Masterarbeit
im Studiengang Mathematik

Theoretische und experimentelle Untersuchungen zu Normalbasen für Erweiterungen endlicher Körper

vorgelegt von

Stefan Hackenberg

am

Institut für Mathematik

der

Universität Augsburg

Erstprüfer

Prof. Dr. Dirk Hachenberger

Zweitprüfer

Prof. Dr. Dieter Jungnickel

Dezember 2014

Inhaltsverzeichnis

Einleitung

In der Mathematik, der Informatik und der Informationstheorie bieten *endliche Körper* ein weites Spektrum an Anwendungen. Sie sind fundamental beispielsweise in den Bereichen der Zahlentheorie, Algebraischen Geometrie, Galoistheorie, Kryptographie und Codierungstheorie. Auch wenn historisch gesehen einige Fragen, die in Verbindung mit endlichen Körpern stehen, ins 17. Jahrhundert zurückreichen, so hat sich das Studium selbiger erst am Ende des 19. Jahrhunderts als eigenständiger Bereich der Mathematik etabliert. In diese Zeit fällt auch die erste Erwähnung des Begriffs *Galois field* durch E. H. Moore (1862-1932), wobei das deutschsprachige Pendant *endlicher Körper* durch Heinrich Weber (1842-1913) geprägt wurde. Moore bezeichnete diese Galois fields durch $GF[q^n]$ für jede Primzahl q und jede positive ganze Zahl n, indem man ein irreduzibles Polynom von Grad n modulo q wähle. Sodann war er in der Lage zu beweisen, dass diese Konstruktion eindeutig ist, in dem Sinne, dass es für jede Wahl einer Primzahl q stets ein Polynom von Grad n gibt, das irreduzibel modulo q ist und das so konstruierte Galois field unabhängig von der konkreten Wahl des Polynoms ist (vgl. Gow [25, Section 1] für einen ausführlichen Überblick über den historischen Kontext).

Die vorliegende Arbeit versteht sich nicht als Einführung in die Thematik endlicher Körper, die wir hier in üblicher Notation mit $\mathbb{F}_q$ für eine Primzahlpotenz $q \geq 2$ bezeichnen wollen. Vielmehr sei ein elementares Verständnis vorausgesetzt. Die zentrale Rolle sollen *Normalbasen* und *normale Elemente* spielen, also solche Elemente u einer Erweiterung $\mathbb{F}_{q^n}$ eines endlichen Körpers $\mathbb{F}_q$, so dass $\{u, u^q, u^{q^2}, \ldots, u^{q^{n-1}}\}$ eine $\mathbb{F}_q$-Basis von $\mathbb{F}_{q^n}$ bildet. Aufgrund der einfachen Darstellbarkeit der Basiselemente sind normale Elemente auf dem Gebiet endlicher Körper von besonderem Interesse. Ihre Existenz wurde bereits von Gotthold Eisenstein (1823-1852) 1850 [6] postuliert; jedoch ohne Beweis. Der älteste vollständige Beweis ihrer Existenz geht auf Kurt Hensel (1861-1941) 1888 [17] zurück (vgl. [18, Section 2.1]). Stellt man sich nun die Frage, ob diese Eigenschaft noch über allen Zwischenkörpern $\mathbb{F}_{q^d}$ erhalten bleibt, genauer ob $u \in \mathbb{F}_{q^n}$ normal über $\mathbb{F}_{q^d}$ für alle $d \mid n$ bleibt, so stößt man auf den Begriff eines *vollständig normalen* Elements bzw. einer *vollständigen Normalbasis*. Wie bei Normalbasen konnte auch hier ein Beweis der Existenz erbracht werden, jedoch erst 1986 von Blessenohl und Johnsen [2]. Auch vollständig normalen Elementen wollen wir einen wesentlichen Teil dieser Arbeit widmen.

Wir beginnen im ersten Kapitel bei der Wiederholung grundlegender Resultate, die weniger dazu dienen sollen, das Verständnis endlicher Körper oder zyklischer Gruppen zu stärken, sondern vielmehr werden bekannte Aussagen über zyklische Gruppen und Automorphismen endlicher Körper, die im späteren Verlauf immer wieder benötigt werden, zusammengetragen. Das zweite Kapitel befasst sich mit den *Kreisteilungspolynomen*. Sie spielen bei der Betrachtung von (vollständigen) Normalbasen eine zentrale Rolle und ihr Zerfall über endlichen Körpern soll daher sehr genau studiert werden. Das daran anschließende Kapitel steht separat und liefert neben den Kreisteilungspolynomen die zweite Säule für genaue Untersuchungen von (vollständigen) Normalbasen:

Dort werden wir uns mit Moduln über Hauptidealbereichen beschäftigen und möglicherweise bekannte Resultate über die Zerlegung solcher Moduln in direkte Summen aufzeigen. Anschließend werden wir erkennen, wie sich Vektorräume als Moduln verstehen lassen und sich die Sätze über Moduln auf diesen Spezialfall anwenden lassen. Im vierten Kapitel sind wir bereit, diese beiden Säulen miteinander in Verbindung zu bringen und auf ihrer Grundlage eine Theorie zu errichten, die eine explizite Charakterisierung von normalen Elementen in Erweiterungen endlicher Körper erlaubt. Insbesondere werden wir feststellen, dass sich in gewissen Konstellationen, die wir später als *regulär* und *stark regulär* bezeichnen wollen, normale Elemente in Termen von Einheitswurzeln explizit beschreiben lassen. Das Kapitel schließt mit der Analyse zweier Artikel von Scheerhorn [26, 27]. Dort werden *Dickson-Polynome* genutzt, um die Modulstruktur von gewissen Erweiterungen endlicher Körper anzugeben und um letztlich (vollständig) normale Elemente zu konstruieren. Wir werden in dieser Arbeit erkennen, dass sich der Inhalt aus diesen Artikeln sehr gut mit Hilfe der erarbeiteten Theorie verdeutlichen lässt.

Dies schließt den primär theoretischen Teil der Arbeit ab, da sich die darauf folgenden Abschnitte mit Fragen experimenteller Natur beschäftigen. Im Zentrum des fünften Kapitels steht die Arbeit von Hachenberger aus [8], der ebenfalls durch Untersuchung der Modulstrukturen von Erweiterungen endlicher Körper eine konkrete Charakterisierung vollständig normaler Elemente möglich machen konnte. Dieses Kapitel enthält keine Beweise und ist als Überblick (ähnlich wie das dazu passende Kapitel im Handbook of Finite Fields [10]) zu verstehen, der die theoretische Grundlage für die nachfolgenden experimentellen Untersuchungen bildet. Konkret werden wir eine Implementierung für das Computeralgebrasystem `Sage` (geschrieben in `C` und `Python`) vorstellen, die sich der Aufgabe stellt, die Anzahlen primitiv normaler, vollständig normaler und primitiv vollständig normaler Elemente für gewisse Erweiterungen endlicher Körper durch Enumeration zu bestimmen. Wie sich vermuten lässt, ist dies ein offenes Problem der Forschung. Zwar kennt man konkrete Formeln für die Zahl der primitiven und die der normalen Elemente einer beliebigen Körpererweiterung, doch existieren bislang nur Abschätzungen oder Formeln für bestimmte Fälle, interessiert man sich für die Anzahlen primitiv normaler, vollständig normaler und primitiv vollständig normaler Elemente. Daher werden wir den aktuellen Stand der Forschung zu Beginn des sechsten Kapitels zusammentragen und uns dann der Aufgabe widmen eine Implementierung anzugeben, die die Theorie des fünften Kapitels bestmöglich ausnutzt, um für einige konkrete Konstellationen die Zahl der vollständig normalen und die Zahl der primitiv vollständig normalen Elemente zu bestimmen. Da die Bestimmung der Zahl der primitiv normalen Elemente offenbar ein Teilproblem der Bestimmung der Anzahl der primitiv vollständig normalen bildet, haben wir in jener Implementierung auch diese Möglichkeit hinterlegt und können auch für einige Fälle diese Werte präsentieren.

Den letzten Teil dieser Arbeit bildet eine computergestützte Suche nach primitiv vollständig normalen Elementen, denn – im Gegensatz zum primitiv normalen Fall – ist die bloße Existenz von primitiv vollständig normalen Elementen im Allgemeinen bislang ungeklärt. Hachenberger konnte diese Frage in [13] und [14] für alle *regulären* Körpererweiterungen beantworten. Seine Arbeit [9] über eine asymptotische Existenz gab nun Anlass für gewisse (fest gewählte) Erweiterungsgrade für die Bereiche, die nicht von diesem asymptotischen Resultat erfasst werden, eine computergestützte Suche nach primitiv vollständig normalen Elementen durchzuführen, um letztlich beweisen zu können, dass für jede gewisse Erweiterungsgrade n für alle Primzahlpotenzen q ein Element $u \in \mathbb{F}_{q^n}$ existiert, das primitiv und vollständig normal über $\mathbb{F}_q$ ist. Alles in allem werden wir zeigen können, dass für beliebige Primzahlpotenzen $q \geq 2$ und für jede ganze Zahl n mit $2 \leq n \leq 33$ ein primitiv vollständig normales Element in $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ existiert.

Besonders möchte ich Prof. Dr. Dirk Hachenberger danken, der durch seine Vorlesungen *Endliche Körper I* und *II* den Grundstein für das Interesse an der Untersuchung (vollständiger) Normalbasen gelegt hat, als Betreuer dieser Arbeit mir stets sehr nützliche Anregungen und Ideen lieferte und bei Fragen und Problemen immer mit einem helfenden Gedanken zur Verfügung stand.

Liste verwendeter Symbole

Allgemeines

$\mathbb{N}$	Menge der natürlichen Zahlen mit 0
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{Z}_m$	ganze Zahlen modulo m
$F[x]$	univariater Polynomring über F
$F[x]_{<k}$	Menge der univariaten Polynome über F vom Grad kleiner k
φ	Eulersche Phifunktion (Definition 1.2)
M^*	Menge M ohne das Element 0

Gruppentheoretisches

$\langle a \rangle$	von a erzeugte Gruppe
$\mathrm{ord}(u)$	gruppentheoretische Ordnung des Elements u

Ringe und Moduln

(r)	von r erzeugtes Ideal
$M^\times$	Einheiten von M
V_r	die von r annihilierten Elemente (Definition 3.18)
$\mathrm{Ann}_R(S)$	Annihilator von S in R (Definition 3.20)
$f(x) \cdot v$	Multiplikation im $K[x]$-Modul bzgl. τ (Definition 3.27)
$\mathrm{Ord}_\tau(v)$	τ-Ordnung von v (Definition 3.30)

Zahlentheoretisches

$\nu(n)$	quadratfreier Teil von n (Definition 1.3)
$\mathrm{ord}_n(q)$	multiplikative Ordnung von q modulo n (Definition 2.6)
$M_q(l \bmod m)$	$:= \{l\, q^i \bmod m : \ i \in \mathbb{N}\}$ (Definition 2.9)

$R_q(m)$	vollständiges Repräsentantensystem von $M_q(1 \bmod n)$ (Definition 2.9)
$r_q(l \bmod m)$	Länge der Bahn von $M_q(l \bmod m)$ (Definition 2.9)
$\mathrm{cl}_r(n)$	Abschluss von r in n (Definition 2.12)
$\Delta_q(m,d)$	$:= \frac{\varphi(d)\,\mathrm{ord}_m(q)}{\mathrm{ord}_{md}(q)}$ (Satz 2.20)

Endliche Körper

$\mathbb{F}_q$	Endlicher Körper mit q Elementen
$\bar{F}$	algebraischer Abschluss eines Körpers F
$\mathrm{Tr}_{E\|F}$	Spurfunktion von E nach F
$\mathrm{Nm}_{E\|F}$	Normfunktion von E nach F
$\mathrm{char}(\mathbb{F}_q)$	Charakteristik von $\mathbb{F}_q$
σ	Frobenius-Endomorphismus (Satz 1.10)
$\mathrm{Gal}(E \mid F)$	Galoisgruppe einer Körpererweiterung E über F
$K^{(n)}$	n-ter Kreisteilungskörper (Definition 2.1)
$U^{(n)}$	Menge der n-ten Einheitswurzeln (Definition 2.1)
$C^{(n)}$	Menge der primitiven n-ten Einheitswurzeln (Definition 2.3)
$\phi_q(f)$	Polynomversion der Eulerschen Phifunktion (Definition 4.4)
$\mathcal{C}_{k,t}$	verallgemeinerter Kreisteilungsmodul (Definition 5.6)
$\tau(q,k)$	τ-Teiler (Definition 5.13)

Spezielle Polynome

$\Phi_n(x)$	n-tes Kreisteilungspolynom (Definition 2.4)
$\Phi_{k,t}(x)$	verallgemeinertes Kreisteilungspolynom (Definition 5.5)
$D_n(x,a)$	Dickson-Polynom erster Art (Definition 4.24)
$E_n(x,a)$	Dickson-Polynom zweiter Art (Definition 4.24)
$f^*(x)$	reziprokes Polynom von $f(x)$ (Definition 4.35)

Spezielle Mengen

$\mathcal{S}_q$	Menge der Grade stark regulärer Erweiterungen über $\mathbb{F}_q$ (Definition 4.7)
$\mathcal{N}(q,n)$, $\mathcal{CN}(q,n)$, $\mathcal{PN}(q,n)$, $\mathcal{PCN}(q,n)$	Anzahl normaler, vollständig normaler, primitiv normaler, primitiv vollständig normaler Elemente von $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ (Definition 6.1)
$\mathcal{G}$	Menge der n, für die für alle Primzahlpotenzen q ein primitiv vollständig normales Element in $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ existiert

Kapitel 1

Grundbegriffe

Tragen wir zunächst einige grundlegende Resultate zusammen, die dem Leser sicherlich bekannt sind. Daher zitieren wir die meisten Aussagen lediglich ohne Beweis. Wir beginnen dabei bei der Gruppentheorie und insbesondere mit zyklischen Gruppen. Diese werden uns später helfen, die Untergruppe der Einheitswurzeln in Kapitel 2 zu verstehen. Im anschließenden Abschnitt rekapitulieren wir ein wenig die Galoistheorie von endlichen Körpern. Insbesondere wollen wir wiederholen, dass die Galoisgruppe endlicher Körper zyklisch ist und von einem speziellen Automorphismus erzeugt wird.

1.1 | Ein wenig Gruppentheorie

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

Satz 1.1.

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

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

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

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

Da wir später ein paar Eigenschaften benötigen werden, wiederholen wir die wohlbekannte Defini-

tion der Eulerschen Phifunktion und geben dann die wichtigsten Rechenregeln an.

Definition 1.2 (Eulersche Phifunktion).

Die Funktion

$$\begin{aligned} \varphi\colon\ \mathbb{N}^* &\to \mathbb{N}^*, \\ n &\mapsto |\{a \in \mathbb{N}\colon\ 1 \le a \le n,\ \mathrm{ggT}(a,n) = 1\}| \end{aligned}$$

heißt *Eulersche Phifunktion.*

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). *Seien $a, b \in \mathbb{N}^*$, so gilt*

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

(2) $a = \sum_{d|a} \varphi(d)$ *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 Gruppe eines endlichen Körpers einen Namen zu geben.

Satz 1.5.

Die multiplikative Gruppe eines endlichen Körpers ist zyklisch.

Beweis. [23, 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 $\mathrm{ord}(u) = q - 1$, also seine gruppentheoretische Ordnung in $\mathbb{F}_q^*$ genau der Gruppenordnung entspricht. ◢

1.2 | Automorphismen über endlichen Körpern

Satz 1.8.

Seien $F := \mathbb{F}_q$ ein endlicher Körper der Charakteristik $p \neq 0$ und $n \in \mathbb{N}^*$. Dann ist

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

ein Automorphismus auf F.

Beweis. [29, 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 Primzahlpotenz und $n \in \mathbb{N}^*$. Der Automorphismus

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

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

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

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

Beweis. [29, Theorem 7.1]. □

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 $\sigma^0, \sigma, \ldots, \sigma^{n-1}$ keine weiteren Automorphismen von $\mathbb{F}_{q^n}$ gibt, die $\mathbb{F}_q$ fixieren.

Beweis. [29, Theorem 7.3]. □

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

Satz 1.12 (Dedekindsches Lemma).

Seien K, L zwei Körper, $n \in \mathbb{N}$ und $\tau_1, \ldots, \tau_n : K \to L$ verschiedene injektive Körperhomomorphismen. Sind $c_1, \ldots, c_n \in L$ mit

$$c_1\tau_1(x) + \ldots + c_n\tau_n(x) \quad \text{für jedes } x \in K\,,$$

so gilt $c_1 = \ldots = c_n = 0$.

Beweis. [19, 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 auch relativ leicht überlegen, dass auch jedes Polynom $f(x) \in \mathbb{F}_q[x]$ vom Grad n eine Nullstelle in $\mathbb{F}_{q^n}$ besitzt. Beides fasst nachstehender Satz zusammen.

Satz 1.13.

Sei $f(x) \in \mathbb{F}_q[x]$ 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

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

Beweis. [23, Theorem 2.14]. □

Kapitel 2

Der Zerfall von $x^n - 1$ und die Kreisteilungspolynome

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 [23] und [29].

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-te Einheitswurzeln* und die Menge derer wird mit $U^{(n)}$ bezeichnet.

Satz 2.2.

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

(1) Sei $p \nmid n$. Dann ist $U^{(n)}$ 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 $U^{(n)} = U^{(m)}$ und die Nullstellen von $x^n - 1 \in K[x]$ sind gerade die Elemente in $U^{(m)}$ jedoch jeweils mit Multiplizität p^e.

Beweis. [23, Theorem 2.42]. □

Definition 2.3 (primitive Einheitswurzeln).

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

Definition 2.4 (Kreisteilungspolynom).

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

$$\Phi_n(x) := \prod_{\zeta \in C^{(n)}} (x - \zeta) \in K^{(n)}[x]$$

heißt *n-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 den Primkörper von K notiere.*

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

(2) Lässt sich per Induktion recht einfach beweisen (vgl. [23, Theorem 2.45 (ii)]). □

Definition 2.6.

Für zwei teilerfremde natürliche Zahlen q, n größer Null sei

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

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

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

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

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

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

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

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

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

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

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

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

Satz 2.8.

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

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

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

Beweis. Sei $f(x) \mid \Phi_n(x)$ ein irreduzibler Teiler über $\mathbb{F}_q$. Ist dann $\zeta \in C^{(n)}$ eine Nullstelle von $f(x)$, so sind nach Satz 1.13 auch

$$\zeta^q, \zeta^{q^2}, \ldots, \zeta^{q^{n-1}}$$

Nullstellen von $f(x)$. Jedoch sind offenbar nur $\mathrm{ord}_n(q)$ dieser verschieden und da f als irreduzibles Polynom wieder nach Satz 1.13 nur einfache Nullstellen besitzt, können wir folgern, dass $\deg f = \mathrm{ord}_n(q)$. Da $f(x)$ als beliebiger irreduzibler Teiler von $\Phi_n(x)$ gewählt wurde, folgt sofort die Behauptung, wenn man sich überlegt, dass der Grad des n-ten Kreisteilungspolynoms per Definition gerade $\varphi(n)$ ist. □

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

Definition 2.9.

Für $m, q \in \mathbb{N}$ mit $\mathrm{ggT}(m,q) = 1$ und $j \in \{0, \ldots, m-1\}$ 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 der Untergruppe $M_q(1 \bmod m)$ in $\mathbb{Z}_m$ sei mit $R_q(m)$ bezeichnet. Für $l = 1, \ldots, m-1$ bezeichne ferner $r_q(l \bmod m) := |\{lq^i : i \in \mathbb{N}\}|$ die Länge der zugehörigen Bahn.

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

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

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

Vertretersystem von Restklassen modulo 21:

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

Nun wissen wir aus Satz 2.5, dass

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

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

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

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

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

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

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

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

Nun können wir uns überlegen, ob und wie unterschiedliche Kreisteilungspolynome zusammenhängen und kommen dabei auf die bekannten Resultate, die z.B. in [8, Proposition 10.6, 10.7] zu finden sind. Um diese anzugeben, benötigen wir jedoch noch eine Definition und zitieren einige Eigenschaften.

Definition 2.12.

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

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

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

$$\mathrm{cl}_r(q^{r^n} - 1) = r^n \,\mathrm{cl}_r(q - 1)\,.$$

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

$$\mathrm{cl}_2(q^{2^n} - 1) \;=\; 2^{n-1}\,\mathrm{cl}_2(q^2 - 1)\,.$$

Beweis. [8, 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

$$\mathrm{cl}_m(q^k - 1) \;=\; k\,\mathrm{cl}_m(q - 1)\,.$$

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

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

Beweis. [8, Lemma 19.5]. □

Satz 2.15.

Seien $t, k \in \mathbb{N}^*$ und K ein Körper der Charakteristik p.

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

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

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 $k = \mathrm{cl}_k(t)\bar{t}$, so gilt

$$\psi_t^{-1}(C^{(k)}) \;=\; \bigcup_{d|\bar{t}} C^{(k\,d\,\mathrm{cl}_k(t))}$$

Nun ist offensichtlich, dass es reicht (3') zu zeigen. Dazu notiere $t_0 := \mathrm{cl}_k(t)$ 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 Definition von $\mathrm{cl}_k(t)$ gerade $\mathrm{ggT}(\bar{t}, kt_0) = 1$. Also gilt $\psi_t(C^{(kdt_0)}) \subseteq C^{(k)}$ und damit

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

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

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

wobei an Lemma 1.4 erinnert sei. Auf der anderen Seite haben wir

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

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 $\beta \in \mathbb{F}_q^*$ 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 $\beta'^p = \beta$ (vgl. Satz 1.8), also 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)

$$x^n - \beta \;=\; \prod_{d|\bar{n}} \mathrm{ggT}(x^n - \beta, \Phi_{en_0 d}(x))$$

für $n = \mathrm{cl}_e(n)\bar{n}$ und diese Zerlegung ist, wie man sich analog zum Beweis von Satz 2.8 überlegen kann, nicht trivial (vgl. [15, Proposition 5.3.5]). Damit ist (2) der Behauptung klar, so dass $x^n - \beta \mid \Phi_{ne}(x)$. Ferner zerfällt $\Phi_{ne}(x)$ nach Satz 2.8 in $\frac{\varphi(ne)}{\mathrm{ord}_{ne}(q)}$ irreduzible Faktoren von jeweils Grad $\mathrm{ord}_{ne}(q)$. Damit wird auch (3) der Behauptung augenblicklich klar. □

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

Satz 2.17.

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

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

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

(3) $\mathrm{cl}_n(q-1) \mid e$ *und* $q \equiv 1 \bmod 4$, *falls* $4 \mid n$.

Beweis. [15, Theorem 5.3.7] und [15, Corollary 5.3.8]. □

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 4.7) nennen und einsehen, dass sie eine wesentliche Rolle bei der expliziten Konstruktion von Normalbasen spielen. Nachstehender Satz hat seinen Ursprung in [8, Lemma 22.2], jedoch mit anderem Beweis.

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 $l := \mathrm{cl}_m(q-1)$, $a := \mathrm{ggT}(l, m)$ und $I_a := \{j \in \mathbb{N}^* : \ j \leq a, \ \mathrm{ggT}(j, a) = 1\}$. Ist $\zeta \in \mathbb{F}_q^*$ eine primitive a-te Einheitswurzel, so ist

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

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

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

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

Also ist auch $\nu(a) = \nu(\mathrm{ggT}(l, m)) = \nu(m)$ und damit folgt $\nu(\frac{m}{a}) \mid \nu(m) = \nu(a) \mid a$, womit auch (2) in Satz 2.16 erfüllt wäre. Da $\nu(m) \mid q-1$ ist $\mathrm{cl}_{\frac{m}{a}}(q-1) \mid m$, also auch $\mathrm{cl}_{\frac{m}{a}}(q-1) \mid a$. 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 [8, 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 $\operatorname{ggT}(m,t) = 1$. Definieren wir für $d \mid t$ ferner

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

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 $U^{(mt)}$:

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

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

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

Notiert wieder σ der Frobenius von $\mathbb{F}_q$, so sind nach Satz 1.13 $\sigma^j(\alpha)$, $j = 0, \ldots, \delta-1$ für $\delta = \operatorname{ord}_q(m)$ die Nullstellen von $f(x)$. Da $p \nmid t$ bleibt die Menge der t-ten Einheitswurzeln invariant unter σ und damit ist die Menge der Nullstellen von $f(x^t)$ gerade

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

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 $j \in \{0, \dots, \delta - 1\}$ 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 $C^{(m)}$. □

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 $\Phi_7(x)$ und $\Phi_{21}(x)$ 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 modulo 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 modulo 20 beschreiben:

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

$$\begin{aligned}\Phi_{10}(x) &= x^4 + 2x^3 + x^2 + 2x + 1 \\ &= (x - \zeta^2)(x - \zeta^6)(x - \zeta^{14})(x - \zeta^{18}).\end{aligned}$$

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

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

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

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

Kapitel 3
Moduln

3.1 | Über Moduln über Hauptidealbereichen

Nähern wir uns der Situation von Normalbasen in möglichst allgemeiner Form, so beginnt die Reise bei der Betrachtung von Moduln über Hauptidealbereichen. Dazu wiederholen wir die wichtigsten Definitionen und Aussagen. Für eine intensivere Betrachtung sei auf Standardwerke der Algebra, z.B. [20] oder [16], 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. [19, 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 $I \subseteq R$ ein Hauptideal ist, d.h. ein $r \in R$ existiert mit $I = (r)$. Dabei notiere wie üblich (r) das von r erzeugte Ideal.

Lemma 3.6. *Sei K ein Körper, so sind K und $K[x]$ Hauptidealbereiche.*

Beweis. [19, 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 jeweils nur bis auf Assoziiertheit eindeutig, 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 u_i für alle $i = 1, \dots, n$.

Satz 3.11.

Hauptidealbereiche sind faktorielle Ringe.

Beweis. [20, 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

$$M = Rx \,.$$

Bemerkung 3.14. 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. ◢

Satz 3.15.

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 ([20, Theorem 7.1]). □

Definition 3.16 (Ordnung eines Moduls).

