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

15. November 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. Wir beginnen dabei bei der Gruppentheorie und zitieren einige Aussagen über zyklische Gruppen. Diese werden uns später helfen, die Untergruppe der (primitiven) Einheitswurzeln in Kapitel 2 zu verstehen. Im anschließend Abschnitt rekapitulieren wir ein wenig die Galoistheorie von endlichen Körpern. Insbesondere wollen wir wiederholen, dass diese zyklisch ist und von einem speziellen Automorphismus erzeugt wird.

1.1 Ein wenig Gruppentheorie

[15, 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 $\frac{m}{\operatorname{ggT}(m,k)}$.

(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 $\varphi(f)$ 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 $\operatorname{ggT}(r,m) = 1$.

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 (Eulersche Phifunktion).

Die Funktion

$$\begin{array}{rcl}\varphi: \mathbb{N}^* & \to & \mathbb{N}^*, \\ n & \mapsto & |\{a \in \mathbb{N}: 1 \leq a \leq n, \operatorname{ggT}(a,n) = 1\}|\end{array}$$

heißt *Eulersche φ-Funktion*.

Definition 1.3 (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 1.4 (Rechenregeln der Eulerschen Phifunktion). *Sei $a, b \in \mathbb{N}^*$, so gilt*

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

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

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

Zyklische Gruppen und endliche Körper hängen eng zusammen, da bekanntlich die multiplikative Gruppe eines endlichen Körpers immer zyklisch ist. Dies können wir nutzen, um Erzeugern (im Sinne der Gruppentheorie) der multiplikativen eines endlichen Körpers einen Namen zu geben.

Satz 1.5.

Die multiplikative Gruppe eines endlichen Körpers ist zyklisch.

Beweis. [15, Theorem 2.8]. □

Definition 1.6 (primitiv).

Sei $\mathbb{F}_q$ ein endlicher Körper. $u \in \mathbb{F}_q$ heißt *primitiv* (oder *primitives Element*), falls $\langle u \rangle = \mathbb{F}_q^*$, also u ein Erzeuger der multiplikativen Gruppe $\mathbb{F}_q^*$ ist.

Bemerkung 1.7. Es ist klar, dass $u \in \mathbb{F}_q$ genau dann primitiv ist, wenn $\operatorname{ord}(u) = q - 1$, also seine gruppentheoretische Ordnung in $\mathbb{F}_q^*$ genau der Gruppenordnung entspricht.

1.2 Automorphismen über endlichen Körpern

Satz 1.8.

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

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

ein Automorphismus auf F.

Beweis. [21, Corollary 3.18]. □

Bemerkung 1.9. 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 1.10.

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

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

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 $k = 1, \dots, n-1$, 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 1.11.

Es gilt

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

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

Beweis. [21, 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 1.12 (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. [11, Satz 27.2]. □

Mit Satz 1.11 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 1.13.

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. [15, 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 [15] und [21].

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. [15, 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. [15, Theorem 2.45 (ii)]). □

Definition 2.6.

Für zwei teilerfremde natürliche Zahlen q, n größer Null sei

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

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* $\operatorname{ggT}(n, q) = 1$, $\operatorname{ggT}(m, q) = 1$ *und* $\operatorname{ggT}(m, n) = 1$, *so gilt*

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

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

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

(2) Nach dem Chinesischen Restsatz (z.B. [3, Kapitel 2 Satz 12]) ist

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

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 $\operatorname{ord}_\bullet(q)$ 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 $\operatorname{ggT}(q,n) = 1$. Dann zerfällt das n-te Kreisteilungspolynom $\Phi_n(X)$ über $\mathbb{F}_q$ in

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

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

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 1.13 auch

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

Nullstellen von $f(X)$. Jedoch sind offenbar nur $\operatorname{ord}_n(q)$ dieser verschieden und da f als irreduzibles Polynom wieder nach Satz 1.13 nur einfache Nullstellen besitzt, können wir folgern, dass $\deg f = \operatorname{ord}_n(q)$. 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. □

Im Beweis obigen Satzes haben wir gesehen, dass die Wirkung der Galoisgruppe $\operatorname{Gal}(\mathbb{F}_{q^n} \mid \mathbb{F}_q)$ auf der Menge der primitiven n-ten Einheitswurzeln $C^{(n)}$ (die Wirkung ist selbstredend durch Einsetzen gegeben) diese in Teilmengen der Mächtigkeit $\operatorname{ord}_n(q)$ 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 2.9.

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

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

Ein vollständiges Repräsentantensystem von Nebenklassen von $M_q(1 \bmod m)$ in $\mathbb{Z}_m$ sei mit $R_q(m)$ bezeichnet. Für $l \in M_q(j \bmod m)$ bezeichne ferner $r_q(l \bmod m) := |\{lq^i : \; i \in \mathbb{N}\}|$ die Länge der zugehörigen Bahn.

Bemerkung 2.10. Per Definition von $\operatorname{ord}_m(q)$ ist für $l \neq 0$

$$r_q(l \bmod m) = \operatorname{ord}_{\frac{m}{\operatorname{ggT}(m,l)}}(q)\,.$$

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 $\Phi_1(X)$.

$$\begin{aligned}\Phi_3(X) &= X^2 + X + 1\\ &= (X-\zeta^7)(X-\zeta^{14}) ,\\ \Phi_1(X) &= X - 1\\ &= X - \zeta^0 .\end{aligned}$$

▲

Nun können wir uns überlegen, ob und wie unterschiedliche Kreisteilungspolynome zusammenhängen und kommen dabei auf die bekannten Resultate, die z.B. in [5, 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. [5, 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. [5, 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 1.8 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

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

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

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

wobei an Lemma 1.4 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 1.8). 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. [7, 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) $\operatorname{ord}_{ne}(q) = n$,

(2) $\operatorname{ggT}(\frac{q-1}{e}, n) = 1$ *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. [7, 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 $l := \pi_m(q-1)$, $a := \operatorname{ggT}(l, m)$ und $I_a := \{j \in \mathbb{N}^* : \ j \geq a, \ \operatorname{ggT}(j, a) = 1\}$. 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 $\operatorname{ord}_a(q) = 1$. Dies ist klar, da l per definitionem $q-1$ teilt und $a = \operatorname{ggT}(l, m)$. 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 $\nu(a) = \nu(\operatorname{ggT}(l, m)) = \nu(m)$ 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 [5, 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 $\mathrm{ggT}(m,t) = 1$. Definieren wir ferner

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

so gilt:

(1) Ist $f(x) \mid \Phi_m(x)$ ein über $\mathbb{F}_q$ irreduzibler monischer Teiler des m-ten Kreisteilungspolynoms, 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.

(2) Sind $f(x) \mid \Phi_m(x)$ und $g(x) \mid \Phi_m(x)$ zwei teilerfremde, monische, über $\mathbb{F}_q$ irreduzible Teiler des m-ten Kreisteilungspolynoms, so sind auch $f(x^t)$ und $g(x^t)$ 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

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

Notiert wieder σ der Frobenius von $\mathbb{F}_q$, so sind nach Satz 1.13 $\sigma^j(\alpha)$, $j = 0, \ldots, \delta-1$ 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

$$\dot{\bigcup_{j=0}^{\delta-1}} \sigma^j(\beta)\,E^{(t)} \;=\; \dot{\bigcup_{j=0}^{\delta-1}} \dot{\bigcup_{d\mid t}} \beta^{q^j}\,C^{(d)} \;=\; \dot{\bigcup_{d\mid t}} \dot{\bigcup_{j=0}^{\delta-1}} \beta^{q^j}\,C^{(d)} \;=:\; \dot{\bigcup_{d\mid t}} N_d\,. \tag{2.1}$$

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 $\operatorname{ord}(\zeta) = md$, 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

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

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 in (1) gezeigt. Der Zusatz (2) folgt sofort, denn ist $\alpha_f \in C^{(m)}$ bzw. $\alpha_g \in C^{(m)}$ Nullstelle von f bzw. g, so gehören diese zu verschiedenen σ-invarianten Teilmengen von $C^{(m)}$ (vgl. auch Beispiel 2.11) und folglich gehören auch $\beta_f \in C^{(m)}$ bzw. $\beta_g \in C^{(m)}$ mit $\beta_f^t = \alpha_f$ bzw. $\beta_g^t = \alpha_g$ zu verschiedenen und damit disjunkten σ-invarianten Teilmengen von $\mathcal{C}^{(m)}$. □

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 $\bmod 20$ 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:

$$f(x^3) = \Big(\overbrace{x^4 + x^3 + x^2 + x + 1}^{d=1\mid 4}\Big) \cdot \Big(\overbrace{x^4 + 2x^3 + x^2 + 2x + 1}^{d=2\mid 4}\Big) \cdot \Big(\overbrace{(x^4 + x^3 + 2x + 1)(x^4 + 2x^3 + x + 1)}^{d=4\mid 4}\Big)$$

▲

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. [12] oder [10], 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. [11, 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. [11, 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:

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

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. [12, 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 ([12, 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 [10, 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 $i = 1, \ldots, k$

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

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*

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

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*

$$0 = Z_\alpha \subsetneq Z_{\alpha-1} \subsetneq \dots \subsetneq Z_0 = Z\,,$$

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 $\chi_\tau(x) := \det(x \operatorname{id}_V - \tau) \in \mathbb{K}[x]$. ▲

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

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

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 1.12) ist $\operatorname{id}_E, \sigma, \dots, \sigma^{n-1}$ linear unabhängig über E, also insbesondere über F, und $\sigma^n = \operatorname{id}_E$.

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

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

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 $\operatorname{Ord}_\tau(v) := g(x)$.

(2) $\mathbb{K}[\tau] \cdot v := \operatorname{im} \psi_v$ 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

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

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

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

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: $\mathrm{im}\,\psi_v \cong \mathbb{K}[x]/\ker\psi_v$ 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

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

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 [5, 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 [5, 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 1.11, 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

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

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:

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

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 $n \in \mathcal{S}_q$.

Lemma 4.6. *Seien $q,m \in \mathbb{N}^*$ mit $q,m > 1$ und $\nu(m) \mid \nu(q-1)$. Dann gilt:*

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

$$\operatorname{ord}_m(q) \;=\; \frac{m}{\operatorname{ggT}(\pi_m(q-1),m)}\,.$$

(2) Ist $m \equiv 0 \bmod 4$ und $q \equiv 2 \bmod 4$, so gilt

$$\operatorname{ord}_m(q) \;=\; 2\,\frac{m}{\operatorname{ggT}(\pi_m(q^2-1),m)}\,.$$

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

$$\operatorname{ord}_m(q) \;=\; \operatorname{ord}_{\frac{m}{2}}(q) \;=\; \frac{m}{\operatorname{ggT}(\pi_m(q-1),m)}\,.$$

Beweis. [5, Lemma 19.6]. □

Lemma 4.7. *Seien $q,m > 1$ zwei teilerfremde ganze Zahlen. Setze $s := \operatorname{ord}_{\nu(m)}(q)$, so gilt*

$$\operatorname{ord}_m(q) \;=\; s \operatorname{ord}_m(q^s)\,.$$

Beweis. [5, Lemma 19.7]. □

Satz 4.8.

Sei $F = \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Seien $l := \pi_m(q-1)$, $a := \operatorname{ggT}(l,m)$ 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) $\operatorname{Ord}_q(u)$ *ist ein irreduzibler Teiler 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 nach Lemma 4.6

$$[F(u):F] = \operatorname{ord}_{ml}(q) = \frac{ml}{\operatorname{ggT}(\pi_{ml}(q-1), ml} = \frac{ml}{l} = m\,,$$

da $\nu(ml) = \nu(m)$, wegen $l = \pi_m(q-1)$.

(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 $i \in I_a$:

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

wobei wie üblich $\sigma : E \to E, x \mapsto x^q$ den Frobenius von $E \mid F$ meint. Wir wollen nun zeigen, dass $\operatorname{Ord}_q(u) = (x^{\frac{m}{a}} - \zeta^i)$ 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:

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

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 4.9. *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*

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

Dann ist

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

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

Beweis. Da $\mathrm{ord}\left(\lambda^{\frac{\pi_n(q-1)}{a(q,m}}\right) = a(q,m)$ und für $m \mid n$ offensichtlich auch $m \in \mathcal{S}_q$, ist obige Aussage lediglich eine Anwendung von Satz 4.8. □

Satz 4.10.

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 4.9

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

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 4.8 anwenden können aus. Es ist trivialerweise

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

und damit sind alle Voraussetzungen erfüllt. □

Bemerkung 4.11. Wie wir in Beispiel 4.4 und Satz 4.8 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 4.12. *Seien die Voraussetzungen wie in Satz 4.8, 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 nun θ eine primitive (nf)-te Einheitswurzel für $l \mid f \mid q-1$, so ist $\mathrm{Ord}_q(\theta)$ ein irreduzibler Teiler von $\Phi_m(x)$ über $F[x]$.*

Beweis. Sei $f = le$ mit $\mathrm{ggT}(e,l) = 1$. 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 4.8 noch einmal nachvollziehen – zu zeigen, dass $\mathrm{ggT}(mle, ck) = ce$ für $c := \mathrm{kgV}(m,l)$ und $q^{\frac{m}{a}} - 1 = ck$ für ein k mit $\mathrm{ggT}(k, ml) = 1$. Da $\mathrm{ggT}(e,l) = 1$, ist per definitionem von l auch $\mathrm{ggT}(e,m) = 1$ und damit auch $\mathrm{ggT}(e,c) = 1$. 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 $\mathrm{ggT}(\bar{k}, e) = 1$. Damit folgt $\mathrm{ggT}(mle, ck) = ce$ und wir haben $\mathrm{ord}(\theta^{ck}) = \frac{mle}{\mathrm{ggT}(mle,ck)} = \frac{mle}{ce} = a$. □

Als Korollar dieses Lemmas können wir einen Satz von Semaev 1989 in [20] beweisen, welcher erneut von Blake, Gao und Mullin 1997 in [1] bewiesen wurde:

Satz 4.13 ([1, Theorem 2.7], [20]).

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 4.12, 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 [5, 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 [5, Theorem 22.5] wiederzufinden ist.

Lemma 4.14. *Seien die Voraussetzungen wie in Satz 4.8, 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 dann u eine primitive (ml)-te Einheitswurzel mit $\operatorname{Ord}_q(u) = f(x)$ für $f(x)$ einen irreduziblen monischen Teiler von $\Phi_m(x)$, so gilt für $v \in E := \mathbb{F}_{q^m}$:*

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

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

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

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 4.15. 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 4.16 (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 4.17.

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. Wie zu beginn dieses Abschnitts erwähnt, sieht man leicht, dass $m \in \mathcal{S}_{q^s}$.

(1) Analog zum Beweis von Satz 4.8 folgern wir $\mathrm{ord}_{ml}(q) = sm$.

(2) Nach Satz 4.8 ist nun $g(x) := \mathrm{Ord}_{q^s}$ ein über $K := \mathbb{F}_{q^s}$ irreduzibler Teiler von $\Phi_m(x)$. Sei nun $\zeta \in K$ eine primitive b-te Einheitswurzel, so gilt ohne Einschränkung

$$g(x) = x^{\frac{m}{b}} - \zeta$$

Definiere nun $f(x) := \prod_{i=0}^{s-1}(x^{\frac{m}{b}} - \zeta^{q^i})$, so ist $f(x)$ ein irreduzibler monischer Teiler von $\Phi_m(x)$ über F, wie man sich leicht überlegt, betrachtet man den Zerfall von $\Phi_m(x)$ über F. Per definitionem von $g(x)$ ist auch $f(\sigma^s)(u) = 0$. Also $\mathrm{Ord}_q(u) \mid f(x^s)$. Es bleibt nun zu zeigen, dass $\mathrm{Ord}_q(u) = f(x^s)$: Schreibe dazu $h(x) := \mathrm{Ord}_q(u)$. Betrachten wir nun

$$\begin{aligned} \Omega_F : \ F[x] &\to E' := \mathbb{F}_{q^{ms}}, \\ p(x) &\mapsto p(\sigma^s)(u), \end{aligned}$$

so stellen wir fest, dass $\ker \Omega_F = (f(x))$, da $f(x)$ ist irreduzibel über F ist. Bezeichne nun $M := \mathrm{im}\, \Omega_F$. Dann ist

$$\dim_F(M) = \deg f = \mathrm{ord}_m(q) = s\,\frac{m}{b},$$

wobei letzte Gleichheit aus Lemma 4.6 und Lemma 4.7 folgt. Analog sehen wir ein, dass für $\Omega_K : K[x] \to E'$, $p(x) \mapsto p(\sigma^s)(u)$ wegen der Irreduzibiltät von $g(x)$ über K

$$\ker \Omega_K = (g(x)) \quad \text{und} \quad \dim_F(N) = [K : F] \dim_K(N) = s\,\frac{m}{b},$$

für $N := \mathrm{im}\, \Omega_K$. Ferne ist klar, dass $M \subseteq N$, da $\Omega_F = \Omega_K\,|_{F[x]}$, womit wir durch Gleichheit der F-Dimensionen $M = N$ schließen können. **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 4.18.

Seien alle Voraussetzungen und Notationen wie in Satz 4.17. 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 $p \nmid \bar{s}$, 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

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

wobei wir uns hierfür noch einmal an Satz 2.20 erinnern müssen, wo wir gezeigt haben, dass in genau dieser Situation

$$f(x^{\bar{s}}) = \prod_{d|m} \prod_{i=1}^{\Delta_q(m,d)} f_{d,i}(x)$$

für monische, irreduzible Teiler $f_{d,i}$ von $\Phi_{md}(x)$ für alle $i = 1, \ldots, \Delta_q(m,d)$ und wir damit

$$\mathrm{ggT}\left(f(x^{\bar{s}})^{p^\beta}, \frac{x^{sm}-1}{x^m-1}\right) = \prod_{\substack{d|m \\ d \neq 1}} \prod_{i=1}^{\Delta_q(m,d)} f_{d,i}(x)$$

folgern können. Abschließend sehen wir dass

$$\Delta_q(m,1) = \frac{\varphi(1)\,\mathrm{ord}_m(q)}{\mathrm{ord}_{1\cdot m}(q)} = 1\,.$$

□

Wir können in gewisser Weise sogar eine Verschärfung obiges Satzes angeben und die Verwendung der Spurfunktion etwas erweitern. Wir müssen lediglich dafür sorgen, dass wir ein Element des Körpers erhalten, für den wir Erzeuger (und letztendlich normale Elemente) konstruieren wollen.

