun $(k, t, pi) \coloneqq (1, 1, 2)$ und wir müssten alle Teiler des Modulcharakters $\frac{kt\pi}{\nu(k)} = 2$, betrachten. Wie man an obig berechnetem `notComplBasicDivisors` erkennt, lässt sich in diesem Fall auch keiner der beiden Teiler $\{1, 2\}$ streichen. Ein zweiter Kniff schafft aber eine Reduktion der Teilerzahl, da $(1, 1, 2)$ regulär über $\mathbb{F}_2$ ist:

$$\mathrm{ord}_{\nu(kt')}(q) \;=\; \mathrm{ord}_1(2) \;=\; 1\,.$$

Da auch $(1, 1, 2)$ nicht ausfallend über $\mathbb{F}_2$ ist, reicht es den einzigen Teiler zu berechnen, den wir benötigen:

$$\tau(q, k) \;=\; \tau(2, 1) \;=\; 1\,,$$

da $\mathrm{ord}_k(q) = \mathrm{ord}_1(2) = \mathrm{ord}_{\nu(k)}(q)$.

$d = 1$. Nun sind alle Kofaktoren einer vollständigen Faktorisierung von $\Phi_{1,1}(x)^2$ über $\mathbb{F}_{2^d} = \mathbb{F}_2$ zu berechnen:

$$\Phi_{1,1}(x)^2 \;=\; (x + 1)^2$$

und der einzige Kofaktor ist durch

$$g_{1,1,1}(x) \;=\; x + 1$$

gegeben.

(2) Nun zum zweiten Kreisteilungsmodul $(k, t, \pi) \coloneqq (3, 1, 2)$. Der Modulcharakter ist wiederum $\frac{kt\pi}{\nu(k)} = 2$. Auch hier können wir mit `notComplBasicDivisors` keinen Teiler wegdiskutieren. Anders als in obigem Fall ist dieser Kreisteilungsmodul nicht einmal regulär, da

$$\mathrm{ord}_{\nu(kt')}(q) \;=\; \mathrm{ord}_3(2) = 2$$

nicht teilerfremd zu $kt = 2$ ist. Also bleiben beide Teiler $\{1, 2\}$ übrig.

$d = 1$. Wir faktorisieren

$$\Phi_{3,1}(x)^2 \;=\; (x^2 + x + 1)^2 \qquad \in \mathbb{F}_2[x]$$

und erhalten als einzigen Kofaktor dieses Teilers

$$g_{2,1,1}(x) \;=\; x^2 + x + 1\,.$$

$d = 2$. Blicken wir noch einmal in die Definition eines vollständigen Erzeugers (Definition 5.6), so sehen wir, dass wir nun $\mathcal{C}_{3,1\cdot 2}$ als $\mathbb{F}_{2^2}[x]$-Modul betrachten müssen. Orientiert man sich an der Definition eines verallgemeinerten Kreisteilungsmoduls (Definition 5.5), so sind wir gezwungen $\Phi_3(x^{\frac{2}{2}})$ über $\mathbb{F}_{2^2}$ zu faktorisieren. Dazu überlassen wir wiederum `Sage` die Repräsentation des endlichen Körpers

$$\mathbb{F}_{2^2} \;=\; \mathbb{F}_2[b]\big/(b^2+b+1)$$

und faktorisieren

$$\Phi_{3,1}(x) \;=\; (x+b)(x+b+1)\,.$$

Ergo erhalten wir die beiden Kofaktoren in $\mathbb{F}_{2^2}[x]$:

$$\begin{aligned} g_{2,2,1}(x) &:= x+b+1\,,\\ g_{2,2,2}(x) &:= x+b\,. \end{aligned}$$

Wie aber in der Beschreibung der Implementierung erwähnt, bietet es sich an, diese Polynome mittels eines injektiven Körperhomomorphismus in $E = \mathbb{F}_{2^6}$ zu lesen. Auch die Berechnung eines solchen überlassen wir `Sage` und wählen

$$\begin{aligned} h\colon\quad \mathbb{F}_2[b]\big/(b^2+b+1) &\;\to\; \mathbb{F}_2[a]\big/(a^6+a^4+a^3+a+1)\,,\\ b &\;\mapsto\; a^2+a^2+a\,. \end{aligned}$$

Damit schreiben wir obige Kofaktoren zu

$$\begin{aligned} g_{2,2,1}(x) &:= x+a^3+a^2+a+1\,,\\ g_{2,2,2}(x) &:= x+a^3+a^2+a \end{aligned}$$

um, gelesen als Elemente von $\big(\mathbb{F}_2[a]\big/(a^6+a^4+a^3+a+1)\big)[x]$.

Als letzten Schritt des Aufstellens der Polynome fassen wir alle Ergebnisse zusammen und erinnern uns an die Implementierung, wo neben den Polynomen auch die Information, welche Polynome bei Vorliegen eines vollständigen Erzeugers in der Frobenius-Auswertung zu Null ausgewertet werden müssen, und die Angabe der Frobenius-Potenzen benötigt werden. Da alle Polynome in *eine einzige* Liste geschrieben werden, muss man selbstredend die Anzahl der Polynome des jeweiligen Kreisteilungsmoduls abspeichern. Zusammengefasst erhalten wir folgende Daten:

$$\begin{aligned}
\texttt{polys} &:= [x^2+1, & x+1, & & x^4+x^2+1, & & x^2+x+1, & & x+a^3+a^2+a+1, & & x+a^3+a^2+a]\\
\texttt{evalToZero} &:= [1, & 0, & & 1, & & 0, & & 0, & & 0 & &]\\
\texttt{frobPowers} &:= [1, & 1, & & 1, & & 1, & & 2, & & 2 & &]\\
\texttt{polysCount} &:= [2, & & & 4 & & & & & & & &]
\end{aligned}$$

Wie man sicherlich bemerkt, führen wir das den zweiten Kreisteilungsmodul definierende Polynom $\Phi_{3,2}$ lediglich für den Teiler $d = 1$ auf. Für den Teiler $d = 2$ hätten wir $\Phi_{3,1}$ jedoch mit Frobenius-Potenz 2. Da ein Element $u \in E$ jedoch genau dann $\Phi_{3,2}(\sigma)(u) = 0$ erfüllt, wenn $\Phi_{3,1}(\sigma^2)(u) = 0$, ist dieser Berechnungsschritt obsolet.