Seien R ein Hauptidealbereich und M ein R-Modul. Falls $r \in R \setminus \{0\}$ existiert mit $rM = 0$, so heißt das bezüglich Teilbarkeit (und bis auf Multiplikation mit Einheiten) kleinste $r \in R$, das weder 0 noch eine Einheit ist, mit

$$rM \; = \; 0$$

Ordnung von M.

Nun können wir ein zentrales Resultat über Moduln mit Ordnung beweisen, das sich so auch beispielsweise in [16, Lemma 8.10] wiederfindet.

Satz 3.17 (Zerlegungssatz für Moduln mit Ordnung).

Seien R ein Hauptidealbereich und M ein nicht-trivialer R-Modul mit Ordnung 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 irreduzible Elemente zerlegen und dann diejenigen, die zueinander assoziiert sind, zusammenfassen. Ferner notieren wir $d_i = \frac{r}{p_i^{\alpha_i}}$. Kümmern wir uns nun um die Eindeutigkeit. Sei also

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

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 $i = 1, \dots, k$

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

denn $d_i M_j = 0$ für $i \neq j$ nach Voraussetzung. Wählen wir ein $i = 1, \dots, k$ 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 $(d_1, \dots, d_k) = (1)$ existeren $t_1, \dots, t_k$ mit $\sum_{i=1}^{k} t_i d_i$ und für alle $x \in M$ folgt

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

Es fehlt nur noch zu zeigen, dass diese Summe auch direkt ist. Dazu sei wieder $i \in \{1, \dots, k\}$ 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 3.18.

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

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

Bemerkung 3.19. Es ist klar, dass V_r wieder zu einem R-Modul wird, da sich V_r auch lesen lässt als der Kern des Modulhomomorphismus

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

Definition 3.20 (Annihilator).

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

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

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

Bemerkung 3.21. Man spricht in obiger Definition auch vom *Annihilator-Ideal*, da in der Tat $\mathrm{Ann}_R(S)$ ein Ideal in R ist. Insbesondere, falls $M = Rx$ ein zyklischer R-Modul über einem Hauptidealbereich R ist, so ist

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

für ein $r \in R$.

Satz 3.22.

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 ohne Einschränkung $rs' = T = sr'$ und

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

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

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

□

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

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

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 $\operatorname{Ann}_R(x)$ ist. Damit folgt die Behauptung sofort aus dem Homomorphiesatz für Moduln. □

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

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

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

Beweis. Nach Lemma 3.23 ist $Z \cong R/(p^\alpha)$. Damit stehen die Teilmoduln von Z in Bijektion zu den Teilmoduln von R (gelesen als R-Modul), die (p^α) enthalten. Solch ein Teilmodul 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 $\operatorname{Ann}_R(x) = (\lambda)$. Für $r \in R$ gilt*

$$\operatorname{Ann}_R(rx) \;=\; (\delta)$$

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

Beweis. Schreibe $r = r't$, so ist $\delta rx = \delta r' tx = r'\lambda x = 0$. Also $\delta \in \operatorname{Ann}_R(rx)$. Für die andere Inklusion sei $s \in \operatorname{Ann}_R(rx)$, also $srx = 0$. Damit ist aber $sr \in \operatorname{Ann}_R(x)$ und $\lambda \mid sr$. Mit $\lambda = \delta t$ und $sr = sr't$ folgt $\delta \mid sr'$. 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 $\operatorname{Ann}_R(x) = (a)$ und $\operatorname{Ann}_R(y) = (b)$. Sind a und b teilerfremd, so gilt $\operatorname{Ann}_R(x + y) = (ab)$.*

Beweis. Zunächst ist klar, dass $ab(x + y) = abx + aby = 0$, also $(ab) \subseteq \operatorname{Ann}_R(x + y)$. 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 3.22 (1) ist aber $V_a \cap V_b = V_1 = (0)$, also auch $z = 0$. Damit ist

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

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

3.2 | Vektorräume als Moduln

Definition 3.27 ((V,τ)).

Sei $\mathbb{K}$ ein Körper, 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) ein $\mathbb{K}[x]$-*Modul bzgl.* τ.

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

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

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

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

der Frobenius von F, so ist

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

denn: Es ist klar, dass $n = \deg \chi_\sigma$ und nach dem Satz von Cayley-Hamilton ist σ Nullstelle von χ_σ. Daher teilt μ_σ das charakteristische Polynom. Jedoch kennen wir das Minimalpolynom von σ: Nach dem Dedekindschen Lemma (Satz 1.12) ist $\mathrm{id}_E, \sigma, \ldots, \sigma^{n-1}$ linear unabhängig über E, also insbesondere über F, und $\sigma^n = \mathrm{id}_E$.

Definition 3.30 (τ-Ordnung, Teilmodul).

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

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

Sei ferner $\dim V < \infty$.

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

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

Bemerkung 3.31. Die Eindeutigkeit der τ-Ordnung wird sofort klar, wenn man sich überlegt, dass für einen Hauptidealbereich R mit $(a) = (b) \subseteq R$ gilt: Schreibe $a = bb'$ und $b = aa'$, so

gilt $a = aa'b'$, also $a'b' = 1$. Damit unterscheiden sich die Erzeuger zweier gleicher Hauptideale lediglich um eine Einheit. Wir haben jedoch $\mathbb{K}[x]^\times = \mathbb{K}^\times$. Die Forderung nach Normiertheit klärt damit abschließend die Eindeutigkeit. ◢

Bemerkung 3.32. In Notation von Abschnitt 3.1 gilt offensichtlich

$$\mathrm{Ann}_{\mathbb{K}[x]}(v) \;=\; \big(\mathrm{Ord}_\tau(v)\big)\,.$$ ◢

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

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

und bezeichne Ord_q mit *q-Ordnung.* ◢

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

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

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

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

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

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

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

Beweis. Nach dem Homomorphiesatz gilt: $\mathrm{im}\,\psi_v \cong \mathbb{K}[x]\big/\ker\psi_v$ 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 $\mathbb{K}[x]\cdot w = V$.

Satz 3.37.

Es gilt:

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

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

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

und

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

wobei die erste Teilbarkeitsrelation per Definition erfüllt ist und die zweite gerade der Satz von Cayley-Hamilton ist. Damit kommen wir zum eigentlichen 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 3.18) 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 3.17 und Lemma 3.24.

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

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

Also $g(x) = h(x)$. 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 ferner $v \in V$ ein Erzeuger von V_g. Dann ist $w = f(x) \cdot v$ für ein $f(x) \in \mathbb{K}[x]$ mit $\mathrm{ggT}(f(x), g(x)) = 1$ 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 [8, 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. $V_g = \operatorname{im} \psi_v \cong \mathbb{K}[x]/(g(x))$, 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 [8, 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 Modul 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 sind.

(4) Ist $V_g = \oplus_{i \in I} V_i$ eine Zerlegung in Teilmoduln, so existieren eine Zerlegung 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 3.17.

(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 3.22. □

Korollar 3.43. *Seien (V,τ) und $g(x) \in \mathbb{K}[x]$ wie oben. Sei ferner $g(x) = \prod_{i=1}^{k} g_i(x)^{\nu_i}$ die vollständige Faktorisierung von $g(x)$ über $\mathbb{K}[x]$. Dann ist die Menge aller Elemente in V mit τ-Ordnung $g(x)$ gleich*

$$\bigoplus_{i=1}^{k} \left(V_{g_i^{\nu_i}} \setminus V_{g_i^{\nu_i - 1}} \right)$$

Beweis. Da obige Faktorisierung vollständig ist, sind die $g_i(x)$ irreduzibel über $\mathbb{K}[x]$ für alle $i = 1, \ldots, k$. Also sind alle Elemente in $V_{g_i^{\nu_i}}$ die nicht von $g_i^{\nu_i - 1}$ anihiliert werden bereits Erzeuger von $V_{g_i^{\nu_i}}$; haben mithin τ-Ordnung $g_i^{\nu_i}$. Der Rest ist klar nach obigem Satz. □

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 $E := \mathbb{F}_{q^n}$ über 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

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

eine F-Basis von E ist. $\{\gamma(w) : \ \gamma \in \mathrm{Gal}(E \mid F)\}$ 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 dem vorhergehenden 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. □

Darüber hinaus können wir eine Formel präsentieren, die die Anzahl aller Elemente mit gegebener q-Ordnung liefert. Insbesondere gewinnen wir so eine Formel für die Anzahl aller normalen Elemente.

Definition 4.4.

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

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

Satz 4.5.

Seien $\mathbb{F}_{q^n}$ über $F := \mathbb{F}_q$ eine Erweiterung endlicher Körper und $g(x) \mid x^n - 1$ monisch mit $g(x) = \prod_{i=1}^{k} g_i(x)^{\nu_i}$ seiner vollständigen Faktorisierung über F. Dann existieren genau

$$\phi_q(g) \;=\; \prod_{i=1}^{k} \left(q^{\nu_i \deg g_i} - q^{(\nu_i - 1)\deg g_i}\right)$$

Elemente in E mit q-Ordnung $g(x)$. Insbesondere existieren

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

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

Beweis. Mit Korollar 3.43 müssen wir lediglich argumentieren, warum es $q^{\nu_i \deg g_i}$ viele Elemente von q-Ordnung $g_i^{\nu_i}$ und $q^{(\nu_i-1)\deg g_i}$ viele von q-Ordnung $g_i^{\nu_i-1}$ gibt. Dies ist jedoch klar mit Lemma 3.35: Ist $v \in E$ mit $\mathrm{Ord}_q(v) = g_i^{\nu_i}$, so gilt

$$\dim_F(F[x]\cdot v) \;=\; \deg(\mathrm{Ord}_q(v)) \;=\; \deg(g_i^{\nu_i}) \;=\; \nu_i \deg(g_i)\,.$$

Damit ist auch die Gleichheit mit $\phi_q(g)$ klar. Die Anzahl der normalen Elemente erhalten wir aus dem bekannten Zerfall von $x^n - 1$ über F nach Satz 2.5 und Satz 2.8. □

4.2 | Stark reguläre Erweiterungen

Beispiel 4.6. Wählen wir einmal $q = 7$ und $n = 9$. Also $F = \mathbb{F}_7$. Ferner wissen wir aus Kapitel 2, dass

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

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

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

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

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

Da $w \neq 0$ und per Definition 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 $\mathbb{F}_{7^9} \,|\, \mathbb{F}_7$. ◢

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 haben wir eine sehr einfache Möglichkeit, Normalbasen explizit anzugeben.

Definition 4.7 (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 4.8. *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

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

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

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

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

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

Beweis. [8, Lemma 19.6]. □

Lemma 4.9. *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. [8, Lemma 19.7]. □

Satz 4.10 ([11, Satz 8.12]).

Sei $F := \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Seien $l := \operatorname{cl}_m(q-1)$, $a := \operatorname{ggT}(l,m)$ 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 $\mathrm{cl}_m(q^m - 1) = m\,\mathrm{cl}_m(q-1) = ml$ und damit nach Lemma 4.8

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

da $\nu(ml) = \nu(m)$, wegen $l = \mathrm{cl}_m(q-1)$.

(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 4.6 betrachten wir für $i \in I_a$:

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

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

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

Nun ist $q^{\frac{m}{a}} - 1 = ck$ für ein $k \in \mathbb{N}$ mit $\mathrm{ggT}(k, ml) = 1$, 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:

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

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

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

Da i teilerfremd zu ml ist, haben wir ferner $\mathrm{ord}(u) = \mathrm{ord}(u^i)$ und können folgern, dass $\mathrm{Ord}_q(u^i) = x^{\frac{m}{a}} - \zeta^{ji}$. 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:

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

□

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

Korollar 4.11 ([11, Korollar 8.11]). *Für $F := \mathbb{F}_q$, $n \in \mathcal{S}_q$ sei $\lambda \in F^*$ eine primitive $\mathrm{cl}_n(q-1)$-te Einheitswurzel. Zu jedem $m \mid n$ seien*

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

Dann ist

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

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

Beweis. Da $\mathrm{ord}\left(\lambda^{\frac{\mathrm{cl}_n(q-1)}{a(q,m}}\right) = a(q,m)$ und für $m \mid n$ offensichtlich auch $m \in \mathcal{S}_q$, ist obige Aussage lediglich eine Anwendung von Satz 4.10. □

Satz 4.12 ([11, Korollar 8.13]).

Seien $F := \mathbb{F}_q$ ein endlicher Körper, $n \in \mathcal{S}_q$ und $L := \mathrm{cl}_n(q-1)$. Ferner sei $u \in \bar{F}$ eine primitive (nL)-te Einheitswurzel. Dann ist mit Notation aus Korollar 4.11

$$w \ := \ \sum_{m \mid n} \ \sum_{i \in I_a} u^{\frac{nL}{m \, \mathrm{cl}_m(q-1)} i}$$

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 4.10 anwenden können, aus. Es ist trivialerweise

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

und damit sind alle Voraussetzungen erfüllt. □

Bemerkung 4.13. Wie wir in Beispiel 4.6 und Satz 4.10 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 Aussagen beweisen. ◢

Lemma 4.14. *Seien die Voraussetzungen wie in Satz 4.10, also $F = \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Setze ferner $l := \mathrm{cl}_m(q-1)$, $a := \mathrm{ggT}(l,m)$. 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 Definition 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 4.10 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 Definition 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 [28] beweisen, welcher erneut von Blake, Gao und Mullin 1997 in [1] bewiesen wurde:

Satz 4.15 ([1, Theorem 2.7], [28]).

Sei $q = p^r$ eine Primzahlpotenz. Sei $n \in \mathbb{N}$ mit $\mathrm{ggT}(n,q) = 1$. 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 $\langle \theta^l \rangle_q := \mathrm{span}_{\mathbb{F}_q}\{\theta^i : \ i \in M_q(l \bmod n)\}$.

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 $p \nmid n$, $\nu(n) \mid f := \mathrm{ord}(a)$, $l := \mathrm{cl}_n(q-1) \mid f$ 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 4.14, dass $\mathrm{Ord}_q(\theta)$ 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 [8, 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 [8, Theorem 22.5] wiederzufinden ist.

Lemma 4.16. *Seien die Voraussetzungen wie in Satz 4.10, also $F = \mathbb{F}_q$ ein endlicher Körper und $m \in \mathcal{S}_q$. Setze ferner $l := \mathrm{cl}_m(q-1)$, $a := \mathrm{ggT}(l,m)$. Ist dann u eine primitive (ml)-te Einheitswurzel mit $\mathrm{Ord}_q(u) = f(x)$ für $f(x)$ einen irreduziblen monischen Teiler von $\Phi_m(x)$, so gilt für $v \in E := \mathbb{F}_{q^m}$:*

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

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

$$v \in V_f \quad \Leftrightarrow \quad v = g(x) \cdot u \text{ für ein } g(x) \in F[x]_{<\deg f} .$$

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

Bemerkung 4.17. 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 $s = \operatorname{ord}_{\nu(m)}(q)$, so erkennen wir, dass $\nu(m) \mid q^s - 1$ per Definition von $\operatorname{ord}_{\nu(m)}(q)$. Mit anderen Worten: $m \in \mathcal{S}_{q^s}$!

Definition 4.18 (regulär).

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

Satz 4.19 ([11, Satz 9.3]).

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 $s := \operatorname{ord}_{\nu(m)}(q)$, $l := \operatorname{cl}_m(q^s - 1)$, $b := b(q, m) = \operatorname{ggT}(l, m)$, $E := \mathbb{F}_{q^m}$ und $E' := \mathbb{F}_{q^{sm}}$. Ist dann u eine primitive (ml)-te Einheitswurzel, so gilt

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

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

(3) $v := \sum_{\substack{j \in R_q(m) \\ \operatorname{ggT}(j,m)=1}} u^j$ *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 4.10 folgern wir $\operatorname{ord}_{ml}(q) = sm$.

(2) Nach Satz 4.10 ist nun $g(x) := \operatorname{Ord}_{q^s}(u)$ 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 Definition von $g(x)$ ist auch $f(\sigma^s)(u) = 0$. Also $\operatorname{Ord}_q(u) \mid f(x^s)$. Es bleibt nun zu zeigen, dass $\operatorname{Ord}_q(u) = f(x^s)$: Schreibe dazu $h(x) := \operatorname{Ord}_q(u)$. Betrachten wir nun

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

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

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

wobei letzte Gleichheit aus Lemma 4.8 und Lemma 4.9 folgt. Analog sehen wir ein, dass für $\Omega_K : K[x] \to E',\ p(x) \mapsto p(\sigma^s)(u)$ wegen der Irreduzibilität 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 $N := \operatorname{im} \Omega_K$. Ferner ist klar, dass $M \subseteq N$, da $\Omega_F = \Omega_K|_{F[x]}$, womit wir durch Gleichheit der F-Dimensionen $M = N$ schließen können. Schließlich sei $\Gamma_u : F[x] \to E',\ p(x) \mapsto p(\sigma)(u)$, so ist $\ker \Gamma_u = (h(x))$ per Definition von $h(x) = \operatorname{Ord}_q(u)$. Nach Satz 3.38 (3) folgt sofort $\operatorname{im} \Gamma_u = V_h$ mit $V_h = \{w \in E' :\ h(\sigma)(w) = 0\}$ und wegen $\Omega_F = \Gamma_u \circ (p(x) \mapsto p(x^s))$ ist $N = M = \operatorname{im} \Omega_F \subseteq \operatorname{im} \Gamma_h$. Nach Satz 3.38 (4) ist V_h zyklisch, also invariant unter σ und wir können folgern, dass $N_i := \sigma^i(N) \subseteq V_h$ für alle $i \in \mathbb{N}$ gilt, wobei klar ist, dass N_i von $x^{\frac{m}{b}} - \zeta^{q^i} \in K[x]$ annihiliert wird. Für $i = 0, \dots, s-1$ sind diese jedoch über K paarweise teilerfremd und wir folgern

$$\sum_{i=0}^{s-1} N_i \ = \ \bigoplus_{i=0}^{s-1} N_i \ = \ \{w \in E' :\ f(\sigma^s)(w) = 0\} = V_{f(x^s)} \ \subseteq \ V_h\,.$$

Ergo $h(x) \mid f(x^s)$, mithin $h(x) = f(x^s)$.

(3) Nach Definition von $R_q(m)$ (vgl. auch Beispiel 2.11) bestimmt $R_q(m)$ den Zerfall von $x^m - 1$ über $\mathbb{F}_q$. Für $j \in R_q(m)$ mit $\operatorname{ggT}(j,m) = 1$ ist damit nach (2) $\operatorname{Ord}_q(u^j) = f_j(x^s)$ für einen irreduziblen Teiler $f_j(x)$ von $\Phi_m(x)$ über F. Für $i, j \in R_q(m)$ mit $\operatorname{ggT}(i,m) = \operatorname{ggT}(j,m) = 1$ und $i \neq j$ sind jene Teiler auch verschieden und damit teilerfremd. Mithin gilt die Behauptung nach Lemma 3.34 (2). □

Nun haben wir also Elemente mit q-Ordnung $\Phi_m(x^s)$ gefunden. Es stellt sich jedoch die Frage, wir daraus Elemente mit q-Ordnung $\Phi_m(x)$ „basteln“ können. Dies zeigt uns der folgende Satz, wobei die Punkte (1) und (2) aus [11, Satz 9.4] stammen:

Satz 4.20.

Seien alle Voraussetzungen und Notationen wie in Satz 4.19. Sei jedoch zusätzlich $\operatorname{ggT}(m,s) = 1$. *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* $v := \sum_{\substack{j \in R_q(m) \\ \operatorname{ggT}(j,m)=1}} u^j$, *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) $\operatorname{Tr}_{E'|E}(v)$ *hat* q*-Ordnung* $\Phi_m(x)$.

(3) $\operatorname{Ord}_q\big(\operatorname{Tr}_{E'|E}(u)\big)$ *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 Voraussetzung $\operatorname{ggT}(\bar{s}, m) = 1$ und wir sind in der Situation von Satz 2.15. Damit gilt

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

Mit konsequenter Anwendung von Lemma 3.34 folgern wir die Behauptungen:

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

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

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

folgern können. Abschließend sehen wir, dass

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

□

Auch für reguläre Erweiterungen wollen wir ein Analogon von Lemma 4.14 beweisen:

Lemma 4.21. *Seien die Voraussetzungen wie in Satz 4.19, 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 $s := \mathrm{ord}_{\nu(m)}(q)$, $l := \mathrm{cl}_m(q^s-1)$, $b := b(q,m) := \mathrm{ggT}(l,m)$, $E := \mathbb{F}_{q^m}$ und $E' = \mathbb{F}_{q^{sm}}$. Ist nun θ eine primitive (mf)-te Einheitswurzel für $l \mid f \mid q^s-1$, so ist $\mathrm{Ord}_q(\theta) = f(x^s)$ für $f(x)$ einen irreduzibler Teiler von $\Phi_m(x)$ über $F[x]$.*

Beweis. Lemma 4.14 mit dem Beweis von Satz 4.19. □

Wir können in gewisser Weise sogar eine Erweiterung von Satz 4.20 angeben und gleichzeitig die Verwendung der Spurfunktion rechtfertigen. Wir müssen lediglich dafür sorgen, dass wir ein Element desjenigen Körpers erhalten, für den wir Erzeuger (und letztendlich normale Elemente) konstruieren wollen.

Satz 4.22.

Seien die Voraussetzungen wie in Satz 4.20 und sei wiederum u eine primitive (mf)-te Einheitswurzel für $l \mid f \mid q^s-1$. Für $g(x) \in F[x]$ gilt: Ist $g(x) \cdot u \in \mathbb{F}_{q^m}^*$ 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

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

da $g(x)\cdot u$ nach Voraussetzung in $\mathbb{F}_{q^m}$ liegt. Andererseits 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 4.20), 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 $\Phi_m(x)$. □

Bemerkung 4.23. Wir hätten den Beweis von Satz 4.20 auch mit obigem Satz führen können, da

$$\mathrm{Tr}_{E'|E}(u) = (1 + x^m + \ldots + x^{m(s-1)})\cdot u$$

und nach Definition der Spurfunktion von E' auf E diese ein Element in E liefert. ◢

4.4 Normalbasen mit Dickson-Polynomen

Im vorangegangenen Abschnitt haben wir uns primitiver Einheitswurzeln, also Nullstellen der Kreisteilungspolynome, bedient, um normale Elemente in (stark) regulären Erweiterungen zu konstruieren. Im Folgenden werden wir erkennen, dass sich auch Nullstellen gewisser anderer Polynome, namentlich Dickson-Polynome, nutzen lassen, um in speziellen regulären Erweiterungen normale Elemente zu konstruieren.

Definition 4.24 (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 4.25. Über den komplexen Zahlen sind die Dickson-Polynome nahe verwandt mit den Chebyshev-Polynomen, wie man z.B. in [23, Absatz nach Corollary 7.15] nachlesen kann. Über endlichen Körpern liefern sie eine spezielle Klasse von Permutations-Polynomen (vgl. [23, Theorem 7.16], [25, Section 9.6]). ◢

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

Proposition 4.26.

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. [23, Gleichung (7.8)]. □

Proposition 4.27.

Sei $a \in \mathbb{F}_q$. 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. [22, Lemma 2.3, Lemma 2.6 (i)]. □

Proposition 4.28.

Sei $a \in \mathbb{F}_q$. Für alle $n \geq 0$ sind die Polynome $D_0(x,a), \ldots, D_n(x,a)$ linear unabhängig über $\mathbb{F}_q$.

Beweis. Dies lässt sich per Induktion und obiger Rekursionsgleichung sehr leicht einsehen: Der Induktionsanfang ist mit $n = 1$ durch $D_0(x,a) = 2$ und $D_1(x,a) = x$ erledigt. Sei also $n > 1$, so ist $D_n(x,a) = xD_{n-1}(x,a) - aD_{n-2}(x,a)$. Die Polynome $D_1(x,a), \ldots, D_{n-1}(x,a)$ sind nach Induktionsvoraussetzung jedoch linear unabhängig. □

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

Proposition 4.29.

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

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

irreduzibel genau dann, wenn 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 $\operatorname{char}\mathbb{F}_q$ 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. [7, 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 4.30. *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 $e_i = \operatorname{ord}(\beta_i)$ 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 $i=1,2$.

Ferner lässt sich spezifizieren, in welchem Körper die Nullstellen β_1 und β_2 liegen.

Korollar 4.31. *Seien $n \geq 3$ eine ungerade Zahl und $a,b \in \mathbb{F}_q^*$. Ist $D_n(x,a)-b$ irreduzibel über $\mathbb{F}_q$, so gilt:*

(1) Ist $\nu(n) \mid (q-1)$, so zerfällt $x^2-bx+a^n = (x-\beta_1)(x-\beta_2)$ über $\mathbb{F}_q$.

(2) Ist $\nu(n) \mid (q^2-1)$, aber $\nu(n) \nmid (q-1)$, so ist x^2-bx+a^n über $\mathbb{F}_q$ irreduzibel.

Beweis. Seien β_1 und β_2 die Nullstellen von x^2-bx+a^n mit $\operatorname{ord}(\beta_i)=e_i$ für $i=1,2$, so ist $\nu(n) \mid e_i$ nach Korollar 4.30, aber $\nu(n) \nmid \frac{q^2-1}{e_i}$ für $i=1,2$.

(1) Wäre x^2-bx+a^n über $\mathbb{F}_q$ irreduzibel und $\beta \in \mathbb{F}_{q^2} \setminus \mathbb{F}_q$ eine Nullstelle, so ist $\operatorname{ord}(\beta) \mid q^2-1$ aber $\operatorname{ord}(\beta) \nmid q-1$, im Widerspruch zu $\nu(n) \mid q-1$.

(2) Lägen β_1, β_2 in $\mathbb{F}_q$, so folgte $e_i \mid q-1$ im Widerspruch zu $\nu(n) \mid e_i$ und $\nu(n) \nmid q-1$. □

Scheerhorn zeigt in [26, 27], dass sich Dickson-Polynome eignen, um Normalbasen zu beschreiben. Im weiteren Verlauf werden wir sogar sehen, dass sich 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 4.32 ([27, Theorem 2]).

Seien $n \geq 3$ ein Produkt ungerader Primzahlen von $(q+1)$ und $a, b \in \mathbb{F}_q =: F$, sodass $D_n(x,a) - b \in F[x]$ 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

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

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

Satz 4.33 ([27, Theorem 3]).

