Faktorisierung von Polynomen über $\mathbb{Z}$

-

Herleitung und Realisierung zweier konkurrierender Algorithmen

Diplomarbeit im Fachbereich Mathematik
der Universität Bremen
vorgelegt von

Lars F.Paape

Gutachter:
Prof. Dr. Jens Gamst
Prof. Dr. Michael Hortmann

Wintersemester 2003/2004

Gedankt sei all denen, die mir diese Arbeit ermöglicht haben.

Inhaltsverzeichnis

1 Einleitung

Sei zunächst *L. Kronecker (1823 -1891)* mit einer Äußerung aus dem ausgehenden 19. Jahrhundert zum Thema ganzzahlige Polynome zitiert: ,, Solange man keine Methode hat, mit der man entscheiden kann, ob eine gegebene Funktion irreduzibel ist, fehlt es der Definition der Irreduzibilität an ihrer Grundlage."
Die Faktorisierung von ganzzahligen Polynomen stellt eines der Hauptprobleme bei der Betrachtung von Polynomen dar. Die Tatsache, dass überhaupt ein Algorithmus zur Faktorisierung existiert, leitet man dabei schnell aus der Tatsache ab, dass die Größe der Koeffizienten eines Teilers eines Polynoms beschränkt ist (s. Anhang A.3, S. 94). Daher wird die Zahl der möglichen Teiler endlich und durch einfaches Testen aller dieser Polynome ist bereits ein Algorithmus gefunden. Untersuchungen und Resultate in diesem Gebiet der Mathematik zielen daher meist auf die Entwicklung möglichst effizienter Algorithmen.
Zwei dieser Algorithmen sind Gegenstand dieser Diplomarbeit. Dabei handelt es sich bei dem ersten um ein von *D. G. Cantor* und *H. Zassenhaus* entwickeltes probabilistisches Verfahren, welches im Mittel eine polynomiale Anzahl von Operationen benötigt. Im Extremfall erhält man allerdings eine Laufzeit von exponentieller Ordnung. 1982 wurde von A. K. Lenstra, H. W. Lenstra und L. Lovász ein Verfahren, der sog. *LLL-Algorithmus* vorgestellt, welches von polynomialer Ordnung ist. Dieses bildet das zweite zu bearbeitende Verfahren dieser Arbeit.
Die vorliegende Diplomarbeit beschäftigt sich also mit einem Bereich der Mathematik, der in die so genannte *Algorithmische Zahlentheorie* fällt. Die Bezeichnung dieses Gebietes enthält dabei schon ein Schlagwort des Computerzeitalters – *den Algorithmus*. Auch wenn der Begriff schon auf eine viel längere Geschichte zurückblickt, so zeigt sich hier die Nähe zur Informatik. Diese ist es, die die Anwendung vieler Algorithmen überhaupt erst ermöglicht. In der Tat liegt das Hauptaugenmerk dieser Arbeit auch auf für die Informatik – hier verstanden als anwendende Wissenschaft – interessanten Gebieten, nämlich:

- der abgeschlossenen Darstellung der Algorithmen,
- der kompromisslosen Realisierung der untersuchten Algorithmen in einer Programmiersprache, welche die leichte Adaption an verschiedene Systeme/Programmiersprachen ermöglicht und
- einer ergänzenden, informellen Laufzeitbetrachtung.

Allerdings soll bei der Darstellung der Algorithmen gerade die mathematische Beweisführung in keiner Weise vernachlässigt werden, denn es gilt zu bedenken, daß nur so ein vollständiges Verständnis für einen Algorithmus erlangt werden kann und gerade dieses ja essentiell bei der Realisierung ist.

Die Arbeit orientiert sich in ihrer Gliederung im Wesentlichen an zwei englischsprachigen Standardwerken ([COH],[KNU]). Es wird ebenso versucht eine ausführliche, in sich geschlossene, theoretische Herleitung des Sachverhaltes zu verwirklichen. Die Betrachtungen führen von einem allgemeinen theoretischen Teil, über allgemeine Beobachtungen zur Polynomfaktorisierung bis hin zu den jeweiligen speziellen Algorithmen. In den ersten drei Kapiteln des Anhangs finden sich zusätzliche, ergänzende Informationen.
Zur Visualisierung der Algorithmen wurden Diagramme mit erklärendem Pseudo-Code gewählt, entgegen der sonst üblichen reinen Pseudo-Code-Darstellung ermöglicht dies ein sofortiges Verstehen der Abläufe und Verschachtelungen.
Da ein wesentlicher Schwerpunkt dieser Arbeit auf der vollständigen Implementierung der beschriebenen Faktorisierungsalgorithmen liegt, findet sich der komplette Quellcode in Form von Skripten für das Multipräzisions-Software-Paket *PARI/GP* im Anhang D. Dies ermöglicht dem Leser unmittelbar nach Bearbeitung der jeweiligen Theorie, das Erfahrene mit lauffähigem Programmcode zu vergleichen. Insbesondere, da auch aufwendige Algorithmen auf ein bis zwei Seiten Platz finden, scheint dieses Vorgehen sinnvoll.