Daten für einen Primitivitätstest Für den in Listing 6.15 beschriebenen Primitivitätstest, müssen wir zunächst $q^n - 1 = 2^6 - 1$ faktorisieren:

$$2^6 - 1 = 3^2 \cdot 7\,.$$

Also sind die zu testenden Kofaktoren gerade 9 und 21. Wir erkennen sofort, dass der größte gemeinsame Teiler beider Faktoren 3 ist und setzen daher in Benennung von `isPrimitive` (Listing 6.15)

$$\texttt{commonBarFactor} := 3\,.$$

Ergo reduzieren sich die Kofaktoren auf 3 und 7. Im nächsten Schritt betrachten wir nur noch alle Kofaktoren, die den größten Primfaktor obiger Faktorisierung enthalten. Hier ist dies nur einer: 7. Wieder berechnen wir den ggT all dieser: 7. Damit haben wir alle restlichen Daten:

$$\begin{aligned}\texttt{commonBiggestBarFactor} &:= 7\\ \texttt{barFactors} &:= [3,1]\end{aligned}$$

Benutzen wir binäre Exponentiation so übersetzen wir die erhaltenen Zahlen ins Binärsystem:

$$\begin{aligned}\texttt{commonBarFactor} &:= [1,1]\\ \texttt{commonBiggestBarFactor} &:= [1,1,1]\\ \texttt{barFactors} &:= \big[[1,1,1],\ [1]\big]\end{aligned}$$

In diesem Fall wären die Zahlen in p-adischer Schreibweise identisch, da ja $p = 2$.

Aufstellen der Frobenius-Matrizen Um den Frobenius von F, also $\bar{F} \to \bar{F}, x \mapsto x^2$, effizient auf Elemente aus E anwenden zu können, müssen wir seine Darstellungsmatrix bezüglich der kanonischen Basis

$$\{1, a, a^2, a^3, a^4, a^5\} \subseteq \mathbb{F}_2[a]\big/(a^6 + a^4 + a^3 + a + 1)$$

berechnen. Dazu fassen wir selbstredend die Elemente aus E als Vektoren in $\mathbb{F}_2^6$ auf:

$$\begin{array}{lll}
1^2 = 1 & \cong & \begin{bmatrix}1&0&0&0&0&0\end{bmatrix}^T\\
a^2 = a^2 & \cong & \begin{bmatrix}0&0&1&0&0&0\end{bmatrix}^T\\
(a^2)^2 = a^4 & \cong & \begin{bmatrix}0&0&0&0&1&0\end{bmatrix}^T\\
(a^3)^2 = a^4 + a^3 + a + 1 & \cong & \begin{bmatrix}1&1&0&1&1&0\end{bmatrix}^T\\
(a^4)^2 = a^5 + a^4 + a^2 + a + 1 & \cong & \begin{bmatrix}1&1&1&0&1&1\end{bmatrix}^T\\
(a^5)^2 = a^5 + a^4 + 1 & \cong & \begin{bmatrix}1&0&0&0&1&1\end{bmatrix}^T
\end{array}$$

Damit erhalten wir eine Darstellungsmatrix des Frobenius:

$$\Gamma_\sigma := \begin{bmatrix}1&0&0&1&1&1\\0&0&0&1&1&0\\0&1&0&0&1&0\\0&0&0&1&0&0\\0&0&1&1&1&1\\0&0&0&0&1&1\end{bmatrix}$$

Wie man an den obigen Polynomen in `polys` erkennen kann, ist die maximale Potenz des Frobenius gerade 4. Daher bleibt noch Γ_σ^2, Γ_σ^3 und Γ_σ^4 zu berechnen:

$$\Gamma_\sigma^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad \Gamma_\sigma^3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad \Gamma_\sigma^4 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Wie man in der Implementierung erkennen kann übergeben wir die Frobenius-Matrizen stets als `FFElem **mats`, d.h. man sollte sich obige vier Matrizen eher als eine (24×6)-Matrix vorstellen, deren Zeilen jeweils aus einem `FFElem` bestehen (vgl. Abschnitt 6.1). Unter dieser Analogie ist dies gerade das Ergebnis der Funktion `genFrobMats`.

Iteration von E auf der Suche nach vollständigen Erzeugern Wie in der Beschreibung von `processFiniteField` (Listing 6.25) angegeben, starten wir die Suche nach vollständig normalen und primitiven Elementen bei einer Iteration des endlichen Körpers E, bis wir für jeden Kreisteilungsmodul der Zerlegung einen vollständigen Erzeuger gefunden haben. Die konkrete Iteration erfolgt dabei lexikographisch in $\mathbb{F}_2^6$, wobei wir der besseren Lesbarkeit geschuldet zwischen den verschiedenen Schreibweisen von Vektoren in $\mathbb{F}_2^6$ und Polynomen in $\mathbb{F}_2[a]/(a^6+a^4+a^3+a+1)$ ohne besondere Kennzeichnung wechseln werden.

$u := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$. Hier gibt es nichts zu tun.

$u := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$. Wir blicken auf `polys` und berechnen

$$(x^2+1)(\sigma)(1) = 0 \checkmark, \qquad (x+1)(\sigma)(1) = 0 \lightning.$$

Also ist 1 kein Erzeuger von $\mathcal{C}_{1,1\cdot 2}$ über $\mathbb{F}_2$. Beim zweiten Kreisteilungsmodul scheitern wir bereits beim ersten Polynom:

$$(x^4+x+1)(\sigma)(1) = 1 \lightning.$$

... Hier sind weitere Elemente zu denken, die ebenfalls keine vollständigen Erzeuger liefern.

$u := \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$. Dieses Element liefert einen vollständigen Erzeuger des zweiten Kreisteilungsmoduls:

$$(x^4+x^2+1)(\sigma)(u) = 0, \qquad (x^2+x+1)(\sigma)(u) = a^5+a^4+a^2+1,$$
$$(x+a^3+a^2+a^1)(\sigma^2)(u) = a^5+a^4+a^3+a, \quad (x+a^3+a^2+a)(\sigma^2)(u) = a^5+a^4.$$

Die Anwendung des Frobenius ist dabei jeweils durch obige Matrizen zu denken.

...