Seien $n \geq 3$ ein Produkt ungerader Primzahlen von $(q-1)$ und $a, b \in \mathbb{F}_q =: F$, sodass $D_n(x,a) - b \in F[x]$ 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 $l \in R_q(n) \setminus \{0\}$

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

In [26, 27] werden die Beweise dieser Resultate ohne Zuhilfenahme der 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 Punkt 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 4.32). 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 $K := \mathbb{F}_{q^2}$ (vgl. Korollar 4.31). 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 4.15 und Lemma 4.21 (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 4.22 gerade

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

für einen monischen, irreduziblen Teiler $f_{1,1}(x)$ von $\Phi_n(x)$ (wieder in Notation von Satz 2.20), denn mit Proposition 4.26 sehen wir, dass

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

Also liegt $\theta + a\theta^{-1}$ in E, sein Minimalpolynom über F ist $D_n(x,a) - b$ und wir setzen ohne Einschränkung $\gamma := \theta + a\theta^{-1}$. Die weiteren irreduziblen Teiler von $\Phi_n(x)$ erhalten wir analog zu Satz 4.19: Für $l \in R_q(n) \setminus \{0\}$ mit $\mathrm{ggT}(l,n) = 1$ ist $\mathrm{Ord}_q(\theta^l) = g(x^s)$ für einen irreduziblen monischen Teiler $g(x)$ von $\Phi_n(x)$ mit $f(x) \neq g(x)$. Nun vollführen wir erneut obigen Kniff und erhalten

$$\mathrm{Ord}_q(D_l(\theta + a\theta^{-1}, a)) \;=\; \mathrm{Ord}_q(\theta^l + a^l\theta^{-l}) \;=\; \mathrm{Ord}_q((1+a^lx^n)\cdot\theta^l) \;=\; g_{1,1}(x)$$

mit $\mathrm{ggT}(f_{1,1}(x), g_{1,1}(x)) = 1$ nach Satz 2.20 (2). Auf diese Weise lassen sich für jeden irreduziblen Teilmodul von V_{Φ_n} Erzeuger angeben:

$$V_{\Phi_n} \;=\; \bigoplus_{\substack{l \in R_q(n) \\ \mathrm{ggT}(l,n)=1}} \big(F[x]\cdot D_l(\theta + a\theta^{-1}, a)\big) \;=\; \bigoplus_{\substack{l \in R_q(n) \\ \mathrm{ggT}(l,n)=1}} \big(F[x]\cdot(\theta^l + a^l\theta^{-l})\big)\,.$$

Für alle $i \in M_q(l \mod n)$ erhalten wir selbstverständlich immer Erzeuger des gleichen irreduziblen Teilmoduls: Für alle $i \in M_q(l \mod n)$ für ein festes $l \in R_q(n)$ mit $\mathrm{ggT}(l,n) = 1$ gilt: $\mathrm{Ord}_q(\theta^i) = h(x^s)$ für einen irreduziblen Teiler $h(x)$ von $\Phi_n(x)$. (Genauer gesagt haben wir im Beweis von Satz 4.19 gesehen, dass sich die q^s-Ordnungen von θ^i und θ^j für $i,j \in M_q(l \mod n)$ für $i \neq j$ unterscheiden; jedoch nicht ihre q-Ordnungen.)

Alle weiteren Teiler $\Phi_m(x)$ von $x^n - 1$ für $m \mid n$ decken wir mit obiger Argumentation analog ab: Für $l \in R_q(n)$ mit $r_q(l \mod n) = \mathrm{ord}_m(q)$ ist $\mathrm{Ord}_q(\theta^l + a^l\theta^{-l})$ ein irreduzibler monischer Teiler von $\Phi_m(x)$.

Letztlich haben wir bisher nur Erzeuger der irreduziblen Teilmoduln im Sinne der Modulstruktur angegeben, jedoch keine F-Basis wie gefordert. Dazu müssen wir uns nur noch kurz überlegen, dass

$$\{1\} \cup \{D_i(\theta + a\theta^{-1}, a) :\; i = 1, \ldots, n-1\}$$

eine F-Basis von $\mathbb{F}_{q^n}$ ist, da alle $D_i(x,a)$ für $i \geq 1$ linear unabhängig über F nach Proposition 4.28 sind. Diese Basis lässt sich nach Erzeugern wie oben gezeigt partitionieren. □

Beweis (von Satz 4.33). Im Grunde brauchen wir nichts zu zeigen, wenn wir uns überlegen, dass hier n stark regulär ist! Nach Lemma 4.14 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 $l \in R_q(n)$ erzeugt. Wie im Beweis von Satz 4.32 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 4.32 $\theta + a\theta^{-1}$ 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 4.20 erkannt haben, dass die Spurfunktion zur Konstruktion von normalen Elementen in regulären Erweiterungen zentral ist. Insbesondere erlaubt es die Theorie des vorherigen Abschnitts, die beiden Sätze auf den Fall $\nu(n) \mid q^2 - 1$ zu erweitern.

Satz 4.34.

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 ist mit Nullstelle $\gamma \in E := \mathbb{F}_{q^n}$, so gilt:

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

(2) Ist $\nu(n) \nmid q-1$, 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 $\mathrm{Ord}_q(D_l(\gamma,a))$ ein irreduzibler monischer 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 $\nu(n) \nmid q-1$ oder $a^l \neq -1$ für alle $l \in R_q(n)$.

Beweis. Für $\nu(n) \nmid q-1$ sehen wir, dass weiterhin $n \in \mathcal{S}_{q^2}$ und damit der Beweis von Satz 4.32 auch hier gültig ist. 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

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

□

4.5 | Normale und vollständig normale Polynome mit Dickson-Polynomen

Mit Hilfe dieser Resultate lassen sich nun relativ einfach vollständig normale Polynome angeben, wie sie in [27, Section 3] und [26, Section 4] zu finden sind. Ihre Beweise beinhalten keine besondere Beachtung der Modulstrukturen und sollen daher hier nur nachvollzogen werden. Wie im vorhergehenden Abschnitt können wir den Fall $\nu(n) \mid q+1$ wieder erweitern. Zunächst brauchen wir jedoch eine Transformation eines Polynoms, welches gerade die Nullstellen desselbigen invertiert.

Definition 4.35 (reziprokes Polynom).

Sei $f(x) \in \mathbb{K}[x]$ ein Polynom über einem Körper $\mathbb{K}$ mit $f(0) \neq 0$. Dann heißt

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

reziprokes Polynom von $f(x)$.

Bemerkung 4.36. Man bemerke, dass das reziproke Polynom stets normiert ist. ◢

Bemerkung 4.37. Ist u eine Nullstelle von $f(x)$, so ist u^{-1} eine Nullstelle von $f^*(x)$, wie man sofort erkennen kann. ◢

Satz 4.38.

Seien $n \geq 3$ ein Produkt aus ungeraden Primteilern von q^2-1 mit $\nu(n) \nmid q-1$ 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, sodass $E_{n-1-i}(t,a) \neq 0$.

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 4.32 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 $\{1\} \cup \{\theta^l + a^l\theta^{-l} : \ l = 1,\dots,n-1\}$ 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 [27, 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 Behauptung des Satzes folgt dann sofort mit Satz 4.32 (oder Satz 4.34). □

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

Satz 4.39.

Seien $n \geq 3$ ein Produkt aus ungeraden Primteilern von q^2-1 mit $\nu(n) \nmid q-1$ 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

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

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, sodass $E_{\frac{n}{d}-1-i}(t,a) \neq 0$.

Beweis. Wie man sich leicht überlegt, ist $\theta^{\frac{n}{d}} + a^{\frac{n}{d}}\theta^{-\frac{n}{d}} \in \mathbb{F}_{q^d}$ 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 4.38 bzw. Satz 4.39 erfüllt sind. Diese werden wir hier nicht beweisen und der Leser sei auf [27] verwiesen. Erneut sind wir in der Lage, die Korollare für den erweiterten Fall $\nu(n) \mid q^2 - 1$ mit $\nu(n) \nmid q - 1$ zu formulieren.

Korollar 4.40. *Sei $n \geq 3$ ein Produkt ungerader Primfaktoren von $q^2 - 1$ mit $\nu(n) \nmid q - 1$ und sei $D_n(x, a) - b$ irreduzibel über $\mathbb{F}_q$. Dann ist*

$$\big(D_n(x, 1) - b\big)^*$$

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

Beweis. [27, Corollary 1] mit Satz 4.39. □

Korollar 4.41. *Sei $n \geq 3$ ein Produkt ungerader Primfaktoren von $q^2 - 1$ mit $\nu(n) \nmid q - 1$ und sei $D_n(x, a) - b$ irreduzibel über $\mathbb{F}_q$. Ist ferner $n \not\equiv 0 \bmod 3$, so ist*

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

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

Beweis. [27, Corollary 2] mit Satz 4.39. □

Korollar 4.42. *Sei $n \geq 3$ ein Produkt ungerader Primfaktoren von $q^2 - 1$ mit $\nu(n) \nmid q - 1$ und sei $D_n(x, a) - b$ irreduzibel über $\mathbb{F}_q$. Dann ist*

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

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

Beweis. [27, Corollary 3] mit Satz 4.39. □

Kapitel 5

Vollständige Normalbasen

In den vorherigen Kapiteln haben wir einige Resultate zu Normalbasen kennengelernt. 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 $E \mid K \mid F$ und wir wollen 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 man es für normale Elemente aus Korollar 4.3 kennt. Hachenberger konnte in [8] 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 [10]. 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 besonderer Fall würde natürlich auftreten, wenn bereits alle normalen Elemente einer 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*[1].

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.

[1] Im englischsprachigen Raum wird dieser Begriff *completely basic* genannt.

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 $\mathrm{ord}_{r'}(q)$.

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

Beweis. [8, 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 [8, 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 ein 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 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 zur Charakteristik von 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 $\frac{kt}{\nu(k)}$.

Definition 5.7 (vollständiger Erzeuger).

Sei $\mathcal{C}_{k,t}$ ein verallgemeinerter Kreisteilungsmodul über $\mathbb{F}_q$. $w \in \bar{\mathbb{F}}_q$ 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 $\mathbb{F}_q$. $\Delta \subseteq \mathbb{F}_q[x]$ heißt eine *Zerlegung von* $\Phi_{k,t}$ *in verallgemeinerte Kreisteilungspolynome über* $\mathbb{F}_q$, 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 $\mathbb{F}_q$. Dann heißt Δ *verträgliche Zerlegung*, falls gilt: Für jedes $(l,s) \in i(\Delta)$ sei $w_{l,s} \in \bar{\mathbb{F}}_q$ ein vollständiger Erzeuger von $\mathcal{C}_{l,s}$ über $\mathbb{F}_q$, so ist

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

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

Nun können wir einen zentralen Satz formulieren, der eine passende Zerlegung eines erweiterten Kreisteilungspolynoms herstellt, sodass 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 bis auf das Nullelement 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 $a = \max\{b \in \mathbb{N} : r^b \mid t\}$ und $t = t'p^b$ für $\operatorname{ggT}(t', p) = 1$.

Beweis. [8, Decomposition Theorem, Section 19]. □

Sicherlich kann man sich nun fragen, in welchen Fällen die kanonische Zerlegung eines erweiterten Kreisteilungspolynoms in Kreisteilungspolynome noch verträglich ist. Nach [8, Theorem 19.10] ist die kanonische Zerlegung von $\Phi_{k,t}(x)^\pi$ für π eine Potenz der Charakteristik verträglich über $\mathbb{F}_q$, falls $\operatorname{ord}_{\nu(kt')}(q)$ und t' teilerfremd sind, wobei wieder $t = t'p^b$ mit $\operatorname{ggT}(t', p) = 1$ für $p = \operatorname{char}\mathbb{F}_q$. Dies motiviert dazu, 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 einem endlichen Körper $\mathbb{F}_q$ der Charakteristik p, falls $\operatorname{ord}_{\nu(kt')}(q)$ und kt teilerfremd sind für $t = t'p^b$ mit $\operatorname{ggT}(t', p) = 1$.

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 einem endlichen Körper $\mathbb{F}_q$ von Charakteristik p. Schreibe

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

wobei $\pi(k)$ die Menge der Primteiler von k bezeichne. 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 $u \in \bar{\mathbb{F}}_q$ genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k,p^b}$, falls

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

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

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

Beweis. [8, 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 existieren. 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 asymptotischen 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 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 nähern, um in den darauffolgenden gezielte experimentelle Untersuchungen auf Grundlage der theoretischen Resultate, die über (vollständig) normale Elemente im bisherigen Verlauf erarbeitet wurden, durchzuführen.

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. [23, 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 [21] nach den Vorarbeiten von Carlitz [4] und Davenport [5].

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. [21]. □

Bleibt also nur noch die Frage nach der Existenz primitiver vollständig normaler Elemente. Auch wenn Hachenberger 2001 [13] und 2014 [9] 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. Dann existiert ein primitives Element in $\mathbb{F}_{q^n}$, das vollständig normal über $\mathbb{F}_q$ ist.

Beweis. [13, Theorem 1.4] und [14]. □

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. [9, 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 der Wiederholung der Formel für normale Elemente aus Satz 4.5.

Satz 6.16.

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

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

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

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 4.5 beantwortet wird. Ferner war Hachenberger in [8, Section 21] in der Lage diese

Frage für reguläre Kreisteilungsmoduln (und damit für reguläre Erweiterungen) zu beantworten:

Satz 6.17.

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. [8, Proposition 21.1, Proposition 21.2]. □

Darüber hinaus lässt sich so eine Abschätzung für die Anzahl von vollständig normalen Elementen einer regulären Körpererweiterung ableiten.

Satz 6.18.

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. [8, 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 Abschätzun-

gen.

Satz 6.19.

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. [12]. □

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 [24] haben bereits 1996 für (q,n) aus $\{(2,2),\ldots,(2,18),\ (3,2),\ldots,(3,12),\ (4,2),\ldots,(4,9),\ (5,2),\ldots,(5,8),\ (7,2),\ldots,(7,6),\ (8,2),\ldots,(8,5),\ (9,2),\ldots,(9,5)\}$ 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 können, so müssen wir feststellen, dass bei den 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 [24] die theoretischen Überlegungen aus Kapitel 5 gänzlich unbeachtet blieben. Wir wollen diese 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

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

die `Pari`-Bibliothek stets mit Ganzzahlen beliebiger Größe. Deren Arithmetik ist selbstredend aufwendiger und langsamer, als maschineninterne `Integer`-Arithmetik. Daher haben wir uns entschlossen eigene `C`-Bibliotheken anzulegen, die auf einfacher (jedoch begrenzter) `Integer`-Arithmetik basieren.

6.2.1 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, monisch 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 werden.

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`

```
1  /**
2   * Finite Field Element.
3   *
4   * !! idcs must be in desc order !!
5   *
6   * Uses int arrays, i.e. you must not consider
7   * PrimeFields of order p with (p-1)*(p-1) > INT_MAX
8   */
9  struct FFElem{
10     int *el;
11     int *idcs;
12     int len;
13 };
```

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

Invariante 6.20.

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

Invariante 6.21.

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

Invariante 6.20 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 6.22. Wollen wir das Element

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

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[]) {8, 6, 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 und ausgegraut. 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[]) {8, 6, 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* (d.h. Verwaltung nicht mehr genutzter Variablen) auskommt, muss man selbst für die entsprechende Speicherverwaltung sorgen. Dies erleichtern die Funktionen `mallocFFElem` und `freeFFElem`.

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

```
1 inline struct FFElem *mallocFFElem(int m){
2     struct FFElem *ff = malloc(sizeof(struct FFElem));
3     ff->el = malloc(m*sizeof(int));
4     ff->idcs = malloc(m*sizeof(int));
5     ff->len = 0;
6     return ff;
7 }
```

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

```
1 inline void freeFFElem(struct FFElem *ff){
2     free(ff->el);
3     free(ff->idcs);
4     free(ff);
5 }
```

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

```
1 /**
2  * Copies the content of ff1 into ff2
3  *
4  * !! ff2 must be malloced!
5  */
6 inline void copyFFElem(struct FFElem *ff1, struct FFElem *ff2){
7     if(ff1 == ff2) return;
8     int i;
9     for(i=0;i < ff1->len;i++){
10        ff2->idcs[i] = ff1->idcs[i];
11        ff2->el[ ff1->idcs[i] ] = ff1->el[ ff1->idcs[i] ];
12    }
13    ff2->len = ff1->len;
14 }
```

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 vergleichsweise langsam ist. Insbesondere bei kleinen Primzahlen (die hier betrachteten Primzahlen waren kleiner gleich 43) hat sich das Anlegen einer Additions- und einer Multiplikationstabelle bewährt. Dies 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 6.23. 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 6.24. 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 Indizes, ist die Addition nicht lediglich gegeben durch komponentenweise Betrachtung, sondern erfordert etwas mehr Aufwand.

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

```
1  /**
2   * Adds two FFElems.
3   *
4   * !! ff1 may be same as ret !!
5   * !! ff2 must not be same as ret !!
6   */
7  inline void addFFElem(struct FFElem *ff1, struct FFElem *ff2,
8          struct FFElem *ret,
9          int *tmp,
10         int *multTable, int *addTable){
11     int i=0,j=0,k=0, i2;
12     bool end = false;
13     //handle trivial cases
14     if(ff1->len == 0){
15         copyFFElem(ff2,ret);
16         return;
17     }
18     if(ff2->len == 0){
19         copyFFElem(ff1,ret);
20         return;
21     }
22     copyArray(ff1->idcs,tmp,ff1->len);
23     while( end == false ){
24         while( tmp[i] != ff2->idcs[j] ){
25             if( tmp[i] > ff2->idcs[j] ){
26                 ret->el[ tmp[i] ] = ff1->el[ tmp[i] ];
27                 ret->idcs[k] = tmp[i];
28                 i++; k++;
29             }else if( tmp[i] < ff2->idcs[j] ){
30                 ret->el[ ff2->idcs[j] ] = ff2->el[ ff2->idcs[j] ];
31                 ret->idcs[k] = ff2->idcs[j];
```

```
32                  j++; k++;
33              }
34              if(i == ff1->len || j == ff2->len){
35                  end = true;
36                  break;
37              }
38
39          }
40          if(end == true) break;
41          //tmp[i] == ff2->idcs[j]
42          i2 = tmp[i];
43          ret->el[i2] = addTable[ ff1->el[i2] + ff2->el[i2] ];
44          if(ret->el[i2] != 0){
45              ret->idcs[k] = i2;
46              k++;
47          }
48          i++; j++;
49          if(i == ff1->len || j == ff2->len) end = true;
50      }
51      //add rest of ff1 or ff2
52      if(i != ff1->len ){
53          while(i<ff1->len){
54              ret->el[ tmp[i] ] = ff1->el[ tmp[i] ];
55              ret->idcs[k] = tmp[i];
56              i++; k++;
57          }
58      }else if(j != ff2->len){
59          while(j<ff2->len){
60              ret->el[ ff2->idcs[j] ] = ff2->el[ ff2->idcs[j] ];
61              ret->idcs[k] = ff2->idcs[j];
62              j++; k++;
63          }
64      }
65      ret->len = k;
66  }
```

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 sich das Anlegen unnötiger Hilfs-`FFElem`s. Wie man schnell einsieht, werden jeweils nur die beiden Indexarrays durchlaufen und lediglich wenn der Wert der jeweiligen Stellen übereinstimmt, 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`

```
1 /**
```

```
2  * Multiplies two FFElems and reduces the result by mipo.
3  *
4  * !! tmp must have at least length m!
5  * !! ret must be malloced!
6  */
7  inline void multiplyFFElem(struct FFElem *ff1, struct FFElem *ff2,
8         struct FFElem *ret,
9         struct FFElem *mipo, int *tmp, int m,
10        int *multTable, int *addTable){
11     /*
12      * catch trivial cases
13      */
14     if(ff1->len == 0 || ff2->len == 0){
15         ret->len = 0;
16         return;
17     }
18     if(ff1->len == 1 && ff1->idcs[0] == 0 && ff1->el[0] == 1){
19         copyFFElem(ff2, ret);
20         return;
21     }
22     if(ff2->len == 1 && ff2->idcs[0] == 0 && ff2->el[0] == 1){
23         copyFFElem(ff1, ret);
24         return;
25     }
26
27     /*
28      * Do multiplication
29      */
30     int maxlen = ff1->idcs[0] + ff2->idcs[0] + 1;
31     int i,j,i2,j2,k;
32     int max2 = maxlen;
33     if( maxlen > m ){
34         max2 = m;
35         initPoly(tmp,maxlen-m);
36     }
37     initPoly(ret->el,max2);
38     //multiply
39     for(i=0;i<(ff1->len);i++){
40         for(j=0;j<(ff2->len);j++){
41             i2 = ff1->idcs[i];
42             j2 = ff2->idcs[j];
43             k = i2+j2;
44             if(k<m){
45                 ret->el[k] = addTable[ ret->el[k] +
46                     multTable[ ff1->el[i2] * ff2->el[j2] ] ];
47             }else{
48                 tmp[k-m] = addTable[ tmp[k-m] +
49                     multTable[ ff1->el[i2] * ff2->el[j2] ] ];
50             }
51         }
52     }
53
54     /*
55      * Reduce mod mipo
```

```
56      */
57     if(maxlen > m){
58         int quo;
59         for(i=maxlen-m-1;i>=0;i--){
60             quo = tmp[i];
61             if(quo == 0) continue;
62             for(j=0;j<(mipo->len); j++){
63                 j2 = mipo->idcs[j];
64                 k = i+j2;
65                 if(k>=m){
66                     tmp[k-m] = addTable[ tmp[k-m] -
67                         multTable[ mipo->el[j2]*quo ] ];
68                 }else{
69                     ret->el[k] = addTable[ ret->el[k] -
70                         multTable[ mipo->el[j2]*quo ] ];
71                 }
72             }
73         }
74     }
75
76     /*
77      * Recalc indices
78      */
79     i2 = 0;
80     for(i=max2-1;i>=0;i--){
81         if(ret->el[i] != 0){
82             ret->idcs[i2] = i;
83             i2++;
84         }
85     }
86     ret->len = i2;
87 }
```

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 um Summen von Produkten von Koeffizienten der beiden Faktoren handelt. Daher muss nach der Reduktion modulo des Minimalpolynoms eine Neuberechnung des Index-Arrays 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`

```
1 /**
2  * Squares an FFElem
```

```
3  *
4  * !! ff is not modified !!
5  * !! tmp must have at least length m !!
6  */
7  inline void squareFFElem(struct FFElem *ff, struct FFElem *mipo,
8         struct FFElem *ret, int *tmp, int m,
9         int *multTable, int *addTable){
10    /*
11     * catch trivial cases
12     */
13    if(ff->len == 0){
14        copyFFElem(ff,ret);
15        return;
16    }
17    if(ff->len == 1 && ff->idcs[0] == 0 && ff->el[0] == 1){
18        copyFFElem(ff,ret);
19        return;
20    }
21
22    /*
23     * Do multiplication
24     */
25    int maxlen = 2*ff->idcs[0] + 1;
26    int i,j,i2,j2,k;
27    int max2 = maxlen;
28    if( maxlen > m ){
29        max2 = m;
30        initPoly(tmp,maxlen-m);
31    }
32    initPoly(ret->el,max2);
33    for(i=0;i<(ff->len);i++){
34        // same index must be squared
35        i2 = ff->idcs[i];
36        k = 2*i2;
37        if(k<m){
38            ret->el[k] = addTable[ ret->el[k] +
39                multTable[ ff->el[i2]*ff->el[i2] ] ];
40        }else{
41            tmp[k-m] = addTable[ tmp[k-m] +
42                multTable[ ff->el[i2]*ff->el[i2] ] ];
43        }
44        // other indices only multipied and doubled
45        for(j=i+1;j<(ff->len);j++){
46            i2 = ff->idcs[i];
47            j2 = ff->idcs[j];
48            k = i2+j2;
49            if(k<m){
50                ret->el[k] = addTable[ ret->el[k] +
51                    multTable[ 2 * multTable[ ff->el[i2] * ff->el[j2] ] ] ];
52            }else{
53                tmp[k-m] = addTable[ tmp[k-m] +
54                    multTable[ 2 * multTable[ ff->el[i2] * ff->el[j2] ] ]];
55            }
56        }
```

```
57      }
58      /*
59       * Reduce mod mipo
60       */
61      if(maxlen > m){
62          int quo;
63          for(i=maxlen-m-1;i>=0;i--){
64              quo = tmp[i];
65              if(quo == 0) continue;
66              for(j=0;j<(mipo->len); j++){
67                  j2 = mipo->idcs[j];
68                  k = i+j2;
69                  if(k>=m){
70                      tmp[k-m] = addTable[ tmp[k-m] -
71                          multTable[ mipo->el[j2]*quo ] ];
72                  }else{
73                      ret->el[k] = addTable[ ret->el[k] -
74                          multTable[ mipo->el[j2]*quo ] ];
75                  }
76              }
77          }
78      }
79
80      /*
81       * Recalc indices
82       */
83      i2 = 0;
84      for(i=max2-1;i>=0;i--){
85          if(ret->el[i] != 0){
86              ret->idcs[i2] = i;
87              i2++;
88          }
89      }
90      ret->len = i2;
91  }
```

6.2.3 Matrizen und Polynome über endlichen Körpern

Matrizen und Matrixmultiplikation

Nach Satz 1.8 ist das Potenzieren mit der Charakteristik in endlichen Körpern eine lineare Abbildung. Dies wollen wir Nutzen und haben daher eine Darstellung für Matrizen über endlichen Körpern – naheliegenderweise ein Array aus `FFElems` – implementiert.