Bemerkung zur Laufzeitbetrachtung: Die in der Herleitung des jeweiligen Algorithmus informell angefügte Laufzeitbetrachtung geht von der üblichen Konvention aus, dass die Elementaroperationen durch die Addition/Subtraktion sowie die Multiplikation/Division geben sind, und dass die *binäre Länge* einer Zahl $x \in \mathbb{Z}$ gegeben ist durch die Zahl der Stellen des Betrages.

2 Polynom-Arithmetik

In diesem vorbereitenden Kapitel werden Grundlagen über den Umgang mit Polynomen behandelt, sowie die allgemein im englischen Sprachgebrauch übliche Nomenklatur eingeführt. Diese Nomenklatur erscheint insbesondere deshalb sinnvoll, da die zugrundeliegende Literatur fast ausschließlich aus dem englischsprachigen Raum kommt. Für weitere Grundlagen zum Thema Polynome sei außerdem auf den Anhang A.1, S. 86 verwiesen.

2.1 Polynomdarstellung

Formal versteht man unter einem *Polynom* über einem noch nicht näher spezifizierten algebraischen System S einen Ausdruck der folgenden Art

$$f(X) = a_n X^n + \ldots + a_0$$

mit $a_i \in S$. Die *Unbestimmte* X ist dabei ein formales Symbol mit unbestimmter Bedeutung. Im Weiteren gehen wir davon aus, dass S mindestens ein kommutativer Ring mit Einselement ist. In offensichtlicher Weise können Polynome durch Vektoren (in der Sprechweise der Informatik: Felder/Arrays) repräsentiert werden. Bedingt durch die Implementierung gibt es hier zwei Darstellungsmöglichkeiten, nämlich durch ein Feld beginnend oder endend mit dem Koeffizienten a_0. Die nachfolgenden Algorithmen werden die Repräsentation der Form $a_0, a_1, \ldots, a_n$ verwenden, da dieser in der verwendeten Literatur meist der Vorzug gegeben wird. *PARI/GP* verwendet die umgekehrte Notation; dies ist in den jeweiligen Realisierungen zu berücksichtigen, wobei auch hierfür entsprechende Funktionen zur Verfügung stehen.
Man sagt, dass ein Polynom f den Grad n, geschrieben $deg(f) = n$, hat, wenn $a_n \neq 0$ und $a_{n+i} = 0$ für $i \in \mathbb{N}$ gilt. Für den Höchstkoeffizienten eines Polynoms g wird im Folgenden die Schreibweise $\mathrm{lc}(g)$ verwendet. Im Vorangehenden gilt also $\mathrm{lc}(f) = a_n$. Soweit nicht ausdrücklich anders beschrieben gelten für das *Nullpolynom* die üblichen Konventionen:

$$\deg(0) = -\infty \quad \text{und} \quad \mathrm{lc}(0) = 0.$$

Gilt $\mathrm{lc}(f) = 1$ für ein Polynom f, so spricht man von einem *normierten* Polynom.
Die Addition, Subtraktion und Multiplikation mit einem Skalar sind in kanonischer Weise durch die entsprechenden Operationen auf Vektoren gegeben. Die *Multiplikation von Polynomen* f, g wird natürlich mittels

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

mit

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

erledigt. Für eine programmtechnische Implementierung ist dies allerdings nicht das optimale Vorgehen. Eine detailliertere Betrachtung findet sich bei [COH], Kapitel 3.1.2. Die Polynomdivision benötigt eine gesonderte Betrachtung.

2.1.1 Polynomdivision

Sei K Körper und man betrachte den Polynomring $K[X]$. Gemäß Anhang A.1 ist $K[X]$ ein *euklidischer Ring*, d. h. für $u, v \in K[X], v \neq 0$ existieren Polynome $q, r \in K[X]$ mit

$$u = qv + r \quad \text{und} \quad \deg(r) < \deg(v).$$

Aus dem Beweis zu Satz A.2 (S. 87) folgt auch unmittelbar ein Verfahren zur Bestimmung von q und r.