Satz 4.19.

Seien die Voraussetzungen wie in Satz 4.18 und sei wiederum u eine primitive (ml)-te Einheitswurzel. Für $g(x) \in F[x]$ gilt: Ist $[F(g(x)\cdot u) : F] = m$, so gilt

$$\mathrm{Ord}_q(g(x)\cdot u) \text{ ist ein monischer irreduzibler Teiler von } \Phi_m(x)\,.$$

Beweis. Auf der einen Seite haben wir

$$\mathrm{Ord}_q(g(x)\cdot u) \mid x^m - 1 = \prod_{d|m} \Phi_m(x)\,,$$

da $g(x)\cdot u$ nach Voraussetzung Grad m über F hat und auf der anderen Seite gilt nach Lemma 3.34

$$\mathrm{Ord}_q(g(x)\cdot u) = \frac{f(x^s)}{\mathrm{ggT}(f(x^s), g(x))} \mid f(x^s)\,.$$

Wir wissen jedoch aus Satz 2.20 (vgl. auch den Beweis von Satz 4.18), dass

$$f(x^s) = \prod_{d|\bar{s}} \prod_{i=1}^{\Delta_q(m,d)} f_{d,i}(x)^{p^\beta}$$

für $s = \bar{s}p^\beta$ mit $p \nmid \bar{s}$ und $f_{d,i}(x)$ monische, irreduzible Teiler von $\Phi_{md}(x)$. Da $p \nmid m$ nach Voraussetzung, kommt $\Phi_m(x)$ in $x^m - 1$ lediglich in einfacher Vielfachheit vor und damit folgt

$$\mathrm{Ord}_q(g(x)\cdot u) = f_{1,1}(x)$$

für den einzigen monischen irreduziblen Teiler in $f(x^s)$ von $\Phi_m(x)$. □

Auch für reguläre Erweiterungen wollen wir ein Analogon von Lemma 4.12 beweisen:

Lemma 4.20. *Seien die Voraussetzungen wie in Satz 4.17, 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 4.12 mit dem Beweis von Satz 4.17. □

4.4 Normalbasen mit Dickson-Polynomen

Definition 4.21 (Dickson-Polynom).

Sei $n \in \mathbb{N}^*$. Für $a \in \mathbb{F}_q$ definieren wir das *n-te Dickson-Polynom erster Art über $\mathbb{F}_q$* (hier auch nur *n-tes Dickson-Polynom*) als

$$D_n(x,a) \;:=\; \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \frac{n}{n-i}\binom{n-i}{i}(-a)^i x^{n-2i}\,.$$

Das *n-te Dickson-Polynom zweiter Art über $\mathbb{F}_q$* ist gegeben durch

$$E_n(x,a) \;:=\; \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-i}{i}(-a)^i x^{n-2i}\,.$$

Bemerkung 4.22. Über den komplexen Zahlen sind die Dickson-Polynome nahe verwandt mit den Chebyshev-Polynomen, wie man z.B. in [15, Absatz nach Corollary 7.15] nachlesen kann. Über endlichen Körpern liefern sie eine spezielle Klasse von Permutations-Polynomen (vgl. [15, Theorem 7.16], [17, Section 9.6]).

Die Dickson-Polynome erfüllen einige interessante Eigenschaften, welche wir im Folgenden zitieren wollen.

Proposition 4.23.

Seien $n \in \mathbb{N}^*$ und $a, y \in \mathbb{F}_q$ beliebig, so gilt

$$D_n(y + ay^{-1}, a) \;=\; y^n + a^n y^{-n}\,.$$

Beweis. [15, Gleichung (7.8)]. □

Proposition 4.24.

Sei $a \in F_q$. Die Dickson-Polynome erster und zweiter Art erfüllen für alle $n \geq 2$ folgende Rekursionsgleichungen:

$$\begin{aligned} D_n(x,a) &= xD_{n-1}(x,a) - aD_{n-2}(x,a)\,, \\ E_n(x,a) &= xE_{n-1}(x,a) - aE_{n-2}(x,a) \end{aligned}$$

mit den Startwerten $D_0(x,a) = 2$ und $D_1(x,a) = x$ bzw. $E_0(x,a) = 1$ und $E_1(x,a) = x$. Ferner gilt

$$D_{mn}(x,a) \;=\; D_m(D_n(x,a), a^n)$$

Beweis. [14, ??] □

Ferner können wir eine relativ einfache Charakterisierung angeben, wann ein Polynom der Form $D_n(x,a) - b$ irreduzibel ist.

Proposition 4.25.

Seien $n \geq 2$ eine ganze Zahl und $a, b \in \mathbb{F}_q$ mit $a \neq 0$. Seien

$$x^2 + bx + a^n \;=\; (x - \beta_1)(x - \beta_2), \qquad \beta_1, \beta_2 \in \mathbb{F}_{q^2}$$

und $e_i = \mathrm{ord}(\beta_i)$ für $i = 1, 2$. Es ist

$$D_n(x,a) + b$$

irreduzibel genau dann, wenn für sowohl (1), als auch (2) für $i = 1, 2$ erfüllt sind:

(1) *Jeder ungerade Primteiler von n teilt e_i, jedoch nicht $\frac{q^2-1}{e_i}$.*

(2) *Ist n gerade, so ist $\mathrm{char}\,\mathbb{F}_q$ ungerade und eine der beiden folgenden Aussagen gilt:*

  (1') *$b^2 - 4a^n \neq 0$ ist ein quadratischer Rest in $\mathbb{F}_q$, $2 \mid e_i$ und falls $4 \mid n$, so $4 \mid (q-1)$.*

  (2') *$b^2 - 4a^n$ ist ein quadratischer Nichtrest in $\mathbb{F}_q$, $-b - 2a^{\frac{n}{2}}$ ist ein quadratischer Rest in $\mathbb{F}_q$, $2 \mid \frac{q^2-1}{e_i}$ und falls $4 \mid n$, so $2 \mid e_i$, aber nicht $\frac{q^2-1}{2e_i}$.*

Beweis. [4, Theorem 4] □

Da die weiteren Aussagen lediglich für ungerade Erweiterungen bewiesen werden, wollen wir obige Proposition in einer handlicheren Form für unsere Zwecke noch einmal verfassen.

Korollar 4.26. *Seien $n \geq 3$ eine ungerade ganze Zahl und $a, b \in \mathbb{F}_q^*$. Seien*

$$x^2 - bx + a^n \;=\; (x - \beta_1)(x - \beta_2), \qquad \beta_1, \beta_2 \in \mathbb{F}_{q^2}$$

und $e_i = \mathrm{ord}(\beta_i)$ für $i = 1, 2$. Es ist

$$D_n(x,a) - b$$

irreduzibel genau dann, wenn $\nu(n) \mid e_i$, aber $\nu(n) \nmid \frac{q^2-1}{e_i}$ für $i = 1, 2$.

Scheerhorn zeigt in [18, 19], dass sich Dickson-Polynome eignen, um Normalbasen zu beschreiben. Im weiteren Verlauf werden wir sogar sehen, dass man daraus vollständig normale Polynome, also Polynome, deren Nullstellen vollständig normale Elemente (Definition 5.1) sind, konstruieren lassen. Alle Folgerungen über (vollständig) normale Polynome basieren jedoch auf zwei zentralen Resultaten über die Modulstruktur der betrachteten Erweiterungskörper, wie die beiden nachstehenden Sätze angeben.

Satz 4.27 ([19, 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 4.28 ([19, 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 $x^n - \beta$. Setze $\gamma := \theta + a\theta^{-1}$, so gilt für $l \in R_q(n) \setminus \{0\}$

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

Genauere Betrachtung, warum $x^2 + bx + a^n$ hier nicht irreduzibel ist!?

In [18, 19] werden die Beweise dieser Resultate ohne Zuhilfenahme der allgemeinen Theorie über (stark) reguläre Erweiterungen geführt, wie wir sie im letzten Abschnitt kennengelernt haben. Wir wollen uns nun im Folgenden überlegen, dass die Theorie der vorherigen Abschnitte an diesem Punk ein besseres Verständnis der Struktur der Zerlegung des Erweiterungskörpers in Teilmoduln liefert. Ferner sind wir in der Lage das vielleicht überraschende Auftauchen von Dickson-Polynomen im Kontext (vollständig) normaler Elemente zu rechtfertigen und die benötigten Eigenschaften herzuleiten, welche an diesem Punkt eine zentrale Rolle spielen und die Verwendung der Dickson-Polynome motivieren.

Beweis (von Satz 4.27). Betrachten wir die vorliegende Situation, so sehen wir, dass $s := \operatorname{ord}_{\nu(n)}(q) = 2$ und damit $n \in \mathcal{S}_{q^2}$. Sei nun $x^2 - bx + a^n = (x-\beta)(x - a^n\beta^{-1})$ über $K := \mathbb{F}_{q^2}$. Dann ist nach Satz 2.16 und Satz 2.17 $x^n - \beta \in \mathbb{F}_{q^2}[x]$ irreduzibel. Ist dann $\theta \in \mathbb{F}_{q^{2n}} =: E'$ eine Wurzel, so wissen wir nach Satz 4.13 und Lemma 4.20 (θ ist eine $(n \operatorname{ord}(\beta))$-te primitive Einheitswurzel), dass $\operatorname{Ord}_q(\theta) = f(x^s)$ für einen irreduziblen Teiler $f(x)$ von $\Phi_n(x)$ über F.

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 4.19 gerade

$$\operatorname{Ord}_q(\theta + a\theta^{-1}) \;=\; \operatorname{Ord}_q((1 + ax^n)\cdot\theta) \;=\; f_{1,1}(x)$$

für einen monischen, irreduziblen Teiler $f_{1,1}(x)$ von $\Phi_n(x)$ (wieder in Notation von Satz 2.20), denn mit Proposition 4.23 sehen wir, dass

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

und da $D_n(x,a) - b \in F[x]$ irreduzibel nach Voraussetzung, hat $\theta + a\theta^{-1}$ Grad n über F. Wählen wir $1 \neq l \in R_q(n)$, so entspricht $\mathrm{Ord}_q(\theta^l) = g(x^s)$ für einen monischen irreduziblen Teiler $g(x) \mid \Phi_m(x)$ mit $m \mid n$ und $\mathrm{ggT}(f,g) = 1$. Wegen $\theta^l + a^l\theta^{-l} = D_l(\theta + a\theta^{-1}, a)$ hat $\theta^l + a^l\theta^{-l}$ Grad kleiner oder gleich n über F (vgl. Bemerkung 4.29) und damit sind auch $\mathrm{Ord}_q(\theta + a\theta^{-1})$ und $\mathrm{Ord}_q(\theta^l + a^l\theta^{-l})$ nach Satz 2.20 (2) teilerfremd. □

Bemerkung 4.29. In der Tag können wir sogar genau angeben, welchen Grad $\theta^l + a^l\theta^{-l}$ über F hat: Ist $l \mid n$, so hat $\theta^l + a^l\theta^{-l}$ Grad $\frac{n}{l}$ über F, denn nach Proposition 4.24 ist

$$\mathbb{F}_q \;\ni\; D_n(\theta + a\theta^{-1}, a) \;=\; D_{\frac{n}{l}}\big(D_l(\theta + a\theta^{-1}, a),\, a^l\big) \;=\; D_{\frac{n}{l}}\big(\theta^l + a^l\theta^{-l}, a^l\big)\,.$$

Für $\mathrm{ggT}(n,l) = 1$ hat $\theta^l + a^l\theta^{-l}$ weiterhin Grad n über F, da $\theta^l + a^l\theta^{-l} = D_l(\theta + a\theta^{-1}, a)$

Beweis (von Satz 4.28). Im Grunde brauchen wir nichts zu zeigen, wenn wir uns überlegen, dass hier n stark regulär ist! Nach Lemma 4.12 ist dann $\mathrm{Ord}_q(\theta)$ ein irreduzibler monischer Teiler von $x^n - 1$. Also werden alle irreduziblen Teilmoduln von $\mathbb{F}_{q^n}$ von θ^l mit $l \in \mathcal{R}_q(n)$ erzeugt. Wie im Beweis von Satz 4.27 ist natürlich

$$D_n(\theta + a\theta^{-1}, a) - b \;=\; 0 \quad \text{und} \quad D_l(\theta + a\theta^{-1}, a) \;=\; \theta^l + a^l\theta^{-l},\ l = 1, \ldots, n\,.$$

Letztlich überlege man sich, dass $\mathrm{Ord}_q(a^l\theta^{-l}) = \mathrm{Ord}_q(\theta^{n-l})$, da $a \in F$. □

Man bemerke, dass im Beweis von Satz 4.27 $\theta + a\theta^{-1}$ lediglich eine Abwandlung von $\theta + \theta^{q^n} = \mathrm{Tr}_{E'|E}(\theta)$ ist! Daher ist das Resultat vielleicht auch nicht besonders überraschend, da wir in Satz 4.18 erkannt haben, wie die Spurfunktion zur Konstruktion von normalen Elementen in regulären Erweiterungen zentral ist. In Notation des vorherigen Abschnittes würde man die beiden Sätze von Scheerhorn vielleicht auf die folgende Art und Weise aufschreiben. Insbesondere erlaubt es die Theorie aus den vorherigen Abschnitten, den Fall $\nu(n) \mid (q-1)$ zu erweitern, um auch dort (unter leichten Zusatzvoraussetzungen) ein normales Element zu konstruieren.

Satz 4.30.

Seien $F := \mathbb{F}_q$ ein endlicher Körper von Charakteristik p und $m \in \mathbb{N}^*$ ungerade mit $p \nmid n$ und $\nu(n) \mid q^2 - 1$. Sind $a, b \in F$, sodass $D_n(x,a) - b \in F[x]$ irreduzibel mit Nullstelle $\gamma \in E := \mathbb{F}_{q^n}$, so gilt:

(1) $E = F(\gamma)$,

(2) Ist $\nu(n) \mid (q+1)$, so gilt: Für alle $l \in R_q(n)$ mit $r_q(l \bmod n) = \mathrm{ord}_q(m)$ für einen Teiler m von n ist $\mathrm{Ord}_q(D_l(\gamma, a))$ ist ein irreduzibler Teiler von $\Phi_m(x)$.

(3) Setze

$$v := \sum_{l \in R_q(n)} D_l(\gamma, a)\,.$$

Dann ist $v \in E$ normal über F, falls $\nu(n) \mid (q+1)$ oder $a^l \neq -1$ für alle $l \in R_q(n)$.

Beweis. Für $\nu(n) \mid (q+1)$ liegt gerade Satz 4.27 vor und es gibt nichts mehr zu zeigen. Aussage (2) haben wir uns bereits in Bemerkung 4.29 überlegt. Für $\nu(n) \mid (q-1)$ bleibt anzugeben, warum v normal über F ist, wenn $a^l \neq -1$ für alle $l \in R_q(n)$. Da in v gerade je zwei Erzeuger pro irreduziblem Teilmodul von E vorkommen, ist sicherzustellen, dass diese sich nicht gegenseitig aufheben: Existierte $a^l = -1$ für ein $l \in R_q(n)$, so wäre

$$\theta^{n-l} + a^l \theta^{n-l} \;=\; 0\,.$$

□

4.4.1 Normale und vollständig normale Polynome mit Dickson-Polynomen

Mit Hilfe dieser Resultate lassen sich nun relativ einfach vollständig normale Polynome angeben, wie sie in [19, Section 3] und [18, Section 4] zu finden sind. Ihre Beweise beinhalten keine besondere Beachtung der Modulstrukturen und sollen daher hier nur nachvollzogen werden. Zunächst brauchen wir jedoch eine Transformation eines Polynoms, welches gerade die Nullstellen desselbigen invertiert.

Definition 4.31 (reziprokes Polynom).

Sei $f(x) \in \mathbb{K}[x]$ ein Polynom über einem Körper $\mathbb{K}$. Dann heißt

$$f^*(x) \;:=\; \tfrac{1}{f(0)}\, x^{\deg f} f\left(\tfrac{1}{x}\right) \quad \in \mathbb{K}[x]$$

reziprokes Polynom von $f(x)$.

Bemerkung 4.32. Man bemerke, dass das reziproke Polynom stets monisch ist.

Bemerkung 4.33. Ist u eine Nullstelle von $f(x)$, so ist u^{-1} eine Nullstelle von $f^*(x)$, wie man sofort erkennen kann.

Satz 4.34 ([19, Theorem 4]).

Seien $n \geq 3$ ein Produkt aus ungeraden Primteilern von $(q+1)$ und $D_n(x,a) - b \in \mathbb{F}_q[x]$ irreduzibel. Seien ferner $s, t \in \mathbb{F}_q$ mit $s \neq 0$. Dann ist

$$\big(D_n(sx+t, a) - b\big)^*$$

normal über $\mathbb{F}_q$ genau dann, wenn für jedes $l \in R_q(n)$ ein $i \in M_q(l \bmod n)$ existiert, so dass $E_{n-1-i}(t,a) \neq 0$.