$u := \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$. Hier haben wir einen vollständigen Erzeuger des ersten Kreisteilungsmoduls, wie nachstehende Rechnung zeigt.

$$(x^2+1)(\sigma)(u) = 0, \qquad (x+1)(\sigma)(u) = 1.$$

Die berechneten vollständigen Erzeuger speichern wir in einem Array aus verketteten Listen (vgl. 6.4.1). Die verketteten Listen sowie das Array wollen wir hier jedoch wieder in `Python`-üblicher Notation angeben. Bisher haben wir also für jeden Kreisteilungsmodul einen Erzeuger gefunden:

$$\texttt{roots} \;=\; \left[\, [a^3 + a^2 + a],\; [a^2 + a] \,\right]$$

Die Benennung `roots` ist hier konsistent mit `processFiniteField` (Listing 6.25) gewählt. Jedoch sind die Elemente der Listen natürlich wieder als `FFElem` zu denken.

Enumeration der einzelnen Kreisteilungsmoduln An diesem Punkt haben wir für jeden Kreisteilungsmodul einen Erzeuger gefunden und können anhand diesem den jeweiligen Modul vollständig enumerieren.

$\Phi^2_{1,1}$. Sei $u := a^3 + a^2 + a$ unser gefundener Erzeuger, so können wir nach Lemma 6.11 den Modul durch Polynome über F, deren Grad kleiner 2 ist, enumerieren:

$$\mathcal{C}_{1,1\cdot 2} \;=\; \{f(\sigma)(u) :\; f \in F[x]_{<2},\; \mathrm{ggT}(f, \Phi^2_{1,1}) = 1\}\,.$$

Wie im Absatz nach Lemma 6.11 erwähnt, führen wir die Berechnung des ggT nicht durch, sondern testen jedes Element auf vollständige Erzeuger-Eigenschaft für diesen Modul.

$f(x)$	$f(\sigma)(u)$	vollst. Erz.
1	$a^3 + a^2 + a$	✓
x	$a^3 + a^2 + a + 1$	✓
$x + 1$	1	↯

$\Phi^2_{3,1}$. Sei in diesem Fall $u := a^2 + a$ der gefundene Erzeuger, so müssen wir Polynome bis zum Grad 3 über F betrachten:

$f(x)$	$f(\sigma)(u)$	vollst. Erz.
1	$a^2 + a$	✓
x	$a^4 + a^2$	✓
$x + 1$	$a^4 + a$	↯
x^2	$a^5 + a^2 + a + 1$	✓
$x^2 + 1$	$a^5 + 1$	✓
$x^2 + x$	$a^5 + a^4 + a + 1$	↯
$x^2 + x + 1$	$a^5 + a^4 + a^2 + 1$	↯
x^3	$a^5 + a^2$	✓
$x^3 + 1$	$a^5 + a$	↯
$x^3 + x$	$a^5 + a^4$	✓
$x^3 + x + 1$	$a^5 + a^4 + a^2 + a$	↯
$x^3 + x^2$	$a + 1$	↯
$x^3 + x^2 + 1$	$a^2 + 1$	↯
$x^3 + x^2 + x$	$a^4 + a^2 + a + 1$	↯
$x^3 + x^2 + x + 1$	$a^4 + 1$	↯

In der konkreten Implementierung speichern wir diese Ergebnisse nicht ab, sondern erzeugen die weiteren Erzeuger des letzten Kreisteilungsmoduls dynamisch (vgl. Unterabschnitt 6.4.2), was wir hier zur besseren Übersichtlichkeit nicht tun wollen.

Nun können wir die aktualisierte Liste `roots` angeben:

$$\texttt{roots} \;=\; \big[\,[a^3+a^2+a,\, a^3+a^2+a+1],\; [a^2+a,\, a^4+a^2,\, a^5+a^2+a+1,\, a^5+1,\, a^5+a^2,\, a^5+a^4]\,\big]$$

An dieser Stelle können wir bereits festhalten, dass in der Erweiterung von Grad 6 über $\mathbb{F}_2$ genau $6 \cdot 2 = 12$ vollständig normale Elemente existieren.

Primitivitätstest Für einen Primitivitätstest müssen wir die 12 vollständig normalen Elemente natürlich erst einmal „zusammenbauen". Dazu durchlaufen wir das kartesische Produkt aus den Listen in `roots` und bilden jeweils die Summe der einzelnen Elemente (vgl. Definition 5.8).

$(a^3+a^2+a) \;+\; (a^2+a)$. Wie in der Beschreibung zu `isPrimitive` (Listing 6.15) erläutert, berechnen wir zunächst $v^{\texttt{commonBarFactor}} = v^3$, wobei $v := a^3+a^2+a+a^2+a = a^3$ das zu testende Element ist. Dies führen wir mittels binärer Exponentiation durch, wie in `powerFFElemSqM` (Listing 6.14) beschrieben, geben hier jedoch nur das Ergebnis an. Es ist

$$v^3 \;=\; a^5+a^4+a^2+1 \;=:\; w\,.$$

Da $w \neq 1$ müssen wir fortfahren mit dem ersten Faktor aus `barFactors`:

$$w^3 \;=\; a^5+a^4+a^2\,.$$

Auch dies ist ungleich 1, also fahren wir fort mit $w^{\texttt{commonBiggestBarFactor}}$ wie in `isPrimitive` (Listing 6.15) angegeben:

$$w^7 \;=\; 1\,.$$

An dieser Stelle können wir abbrechen und wissen, dass v kein primitives Element ist.

$(a^3+a^2+a+1) \;+\; (a^2+a)$. Auch hier beginnen wir mit $v := a^3+a^2+a+1+a^2+a = a^3+1$ und berechnen

$$v^3 \;=\; a^5+a^2+a+1 \;=:\; w\,.$$

Wieder ist $w \neq 1$ und wir fahren fort mit dem ersten `barFactor`.

$$w^3 \;=\; a^4+a^2+a\,.$$

Für den Exponenten `biggestCommonBarFactor` $= 7$ erhalten wir:

$$w^7 \;=\; a^3+a^2+a \;=:\; z\,.$$

Diesen müssen wir nun mit allen verbleibenden `barFactors` potenzieren. In unserem Fall lediglich einer:

$$z^1 \;=\; a^3+a^2+a$$

und somit ist v ein primitives Element in E.

Für alle weiteren Elemente wollen wir nur das Ergebnis der Primitivitätstests in tabellarischer Form angeben.