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

```
1  /**
2   * Matrix multiplication
3   *
4   * !! tmp must have at least length m!
5   */
6  inline void matmul(struct FFElem **mat, struct FFElem *ff,
```

```
7         struct FFElem *ret,
8         int m, int *multTable, int *addTable){
9     int i,j,i2, row;
10    bool end;
11    for(row=0;row<m;row++){
12        ret->el[row] = 0;
13        i=0; j=0;
14        end = false;
15        while(end == false){
16            while(ff->idcs[i] != mat[row]->idcs[j]){
17                if(ff->idcs[i] > mat[row]->idcs[j]) i++;
18                else if(ff->idcs[i] < mat[row]->idcs[j]) j++;
19                if(i == ff->len || j == mat[row]->len){
20                    end = true;
21                    break;
22                }
23            }
24            if(end == true) break;
25            i2 = ff->idcs[i]; // == mat[row]->idcs[j]
26            ret->el[row] = addTable[ ret->el[row]
27                + multTable[ mat[row]->el[i2]*ff->el[i2] ] ];
28            i++;
29            j++;
30            if(i==ff->len || j==mat[row]->len) end = true;
31        }
32    }
33    i2 = 0;
34    for(i=m-1;i>=0;i--){
35        if(ret->el[i] != 0){
36            ret->idcs[i2] = i;
37            i2++;
38        }
39    }
40    ret->len = i2;
41 }
```

Hier wird – anders als bei der Addition – nur nach den gemeinsamen Indizes gesucht (alle anderen Produkte sind schließlich 0). Invariante 6.20 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`

```
1 inline void freeFFElemMatrix(struct FFElem **mat, int len){
2     if(mat==0) return;
3     int i;
4     for(i=0;i<len;i++) freeFFElem(mat[i]);
5     free(mat);
6 }
```

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`

```
1 struct FFPoly{
2     struct FFElem **poly;
3     int lenPoly;
4 };
```

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

```
1 inline struct FFPoly *mallocFFPoly(int m, int lenPoly){
2     struct FFPoly *poly = malloc(lenPoly*sizeof(struct FFElem*));
3     poly->lenPoly = lenPoly;
4     int i;
5     for(i=0;i<lenPoly;i++) poly->poly[i] = mallocFFElem(m);
6     return poly;
7 }
```

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

```
1 inline void freeFFPoly(struct FFPoly *poly){
2     int i;
3     for(i=0;i<poly->lenPoly;i++) freeFFElem(poly->poly[i]);
4     free(poly->poly);
5     free(poly);
6 }
```

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

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

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 3 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;
        }
```

```
27          matmul(matCharac+lenCurGap*m, ff, ffTmp, m, multTable,addTable);
28          ffSwitch = ff; ff = ffTmp; ffTmp = ffSwitch;
29          lenCurGap = 0;
30      }
31      copyFFElem(ffRetInt,ret);
32  }
```

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

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

```
1  /**
2   * Square and multiply
3   *
4   * !! ff is modified !!
5   */
6  inline void powerFFElemSqM(struct FFElem *ff, struct FFElem *mipo,
7          struct FFElem *ret,
8          int m, int *power, int powerLen,
9          int *tmp, struct FFElem *ffTmp,
10         int *multTable, int *addTable){
11     int i,j,k;
12     int lenCurGap = 0;
13     struct FFElem *ffSwitch = 0;
14     struct FFElem *ffRetInt = ret;
15     // init ret to 1
16     ffRetInt->el[0] = 1; ffRetInt->idcs[0] = 0; ffRetInt->len = 1;
17     for(j=powerLen-1;j>=0;j--){
18         if(power[j] == 1){
19             multiplyFFElem(ffRetInt,ff,ffTmp, mipo,tmp,m,multTable,addTable);
20             //switch ffTmp and ffRetInt
21             ffSwitch = ffRetInt; ffRetInt = ffTmp; ffTmp = ffSwitch;
22         }
23         if(j>0){
24             squareFFElem(ff,mipo,ffTmp,tmp,m,multTable,addTable);
25             //switch ffTmp and ff
26             ffSwitch = ff; ff = ffTmp; ffTmp = ffSwitch;
27         }
28     }
29     copyFFElem(ffRetInt,ret);
30 }
```

6.3.2 Primitivitätstest

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

Lemma 6.26. *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 $i = 1, \ldots, r$

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

Dann gilt: u ist primitiv genau dann, wenn

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

Beweis. Klar. □

Es bleibt jedoch immer noch offen diese r Potenzierungen möglichst gut zu organisieren. Dazu formulieren wir folgendes Lemma, das den Aufwand der Berechnung in vielen Fällen deutlich verringert hat.

Lemma 6.27. *Sei $q-1 = p_1^{\nu_1} \cdot \ldots \cdot p_r^{\nu_r}$ die absteigend sortierte Primfaktorzerlegung von $q-1$, d.h. $p_1 > p_2 > \ldots > p_r$. Notiere*

- $\bar{p}_i := \frac{q-1}{p_i}$,
- $d := \mathrm{ggT}\{\bar{p}_i : \; i = 1, \ldots, r\}, \quad d' := \mathrm{ggT}\{\frac{\bar{p}_i}{d} : \; i = 2, \ldots, r\}$,
- $v := u^d, \quad w := v^{d'}$,
- $\bar{n}_1 := \frac{\bar{p}_1}{d}, \quad \bar{n}_i := \frac{\bar{p}_i}{d\,d'}$ *für* $i = 2, \ldots, r$,
- $u_2 := w^{\bar{n}_2}$ *und* $u_i := w^{\bar{n}_i - \bar{n}_{i-1}}$ *für* $i = 3, \ldots, r$.

Es gilt: $u \in \mathbb{F}_q$ ist genau dann nicht primitiv, falls eine der nachstehenden Bedingungen erfüllt ist:

(1) $v \;=\; 1$.

(2) $v^{\bar{n}_1} \;=\; 1$.

(3) $w \;=\; 1$.

(4) $u_2 \;=\; 1$.

(5) $u_i \cdot u_{i-1} \;=\; 1$ *für ein* $i = 3, \ldots r$.

Ferner gilt: Die Differenzen $\bar{n}_i - \bar{n}_{i-1}$ für $i = 3, \ldots, r$ sind alle größer 0.

Beweis. Nach vorausgehendem Lemma müssen wir also zeigen, dass für alle $i = 1, \ldots, r$ überprüft wird, ob $u^{\bar{p}_i} \neq 1$. Genau dann ist u primitiv. Zunächst erkennen wir, dass mit

$$v^{\bar{n}_1} \;=\; u^{\bar{p}_1}$$

und

$$u_2 \;=\; w^{\bar{n}_2} \;=\; u^{\bar{p}_2}$$

die ersten beiden Kofaktoren abgedeckt wären. Ist $v = 1$ bzw. $w = 1$ hat u Ordnung d bzw. $d\,d'$, ist also nicht primitiv. Sei also $i \in \{3, \ldots, r\}$, so folgern wir

$$u_i \cdot u_{i-1} \;=\; w^{\bar{n}_i - \bar{n}_{i-1}} \cdot u_{i-1} \;=\; w^{\bar{n}_i - \bar{n}_{i-1} + \bar{n}_{i-1}} \;=\; w^{\bar{n}_i} \;=\; u^{\bar{p}_i}\,.$$

Zuletzt erkennen wir, dass aufgrund der absteigenden Sortierung der p_is, die $\bar{p}_i$s und damit die $\bar{n}_i$s aufsteigend sortiert sind, also $\bar{n}_i - \bar{n}_{i-1} > 0$ für alle $i = 3, \ldots, r$. □

Beispiel 6.28. Sei $u \in \mathbb{F}_{3^{10}}$. Da

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

sind folgende Potenzen von u zu berechnen:

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

Wir sehen jedoch dass die Potenzen

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

und

$$d' \;:=\; \mathrm{ggT}\{\tfrac{\bar{p}_2}{d}, \tfrac{\bar{p}_3}{d}\} \;=\; 61$$

uns die Arbeit erheblich erleichtern können: Berechnen wir $v := u^d = u^{44}$ und $w := v^{d'} = v^{61}$ separat, so schreiben sich die restlichen Potenzen für $\bar{n}_2 := 2$ und $\bar{n}_3 := 11$ wie folgt:

$$\begin{aligned} u^{\bar{p}_1} &= v^2\,, \\ u^{\bar{p}_2} &= u_2 \;\;\;\;\;\; = w^2\,, \\ u^{\bar{p}_3} &= u_3 \cdot u_2 = w^9 \cdot u_2\,. \end{aligned}$$

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 Potenzen d mit `commonBarFactor` und d' mit `commonBiggestBarFactor` bezeichnet und werden in p-adischer bzw. binärer Darstellung erwartet. Alle anderen zu berechnenden Potenzen, d.h. nach obigem Lemma die Liste

$$\left[\bar{n}_1,\; \bar{n}_2,\; \bar{n}_3 - \bar{n}_2, \ldots,\; \bar{n}_r - \bar{n}_{r-1}\right],$$

werden in `barFactors` wiederum in p-adischer bzw. binärer Darstellung übergeben. `barFactors` ist dabei ein einziges `int`-Array, 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`

```
1  /**
2   * Test if element is primitive.
3   *
4   * !! if matCharac is Zero, all powers are assumed as binary arrays !!
5   *
6   * !! fff,ffTmp,ffTmp2,ffTmp3,ffRet must be malloced !!
7   * !! x is NOT modified !!
8   */
9  inline bool isPrimitive(struct FFElem *ff, struct FFElem *mipo,
10         int m,
11         int *barFactors, int *lenBarFactors, int countBarFactors,
12         int *commonBarFactor, int lenCommonBarFactor,
```

```
13        int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
14        struct FFElem **matCharac,
15        struct FFElem *fff, struct FFElem *ffff, struct FFElem *ffTmp,
16        struct FFElem *ffTmp2, struct FFElem *ffRet,
17        int *tmp, int *multTable, int *addTable){
18    int i;
19    int curPos = 0;
20    bool binarySqM = (matCharac == 0);
21    struct FFElem *ffSwitch = 0;
22
23    copyFFElem(ff,fff);
24    // all barFactors are power of commonBarFactor
25    if(binarySqM)
26        powerFFElemSqM(fff,mipo,ffTmp,
27                m,commonBarFactor,lenCommonBarFactor,
28                tmp,ffTmp2,
29                multTable,addTable);
30    else
31        powerFFElem(fff,mipo,ffTmp,
32                m,commonBarFactor,lenCommonBarFactor,
33                matCharac,tmp,ffTmp2,
34                multTable,addTable);
35    if(isOne(ffTmp)) return false;
36    //switch ffTmp and fff
37    ffSwitch = fff; fff = ffTmp; ffTmp = ffSwitch;
38    copyFFElem(fff,ffff);
39    //test first barFactor
40    if(binarySqM)
41        powerFFElemSqM(ffff,mipo,ffTmp,
42                m,barFactors,lenBarFactors[0],
43                tmp,ffTmp2,
44                multTable,addTable);
45    else
46        powerFFElem(ffff,mipo,ffTmp,
47                m,barFactors,lenBarFactors[0],
48                matCharac,tmp,ffTmp2,
49                multTable,addTable);
50    if(isOne(ffTmp)) return false;
51    curPos += lenBarFactors[0];
52    //test further factors which are powers of commonBiggestBarFactor
53    //so first, calc y^commonBiggestBarFactor
54    copyFFElem(fff, ffff);
55    if(binarySqM)
56        powerFFElemSqM(ffff,mipo,ffTmp,
57                m,commonBiggestBarFactor,lenCommonBiggestBarFactor,
58                tmp,ffTmp2,
59                multTable,addTable);
60    else
61        powerFFElem(ffff,mipo,ffTmp,
62                m,commonBiggestBarFactor,lenCommonBiggestBarFactor,
63                matCharac,tmp,ffTmp2,
64                multTable,addTable);
65    if(isOne(ffTmp)) return false;
66    ffSwitch = fff; fff = ffTmp; ffTmp = ffSwitch;
```

```
67      for(i=1;i<countBarFactors;i++){
68          // copy z (fff) to ffff
69          copyFFElem(fff,ffff);
70          // *** ffff == fff == y^commonBiggestBarFactor
71          if(binarySqM)
72              powerFFElemSqM(ffff, mipo, ffTmp,
73                     m,barFactors+curPos, lenBarFactors[i],
74                     tmp,ffTmp2,
75                     multTable,addTable);
76          else
77              powerFFElem(ffff, mipo, ffTmp,
78                     m,barFactors+curPos, lenBarFactors[i],
79                     matCharac,tmp,ffTmp2,
80                     multTable,addTable);
81
82          if(i>1){
83              multiplyFFElem(ffRet,ffTmp,ffTmp2,mipo,
84                     tmp,m,multTable,addTable);
85          }else{
86              ffSwitch = ffTmp2; ffTmp2 = ffTmp; ffTmp = ffSwitch;
87          }
88          if(isOne(ffTmp2)) return false;
89          curPos += lenBarFactors[i];
90          ffSwitch = ffRet; ffRet = ffTmp2; ffTmp2 = ffSwitch;
91      }
92      return true;
93  }
```

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. Seien $\mathcal{C}_{k,t}$ 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 Satz 3.38 (3) ist u genau dann ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$, wenn

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

Folglich müssen wir, um q-Ordnungen berechnen zu können, in der Lage sein, für beliebige Zwischenkörper $E \mid K \mid F$ 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 von 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 6.29. *Seien* $F := \mathbb{F}_p[y]/(m_F(y))$, $K := \mathbb{F}_p[y]/(m_K(y))$ *und* $E := \mathbb{F}_p[y]/(m_E(y))$ *drei endliche Körper mit* $\deg(m_F) \mid \deg(m_K) \mid \deg(m_E)$. *Notiere* $q := p^{\deg(m_F)}$, $d := \frac{\deg(m_K)}{\deg(m_F)}$ *und* $\sigma : E \to E,\ v \mapsto v^q$ *den Frobenius von* F. *Sind ferner*

$$\begin{array}{rccc} \alpha_1 : & K & \to & E, \\ & [y] & \mapsto & \alpha_1(y) \end{array} \qquad \text{und} \qquad \begin{array}{rccc} \alpha_2 : & K & \to & E, \\ & [y] & \mapsto & \alpha_2(y) \end{array}$$

zwei injektive Körperhomomorphismen ($[y]$ bezeichne dabei die Restklasse von y in $\mathbb{F}_p[y]/(m_K(y))$), so gilt für $u \in E$: Entweder ist

$$\alpha_1(f)(\sigma^d)(u) = 0 \quad \text{und} \quad \alpha_2(f)(\sigma^d)(u) = 0$$

oder

$$\alpha_1(f)(\sigma^d)(u) \neq 0 \quad \text{und} \quad \alpha_2(f)(\sigma^d)(u) \neq 0\,.$$

Mit anderen Worten hängt also die Frage, ob eine Frobenius-Auswertung 0 *ist oder nicht, nicht von der Wahl der konkreten Einbettung ab.*

Beweis. Wir bemerken, dass sich $\alpha_1(y) \in \mathbb{F}_p[y]/(m_E(y))$ durch ein Polynom $\beta_1(y) \in \mathbb{F}_p[y]$ mit $\deg(\beta_1) < \deg(m_E)$ repräsentieren lässt. Analog sei $\beta_2(y) \in \mathbb{F}_p[y]$ jenes für $\alpha_2(y)$. Da es sich bei α_1 und α_2 um injektive Körperhomomorphismen handelt, sind $\operatorname{im}\alpha_1$ und $\operatorname{im}\alpha_2$ wieder endliche Körper von gleicher Ordnung und $\beta_1(y)$ bzw. $\beta_2(y)$ sind über $\mathbb{F}_p$ irreduzible Polynome von Grad $\deg(m_K)$. Wegen der Eindeutigkeit endlicher Körper sind damit die Bilder von α_1 und α_2 als Körper isomorph, was obige Behauptung zeigt. □

Damit können wir erst einmal 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 `19` 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`

```
1 /*
2  * calculates g(sigma^frobPower)(x) where g is a polynomial
3  * and sigma the frobenius
4  * application of frobenius is given by mats
```

```
 */
inline void applyFrob(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,
        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){
```

```
12      int i,j;
13      ret->len = 0;
14
15      for(i=0;i<poly->lenPoly;i++){
16          if(poly->poly[i]->len == 0) continue;
17          if(i>0){
18              j = i*frobPower-1;
19              matmul(mats+j*m, ff, ffTmp, m, multTable,addTable);
20          }else{
21              copyFFElem(ff,ffTmp);
22          }
23          multiplyFFElem(ffTmp, poly->poly[i],
24                  ffTmp2, mipo,
25                  tmp,m,multTable,addTable);
26          addFFElem(ret,ffTmp2,ret,tmp,multTable,addTable);
27      }
28  }
```

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

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

Analog zum Primitivitätstest reicht es, lediglich maximale Kofaktoren des jeweiligen verallgemeinerten Kreisteilungspolynoms zu testen. Dies ist in nachstehendem Lemma für *einen* Teiler formuliert, wobei wir uns ohne Einschränkung auf den Fall $d = 1$ beschränken können; andernfalls vollführe man lediglich eine Änderung der Notation von q, m, k und t.

Lemma 6.30. *Seien $u \in \mathbb{F}_{q^m}$ und $\Phi_{k,t}(x) \in \mathbb{F}_q[x]$ ein verallgemeinertes Kreisteilungspolynom. Sei ferner*

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

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

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

Beweis. Klar per Definition der q-Ordnung und Lemma 6.29. □

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

```
1  inline bool testSubmod(struct FFElem *ff, struct FFElem *mipo,
2          struct FFPoly **polys,
3          int polysCount, bool *evalToZero,
4          struct FFElem **mats, int *frobPowers,
5          int m, int *tmp,
6          struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
7          struct FFElem **matmulCache, bool *matmulCacheCalced,
8          int *multTable, int *addTable){
9      int i;
10     int goodCounter = 0;
11     for(i=0;i<polysCount;i++){
12         applyFrob(ff,mipo,
13                 polys[i],
14                 mats,frobPowers[i], ffTmp,
15                 m,tmp,ffTmp2,ffTmp3,
16                 matmulCache,matmulCacheCalced,
17                 multTable,addTable);
18         if( isZero(ffTmp) == evalToZero[i] ){
19             goodCounter++;
20         }else{
21             return false;
22         }
23     }
24     if(goodCounter == polysCount){
25         return true;
26     }
27     return false;
28 }
```

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

```
1  inline int testAllSubmods(struct FFElem *ff, struct FFElem *mipo,
2          int decompCount, struct FFPoly **polys,
3          int *polysCountPerDecomp, bool *evalToZero,
4          struct FFElem **mats, int *frobPowers, bool *toTestIndicator,
5          int m, int *tmp,
6          struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
7          struct FFElem **matmulCache, bool *matmulCacheCalced,
8          int *multTable, int *addTable){
9      if(ff->len == 0) return -1;
10     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 6.31. *Sei $u \in E$ ein vollständiger Erzeuger von $\mathcal{C}_{k,t}$ über F. Dann gilt*

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

wobei wiederum σ den Frobenius von F und φ die Eulersche Phifunktion 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). 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.

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`

```
1  inline void calcSubmoduleElements(struct Node *root,
2          struct FFElem *mipo,
3          int maxLenPoly,
4          unsigned long long *genCounts, int curGen,
5          struct FFPoly **polys, int polysCount, bool *evalToZero,
6          struct FFElem **mats, int matLen, int *frobPowers,
7          struct FFElem **elementsF,
8          int m, int q, int *tmp,
9          struct FFElem *ffTmp, struct FFElem *ffTmp2, struct FFElem *ffTmp3,
10         struct FFElem *ffTmp4,
11         struct FFElem **matmulCache, bool *matmulCacheCalced,
12         int *multTable, int *addTable){
13     int i,j;
14     struct Node *curRoot = root;
15     struct FFElem *ff = root->ff;
16     int *curPoly = malloc( maxLenPoly*sizeof(int) );
17     struct FFPoly *curFPoly = malloc( sizeof(struct FFPoly) );
18     curFPoly->poly = malloc( maxLenPoly*sizeof(struct FFElem*) );
19     curFPoly->lenPoly = 0;
20
21     initPoly(curPoly,maxLenPoly);
22     curPoly[0] = 2;
23     int curLenPoly = 1;
24     if( q == 2 && maxLenPoly > 1){
25         curLenPoly = 2;
26         curPoly[0] = 0;
27         curPoly[1] = 1;
28     }
29     if(q != 2 || maxLenPoly > 1){
30         while(true){
31             //setup curFPoly
32             for(i=0;i<curLenPoly;i++)
33                 curFPoly->poly[i] = elementsF[curPoly[i]];
34             curFPoly->lenPoly = curLenPoly;
35             //apply Frobenius
36             applyFrob_noCache(ff,mipo,
37                     curFPoly,
38                     mats,1, ffTmp, //return value
39                     m,tmp,ffTmp2,ffTmp3,
```

```
40                  multTable,addTable);
41          //test generated element
42          for(i=0;i<matLen;i++) matmulCacheCalced[i] = false;
43          if(testSubmod(ffTmp, mipo,
44                  polys,polysCount,evalToZero,
45                  mats,frobPowers,m,tmp,
46                  ffTmp2,ffTmp3,ffTmp4,
47                  matmulCache,matmulCacheCalced, multTable,addTable)){
48              curRoot = appendToEnd(curRoot,ffTmp,m);
49              genCounts[curGen]++;
50          }
51          //generate next element
52          curPoly[0] += 1;
53          if( curPoly[0] == q ){
54              for(i=0;i<maxLenPoly-1 && curPoly[i]==q;i++){
55                  curPoly[i] = 0;
56                  curPoly[i+1] += 1;
57              }
58              if(i+1>curLenPoly)
59                  curLenPoly = i+1;
60              if( curPoly[maxLenPoly-1]==q){
61                  break;
62              }
63          }
64      }
65  }
66  free(curPoly);
67  free(curFPoly->poly);
68  free(curFPoly);
69 }
```

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 $\{0,\ldots,q-1\}^l$. Das korrekte Polynom in $E[x]$ wird dann durch Einsetzen jeder Stelle dieses Tupels aus $\{0,\ldots,q-1\}^l$ in die `elementsF` erzeugt und in `curFPoly` gespeichert.

Die Anzahl der berechneten Erzeuger wird 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`

```
1 struct Node {
2     struct FFElem *ff;
```

```
3     struct Node *next;
4 };
```

Ebenfalls implementiert sich das Anheften eines Elements ans Ende der Liste erwartungsgemäß, 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`

```
1 /**
2  * appends element to end of root, where element is copied to new FFElem.
3  */
4 inline struct Node *appendToEnd(struct Node *root, struct FFElem *element,int m
      ){
5     struct Node *nextNode = root;
6     if( nextNode != 0){
7         while(nextNode->next != 0){
8             nextNode = nextNode->next;
9         }
10        if( nextNode->ff != 0){
11            nextNode->next = malloc( sizeof(struct Node) );
12            nextNode = nextNode->next;
13        }
14        if( nextNode != 0){
15            nextNode->next = 0;
16            nextNode->ff = mallocFFElem(m);
17            copyFFElem(element,nextNode->ff);
18            return nextNode;
19        }
20    }
21    return NULL;
22 }
```

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

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

```
1 inline void freeNode(struct Node* head){
2     struct Node *next_n = NULL;
3     struct Node *tmp_n = NULL;
4     for(tmp_n=head; tmp_n !=NULL; ){
5         next_n = tmp_n->next;
6         freeFFElem(tmp_n->ff);
7         free(tmp_n);
8         tmp_n = next_n;
9     }
10    head = 0;
11 }
```

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 können) 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 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 (primitiv) vollständig normalen 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 120 am Ende der Funktion korrigiert. Der Erzeuger selbst befindet sich wieder am aktuellen Knoten der letzten Liste des Arrays `roots`, da die Datenstrukturen so aufgebaut werden, dass dieser größte verallgemeinerte Kreisteilungsmodul der letzte ist (siehe Zeilen 26 und 28).

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

```
1  unsigned long long processLastSubmoduleAndTestPrimitivity(struct Node **roots,
2          struct FFElem *mipo, int decompCount,
3          int maxLenPoly,
4          unsigned long long *genCounts,
5          struct FFPoly **polys, int polysCount, bool *evalToZero,
6          struct FFElem **mats, int matLen, int *frobPowers,
7          struct FFElem **elementsF,
8          int m, int q,
9          int *barFactors, int *lenBarFactors, int countBarFactors,
10         int *commonBarFactor, int lenCommonBarFactor,
11         int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
12         struct FFElem **matCharac,
13         struct FFElem **matmulCache, bool *matmulCacheCalced,
14         int *multTable, int *addTable){
15     //generate temporary variables
16     struct FFElem *fff = mallocFFElem(m);
17     struct FFElem *ffff = mallocFFElem(m);
18     struct FFElem *ffTmp = mallocFFElem(m);
19     struct FFElem *ffTmp2 = mallocFFElem(m);
20     struct FFElem *ffTmp3 = mallocFFElem(m);
21     struct FFElem *ffTmp4 = mallocFFElem(m);
22     struct FFElem *ffTmp5 = mallocFFElem(m);
23     int *tmp = malloc(m*sizeof(int));
24
25     int i,j;
```

```
26    int curGen = decompCount-1;
27    struct Node **curRoots = malloc( decompCount*sizeof(struct Node*) );
28    struct FFElem *ff = roots[curGen]->ff;
29    int *curPoly = malloc( maxLenPoly*sizeof(int) );
30    struct FFPoly *curFPoly = malloc( sizeof(struct FFPoly) );
31    curFPoly->poly = malloc( maxLenPoly*sizeof(struct FFElem*) );
32    curFPoly->lenPoly = 0;
33
34    initPoly(curPoly,maxLenPoly);
35    curPoly[0] = 1;
36    int curLenPoly = 1;
37
38    unsigned long long pcn = 0;
39    while(true){
40        //setup curFPoly
41        for(i=0;i<curLenPoly;i++)
42            curFPoly->poly[i] = elementsF[curPoly[i]];
43        curFPoly->lenPoly = curLenPoly;
44        //apply Frobenius
45        applyFrob_noCache(ff,mipo,
46                curFPoly,
47                mats,1, fff, //return value
48                m,tmp,ffTmp,ffTmp2,
49                multTable,addTable);
50        //test generated element
51        for(i=0;i<matLen;i++) matmulCacheCalced[i] = false;
52        if(testSubmod(fff, mipo,
53                polys,polysCount,evalToZero,
54                mats,frobPowers,m,tmp,
55                ffTmp,ffTmp2,ffTmp3,
56                matmulCache,matmulCacheCalced, multTable,addTable)){
57            genCounts[curGen]++;
58            // build element as sum of already calced Nodes and ffTmp
59            for(i=0;i<decompCount;i++) curRoots[i] = roots[i];
60            // cycle through Nodes, build element and test primitivity
61            while(true){
62                copyFFElem(fff,ffff);
63                //build element
64                for(i=0;i<decompCount-1;i++){
65                    addFFElem(ffff, curRoots[i]->ff, ffff, tmp,
66                            multTable,addTable);
67                }
68                //test primitivity
69                if(countBarFactors > 0){
70                    if(isPrimitive(ffff, mipo,m,
71                                barFactors,lenBarFactors,countBarFactors,
72                                commonBarFactor,lenCommonBarFactor,
73                                commonBiggestBarFactor,lenCommonBiggestBarFactor,
74                                matCharac,
75                                ffTmp,ffTmp2,ffTmp3,ffTmp4,ffTmp5,
76                                tmp,multTable,addTable)){
77                        pcn++;
78                    }
79                }
```

```
80
81                  //next element
82                  curRoots[0] = curRoots[0]->next;
83                  if( curRoots[0] == 0 ){
84                     for(i=0;i<decompCount-1 && curRoots[i]==0;i++){
85                         curRoots[i] = roots[i];
86                         curRoots[i+1] = curRoots[i+1]->next;
87                     }
88                  }
89                  if( curRoots[decompCount-1] == 0){
90                     break;
91                  }
92              }
93          }
94          //generate next element
95          curPoly[0] += 1;
96          if( curPoly[0] == q ){
97              for(i=0;i<maxLenPoly-1 && curPoly[i]==q;i++){
98                  curPoly[i] = 0;
99                  curPoly[i+1] += 1;
100             }
101             if(i+1>curLenPoly)
102                 curLenPoly = i+1;
103             if( curPoly[maxLenPoly-1]==q){
104                 break;
105             }
106         }
107     }
108     free(curPoly);
109     free(curFPoly->poly);
110     free(curFPoly);
111     freeFFElem(fff);
112     freeFFElem(ffff);
113     freeFFElem(ffTmp);
114     freeFFElem(ffTmp2);
115     freeFFElem(ffTmp3);
116     freeFFElem(ffTmp4);
117     freeFFElem(ffTmp5);
118
119     //we added first element twice
120     genCounts[curGen]--;
121     return pcn;
122 }
```

Bemerkung 6.32. Wie Zeile 69 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 zum 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 primitiv vollständig normalen Elemente trägt. Alle zu übergebenden Parameter werden in `Sage` erzeugt und wurden bereits erklärt.