Beweis. Setzen wir wieder $x^2 - bx + a^n = (x-\beta)(x - a^n\beta^{-1})$ über $\mathbb{F}_{q^2}$ wie im Beweis von Satz 4.27 mit $\theta \in \mathbb{F}_{q^{2n}}$ einer Nullstelle von $x^n - \beta$, so erkennen wir, dass $s(\theta - t + a\theta^{-1})^{-1}$ eine Nullstelle von $\big(D_n(sx+t,a) - b\big)^*$ ist. Dort haben wir ebenfalls gesehen, dass $\{1\} \cup \{\theta^l + a^l\theta^{-l} :\; l = 1, \ldots, n-1\}$ eine $\mathbb{F}_q$-Basis von F_{q^n} bildet. Offensichtlich ist $s(\theta - t + a\theta^{-1})^{-1}$ genau dann normal über $\mathbb{F}_q$, wenn $(\theta + t + a\theta^{-1}$ normal über $\mathbb{F}_q$ ist. Also überlegen wir uns, für welche $\alpha = a_0 + \sum_{i=1}^{n-1} a_i(\theta^i + a^i\theta^{-i})$ das Produkt $\alpha(\theta - t + a\theta^{-1})$ in $\mathbb{F}_q^*$ liegt. Durch Ausmultiplizieren erkennt man, dass die Koeffizienten

a_i gerade die Rekursionsgleichung der Dickson-Polynome zweiter Art erfüllen müssen (siehe [19, Beweis zu Theorem 4] für die konkrete Rechnung). Damit ist $\alpha(\theta - t + a\theta^{-1}) \in \mathbb{F}_q^*$, falls

$$\alpha = E_{n-1}(t,a) + \sum_{i=1}^{n-1} E_{n-1-i}(t,a)(\theta^i a^i \theta^{-i}).$$

Die Aussage folgt dann sofort mit Satz 4.27 (oder Satz 4.30) □

Ferner können wir genau diese Folgerung benutzen, um vollsändig normale Polynome, also solche, die über jedem Zwischenkörper normal sind, anzugeben.

Satz 4.35 ([19, Theorem 5]).

Seien $n \geq 3$ ein Produkt aus ungeraden Primteilern von $(q+1)$ und $D_n(x,a) - b \in \mathbb{F}_q[x]$ irreduzibel. Seien ferner $s, t \in \mathbb{F}_q$ mit $s \neq 0$. Ist d ein Teiler von n, so ist

$$\left(D_n(sx+t,a) - b\right)^*$$

normal über $\mathbb{F}_{q^d}$ genau dann, wenn für jedes $l \in R_{q^d}(\frac{n}{d})$ ein $i \in M_{q^d}(l \bmod \frac{n}{d})$ existiert, so dass $E_{\frac{n}{d}-1-i}(t,a) \neq 0$.

Beweis. Wie wir uns bereits in Bemerkung 4.29 überlegt haben, hat $\theta^{\frac{n}{d}} + a^{\frac{n}{d}}\theta^{-\frac{n}{d}}$ Grad $\frac{n}{d}$ über $\mathbb{F}_q$ und das Minimalpolynom von $(\theta - t + a\theta^{-1})$ über F_{q^d} ist $D_{\frac{n}{d}}(x+t,a) - (\theta^{\frac{n}{d}} + a^{\frac{n}{d}}\theta^{-\frac{n}{d}})$. Dann folgt die Aussage analog zu vorherigem Satz. □

Nun lassen sich einige Korollare ziehen, in denen spezielle Werte für a, b, s, t dazu führen, dass die geforderten Eigenschaften aus Satz 4.34 bzw. Satz 4.35 erfüllt sind. Diese werden wir hier nicht beweisen und der Leser sei auf [19] verwiesen.

Korollar 4.36. *Sei $n \geq 3$ ein Produkt ungerader Primfaktoren von $(q-1)$ und sei $D_n(x,a) - b$ irreduzibel über $\mathbb{F}_q$. Ist ferner $n \not\equiv 0 \bmod 3$, so ist*

$$\left(D_n(x \pm 1, 1) - b\right)^*$$

ein vollständig normales Polynom über $\mathbb{F}_q$.

Beweis. [19, Corollary 2]. □

Korollar 4.37. *Sei $n \geq 3$ ein Produkt ungerader Primfaktoren von $(q-1)$ und sei $D_n(x,a) - b$ irreduzibel über $\mathbb{F}_q$. Dann ist*

$$\left(D_n(x \pm 2, 1) - b\right)^*$$

ein vollständig normales Polynom über $\mathbb{F}_q$.

Beweis. [19, Corollary 3]. □

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 $E \mid K \mid F$, so wollen wir untersuchen, ob ein Element $w \in E$, welches normal über F ist, auch normal über allen Zwischenkörpern bleibt. Solche Elemente wollen wir *vollständig normal* nennen. Ähnlich zu normalen Elementen kann man mit Hilfe der Modulstrukturen von Körpererweiterungen eine Theorie aufbauen, die es erlaubt, vollständig normale Elemente in vollständige Erzeuger (Definition 5.7) zu zerlegen, wie es für normale Elemente aus Korollar 4.3 kennt. Hachenberger konnte in [5] ausarbeiten, wie die simultan auftretenden Modulstrukturen zu behandeln sind. Wir wollen hier die zentralen Resultate lediglich ohne Beweise zitieren. Eine ebenfalls gute Übersicht dazu findet man in [17, Section 5.4]. Wir beginnen bei der grundlegendsten Definition, die sich ja bereits in der Kapitelüberschrift wiederfinden lässt.

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.

Die Begriffe *vollständige Normalbasis*, *vollständig normales Polynom* sind analog zu Definition 4.1 zu setzen.

Ein besonders trivialer Fall würde auftreten, wenn bereits alle normalen Elemente eine Körpererweiterung auch vollständig normal wären. Man kann zeigen, dass dies in der Tat unter gewissen Bedingungen auftreten kann und verleiht dieser Konstellation den Namen *einfach*.

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 von Charakteristik p. Dann sind 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* $\mathrm{ord}_{(\frac{m}{r})'}(q)$.

Dabei ist $\frac{m}{r} = (\frac{m}{r})' p^b$ *mit* $\mathrm{ggT}((\frac{m}{r})', p) = 1$.

Beweis. [5, Corollary 15.8]. □

Korollar 5.4. *Insbesondere ist* $\mathbb{F}_{q^m}$ *über* $\mathbb{F}_q$ *einfach, falls*

(1) $m = r$ *oder* $m = r^2$ *für eine Primzahl* r.

(2) $m' \mid (q-1)$, *wobei* $m = m'p^b$ *mit* $\mathrm{ggT}(m', p) = 1$.

(3) $m = p^b$ *für* $b \geq 0$.

Beweis. (1) ist klar. Für (2) sei auf [5, Theorem 15.9] verwiesen und (3) ist eine Folgerung aus (2). □

Im Abschnitt über normale Elemente konnten wir herausarbeiten, dass die Zerlegung von $x^n - 1$ in Kreisteilungspolynome eine guter Startpunkt ist, um normale Elemente zu konstruieren und die Modulstrukturen zu beschreiben. Jedoch zeigt es sich, dass im Allgemeinen ein Element, dessen q-Ordnung einem Kreisteilungspolynom entspricht, eine q^d-Ordnung für einen Teiler d von n besitzt, die kein reines Kreisteilungspolynom mehr ist. Also muss eine passende Klasse von Polynomen gefunden werden, um die verschiedenen simultan auftauchenden q^d-Ordnungen zu erfassen: *verallgemeinerte Kreisteilungspolynome.*

Definition 5.5 (verallgemeinertes Kreisteilungspolynom).

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

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

verallgemeinertes Kreisteilungspolynom.

Definition 5.6 (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 $\frac{k\,t}{\nu(k)}$.

Definition 5.7 (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 5.8 (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 5.9 (verträgliche Zerlegung).

Sei Δ eine Zerlegung von $\Phi_{k,t}$ in verallgemeinerte Kreisteilungspolynome über F. Dann heißt Δ *verträgliche Zerlegung* falls gilt: Für jedes $(l,s) \in i(\Delta)$ sei $w_{l,s} \in \bar{F}$ 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.

Nun können wir den einen zentralen Satz formulieren, der eine passende Zerlegung eines erweiterten Kreisteilungspolynoms herstellt, so dass sich ein vollständiger Erzeuger als Summe von vollständigen Erzeugern der entsprechenden Teilmoduln zusammensetzen lässt. Man bemerke an dieser Stelle, dass das Problem der vollständigen Erzeuger (und damit der vollständigen Normalbasen) ungleich schwerer ist, als das der normalen Elemente, da sich dort Elemente mit teilerfremden q-Ordnungen *immer* zu einem Element summieren, dessen q-Ordnung gerade das Produkt der q-Ordnungen ist (vgl. Satz 3.42); mit anderen Worten also die Summe von Erzeugern disjunkter Teilmoduln stets wieder einen Erzeuger liefert. Dies ist bei vollständigen Erzeugern nur bedingt gegeben, wie nachstehender Zerlegungssatz beschreibt.

Satz 5.10 (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 $a = \max\{b \in \mathbb{N} : r^b \mid t\}$.

Beweis. [5, Decomposition Theorem, Section 19]. □

Sicherlich kann man sich nun fragen, in welchen Fällen die kanonische Zerlegung von eines erweiterten Kreisteilungspolynoms in Kreisteilungspolynome noch verträglich ist. Nach [5, Theorem 19.10] ist die kanonische Zerlegung von $\Phi_{k,t}(x)^\pi$ verträglich über $\mathbb{F}_q$, falls $\operatorname{ord}_{\nu(kt')}(q)$ und t' teilerfremd sind. Dies motiviert dieser Klasse von Kreisteilungsmoduln einen eigenen Namen zu geben:

Definition 5.11 (regulär).

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

Eine Körpererweiterung $\mathbb{F}_{q^m} \mid \mathbb{F}_q$ heißt *regulär*, falls $\mathcal{C}_{1,m}$ regulär ist.

Definition 5.12 (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
- $\operatorname{ord}_{2^c}(q) = 2$.

Hachenberger war es nun möglich, für reguläre Kreisteilungsmoduln zu beweisen, dass alle auftretenden Zwischenkörper, deren Betrachtung bei der Suche nach vollständigen Erzeugern notwendig ist, von einem einzigen Zwischenkörper (oder zwei Zwischenkörpern) dominiert werden. Das bedeutet, dass ein Element eines regulären Kreisteilungsmoduls maximal zwei bestimmte $q^\bullet$-Ordnungen besitzen muss, um bereits den Kreisteilungsmodul vollständig zu erzeugen. Die geforderten $q^\bullet$-Ordnungen werden durch nachstehende Definition gegeben und wir schließen dieses Kapitel mit der Angabe dieses wahrlich beachtlichen Resultats.

Definition 5.13 (τ-Teiler).

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

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

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 5.14 (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

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

Beweis. [5, Theorem 20.3]. □

Kapitel 6

Existenz und Enumeration primitiv vollständig normaler Elemente

In den vorangegangenen Kapiteln haben wir lediglich normale und vollständig normale Elemente in Erweiterungen endlicher Körper betrachtet. Es existiert jedoch eine weitere besondere Eigenschaft, die gerade in der Anwendung von großem Interesse ist: Primitivität (Definition 1.6). Zusammen haben wir nun drei Eigenschaften eines Elements $u \in E$ einer Erweiterung endlicher Körper $E \mid F$ kennengelernt, die von Interesse sind. Daher ist es nur sinnvoll sich der Frage zu widmen, *wie viele* Elemente mit den jeweiligen Eigenschaften es gibt. Mit Satz 1.1 und Satz 1.5 ist sofort klar, dass es in $\mathbb{F}_q$ genau $\varphi(q-1)$ primitive Elemente gibt! Daher ist die Fragestellung nach der Anzahl primitiver Elemente schnell gelöst. Darüber hinaus wollen wir die folgenden Notationen treffen.

Definition 6.1.

Seien $\mathbb{F}_q$ ein endlicher Körper und $n \in \mathbb{N}^*$, so bezeichne $\mathcal{N}(q,n)$, $\mathcal{CN}(q,n)$, $\mathcal{PN}(q,n)$ bzw. $\mathcal{PCN}(q,n)$ die Anzahl der normalen, vollständig normalen, primitiv normalen bzw. primitiv vollständig normalen Elemente in $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$, d.h.

$$\begin{aligned}
\mathcal{N}(q,n) &:= |\{u \in \mathbb{F}_{q^n} : u \text{ ist normal über } \mathbb{F}_q\}| \\
\mathcal{CN}(q,n) &:= |\{u \in \mathbb{F}_{q^n} : u \text{ ist vollständig normal über } \mathbb{F}_q\}| \\
\mathcal{PN}(q,n) &:= |\{u \in \mathbb{F}_{q^n} : u \text{ ist primitiv und normal über } \mathbb{F}_q\}| \\
\mathcal{PCN}(q,n) &:= |\{u \in \mathbb{F}_{q^n} : u \text{ ist primitiv und vollständig normal über } \mathbb{F}_q\}| \\
\mathcal{G} &:= \{n \in \mathbb{N}^*, n \geq 2 : \forall q \text{ Primzahlpotenz gilt } \mathcal{PCN}(q,n) > 0\}
\end{aligned}$$

Vielleicht erscheint die Definition von $\mathcal{G}$ etwas überraschend, da für jedes Element in $\mathcal{G}$ schließlich *unendlich viele* Körpererweiterungen auf die Existenz eines $\mathcal{PCN}$-Elements getestet werden müssen. Doch es sei an dieser Stelle vorweg genommen, dass wir in der Lage sind mit Hilfe eines asymptoptischen Resultats und der konkreten Angabe von endlich vielen $\mathcal{PCN}$-Elementen zu zeigen, dass $\mathcal{G}$ nicht leer ist!

Nun können wir folgende Probleme definieren:

Problem 6.2 ($\mathcal{N}(q,n)$ =?).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Was ist $\mathcal{N}(q,n)$?

Problem 6.3 ($\mathcal{CN}(q,n) = ?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Was ist $\mathcal{CN}(q,n)$?

Problem 6.4 ($\mathcal{PN}(q,n) = ?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Was ist $\mathcal{PN}(q,n)$?

Problem 6.5 ($\mathcal{PCN}(q,n) = ?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Was ist $\mathcal{PCN}(q,n)$?

Offensichtlich können wir die obigen Problemstellungen leicht abschwächen und uns zunächst fragen, ob überhaupt Elemente mit den geforderten Eigenschaften existieren. Auch dazu wollen wir passende Probleme formulieren.

Problem 6.6 ($\mathcal{N}(q,n) > 0?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Ist $\mathcal{N}(q,n) > 0$?

Problem 6.7 ($\mathcal{CN}(q,n) > 0?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Ist $\mathcal{CN}(q,n) > 0$?

Problem 6.8 ($\mathcal{PN}(q,n) > 0?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Ist $\mathcal{PN}(q,n) > 0$?

Problem 6.9 ($\mathcal{PCN}(q,n) > 0?$).

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$. Ist $\mathcal{PCN}(q,n) > 0$?

Zuletzt wollen wir natürlich auch für $\mathcal{G}$ eine Problemstellung zu formulieren:

Problem 6.10 ($n \in \mathcal{G}?$).

Finde möglichst viele $n \in \mathbb{N}^*$, $n \geq 2$ mit $n \in \mathcal{G}$.

Bisher haben wir all diese Probleme nicht ausreichend geklärt. Doch im folgenden Abschnitt wollen wir uns jenen Fragestellungen zunächst theoretisch widmen, um in den darauffolgenden gezielte Enumerationen auf Basis der theoretischen Resultate, die über (vollständig) normale Elemente im bisherigen Verlauf erarbeitet wurden, durchzuführen, um für die offen bleibenden Fragen

6.1 Theoretische Enumerationen und Existenzaussagen

Wir starten mit einem wohlbekannten Resultat, das eine Antwort auf die Frage nach der Existenz von normalen Elementen (Problem 6.6) gibt:

Satz 6.11 (Satz von der Normalbasis).

Zu jedem endlichen Körper F und jeder endlichen Erweiterung E von F existiert eine Normalbasis von E über F.

Beweis. [15, Theorem 2.35]. □

Selbige Aussage können wir auch für vollständig normale Elemente treffen, was zuerst 1986 von Blessenohl und Johnsen [2] bewiesen wurde.

Satz 6.12 (Verschärfung des Satzes von der Normalbasis).

Zu jedem endlichen Körper F und jeder endlichen Erweiterung E von F existiert eine vollständige Normalbasis von E über F.

Beweis. [2, Satz 1.2]. □

Damit wäre auch Problem 6.7 beantwortet! Von den Existenzfragen bleibt damit noch die Existenz von primitiv normalen und primitiv vollständig normalen Elementen in beliebigen Erweiterungen offen. Erstere beantwortete Lenstra, Jr. und Schoof 1987 [13] nach den Vorarbeiten von Carlitz und Davenport.

Satz 6.13 (Satz von der primitiven Normalbasis).

Zu jedem endlichen Körper F und jeder endlichen Erweiterung E von F existiert eine primitive Normalbasis von E über F.

Beweis. [13]. □

Bleibt also nur noch die Frage nach der Existenz primitiver vollständig normaler Elemente. Auch wenn Hachenberger 2001 [9] und 2015 [6] die beiden nachstehenden bedeutsamen Resultate beweisen konnte, bleibt die Suche nach $\mathcal{PCN}$-Elementen weiterhin ein offenes Problem, dem wir uns im weiteren Verlauf experimentell widmen wollen.

Satz 6.14.

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$, so dass $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ eine reguläre Erweiterung ist. Sei ferner $4 \mid (q-1)$, falls q ungerade und n gerade ist. Dann existiert ein primitives Element in $\mathbb{F}_{q^n}$, das vollständig normal über $\mathbb{F}_q$ ist.

Beweis. [9, Theorem 1.4]. □

Satz 6.15.

Sei $n \in \mathbb{N}^*$ mit $n \geq 2$. Dann gilt: Für Primzahlpotenzen q mit $q \geq n^4$ existiert ein primitives Element in $\mathbb{F}_{q^n}$, das vollständig normal über $\mathbb{F}_q$ ist.