Gen von $\mathcal{C}_{1,1\cdot 2}$	Gen von $\mathcal{C}_{3,1\cdot 2}$	Element	primitiv
a^3+a^2+a	a^4+a^2	a^4+a^3+a	✓
a^3+a^2+a+1	a^4+a^2	a^4+a^3+a+1	↯
a^3+a^2+a	a^5+a^2+a+1	a^5+a^3+1	↯
a^3+a^2+a+1	a^5+a^2+a+1	a^5+a^3	✓
a^3+a^2+a	a^5+1	$a^5+a^3+a^2+a+1$	↯
a^3+a^2+a+1	a^5+1	$a^5+a^3+a^2+a$	✓
a^3+a^2+a	a^5+a^2	a^5+a^3+a	✓
a^3+a^2+a+1	a^5+a^2	a^5+a^3+a+1	↯
a^3+a^2+a	a^5+a^4	$a^5+a^4+a^3+a^2+a$	✓
a^3+a^2+a+1	a^5+a^4	$a^5+a^4+a^3+a^2+a+1$	↯

Zusammenfassend existieren also 6 primitiv vollständig normale Elemente in der Erweiterung von Grad 6 über $\mathbb{F}_2$.

Literatur

[1] I. F. Blake, S. Gao und R. C. Mullin. »Specific irreducible polynomials with linearly independent roots over finite fields«. In: *Linear Algebra and its Applications* 253.1–3 (1997), S. 227–249.

[2] S. Bosch. *Algebra*. Springer-Lehrbuch. Springer, 2009.

[3] D. Hachenberger. *Finite fields: normal bases and completely free elements*. Developments in Molecular and Cellular Biochemistry. Kluwer Academic Publishers, 1997. URL: `http://books.google.de/books?id=Fv7uAAAAMAAJ`.

[4] B. Hartley und T. Hawkes. *Rings, Modules and Linear Algebra*. Chapman and Hall mathematics series. Chapman und Hall Limited, 1974.

[5] C. Karpfinger und K. Meyberg. *Algebra Gruppen - Ringe - Korper*. Spektrum Akademischer Verlag GmbH, 2010.

[6] S. Lang. *Algebra*. Graduate Texts in Mathematics. Springer New York, 2002. URL: `http://books.google.de/books?id=Fge-BwqhqIYC`.

[7] R. Lidl und H. Niederreiter. *Finite Fields*. Encyclopedia of Mathematics and its Applications Bd. 20,Teil 1. Cambridge University Press, 1997. URL: `http://books.google.de/books?id=xqMqxQTFUkMC`.

[8] *Neues Buch*.

[9] A. Scheerhorn. »Applications of Finite Fields«. In: Hrsg. von D. Gollmann. New York, NY, USA: Oxford University Press, Inc., 1996. Kap. Dickson Polynomials and Completely Normal Elements over Finite Fields, S. 47–55.

[10] A. Scheerhorn. »Dickson Polynomials, Completely Normal Polynomials and the Cyclic Module Structure of Specific Extensions of Finite Fields«. In: *Des. Codes Cryptography* 9.2 (1996), S. 193–202.

[11] I. A. Semaev. »Construction of polynomials irreducible over a finite field with linearly independent roots«. In: *Mathematics of the USSR-Sbornik* 63.2 (1989), S. 507.

[12] Z. Wan. *Lectures on Finite Fields and Galois Rings*. World Scientific, 2003.

Anhang A
Tabellen

Im Folgenden stellen wir die mit Hilfe der vorgestellten Algorithmen berechneten Werte vor. Dabei ist folgende Legende zu beachten:

q, p, r sind die Daten des betrachteten Grundkörpers $\mathbb{F}_q$, wobei $q = p^r$ gilt.

$\mathcal{N}(q,n)$ gibt die Anzahl der normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

$\mathcal{CN}(q,n)$ gibt die Anzahl der vollständig normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

$\mathcal{PCN}(q,n)$ gibt die Anzahl der primitiv vollständig normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

$\mathcal{PN}(q,n)$ gibt die Anzahl der primitiv normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

Erzeuger. Hier ist die Anzahl der vollständigen Erzeuger der Zerlegung nach Satz 5.9 gegeben, wobei ein Datum $(k,t,\pi) : N$ bedeutet, dass für den Kreisteilungsmodul $\mathcal{C}_{k,t\pi}$ gerade N vollständige Erzeuger in $\mathbb{F}_q$ existieren.

$(\ .\ )^*$ gibt bei Vorhandensein in der Spalte $\mathcal{CN}(q,n)$ an, ob die aktuelle Körpererweiterung einfach (Definition 5.2) ist. Falls ja, so gilt per definitionem $\mathcal{CN}(q,n) = \mathcal{N}(q,n)$ und $\mathcal{PCN}(q,n) = \mathcal{PN}(q,n)$. Daher sind in den Tabellen mit (primitiv) normalen Elementen lediglich diejenigen Erweiterungen gelistet, die nicht einfach sind.

$(\ .\ )^\dagger$ gibt bei Vorhandensein hinter einem Erzeuger-Datum an, ob dieser regulär ist (Definition 5.10).

Tabelle A.1: Enumerationen $p = 2$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	2	2^*	2	$(1,1,2)^\dagger$: 2
2	2	1	3	3^*	3	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3
2	2	1	4	8^*	4	$(1,1,4)^\dagger$: 8
2	2	1	5	15^*	15	$(1,1,1)^\dagger$: 1, $(5,1,1)^\dagger$: 15
2	2	1	6	12	6	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6
2	2	1	7	49^*	49	$(1,1,1)^\dagger$: 1, $(7,1,1)^\dagger$: 49
2	2	1	8	128^*	56	$(1,1,8)^\dagger$: 128
2	2	1	9	189^*	171	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(9,1,1)^\dagger$: 63