Algorithmus 2.1 (Euklidische Division) *Seien $u, v \in K[X], v \neq 0$. Unter der Voraussetzung, dass $m = \deg(u) \geq \deg(v) = n \geq 0$ gilt, bestimmt der Algorithmus $q, r \in K[X]$ mit $u = qv + r$ und $\deg(r) < \deg(v)$.*

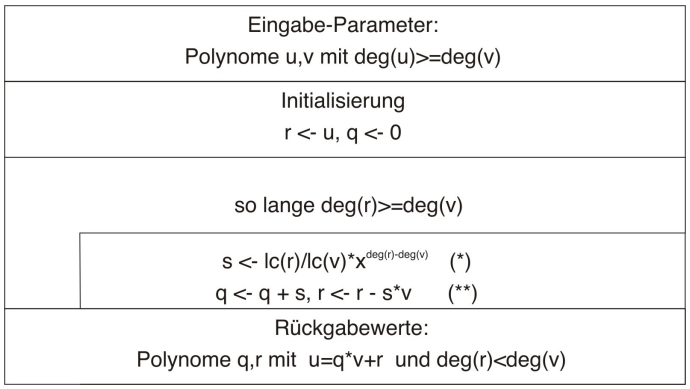

Diagramm 1: Euklidische Division

Bemerkung: Die Implementierung der euklidischen Division erfolgt in der vorliegenden Arbeit unter zur Hilfenahme des nachfolgend betrachteten Pseudo-Divisions-Algorithmus (Source-Code: Anhang B.2.1, S. 100). Betrachtet man die o. g. Division, erkennt man, dass lediglich unter (*) die Division im Körper K benötigt wird und diese genau $m - n + 1$ Mal durchgeführt wird. Unter (**) wird in Wirklichkeit keine Polynommultiplikation ausgeführt, sondern lediglich eine Skalarmultiplikation verbunden mit einem

Koeffizientenshift. Insgesamt kann man daher für Polynome ohne Einsatz von Divisionen folgende Darstellung erreichen:

$$\text{lc}(v)^{m-n+1}u = qv + r \quad \text{mit} \quad \deg(r) < n. \tag{2.1}$$

Im Falle von ganzzahligen Polynomen erkennt man an der Darstellung 2.1 auch, dass q und r nur ganzzahlige Koeffizienten haben.Der obige Algorithmus kann zum sog. Pseudo-Divisions-Algorithmus für Polynome über kommutativen Ringen mit Einselement erweitert werden.

Algorithmus 2.2 (Pseudo-Division) *Seien* $u, v \in R[X], v \neq 0$ *und* R *kommutativer Ring mit Einselement. Unter der Voraussetzung, dass* $m = \deg(u) \geq \deg(v) = n \geq 0$ *gilt, bestimmt der Algorithmus* $q, r \in R[X]$ *mit* $lc(v)^{m-n+1}u = qv + r$ *und* $\deg(r) < \deg(v)$.

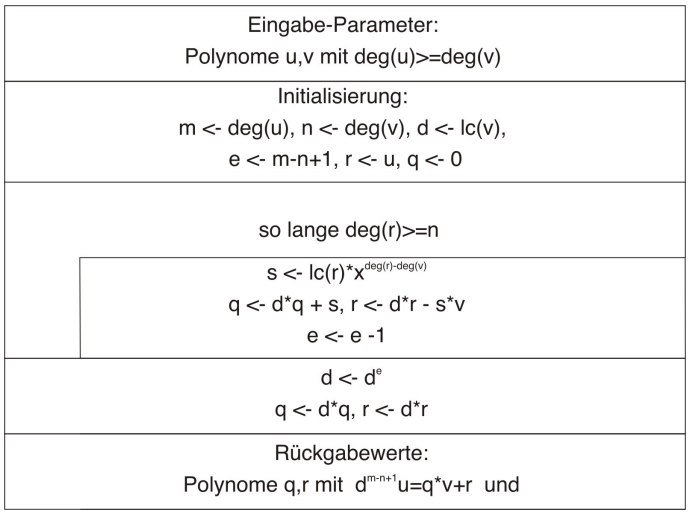

Diagramm 2: Pseudo-Division
Sourcen-Code siehe polynomutilities.par*, Anhang S.117*

Bemerkung: Bei Knuth [KNU] findet sich eine Implementierung des obigen Algorithmus auf Basis der Koeffizienten der beteiligten Polynome. Dieser Variante sollte im Falle der Implementierung in einer Hochsprache der Vorzug gegeben werden; in *PARI/GP* hingegen würden so zusätzliche Konvertierungen benötigt, und eine unnötige Laufzeitverlängerung wäre die Folge.