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

```
1  unsigned long long processFiniteField(struct FFElem *mipo, int decompCount,
2          struct FFPoly **polys, int *polysCountPerDecomp,
3          bool *evalToZero, int *maxLenPolysPerDecomp,
4          struct FFElem **mats, int matLen, int *frobPowers,
5          unsigned long long *genCounts, int m, int charac, int q,
6          int *barFactors, int *lenBarFactors, int countBarFactors,
7          int *commonBarFactor, int lenCommonBarFactor,
8          int *commonBiggestBarFactor, int lenCommonBiggestBarFactor,
9          struct FFElem **matCharac, struct FFElem **elementsF,
10         int *multTable, int *addTable){
11     time_t TIME = time(NULL);
12     int i,j;
13
14     //setup temporary variables ------------------------------------------
15     int *tmp = malloc(m*sizeof(int));
16     struct FFElem *ff = mallocFFElem(m);
17     initPoly(ff->el,m);
18     struct FFElem *ffRet = mallocFFElem(m);
19     struct FFElem *ffTmp = mallocFFElem(m);
20     struct FFElem *ffTmp2 = mallocFFElem(m);
21     struct FFElem *ffTmp3 = mallocFFElem(m);
22     struct FFElem *ffTmp4 = mallocFFElem(m);
23
24     struct FFElem **matmulCache = malloc(matLen*sizeof(struct FFElem));
25     for(i=0;i<matLen;i++) matmulCache[i] = mallocFFElem(m);
26     bool *matmulCacheCalced = malloc(matLen*sizeof(bool));
27
28     bool *toTestIndicator = malloc(decompCount*sizeof(bool));
29     struct Node **roots = malloc( decompCount*sizeof(struct Node) );
30     struct Node **curRoots = malloc(decompCount*sizeof(struct Node*));
31     for(i=0;i<decompCount;i++){
32         roots[i] = malloc( sizeof(struct Node) );
33         roots[i]->ff = 0;
34         roots[i]->next = 0;
35         curRoots[i] = roots[i];
36         toTestIndicator[i] = true;
37     }
38     //------------------------------------------------------------------------
39
40     int foundCounter = 0;
41     for(i=0;i<decompCount;i++) genCounts[i] = 0;
42
43     // chase for elements ------------------------------------------------
44     while(true){
45         for(i=0;i<matLen;i++) matmulCacheCalced[i] = 0;
46         int curGen = testAllSubmods(ff,mipo,decompCount,
47                 polys,polysCountPerDecomp,evalToZero,
48                 mats,frobPowers,toTestIndicator,
```

```
49              m,tmp,ffTmp,ffTmp2,ffTmp3,
50              matmulCache,matmulCacheCalced,
51              multTable,addTable);
52      if( curGen != -1 ){
53          if(toTestIndicator[curGen] == true){
54              genCounts[curGen]++;
55              appendToEnd(roots[curGen], ff, m);
56              foundCounter++;
57              toTestIndicator[curGen] = false;
58          }
59          if(foundCounter == decompCount) break;
60      }
61      //generate next element
62      // (for sure there is a more efficient method)
63      ff->el[0] += 1;
64      if( ff->el[0] == charac ){
65          for(i=0; i<m-1 && ff->el[i]==charac; i++){
66              ff->el[i] = 0;
67              ff->el[i+1] += 1;
68          }
69          if( ff->el[m-1] == charac )
70              break;
71      }
72      updateFFElem(ff,m);
73  }
74  if( foundCounter != decompCount ){
75      printf("BAAAD␣ERROR!!!␣foundCounter=%i␣<␣decompCount=%i\n",
76              foundCounter,decompCount);
77      exit(0);
78  }
79  printf("finding␣time:␣%.2f\n", (double)(time(NULL)-TIME));
80  //------------------------------------------------------------------------
81
82
83
84  // Process found elements ------------------------------------------------
85  int curDecompPosition = 0;
86  for(i=0;i<decompCount-1;i++){
87      calcSubmoduleElements(roots[i], mipo,
88              maxLenPolysPerDecomp[i], // *** == maxLenPoly
89              genCounts,i, // *** i == curGen
90              polys+curDecompPosition, polysCountPerDecomp[i],
91              evalToZero+curDecompPosition,
92              mats, matLen, frobPowers+curDecompPosition,
93              elementsF,
94              m,q,tmp,
95              ffTmp,ffTmp2,ffTmp3,ffTmp4,
96              matmulCache,matmulCacheCalced,
97              multTable,addTable);
98      curDecompPosition += polysCountPerDecomp[i];
99  }
100 printf("all␣not␣last␣time:␣%.2f\n", (double)(time(NULL)-TIME));
101 //------------------------------------------------------------------------
102
```

```
103     // Process last Decomposition ------------------------------------------
104     int curGen = decompCount-1;
105     unsigned long long pcn =
106         processLastSubmoduleAndTestPrimitivity(roots,mipo,decompCount,
107             maxLenPolysPerDecomp[curGen], // *** == maxLenPoly
108             genCounts,
109             polys+curDecompPosition,polysCountPerDecomp[curGen],
110             evalToZero+curDecompPosition,
111             mats,matLen,frobPowers+curDecompPosition,
112             elementsF,
113             m,q,
114             barFactors,lenBarFactors,countBarFactors,
115             commonBarFactor,lenCommonBarFactor,
116             commonBiggestBarFactor,lenCommonBiggestBarFactor,
117             matCharac,
118             matmulCache,matmulCacheCalced,
119             multTable,addTable);
120     //-------------------------------------------------------------------------
121
122     //free variables
123     for(i=0;i<decompCount;i++)
124         freeNode(roots[i]);
125     free(roots); free(curRoots);
126
127     //free temporary variables
128     free(tmp);
129     freeFFElem(ff);
130     freeFFElem(ffRet);
131     freeFFElem(ffTmp);
132     freeFFElem(ffTmp2);
133     freeFFElem(ffTmp3);
134     freeFFElem(ffTmp4);
135     for(i=0;i<matLen;i++) freeFFElem(matmulCache[i]);
136     free(matmulCache);
137     free(matmulCacheCalced);
138     free(toTestIndicator);
139
140
141     printf("total␣time:␣%.2f\n", (double)(time(NULL)-TIME));
142     return pcn;
143 }
```

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 6.33. 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

$$\texttt{[(1, 1, 1), (2, 1, 1), (4, 1, 1), (5, 4, 1)]}.$$

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):
    Zn = IntegerModRing(m)
    return Zn(q).multiplicative_order()
```

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 *alle* Teiler des Modulcharakters testen, um vollständige Erzeuger zu finden. Jedoch garantiert Satz 5.3, dass dies in manchen Fällen überflüssig ist, da beispielsweise 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 d als echten Faktor enthalten, da – wie man sich sehr leicht überlegt – falls $\mathbb{F}_{q^n}$ über $\mathbb{F}_{q^d}$ einfach ist, auch $\mathbb{F}_{q^n}$ über $\mathbb{F}_{q^{de}}$ für alle Teiler $e \mid \frac{n}{d}$ einfach ist.

Dies wollen wir in nachstehender Funktion nutzen, die gerade die zu betrachtenden Teiler einer Erweiterung liefert.

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

```
# returns the NOT completely basic divisors of an
# extension n over GF(p^e)
def get_not_completely_basic_divisors(p,e,n):
    n = Integer(n)
    q = Integer(p**e)
    divs = []
    divsN = divisors(n)
    while len(divsN) > 0:
        d = divsN.pop(0)
        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
```

```
15          divs += [d]
16          if isComplBasic:
17              divsN = filter(lambda k: not d.divides(k), divsN)
18      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 den Daten (k, t, π) des zu betrachtenden Kreisteilungsmoduls $\mathcal{C}_{k,t\pi}$ 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 im Absatz vor der Definition von Regularität (Definition 5.11) erwähnt, 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 (primitiv) 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 durch die Übergabe von `testPrimitivity=False` (vgl. Bemerkung 6.32) vollständig deaktivieren. Durch den Parameter `onlyNormal=True` wird lediglich die Anzahl der (primitiv) normalen Elemente durch Auslassen aller echten Teiler von n (vgl. Zeile 21 in Listing 6.36) bestimmt.

Der Rückgabewert der Funktion enthält die Anzahl aller vollständig 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,\
        onlyNormal=False):
    TIME = time.time()
    p = F.characteristic()
```

```
6     q = F.order();
7     e = q.log(p)
8     E = F.extension(Integer(n),'a');
9     P = E.prime_subfield()
10    #generate factors
11    polys = []
12    polysCount = []
13    evalToZero = []
14    frobPowers = []
15    maxLenPolysPerDecomp = []
16    notComplBasicDivisors = get_not_completely_basic_divisors(p,e,n)
17    decomposition = decompose(p,e,n)
18    for decomp in decomposition:
19        k,t,pi = decomp
20        if onlyNormal:
21            divs = [1]
22        else:
23            divs = divisors(get_module_character(*decomp))
24            divs = filter(lambda x: x in notComplBasicDivisors, divs)
25            if isRegular(p,e,k,t,pi):
26                if get_tau_divisors(p,e, k,t,pi) != divs:
27                    divs = get_tau_divisors(p,e, k,t,pi)
28        maxLenPolysPerDecomp += [ euler_phi(k)*t*pi ]
29        countPolysForThisDecomp = 0
30        for d in divs:
31            G = F.extension(Integer(d), 'c');
32            Gx = PolynomialRing(G,'x');
33            h = Hom(G,E)[0]
34            cycl = Gx.cyclotomic_polynomial(squarefree(k))\
35                   (Gx.gen()**(k*t*pi/squarefree(k)/d))
36            if countPolysForThisDecomp == 0:
37                polys += [map(lambda x: x.polynomial().list(),
38                    cycl.map_coefficients(h).list())]
39                frobPowers += [d]
40                evalToZero += [1]
41                countPolysForThisDecomp += 1
42            # add Co-Factors
43            for f,mult in cycl.factor():
44                g = cycl.quo_rem(f)[0]
45                gE = g.map_coefficients(h)
46                polys += [map(lambda x: x.polynomial().list(), gE.list())]
47                frobPowers += [d]
48                evalToZero += [0]
49                countPolysForThisDecomp +=1
50        polysCount += [countPolysForThisDecomp]
```

Wie man gut erkennen kann, werden einfache Zerlegungen in den Zeilen 16 und 24 ausgenutzt und für reguläre Kreisteilungsmoduln wird die Funktion `get_tau_divisors` (Zeilen 25f) aufgerufen.

Anschließend (Listing 6.37) 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 23) oder p-adischer Form (ab Zeile 28) ge-

nutzt 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 36).

Listing 6.37: `countCompleteSubmoduleGenerators` Fortsetzung (I)

```
1     charac = int(E.characteristic())
2         #mipo, idcsMipo
3     mipo = E.modulus().list()
4     m = len(mipo)-1
5
6     #calc prime factors of order
7     barFactors = []
8     primitiveOrder = E.order()-1
9     if testPrimitivity:
10        factors = reversed(factor(primitiveOrder))
11        for r,k in factors:
12            barFactors += [primitiveOrder/r]
13        countBarFactors = len(barFactors)
14        commonBarFactor = gcd(barFactors)
15        commonBiggestBarFactor = max(gcd(barFactors[1:]) / commonBarFactor,1)
16        barFactors = map(lambda b: b/commonBarFactor, barFactors)
17        curF = 0
18        barFactors_tmp = [barFactors[0]]
19        for b in barFactors[1:]:
20            barFactors_tmp += [ b/commonBiggestBarFactor - curF]
21            curF = b/commonBiggestBarFactor
22        barFactors = barFactors_tmp
23        if binaryPowers:
24            barFactors = map(lambda b: get_padic_representation(b,2),barFactors)
25            commonBarFactor = get_padic_representation(commonBarFactor,2)
26            commonBiggestBarFactor = \
27                    get_padic_representation(commonBiggestBarFactor,2)
28        else:
29            barFactors = map(lambda b: get_padic_representation(b,p),barFactors)
30            commonBarFactor = get_padic_representation(commonBarFactor,p)
31            commonBiggestBarFactor = \
32                    get_padic_representation(commonBiggestBarFactor,p)
33        lenCommonBarFactor = len(commonBarFactor)
34        lenCommonBiggestBarFactor = len(commonBiggestBarFactor)
35
36        lenBiggestZeroGap = 0
37        if not binaryPowers:
38            #find biggest gap (i.e. zero-interval)
39            lenCurGap = 0
40            for b in barFactors+[commonBarFactor]+[commonBiggestBarFactor]:
41                i = 0
42                while i < len(b):
43                    lenCurGap = 0
44                    while i<len(b) and b[i] == 0:
45                        lenCurGap+= 1
46                        i += 1
47                    lenBiggestZeroGap = max(lenBiggestZeroGap, lenCurGap)
48                    i += 1
49    else:
```

```
50          countBarFactors = 0
51          barFactors = []
52          commonBarFactor = []
53          commonBiggestBarFactor = []
54          lenBiggestZeroGap = 0
```

Im letzten Teil der reinen Sage-Aufbereitung (Listing 6.38), 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)

```
1           #generate F elements in E
2       elementsF = []
3       if e == 1:
4           elementsF = map(lambda e: [e], list(F))
5       else:
6           h = Hom(F,E)[0]
7           for e in itertools.product(xrange(p),repeat=e):
8               elementsF += [h( F(list(reversed(e))) ).polynomial().list()]
9
10          #calculate addition and multiplication tables
11      ps = range(p)
12      addTable = ps[P(-2*(p-1)):] + ps*2 + ps[:Integer(P(2*(p-1)))+1]
13      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 ist. Beispielsweise sind Listen von Ganzzahlen in Sage keineswegs `C`-kompatible Arrays. Es existiert jedoch gerade für diesen Zweck die Möglichkeit, die komfortable Syntax von `numpy`-Arrays zu nutzen, die direkt auf `C`-Arrays basieren.[2]

Andere Datenstrukturen, wie die selbst erstellten **`struct FFElem`**s, 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`

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

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

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):
        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 Übersetzung 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,
```

```
21              p, multTableC, addTableC)
22    #setup polynomials, polyLength, frobPowers, evaltoZero
23    decompCount = int(len(polysCount))
24        #evalToZeroC
25    cdef np.ndarray[char,ndim=1,mode="c",cast=True] evalToZeroC\
26           = np.array(evalToZero, dtype=np.uint8)
27        #frobPowersC
28    cdef np.ndarray[int,ndim=1,mode="c"] frobPowersC\
29           = np.array(frobPowers, dtype=np.int32)
30        #polysCountC
31    cdef np.ndarray[int,ndim=1,mode="c"] polysCountC\
32           = np.array(polysCount, dtype=np.int32)
33    cdef FFPoly **polysC = pyList2PointFFPoly(polys,m)
34    cdef np.ndarray[int,ndim=1,mode="c"] maxLenPolysPerDecompC\
35           = np.array(maxLenPolysPerDecomp, dtype=np.int32)
36        # bar Factors
37    cdef np.ndarray[int,ndim=1,mode="c"] barFactorsC \
38        = np.array(list(itertools.chain(*barFactors)), dtype=np.int32)
39    cdef np.ndarray[int,ndim=1,mode="c"] lenBarFactorsC \
40        = np.array(map(len,barFactors), dtype=np.int32)
41    cdef np.ndarray[int,ndim=1,mode="c"] commonBarFactorC \
42        = np.array(commonBarFactor, dtype=np.int32)
43    cdef np.ndarray[int,ndim=1,mode="c"] commonBiggestBarFactorC \
44        = np.array(commonBiggestBarFactor, dtype=np.int32)
45        # F elements in E
46    cdef FFElem **elementsFC = pyList2PointFFElem(elementsF,m)
47    #============================================================================
```

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 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 2 in Listing 6.43 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)

```
1     #setup return values
2     cdef np.ndarray[unsigned long long,ndim=1,mode="c"] genCountsC
3     genCountsC = np.zeros(decompCount, dtype=np.ulonglong)
4
5     cdef unsigned long long pcn = \
6            processFiniteField(mipoC, decompCount,
7                   polysC,<int*>polysCountC.data,
8                   <char*>evalToZeroC.data,
9                   <int*>maxLenPolysPerDecompC.data,
10                  matsC,maxMatPower,<int*>frobPowersC.data,
```

```
                    <unsigned long long*>genCountsC.data, m, p, q,
                    <int*>barFactorsC.data, <int*>lenBarFactorsC.data,
                    countBarFactors,
                    <int*>commonBarFactorC.data,lenCommonBarFactor,
                    <int*>commonBiggestBarFactorC.data,lenCommonBiggestBarFactor,
                    matCharacC,elementsFC,
                    multTableC,addTableC)

    genCounts = dict()
    for i,d in enumerate(decomposition):
        genCounts[d] = Integer(genCountsC[i])

    # Free all malloced variables at the end ================================
    freeFFElem(mipoC)
    freeFFElemMatrix(matsC,m*maxMatPower)
    for i in range(len(polys)):
        freeFFPoly(polysC[i])
    free(polysC)
    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 der 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_not_completely_basic_divisors` (Listing 6.31). Dazu gehen wir alle Teiler von $n = 6$ durch und überprüfen, ob die jeweiligen Erweiterungen einfach sind, d.h. für jeden Teiler $d \mid n$ testen wir für jeden Primteiler $r \mid \frac{n}{d}$, ob $r \nmid \operatorname{ord}_{(\frac{n}{dr})'}(q^d)$ (vgl. Satz 5.3). Wir brechen jeweils ab, falls ein r die Teilbarkeitsbedingung nicht erfüllt.

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

Damit sind alle zu betrachtenden Teiler von n gegeben durch

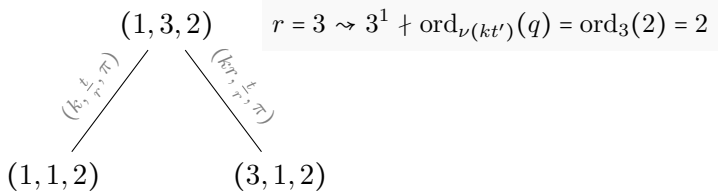
$$\texttt{notComplBasicDivisors} \coloneqq [1,2,3]\,.$$

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

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

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

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



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

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

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

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

(1) Wir starten beim ersten erweiterten Kreisteilungspolynom

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

Es ist nun $(k,t,\pi) \coloneqq (1,1,2)$ und wir müssten alle Teiler des Modulcharakters $\frac{k\,t\,\pi}{\nu(k)} = 2$, betrachten. Wie man an obig berechnetem `notComplBasicDivisors` erkennt, lässt sich in diesem Fall einer der drei Teiler $\{1,2,3\}$ streichen. Ein zweiter Kniff schafft eine weitere Reduktion der Teilerzahl, da $(1,1,2)$ regulär über $\mathbb{F}_2$ ist:

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

Da $(1,1,2)$ 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 $\operatorname{ord}_k(q) = \operatorname{ord}_1(2) = \operatorname{ord}_{\nu(k)}(q)$.

$d = 1$. 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 von `notComplBasicDivisors` einen Teiler wegdiskutieren. Anders als in obigem Fall ist dieser Kreisteilungsmodul nicht regulär, da

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

nicht teilerfremd zu $kt = 2$ ist. Also bleiben die beiden 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 $\mathcal{C}_{3,1\cdot 2}$ als $\mathbb{F}_{2^2}[x]$-Modul zu betrachten ist. 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

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

und faktorisieren

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

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

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

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

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

Damit schreiben wir obige Kofaktoren zu

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

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

$u := \begin{bmatrix}1&0&0&0&0&0\end{bmatrix}^T$. 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 am ersten Polynom:

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

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

$u := \begin{bmatrix}0 & 1 & 1 & 0 & 0 & 0\end{bmatrix}^T$. 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.

...

$u := \begin{bmatrix}0 & 1 & 1 & 1 & 0 & 0\end{bmatrix}^T$. 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. Unterabschnitt 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 6.31 den Modul durch Polynome über F, deren Grad kleiner 2 ist, enumerieren:

$f(x)$	$f(\sigma)(u)$	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:

$$\texttt{roots} \;=\; \left[\, [a^3 + a^2 + a,\; a^3 + a^2 + a + 1],\; [a^2 + a,\; a^4 + a^2,\; a^5 + a^2 + a + 1,\; a^5 + 1,\; a^5 + a^2,\; a^5 + a^4] \,\right]$$

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 mit dem ersten Faktor aus `barFactors` fortfahren:

$$w^3 \;=\; a^5 + a^4 + a^2\,.$$

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.

Erz. von $\mathcal{C}_{1,1\cdot 2}$	Erz. von $\mathcal{C}_{3,1\cdot 2}$	vollst. norm. 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

Mit Hilfe obig vorgestellter Implementierung konnten wir die Werte $\mathcal{CN}(q,n)$ und $\mathcal{PCN}(q,n)$, sowie die konkreten Anzahlen der Erzeuger der Teilmoduln nach Anwendung des Zerlegungssatzes für die in Tabelle 6.1 aufgeführten Paare (q,n) bestimmen. Falls ein Paar (q,n) nicht einfach ist, so haben wir für die Primzahlen $p = 2,3,5$ auch einige Anzahlen normaler und primitiv normaler Elemente bestimmt und diese Bereiche in Tabelle 6.2 gelistet.

Alle berechneten Ergebnisse werden in Tabellen im Anhang bereitgestellt. Wie man erkennen kann, konnten wir alle Werte von Morgan und Mullen aus [24] reproduzieren und viele weitere Werte liefern. 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

Tabelle 6.1: Bereiche der $\mathcal{PCN}$-Enumerationen

q	n
2	$2, \ldots, 31$
3	$2, \ldots, 20$
4	$2, \ldots, 14$
5	$2, \ldots, 12$
7	$2, \ldots, 11$
$8, 9$	$2, \ldots, 9$
$11, 13, 16, 17, 19, 23$	$2, \ldots, 7$
25	$2, \ldots, 6$
27	$2, \ldots, 4$
29	$2, \ldots, 7$
31	$2, \ldots, 6$
32	$2, \ldots, 4$
$37, 41, 43$	$2, \ldots, 6$
$121, 169$	$2, \ldots, 4$
$361, 529, 841, 961$	$2, 3$
$1369, 1681, 1849$	2

q	n
$2, \ldots, 961$ Primzahlpotenzen	3
$2, \ldots, 243$ Primzahlpotenzen	4
$2, \ldots, 43$ Primzahlpotenzen	6

Tabelle 6.2: Bereiche der $\mathcal{PN}$-Enumerationen

q	n
2	$6, 10, 12, 18, 20, 21, 22, 24, 26, 27, 30$
3	$8, 10, 14, 16, 20$
4	10
5	$6, 12$

der Suche nach einer allgemeinen Formel für die vollständigen Erzeuger eines Kreisteilungsmoduls helfen können (Wir erinnern uns, dass Satz 6.17 lediglich für reguläre Kreisteilungsmodule gilt). Der kleinste Fall, für den wir nicht in der Lage waren ein Ergebnis zu erlangen, ist $(q, n) = (2, 32)$.

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 die Erweiterungen von Grad n eines *jeden* endlichen Köpers 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. [13, Abschnitt vor Section 2]):

Lemma 6.34. *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 ist eine Carmichael Zahl[3].

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 6.35.

Es gilt:

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

Beweis. Klar nach Korollar 5.4 und Lemma 6.34. □

Doch wie können wir $\mathcal{G}$ noch größer werden lassen? Betrachten wir einmal alle $n \in \{2, 3, \ldots, 33\}$, so sehen wir mit obiger Proposition, dass für n gleich

$$6,\ 12,\ 14,\ 15,\ 18,\ 20,\ 21,\ 22,\ 24,\ 26,\ 28,\ 30$$

[3] Eine *Carmichael Zahl* ist eine ungerade natürliche Zahl N, sodass für jeden Primteiler r von N gilt: $r - 1$ teilt $N - 1$

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 6.36.

Für alle $n \in \mathbb{N}^*$ mit $2 \leq n \leq 33$ gilt $n \in \mathcal{G}$.