Beweis. [6, Theorem 2]. □

Wir können nun zusammenfassen, dass von obigen Existenzproblemen lediglich Problem 6.9 überlebt hat und alle anderen durch theoretische Resultate abgedeckt werden konnten. Nun können wir versuchen die Zählprobleme anzugehen und starten mit einem allgemein bekannten Resultat.

Definition 6.16.

Sei $f(x) \in \mathbb{F}_q[x]$ ein Polynom über einem endlichen Körper. Definiere

$$\phi_q(f) \ := \ |\{g(x) \in \mathbb{F}_q[x] : \ \deg g < \deg f, \ \mathrm{ggT}(f,g) = 1\}| .$$

Bemerkung 6.17. $\phi_q(f)$ ist das Analogon zur Eulerschen Phifunktion für Polynome, da $\phi_q(f)$ gerade die Anzahl der Einheiten im Ring $\mathbb{F}_q[x]/(f(x))$ angibt.

Satz 6.18.

Seien q eine Primzahlpotenz und $n \in \mathbb{N}^*$ mit $n \geq 2$, so existieren in $\mathbb{F}_{q^n}$ genau

$$\phi_q(x^n - 1) \ = \ q^{n'(\pi-1)} \prod_{d \mid n} \left(q^{\mathrm{ord}_d(q)} - 1\right)^{\frac{\varphi(d)}{\mathrm{ord}_d(q)}}$$

Elemente, die normal über $\mathbb{F}_q$ sind, wobei $n = n'\pi$ mit $\mathrm{ggT}(n', q) = 1$.

Beweis. [15, Theorem 3.73] oder [5, Theorem 10.5]. □

Obiger Satz beantwortet also Problem 6.2 vollständig. Für die Anzahlen von primitiv normalen und primitiv vollständig normalen Elementen einer gegebenen Körpererweiterung war es jedoch bisher nicht möglich Aussagen zu formulieren, die alle Paare (q, n) erfassen.

Ein kleiner Schritt in Richtung der Bestimmung der Anzahl von vollständig normalen Elementen besteht sicherlich in der Erkenntnis, dass für einfache Erweiterungen (Definition 5.2) diese Frage von Satz 6.18 beantwortet wird. Ferner war Hachenberger in [5, Section 21] in der Lage diese Frage für reguläre Kreisteilungsmoduln (und damit für reguläre Erweiterungen) zu beantworten:

Satz 6.19.

Seien $\mathbb{F}_q$ ein endlicher Körper von Charakteristik p und $m \in \mathbb{N}^*$ mit $\mathrm{ggT}(m,q) = 1$. Ist dann $\mathcal{C}_{m,p^b}$ ein regulärer Kreisteilungsmodul über $\mathbb{F}_q$, so ist die Anzahl der vollständigen Erzeuger von $\mathcal{C}_{m,p^b}$ gleich

(1) $\left(q^{\frac{\mathrm{ord}_m(q)}{\tau}} - 1\right)^{\frac{\tau\,\varphi(m)}{\mathrm{ord}_m(q)}} \cdot q^{(p^b-1)\varphi(m)}$, *falls $\mathcal{C}_{m,p^b}$ nicht ausfallend ist und*

(2) $\left(q^{\frac{2\,\mathrm{ord}_m(q)}{\tau}} - 4q^{\frac{\mathrm{ord}_m(q)}{\tau}} + 3\right)^{\frac{\tau\,\varphi(m)}{2\,\mathrm{ord}_m(q)}} \cdot q^{(p^b-1)\varphi(m)}$, *falls $\mathcal{C}_{m,p^b}$ ausfallend ist,*

wobei $\tau = \tau(q,n)$ aus Definition 5.13.

Beweis. [5, Proposition 21.1, Proposition 21.2]. □

Ferner lässt sich daraus eine Abschätzung für die Anzahl von vollständig normalen Elementen einer regulären Körpererweiterung ableiten.

Satz 6.20.

Sei $\mathbb{F}_{q^m}$ über $\mathbb{F}_q$ eine reguläre Erweiterung endlicher Körper, so ist die Anzahl der vollständig normalen Elemente in $\mathbb{F}_{q^m} \mid \mathbb{F}_q$ mindestens

$$(q-1)^{m'} \cdot q^{m'(p^b-1)},$$

wobei $m = m' p^b$ mit $\mathrm{ggT}(p, m') = 1$ und p der Charakteristik von $\mathbb{F}_q$. Gleichheit gilt genau dann, wenn $m' \mid (q-1)$ und die Erweiterung damit einfach ist.

Beweis. [5, Theorem 21.3, Theorem 6.1]. □

Bei primitiv normalen und primitiv vollständig normalen Elementen ist das Wissen über deren Anzahlen leider noch spärlicher gesät. Für $\mathcal{PCN}$-Elemente existieren die folgenden Abschätzungen.

Satz 6.21.

Sei p eine Primzahl und q eine Potenz von p. Dann gilt:

(1) $\mathcal{PCN}(q,2^l) \geq 4(q-1)^{2^{l-2}}$, *falls* $q \equiv 3 \bmod 4$ *und* $l \geq e+3$ *für* e *maximal, so dass* $2^e \mid (q^2-1)$, *oder* $q \equiv 1 \bmod 4$ *und* $l \geq 5$.

(2) $\mathcal{PCN}(q,r^l) \geq r^2(q-1)^{r^{l-2}}$, *falls* $r \neq p$ *eine ungerade Primzahl und* $l \geq 2$.

(3) $\mathcal{PCN}(q,r^l) \geq r(q-1)^{r^{l-1}}\varphi(q^{r^{l-1}}-1)$, *falls* $r \geq 7, \geq p$ *eine Primzahl und* $l \geq 2$.

(4) $\mathcal{PCN}(q,p^l) \geq p\,q^{p^{l-1}-1}(q-1)$, *falls* $l \geq 2$.

(5) $\mathcal{PCN}(q,p^l) \geq p\,q^{p^{l-1}-1}(q-1)\,\varphi(q^{p^{l-1}}-1)$, *falls* $p \geq 7$ *und* $l \geq 2$.

Beweis. [8]. □

Wie man nun deutlich erkennen kann, besteht immer noch große Unklarheit über die Anzahl von primitiv (vollständig) normalen Elementen und selbst die Zahl vollständig normaler Elemente ist nicht hinreichend geklärt. Daher bietet es sich freilich an, eine computergestützte Enumeration durchzuführen, um den Nebel ein wenig mehr lichten zu können. Morgan und Mullen [16] haben bereits 1996 für (q,n) aus $\{(2,2),\dots,(2,18),\ (3,2),\dots,(3,12),\ (4,2),\dots,(4,9),\ (5,2),\dots,(5,8),\ (7,2),\dots,(7,6),\ (8,2),\dots,(8,5),\ (9,2),\dots,(9,5)\}$ die Werte $\mathcal{CN}(q,n)$ und $\mathcal{PCN}(q,n)$ bestimmen können. Betrachten wir einmal diese Zahlenkonstellationen und fragen uns, ob wir diese nicht durch obige theoretische Resultate abdecken kann, so müssen wir feststellen, dass die Anzahlen der vollständig normalen Elemente lediglich die Paare $(2,6),(2,10),(2,12),(2,18),(3,8),(3,10),(5,6)$ nicht einfach und nicht regulär sind, also $\mathcal{CN}(q,n)$ nicht bereits aus obigen Sätzen folgt.

Daher setzen wir uns im Folgenden das Ziel, die Tabelle von Morgan und Mullen einerseits zu verifizieren und andererseits zu erweitern, um ein weitaus breiteres Spektrum an Zahlwerten präsentieren zu können. Dabei sei angemerkt, dass in [16] die theoretischen Überlegungen aus Kapitel 5 gänzlich unbeachtet blieben. Wir wollen diese jedoch in nachfolgender Implementierung intensiv nutzen, um bestmögliche Ausbeute vorhanderer Rechenleistung zu erhalten.

6.2 Implementierung endlicher Körper und Körpererweiterungen

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

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

eigene `C`-Bibliotheken anzulegen, die auf einfacher (jedoch begrenzter) `Integer`-Arithmetik basieren.

6.2.1 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.

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`

```c
/**
 * Finite Field Element.
 *
 * !! idcs must be in desc order !!
 *
 * Uses int arrays, i.e. you must not consider
 * PrimeFields of order p with (p-1)*(p-1) > INT_MAX
 */
struct FFElem{
    int *el;
    int *idcs;
    int len;
};
```

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

Invariante 6.22.

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

Invariante 6.23.

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

Satz 6.22 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 6.24. 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`

```
inline struct FFElem *mallocFFElem(int m){
    struct FFElem *ff = malloc(sizeof(struct FFElem));
    ff->el = malloc(m*sizeof(int));
    ff->idcs = malloc(m*sizeof(int));
    ff->len = 0;
    return ff;
}
```

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

```
inline void freeFFElem(struct FFElem *ff){
    free(ff->el);
    free(ff->idcs);
    free(ff);
}
```

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`

```
/**
 * Copies the content of ff1 into ff2
 *
 * !! ff2 must be malloced!
 */
inline void copyFFElem(struct FFElem *ff1, struct FFElem *ff2){
    if(ff1 == ff2) return;
    int i;
    for(i=0;i < ff1->len;i++){
        ff2->idcs[i] = ff1->idcs[i];
        ff2->el[ ff1->idcs[i] ] = ff1->el[ ff1->idcs[i] ];
    }
    ff2->len = ff1->len;
}
```

6.2.2 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 6.25. 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 6.26. 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

```c
/**
 * Adds two FFElems.
 *
 * !! ff1 may be same as ret !!
 * !! ff2 must not be same as ret !!
 */
inline void addFFElem(struct FFElem *ff1, struct FFElem *ff2,
        struct FFElem *ret,
        int *tmp,
        int *multTable, int *addTable){
    int i=0,j=0,k=0, i2;
    bool end = false;
    //handle trivial cases
    if(ff1->len == 0){
        copyFFElem(ff2,ret);
        return;
    }
    if(ff2->len == 0){
        copyFFElem(ff1,ret);
        return;
    }
    copyArray(ff1->idcs,tmp,ff1->len);
    while( end == false ){
        while( tmp[i] != ff2->idcs[j] ){
            if( tmp[i] > ff2->idcs[j] ){
                ret->el[ tmp[i] ] = ff1->el[ tmp[i] ];
                ret->idcs[k] = tmp[i];
                i++; k++;
            }else if( tmp[i] < ff2->idcs[j] ){
                ret->el[ ff2->idcs[j] ] = ff2->el[ ff2->idcs[j] ];
                ret->idcs[k] = ff2->idcs[j];
                j++; k++;
            }
            if(i == ff1->len || j == ff2->len){
                end = true;
                break;
            }

        }
        if(end == true) break;
        //tmp[i] == ff2->idcs[j]
        i2 = tmp[i];
        ret->el[i2] = addTable[ ff1->el[i2] + ff2->el[i2] ];
        if(ret->el[i2] != 0){
            ret->idcs[k] = i2;
            k++;
```

```
        }
        i++; j++;
        if(i == ff1->len || j == ff2->len) end = true;
    }
    //add rest of ff1 or ff2
    if(i != ff1->len ){
        while(i<ff1->len){
            ret->el[ tmp[i] ] = ff1->el[ tmp[i] ];
            ret->idcs[k] = tmp[i];
            i++; k++;
        }
    }else if(j != ff2->len){
        while(j<ff2->len){
            ret->el[ ff2->idcs[j] ] = ff2->el[ ff2->idcs[j] ];
            ret->idcs[k] = ff2->idcs[j];
            j++; k++;
        }
    }
    ret->len = k;
}
```

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-`FFElems`. 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`

```
/**
 * Multiplies two FFElems and reduces the result by mipo.
 *
 * !! tmp must have at least length m!
 * !! ret must be malloced!
 */
inline void multiplyFFElem(struct FFElem *ff1, struct FFElem *ff2,
        struct FFElem *ret,
        struct FFElem *mipo, int *tmp, int m,
        int *multTable, int *addTable){
    /*
     * catch trivial cases
     */
    if(ff1->len == 0 || ff2->len == 0){
        ret->len = 0;
        return;
    }
```

```
    if(ff1->len == 1 && ff1->idcs[0] == 0 && ff1->el[0] == 1){
        copyFFElem(ff2, ret);
        return;
    }
    if(ff2->len == 1 && ff2->idcs[0] == 0 && ff2->el[0] == 1){
        copyFFElem(ff1, ret);
        return;
    }

    /*
     * Do multiplication
     */
    int maxlen = ff1->idcs[0] + ff2->idcs[0] + 1;
    int i,j,i2,j2,k;
    int max2 = maxlen;
    if( maxlen > m ){
        max2 = m;
        initPoly(tmp,maxlen-m);
    }
    initPoly(ret->el,max2);
    //multiply
    for(i=0;i<(ff1->len);i++){
        for(j=0;j<(ff2->len);j++){
            i2 = ff1->idcs[i];
            j2 = ff2->idcs[j];
            k = i2+j2;
            if(k<m){
                ret->el[k] = addTable[ ret->el[k] +
                    multTable[ ff1->el[i2] * ff2->el[j2] ] ];
            }else{
                tmp[k-m] = addTable[ tmp[k-m] +
                    multTable[ ff1->el[i2] * ff2->el[j2] ] ];
            }
        }
    }

    /*
     * Reduce mod mipo
     */
    if(maxlen > m){
        int quo;
        for(i=maxlen-m-1;i>=0;i--){
            quo = tmp[i];
            if(quo == 0) continue;
            for(j=0;j<(mipo->len); j++){
                j2 = mipo->idcs[j];
                k = i+j2;
                if(k>=m){
                    tmp[k-m] = addTable[ tmp[k-m] -
                        multTable[ mipo->el[j2]*quo ] ];
                }else{
                    ret->el[k] = addTable[ ret->el[k] -
                        multTable[ mipo->el[j2]*quo ] ];
                }
```

```
                }
            }
        }

        /*
         * Recalc indices
         */
        i2 = 0;
        for(i=max2-1;i>=0;i--){
            if(ret->el[i] != 0){
                ret->idcs[i2] = i;
                i2++;
            }
        }
        ret->len = i2;
}
```

Außer den beiden zu multiplizierenden `FFElems` 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 FFElems` auf Primitivität und dem damit verbundenen Potenzieren, existiert eine separate Funktion zur Quadrierung eines `FFElems`. 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`

```
/**
 * Squares an FFElem
 *
 * !! ff is not modified !!
 * !! tmp must have at least length m !!
 */
inline void squareFFElem(struct FFElem *ff, struct FFElem *mipo,
        struct FFElem *ret, int *tmp, int m,
        int *multTable, int *addTable){
    /*
     * catch trivial cases
     */
    if(ff->len == 0){
        copyFFElem(ff,ret);
        return;
    }
    if(ff->len == 1 && ff->idcs[0] == 0 && ff->el[0] == 1){
        copyFFElem(ff,ret);
        return;
```

```
    }

    /*
     * Do multiplication
     */
    int maxlen = 2*ff->idcs[0] + 1;
    int i,j,i2,j2,k;
    int max2 = maxlen;
    if( maxlen > m ){
        max2 = m;
        initPoly(tmp,maxlen-m);
    }
    initPoly(ret->el,max2);
    for(i=0;i<(ff->len);i++){
        // same index must be squared
        i2 = ff->idcs[i];
        k = 2*i2;
        if(k<m){
            ret->el[k] = addTable[ ret->el[k] +
                multTable[ ff->el[i2]*ff->el[i2] ] ];
        }else{
            tmp[k-m] = addTable[ tmp[k-m] +
                multTable[ ff->el[i2]*ff->el[i2] ] ];
        }
        // other indices only multipied and doubled
        for(j=i+1;j<(ff->len);j++){
            i2 = ff->idcs[i];
            j2 = ff->idcs[j];
            k = i2+j2;
            if(k<m){
                ret->el[k] = addTable[ ret->el[k] +
                    multTable[ 2 * multTable[ ff->el[i2] * ff->el[j2] ] ] ];
            }else{
                tmp[k-m] = addTable[ tmp[k-m] +
                    multTable[ 2 * multTable[ ff->el[i2] * ff->el[j2] ] ]];
            }
        }
    }
    /*
     * Reduce mod mipo
     */
    if(maxlen > m){
        int quo;
        for(i=maxlen-m-1;i>=0;i--){
            quo = tmp[i];
            if(quo == 0) continue;
            for(j=0;j<(mipo->len); j++){
                j2 = mipo->idcs[j];
                k = i+j2;
                if(k>=m){
                    tmp[k-m] = addTable[ tmp[k-m] -
                        multTable[ mipo->el[j2]*quo ] ];
                }else{
                    ret->el[k] = addTable[ ret->el[k] -
```

```
                        multTable[ mipo->el[j2]*quo ] ];
                    }
                }
            }
        }

        /*
         * Recalc indices
         */
        i2 = 0;
        for(i=max2-1;i>=0;i--){
            if(ret->el[i] != 0){
                ret->idcs[i2] = i;
                i2++;
            }
        }
        ret->len = i2;
    }
```

6.2.3 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`

```
/**
 * Matrix multiplication
 *
 * !! tmp must have at least length m!
 */
inline void matmul(struct FFElem **mat, struct FFElem *ff,
        struct FFElem *ret,
        int m, int *multTable, int *addTable){
    int i,j,i2, row;
    bool end;
    for(row=0;row<m;row++){
        ret->el[row] = 0;
        i=0; j=0;
        end = false;
        while(end == false){
            while(ff->idcs[i] != mat[row]->idcs[j]){
                if(ff->idcs[i] > mat[row]->idcs[j]) i++;
                else if(ff->idcs[i] < mat[row]->idcs[j]) j++;
                if(i == ff->len || j == mat[row]->len){
                    end = true;
                    break;
                }
            }
```

```
            if(end == true) break;
            i2 = ff->idcs[i]; // == mat[row]->idcs[j]
            ret->el[row] = addTable[ ret->el[row]
                + multTable[ mat[row]->el[i2]*ff->el[i2] ] ];
            i++;
            j++;
            if(i==ff->len || j==mat[row]->len) end = true;
        }
    }
    i2 = 0;
    for(i=m-1;i>=0;i--){
        if(ret->el[i] != 0){
            ret->idcs[i2] = i;
            i2++;
        }
    }
    ret->len = i2;
}
```