2.2 Polynome über faktoriellen Ringen

Würde man sich auf Polynome über Körpern beschränken, so würde man die interessanten Eigenschaften im Falle der ganzzahligen Polynome damit bereits ausklammern. Daher betrachtet man Polynome über *faktoriellen Ringen*. Informell sei nachfolgend noch einmal die Definition gegeben; die grundlegenden Eigenschaften können im Anhang A.1.2 (S. 88) gefunden werden.

Definition 2.1 (faktorieller Ring) *Ein Integritätsring R heißt* faktorieller Ring*(UFD, Unique Factorization Domain), wenn R zusätzlich die folgende Eigenschaft hat:*
Zu jedem $a \in R$ mit $a \neq 0$ und $a \notin R^*$ gibt es Primelemente $p_1, \ldots, p_r \in R$ mit $a = p_1 \cdots p_r$. Diese Darstellung ist bis auf Permutation eindeutig.

Eine Menge von Elementen eines faktoriellen Rings heißt *relativ prim*, falls kein Primelement aus dem faktoriellen Ring existiert, welches alle Elemente teilt. Ein Polynom f eines faktoriellen Rings R heißt *primitiv*, falls die Koeffizienten relativ prim sind; mit anderen Worten, der ggT der Koeffizienten ist eine Einheit.

Lemma 2.1 (Lemma von Gauß) *Das Produkt zweier primitiver Polynome über einem faktoriellen Ring R ist wieder primitiv.*

Beweis: Sei

$$f = \sum_{i=0}^{n} a_i X^i \quad \text{und} \quad g = \sum_{j=0}^{m} b_j X^j.$$

Für das Produkt $h = fg$ gilt: $h = \sum_{k=0}^{n+m} c_k X^k$, wobei die c_k durch

$$c_k = \sum_{i+j=k} a_i b_j$$

gegeben sind. Zu zeigen ist, dass die c_k keinen gemeinsamen Teiler > 1 haben. Angenommen dies wäre nicht der Fall, dann gäbe es insbesondere einen gemeinsamen Primteiler p. Da f, g primitiv sind, muss es a_r bzw. b_s mit jeweils dem größten Index geben, so dass $p \nmid a_r$ bzw. $p \nmid b_s$. Betrachtet man nun den Koeffizienten c_{r+s}, so hat dieser die Darstellung

$$c_{r+s} = a_r b_s + \sum_{i+j=r+s, i \neq r} a_i b_j$$

Wegen des Summanden $a_r b_s$ ist c_{r+s} nicht durch p teilbar, was im Widerspruch zur Annahme steht.

□

Lemma 2.2 (Zerlegung mit primitivem Anteil) *In einem faktoriellen Ring R kann jedes von Null verschiedene Polynom in die Form $u = cv$ zerlegt werden, wobei $c \in R$ und v primitives Polynom ist. Diese Darstellung ist abgesehen von Einheiten eindeutig. D. h. , existieren zwei Darstellungen $u = c_1 v_1$ und $u = c_2 v_2$, dann existiert ein $a \in R^*$ so, dass $c_1 = ac_2$ und $v_1 = av_2$ gilt.*

Beweis: Existenz: Sei $u \in R[X] \backslash \{0\}$. Falls u nicht primitiv ist, existiert ein Primelement $p_1 \in R$, welches alle Koeffizienten von u teilt und somit ein weiteres Polynom u_1 mit $u = p_1 u_1$. Falls u_1 primitiv ist, ist man fertig. Falls u_1 nicht primitiv ist, erhält man mit erneuter Anwendung des obigen Vorgehens ein Primelement p_2 und ein Polynom u_2. Da R faktoriell und in jedem Schritt die Zahl der Primfaktoren der Koeffizienten um eins abnimmt, muss das Verfahren nach endlich vielen Schritten abbrechen.
Eindeutigkeit: Angenommen es existieren zwei Darstellungen mit $c_1 v_1 = c_2 v_2$ und v_1, v_2 primitiv. Sei nun p ein beliebiges Primelement von R. Falls $p^k \mid c_1$ gilt, folgt $p^k \mid c_2$, sonst würde $p^k \mid v_2$ gelten, was der Tatsache widerspricht, dass v_2 primitiv ist. Ebenso folgt die umgekehrte Richtung, und damit sind c_1, c_2 assoziiert. Aus $0 = ac_2 v_1 - c_2 v_2 = c_2(av_1 - v_2)$ folgt $v_2 = av_1$.