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

Im Gegensatz zur Enumeration einer einzigen Körpererweiterung geht es nun darum, viele Körpererweiterungen mit immer gleichem Erweiterungsgrad zu betrachten. Beispielsweise sind für $n = 30$ genau 64902 Körpererweiterungen zu prüfen, wobei das größte auftauchende q gleich 809993 ist. 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 6.37. *Sei $E := \mathbb{F}_{q^n}$ über $F := \mathbb{F}_q$ eine Erweiterung endlicher Körper. Sei $x^n - 1 = \prod_{i=1}^{l} \Phi^{\pi}_{k_i,t_i}$ die feinst mögliche Zerlegung über F nach dem Zerlegunssatz (Satz 5.10). Notiere*

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

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 $x^{\frac{n}{d}} - 1 = \prod_{i=1}^{l_d} f_i(x)^{\nu_i}$ 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 liefern. 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 $\mathrm{Ord}_{q^d}(u_i) = \Phi_{\nu(k_i),\frac{k_i t_i \pi}{\nu(k_i)d}}$ 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;
```

```
10        facsAll[d] = list((Gx.gen()**(n/d)-1).factor());
11        prodsAll[d] = dict();
12        for idx,(f,mult) in enumerate(facsAll[d]):
13            prodsAll[d][idx] = (Gx.gen()**(n/d)-1).quo_rem(f)[0]
```

Bereits Morgan und Mullen haben in [24] 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, also Polynome, bei denen lediglich drei Koeffizienten nicht verschwinden, 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 6.38. *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) $a_{n-1} = -\mathrm{Tr}_{\mathbb{F}_{q^n}|\mathbb{F}_q}(u) \neq 0$ *und*

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

Beweis. Die beiden Identitäten sind klar durch Koeffizientenvergleich von $f(x) = (x-u)(x-u^q)(x-u^{q^2}) \cdot \ldots \cdot (x-u^{q^{n-1}})$.[4] Die Spur eines (vollständig) normalen Elements einer Körpererweiterung 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

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

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

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

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

dabei auf folgende Ordnungsrelation:

Definition 6.39.

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 $a_j \neq 0$ genau für $j \in \{i_1, \ldots, i_k\}$. Analog sei $a(f)$ das Wort über dem Alphabet $\mathbb{F}_q$, das die nicht verschwindenden Koeffizienten von f ihrem Index absteigend 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_j ist der nicht verschwindende Koeffizient von x^j in f.

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

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

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

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

- 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 $a(x) \preceq b(x)$ via dieser Definition für Repräsentanten $a(x)$ von u und $b(x)$ von v mit $a(x), b(x) \in \mathbb{F}_p[x]$ 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

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

bevorzugt werden.

Bemerkung 6.40. Die oben vorgestellte Ordnung entspricht nicht ganz derer, die von Morgan und Mullen in [24] verwendet wurde. Die dortige basierte auf einem Vergleich der Hamming-Gewichte und bei Gleichheit auf der Zahl $N_f \in \mathbb{N}^*$, die für $f(x) = x^n + \sum_{i=0}^{n-1} a_i x^i$ über $\mathbb{F}_p$ durch $N_f := p^n + \sum_{i=0}^{n-1} a_i p^i$ (für $a_i \in \{0, \ldots, p-1\}$) gegeben war. Wir empfanden jedoch beispielsweise das Polynom $f(x) = x^4 + 2x^3 + x + 2$ kleiner als das Polynom $g(x) = x^4 + x^3 + x^2 + 2$. In der Ordnung von Morgan und Mullen ist jedoch $N_f = 140$ größer als $N_g = 119$. ◢

Die konkrete Suche nach einem $\mathcal{PCN}$-Polynom für ein Paar (q, n) 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 $q = p^r$ und $r > 1$ 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,primitivity_with_bar_factors=False):
    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:
                if F(coeffN).multiplicative_order() != p-1: continue
                if is_even(n):
                    primitives += [coeffN]
                else:
                    coeffN *= (-1)
                    primitives = [coeffN] + primitives
                f = Fx.gen()**n + coeffT*Fx.gen()**(n-1) + coeffN
                if not f.is_irreducible(): continue

                E = GF(orderE, name='a', modulus=f)

                if E.gen().multiplicative_order() == primOrder \
                        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 isPrimitive(E.gen(),primOrder,barFactors) \
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 6.38 zu Nutze und bestimmen anfangs dynamisch die primitiven Elemente des Grundkörpers (vgl. Zeilen 37-43). Um die Ordnung nach Definition 6.39 beizubehalten, sind wir hier gezwungen 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 gezwungen sind, einige „Umwege“ zu gehen.

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

```
1 def findAnyPCN_polynom(p,r,n):
2     if r == 1:
3         return findAnyPCN_polynom_prime(p,n)
4
5     q = p**r
6     F = GF(q,'a')
7
```

```
8   E = F.extension(n,'a')
9   P = E.prime_subfield()
10
11  Px = PolynomialRing(P,'x')
12  Fx = PolynomialRing(F,'x')
13  Ex = PolynomialRing(E,'x')
14  h = Hom(F,E)[0]
15  primOrder = E.order()-1
16
17  primitives = []
18
19  #setup factors of x^n-1
20  divs = get_proper_subfield_divisors(p,r,n)
21  facsAll = dict();
22  prodsAll = dict();
23  fieldsAll = dict();
24  for d in divs:
25      G = F.extension(Integer(d), 'c');
26      Gx = PolynomialRing(G,'x');
27      fieldsAll[d] = G;
28      facsAll[d] = list((Gx.gen()**(n/d)-1).factor());
29      prodsAll[d] = dict();
30      for idx,(f,mult) in enumerate(facsAll[d]):
31          prodsAll[d][idx] = (Gx.gen()**(n/d)-1).quo_rem(f)[0]
32
33  # list elements of F
34  Flist = [F.zero()]
35  for i in xrange(1,r+1):
36      for idcs in itertools.combinations(xrange(0,r),i):
37          for koeffs in itertools.product(xrange(1,p),repeat=i):
38              Flist += [F(list(sum([e * Px.gen()**idcs[j] for \
39                     j,e in enumerate(reversed(koeffs))])))]
40
41  if not is_even(n):
42      FprimList = [F.zero()]
43      for i in xrange(1,r+1):
44          for idcs in itertools.combinations(xrange(0,r),i):
45              for koeffs in itertools.product(xrange(p-1,0,-1),repeat=i):
46                  FprimList += [F(list(sum([e * Px.gen()**idcs[j] for \
47                         j,e in enumerate(reversed(koeffs))])))]
48
49  firstRun = True
50  # first test trinoms!
51  for coeffT in xrange(1,q):
52      coeffTF = Flist[coeffT]
53      if firstRun:
54          for coeffN in xrange(p,q):
55              if is_even(n):
56                  coeffNF = Flist[coeffN]
57              else:
58                  coeffNF = FprimList[coeffN]
59              if coeffNF.multiplicative_order() != F.order()-1: continue
60              if not is_even(n): coeffNF *= (-1)
61              primitives += [coeffNF]
```

```
62                f = Fx.gen()**n + coeffTF*Fx.gen()**(n-1) + coeffNF
63                if not f.is_irreducible(): continue
64                for fac,mul in Ex(f.map_coefficients(h)).factor():
65                    if fac.degree() == 1:
66                        x = -fac[0]
67                        if x.multiplicative_order() == primOrder \
68                                and isCompletelyNormal(x,E,q,divs,\
69                                fieldsAll,facsAll,prodsAll):
70                            return x,f
71                        else: break
72                    else: break
73            firstRun = False
74        else:
75            for coeffNF in primitives:
76                f = Fx.gen()**n + coeffTF*Fx.gen()**(n-1) + coeffNF
77                if not f.is_irreducible(): continue
78                for fac,mul in Ex(f.map_coefficients(h)).factor():
79                    if fac.degree() == 1:
80                        x = -fac[0]
81                        if x.multiplicative_order() == primOrder \
82                                and isCompletelyNormal(x,E,q,divs,\
83                                fieldsAll,facsAll,prodsAll):
84                            return x,f
85                        else: break
86                    else: break
87    # test rest
88    for length in xrange(1,n-1):
89        for idcs in itertools.combinations(xrange(1,n-1),length):
90            for xs in itertools.product(xrange(1,q),repeat=length+1):
91                for x in primitives:
92                    f = Fx.gen()**n + Flist[xs[0]]*Fx.gen()**(n-1) + x
93                    for j,j2 in enumerate(idcs):
94                        f += Flist[xs[length-j]] * Fx.gen() ** j2
95                    if not f.is_irreducible(): continue
96                    for fac,mul in Ex(f.map_coefficients(h)).factor():
97                        if fac.degree() == 1:
98                            y = -fac[0]
99                            if y.multiplicative_order() == primOrder \
100                                   and isCompletelyNormal(y,E,q,divs,\
101                                   fieldsAll,facsAll,prodsAll):
102                               return y,f
103                           else: break
104                   else: break
```

Man erkennt, dass wir hier die Elemente aus `F` eigens generieren müssen, um den Bedingungen aus Definition 6.39 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`

```
1 def findAnyPCN_polynom_wrapper(n, border=lambda n:n**4, \
2         fileoutput=False, filepath="pcns_trinom_", \
```

```
3         startPrime=1, stopPrime=0, onlyR=None, \
4         cpuNum=1):
5     if fileoutput:
6         st = datetime.datetime.\
7                fromtimestamp(time.time()).strftime('%Y-%m-%d_%H:%M:%S')
8         filepath += str(n)+"_"
9         if onlyR != None: filepath += str(onlyR)+"_"
10        filepath += st
11    border = border(n)
12    p = startPrime
13    if onlyR != None and p**onlyR > border: return
14
15    gen = runGenerator(border,[startPrime,stopPrime],[onlyR,onlyR])
16    pool = Pool(cpuNum)
17    for p,r,n,(x,pol) in pool.imap( findAnyPCN_polynom__star, \
18            ((p,r,n) for p,r in gen) ):
19        print "(",p,",␣",r,")␣=␣", pol
20        if fileoutput:
21            with open(filepath,'a') as f:
22                f.write(str(p)+"\t"+str(r)
23                       +"\t"+str(pol)+"\n")
24            f.close();
25    pool.close()
26    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 immer noch 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`

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

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

```
1 def runGenerator(border,pRange=None,rRange=None):
2     if pRange == None:
3         p = 1
4     else:
5         p = pRange[0]
6     while p < border :
7         p = next_prime(p)
8         if pRange != None and p > pRange[1]: return
9         # consider only rs in rRange
10        if rRange != None:
11            for r in xrange(rRange[0],rRange[1]+1):
12                if p**r > border: break
13                yield p,r
```

```
14          # consider all rs
15          else:
16              r = 1
17              q = p**r
18              while q < border:
19                  yield p,r
20                  r += 1
21                  q = p**r
```

In den Tabellen der angehängten CD findet man die Ergebnisse unsere computergestützten Suche. (Die genaue Syntax der `csv`-Dateien wird später beschrieben.)

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 6.39). 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()` oder übergeben bereits ein bekanntes primitives Element durch das Argument `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`

```
1  def findAnyPCN(p,r,n, primitive_element=None):
2      q = p**r
3      F = GF(q,'a')
4
5      E = F.extension(n,'a')
6      P = E.prime_subfield()
7
8      Px = PolynomialRing(P,'x')
9      Fx = PolynomialRing(F,'x')
10     Ex = PolynomialRing(E,'x')
11     h = Hom(F,E)[0]
12     primOrder = E.order()-1
13
14     primitives = []
15
16     #setup factors of x^n-1
17     divs = get_proper_subfield_divisors(p,r,n)
18     facsAll = dict();
19     prodsAll = dict();
20     fieldsAll = dict();
21     for d in divs:
22         G = F.extension(Integer(d), 'c');
23         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
    if primitive_element == None:
        x = E.primitive_element()
    else:
        x = primitive_element
        E = primitive_element.parent()

    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.map_coefficients(h)(y) == E.zero():
                    return f
```

Die „unschönen“ Ergebnisse dieser Funktion wurden in den Tabellen speziell gekennzeichnet.

6.7.4 Auswertung der Ergebnisse

Mit Hilfe der hier vorgestellten Funktionen zur Findung von $\mathcal{PCN}$-Polynomen und dem asymptotischen Resultat Satz 6.15 von Hachenberger konnte nun bewiesen werden, dass für alle $n \in \{2, 3, \ldots, 33\}$ und für alle Primzahlpotenzen q ein primitiv vollständig normales Polynom von Grad n über $\mathbb{F}_q$ existiert (Satz 6.36). Dies stellt in der Tat eine Neuerung in der Forschung über die Existenz von primitiv vollständig normalen Elementen in Erweiterungen endlicher Körper dar und trägt letztendlich dazu bei, einen Teil der Antwort auf die große Frage der generellen Existenz von $\mathcal{PCN}$-Elementen zu liefern.

In der angehängten CD befinden sich im Ordner `Tables/PCNs` die Ergebnisse der $\mathcal{PCN}$-Suche. Die Dateinamen entsprechen dem Schema `pcns_`n`_`r`.csv` und ein Eintrag p`, poly, modulus` bedeutet, dass `poly` ein primitiv vollständig normales Polynom von Grad n über

$$\mathbb{F}_{p^r} \cong \mathbb{F}_p[a]/(\texttt{modulus})$$

ist. `modulus` entfällt selbstredend, falls $r = 1$ gilt. Die Primzahlen p sind innerhalb einer Datei in aufsteigender Reihenfolge sortiert. Für die Ergebnisse der Funktion `findAnyPCN` aus Unterabschnitt 6.7.3 wurde an die Primzahl p ein `!` angehängt, um herauszustellen, dass diese Polynome nicht die kleinsten im Sinne der gewählten Ordnung (Definition 6.39) sind.

Da Morgan und Mullen ihre Suche nach $\mathcal{PCN}$-Polynomen keinem konkreten Ziel widmeten (wir wollten Satz 6.36 beweisen), präsentiert ihre Arbeit für jede Primzahl $p \leq 97$ und jede ganze Zahl $n \geq 2$ mit $p^n < 10^{50}$ ein $\mathcal{PCN}$-Polynom. Auch diesen Bereich konnten wir erweitern und geben

auf beiliegender CD (vgl. Anhang B.1) für $r = 1$ und alle Primzahlen $p < 1000$ mit $p^n < 10^{70}$ primitiv vollständig normale Polynome von Grad n über $\mathbb{F}_{p^r}$ an.

Wir schließen die vorliegende Arbeit mit einer interessanten Entdeckung, die bei Betrachtung der berechneten $\mathcal{PCN}$-Polynome besonders auffällig erscheint: Für sehr viele Erweiterungen existieren primitiv vollständig normale Trinome. Inbesondere kann man erkennen, dass für konstantes n und r, wenn p 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 6.41.

Seien $n \in \mathbb{N}^*$ und $r \in \mathbb{N}^*$ beliebig. Dann existiert ein $P_{n,r} \in \mathbb{N}^*$, so dass für alle Primzahlen $p \geq P_{n,r}$ ein primitiv vollständig normales Trinom von Grad n über $\mathbb{F}_{p^r}$ existiert.

Fazit und Ausblick

Im Verlauf der vorliegenden Arbeit konnten wir erkennen, wie sich die Modulstrukturen in Erweiterungen endlicher Körper sehr gut nutzen lassen, um eine ausgezeichnete Beschreibung der Bedingungen zu erhalten, die ein (vollständig) normales Element erfüllen muss. Genauer gesagt konnten wir erkennen, dass der Zerfall des Polynoms $x^n - 1$ über einem endlichen Körper $\mathbb{F}_q$ in direkter Korrespondenz mit der Zerlegung der Modulstruktur von $\mathbb{F}_{q^n}$ in direkte Summen von $\mathbb{F}_q$-Moduln steht. Wir erinnern uns auch daran, dass in diesem Kontext für (stark) reguläre Erweiterungen gewisse primitive Einheitswurzeln prototypische Erzeuger im Sinne der Modulstruktur und damit auch für normale Elemente darstellen. Am Ende des vierten Kapitels waren wir so in der Lage, die Resultate aus [26, 27] in Termen dieser Theorie zu beschreiben und konnten auf diese Weise eine neue Beweismöglichkeit der dort gezeigten Aussagen aufzeigen. Insbesondere waren wir fähig eine Rechtfertigung zu liefern, warum die Dickson-Polynome in diesem Kontext geeignet sind, um explizit normale Elemente zu beschreiben.

Im anschließenden experimentellen Teil der Arbeit, den wir der Enumeration (primitiv) vollständig normaler Elemente gewidmet haben, konnten wir durch die intensive Nutzung des Zerlegungssatzes (Satz 5.10) und der damit verbundenen Aussagen von Hachenberger einerseits die Ergebnisse der bereits durchgeführten computergestützten Enumerationen durch Morgan und Mullen [24] aus dem Jahr 1996 verifizieren und andererseits bislang unbekannte Werte, d.h. solche, die weder aus theoretischen Betrachtungen noch aus obig genannten Enumerationen hervorgehen, zu Tage fördern und so ein wenig mehr Licht in das Dunkel der Anzahlen (primitiv) vollständig normaler Elemente bringen.

Es wurde aufgezeigt, dass neben der Anzahl der (primitiv) vollständig normalen Elemente selbst die Frage nach der Existenz von $\mathcal{PCN}$-Elementen im Allgemeinen ungeklärt ist. Daher war es Ziel des letzten Teils dieser Arbeit, mit Hilfe der asymptotischen Existenz, die Hachenberger in [9] präsentieren konnte, und einer computergestützten Suche für die Restfälle, für möglichst viele Erweiterungsgrade die Existenz eines $\mathcal{PCN}$-Elements für *alle* Primzahlpotenzen q belegen zu können. Leider sind es derer nur 12 Grade, für die wir diese Existenzfrage computergestützt beantworten konnten, da die Anzahl der Körpererweiterungen, für die ein primitiv vollständig normales Elemente anzugeben ist, für $n = 30$ (das größte hier betrachtete n) bereits bei 64902 liegt. Für den nächstgrößeren zu testenden Erweiterungsgrad $n = 34$ war es nicht mehr möglich im zeitlichen Rahmen dieser Arbeit die 102852 Körpererweiterungen abzuarbeiten. Zusammengefasst konnten wir mit Hilfe der von Hachenberger bewiesenen asymptotischen Resultate die Existenz von primitiv vollständig normalen Elementen in allen Körpererweiterungen $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ für beliebige Primzahlpotenzen q und $2 \leq n \leq 33$ sichern. Konkret geben wir für die computergestützt verarbeiteten Körpererweiterungen primitiv vollständig normale Polynome an, d.h. irreduzible Polynome über $\mathbb{F}_q$, deren Nullstellen in $\mathbb{F}_{q^n}$ primitiv und vollständig normal über $\mathbb{F}_q$

sind. Die Ordnung, in welcher die Polynome auf diese Eigenschaften untersucht wurden, bevorzugte bewusst solche, deren Hamming-Gewicht klein ist, also die wenige nicht verschwindende Koeffizienten vorweisen. Ein besonders auffälliges Faktum ist dabei, dass sich für fast alle getesteten Körpererweiterungen ein primitiv vollständig normales *Trinom* finden ließ, also ein Polynom, bei dem lediglich drei Koeffizienten nicht verschwinden (wobei selbst die Positionen der Koeffizienten – wie in Lemma 6.38 gezeigt – genau festgelegt sind, d.h. jedes $\mathcal{PCN}$-Trinom für (q, n) ist von der Form $x^n + a_{n-1}x^{n-1} + a_0$). Insbesondere lässt sich eine asymptotische Existenz beobachten, d.h. ist nur q groß genug, so existierte stets ein primitiv vollständig normales Trinom. Dies führte uns zu einer Vermutung, die bisher noch in keiner Literatur (weder als Vermutung noch als bewiesenes Resultat) gefunden werden konnte: Die Existenz eines primitiv vollständig normalen Trinoms bei gegebenem n und r für alle Körpererweiterungen $\mathbb{F}_{(p^r)^n}$ über $\mathbb{F}_{p^r}$, wenn nur p groß genug ist. Leider bildet dies auch das Ende dieser Arbeit und es muss bei einer *Vermutung* bleiben, da im vorgegebenen Zeitrahmen weder eine begründende Idee noch ein Ansatz für einen Beweis dieser Vermutung ausgearbeitet werden konnte. Abschließend bleibt zu hoffen, dass sich in Zukunft jemand dieser äußerst interessanten Hypothese annimmt und möglicherweise einen Beweis (oder ein Gegenbeispiel) zu Tage fördern kann.

Listings

Literatur

[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**, 1997, 227–249.

[2] D. Blessenohl und K. Johnsen. »Eine Verschärfung des Satzes von der Normalbasis«. *Journal of Algebra* **103**, 1986, 141–159.

[3] S. Bosch. *Algebra*. Springer-Verlag, Berlin, 2009.

[4] L. Carlitz. »Primitive roots in a finite field«. *Trans. Amer. Math. Soc.* **73**, 1952, 373–382.

[5] H. Davenport. »Bases for finite fields«. *J. London Math. Soc.* **43**, 1968, 21–39.

[6] G. Eisenstein. *Journal für die reine und angewandte Mathematik*. **39**, 1850, 180–182. *Mathematische Werke, Vol. 2*. Chelsea, New York, 1975, 620–622.

[7] S. Gao und G. Mullen. »Dickson Polynomials and Irreducible Polynomials Over Finite Fields«. *Journal of Number Theory* **49**, 1994, 118–132.

[8] D. Hachenberger. *Finite Fields. Normal Bases and Completely Free Elements*. Kluwer Academic Publishers, Boston, 1997.

[9] D. Hachenberger. »Asymptotic existence results for primitive completely normal elements in Galois field extensions«. Artikel, zur Veröffentlichung eingereicht, 2014.

[10] D. Hachenberger. »Completely normal bases«. In: *Handbook of Finite Fields*. Hrsg.: G. Mullen und D. Panario. CRC Press, Boca Raton, 2013. Kap. 5.4, 128–138.

[11] D. Hachenberger. *Endliche Körper II*. Vorlesungsmitschrift. Sommersemester 2013.

[12] D. Hachenberger. »Primitive complete normal bases: Existence in certain 2-power extensions and lower bounds«. *Discrete Mathematics* **310**, 2010, 3246–3250.

[13] D. Hachenberger. »Primitive complete normal bases for regular extensions«. *Glasgow Mathematical Journal* **43**, 2001, 383–398.

[14] D. Hachenberger. »Primitive complete normal bases for regular extensions: Exceptional cyclotomic modules«. Artikel, zur Veröffentlichung eingereicht, 2014.

[15] D. Hachenberger und D. Jungnickel. *Topics in Galois Fields*. Lehrbuch in Vorbereitung, Version vom 06.03.2014. Erscheint bei Springer.

[16] B. Hartley und T. Hawkes. *Rings, Modules and Linear Algebra*. Chapman and Hall, London, 1983.

[17] K. Hensel. »Über die Darstellung der Zahlen eines Gattungsbereiches für einen beliebigen Primdivisor«. *Journal für die reine und angewandte Mathematik* **103**, 1888, 230–237.

[18] S. Huczynska. »Existence Results for Finite Field Polynomials with Specified Properties«. In: *Finite Fields and Their Applications: Character Sums and Polynomials*. Hrsg.: P. Charpin, A. Pott und A. Winterhof. De Gruyter, Berlin, 2013, 65–88.

[19] C. Karpfinger und K. Meyberg. *Algebra. Gruppen - Ringe - Körper*. Spektrum Akademischer Verlag, Heidelberg, 2010.

[20] S. Lang. *Algebra*. Springer-Verlag, New York, 2002.

[21] H. W. Lenstra, Jr. und R. J. Schoof. »Primitive Normal Bases for Finite Fields«. *Mathematics of Computation* **48**, 1987, 217–231.

[22] R. Lidl, G. Mullen und G. Turnwald. *Dickson polynomials*. 65. Longman Scientific & Technical, Harlow, 1993.

[23] R. Lidl und H. Niederreiter. *Finite Fields*. 20. Cambridge University Press, Cambridge, 1997.

[24] I. Morgan und G. Mullen. »Completely Normal Primitive Basis Generators Of Finite Fields«. *Utilitas Mathematica* **49**, 1996, 21–43.

[25] G. Mullen und D. Panario (Hrsg.) *Handbook of Finite Fields*. CRC Press, Boca Raton, 2013.

[26] A. Scheerhorn. »Dickson Polynomials and Completely Normal Elements over Finite Fields«. In: *Applications of Finite Fields*. Oxford University Press, New York, 1996, 47–55.

[27] A. Scheerhorn. »Dickson Polynomials, Completely Normal Polynomials and the Cyclic Module Structure of Specific Extensions of Finite Fields«. *Des. Codes Cryptography* **9**, 1996, 193–202.

[28] I. A. Semaev. »Construction of polynomials irreducible over a finite field with linearly independent roots«. *Mathematics of the USSR-Sbornik* **63**, 1989, 507.

[29] Z. Wan. *Lectures on Finite Fields and Galois Rings*. World Scientific, Singapur, 2003.

Anhang A
Tabellen

A.1 | Enumerationen

Im Folgenden stellen wir die mit Hilfe der vorgestellten Algorithmen berechneten Werte vor. Dabei ist folgende Legende zu beachten:

q, p, r sind die Daten des betrachteten Grundkörpers $\mathbb{F}_q$, wobei $q = p^r$ gilt.

$\mathcal{N}(q,n)$ gibt die Anzahl der normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

$\mathcal{CN}(q,n)$ gibt die Anzahl der vollständig normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

$\mathcal{PCN}(q,n)$ gibt die Anzahl der primitiv vollständig normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an. Ist in dieser Spalte ein – vorzufinden, so konzentrierte sich diese Berechnung lediglich auf die Daten für $\mathcal{CN}(q,n)$.

$\mathcal{PN}(q,n)$ gibt die Anzahl der primitiv normalen Elemente der Erweiterung von Grad n über $\mathbb{F}_q$ an.

Erzeuger. Hier ist die Anzahl der vollständigen Erzeuger der Zerlegung nach Satz 5.10 gegeben, wobei ein Datum $(k,t,\pi) : N$ bedeutet, dass für den Kreisteilungsmodul $\mathcal{C}_{k,t\pi}$ gerade N vollständige Erzeuger in $\mathbb{F}_{q^n}$ über $\mathbb{F}_q$ existieren.

$(\ .\ )^*$ gibt bei Vorhandensein in der Spalte $\mathcal{CN}(q,n)$ an, ob die aktuelle Körpererweiterung einfach (Definition 5.2) ist. Falls ja, so gilt per Definition $\mathcal{CN}(q,n) = \mathcal{N}(q,n)$ und $\mathcal{PCN}(q,n) = \mathcal{PN}(q,n)$. Daher sind in den Tabellen mit (primitiv) normalen Elementen lediglich Erweiterungen gelistet, die nicht einfach sind.