Hier wird – anders als bei der Addition – nur nach den gemeinsamen Indizes gesucht (alle anderen Produkte sind schließlich 0). Invariante 6.22 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`

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

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`

```
struct FFPoly{
    struct FFElem **poly;
    int lenPoly;
};
```

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

```
inline struct FFPoly *mallocFFPoly(int m, int lenPoly){
    struct FFPoly *poly = malloc(lenPoly*sizeof(struct FFElem*));
    poly->lenPoly = lenPoly;
    int i;
```

```
    for(i=0;i<lenPoly;i++) poly->poly[i] = mallocFFElem(m);
    return poly;
}
```

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

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

6.3 Potenzieren und Primitivitätstest

6.3.1 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 6.27. 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`

```
/**
 * Square and multiply in charac
 * mat is powering by charac
 *
 * !! ff is modified !!
 */
inline void powerFFElem(struct FFElem *ff, struct FFElem *mipo,
        struct FFElem *ret,
        int m, int *power, int powerLen,
        struct FFElem **matCharac, int *tmp, struct FFElem *ffTmp,
        int *multTable, int *addTable){
    int i,j,k;
    int lenCurGap = 0;
    struct FFElem *ffSwitch = 0;
    struct FFElem *ffRetInt = ret;
    // init ret to 1
    ffRetInt->el[0] = 1; ffRetInt->idcs[0] = 0; ffRetInt->len = 1;
    for(j=powerLen-1;j>=0;j--){
        for(k=0;k<power[j];k++){
            multiplyFFElem(ffRetInt,ff,ffTmp, mipo,tmp,m,multTable,addTable);
            ffSwitch = ffRetInt; ffRetInt = ffTmp; ffTmp = ffSwitch;
        }
        if(j==0 || power[j-1] == 0){
            lenCurGap++;
            continue;
        }
        matmul(matCharac+lenCurGap*m, ff, ffTmp, m, multTable,addTable);
        ffSwitch = ff; ff = ffTmp; ffTmp = ffSwitch;
        lenCurGap = 0;
    }
    copyFFElem(ffRetInt,ret);
}
```

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

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

```
/**
 * Square and multiply
 *
 * !! ff is modified !!
 */
inline void powerFFElemSqM(struct FFElem *ff, struct FFElem *mipo,
```

```
        struct FFElem *ret,
        int m, int *power, int powerLen,
        int *tmp, struct FFElem *ffTmp,
        int *multTable, int *addTable){
    int i,j,k;
    int lenCurGap = 0;
    struct FFElem *ffSwitch = 0;
    struct FFElem *ffRetInt = ret;
    // init ret to 1
    ffRetInt->el[0] = 1; ffRetInt->idcs[0] = 0; ffRetInt->len = 1;
    for(j=powerLen-1;j>=0;j--){
        if(power[j] == 1){
            multiplyFFElem(ffRetInt,ff,ffTmp, mipo,tmp,m,multTable,addTable);
            //switch ffTmp and ffRetInt
            ffSwitch = ffRetInt; ffRetInt = ffTmp; ffTmp = ffSwitch;
        }
        if(j>0){
            squareFFElem(ff,mipo,ffTmp,tmp,m,multTable,addTable);
            //switch ffTmp and ff
            ffSwitch = ff; ff = ffTmp; ffTmp = ffSwitch;
        }
    }
    copyFFElem(ffRetInt,ret);
}
```

6.3.2 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 6.28. *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 6.29. 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:

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

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`

```
/**
 * Test if element is primitive.
 *
 * !! if matCharac is Zero, all powers are assumed as binary arrays !!
 *
 * !! fff,ffTmp,ffTmp2,ffTmp3,ffRet must be malloced !!
 * !! x is NOT modified !!
 */
inline bool isPrimitive(struct FFElem *ff, struct FFElem *mipo,
```

```
        int m,
        int *barFactors, int *lenBarFactors, int countBarFactors,
        int *commonBarFactor, int lenCommonBarFactor,
        int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
        struct FFElem **matCharac,
        struct FFElem *fff, struct FFElem *ffff, struct FFElem *ffTmp,
        struct FFElem *ffTmp2, struct FFElem *ffRet,
        int *tmp, int *multTable, int *addTable){
    int i;
    int curPos = 0;
    bool binarySqM = (matCharac == 0);
    struct FFElem *ffSwitch = 0;

    copyFFElem(ff,fff);
    // all barFactors are power of commonBarFactor
    if(binarySqM)
        powerFFElemSqM(fff,mipo,ffTmp,
                m,commonBarFactor,lenCommonBarFactor,
                tmp,ffTmp2,
                multTable,addTable);
    else
        powerFFElem(fff,mipo,ffTmp,
                m,commonBarFactor,lenCommonBarFactor,
                matCharac,tmp,ffTmp2,
                multTable,addTable);
    if(isOne(ffTmp)) return false;
    //switch ffTmp and fff
    ffSwitch = fff; fff = ffTmp; ffTmp = ffSwitch;
    copyFFElem(fff,ffff);
    //test first barFactor
    if(binarySqM)
        powerFFElemSqM(ffff,mipo,ffTmp,
                m,barFactors,lenBarFactors[0],
                tmp,ffTmp2,
                multTable,addTable);
    else
        powerFFElem(ffff,mipo,ffTmp,
                m,barFactors,lenBarFactors[0],
                matCharac,tmp,ffTmp2,
                multTable,addTable);
    if(isOne(ffTmp)) return false;
    curPos += lenBarFactors[0];
    //test further factors which are powers of commonBiggestBarFactor
    //so first, calc y^commonBiggestBarFactor
    copyFFElem(fff, ffff);
    if(binarySqM)
        powerFFElemSqM(ffff,mipo,ffTmp,
                m,commonBiggestBarFactor,lenCommonBiggestBarFactor,
                tmp,ffTmp2,
                multTable,addTable);
    else
        powerFFElem(ffff,mipo,ffTmp,
                m,commonBiggestBarFactor,lenCommonBiggestBarFactor,
                matCharac,tmp,ffTmp2,
```

```
                multTable,addTable);
    if(isOne(ffTmp)) return false;
    ffSwitch = fff; fff = ffTmp; ffTmp = ffSwitch;
    for(i=1;i<countBarFactors;i++){
        // copy z (fff) to ffff
        copyFFElem(fff,ffff);
        // *** ffff == fff == y^commonBiggestBarFactor
        if(binarySqM)
            powerFFElemSqM(ffff, mipo, ffTmp,
                    m,barFactors+curPos, lenBarFactors[i],
                    tmp,ffTmp2,
                    multTable,addTable);
        else
            powerFFElem(ffff, mipo, ffTmp,
                    m,barFactors+curPos, lenBarFactors[i],
                    matCharac,tmp,ffTmp2,
                    multTable,addTable);

        if(i>1){
            multiplyFFElem(ffRet,ffTmp,ffTmp2,mipo,
                    tmp,m,multTable,addTable);
        }else{
            ffSwitch = ffTmp2; ffTmp2 = ffTmp; ffTmp = ffSwitch;
        }
        if(isOne(ffTmp2)) return false;
        curPos += lenBarFactors[i];
        ffSwitch = ffRet; ffRet = ffTmp2; ffTmp2 = ffSwitch;
    }
    return true;
}
```

6.4 Frobenius-Auswertung und Test auf vollständige Erzeuger-Eigenschaft

6.4.1 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 5.6) und $u \in E$ ein Element, das wir als vollständigen Erzeuger in Betracht ziehen (vgl. Definition 5.7). Nach **??** (3) ist u genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$, wenn

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

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 6.2.1) 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 6.30. *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,
```

```
        int *multTable, int *addTable){
    int i,j;

    ret->len = 0;
    for(i=0;i<poly->lenPoly;i++){
        if(poly->poly[i]->len == 0) continue;
        j = i*frobPower-1;
        if(i>0 && matmulCacheCalced[j] == true){
            multiplyFFElem(matmulCache[j],poly->poly[i],
                    ffTmp, mipo,
                    tmp,m,multTable,addTable);
            addFFElem(ret,ffTmp,ret,tmp,multTable,addTable);
        }else{
            if(i>0){
                matmul(mats+j*m, ff, ffTmp, m, multTable,addTable);
                //update matmulCache
                copyFFElem(ffTmp, matmulCache[j]);
                matmulCacheCalced[j] = true;
            }else{
                copyFFElem(ff,ffTmp);
            }
            //go on and multiply ffTmp with current coefficient
            multiplyFFElem(ffTmp, poly->poly[i],
                    ffTmp2, mipo,
                    tmp,m,multTable,addTable);
            addFFElem(ret,ffTmp2,ret,tmp,multTable,addTable);
        }
    }
}
```

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`

```
/*
 * calculates g(sigma^frobPower)(x) where g is a polynomial
 * and sigma the frobenius
 * application of frobenius is given by mats
 */
inline void applyFrob_noCache(struct FFElem *ff, struct FFElem *mipo,
        struct FFPoly *poly,
        struct FFElem **mats,
        int frobPower, struct FFElem *ret,
        int m, int *tmp, struct FFElem *ffTmp, struct FFElem *ffTmp2,
        int *multTable, int *addTable){
    int i,j;
    ret->len = 0;

    for(i=0;i<poly->lenPoly;i++){
        if(poly->poly[i]->len == 0) continue;
        if(i>0){
            j = i*frobPower-1;
```

```
            matmul(mats+j*m, ff, ffTmp, m, multTable,addTable);
        }else{
            copyFFElem(ff,ffTmp);
        }
        multiplyFFElem(ffTmp, poly->poly[i],
                ffTmp2, mipo,
                tmp,m,multTable,addTable);
        addFFElem(ret,ffTmp2,ret,tmp,multTable,addTable);
    }
}
```

6.4.2 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

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

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

Lemma 6.31. *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

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

Beweis. Klar per definitionem der q-Ordnung und Lemma 6.30. □

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`

```
inline bool testSubmod(struct FFElem *ff, struct FFElem *mipo,
        struct FFPoly **polys,
        int polysCount, bool *evalToZero,
        struct FFElem **mats, int *frobPowers,
        int m, int *tmp,
        struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
```

```
        struct FFElem **matmulCache, bool *matmulCacheCalced,
        int *multTable, int *addTable){
    int i;
    int goodCounter = 0;
    for(i=0;i<polysCount;i++){
        applyFrob(ff,mipo,
                polys[i],
                mats,frobPowers[i], ffTmp,
                m,tmp,ffTmp2,ffTmp3,
                matmulCache,matmulCacheCalced,
                multTable,addTable);
        if( isZero(ffTmp) == evalToZero[i] ){
            goodCounter++;
        }else{
            return false;
        }
    }
    if(goodCounter == polysCount){
        return true;
    }
    return false;
}
```

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`

```
inline int testAllSubmods(struct FFElem *ff, struct FFElem *mipo,
        int decompCount, struct FFPoly **polys,
        int *polysCountPerDecomp, bool *evalToZero,
        struct FFElem **mats, int *frobPowers, bool *toTestIndicator,
        int m, int *tmp,
        struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
        struct FFElem **matmulCache, bool *matmulCacheCalced,
        int *multTable, int *addTable){
    if(ff->len == 0) return -1;
    int i,j,k;
    int goodCounter = 0;
    int curDecompPosition = 0;
    for(i=0;i<decompCount;i++){
        if(toTestIndicator != 0 && toTestIndicator[i] == false){
            curDecompPosition += polysCountPerDecomp[i];
            continue;
        }
```

```
        goodCounter = 0;
        for(j=0;j<polysCountPerDecomp[i];j++){
            applyFrob(ff,mipo,
                    polys[curDecompPosition+j],
                    mats,frobPowers[curDecompPosition+j], ffTmp,
                    m,tmp,ffTmp2,ffTmp3,
                    matmulCache,matmulCacheCalced,
                    multTable,addTable);
            if( isZero(ffTmp) == evalToZero[curDecompPosition+j] ){
                goodCounter++;
            }else break;
        }
        if(goodCounter == polysCountPerDecomp[i]){
            return i;
        }
        curDecompPosition += polysCountPerDecomp[i];
    }
    return -1;
}
```

6.5 Implementierung der gezielten Enumeration

6.5.1 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 5.10) 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 6.32. *Sei $u \in E$ ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$ über F. Dann gilt*

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

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 $\mathrm{ggT}(f, \Phi_{k,t})$ 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 mar
hier
ausführliche
argumentier

Bemerkung 6.33. Wiederum überzeuge man sich kurz mit der gleichen Argumentation wie in Lemma 6.30, 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`

```
inline void calcSubmoduleElements(struct Node *root,
    struct FFElem *mipo,
    int maxLenPoly,
    int *genCounts, int curGen,
    struct FFPoly **polys, int polysCount, bool *evalToZero,
    struct FFElem **mats, int matLen, int *frobPowers,
    struct FFElem **elementsF,
    int m, int q, int *tmp,
    struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
    struct FFElem *ffTmp4,
    struct FFElem **matmulCache, bool *matmulCacheCalced,
    int *multTable, int *addTable){
  int i,j;
  struct Node *curRoot = root;
  struct FFElem *ff = root->ff;
  int *curPoly = malloc( maxLenPoly*sizeof(int) );
  struct FFPoly *curFPoly = malloc( sizeof(struct FFPoly) );
  curFPoly->poly = malloc( maxLenPoly*sizeof(struct FFElem*) );
  curFPoly->lenPoly = 0;

  initPoly(curPoly,maxLenPoly);
  curPoly[0] = 2;
  int curLenPoly = 1;
  if( q == 2 && maxLenPoly > 1){
    curLenPoly = 2;
    curPoly[0] = 0;
    curPoly[1] = 1;
  }
  if(q != 2 || maxLenPoly > 1){
    while(true){
      //setup curFPoly
      for(i=0;i<curLenPoly;i++)
        curFPoly->poly[i] = elementsF[curPoly[i]];
      curFPoly->lenPoly = curLenPoly;
      //apply Frobenius
      applyFrob_noCache(ff,mipo,
          curFPoly,
          mats,1, ffTmp, //return value
          m,tmp,ffTmp2,ffTmp3,
          multTable,addTable);
      //test generated element
      for(i=0;i<matLen;i++) matmulCacheCalced[i] = false;
```

```
        if(testSubmod(ffTmp, mipo,
                polys,polysCount,evalToZero,
                mats,frobPowers,m,tmp,
                ffTmp2,ffTmp3,ffTmp4,
                matmulCache,matmulCacheCalced, multTable,addTable)){
            curRoot = appendToEnd(curRoot,ffTmp,m);
            genCounts[curGen]++;
        }
        //generate next element
        curPoly[0] += 1;
        if( curPoly[0] == q ){
            for(i=0;i<maxLenPoly-1 && curPoly[i]==q;i++){
                curPoly[i] = 0;
                curPoly[i+1] += 1;
            }
            if(i+1>curLenPoly)
                curLenPoly = i+1;
            if( curPoly[maxLenPoly-1]==q){
                break;
            }
        }
    }
  }
  free(curPoly);
  free(curFPoly->poly);
  free(curFPoly);
}
```

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`

```
struct Node {
    struct FFElem *ff;
    struct Node *next;
};
```

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`

```
/**
 * appends element to end of root, where element is copied to new FFElem.
 */
inline struct Node *appendToEnd(struct Node *root, struct FFElem *element,int m){
    struct Node *nextNode = root;
    if( nextNode != 0){
        while(nextNode->next != 0){
            nextNode = nextNode->next;
        }
        if( nextNode->ff != 0){
            nextNode->next = malloc( sizeof(struct Node) );
            nextNode = nextNode->next;
        }
        if( nextNode != 0){
            nextNode->next = 0;
            nextNode->ff = mallocFFElem(m);
            copyFFElem(element,nextNode->ff);
            return nextNode;
        }
    }
    return NULL;
}
```

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

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

```
inline void freeNode(struct Node* head){
    struct Node *next_n = NULL;
    struct Node *tmp_n = NULL;
    for(tmp_n=head; tmp_n !=NULL; ){
        next_n = tmp_n->next;
        freeFFElem(tmp_n->ff);
        free(tmp_n);
        tmp_n = next_n;
    }
    head = 0;
}
```

6.5.2 Dynamische Enumeration des größten Kreisteilungsmoduls

Da der Zerlegungssatz (Satz 5.10) 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));
```

```

    int i,j;
    int curGen = decompCount-1;
    struct Node **curRoots = malloc( decompCount*sizeof(struct Node*) );
    struct FFElem *ff = roots[curGen]->ff;
    int *curPoly = malloc( maxLenPoly*sizeof(int) );
    struct FFPoly *curFPoly = malloc( sizeof(struct FFPoly) );
    curFPoly->poly = malloc( maxLenPoly*sizeof(struct FFElem*) );
    curFPoly->lenPoly = 0;

    initPoly(curPoly,maxLenPoly);
    curPoly[0] = 1;
    int curLenPoly = 1;

    unsigned long long pcn = 0;
    while(true){
        //setup curFPoly
        for(i=0;i<curLenPoly;i++)
            curFPoly->poly[i] = elementsF[curPoly[i]];
        curFPoly->lenPoly = curLenPoly;
        //apply Frobenius
        applyFrob_noCache(ff,mipo,
                curFPoly,
                mats,1, fff, //return value
                m,tmp,ffTmp,ffTmp2,
                multTable,addTable);
        //test generated element
        for(i=0;i<matLen;i++) matmulCacheCalced[i] = false;
        if(testSubmod(fff, mipo,
                polys,polysCount,evalToZero,
                mats,frobPowers,m,tmp,
                ffTmp,ffTmp2,ffTmp3,
                matmulCache,matmulCacheCalced, multTable,addTable)){
            genCounts[curGen]++;
            // build element as sum of already calced Nodes and ffTmp
            for(i=0;i<decompCount;i++) curRoots[i] = roots[i];
            // cycle through Nodes, build element and test primitivity
            while(true){
                copyFFElem(fff,ffff);
                //build element
                for(i=0;i<decompCount-1;i++){
                    addFFElem(ffff, curRoots[i]->ff, ffff, tmp,
                            multTable,addTable);
                }
                //test primitivity
                if(countBarFactors > 0){
                    if(isPrimitive(ffff, mipo,m,
                                barFactors,lenBarFactors,countBarFactors,
                                commonBarFactor,lenCommonBarFactor,
                                commonBiggestBarFactor,lenCommonBiggestBarFactor,
                                matCharac,
                                ffTmp,ffTmp2,ffTmp3,ffTmp4,ffTmp5,
                                tmp,multTable,addTable)){
                        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 6.34. 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`