Tabelle A.1: Enumerationen $p = 2$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	10	420	250	$(1,1,2)^\dagger$: 2, $(5,1,2)$: 210
2	2	1	11	1023*	957	$(1,1,1)^\dagger$: 1, $(11,1,1)^\dagger$: 1023
2	2	1	12	768	360	$(1,1,4)^\dagger$: 8, $(3,1,4)$: 96
2	2	1	13	4095*	4095	$(1,1,1)^\dagger$: 1, $(13,1,1)^\dagger$: 4095
2	2	1	14	6272*	4074	$(1,1,2)^\dagger$: 2, $(7,1,2)^\dagger$: 3136
2	2	1	15	10125*	8430	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(5,1,1)^\dagger$: 15, $(15,1,1)^\dagger$: 225
2	2	1	16	32768*	16272	$(1,1,16)^\dagger$: 32768
2	2	1	17	65025*	65025	$(1,1,1)^\dagger$: 1, $(17,1,1)^\dagger$: 65025
2	2	1	18	46872	24948	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6, $(9,1,2)$: 3906
2	2	1	19	262143*	262143	$(1,1,1)^\dagger$: 1, $(19,1,1)^\dagger$: 262143
2	2	1	20	329280	150320	$(1,1,4)^\dagger$: 8, $(5,1,4)$: 41160
2	2	1	21	259308	220374	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(7,3,1)$: 86436
2	2	1	22	2091012	1317250	$(1,1,2)^\dagger$: 2, $(11,1,2)$: 1045506
2	2	1	23	4190209*	4099957	$(1,1,1)^\dagger$: 1, $(23,1,1)^\dagger$: 4190209
2	2	1	24	3145728	1246752	$(1,1,8)^\dagger$: 128, $(3,1,8)$: 24576
2	2	1	25	15728625*	15188050	$(1,1,1)^\dagger$: 1, $(5,1,1)^\dagger$: 15, $(25,1,1)^\dagger$: 1048575
2	2	1	26	33529860	22345232	$(1,1,2)^\dagger$: 2, $(13,1,2)$: 16764930
2	2	1	27	47258883	39950874	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(9,1,1)^\dagger$: 63, $(27,1,1)^\dagger$: 250047
2	2	1	28	102760448*	50821260	$(1,1,4)^\dagger$: 8, $(7,1,4)^\dagger$: 12845056
2	2	1	29	268435455*	266908663	$(1,1,1)^\dagger$: 1, $(29,1,1)^\dagger$: 268435455
2	2	1	30	111132000	55308540	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6, $(5,1,2)$: 210, $(15,1,2)$: 44100
2	2	1	31	887503681*	887503681	$(1,1,1)^\dagger$: 1, $(31,1,1)^\dagger$: 887503681
4	2	2	2	12*	8	$(1,1,2)^\dagger$: 12
4	2	2	3	27*	18	$(1,1,1)^\dagger$: 3, $(3,1,1)^\dagger$: 9
4	2	2	4	192*	96	$(1,1,4)^\dagger$: 192
4	2	2	5	675*	400	$(1,1,1)^\dagger$: 3, $(5,1,1)^\dagger$: 225
4	2	2	6	1728*	792	$(1,1,2)^\dagger$: 12, $(3,1,2)^\dagger$: 144
4	2	2	7	11907*	7784	$(1,1,1)^\dagger$: 3, $(7,1,1)^\dagger$: 3969
4	2	2	8	49152*	24448	$(1,1,8)^\dagger$: 49152
4	2	2	9	107163*	57186	$(1,1,1)^\dagger$: 3, $(3,1,1)^\dagger$: 9, $(9,1,1)^\dagger$: 3969
4	2	2	10	529200	241400	$(1,1,2)^\dagger$: 12, $(5,1,2)$: 44100
4	2	2	11	3139587*	1978020	$(1,1,1)^\dagger$: 3, $(11,1,1)^\dagger$: 1046529
4	2	2	12	7077888*	2803392	$(1,1,4)^\dagger$: 192, $(3,1,4)^\dagger$: 36864
4	2	2	13	50307075*	33525908	$(1,1,1)^\dagger$: 3, $(13,1,1)^\dagger$: 16769025
4	2	2	14	195084288*	96481224	$(1,1,2)^\dagger$: 12, $(7,1,2)^\dagger$: 16257024
8	2	3	2	56*	36	$(1,1,2)^\dagger$: 56
8	2	3	3	441*	378	$(1,1,1)^\dagger$: 7, $(3,1,1)^\dagger$: 63
8	2	3	4	3584*	1512	$(1,1,4)^\dagger$: 3584
8	2	3	5	28665*	23760	$(1,1,1)^\dagger$: 7, $(5,1,1)^\dagger$: 4095
8	2	3	6	218736	117288	$(1,1,2)^\dagger$: 56, $(3,1,2)$: 3906

Tabelle A.1: Enumerationen $p = 2$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
8	2	3	7	823543*	698544	$(1,1,1)^\dagger$: 7, $(7,1,1)^\dagger$: 117649
8	2	3	8	14680064*	5804640	$(1,1,8)^\dagger$: 14680064
8	2	3	9	110270727*	93223872	$(1,1,1)^\dagger$: 7, $(3,1,1)^\dagger$: 63, $(9,1,1)^\dagger$: 250047

Tabelle A.2: Enumerationen $p = 2$

q	p	r	n	$\mathcal{N}(q,n)$	$\mathcal{PN}(q,n)$	Erzeuger
2	2	1	6	24	18	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 12
2	2	1	10	480	290	$(1,1,2)^\dagger$: 2, $(5,1,2)$: 240
2	2	1	12	1536	624	$(1,1,4)^\dagger$: 8, $(3,1,4)$: 192
2	2	1	18	96768	51660	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 12, $(9,1,2)$: 4032
2	2	1	20	491520	225100	$(1,1,4)^\dagger$: 8, $(5,1,4)$: 61440
2	2	1	21	583443	495159	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(7,3,1)$: 194481
2	2	1	22	2095104	1319692	$(1,1,2)^\dagger$: 2, $(11,1,2)$: 1047552
2	2	1	24	6291456	2488320	$(1,1,8)^\dagger$: 128, $(3,1,8)$: 49152
2	2	1	26	33546240	22356074	$(1,1,2)^\dagger$: 2, $(13,1,2)$: 16773120
2	2	1	27	49545027	41883129	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(9,1,1)^\dagger$: 63, $(27,1,1)^\dagger$: 262143

Tabelle A.3: Enumerationen $p = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
3	3	1	2	4*	4	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2
3	3	1	3	18*	9	$(1,1,3)^\dagger$: 18
3	3	1	4	32*	16	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8
3	3	1	5	160*	75	$(1,1,1)^\dagger$: 2, $(5,1,1)^\dagger$: 80
3	3	1	6	324*	144	$(1,1,3)^\dagger$: 18, $(2,1,3)^\dagger$: 18
3	3	1	7	1456*	728	$(1,1,1)^\dagger$: 2, $(7,1,1)^\dagger$: 728
3	3	1	8	1536	576	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(8,1,1)^\dagger$: 48
3	3	1	9	13122*	6075	$(1,1,9)^\dagger$: 13122
3	3	1	10	24960	11160	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(5,2,1)$: 6240
3	3	1	11	117128*	55979	$(1,1,1)^\dagger$: 2, $(11,1,1)^\dagger$: 58564
3	3	1	12	209952*	65424	$(1,1,3)^\dagger$: 18, $(2,1,3)^\dagger$: 18, $(4,1,3)^\dagger$: 648
3	3	1	13	913952*	456976	$(1,1,1)^\dagger$: 2, $(13,1,1)^\dagger$: 456976
3	3	1	14	2114112	1054368	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(7,2,1)$: 528528
3	3	1	15	9447840*	3962700	$(1,1,3)^\dagger$: 18, $(5,1,3)^\dagger$: 524880
3	3	1	16	6291456	2289984	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(8,1,1)^\dagger$: 48, $(16,1,1)^\dagger$: 4096
3	3	1	17	86093440*	43022053	$(1,1,1)^\dagger$: 2, $(17,1,1)^\dagger$: 43046720
3	3	1	18	172186884*	62696736	$(1,1,9)^\dagger$: 13122, $(2,1,9)^\dagger$: 13122
3	3	1	19	774840976*	387177364	$(1,1,1)^\dagger$: 2, $(19,1,1)^\dagger$: 387420488
3	3	1	20	1184481280	423266160	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(5,4,1)$: 37015040