□

Daher kann man jedes Polynom u mit $u \neq 0$ als

$$u = \mathrm{cont}(u)\mathrm{pp}(u)$$

schreiben, wobei $\mathrm{pp}(u)$ den primitiven Anteil von u bestimmt und $\mathrm{cont}(u)$ dem c aus dem vorherigen Lemma entspricht. Für das Nullpolynom verwendet man die Konvention

$$\mathrm{cont}(0) = \mathrm{pp}(0) = 0.$$

Die *cont*-Funktion ist dabei abgesehen von Einheiten in offensichtlicher Weise durch einen ggT der Koeffizienten des Polynoms bestimmt. Im Falle von Polynomen über $\mathbb{Z}$ gilt zusätzlich die Konvention, dass der Höchstkoeffizient von $\mathrm{pp}(u)$ positiv sein soll. Die Realisierung sowie weitere Details zu den jeweiligen Routinen finden sich im Anhang B.2.3 (S. 101).
Durch Kombination von Lemma 2.1 und 2.2 erhält man die Beziehungen:

Lemma 2.3 *Seien $u, v \in R[X]$, R faktorieller Ring. Dann gilt*

$$\begin{aligned} \mathrm{cont}(uv) &= a\,\mathrm{cont}(u)\mathrm{cont}(v), \\ \mathrm{pp}(uv) &= b\,\mathrm{pp}(u)\mathrm{pp}(v), \end{aligned}$$

wobei a und b Einheiten sind mit $ab = 1$. Falls Polynome über $\mathbb{Z}$ betrachtet werden, gilt wg. obiger Konvention $a = b = 1$.

Lemma 2.4 *Sei R faktorieller Ring. Jedes irreduzible Polynom $f \in R[X]$ mit $\deg(f) > 0$ ist primitiv.*

Beweis: Sei $a \in R$ ein gemeinsamer Teiler der Koeffizienten von f, so gilt $f = ag$ für ein $g \in R[X]$. Wegen $\deg(g) = \deg(f) > 0$ liegt g nicht in $R^* = (R[X])^*$. Da f irreduzibel in $R[X]$ ist, muss daher $a \in R^*$ gelten. Damit ist f primitiv.

□

Lemma 2.5 *Sei R faktorieller Ring und K sein Quotientenkörper. Dann gilt:*

i.) Für alle $g \in K[X]\backslash\{0\}$ existiert ein $a \in K$, so dass $g_1 := ag \in R[X]$ und primitiv ist.

ii.) Falls $f, g \in R[X]$ und g primitiv ist, so folgt aus $f = ag$ mit einem $a \in K$, dass $a \in R$ ist.

Beweis: Zu i.): Sei $g = \sum_{i=0}^{n} a_i X^i$ mit $a_i \in K$. Da K Quotientenkörper von R ist, existieren $r_i, s_i \in R$ für $i = \{1, \ldots, n\}$ so, dass $a_i = \frac{r_i}{s_i}$. Mit $s := \prod_{i=1}^{n} s_i$ gilt dann $sg \in R[x]\backslash\{0\}$. Setze $d := \mathrm{ggT}(\{\text{Koeffizienten von } sg\})$. Dann existiert ein primitives Polynom $g_1 \in R[X]$ mit $sg = dg_1$ und man erhält $g_1 = \frac{s}{d} g$.
Zu ii.):Da K Quotientenkörper von R und R faktoriell ist, gibt es teilerfremde $r, s \in R$ mit $a = \frac{r}{s}$. Angenommen $a \notin R$, so ist s keine Einheit; s hat also insbesondere einen Primteiler p. Da g primitiv ist, teilt dieser nicht alle Koeffizienten von g. Also müßte p, da p Primteiler ist, wg. $sf = rg$, r oder g teilen, Widerspruch.

□

Satz 2.1 *Sei R faktorieller Ring und K sein Quotientenkörper. Für jedes $f \in R[X]$ mit $\deg(f) > 0$ gilt:*

i.) Ist f primitiv und $g \in R[X]\backslash\{0\}$, dann folgt aus $f \mid g$ in $K[X]$ auch $f \mid g$ in $R[X]$.

ii.) Ist f irreduzibel in $R[X]$, so auch in $K[X]$.