$(\ .\ )^\dagger$ gibt bei Vorhandensein hinter einem Erzeuger-Datum an, ob dieser regulär ist (Definition 5.11).

Tabelle A.1: Enumerationen $p = 2$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	2	2^*	2	$(1,1,2)^\dagger$: 2
2	2	1	3	3^*	3	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3
2	2	1	4	8^*	4	$(1,1,4)^\dagger$: 8
2	2	1	5	15^*	15	$(1,1,1)^\dagger$: 1, $(5,1,1)^\dagger$: 15

Tabelle A.1: Enumerationen $p = 2$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	6	12	6	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6
2	2	1	7	49*	49	$(1,1,1)^\dagger$: 1, $(7,1,1)^\dagger$: 49
2	2	1	8	128*	56	$(1,1,8)^\dagger$: 128
2	2	1	9	189*	171	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(9,1,1)^\dagger$: 63
2	2	1	10	420	250	$(1,1,2)^\dagger$: 2, $(5,1,2)$: 210
2	2	1	11	1023*	957	$(1,1,1)^\dagger$: 1, $(11,1,1)^\dagger$: 1023
2	2	1	12	768	360	$(1,1,4)^\dagger$: 8, $(3,1,4)$: 96
2	2	1	13	4095*	4095	$(1,1,1)^\dagger$: 1, $(13,1,1)^\dagger$: 4095
2	2	1	14	6272*	4074	$(1,1,2)^\dagger$: 2, $(7,1,2)^\dagger$: 3136
2	2	1	15	10125*	8430	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(5,1,1)^\dagger$: 15, $(15,1,1)^\dagger$: 225
2	2	1	16	32768*	16272	$(1,1,16)^\dagger$: 32768
2	2	1	17	65025*	65025	$(1,1,1)^\dagger$: 1, $(17,1,1)^\dagger$: 65025
2	2	1	18	46872	24948	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6, $(9,1,2)$: 3906
2	2	1	19	262143*	262143	$(1,1,1)^\dagger$: 1, $(19,1,1)^\dagger$: 262143
2	2	1	20	329280	150320	$(1,1,4)^\dagger$: 8, $(5,1,4)$: 61440
2	2	1	21	259308	220374	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(7,3,1)$: 194481
2	2	1	22	2091012	1317250	$(1,1,2)^\dagger$: 2, $(11,1,2)$: 1047552
2	2	1	23	4190209*	4099957	$(1,1,1)^\dagger$: 1, $(23,1,1)^\dagger$: 4190209
2	2	1	24	3145728	1246752	$(1,1,8)^\dagger$: 128, $(3,1,8)$: 49152
2	2	1	25	15728625*	15188050	$(1,1,1)^\dagger$: 1, $(5,1,1)^\dagger$: 15, $(25,1,1)^\dagger$: 1048575
2	2	1	26	33529860	22345232	$(1,1,2)^\dagger$: 2, $(13,1,2)$: 16764930
2	2	1	27	47258883	39950874	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3, $(9,1,1)^\dagger$: 63, $(27,1,1)^\dagger$: 250047
2	2	1	28	102760448*	50821260	$(1,1,4)^\dagger$: 8, $(7,1,4)^\dagger$: 12845056
2	2	1	29	268435455*	266908663	$(1,1,1)^\dagger$: 1, $(29,1,1)^\dagger$: 268435455
2	2	1	30	111132000	55308540	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 12, $(5,1,2)$: 240, $(15,1,2)$: 57600
2	2	1	31	887503681*	887503681	$(1,1,1)^\dagger$: 1, $(31,1,1)^\dagger$: 887503681
4	2	2	2	12*	8	$(1,1,2)^\dagger$: 12
4	2	2	3	27*	18	$(1,1,1)^\dagger$: 3, $(3,1,1)^\dagger$: 9
4	2	2	4	192*	96	$(1,1,4)^\dagger$: 192
4	2	2	5	675*	400	$(1,1,1)^\dagger$: 3, $(5,1,1)^\dagger$: 225
4	2	2	6	1728*	792	$(1,1,2)^\dagger$: 12, $(3,1,2)^\dagger$: 144
4	2	2	7	11907*	7784	$(1,1,1)^\dagger$: 3, $(7,1,1)^\dagger$: 3969
4	2	2	8	49152*	24448	$(1,1,8)^\dagger$: 49152
4	2	2	9	107163*	57186	$(1,1,1)^\dagger$: 3, $(3,1,1)^\dagger$: 9, $(9,1,1)^\dagger$: 3969
4	2	2	10	529200	241400	$(1,1,2)^\dagger$: 12, $(5,1,2)$: 57600
4	2	2	11	3139587*	1978020	$(1,1,1)^\dagger$: 3, $(11,1,1)^\dagger$: 1046529
4	2	2	12	7077888*	2803392	$(1,1,4)^\dagger$: 192, $(3,1,4)^\dagger$: 36864
4	2	2	13	50307075*	33525908	$(1,1,1)^\dagger$: 3, $(13,1,1)^\dagger$: 16769025
4	2	2	14	195084288*	96481224	$(1,1,2)^\dagger$: 12, $(7,1,2)^\dagger$: 16257024
8	2	3	2	56*	36	$(1,1,2)^\dagger$: 56

Tabelle A.1: Enumerationen $p = 2$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
8	2	3	3	441*	378	$(1,1,1)^\dagger$: 7, $(3,1,1)^\dagger$: 63
8	2	3	4	3584*	1512	$(1,1,4)^\dagger$: 3584
8	2	3	5	28665*	23760	$(1,1,1)^\dagger$: 7, $(5,1,1)^\dagger$: 4095
8	2	3	6	218736	117288	$(1,1,2)^\dagger$: 56, $(3,1,2)$: 3906
8	2	3	7	823543*	698544	$(1,1,1)^\dagger$: 7, $(7,1,1)^\dagger$: 117649
8	2	3	8	14680064*	5804640	$(1,1,8)^\dagger$: 14680064
8	2	3	9	110270727*	93223872	$(1,1,1)^\dagger$: 7, $(3,1,1)^\dagger$: 63, $(9,1,1)^\dagger$: 250047
16	2	4	2	240*	128	$(1,1,2)^\dagger$: 240
16	2	4	3	3375*	1440	$(1,1,1)^\dagger$: 15, $(3,1,1)^\dagger$: 225
16	2	4	4	61440*	30720	$(1,1,4)^\dagger$: 61440
16	2	4	5	759375*	346400	$(1,1,1)^\dagger$: 15, $(5,1,1)^\dagger$: 50625
16	2	4	6	13824000*	5469696	$(1,1,2)^\dagger$: 240, $(3,1,2)^\dagger$: 57600
16	2	4	7	251535375*	124407360	$(1,1,1)^\dagger$: 15, $(7,1,1)^\dagger$: 16769025
32	2	5	2	992*	600	$(1,1,2)^\dagger$: 992
32	2	5	3	31713*	26100	$(1,1,1)^\dagger$: 31, $(3,1,1)^\dagger$: 1023
32	2	5	4	1015808*	465000	$(1,1,4)^\dagger$: 1015808
32	2	5	6	1037141952	516358800	$(1,1,2)^\dagger$: 992, $(3,1,2)$: 1045506
64	2	6	3	250047*	134136	$(1,1,1)^\dagger$: 63, $(3,1,1)^\dagger$: 3969
64	2	6	4	16515072*	6531840	$(1,1,4)^\dagger$: 16515072
128	2	7	3	2080641*	1764882	$(1,1,1)^\dagger$: 127, $(3,1,1)^\dagger$: 16383
128	2	7	4	266338304*	131721408	$(1,1,4)^\dagger$: 266338304
256	2	8	3	16581375*	6561792	$(1,1,1)^\dagger$: 255, $(3,1,1)^\dagger$: 65025
512	2	9	3	133955073*	113245776	$(1,1,1)^\dagger$: 511, $(3,1,1)^\dagger$: 262143

Tabelle A.2: Enumerationen $p = 2$

q	p	r	n	$\mathcal{N}(q,n)$	$\mathcal{PN}(q,n)$
2	2	1	6	24	18
2	2	1	10	480	290
2	2	1	12	1536	624
2	2	1	18	96768	51660
2	2	1	20	491520	225100
2	2	1	21	583443	495159
2	2	1	22	2095104	1319692
2	2	1	24	6291456	2488320
2	2	1	26	33546240	22356074
2	2	1	27	49545027	41883129
2	2	1	30	331776000	165178980
4	2	2	10	691200	316760

Tabelle A.3: Enumerationen $p = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
3	3	1	2	4*	4	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2

Tabelle A.3: Enumerationen $p = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
3	3	1	3	18*	9	$(1,1,3)^\dagger$: 18
3	3	1	4	32*	16	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8
3	3	1	5	160*	75	$(1,1,1)^\dagger$: 2, $(5,1,1)^\dagger$: 80
3	3	1	6	324*	144	$(1,1,3)^\dagger$: 18, $(2,1,3)^\dagger$: 18
3	3	1	7	1456*	728	$(1,1,1)^\dagger$: 2, $(7,1,1)^\dagger$: 728
3	3	1	8	1536	576	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(8,1,1)^\dagger$: 48
3	3	1	9	13122*	6075	$(1,1,9)^\dagger$: 13122
3	3	1	10	24960	11160	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(5,2,1)$: 6240
3	3	1	11	117128*	55979	$(1,1,1)^\dagger$: 2, $(11,1,1)^\dagger$: 58564
3	3	1	12	209952*	65424	$(1,1,3)^\dagger$: 18, $(2,1,3)^\dagger$: 18, $(4,1,3)^\dagger$: 648
3	3	1	13	913952*	456976	$(1,1,1)^\dagger$: 2, $(13,1,1)^\dagger$: 456976
3	3	1	14	2114112	1054368	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(7,2,1)$: 529984
3	3	1	15	9447840*	3962700	$(1,1,3)^\dagger$: 18, $(5,1,3)^\dagger$: 524880
3	3	1	16	6291456	2289984	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(8,1,1)^\dagger$: 64, $(16,1,1)^\dagger$: 6400
3	3	1	17	86093440*	43022053	$(1,1,1)^\dagger$: 2, $(17,1,1)^\dagger$: 43046720
3	3	1	18	172186884*	62696736	$(1,1,9)^\dagger$: 13122, $(2,1,9)^\dagger$: 13122
3	3	1	19	774840976*	387177364	$(1,1,1)^\dagger$: 2, $(19,1,1)^\dagger$: 387420488
3	3	1	20	1184481280	423266160	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8, $(5,4,1)$: 37015040
3	3	1	21	6935383728	–	$(1,1,3)^\dagger$: 18, $(7,1,3)$: 385299096
3	3	1	22	13718968384*	–	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(11,1,1)^\dagger$: 58564, $(22,1,1)^\dagger$: 58564
9	3	2	2	64*	32	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8
9	3	2	3	648*	264	$(1,1,3)^\dagger$: 648
9	3	2	4	4096*	1536	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8, $(4,1,1)^\dagger$: 64
9	3	2	5	51200*	23000	$(1,1,1)^\dagger$: 8, $(5,1,1)^\dagger$: 6400
9	3	2	6	419904*	130848	$(1,1,3)^\dagger$: 648, $(2,1,3)^\dagger$: 648
9	3	2	7	4239872*	2115008	$(1,1,1)^\dagger$: 8, $(7,1,1)^\dagger$: 529984
9	3	2	8	16777216*	6117376	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8, $(4,1,1)^\dagger$: 64, $(8,1,1)^\dagger$: 4096
9	3	2	9	344373768*	125421768	$(1,1,9)^\dagger$: 344373768
27	3	3	2	676*	288	$(1,1,1)^\dagger$: 26, $(2,1,1)^\dagger$: 26
27	3	3	3	18954*	8748	$(1,1,3)^\dagger$: 18954
27	3	3	4	492128*	154368	$(1,1,1)^\dagger$: 26, $(2,1,1)^\dagger$: 26, $(4,1,1)^\dagger$: 728
27	3	3	6	359254116*	130838112	$(1,1,3)^\dagger$: 18954, $(2,1,3)^\dagger$: 18954
81	3	4	3	524880*	163584	$(1,1,3)^\dagger$: 524880
81	3	4	4	40960000*	14962688	$(1,1,1)^\dagger$: 80, $(2,1,1)^\dagger$: 80, $(4,1,1)^\dagger$: 6400
243	3	5	3	14289858*	5994450	$(1,1,3)^\dagger$: 14289858
243	3	5	4	3458087072*	1235872000	$(1,1,1)^\dagger$: 242, $(2,1,1)^\dagger$: 242, $(4,1,1)^\dagger$: 59048
729	3	6	3	386889048*	140901120	$(1,1,3)^\dagger$: 386889048

Tabelle A.4: Enumerationen $p = 3$

q	p	r	n	$\mathcal{N}(q,n)$	$\mathcal{PN}(q,n)$
3	3	1	8	2048	832
3	3	1	10	25600	11520
3	3	1	14	2119936	1057392
3	3	1	16	13107200	4790656
3	3	1	20	1310720000	468392880

Tabelle A.5: Enumerationen $p = 5$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
5	5	1	2	$16^\star$	8	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4
5	5	1	3	$96^\star$	48	$(1,1,1)^\dagger$: 4, $(3,1,1)^\dagger$: 24
5	5	1	4	$256^\star$	64	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(4,1,1)^\dagger$: 16
5	5	1	5	$2500^\star$	1130	$(1,1,5)^\dagger$: 2500
5	5	1	6	8448	2376	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(3,2,1)$: 528
5	5	1	7	$62496^\star$	31248	$(1,1,1)^\dagger$: 4, $(7,1,1)^\dagger$: 15624
5	5	1	8	$147456^\star$	44928	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(4,1,1)^\dagger$: 16, $(8,1,1)^\dagger$: 576
5	5	1	9	$1499904^\star$	687132	$(1,1,1)^\dagger$: 4, $(3,1,1)^\dagger$: 24, $(9,1,1)^\dagger$: 15624
5	5	1	10	$6250000^\star$	1862760	$(1,1,5)^\dagger$: 2500, $(2,1,5)^\dagger$: 2500
5	5	1	11	$39037504^\star$	19518752	$(1,1,1)^\dagger$: 4, $(11,1,1)^\dagger$: 9759376
5	5	1	12	71368704	18178944	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(3,2,1)$: 576, $(4,1,1)^\dagger$: 16, $(12,1,1)$: 576
25	5	2	2	$576^\star$	192	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24
25	5	2	3	$13824^\star$	3888	$(1,1,1)^\dagger$: 24, $(3,1,1)^\dagger$: 576
25	5	2	4	$331776^\star$	101376	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24, $(4,1,1)^\dagger$: 576
25	5	2	5	$9375000^\star$	2796400	$(1,1,5)^\dagger$: 9375000
25	5	2	6	$191102976^\star$	48691008	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24, $(3,1,1)^\dagger$: 576, $(6,1,1)^\dagger$: 576
125	5	3	3	$1937376^\star$	887220	$(1,1,1)^\dagger$: 124, $(3,1,1)^\dagger$: 15624
125	5	3	4	$236421376^\star$	60235200	$(1,1,1)^\dagger$: 124, $(2,1,1)^\dagger$: 124, $(4,1,1)^\dagger$: 15376
625	5	4	3	$242970624^\star$	61910784	$(1,1,1)^\dagger$: 624, $(3,1,1)^\dagger$: 389376

Tabelle A.6: Enumerationen $p = 5$

q	p	r	n	$\mathcal{N}(q,n)$	$\mathcal{PN}(q,n)$
5	5	1	6	9216	2568
5	5	1	12	84934656	21645024

Tabelle A.7: Enumerationen $p = 7$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
7	7	1	2	$36^\star$	16	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6

Tabelle A.7: Enumerationen $p = 7$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
7	7	1	3	216*	72	$(1,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36
7	7	1	4	1728*	480	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(4,1,1)^\dagger$: 48
7	7	1	5	14400*	4800	$(1,1,1)^\dagger$: 6, $(5,1,1)^\dagger$: 2400
7	7	1	6	46656*	14832	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36, $(6,1,1)^\dagger$: 36
7	7	1	7	705894*	227010	$(1,1,7)^\dagger$: 705894
7	7	1	8	3815424	1016320	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(4,1,1)^\dagger$: 48, $(8,1,1)^\dagger$: 2208
7	7	1	9	25264224*	7753806	$(1,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36, $(9,1,1)^\dagger$: 116964
7	7	1	10	207187200	62435920	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(5,2,1)$: 5755200
7	7	1	11	1694851488*	564443264	$(1,1,1)^\dagger$: 6, $(11,1,1)^\dagger$: 282475248
49	7	2	3	110592*	34272	$(1,1,1)^\dagger$: 48, $(3,1,1)^\dagger$: 2304
49	7	2	4	5308416*	1413120	$(1,1,1)^\dagger$: 48, $(2,1,1)^\dagger$: 48, $(4,1,1)^\dagger$: 2304
343	7	3	3	40001688*	12279276	$(1,1,1)^\dagger$: 342, $(3,1,1)^\dagger$: 116964

Tabelle A.8: Enumerationen $p = 11$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
11	11	1	2	100*	32	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10
11	11	1	3	1200*	384	$(1,1,1)^\dagger$: 10, $(3,1,1)^\dagger$: 120
11	11	1	4	12000*	3200	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10, $(4,1,1)^\dagger$: 120
11	11	1	5	100000*	40000	$(1,1,1)^\dagger$: 10, $(5,1,1)^\dagger$: 10000
11	11	1	6	1416000	298848	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10, $(3,2,1)$: 14160
11	11	1	7	17689000*	6911072	$(1,1,1)^\dagger$: 10, $(7,1,1)^\dagger$: 1768900
121	11	2	2	14400*	3840	$(1,1,1)^\dagger$: 120, $(2,1,1)^\dagger$: 120
121	11	2	3	1728000*	364608	$(1,1,1)^\dagger$: 120, $(3,1,1)^\dagger$: 14400
121	11	2	4	207360000*	54374400	$(1,1,1)^\dagger$: 120, $(2,1,1)^\dagger$: 120, $(4,1,1)^\dagger$: 14400

Tabelle A.9: Enumerationen $p = 13$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
13	13	1	2	144*	48	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12
13	13	1	3	1728*	576	$(1,1,1)^\dagger$: 12, $(3,1,1)^\dagger$: 144
13	13	1	4	20736*	4352	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12, $(4,1,1)^\dagger$: 144
13	13	1	5	342720*	114240	$(1,1,1)^\dagger$: 12, $(5,1,1)^\dagger$: 28560
13	13	1	6	2985984*	834048	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12, $(3,1,1)^\dagger$: 144, $(6,1,1)^\dagger$: 144
13	13	1	7	56899584*	18966528	$(1,1,1)^\dagger$: 12, $(7,1,1)^\dagger$: 4741632
169	13	2	2	28224*	6144	$(1,1,1)^\dagger$: 168, $(2,1,1)^\dagger$: 168
169	13	2	3	4741632*	1325376	$(1,1,1)^\dagger$: 168, $(3,1,1)^\dagger$: 28224
169	13	2	4	796594176*	171343872	$(1,1,1)^\dagger$: 168, $(2,1,1)^\dagger$: 168, $(4,1,1)^\dagger$: 28224

Tabelle A.10: Enumerationen $p = 17$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
17	17	1	2	256*	96	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16
17	17	1	3	4608*	2304	$(1,1,1)^\dagger$: 16, $(3,1,1)^\dagger$: 288
17	17	1	4	65536*	16896	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16, $(4,1,1)^\dagger$: 256
17	17	1	5	1336320*	668160	$(1,1,1)^\dagger$: 16, $(5,1,1)^\dagger$: 83520
17	17	1	6	21086208	5546304	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16, $(3,2,1)$: 82368
17	17	1	7	386201088*	193100544	$(1,1,1)^\dagger$: 16, $(7,1,1)^\dagger$: 24137568
289	17	2	2	82944*	21504	$(1,1,1)^\dagger$: 288, $(2,1,1)^\dagger$: 288
289	17	2	3	23887872*	6283008	$(1,1,1)^\dagger$: 288, $(3,1,1)^\dagger$: 82944

Tabelle A.11: Enumerationen $p = 19$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
19	19	1	2	324*	96	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18
19	19	1	3	5832*	1944	$(1,1,1)^\dagger$: 18, $(3,1,1)^\dagger$: 324
19	19	1	4	116640*	31104	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18, $(4,1,1)^\dagger$: 360
19	19	1	5	2332800*	771510	$(1,1,1)^\dagger$: 18, $(5,1,1)^\dagger$: 129600
19	19	1	6	34012224*	7711200	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18, $(3,1,1)^\dagger$: 324, $(6,1,1)^\dagger$: 324
19	19	1	7	846825840*	281869602	$(1,1,1)^\dagger$: 18, $(7,1,1)^\dagger$: 47045880
361	19	2	2	129600*	34560	$(1,1,1)^\dagger$: 360, $(2,1,1)^\dagger$: 360
361	19	2	3	46656000*	10584000	$(1,1,1)^\dagger$: 360, $(3,1,1)^\dagger$: 129600

Tabelle A.12: Enumerationen $p = 23$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
23	23	1	2	484*	160	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22
23	23	1	3	11616*	4440	$(1,1,1)^\dagger$: 22, $(3,1,1)^\dagger$: 528
23	23	1	4	255552*	60640	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22, $(4,1,1)^\dagger$: 528
23	23	1	5	6156480*	2798400	$(1,1,1)^\dagger$: 22, $(5,1,1)^\dagger$: 279840
23	23	1	6	134420352	31821840	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22, $(3,2,1)$: 277728
23	23	1	7	3256254232*	1429085280	$(1,1,1)^\dagger$: 22, $(7,1,1)^\dagger$: 148011556
529	23	2	2	278784*	66560	$(1,1,1)^\dagger$: 528, $(2,1,1)^\dagger$: 528
529	23	2	3	147197952*	34848000	$(1,1,1)^\dagger$: 528, $(3,1,1)^\dagger$: 278784

Tabelle A.13: Enumerationen $p = 29$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
29	29	1	2	784*	192	$(1,1,1)^\dagger$: 28, $(2,1,1)^\dagger$: 28
29	29	1	3	23520*	9180	$(1,1,1)^\dagger$: 28, $(3,1,1)^\dagger$: 840
29	29	1	4	614656*	139776	$(1,1,1)^\dagger$: 28, $(2,1,1)^\dagger$: 28, $(4,1,1)^\dagger$: 784
29	29	1	5	19756800*	8467200	$(1,1,1)^\dagger$: 28, $(5,1,1)^\dagger$: 705600
29	29	1	6	551873280	114307056	$(1,1,1)^\dagger$: 28, $(2,1,1)^\dagger$: 28, $(3,2,1)$: 703920
29	29	1	7	13492928512*	5782683648	$(1,1,1)^\dagger$: 28, $(7,1,1)^\dagger$: 481890304

Tabelle A.13: Enumerationen $p = 29$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
841	29	2	2	705600*	161280	$(1,1,1)^\dagger$: 840, $(2,1,1)^\dagger$: 840
841	29	2	3	592704000*	122760576	$(1,1,1)^\dagger$: 840, $(3,1,1)^\dagger$: 705600

Tabelle A.14: Enumerationen $p = 31$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
31	31	1	2	900*	256	$(1,1,1)^\dagger$: 30, $(2,1,1)^\dagger$: 30
31	31	1	3	27000*	7200	$(1,1,1)^\dagger$: 30, $(3,1,1)^\dagger$: 900
31	31	1	4	864000*	207360	$(1,1,1)^\dagger$: 30, $(2,1,1)^\dagger$: 30, $(4,1,1)^\dagger$: 960
31	31	1	5	24300000*	5895200	$(1,1,1)^\dagger$: 30, $(5,1,1)^\dagger$: 810000
31	31	1	6	729000000*	157394880	$(1,1,1)^\dagger$: 30, $(2,1,1)^\dagger$: 30, $(3,1,1)^\dagger$: 900, $(6,1,1)^\dagger$: 900
961	31	2	2	921600*	221184	$(1,1,1)^\dagger$: 960, $(2,1,1)^\dagger$: 960
961	31	2	3	884736000*	191020032	$(1,1,1)^\dagger$: 960, $(3,1,1)^\dagger$: 921600

Tabelle A.15: Enumerationen $p = 37$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
37	37	1	2	1296*	432	$(1,1,1)^\dagger$: 36, $(2,1,1)^\dagger$: 36
37	37	1	3	46656*	13176	$(1,1,1)^\dagger$: 36, $(3,1,1)^\dagger$: 1296
37	37	1	4	1679616*	420864	$(1,1,1)^\dagger$: 36, $(2,1,1)^\dagger$: 36, $(4,1,1)^\dagger$: 1296
37	37	1	6	2176782336*	548654688	$(1,1,1)^\dagger$: 36, $(2,1,1)^\dagger$: 36, $(3,1,1)^\dagger$: 1296, $(6,1,1)^\dagger$: 1296
1369	37	2	2	1871424*	470016	$(1,1,1)^\dagger$: 1368, $(2,1,1)^\dagger$: 1368

Tabelle A.16: Enumerationen $p = 41$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
41	41	1	2	1600*	384	$(1,1,1)^\dagger$: 40, $(2,1,1)^\dagger$: 40
41	41	1	3	67200*	26880	$(1,1,1)^\dagger$: 40, $(3,1,1)^\dagger$: 1680
41	41	1	4	2560000*	564224	$(1,1,1)^\dagger$: 40, $(2,1,1)^\dagger$: 40, $(4,1,1)^\dagger$: 1600
41	41	1	6	215496704	1028522880	$(1,1,1)^\dagger$: 40, $(2,1,1)^\dagger$: 40, $(3,2,1)$: 2819040
1681	41	2	2	2822400*	623616	$(1,1,1)^\dagger$: 1680, $(2,1,1)^\dagger$: 1680

Tabelle A.17: Enumerationen $p = 43$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
43	43	1	2	1764*	480	$(1,1,1)^\dagger$: 42, $(2,1,1)^\dagger$: 42
43	43	1	3	74088*	21168	$(1,1,1)^\dagger$: 42, $(3,1,1)^\dagger$: 1764
43	43	1	4	3259872*	659712	$(1,1,1)^\dagger$: 42, $(2,1,1)^\dagger$: 42, $(4,1,1)^\dagger$: 1848
43	43	1	5	143589600*	41025600	$(1,1,1)^\dagger$: 42, $(5,1,1)^\dagger$: 3418800