9	3	2	2	64*	32	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8
9	3	2	3	648*	264	$(1,1,3)^\dagger$: 648
9	3	2	4	4096*	1536	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8, $(4,1,1)^\dagger$: 64
9	3	2	5	51200*	23000	$(1,1,1)^\dagger$: 8, $(5,1,1)^\dagger$: 6400
9	3	2	6	419904*	130848	$(1,1,3)^\dagger$: 648, $(2,1,3)^\dagger$: 648
9	3	2	7	4239872*	2115008	$(1,1,1)^\dagger$: 8, $(7,1,1)^\dagger$: 529984
9	3	2	8	16777216*	6117376	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8, $(4,1,1)^\dagger$: 64, $(8,1,1)^\dagger$: 4096
9	3	2	9	344373768*	125421768	$(1,1,9)^\dagger$: 344373768

Tabelle A.4: Enumerationen $p = 3$

q	p	r	n	$\mathcal{N}(q,n)$	$\mathcal{PN}(q,n)$	Erzeuger
3	3	1	8	2048	832	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(8,1,1)^\dagger$: 64
3	3	1	10	25600	11520	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(5,2,1)$: 6400
3	3	1	14	2119936	1057392	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(7,2,1)$: 529984
3	3	1	16	13107200	4790656	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(8,1,1)^\dagger$: 64, $(16,1,1)^\dagger$: 6400
3	3	1	20	1310720000	468392880	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(5,4,1)$: 40960000

Tabelle A.5: Enumerationen $p = 5$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
5	5	1	2	16*	8	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4
5	5	1	3	96*	48	$(1,1,1)^\dagger$: 4, $(3,1,1)^\dagger$: 24
5	5	1	4	256*	64	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(4,1,1)^\dagger$: 16
5	5	1	5	2500*	1130	$(1,1,5)^\dagger$: 2500
5	5	1	6	8448	2376	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(3,2,1)$: 528
5	5	1	7	62496*	31248	$(1,1,1)^\dagger$: 4, $(7,1,1)^\dagger$: 15624
5	5	1	8	147456*	44928	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(4,1,1)^\dagger$: 16, $(8,1,1)^\dagger$: 576
5	5	1	9	1499904*	687132	$(1,1,1)^\dagger$: 4, $(3,1,1)^\dagger$: 24, $(9,1,1)^\dagger$: 15624
5	5	1	10	6250000*	1862760	$(1,1,5)^\dagger$: 2500, $(2,1,5)^\dagger$: 2500
5	5	1	11	39037504*	19518752	$(1,1,1)^\dagger$: 4, $(11,1,1)^\dagger$: 9759376
5	5	1	12	71368704	18178944	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(3,2,1)$: 528, $(4,1,1)^\dagger$: 16, $(12,1,1)$: 528

Tabelle A.6: Enumerationen $p = 7$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
7	7	1	2	36*	16	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6
7	7	1	3	216*	72	$(1,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36
7	7	1	4	1728*	480	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(4,1,1)^\dagger$: 48
7	7	1	5	14400*	4800	$(1,1,1)^\dagger$: 6, $(5,1,1)^\dagger$: 2400

Tabelle A.6: Enumerationen $p = 7$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
7	7	1	6	46656*	14832	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36, $(6,1,1)^\dagger$: 36
7	7	1	7	705894*	227010	$(1,1,7)^\dagger$: 705894
7	7	1	8	3815424	1016320	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(4,1,1)^\dagger$: 48, $(8,1,1)^\dagger$: 2208
7	7	1	9	25264224*	7753806	$(1,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36, $(9,1,1)^\dagger$: 116964
7	7	1	10	207187200	62435920	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(5,2,1)$: 5755200
7	7	1	11	1694851488*	564443264	$(1,1,1)^\dagger$: 6, $(11,1,1)^\dagger$: 282475248