```c
unsigned long long processFiniteField(struct FFElem *mipo, int decompCount,
        struct FFPoly **polys, int *polysCountPerDecomp,
        bool *evalToZero,
        struct FFElem **mats, int matLen, int *frobPowers,
        int *genCounts, int m, int charac, int q,
        int *barFactors, int *lenBarFactors, int countBarFactors,
        int *commonBarFactor, int lenCommonBarFactor,
        int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
        struct FFElem **matCharac, struct FFElem **elementsF,
        int *multTable, int *addTable){
    time_t TIME = time(NULL);
    int i,j;

    //setup temporary variables -------------------------------------------
    int *tmp = malloc(m*sizeof(int));
    struct FFElem *ff = mallocFFElem(m);
    initPoly(ff->el,m);
    struct FFElem *ffRet = mallocFFElem(m);
    struct FFElem *ffTmp = mallocFFElem(m);
    struct FFElem *ffTmp2 = mallocFFElem(m);
    struct FFElem *ffTmp3 = mallocFFElem(m);
    struct FFElem *ffTmp4 = mallocFFElem(m);

    struct FFElem **matmulCache = malloc(matLen*sizeof(struct FFElem));
    for(i=0;i<matLen;i++) matmulCache[i] = mallocFFElem(m);
    bool *matmulCacheCalced = malloc(matLen*sizeof(bool));

    bool *toTestIndicator = malloc(decompCount*sizeof(bool));
    struct Node **roots = malloc( decompCount*sizeof(struct Node) );
    struct Node **curRoots = malloc(decompCount*sizeof(struct Node*));
    for(i=0;i<decompCount;i++){
        roots[i] = malloc( sizeof(struct Node) );
        roots[i]->ff = 0;
        roots[i]->next = 0;
        curRoots[i] = roots[i];
        toTestIndicator[i] = true;
    }
    //---------------------------------------------------------------------

    int foundCounter = 0;
    initPoly(genCounts,decompCount);

    // chase for elements ------------------------------------------------
```

```
	while(true){
		for(i=0;i<matLen;i++) matmulCacheCalced[i] = 0;
		int curGen = testAllSubmods(ff,mipo,decompCount,
				polys,polysCountPerDecomp,evalToZero,
				mats,frobPowers,toTestIndicator,
				m,tmp,ffTmp,ffTmp2,ffTmp3,
				matmulCache,matmulCacheCalced,
				multTable,addTable);
		if( curGen != -1 ){
			if(toTestIndicator[curGen] == true){
				genCounts[curGen]++;
				appendToEnd(roots[curGen], ff, m);
				foundCounter++;
				toTestIndicator[curGen] = false;
			}
			if(foundCounter == decompCount) break;
		}
		//generate next element
		// (for sure there is a more efficient method)
		ff->el[0] += 1;
		if( ff->el[0] == charac ){
			for(i=0; i<m-1 && ff->el[i]==charac; i++){
				ff->el[i] = 0;
				ff->el[i+1] += 1;
			}
			if( ff->el[m-1] == charac )
				break;
		}
		updateFFElem(ff,m);
	}
	if( foundCounter != decompCount ){
		printf("BAAAD ERROR!!! foundCounter=%i < decompCount=%i\n",
				foundCounter,decompCount);
		exit(0);
	}
	printf("finding time: %.2f\n", (double)(time(NULL)-TIME));
	//-------------------------------------------------------------------



	// Process found elements -----------------------------------------------
	int curDecompPosition = 0;
	for(i=0;i<decompCount-1;i++){
		calcSubmoduleElements(roots[i], mipo,
				polys[curDecompPosition]->lenPoly-1, // *** == maxLenPoly
				genCounts,i, // *** i == curGen
				polys+curDecompPosition, polysCountPerDecomp[i],
				evalToZero+curDecompPosition,
				mats, matLen, frobPowers+curDecompPosition,
				elementsF,
				m,q,tmp,
				ffTmp,ffTmp2,ffTmp3,ffTmp4,
				matmulCache,matmulCacheCalced,
				multTable,addTable);
```

```
        curDecompPosition += polysCountPerDecomp[i];
    }
    printf("all not last time: %.2f\n", (double)(time(NULL)-TIME));
    //----------------------------------------------------------------------

    // Process last Decomposition -------------------------------------------
    int curGen = decompCount-1;
    unsigned long long pcn =
        processLastSubmoduleAndTestPrimitivity(roots,mipo,decompCount,
            polys[curDecompPosition]->lenPoly-1, // *** == maxLenPoly
            genCounts,
            polys+curDecompPosition,polysCountPerDecomp[curGen],
            evalToZero+curDecompPosition,
            mats,matLen,frobPowers+curDecompPosition,
            elementsF,
            m,q,
            barFactors,lenBarFactors,countBarFactors,
            commonBarFactor,lenCommonBarFactor,
            commonBiggestBarFactor,lenCommonBiggestBarFactor,
            matCharac,
            matmulCache,matmulCacheCalced,
            multTable,addTable);
    //----------------------------------------------------------------------

    //free variables
    for(i=0;i<decompCount;i++)
        freeNode(roots[i]);
    free(roots); free(curRoots);

    //free temporary variables
    free(tmp);
    freeFFElem(ff);
    freeFFElem(ffRet);
    freeFFElem(ffTmp);
    freeFFElem(ffTmp2);
    freeFFElem(ffTmp3);
    freeFFElem(ffTmp4);
    for(i=0;i<matLen;i++) freeFFElem(matmulCache[i]);
    free(matmulCache);
    free(matmulCacheCalced);
    free(toTestIndicator);


    printf("total time: %.2f\n", (double)(time(NULL)-TIME));
    return pcn;
}
```

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.

6.5.3 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 5.10) steht.

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

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

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

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

Beispiel 6.35. 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`

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

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

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

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

```
# computes the quadratic free part of an integer
def squarefree(n):
    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`

```
# returns the NOT completely basic divisors of an
# extension n over GF(p^e)
def get_completely_basic_divisors(p,e,n):
    n = Integer(n)
    q = Integer(p**e)
    divs = []
    for d in divisors(n):
        isComplBasic = True
        for r in prime_divisors(n/d):
            if r.divides(ordn(p_free_part(n/d/r,p),q**d)):
                isComplBasic = False
```

```
                break
        divs += [d]
        if isComplBasic: return divs
    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`

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

Ausnutzen regulärer Kreisteilungsmoduln

Sicherlich wollen wir auch Regularität (Definition 5.11) 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`

```
# tests if cyclotomic module C_k,t is regular over F_p^e
def isRegular(p,e, k,t,pi):
    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 5.14 gegeben. Da Regularität lediglich die Anzahl der Teiler des Erweiterungsgrades, deren zugehörige Kreisteilungsmoduln auf vollständige Erzeuger getestet werden müssen, reduziert, wird Satz 5.14 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`

```
# returns tau-divisors for complete generator test of
# the cyclotomic module C_k,t*pi over F_p^e
def get_tau_divisors(p,e, k,t,pi):
    if t != 1:
        print "ERROR␣get_tau_divisors:␣t␣!=␣1␣for␣p=",p,"␣e=",e\
                ,"␣k=",k,"␣t=",t,"␣pi=",pi
        raise Exception("Error␣t!=1")
    q = p**e
    tau = ordn(k,q) / ordn(squarefree(k),q)
    tau = prod(map(lambda ra: ra[0]**floor(ra[1]/2), factor(tau)))
    if isExceptional(p,e, k):
        return [ tau, 2*tau ]
    else:
        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 5.12).

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

```
# tests if n is exceptional over F_p^e
def isExceptional(p,e, n):
    q = p**e
    c = 0
    nbar = n
    while Integer(2).divides(nbar):
        c += 1
        nbar /= 2
    if (q).mod(4) == 3 and c >= 3 and ordn(q, 2**c) == 2:
        return True
    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 6.3.2). Der Test auf Primitivität lässt sich auch vollständig deaktivieren durch die Übergabe von `testPrimitivity =False` (vgl. Bemerkung 6.34).

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 5.10 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`

```
def countCompleteSubmoduleGenerators(F,n, binaryPowers=True, \
        testPrimitivity=True):
    TIME = time.time()
    p = F.characteristic()
    q = F.order();
    e = q.log(p)
    E = F.extension(Integer(n),'a');
    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))
```

```
        for r,k in factors:
            barFactors += [primitiveOrder/r]
        countBarFactors = len(barFactors)
        commonBarFactor = gcd(barFactors)
        commonBiggestBarFactor = max(gcd(barFactors[1:]) / commonBarFactor,1)
        barFactors = map(lambda b: b/commonBarFactor, barFactors)
        curF = 0
        barFactors_tmp = [barFactors[0]]
        for b in barFactors[1:]:
            barFactors_tmp += [ b/commonBiggestBarFactor - curF]
            curF = b/commonBiggestBarFactor
        barFactors = barFactors_tmp

        if binaryPowers:
            barFactors = map(lambda b: get_padic_representation(b,2),barFactors)
            commonBarFactor = get_padic_representation(commonBarFactor,2)
            commonBiggestBarFactor = \
                    get_padic_representation(commonBiggestBarFactor,2)
        else:
            barFactors = map(lambda b: get_padic_representation(b,p),barFactors)
            commonBarFactor = get_padic_representation(commonBarFactor,p)
            commonBiggestBarFactor = \
                    get_padic_representation(commonBiggestBarFactor,p)
        lenCommonBarFactor = len(commonBarFactor)
        lenCommonBiggestBarFactor = len(commonBiggestBarFactor)

        lenBiggestZeroGap = 0
        if not binaryPowers:
            #find biggest gap (i.e. zero-interval)
            lenCurGap = 0
            for b in barFactors+[commonBarFactor]+[commonBiggestBarFactor]:
                i = 0
                while i < len(b):
                    lenCurGap = 0
                    while i<len(b) and b[i] == 0:
                        lenCurGap+= 1
                        i += 1
                    lenBiggestZeroGap = max(lenBiggestZeroGap, lenCurGap)
                    i += 1
    else:
        countBarFactors = 0
        barFactors = []
        commonBarFactor = []
        commonBiggestBarFactor = []
        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 6.2.2 auf.

Listing 6.38: `countCompleteSubmoduleGenerators` Fortsetzung (II)

```
        #generate F elements in E
```

```
    elementsF = []
    if e == 1:
        elementsF = map(lambda e: [e], list(F))
    else:
        h = Hom(F,E)[0]
        for e in itertools.product(xrange(p),repeat=e):
            elementsF += [h( F(list(reversed(e))) ).polynomial().list()]

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

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

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

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

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

```
cdef FFPoly *pyList2FFPoly(listPoly, int m):
    lenPoly = len(listPoly)
    cdef FFPoly *poly = <FFPoly*>malloc(sizeof(FFPoly))
    poly.poly = <FFElem**>malloc(lenPoly*sizeof(FFElem*))
    poly.lenPoly = lenPoly
    for i,e in enumerate(listPoly):
```

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

```
        poly.poly[i] = pyList2FFElem(e,m)
    return poly
```

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

```
cdef FFPoly **pyList2PointFFPoly(listPolys, int m):
    countPolys = len(listPolys)
    cdef FFPoly **polys = <FFPoly**>malloc(countPolys*sizeof(FFPoly*))
    for i,e in enumerate(listPolys):
        polys[i] = pyList2FFPoly(e,m)
    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)

```
    # SETUP C DATA ==============================================================
    maxMatPower = max(map(lambda d: euler_phi(d[0])*d[1]*d[2], decomposition))
        # multiplication and addition table
    cdef np.ndarray[int,ndim=1,mode="c"] multTableRawC\
        = np.array(multTable, dtype=np.int32)
    cdef np.ndarray[int,ndim=1,mode="c"] addTableRawC\
        = np.array(addTable, dtype=np.int32)
    cdef int* multTableC = <int*>multTableRawC.data + <int>((p-1)**2)
    cdef int* addTableC = <int*>addTableRawC.data + <int>(2*(p-1))
        #setup mipo
    cdef FFElem *mipoC = pyList2FFElem(mipo,m+1)
       #setup matrices
    cdef FFElem **matsC = genFrobMats(mipoC,m,maxMatPower,q,
            multTableC, addTableC)
        # mat charac
    cdef FFElem **matCharacC
    if binaryPowers:
        matCharacC = <FFElem**>0
    else:
        matCharacC = genFrobMats(mipoC,m,lenBiggestZeroGap+1,
               p, multTableC, addTableC)
    #setup polynomials, polyLength, frobPowers, evaltoZero
    decompCount = int(len(polysCount))
        #evalToZeroC
    cdef np.ndarray[char,ndim=1,mode="c",cast=True] evalToZeroC\
            = np.array(evalToZero, dtype=np.uint8)
        #frobPowersC
    cdef np.ndarray[int,ndim=1,mode="c"] frobPowersC\
            = np.array(frobPowers, dtype=np.int32)
        #polysCountC
    cdef np.ndarray[int,ndim=1,mode="c"] polysCountC\
            = np.array(polysCount, dtype=np.int32)
    cdef FFPoly **polysC = pyList2PointFFPoly(polys,m)
        # bar Factors
    cdef np.ndarray[int,ndim=1,mode="c"] barFactorsC \
        = 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))
```