Beweis: Zu i.): Falls $f \mid g$ in $K[X]$ gilt, so existiert ein $h \in K[X]$ mit $g = fh$. Nach dem vorhergehenden Lemma können wir ein $a \in K$ wählen so, dass $h_1 := ah$ ein primitives Polynom über R ist. Man erhält also $g = a^{-1} f h_1$ wobei fh_1 nach dem Lemma von Gauß primitiv ist. Erneute Anwendung von Lemma 2.5 führt zu $a^{-1} \in R$ und damit $f \mid g$ in $R[X]$.

Zu ii.): Sei $g \in K[X]$ mit $g \mid f$. Sei $a \in K$ wie im Beweis zu i.) gewählt so, dass $g_1 := ag$ primitives Polynom in $R[X]$ ist. Dann gilt aber $g_1 \mid f$ in $K[X]$ und mit i.) folgt auch $g_1 \mid f$ in $R[X]$. Somit muss $g_1 \in R^*$ oder g_1 ist assoziiert zu f gelten, da f irreduzibel ist. Wegen $g_1 = ag$ gilt damit $g \in K^*$, oder es existiert ein $b \in K^*$ mit $f = bg$. Also hat f nur triviale Teiler in $K[X]$ und ist wg. $\deg(f) > 0$ damit irreduzibel in $K[X]$.

□

Eine der Haupteigenschaften, die faktorielle Ringe auszeichnet, ist durch den *Satz von Gauß* gegeben.

Satz 2.2 (Satz von Gauß) *Sei R faktorieller Ring. Dann ist auch $R[X]$ faktorieller Ring.*

Beweis nach [FUS] unter Verwendung von Satz A.8, S. 89:
Existenz der faktoriellen Zerlegung: Sei $f \in R[X]$ mit $f \neq 0$ und $f \notin R^* = (R[X])^*$ beliebig. Mittels Induktion über $n = \deg(f)$ wird gezeigt, dass solch ein f endliches Produkt irreduzibler Elemente ist.
Induktionsanfang ($n = \deg(f) = 0$)*:* Für jedes f mit $\deg(f) = 0$ gilt: f ist von Null verschiedene Nichteinheit und hat, da R faktoriell ist, eine Darstellung als endliches Produkt irreduzibler Elemente.
Induktionschluss ($(\deg(f) < n) \rightarrow n$)*:* Sei die Behauptung für alle Polynome $h \in R[X] \backslash (R^* \cup \{0\})$ und $\deg(h) < n$ gezeigt. Ferner sei f mit $\deg(f) = n$ sowie obigen Voraussetzungen gegeben. Ist a ein ggT der Koeffizienten von f, so gibt es ein primitives Polynom f^* mit $f = af^*$. Da R faktoriell ist, ist a entweder eine Einheit oder ein endliches Produkt irreduzibler Elemente. Ist f^* irreduzibel, ist damit die Behauptung gezeigt. Falls f^* nicht irreduzibel ist, existieren Polynome $g, h \notin R^*$ und $f^* = gh$. Da f^* primitiv ist, gilt $\deg(g) < n$ und $\deg(h) < n$. Nach der Induktionsvoraussetzung ist dann f^* endliches Produkt irreduzibler Elemente.
Eindeutigkeit der faktoriellen Zerlegung: Seien $c_1, \ldots, c_m, p_1, \ldots, p_n$ und $d_1, \ldots, d_k, q_1, \ldots, q_l$ irreduzible Elemente von $R[X]$ wobei

$$c_1 \cdots c_m p_1 \cdots p_n = d_1 \cdots d_k q_1 \cdots q_l.$$

Zudem seien die Bezeichnungen so gewählt, dass $deg(c_i) = 0$ bzw. $deg(d_j) = 0$ für $i \in \{1, \ldots, m\}$ bzw. $j \in \{1, \ldots, k\}$, sowie $deg(p_r) > 0$ bzw. $deg(q_s) > 0$ für $r \in \{1, \ldots, n\}$ bzw. $s \in \{1, \ldots, l\}$ gelte. Als irreduzible Polynome mit Grad> 0 sind $p_1, \ldots, p_n, q_1, \ldots, q_l$ primitiv und damit auch die beiden Produkte

$$p_1 \cdots p_n \quad \text{und} \quad q_1 \cdots q_l.$$

Dies hat nach dem Lemma 2.2 zur Folge, dass

$$c_1 \cdots c_m \quad \text{und} \quad d_1 \cdots d_k$$

assoziiert sein müssen. Da R faktoriell ist, ist damit $m = k$ und ggf. nach Permutation sind c_i und d_i für $i \in \{1, \ldots, m\}$ assoziiert. Damit sind auch