Tabelle A.7: Enumerationen $n = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	3	3*	3	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3
3	3	1	3	18*	9	$(1,1,3)^\dagger$: 18
4	2	2	3	27*	18	$(1,1,1)^\dagger$: 3, $(3,1,1)^\dagger$: 9
5	5	1	3	96*	48	$(1,1,1)^\dagger$: 4, $(3,1,1)^\dagger$: 24
7	7	1	3	216*	72	$(1,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36
8	2	3	3	441*	378	$(1,1,1)^\dagger$: 7, $(3,1,1)^\dagger$: 63
9	3	2	3	648*	264	$(1,1,3)^\dagger$: 648
11	11	1	3	1200*	384	$(1,1,1)^\dagger$: 10, $(3,1,1)^\dagger$: 120
13	13	1	3	1728*	576	$(1,1,1)^\dagger$: 12, $(3,1,1)^\dagger$: 144
16	2	4	3	3375*	1440	$(1,1,1)^\dagger$: 15, $(3,1,1)^\dagger$: 225
17	17	1	3	4608*	2304	$(1,1,1)^\dagger$: 16, $(3,1,1)^\dagger$: 288
19	19	1	3	5832*	1944	$(1,1,1)^\dagger$: 18, $(3,1,1)^\dagger$: 324
23	23	1	3	11616*	4440	$(1,1,1)^\dagger$: 22, $(3,1,1)^\dagger$: 528
25	5	2	3	13824*	3888	$(1,1,1)^\dagger$: 24, $(3,1,1)^\dagger$: 576
27	3	3	3	18954*	8748	$(1,1,3)^\dagger$: 18954
29	29	1	3	23520*	9180	$(1,1,1)^\dagger$: 28, $(3,1,1)^\dagger$: 840
31	31	1	3	27000*	7200	$(1,1,1)^\dagger$: 30, $(3,1,1)^\dagger$: 900
32	2	5	3	31713*	26100	$(1,1,1)^\dagger$: 31, $(3,1,1)^\dagger$: 1023
37	37	1	3	46656*	13176	$(1,1,1)^\dagger$: 36, $(3,1,1)^\dagger$: 1296
41	41	1	3	67200*	26880	$(1,1,1)^\dagger$: 40, $(3,1,1)^\dagger$: 1680
43	43	1	3	74088*	21168	$(1,1,1)^\dagger$: 42, $(3,1,1)^\dagger$: 1764
47	47	1	3	101568*	46596	$(1,1,1)^\dagger$: 46, $(3,1,1)^\dagger$: 2208
49	7	2	3	110592*	34272	$(1,1,1)^\dagger$: 48, $(3,1,1)^\dagger$: 2304
53	53	1	3	146016*	57744	$(1,1,1)^\dagger$: 52, $(3,1,1)^\dagger$: 2808
59	59	1	3	201840*	97440	$(1,1,1)^\dagger$: 58, $(3,1,1)^\dagger$: 3480
61	61	1	3	216000*	52848	$(1,1,1)^\dagger$: 60, $(3,1,1)^\dagger$: 3600
64	2	6	3	250047*	134136	$(1,1,1)^\dagger$: 63, $(3,1,1)^\dagger$: 3969
67	67	1	3	287496*	72000	$(1,1,1)^\dagger$: 66, $(3,1,1)^\dagger$: 4356
71	71	1	3	352800*	120960	$(1,1,1)^\dagger$: 70, $(3,1,1)^\dagger$: 5040
73	73	1	3	373248*	124416	$(1,1,1)^\dagger$: 72, $(3,1,1)^\dagger$: 5184
79	79	1	3	474552*	122040	$(1,1,1)^\dagger$: 78, $(3,1,1)^\dagger$: 6084
81	3	4	3	524880*	163584	$(1,1,3)^\dagger$: 524880

Tabelle A.7: Enumerationen $n = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
83	83	1	3	564816*	260280	$(1,1,1)^\dagger$: 82, $(3,1,1)^\dagger$: 6888
89	89	1	3	696960*	316800	$(1,1,1)^\dagger$: 88, $(3,1,1)^\dagger$: 7920
97	97	1	3	884736*	294912	$(1,1,1)^\dagger$: 96, $(3,1,1)^\dagger$: 9216
121	11	2	3	1728000*	364608	$(1,1,1)^\dagger$: 120, $(3,1,1)^\dagger$: 14400
125	5	3	3	1937376*	887220	$(1,1,1)^\dagger$: 124, $(3,1,1)^\dagger$: 15624
128	2	7	3	2080641*	1764882	$(1,1,1)^\dagger$: 127, $(3,1,1)^\dagger$: 16383
169	13	2	3	4741632*	1325376	$(1,1,1)^\dagger$: 168, $(3,1,1)^\dagger$: 28224
243	3	5	3	14289858*	5994450	$(1,1,3)^\dagger$: 14289858
256	2	8	3	16581375*	6561792	$(1,1,1)^\dagger$: 255, $(3,1,1)^\dagger$: 65025
289	17	2	3	23887872*	6283008	$(1,1,1)^\dagger$: 288, $(3,1,1)^\dagger$: 82944
343	7	3	3	40001688*	12279276	$(1,1,1)^\dagger$: 342, $(3,1,1)^\dagger$: 116964
361	19	2	3	46656000*	10584000	$(1,1,1)^\dagger$: 360, $(3,1,1)^\dagger$: 129600
512	2	9	3	133955073*	113245776	$(1,1,1)^\dagger$: 511, $(3,1,1)^\dagger$: 262143
529	23	2	3	147197952*	34848000	$(1,1,1)^\dagger$: 528, $(3,1,1)^\dagger$: 278784
625	5	4	3	242970624*	61910784	$(1,1,1)^\dagger$: 624, $(3,1,1)^\dagger$: 389376
729	3	6	3	386889048*	140901120	$(1,1,3)^\dagger$: 386889048
841	29	2	3	592704000*	122760576	$(1,1,1)^\dagger$: 840, $(3,1,1)^\dagger$: 705600
961	31	2	3	884736000*	191020032	$(1,1,1)^\dagger$: 960, $(3,1,1)^\dagger$: 921600

Tabelle A.8: Enumerationen $n = 4$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	4	8*	4	$(1,1,4)^\dagger$: 8
3	3	1	4	32*	16	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8
4	2	2	4	192*	96	$(1,1,4)^\dagger$: 192
5	5	1	4	256*	64	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(4,1,1)^\dagger$: 16
7	7	1	4	1728*	480	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(4,1,1)^\dagger$: 48
8	2	3	4	3584*	1512	$(1,1,4)^\dagger$: 3584
9	3	2	4	4096*	1536	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8, $(4,1,1)^\dagger$: 64
11	11	1	4	12000*	3200	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10, $(4,1,1)^\dagger$: 120
13	13	1	4	20736*	4352	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12, $(4,1,1)^\dagger$: 144
16	2	4	4	61440*	30720	$(1,1,4)^\dagger$: 61440
17	17	1	4	65536*	16896	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16, $(4,1,1)^\dagger$: 256
19	19	1	4	116640*	31104	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18, $(4,1,1)^\dagger$: 360
23	23	1	4	255552*	60640	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22, $(4,1,1)^\dagger$: 528
25	5	2	4	331776*	101376	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24, $(4,1,1)^\dagger$: 576
27	3	3	4	492128*	154368	$(1,1,1)^\dagger$: 26, $(2,1,1)^\dagger$: 26, $(4,1,1)^\dagger$: 728
29	29	1	4	614656*	139776	$(1,1,1)^\dagger$: 28, $(2,1,1)^\dagger$: 28, $(4,1,1)^\dagger$: 784
31	31	1	4	864000*	207360	$(1,1,1)^\dagger$: 30, $(2,1,1)^\dagger$: 30, $(4,1,1)^\dagger$: 960
32	2	5	4	1015808*	465000	$(1,1,4)^\dagger$: 1015808
37	37	1	4	1679616*	420864	$(1,1,1)^\dagger$: 36, $(2,1,1)^\dagger$: 36, $(4,1,1)^\dagger$: 1296
41	41	1	4	2560000*	564224	$(1,1,1)^\dagger$: 40, $(2,1,1)^\dagger$: 40, $(4,1,1)^\dagger$: 1600
43	43	1	4	3259872*	659712	$(1,1,1)^\dagger$: 42, $(2,1,1)^\dagger$: 42, $(4,1,1)^\dagger$: 1848
47	47	1	4	4672128*	1036288	$(1,1,1)^\dagger$: 46, $(2,1,1)^\dagger$: 46, $(4,1,1)^\dagger$: 2208