```
freeFFElemMatrix(elementsFC,len(elementsF))
#==========================================================================
return prod(genCounts.values()), Integer(pcn), genCounts,\
       strfdelta(datetime.timedelta(seconds=(time.time()-TIME)))
```

6.5.4 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

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

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 $r \nmid \mathrm{ord}_{(\frac{n}{dr})'}(q^d)$ (vgl. Satz 5.3). Wir brechen jeweils ab, falls ein r die Teilbarkeitsbedingung nicht erfüllt.

d	$\frac{n}{d}$	r	$(\frac{n}{dr})'$	$\mathrm{ord}_{(\frac{n}{dr})'}(q^d)$	$r \nmid \mathrm{ord}_{(\frac{n}{dr})'}(q^d)$
1	6	2	3	2	$\lightning$ $\rightsquigarrow$ nicht einfach
2	3	3	1	1	$\checkmark$
					$\rightsquigarrow$ 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 $[\ ,\ ,\ldots]$ 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.

$$\begin{tikzcd}[column sep=small] & (1,3,2) \arrow[dl, dash, "{(k,\frac{t}{r},\pi)}"'] \arrow[dr, dash, "{(kr,\frac{t}{r},\pi)}"] & & \boxed{r = 3 \rightsquigarrow 3^1 \nmid \mathrm{ord}_{\nu(kt')}(q) = \mathrm{ord}_3(2) = 2} \\ (1,1,2) & & (3,1,2) & \end{tikzcd}$$

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 $(k, t, pi) := (1, 1, 2)$ und wir müssten alle Teiler des Modulcharakters $\frac{k\,t\,\pi}{\nu(k)} = 2$, 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:

$$\mathrm{ord}_{\nu(k\,t')}(q) \;=\; \mathrm{ord}_1(2) \;=\; 1\,.$$

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 $(k, t, \pi) := (3, 1, 2)$. Der Modulcharakter ist wiederum $\frac{k\,t\,\pi}{\nu(k)} = 2$. 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 5.7), 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 5.6), 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

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

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

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

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

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:

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

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:

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

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 6.2). 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. 6.5.1). 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:

$$\texttt{roots} = \big[\,[a^3 + a^2 + a],\ [a^2 + a]\,\big]$$

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 6.32 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 6.32 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 6.5.2), 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 5.9).

$(a^3+a^2+a) + (a^2+a)$. 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 $\mathbb{F}_2$.

6.6 Auswertung der Ergebnisse

Alle berechneten Ergebnisse werden in Tabellen im Anhang bereitgestellt. Wie man erkennen kann, konnten wir alle Werte von Morgan und Mullen aus [16] reproduzieren. Darüber hinaus konnten wir die konkreten Anzahlen der Erzeuger der jeweiligen nicht weiter zerlegbaren Kreisteilungsmoduln bestimmen (vgl. Tabellen), was gerade aus theoretischer Sicht ein interessantes Resultat ist, da diese Zahlen möglicherweise auf der Suche nach einer allgemeinen Formel für die vollständigen Erzeuger eines Kreisteilungsmoduls helfen können (Wir erinnern uns, dass Satz 6.19 lediglich für reguläre Kreisteilungsmodule gilt). Zuletzt sei erwähnt, dass die obig vorgestellte Implementierung nur eine unwesentliche Menge an Arbeitsspeicher erfordert, wodurch weitere Ergebnisse problemlos produziert werden könnten, wenn man Rechenzeiten von über einer Woche pro Körpererweiterung in Kauf nimmt. Die längste hier in Kauf genommene Rechenzeit lag für $(q,n)=(243,4)$ bei 98 Stunden und 4 Minuten.

6.7 Existenz von primitiv vollständig normalen Elementen

6.7.1 Theoretische Aspekte

Zu Beginn dieses Kapitels wurde die Bezeichnung $\mathcal{G}$ als die Menge der $n \in \mathbb{N}^*$ eingeführt, für die Erweiterungen von Grad n über *jedem* endlichen Köper ein $\mathcal{PCN}$-Element enthalten und in Problem 6.10 haben wir uns das Ziel gesetzt, möglichst viele n anzugeben, die in $\mathcal{G}$ liegen. Zunächst können wir alle Erweiterungen aufnehmen, für die n über jedem q einfach ist, da hier die Existenz eines primitiv vollständig normalen Elementes der eines primitiv normalen entspricht, die nach Satz 6.13 gesichert ist. Des Weiteren sichert Satz 6.14, dass wir auch a priori alle ungeraden

Erweiterung aufnehmen können, die regulär über jedem Grundkörper sind. Dazu geben wir einige Beispiele an (vgl. [9, Abschnitt vor Section 2]):

Lemma 6.36. *Sei $n \in \mathbb{N}^*$. Dann gilt: n ist regulär über jeder Primzahlpotenz $q > 1$, falls eine der nachstehenden Bedingungen erfüllt ist:*

(1) n ist Potenz einer beliebigen Primzahl.

(2) $n = N^s$ für $s \geq 1$ und N eine Carmichael Zahl *(Eine* Carmichael Zahl *ist eine ungerade natürliche Zahl N, sodass für jeden Primteiler r von N gilt: $r-1$ teilt $N-1$).*

Beweis. (1) Sei $n = r^s$ für eine Primzahl r und $q = p^e$ für eine Primzahl p, so ist für $r = p$ klar, dass $\mathrm{ord}_{\nu(n)}(q) = \mathrm{ord}_1(q) = 1$ gilt. Für $r \neq p$ haben wir $\mathrm{ord}_{\nu(n)}(q) = \mathrm{ord}_r(q) \mid \varphi(r) = r-1$ nach Lemma 2.7 und damit ist (n, q) regulär, da $\mathrm{ggT}(r, r-1) = 1$.

(2) Schreibe $\nu(N^s) = \prod_{i=1}^k r_i$ für Primzahlen $r_1, \ldots, r_k$. Dann ist wie oben $\mathrm{ord}_{\nu(N^s)}(q)$ ein Teiler von $\varphi(\nu(N^s)) = \prod_{i=1}^k \varphi(r_i) = \prod_{i=1}^k (r_i - 1)$. Nun gilt jedoch $(r_i - 1) \mid (N-1) \mid (N^s - 1)$ und damit $\mathrm{ggT}(r_i - 1, N^s) = 1$ für alle $i = 1, \ldots, k$. □

Damit können wir bereits eine große Teilmenge von $\mathcal{G}$ ausmachen:

Proposition 6.37.

Es gilt:

$$\{N^s : N \text{ Carmichael Zahl}, s \in \mathbb{N}^*\} \cup \{r^s : r \text{ Primzahl}, s \in \mathbb{N}^*\} \quad \subset \quad \mathcal{G}.$$

Beweis. Klar nach Korollar 5.4 und Lemma 6.36. □

Doch wie können wir $\mathcal{G}$ noch größer werden lassen? Betrachten wir einmal alle $n \in \{2, 3, \ldots, 40\}$, so sehen wir mit obiger Proposition, dass lediglich für

$$6,\ 12,\ 14,\ 15,\ 18,\ 20,\ 21,\ 22,\ 24,\ 26,\ 28,\ 30,\ 34,\ 35,\ 36,\ 38,\ 40$$

diese Frage noch offen ist. Mit Hilfe von Satz 6.15 brauchen wir für jedes n „nur“ alle $q < n^4$ auf Existenz eines $\mathcal{PCN}$-Elements zu überprüfen. Daher bietet es sich an, an dieser Stelle erneut `Sage` zu bemühen, um letztlich nachstehenden Satz beweisen zu können:

Satz 6.38.

Für alle $n \in \mathbb{N}^*$ mit $2 \leq n \leq 40$ gilt $n \in \mathcal{G}$.

6.7.2 Implementierung einer $\mathcal{PCN}$-Suche I

Im Gegensatz zur Enumeration einer einzigen Körpererweiterungen geht es nun darum, viele Körpererweiterungen mit immer gleichem Erweiterungsgrad zu betrachten. Beispielsweise für $n = 40$ haben wir 187468 Körpererweiterungen zu prüfen, wobei das größte auftauchende q gleich 2560021 ist und damit der größte zu betrachtende Körper 21 366 880 187 906 632 687 514 074 105 613 675 429 853 637 389 181 949 057 288 253 929 247 658 456 282 382 468 815 886 964 073 599 674 916 653 373 765 754

199 896 430 060 696 944 213 682 519 652 698 462 471 628 340 522 570 646 913 923 734 591 660 041 323 668 864 602 928 439 707 352 577 304 227 818 869 137 311 550 893 478 904 281 581 752 801 Elemente besitzt. Da in diesen Bereichen das Aufstellen von Frobenius-Matrizen, Additions- und Multiplikationstabellen nicht mehr praktikabel ist, haben wir uns entschieden, die gesamte Existenzsuche in `Sage` zu implementieren.

Den Test auf vollständige Normalität organisieren wir anhand nachstehendem Lemma:

Lemma 6.39. *Sei $E := \mathbb{F}_{q^n}$ über $F := \mathbb{F}_q$ eine Erweiterung endlicher Körper. Sei $x^n - 1 = \prod_{i=1}^{l} \Phi_{k_i,t_i}^{\pi}$ die feinst mögliche Zerlegung nach dem Zerlegunssatz (Satz 5.10). Notiere*

$$D_1 \;:=\; \{d \mid n :\ \nexists d' \mid d :\ \mathbb{F}_{q^n} \text{ über } \mathbb{F}_{q^{d'}} \text{ einfach}\}$$

und

$$D_2 \;:=\; \{d \in \mathbb{N}^* :\ d \mid \tfrac{k_i t_i \pi}{\nu(k_i)} \text{ für ein } i = 1, \ldots, l\}\,.$$

Dann sind für $u \in E$ äquivalent:

(1) u ist vollständig normal über F.

(2) Für alle $d \in D := D_1 \cap D_2$ ist $\mathrm{Ord}_{q^d}(u) = x^{\frac{n}{d}} - 1$.

(3) Für alle $d \in D$ gilt: Seien $x^{\frac{n}{d}} - 1 = \prod_{j=1}^{l_d} f_i(x)^{\nu_i}$ die vollständige Faktorisierung von $x^{\frac{n}{d}} - 1$ über $\mathbb{F}_{q^d}$ und $\bar{f}_i(x) = \frac{x^{\frac{n}{d}}-1}{f_i(x)}$ für $i = 1, \ldots, l_d$ die jeweiligen Kofaktoren, so gilt

$$\bar{f}_i(\sigma^d)(u) \;\neq\; 0$$

für alle $i = 1, \ldots, l_d$, wobei wie immer $\sigma : \bar{F} \to \bar{F}, x \mapsto x^q$ den Frobenius-Endomorphismus von F bezeichne.

Beweis. Die Äquivalenz von (2) und (3) ist klar mit der Definition der q-Ordnung, wobei es klar ist, dass $(x^{\frac{n}{d}} - 1)(\sigma^d)(u) = 0$ nicht mehr überprüft werden muss, da es das Minimalpolynom von σ^d über $\mathbb{F}_{q^d}$ ist. Letztlich bleibt also nur ein Wort darüber zu verlieren, warum es ausreicht nur Teiler aus D zu betrachten: Hier stellen wir fest, dass es klar ist, lediglich Teiler aus D_1 zu betrachten, da alle weiteren Teiler von n bereits Zwischenkörper einer einfachen Erweiterung sind. Bleibt D_2 zu klären. Dazu zerlege $u = u_1 + \ldots + u_l$ mit $u_i \in \mathcal{C}_{k_i,t_i\pi}$ für $i = 1, \ldots, l$. Nun gilt nach der Definition von vollständigen Erzeugern (Definition 5.7), dass u_i genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k_i,t_i\pi}$ ist, wenn $\mathrm{Ord}_{q^d}(u_i) = \Phi_{k_i,t_i}^{\pi}$ für alle Teiler d des Modulcharakters $\frac{k_i t_i \pi}{\nu(k_i)}$. Ist damit (3) erfüllt, so sind alle u_i vollständige Erzeuger und nach Definition verträglicher Zerlegungen (Definition 5.9) folgt (1). □

Konkret ist der Test auf vollständige Normalität wie folgt gegeben.

Listing 6.45: Aus `../Sage/findAnyPCN_trinom.spyx`

```
# Tests x as Element of E on complete normality, i.e. tests for each
# d in divs, if the corresponding polynomials in prodsAll over the corresponding
# field in fieldsAll vanishes on frobenius evaluation of x.
# fieldsAll and facsAll are dicts indexed by the divisors of divs, where
# fieldsAll[d] is the corresponding intermediate field of order q^d
# and facsAll[d] is the factorization of x^(n/d)-1 over GF(q^d).
# prodsAll[d] is the list of all possible cofactors of above factorization.
```

```
def isCompletelyNormal(x,E, q, divs, fieldsAll, facsAll, prodsAll):
    if x == E.zero(): return False
    #test isNormal for each divisor
    pows = dict()
    for d in divs:
        h = Hom(fieldsAll[d],E)[0];
        for idx,(f,mult) in enumerate(facsAll[d]):
            g = prodsAll[d][idx];
            ret = E.zero();
            iold = 0
            xiold = x
            for i,gi in enumerate(list(g)):
                if pows.has_key(i*d):
                    xi = pows[i*d];
                    iold = i*d
                    xiold = xi
                else:
                    xi = xiold**(q**(d*i-iold));
                    pows[i*d] = xi;
                    xiold = xi
                    iold = i*d
                ret += h(gi)*xi
            if ret == 0: return False;
    return True
```

Hier wurde auf eine Anwendung der Frobenius-Auswertung durch Matrixmultiplikation verzichtet, da dies via purem `Sage`-Code wesentlich langsamer ist, als Potenzieren. Wie man jedoch erkennen kann, werden bereits berechnete Potenzen wiederverwendet, um hier unnötigen Rechenaufwand einzusparen. Die Parameter `divs`, `fieldsAll`, `facsAll` und `prodsAll` werden jeweils in den übergeordneten Funktionen `findAnyPCN_polynom` (Listing 6.48) und `findAnyPCN_polynom_prime` (Listing 6.47) wie folgt generiert.

Listing 6.46: Aus `../Sage/findAnyPCN_trinom.spyx`

```
    #setup factors of x^n-1
    divs = get_proper_subfield_divisors(p,r,n)
    facsAll = dict();
    prodsAll = dict();
    fieldsAll = dict();
    for d in divs:
        G = F.extension(Integer(d), 'c');
        Gx = PolynomialRing(G,'x');
        fieldsAll[d] = G;
        facsAll[d] = list((Gx.gen()**(n/d)-1).factor());
        prodsAll[d] = dict();
        for idx,(f,mult) in enumerate(facsAll[d]):
            prodsAll[d][idx] = (Gx.gen()**(n/d)-1).quo_rem(f)[0]
```

Bereits Morgan und Mullen haben in [16] für alle $p^n < 10^{50}$ mit $p \leq 97$ ein primitives, vollständig normales Polynom von Grad n über $\mathbb{F}_p$ angegeben. Betrachtet man diese Tabellen, so ist auffällig, dass sehr viele dieser Polynome Trinome sind. Man kann sich leicht überlegen, dass ein Binom nicht Minimalpolynom eines vollständig normalen Elements einer Körpererweiterung sein kann und bemerke, dass sich zwei Koeffizienten des Minimalpolynoms eines primitiv vollständig normalen Elements etwas eingrenzen lassen:

Lemma 6.40. *Sei $u \in \mathbb{F}_{q^n}$ über $\mathbb{F}_q$ ein primitiv vollständig normales Element und $f(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_0 \in \mathbb{F}_q[x]$ sein Minimalpolynom. Dann gilt*

(1) $a_{n-1} = -\operatorname{Tr}_{\mathbb{F}_{q^n}|\mathbb{F}_q}(u) \neq 0$ und

(2) $(-1)^n a_0 = \operatorname{Nm}_{\mathbb{F}_{q^n}|\mathbb{F}_q}(u)$ ist primitiv in $\mathbb{F}_q$.

Beweis. Die beiden Identitäten sind klar durch Koeffizientenvergleich von $f(x) = (x-u)(x-u^q)(x-u^{q^2})\cdot\ldots\cdot(x-u^{q^{n-1}})$.[3] Die Spur eines (vollständig) normalen Elements einer Körpererweiterungen ist stets ungleich Null, da sie ja gerade die Summe aller Basiselemente der von jenem Element erzeugten Normalbasis ist und diese ja über dem Grundkörper linear unabhängig sind. Die Primitivität der Norm erhalten wir sofort durch

$$\operatorname{Nm}_{\mathbb{F}_{q^n}|\mathbb{F}_q}(u) = u \cdot u^q \cdot \ldots \cdot u^{q^n-1} u^{\frac{q^n-1}{q-1}} .$$

Aus der Primitivität von u folgt nun $\operatorname{ord}(u^{\frac{q^n-1}{q-1}}) = q-1$ und damit $\operatorname{ord}((-1)^n a_0) = q-1$. □

In der Hoffnung möglichst viele Trinome vorzufinden, haben wir uns das Ziel gesetzt für jede Primzahlpotenz q mit $q < n^4$ für einen gegebenen Erweiterungsgrad n das „kleinste“ primitiv vollständig normale Polynom über $\mathbb{F}_q$ von Grad n zu bestimmen. Das „kleinste“ beziehe sich dabei auf folgende Ordnungsrelation:

Definition 6.41.

Seien $f(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_0$ und $g(x) = x^n + b_{n-1}x^{n-1} + \ldots + b_0$ zwei Polynome gleichen Grades über $\mathbb{F}_q$. Ferner bezeichne $i(f)$ dasjenige Wort über dem Alphabet $\{0, \ldots, n\}$, das die Indizes der nicht verschwindenden Koeffizienten von f in aufsteigender Reihenfolge repräsentiert, d.h. ist $i(f) = i_1 i_2 \ldots i_k$, so gilt $i_1 < i_2 < \ldots < i_k$ und $a_{i_j} \neq 0$ für alle $j \in \{i_1, \ldots, i_k\}$. Analog sei $a(f)$ das Wort über dem Alphabet $\mathbb{F}_q$, das die nicht verschwindenden Koeffizienten von f ihrem Index aufsteigend nach repräsentiert, d.h. ist $a(f) = a_{i_1} a_{i_2} \ldots a_{i_k}$, so ist $i_1 < i_2 < \ldots < i_k$ und für alle $j \in \{i_1, \ldots, i_k\}$ gilt: a_{i_j} ist der nicht verschwindende Koeffizient von x^{i_j} in f.

Dann heißt f *kleiner oder gleich* g, geschrieben $f \leq g$, falls gilt:

(1) $i(f)$ ist lexikographisch kleiner oder gleich $i(g)$ und bei Gleichheit gilt zusätzlich:

(2) $a(f)$ ist kleiner oder gleich $a(g)$, wobei eine Ordnung auf $\mathbb{F}_q$ gegeben ist ist wie folgt:

- Ist $q = p$ für eine Primzahl p, so wähle die natürliche Ordnung von $\mathbb{F}_p \cong \{0, 1, \ldots, p-1\}$.
- Sonst wähle einmalig eine Repräsentation von $\mathbb{F}_q$ durch $\mathbb{F}_p[x]/(h(x))$ und definiere u kleiner gleich v für $u, v \in \mathbb{F}_q$ als $a(x) \leq b(x)$ via dieser Definition für Repräsentanten $a(x)$ von u und $b(x)$ von v mit $a(x), b(x) \in F_p[x]$ und $\deg(a), \deg(b) < \deg(h)$.

Wir haben uns für diese Definition einer Ordnung auf $\mathbb{F}_q[x]$ entschieden, da hier stets Polynome

[3] Man kennt diesen Zusammenhang der Koeffizienten eines Polynoms mit seinen Nullstellen auch unter dem Namen „elementarsymmetrische Funktionen“.

- mit kleinerem Hamming-Gewicht, d.h. mit kleinerer Anzahl an nicht verschwindenden Koeffizienten,
- mit kleineren Exponenten,
- mit „kleineren" Koeffizienten (vgl. obige Definition)

bevorzugt werden.

Die konkrete Suche nach einem $\mathcal{PCN}$-Polynom für ein Paar (n, q) teilen wir in zwei verschiedene Funktionen auf, wobei wir `findAnyPCN_polynom_prime` nutzen wollen, falls q eine Primzahl ist und `findAnyPCN_polynom`, falls q eine echte Primzahlpotenz ist. Der wesentliche Unterschied liegt dabei in der Enumeration des Grundkörpers $\mathbb{F}_q$, einmal simplerweise durch $\{0, 1, \ldots, p-1\}$ darstellbar, falls $q = p$ eine Primzahl ist. Andernfalls müssen wir uns wieder des Polynomrings über dem Primkörper $\mathbb{F}_p$ bedienen, um die Elemente in $\mathbb{F}_q \cong \mathbb{F}_p[x]/(f(x))$ für $q = p^r$ für $r > 1$ darstellen zu können. Konkret:

Listing 6.47: Aus `../Sage/findAnyPCN_trinom.spyx`

```
# special function for testing extensions of PrimeFields
def findAnyPCN_polynom_prime(p,n):
    p = Integer(p)
    n = Integer(n)
    F = GF(p)

    Fx = PolynomialRing(F,'x')

    orderE = p**n
    primOrder = orderE-1

    primitives = []