$$p_1 \cdots p_n \quad \text{und} \quad q_1 \cdots q_l$$

assoziiert in $K[X]$. Mit dem 2. Teil von Satz 2.1 sind $p_1, \ldots, p_n, q_1, \ldots, q_l$ irreduzibel in $K[X]$. $K[X]$ ist, da K Körper ist, faktorieller Ring, und damit gilt $n = l$. Ggf. nach geeigneter Nummerierung kann erreicht werden, dass p_i, q_i assoziiert in $K[X]$ sind. Nach Satz A.1 (S. 86) bedeutet dies $p_i \mid q_i$ und $q_i \mid p_i$ in $K[X]$, so dass nach dem 1. Teil von Satz 2.1 gilt, dass p_i, q_i in $R[X]$ assoziiert sind.

□

2.3 Euklidischer Algorithmus

2.3.1 Fall 1: Polynome über Körpern

Falls wir Polynome über einem Körper K betrachten, so handelt es sich bei $K[X]$ gemäß Satz A.2 (S. 87) um einen euklidischen Ring, und Satz A.3 (S. 87) liefert im Beweis den wohl bekannten *euklidischen Algorithmus*.

Algorithmus 2.3 (Euklidischer Algorithmus) *Seien $a, b \in K[X]$ und K Körper. Der Algorithmus bestimmt einen ggT von a und b.*

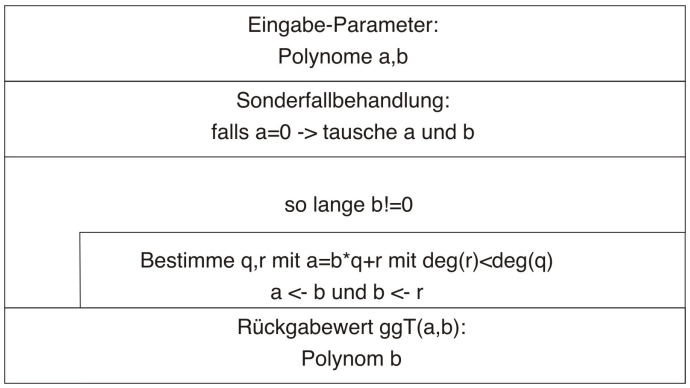

Diagramm 3: euklidischer Algorithmus

Mittels Satz A.6 (S. 89) folgert man dann den *erweiterten euklidischen Algorithmus*.

Algorithmus 2.4 (erweiterter Euklidischer Algorithmus) *Seien* $a, b \in K[X]$ *und* K *Körper. Der Algorithmus bestimmt Polynome* $u, v \in K[X]$ *so, dass* $au + bv = d$ *mit* $d = \mathrm{ggT}(a, b)$ *gilt.*

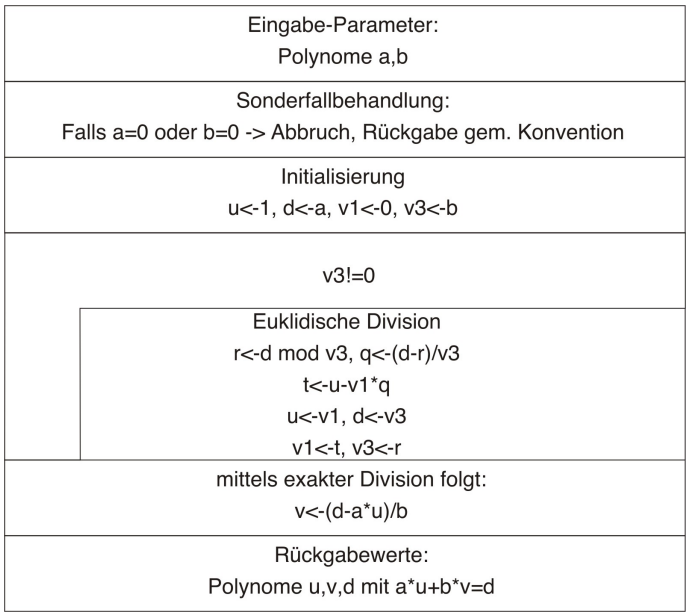

Diagramm 4: erweiterter euklidischer Algorithmus
Sourcen-Code siehe polynomutilities.par*, Anhang S.117*

2.3.2 Fall 2: Polynome über faktoriellen Ringen

Mit den bisher gezeigten Sachverhalten ist man nun in der Lage, den euklidischen Algorithmus in einer angepaßten Version für Polynome über faktoriellen Ringen anzugeben. Dabei sind die auszuführenden Operationen im Wesentlichen die gleichen wie bei der Betrachtung des gewöhnlichen euklidischen Algorithmus, falls das Polynom f als Element des Quotientenkörpers angesehen wird.