Tabelle A.8: Enumerationen $n = 4$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
49	7	2	4	5308416*	1413120	$(1,1,1)^\dagger$: 48, $(2,1,1)^\dagger$: 48, $(4,1,1)^\dagger$: 2304
53	53	1	4	7311616*	1794816	$(1,1,1)^\dagger$: 52, $(2,1,1)^\dagger$: 52, $(4,1,1)^\dagger$: 2704
59	59	1	4	11706720*	3014144	$(1,1,1)^\dagger$: 58, $(2,1,1)^\dagger$: 58, $(4,1,1)^\dagger$: 3480
61	61	1	4	12960000*	3340800	$(1,1,1)^\dagger$: 60, $(2,1,1)^\dagger$: 60, $(4,1,1)^\dagger$: 3600
64	2	6	4	16515072*	6531840	$(1,1,4)^\dagger$: 16515072
67	67	1	4	19549728*	4453760	$(1,1,1)^\dagger$: 66, $(2,1,1)^\dagger$: 66, $(4,1,1)^\dagger$: 4488
71	71	1	4	24696000*	5644800	$(1,1,1)^\dagger$: 70, $(2,1,1)^\dagger$: 70, $(4,1,1)^\dagger$: 5040
73	73	1	4	26873856*	6279168	$(1,1,1)^\dagger$: 72, $(2,1,1)^\dagger$: 72, $(4,1,1)^\dagger$: 5184
79	79	1	4	37964160*	9345024	$(1,1,1)^\dagger$: 78, $(2,1,1)^\dagger$: 78, $(4,1,1)^\dagger$: 6240
81	3	4	4	40960000*	14962688	$(1,1,1)^\dagger$: 80, $(2,1,1)^\dagger$: 80, $(4,1,1)^\dagger$: 6400
83	83	1	4	46314912*	9351040	$(1,1,1)^\dagger$: 82, $(2,1,1)^\dagger$: 82, $(4,1,1)^\dagger$: 6888
89	89	1	4	59969536*	13620480	$(1,1,1)^\dagger$: 88, $(2,1,1)^\dagger$: 88, $(4,1,1)^\dagger$: 7744
97	97	1	4	84934656*	19390976	$(1,1,1)^\dagger$: 96, $(2,1,1)^\dagger$: 96, $(4,1,1)^\dagger$: 9216
121	11	2	4	207360000*	54374400	$(1,1,1)^\dagger$: 120, $(2,1,1)^\dagger$: 120, $(4,1,1)^\dagger$: 14400
125	5	3	4	236421376*	60235200	$(1,1,1)^\dagger$: 124, $(2,1,1)^\dagger$: 124, $(4,1,1)^\dagger$: 15376
128	2	7	4	266338304*	131721408	$(1,1,4)^\dagger$: 266338304
169	13	2	4	796594176*	171343872	$(1,1,1)^\dagger$: 168, $(2,1,1)^\dagger$: 168, $(4,1,1)^\dagger$: 28224
243	3	5	4	3458087072*	1235872000	$(1,1,1)^\dagger$: 242, $(2,1,1)^\dagger$: 242, $(4,1,1)^\dagger$: 59048

Tabelle A.9: Enumerationen $n = 6$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	6	12	6	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6
3	3	1	6	324*	144	$(1,1,3)^\dagger$: 18, $(2,1,3)^\dagger$: 18
4	2	2	6	1728*	792	$(1,1,2)^\dagger$: 12, $(3,1,2)^\dagger$: 144
5	5	1	6	8448	2376	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(3,2,1)$: 528
7	7	1	6	46656*	14832	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36, $(6,1,1)^\dagger$: 36
8	2	3	6	218736	117288	$(1,1,2)^\dagger$: 56, $(3,1,2)$: 3906
9	3	2	6	419904*	130848	$(1,1,3)^\dagger$: 648, $(2,1,3)^\dagger$: 648
11	11	1	6	1416000	298848	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10, $(3,2,1)$: 14160
13	13	1	6	2985984*	834048	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12, $(3,1,1)^\dagger$: 144, $(6,1,1)^\dagger$: 144
16	2	4	6	13824000*	5469696	$(1,1,2)^\dagger$: 240, $(3,1,2)^\dagger$: 57600
17	17	1	6	21086208	5546304	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16, $(3,2,1)$: 82368
19	19	1	6	34012224*	7711200	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18, $(3,1,1)^\dagger$: 324, $(6,1,1)^\dagger$: 324
23	23	1	6	134420352	31821840	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22, $(3,2,1)$: 277728
25	5	2	6	191102976*	48691008	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24, $(3,1,1)^\dagger$: 576, $(6,1,1)^\dagger$: 576

Tabelle A.9: Enumerationen $n = 6$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
27	3	3	6	359254116*	130838112	$(1,1,3)^{\dagger}$: 18954, $(2,1,3)^{\dagger}$: 18954
29	29	1	6	551873280	114307056	$(1,1,1)^{\dagger}$: 28, $(2,1,1)^{\dagger}$: 28, $(3,2,1)$: 703920
31	31	1	6	729000000*	157394880	$(1,1,1)^{\dagger}$: 30, $(2,1,1)^{\dagger}$: 30, $(3,1,1)^{\dagger}$: 900, $(6,1,1)^{\dagger}$: 900
32	2	5	6	1037141952	516358800	$(1,1,2)^{\dagger}$: 992, $(3,1,2)$: 1045506
37	37	1	6	2176782336*	548654688	$(1,1,1)^{\dagger}$: 36, $(2,1,1)^{\dagger}$: 36, $(3,1,1)^{\dagger}$: 1296, $(6,1,1)^{\dagger}$: 1296
41	41	1	6	4510464000	1028522880	$(1,1,1)^{\dagger}$: 40, $(2,1,1)^{\dagger}$: 40, $(3,2,1)$: 2819040
43	43	1	6	5489031744*	1304511264	$(1,1,1)^{\dagger}$: 42, $(2,1,1)^{\dagger}$: 42, $(3,1,1)^{\dagger}$: 1764, $(6,1,1)^{\dagger}$: 1764