Tabelle A.17: Enumerationen $p = 43$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
43	43	1	6	1194064448*	1304511264	$(1,1,1)^\dagger$: 42, $(2,1,1)^\dagger$: 42, $(3,1,1)^\dagger$: 1764, $(6,1,1)^\dagger$: 1764
1849	43	2	2	3415104*	691200	$(1,1,1)^\dagger$: 1848, $(2,1,1)^\dagger$: 1848

Tabelle A.18: Enumerationen $n = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	3	3*	3	$(1,1,1)^\dagger$: 1, $(3,1,1)^\dagger$: 3
3	3	1	3	18*	9	$(1,1,3)^\dagger$: 18
4	2	2	3	27*	18	$(1,1,1)^\dagger$: 3, $(3,1,1)^\dagger$: 9
5	5	1	3	96*	48	$(1,1,1)^\dagger$: 4, $(3,1,1)^\dagger$: 24
7	7	1	3	216*	72	$(1,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36
8	2	3	3	441*	378	$(1,1,1)^\dagger$: 7, $(3,1,1)^\dagger$: 63
9	3	2	3	648*	264	$(1,1,3)^\dagger$: 648
11	11	1	3	1200*	384	$(1,1,1)^\dagger$: 10, $(3,1,1)^\dagger$: 120
13	13	1	3	1728*	576	$(1,1,1)^\dagger$: 12, $(3,1,1)^\dagger$: 144
16	2	4	3	3375*	1440	$(1,1,1)^\dagger$: 15, $(3,1,1)^\dagger$: 225
17	17	1	3	4608*	2304	$(1,1,1)^\dagger$: 16, $(3,1,1)^\dagger$: 288
19	19	1	3	5832*	1944	$(1,1,1)^\dagger$: 18, $(3,1,1)^\dagger$: 324
23	23	1	3	11616*	4440	$(1,1,1)^\dagger$: 22, $(3,1,1)^\dagger$: 528
25	5	2	3	13824*	3888	$(1,1,1)^\dagger$: 24, $(3,1,1)^\dagger$: 576
27	3	3	3	18954*	8748	$(1,1,3)^\dagger$: 18954
29	29	1	3	23520*	9180	$(1,1,1)^\dagger$: 28, $(3,1,1)^\dagger$: 840
31	31	1	3	27000*	7200	$(1,1,1)^\dagger$: 30, $(3,1,1)^\dagger$: 900
32	2	5	3	31713*	26100	$(1,1,1)^\dagger$: 31, $(3,1,1)^\dagger$: 1023
37	37	1	3	46656*	13176	$(1,1,1)^\dagger$: 36, $(3,1,1)^\dagger$: 1296
41	41	1	3	67200*	26880	$(1,1,1)^\dagger$: 40, $(3,1,1)^\dagger$: 1680
43	43	1	3	74088*	21168	$(1,1,1)^\dagger$: 42, $(3,1,1)^\dagger$: 1764
47	47	1	3	101568*	46596	$(1,1,1)^\dagger$: 46, $(3,1,1)^\dagger$: 2208
49	7	2	3	110592*	34272	$(1,1,1)^\dagger$: 48, $(3,1,1)^\dagger$: 2304
53	53	1	3	146016*	57744	$(1,1,1)^\dagger$: 52, $(3,1,1)^\dagger$: 2808
59	59	1	3	201840*	97440	$(1,1,1)^\dagger$: 58, $(3,1,1)^\dagger$: 3480
61	61	1	3	216000*	52848	$(1,1,1)^\dagger$: 60, $(3,1,1)^\dagger$: 3600
64	2	6	3	250047*	134136	$(1,1,1)^\dagger$: 63, $(3,1,1)^\dagger$: 3969
67	67	1	3	287496*	72000	$(1,1,1)^\dagger$: 66, $(3,1,1)^\dagger$: 4356
71	71	1	3	352800*	120960	$(1,1,1)^\dagger$: 70, $(3,1,1)^\dagger$: 5040
73	73	1	3	373248*	124416	$(1,1,1)^\dagger$: 72, $(3,1,1)^\dagger$: 5184
79	79	1	3	474552*	122040	$(1,1,1)^\dagger$: 78, $(3,1,1)^\dagger$: 6084
81	3	4	3	524880*	163584	$(1,1,3)^\dagger$: 524880
83	83	1	3	564816*	260280	$(1,1,1)^\dagger$: 82, $(3,1,1)^\dagger$: 6888
89	89	1	3	696960*	316800	$(1,1,1)^\dagger$: 88, $(3,1,1)^\dagger$: 7920
97	97	1	3	884736*	294912	$(1,1,1)^\dagger$: 96, $(3,1,1)^\dagger$: 9216
121	11	2	3	1728000*	364608	$(1,1,1)^\dagger$: 120, $(3,1,1)^\dagger$: 14400
125	5	3	3	1937376*	887220	$(1,1,1)^\dagger$: 124, $(3,1,1)^\dagger$: 15624

Tabelle A.18: Enumerationen $n = 3$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
128	2	7	3	2080641*	1764882	$(1,1,1)^\dagger$: 127, $(3,1,1)^\dagger$: 16383
169	13	2	3	4741632*	1325376	$(1,1,1)^\dagger$: 168, $(3,1,1)^\dagger$: 28224
243	3	5	3	14289858*	5994450	$(1,1,3)^\dagger$: 14289858
256	2	8	3	16581375*	6561792	$(1,1,1)^\dagger$: 255, $(3,1,1)^\dagger$: 65025
289	17	2	3	23887872*	6283008	$(1,1,1)^\dagger$: 288, $(3,1,1)^\dagger$: 82944
343	7	3	3	40001688*	12279276	$(1,1,1)^\dagger$: 342, $(3,1,1)^\dagger$: 116964
361	19	2	3	46656000*	10584000	$(1,1,1)^\dagger$: 360, $(3,1,1)^\dagger$: 129600
512	2	9	3	133955073*	113245776	$(1,1,1)^\dagger$: 511, $(3,1,1)^\dagger$: 262143
529	23	2	3	147197952*	34848000	$(1,1,1)^\dagger$: 528, $(3,1,1)^\dagger$: 278784
625	5	4	3	242970624*	61910784	$(1,1,1)^\dagger$: 624, $(3,1,1)^\dagger$: 389376
729	3	6	3	386889048*	140901120	$(1,1,3)^\dagger$: 386889048
841	29	2	3	592704000*	122760576	$(1,1,1)^\dagger$: 840, $(3,1,1)^\dagger$: 705600
961	31	2	3	884736000*	191020032	$(1,1,1)^\dagger$: 960, $(3,1,1)^\dagger$: 921600

Tabelle A.19: Enumerationen $n = 4$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	4	8*	4	$(1,1,4)^\dagger$: 8
3	3	1	4	32*	16	$(1,1,1)^\dagger$: 2, $(2,1,1)^\dagger$: 2, $(4,1,1)^\dagger$: 8
4	2	2	4	192*	96	$(1,1,4)^\dagger$: 192
5	5	1	4	256*	64	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(4,1,1)^\dagger$: 16
7	7	1	4	1728*	480	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(4,1,1)^\dagger$: 48
8	2	3	4	3584*	1512	$(1,1,4)^\dagger$: 3584
9	3	2	4	4096*	1536	$(1,1,1)^\dagger$: 8, $(2,1,1)^\dagger$: 8, $(4,1,1)^\dagger$: 64
11	11	1	4	12000*	3200	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10, $(4,1,1)^\dagger$: 120
13	13	1	4	20736*	4352	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12, $(4,1,1)^\dagger$: 144
16	2	4	4	61440*	30720	$(1,1,4)^\dagger$: 61440
17	17	1	4	65536*	16896	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16, $(4,1,1)^\dagger$: 256
19	19	1	4	116640*	31104	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18, $(4,1,1)^\dagger$: 360
23	23	1	4	255552*	60640	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22, $(4,1,1)^\dagger$: 528
25	5	2	4	331776*	101376	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24, $(4,1,1)^\dagger$: 576
27	3	3	4	492128*	154368	$(1,1,1)^\dagger$: 26, $(2,1,1)^\dagger$: 26, $(4,1,1)^\dagger$: 728
29	29	1	4	614656*	139776	$(1,1,1)^\dagger$: 28, $(2,1,1)^\dagger$: 28, $(4,1,1)^\dagger$: 784
31	31	1	4	864000*	207360	$(1,1,1)^\dagger$: 30, $(2,1,1)^\dagger$: 30, $(4,1,1)^\dagger$: 960
32	2	5	4	1015808*	465000	$(1,1,4)^\dagger$: 1015808
37	37	1	4	1679616*	420864	$(1,1,1)^\dagger$: 36, $(2,1,1)^\dagger$: 36, $(4,1,1)^\dagger$: 1296
41	41	1	4	2560000*	564224	$(1,1,1)^\dagger$: 40, $(2,1,1)^\dagger$: 40, $(4,1,1)^\dagger$: 1600
43	43	1	4	3259872*	659712	$(1,1,1)^\dagger$: 42, $(2,1,1)^\dagger$: 42, $(4,1,1)^\dagger$: 1848
47	47	1	4	4672128*	1036288	$(1,1,1)^\dagger$: 46, $(2,1,1)^\dagger$: 46, $(4,1,1)^\dagger$: 2208
49	7	2	4	5308416*	1413120	$(1,1,1)^\dagger$: 48, $(2,1,1)^\dagger$: 48, $(4,1,1)^\dagger$: 2304
53	53	1	4	7311616*	1794816	$(1,1,1)^\dagger$: 52, $(2,1,1)^\dagger$: 52, $(4,1,1)^\dagger$: 2704
59	59	1	4	11706720*	3014144	$(1,1,1)^\dagger$: 58, $(2,1,1)^\dagger$: 58, $(4,1,1)^\dagger$: 3480
61	61	1	4	12960000*	3340800	$(1,1,1)^\dagger$: 60, $(2,1,1)^\dagger$: 60, $(4,1,1)^\dagger$: 3600
64	2	6	4	16515072*	6531840	$(1,1,4)^\dagger$: 16515072

Tabelle A.19: Enumerationen $n = 4$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
67	67	1	4	19549728*	4453760	$(1,1,1)^\dagger$: 66, $(2,1,1)^\dagger$: 66, $(4,1,1)^\dagger$: 4488
71	71	1	4	24696000*	5644800	$(1,1,1)^\dagger$: 70, $(2,1,1)^\dagger$: 70, $(4,1,1)^\dagger$: 5040
73	73	1	4	26873856*	6279168	$(1,1,1)^\dagger$: 72, $(2,1,1)^\dagger$: 72, $(4,1,1)^\dagger$: 5184
79	79	1	4	37964160*	9345024	$(1,1,1)^\dagger$: 78, $(2,1,1)^\dagger$: 78, $(4,1,1)^\dagger$: 6240
81	3	4	4	40960000*	14962688	$(1,1,1)^\dagger$: 80, $(2,1,1)^\dagger$: 80, $(4,1,1)^\dagger$: 6400
83	83	1	4	46314912*	9351040	$(1,1,1)^\dagger$: 82, $(2,1,1)^\dagger$: 82, $(4,1,1)^\dagger$: 6888
89	89	1	4	59969536*	13620480	$(1,1,1)^\dagger$: 88, $(2,1,1)^\dagger$: 88, $(4,1,1)^\dagger$: 7744
97	97	1	4	84934656*	19390976	$(1,1,1)^\dagger$: 96, $(2,1,1)^\dagger$: 96, $(4,1,1)^\dagger$: 9216
121	11	2	4	207360000*	54374400	$(1,1,1)^\dagger$: 120, $(2,1,1)^\dagger$: 120, $(4,1,1)^\dagger$: 14400
125	5	3	4	236421376*	60235200	$(1,1,1)^\dagger$: 124, $(2,1,1)^\dagger$: 124, $(4,1,1)^\dagger$: 15376
128	2	7	4	266338304*	131721408	$(1,1,4)^\dagger$: 266338304
169	13	2	4	796594176*	171343872	$(1,1,1)^\dagger$: 168, $(2,1,1)^\dagger$: 168, $(4,1,1)^\dagger$: 28224
243	3	5	4	3458087072*	1235872000	$(1,1,1)^\dagger$: 242, $(2,1,1)^\dagger$: 242, $(4,1,1)^\dagger$: 59048

Tabelle A.20: Enumerationen $n = 6$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
2	2	1	6	12	6	$(1,1,2)^\dagger$: 2, $(3,1,2)$: 6
3	3	1	6	324*	144	$(1,1,3)^\dagger$: 18, $(2,1,3)^\dagger$: 18
4	2	2	6	1728*	792	$(1,1,2)^\dagger$: 12, $(3,1,2)^\dagger$: 144
5	5	1	6	8448	2376	$(1,1,1)^\dagger$: 4, $(2,1,1)^\dagger$: 4, $(3,2,1)$: 528
7	7	1	6	46656*	14832	$(1,1,1)^\dagger$: 6, $(2,1,1)^\dagger$: 6, $(3,1,1)^\dagger$: 36, $(6,1,1)^\dagger$: 36
8	2	3	6	218736	117288	$(1,1,2)^\dagger$: 56, $(3,1,2)$: 3906
9	3	2	6	419904*	130848	$(1,1,3)^\dagger$: 648, $(2,1,3)^\dagger$: 648
11	11	1	6	1416000	298848	$(1,1,1)^\dagger$: 10, $(2,1,1)^\dagger$: 10, $(3,2,1)$: 14160
13	13	1	6	2985984*	834048	$(1,1,1)^\dagger$: 12, $(2,1,1)^\dagger$: 12, $(3,1,1)^\dagger$: 144, $(6,1,1)^\dagger$: 144
16	2	4	6	13824000*	5469696	$(1,1,2)^\dagger$: 240, $(3,1,2)^\dagger$: 57600
17	17	1	6	21086208	5546304	$(1,1,1)^\dagger$: 16, $(2,1,1)^\dagger$: 16, $(3,2,1)$: 82368
19	19	1	6	34012224*	7711200	$(1,1,1)^\dagger$: 18, $(2,1,1)^\dagger$: 18, $(3,1,1)^\dagger$: 324, $(6,1,1)^\dagger$: 324
23	23	1	6	134420352	31821840	$(1,1,1)^\dagger$: 22, $(2,1,1)^\dagger$: 22, $(3,2,1)$: 277728
25	5	2	6	191102976*	48691008	$(1,1,1)^\dagger$: 24, $(2,1,1)^\dagger$: 24, $(3,1,1)^\dagger$: 576, $(6,1,1)^\dagger$: 576
27	3	3	6	359254116*	130838112	$(1,1,3)^\dagger$: 18954, $(2,1,3)^\dagger$: 18954
29	29	1	6	551873280	114307056	$(1,1,1)^\dagger$: 28, $(2,1,1)^\dagger$: 28, $(3,2,1)$: 703920
31	31	1	6	729000000*	157394880	$(1,1,1)^\dagger$: 30, $(2,1,1)^\dagger$: 30, $(3,1,1)^\dagger$: 900, $(6,1,1)^\dagger$: 900
32	2	5	6	1037141952	516358800	$(1,1,2)^\dagger$: 992, $(3,1,2)$: 1045506

Tabelle A.20: Enumerationen $n = 6$

q	p	r	n	$\mathcal{CN}(q,n)$	$\mathcal{PCN}(q,n)$	Erzeuger
37	37	1	6	2176782336*	548654688	$(1,1,1)^{\dagger}$: 36, $(2,1,1)^{\dagger}$: 36, $(3,1,1)^{\dagger}$: 1296, $(6,1,1)^{\dagger}$: 1296
41	41	1	6	215496704	1028522880	$(1,1,1)^{\dagger}$: 40, $(2,1,1)^{\dagger}$: 40, $(3,2,1)$: 2819040
43	43	1	6	1194064448*	1304511264	$(1,1,1)^{\dagger}$: 42, $(2,1,1)^{\dagger}$: 42, $(3,1,1)^{\dagger}$: 1764, $(6,1,1)^{\dagger}$: 1764

Anhang B

CD- und Online-Resourcen

B.1 | CD

Auf beiliegender CD findet man die folgenden Ordner:

```
./Tables/Enumerations/
./Tables/PCNs/
./Sage
```

B.1.1 | ./Tables/Enumerations

In analoger Syntax zu Anhang A.1 finden sich in diesem Ordner die Tabellen der Enumerationen als digitale Version wieder. Wie in Anhang A.1 sind die Tabellen dort nach konstantem p (`enumerationsPCN_P_`p`.csv`) und konstantem n (`enumerationsPCN_N_`n`.csv`) separiert. Die Enumerationen, bei denen lediglich die Anzahl normaler und primitiv normaler Elemente bestimmt wurde, liegen in den Dateien `enumerationsPN_P_`p`.csv`.

Die Syntax der Einträge entspricht genau der der entsprechenden Tabellen. Lediglich das Symbol † zur Kennzeichnung von regulären Kreisteilungsmoduln wurde durch $*$ ersetzt.

Es sind folgende Dateien in diesem Ordner enthalten:

`enumerationsPCN_P_2.csv`, `enumerationsPCN_P_3.csv`, `enumerationsPCN_P_5.csv`,
`enumerationsPCN_P_7.csv`, `enumerationsPCN_P_11.csv`, `enumerationsPCN_P_13.csv`,
`enumerationsPCN_P_17.csv`, `enumerationsPCN_P_19.csv`, `enumerationsPCN_P_23.csv`,
`enumerationsPCN_P_29.csv`, `enumerationsPCN_P_31.csv`, `enumerationsPCN_P_37.csv`,
`enumerationsPCN_P_41.csv`, `enumerationsPCN_P_43.csv`

`enumerationsPCN_N_3.csv`, `enumerationsPCN_N_4.csv`, `enumerationsPCN_N_6.csv`

`enumerationsPN_P_2.csv`, `enumerationsPN_P_3.csv`, `enumerationsPN_P_5.csv`

B.1.2 ./Tables/PCNs

In diesem Ordner sind die Ergebnisse der Suche nach primitiv vollständig normalen Polynomen (Abschnitt 6.7) gespeichert. In der Datei `pcns_`n`_`r`.csv` ist für jede Primzahlpotenz $p^r < n^4$ ein primitiv vollständig normales Polynom hinterlegt. Ein Eintrag p`, poly, modulus` bedeutet, dass `poly` ein primitiv vollständig normales Polynom von Grad n über

$$\mathbb{F}_{p^r} = \mathbb{F}_p[a]/(\texttt{modulus})$$

ist. `modulus` entfällt selbstredend, falls $r = 1$ gilt. Die Primzahlen p sind innerhalb einer Datei in aufsteigender Reihenfolge sortiert. Das Polynom ist in den meisten Fällen das kleinste bezüglich der Ordnung aus Definition 6.39. Ist dies bei einem Eintrag nicht der Fall (vgl. Unterabschnitt 6.7.3), so wurde ein `!` der Primzahl p angehängt, um dies kenntlich zu machen.

Es sind folgende Dateien in diesem Ordner enthalten:

```
pcns_6_1.csv, pcns_6_2.csv, pcns_6_3.csv, pcns_6_4.csv, pcns_6_5.csv, pcns_6_6.csv,
pcns_6_7.csv, pcns_6_8.csv, pcns_6_9.csv, pcns_6_10.csv

pcns_10_1.csv, pcns_10_2.csv, pcns_10_3.csv, pcns_10_4.csv, pcns_10_5.csv, pcns_10_6.csv,
pcns_10_7.csv, pcns_10_8.csv, pcns_10_9.csv, pcns_10_10.csv, pcns_10_11.csv,
pcns_10_12.csv, pcns_10_13.csv

pcns_12_1.csv, pcns_12_2.csv, pcns_12_3.csv, pcns_12_4.csv, pcns_12_5.csv, pcns_12_6.csv,
pcns_12_7.csv, pcns_12_8.csv, pcns_12_9.csv, pcns_12_10.csv, pcns_12_11.csv,
pcns_12_12.csv, pcns_12_13.csv, pcns_12_14.csv

pcns_14_1.csv, pcns_14_2.csv, pcns_14_3.csv, pcns_14_4.csv, pcns_14_5.csv, pcns_14_6.csv,
pcns_14_7.csv, pcns_14_8.csv, pcns_14_9.csv, pcns_14_10.csv, pcns_14_11.csv,
pcns_14_12.csv, pcns_14_13.csv, pcns_14_14.csv, pcns_14_15.csv

pcns_15_1.csv, pcns_15_2.csv, pcns_15_3.csv, pcns_15_4.csv, pcns_15_5.csv, pcns_15_6.csv,
pcns_15_7.csv, pcns_15_8.csv, pcns_15_9.csv, pcns_15_10.csv, pcns_15_11.csv,
pcns_15_12.csv, pcns_15_13.csv, pcns_15_14.csv, pcns_15_15.csv

pcns_18_1.csv, pcns_18_2.csv, pcns_18_3.csv, pcns_18_4.csv, pcns_18_5.csv, pcns_18_6.csv,
pcns_18_7.csv, pcns_18_8.csv, pcns_18_9.csv, pcns_18_10.csv, pcns_18_11.csv,
pcns_18_12.csv, pcns_18_13.csv, pcns_18_14.csv, pcns_18_15.csv, pcns_18_16.csv

pcns_20_1.csv, pcns_20_2.csv, pcns_20_3.csv, pcns_20_4.csv, pcns_20_5.csv, pcns_20_6.csv,
pcns_20_7.csv, pcns_20_8.csv, pcns_20_9.csv, pcns_20_10.csv, pcns_20_11.csv,
pcns_20_12.csv, pcns_20_13.csv, pcns_20_14.csv, pcns_20_15.csv, pcns_20_16.csv,
pcns_20_17.csv

pcns_21_1.csv, pcns_21_2.csv, pcns_21_3.csv, pcns_21_4.csv, pcns_21_5.csv, pcns_21_6.csv,
pcns_21_7.csv, pcns_21_8.csv, pcns_21_9.csv, pcns_21_10.csv, pcns_21_11.csv,
pcns_21_12.csv, pcns_21_13.csv, pcns_21_14.csv, pcns_21_15.csv, pcns_21_16.csv,
pcns_21_17.csv

pcns_22_1.csv, pcns_22_2.csv, pcns_22_3.csv, pcns_22_4.csv, pcns_22_5.csv, pcns_22_6.csv,
pcns_22_7.csv, pcns_22_8.csv, pcns_22_9.csv, pcns_22_10.csv, pcns_22_11.csv,
pcns_22_12.csv, pcns_22_13.csv, pcns_22_14.csv, pcns_22_15.csv, pcns_22_16.csv,
pcns_22_17.csv
```

`pcns_24_1.csv, pcns_24_2.csv, pcns_24_3.csv, pcns_24_4.csv, pcns_24_5.csv, pcns_24_6.csv, pcns_24_7.csv, pcns_24_8.csv, pcns_24_9.csv, pcns_24_10.csv, pcns_24_11.csv, pcns_24_12.csv, pcns_24_13.csv, pcns_24_14.csv, pcns_24_15.csv, pcns_24_16.csv, pcns_24_17.csv, pcns_24_18.csv`

`pcns_26_1.csv, pcns_26_2.csv, pcns_26_3.csv, pcns_26_4.csv, pcns_26_5.csv, pcns_26_6.csv, pcns_26_7.csv, pcns_26_8.csv, pcns_26_9.csv, pcns_26_10.csv, pcns_26_11.csv, pcns_26_12.csv, pcns_26_13.csv, pcns_26_14.csv, pcns_26_15.csv, pcns_26_16.csv, pcns_26_17.csv, pcns_26_18.csv`

`pcns_28_1.csv, pcns_28_2.csv, pcns_28_3.csv, pcns_28_4.csv, pcns_28_5.csv, pcns_28_6.csv, pcns_28_7.csv, pcns_28_8.csv, pcns_28_9.csv, pcns_28_10.csv, pcns_28_11.csv, pcns_28_12.csv, pcns_28_13.csv, pcns_28_14.csv, pcns_28_15.csv, pcns_28_16.csv, pcns_28_17.csv, pcns_28_18.csv, pcns_28_19.csv`

`pcns_30_1.csv, pcns_30_2.csv, pcns_30_3.csv, pcns_30_4.csv, pcns_30_5.csv, pcns_30_6.csv, pcns_30_7.csv, pcns_30_8.csv, pcns_30_9.csv, pcns_30_10.csv, pcns_30_11.csv, pcns_30_12.csv, pcns_30_13.csv, pcns_30_14.csv, pcns_30_15.csv, pcns_30_16.csv, pcns_30_17.csv, pcns_30_18.csv, pcns_30_19.csv`

Ferner ist die Datei

`pcns_range.csv`

enthalten, in der die jeweils kleinsten (bzgl. Definition 6.39) $\mathcal{PCN}$-Polynome von Grad n über $\mathbb{F}_{p^r}$ für $r = 1$ und alle Primzahlen $p < 1000$ mit $p^n < 10^{70}$ aufgeführt werden. Die Syntax der Einträge ist dabei analog zu oben, wobei n ergänzt wurde, d.h. der Eintrag `p, n, poly` bedeutet, dass `poly` das kleinste $\mathcal{PCN}$-Polynom von Grad n über $\mathbb{F}_p$ ist.

B.1.3 | ./Sage

Dieser Ordner beinhaltet die Quellcodes der in Kapitel 6 vorgestellten Funktionen:

```
enumeratePCNs.c
enumeratePCNs.spyx
findAnyPCN_additional.spyx
findAnyPCN_trinom.spyx
```

B.2 | Online

Das gesamte Arbeitsverzeichnis dieser Arbeit ist unter

`https://github.com/hackenbergstefan/masterarbeit/`

zugänglich. Die Datenstruktur ist dabei wie folgt gegeben:

`./CD`	Vollständiger Inhalt der angehängten CD wie in Anhang B.1 beschrieben
`./Latex`	Ordner mit LaTeX-Quellcodes
`./Sage`	Ordner mit `Sage`-Quellcodes und Ausgabedateien
`./Tables`	Ergebnisse der `Sage`-Funktionen, identisch mit dem Ordner `Tables` auf angehängter CD (vgl. Anhang B.1)
`./Masterarbeit.pdf`	Kompilierte Version der Arbeit