    #setup factors of x^n-1
    divs = get_proper_subfield_divisors(p,1,n)
    facsAll = dict();
    prodsAll = dict();
    fieldsAll = dict();
    for d in divs:
        G = F.extension(Integer(d), 'c');
        Gx = PolynomialRing(G,'x');
        fieldsAll[d] = G;
        facsAll[d] = list((Gx.gen()**(n/d)-1).factor());
        prodsAll[d] = dict();
        for idx,(f,mult) in enumerate(facsAll[d]):
            prodsAll[d][idx] = (Gx.gen()**(n/d)-1).quo_rem(f)[0]

    firstRun = True
    # first test trinoms!
    for coeffT in xrange(1,p):
        if firstRun:
            if is_even(n):
                prange = xrange(1,p)
            else:
                prange = xrange(p-1,0,-1)
            for coeffN in prange:
```

```
37                 if F(coeffN).multiplicative_order() != p-1: continue
38                 if is_even(n):
39                     primitives += [coeffN]
40                 else:
41                     coeffN *= (-1)
42                     primitives = [coeffN] + primitives
43                 f = Fx.gen()**n + coeffT*Fx.gen()**(n-1) + coeffN
44                 if not f.is_irreducible(): continue
45 
46                 E = GF(orderE, name='a', modulus=f)
47 
48                 if E.gen().multiplicative_order() == primOrder \
49                         and isCompletelyNormal(E.gen(),E,p,\
50                             divs,fieldsAll,facsAll,prodsAll):
51                     return E.gen(),f
52             firstRun = False
53         else:
54             for coeffN in primitives:
55                 f = Fx.gen()**n + coeffT*Fx.gen()**(n-1) + coeffN
56                 if not f.is_irreducible(): continue
57                 E = GF(orderE, name='a', modulus=f)
58 
59                 if E.gen().multiplicative_order() == primOrder \
60                         and isCompletelyNormal(E.gen(),E,p,\
61                             divs,fieldsAll,facsAll,prodsAll):
62                     return E.gen(),f
63     # test rest
64     for length in xrange(1,n-1):
65         for idcs in itertools.combinations(xrange(1,n-1),length):
66             for xs in itertools.product(xrange(1,p),repeat=length+1):
67                 for x in primitives:
68                     f = Fx.gen()**n + xs[0]*Fx.gen()**(n-1) + x
69                     for j,j2 in enumerate(idcs):
70                         f += xs[length-j] * Fx.gen() ** j2
71                     if not f.is_irreducible(): continue
72                     E = GF(orderE, name='a', modulus=f)
73 
74                     if E.gen().multiplicative_order() == primOrder \
75                             and isCompletelyNormal(E.gen(),E,p,\
76                                 divs,fieldsAll,facsAll,prodsAll):
77                         return E.gen(),f
```

Wie man erkennen kann stützen wir uns auf die Hoffnung Trinome vorzufinden und beginnen unsere Suche daher bei deren Enumeration. Ferner machen wir uns Lemma 6.40 zu Nutze und bestimmen anfangs dynamisch die primitiven Elemente des Grundkörpers (vgl. Zeilen 37-43). Um die Ordnung nach Definition 6.41 beizubehalten, sind wir hier gewzungen eine Fallunterscheidung in n gerade oder ungerade zu vollziehen (vgl. Zeile 38). Ebenfalls ein Vorteil gegenüber `findAnyPCN_polynom` ist die Tatsache, dass der Erweiterungskörper `E` durch das zu testende Polynom gegeben werden kann und so dieses nicht erst über `E` faktorisiert werden muss (vgl. Zeilen 46, 48).

Es sei ferner angemerkt, dass `Sage` eine Funktion `is_primitive` für Polynome besitzt, die gerade testet, ob ein Polynom primitiv ist oder nicht. Jedoch hat sich herausgestellt, dass dies um ein Vielfaches länger dauert, als die hier beschriebene Methode in Zeile 48.

Falls der Grundkörper kein Primkörper ist, organisieren wir die Suche analog, wobei wir gewzungen sind, einige „Umwege“ zu gehen.

Listing 6.48: Aus ../Sage/findAnyPCN_trinom.spyx

```
def findAnyPCN_polynom(p,r,n):
    if r == 1:
        return findAnyPCN_polynom_prime(p,n)

    q = p**r
    F = GF(q,’a’)

    E = F.extension(n,’a’)
    P = E.prime_subfield()

    Px = PolynomialRing(P,’x’)
    Fx = PolynomialRing(F,’x’)
    Ex = PolynomialRing(E,’x’)
    h = Hom(F,E)[0]
    primOrder = E.order()-1

    primitives = []

    #setup factors of x^n-1
    divs = get_proper_subfield_divisors(p,r,n)
    facsAll = dict();
    prodsAll = dict();
    fieldsAll = dict();
    for d in divs:
        G = F.extension(Integer(d), ’c’);
        Gx = PolynomialRing(G,’x’);
        fieldsAll[d] = G;
        facsAll[d] = list((Gx.gen()**(n/d)-1).factor());
        prodsAll[d] = dict();
        for idx,(f,mult) in enumerate(facsAll[d]):
            prodsAll[d][idx] = (Gx.gen()**(n/d)-1).quo_rem(f)[0]

    # list elements of F
    Flist = [F.zero()]
    for i in xrange(1,r+1):
        for idcs in itertools.combinations(xrange(0,r),i):
            for koeffs in itertools.product(xrange(1,p),repeat=i):
                Flist += [F(list(sum([e * Px.gen()**idcs[j] for \
                              j,e in enumerate(reversed(koeffs))])))]

    if not is_even(n):
        FprimList = [F.zero()]
        for i in xrange(1,r+1):
            for idcs in itertools.combinations(xrange(0,r),i):
                for koeffs in itertools.product(xrange(p-1,0,-1),repeat=i):
                    FprimList += [F(list(sum([e * Px.gen()**idcs[j] for \
                                  j,e in enumerate(reversed(koeffs))])))]

    firstRun = True
    # first test trinoms!
```

```
for coeffT in xrange(1,q):
    coeffTF = Flist[coeffT]
    if firstRun:
        for coeffN in xrange(p,q):
            if is_even(n):
                coeffNF = Flist[coeffN]
            else:
                coeffNF = FprimList[coeffN]
            if coeffNF.multiplicative_order() != F.order()-1: continue
            if not is_even(n): coeffNF *= (-1)
            primitives += [coeffNF]
            f = Fx.gen()**n + coeffTF*Fx.gen()**(n-1) + coeffNF
            if not f.is_irreducible(): continue
            for fac,mul in Ex(f.map_coefficients(h)).factor():
                if fac.degree() == 1:
                    x = -fac[0]
                    if isCompletelyNormal(x,E,q,divs,\
                                fieldsAll,facsAll,prodsAll) and \
                            x.multiplicative_order() == primOrder:
                        return x,f
                    else: break
                else: break
        firstRun = False
    else:
        for coeffNF in primitives:
            f = Fx.gen()**n + coeffTF*Fx.gen()**(n-1) + coeffNF
            if not f.is_irreducible(): continue
            for fac,mul in Ex(f.map_coefficients(h)).factor():
                if fac.degree() == 1:
                    x = -fac[0]
                    if isCompletelyNormal(x,E,q,divs,\
                                fieldsAll,facsAll,prodsAll) and \
                            x.multiplicative_order() == primOrder:
                        return x,f
                    else: break
                else: break
# test rest
for length in xrange(1,n-2):
    for idcs in itertools.combinations(xrange(1,n-1),length):
        for xs in itertools.product(xrange(1,q),repeat=length+1):
            for x in primitives:
                f = Fx.gen()**n + Flist[xs[0]]*Fx.gen()**(n-1) + x
                for j,j2 in enumerate(idcs):
                    f += Flist[xs[length-j]] * Fx.gen() ** j2
                if not f.is_irreducible(): continue
                for fac,mul in Ex(f.map_coefficients(h)).factor():
                    if fac.degree() == 1:
                        y = -fac[0]
                        if isCompletelyNormal(y,E,q,divs,\
                                fieldsAll,facsAll,prodsAll) and \
                                y.multiplicative_order() == primOrder:
                            return y,f
                        else: break
                    else: break
```

Man erkennt, dass wir hier die Elemente aus F eigens generieren müssen, um den Bedingungen aus Definition 6.41 gerecht zu werden.

Sicherlich ist klar, dass man auch eine Funktion braucht, die gleich alle q für gegebenes n testet:

Listing 6.49: Aus `../Sage/findAnyPCN_trinom.spyx`

```
def findAnyPCN_polynom_wrapper(n, border=lambda n:n**4, \
        fileoutput=False, filepath="pcns_trinom_", \
        startPrime=1, stopPrime=0, onlyR=None, \
        cpuNum=1):
    if fileoutput:
        st = datetime.datetime.\
                fromtimestamp(time.time()).strftime('%Y-%m-%d_%H:%M:%S')
        filepath += str(n)+"_"
        if onlyR != None: filepath += str(onlyR)+"_"
        filepath += st
    border = border(n)
    p = startPrime
    if onlyR != None and p**onlyR > border: return

    gen = runGenerator(border,startPrime,stopPrime,onlyR)
    pool = Pool(cpuNum)
    for p,r,n,(x,pol) in pool.imap( findAnyPCN_polynom__star, \
            ((p,r,n) for p,r in gen) ):
        print "(",p,", ",r,") = ", pol
        if fileoutput:
            with open(filepath,'a') as f:
                f.write(str(p)+"\t"+str(r)
                        +"\t"+str(pol)+"\n")
            f.close();
    pool.close()
    pool.join()
```

Wie man erkennen kann, hat es sich als Vorteilhaft erwiesen die Argumente `startPrime`, `stopPrime` und `onlyR` einzuführen, wobei letzteres lediglich Paare (q, n) testet, bei denen $q = p^r$ mit $r =$ `onlyR`. (Insbesondere eine Trennung zwischen `onlyR` $= 1$ und dem Rest war hilfreich, da es stets sehr viele Primzahlen p mit $p < n^4$ gibt, jedoch nur wenige, für die eine echte Potenz immernoch kleiner n^4 ist.)

Wir schließen mit den beiden Hilfsfunktionen, die in obiger Funktion benutzt werden, um der Syntax von `Pool.imap` gerecht zu werden.

Listing 6.50: Aus `../Sage/findAnyPCN_trinom.spyx`

```
def findAnyPCN_polynom__star(prn):
    return prn[0],prn[1],prn[2],findAnyPCN_polynom(*prn)
```

Listing 6.51: Aus `../Sage/findAnyPCN_trinom.spyx`

```
def runGenerator(border,startPrime,stopPrime,onlyR):
    p = startPrime
```

```
    while p < border :
        if stopPrime != 0 and p > stopPrime: return
        p = next_prime(p)
        # consider only one r
        if onlyR != None:
            r = onlyR
            if p**r > border: continue
            yield p,r
        # consider all rs
        else:
            r = 1
            q = p**r
            while q < border:
                yield p,r
                r += 1
                q = p**r
```

In den Tabellen in **??** findet man die Ergebnisse unsere computergestützten Suche.

6.7.3 Implementierung einer $\mathcal{PCN}$-Suche II

Leider waren wir nicht in der Lage für alle Paare (p^r, n) obige Implementierung zu nutzen, da gerade für große r in sinnvoller Rechenzeit kein $\mathcal{PCN}$-Polynom gefunden werden konnte. Um für diese Ausnahmen dennoch ein $\mathcal{PCN}$-Polynom präsentieren zu können, geben wir für diese Erweiterungen ein beliebiges $\mathcal{PCN}$-Polynom an (also nicht wie oben das beste im Sinne der Ordnung aus Definition 6.41). Dieses finden wir wie folgt: Für gegebenes (q, n) bestimmen wir ein primitives Element $u \in \mathbb{F}_{q^n}$ via der `Sage`-Funktion `primitive_element()`. Anschließend iterieren wir aufsteigend über alle $i \in \mathbb{N}^*$ mit $\mathrm{ggT}(i, q^n - 1) = 1$ und beenden die Suche mit Ausgabe des Minimalpolynoms von u^i über $\mathbb{F}_q$, falls u^i vollständig normal ist. Nach Satz 1.1 (5) ist klar, dass wir u^i nicht mehr auf Primitivität zu testen brauchen. In `Sage` sieht dieses Vorgehen konkret wie folgt aus.

Listing 6.52: Aus `../Sage/findAnyPCN_additional.spyx`

```
def findAnyPCN(p,r,n):
    q = p**r
    F = GF(q,'a')

    E = F.extension(n,'a')
    P = E.prime_subfield()

    Px = PolynomialRing(P,'x')
    Fx = PolynomialRing(F,'x')
    Ex = PolynomialRing(E,'x')
    h = Hom(F,E)[0]
    primOrder = E.order()-1

    primitives = []

    #setup factors of x^n-1
    divs = get_proper_subfield_divisors(p,r,n)
```

```
    facsAll = dict();
    prodsAll = dict();
    fieldsAll = dict();
    for d in divs:
        G = F.extension(Integer(d), 'c');
        Gx = PolynomialRing(G,'x');
        fieldsAll[d] = G;
        facsAll[d] = list((Gx.gen()**(n/d)-1).factor());
        prodsAll[d] = dict();
        for idx,(f,mult) in enumerate(facsAll[d]):
            prodsAll[d][idx] = (Gx.gen()**(n/d)-1).quo_rem(f)[0]

    # get one primitive element
    x = E.primitive_element()

    lasti = 0
    y = x
    for i in itertools.count(1):
        if gcd(i,E.order()-1) != 1: continue
        y = y*x**(i-lasti)
        lasti = i
        if isCompletelyNormal(y,E,q,divs,fieldsAll,facsAll,prodsAll):
            mipo = y.minpoly()
            for f,i in Fx(mipo).factor():
                if f(y) == E.zero():
                    return f
```

Die „unschönen" Ergebnisse dieser Funktion wurden im Anhang speziell gekennzeichnet.

6.7.4 Auswertung der Ergebnisse

Wie man anhand der Tabellen im Anhang erkennen kann, konnten wir nachweisen, dass für sehr viele Erweiterungen primitiv vollständig normale Trinome existieren. Inbesondere kann man erkennen, dass wenn q nur groß genug wird, offenbar stets ein $\mathcal{PCN}$-Trinom vorhanden ist. Dieses faszinierende Resultat legt die folgende Vermutung nahe, die bisher gänzlich unklar bleibt und für die zumindest im Rahmen dieser Arbeit keine Möglichkeit eines Beweises gefunden werden konnte.

Vermutung 6.42.

Für jedes $n \in \mathbb{N}^*$ existiert ein $P \in \mathbb{N}^*$, so dass für alle Primzahlpotenz $q \geq p$ ein primitiv vollständig normales Trinom von Grad n über $\mathbb{F}_q$ existiert.

Literaturverzeichnis

[1] I. F. Blake, S. Gao und R. C. Mullin. Specific irreducible polynomials with linearly independent roots over finite fields. *Linear algebra and its applications*, **253**, 227–249, 1997.

[2] D. Blessenohl und K. Johnsen. Eine verschärfung des satzes von der normalbasis. *Journal of algebra*, **103**, 141–159, 1986.

[3] S. Bosch. *Algebra*. Der Reihe *Springer-Lehrbuch*. Springer, 2009.

[4] S. Gao und G. Mullen. Dickson polynomials and irreducible polynomials over finite fields. *Journal of number theory*, **49**, 118–132, 1994.

[5] D. Hachenberger. *Finite fields: normal bases and completely free elements*. Der Reihe *Developments in Molecular and Cellular Biochemistry*. Kluwer Academic Publishers, 1997.

[6] D. Hachenberger. Asymptotic existence results for primitive completely normal elements in galois field extensions, 2015.

[7] D. Hachenberger. *Neues buch*.

[8] D. Hachenberger. Primitive complete normal bases: existence in certain 2-power extensions and lower bounds. *Discrete mathematics*, **310**, 3246–3250, 2010. Combinatorics 2008.

[9] D. Hachenberger. Primitive complete normal bases for regular extensions. *Glasgow mathematical journal*, **43**, 383–398, 03, Mai 2001.

[10] B. Hartley und T. Hawkes. *Rings, modules and linear algebra*. Der Reihe *Chapman and Hall mathematics series*. Chapman und Hall Limited, 1974.

[11] C. Karpfinger und K. Meyberg. *Algebra gruppen - ringe - korper*. Spektrum Akademischer Verlag GmbH, 2010.

[12] S. Lang. *Algebra*. Der Reihe *Graduate Texts in Mathematics*. Springer New York, 2002.

[13] H. W. Lenstra, Jr. und R. J. Schoof. Primitive normal bases for finite fields. *Mathematics of computation*, **48**, 217–231, Januar 1987.

[14] R. Lidl, G. Mullen und G. Turnwald. *Dickson polynomials*. Der Reihe *Pitman monographs and surveys in pure and applied mathematics*. Longman Scientific & Technical, 1993.

[15] R. Lidl und H. Niederreiter. *Finite fields*. (Bd. 20,Teil 1) der Reihe *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1997.

[16] I. Morgan und G. Mullen. Completely normal primitive basis generators of finite fields. *Utilitas mathematica*: 21–43, 1996.

[17] G. Mullen und D. Panario. *Handbook of finite fields*. Der Reihe *Discrete Mathematics and Its Applications*. Taylor & Francis, 2013.

[18] A. Scheerhorn. Applications of finite fields. In D. Gollmann, Herausgeber, teil Dickson Polynomials and Completely Normal Elements over Finite Fields, Seiten 47–55. Oxford University Press, Inc., New York, NY, USA, 1996.

[19] A. Scheerhorn. Dickson polynomials, completely normal polynomials and the cyclic module structure of specific extensions of finite fields. *Des. codes cryptography*, **9**, 193–202, 1996.

[20] I. A. Semaev. Construction of polynomials irreducible over a finite field with linearly independent roots. *Mathematics of the ussr-sbornik*, **63**, 507, 1989.

[21] Z. Wan. *Lectures on finite fields and galois rings*. World Scientific, 2003.