Der u. g. Algorithmus verwendet aufgrund einer Modifikation allerdings nur Operationen in R und reduziert daher die Gesamtzahl der Operationen erheblich, da hier unnötige ggT-Bestimmungen im Quotientenkörper entfallen. Die Arbeit mit primitiven Polynomen verhindert im Algorithmus im Falle von Polynomen über $\mathbb{Z}$ die Koeffizienten-Explosion (s. Beispiel 2.1, S.18).

Algorithmus 2.5 (verallgemeinerter euklidischer Algorithmus)
Seien $a, b \in R[X]$ und R faktorieller Ring. Der Algorithmus bestimmt einen ggT von a und b, wobei lediglich Operationen in R ausgeführt werden.

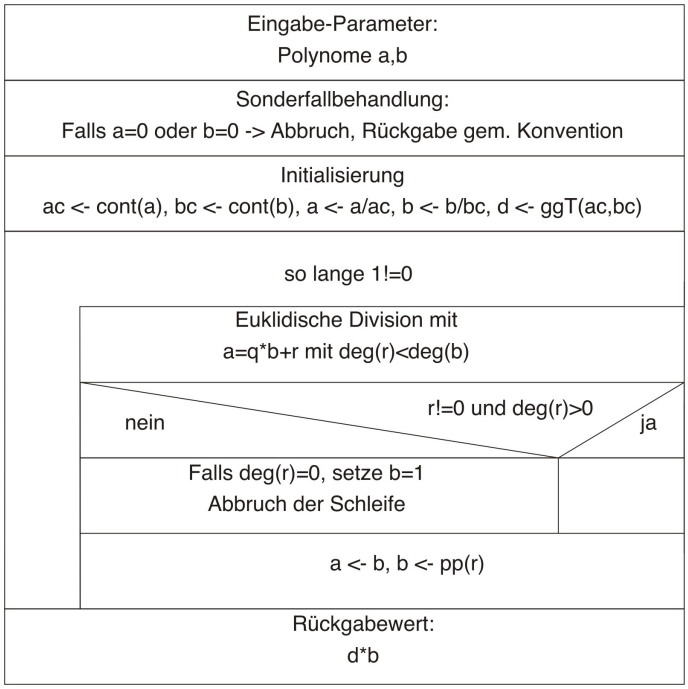

Diagramm 5: verallgemeinerter euklidischer Algorithmus

Beweis: Die Verallgemeinerung des euklidischen Algorithmus für primitive Polynome erhält man durch folgende Betrachtung:

Seien $a, b \in R[X]$ primitive Polynome mit $\deg(a) \geq \deg(b) >= 0$. Mittels der nach Algorithmus 2.2 (S. 10) gültigen Pseudo-Division erhält man aus der Formel $l^{\deg(a)-\deg(b)+1}a = bq + r$ mit $q, r \in R[X]$ die Gültigkeit von $\mathrm{ggT}(a, b) = \mathrm{ggT}(b, r)$, denn offensichtlich teilt jeder gemeinsame Teiler von a und b auch b und r. Andererseits teilt jeder gemeinsame Teiler von b und r $l^{\deg(a)-\deg(b)+1}a$. Da b primitiv ist, muss der gemeinsame Teiler primitiv sein, also a teilen. Falls $r = 0$ ist, gilt $\mathrm{ggT}(a, b) = b$; falls $r \nmid 0$ ist, führt wegen des bisher Gesagten mittels $\mathrm{ggT}(a, b) = \mathrm{ggT}(b, pp(r))$ (beachte: b ist primitiv) die Iteration des vorangehenden Prozesses zum Erfolg.

□

Die sog. Koeffizienten-Explosion sei hier noch einmal an Hand eines klassischen Beispiels von Knuth ([KNU]) gezeigt.

Beispiel 2.1 (Koeffizientenexplosion) *Man betrachte die beiden ganzzahligen Polynome a, b, welche durch*

$$\begin{aligned} a &= X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5, \\ b &= 3X^6 + 5X^4 - 4X^2 - 9X + 21 \end{aligned}$$

gegeben sind.
Nur die Anwendung der Pseudo-Division ohne Reduzierung auf primitive Polynome würde in einer entsprechend abgeänderten Version des vorangehenden Algorithmus in jedem Pseudo-Divisions-Schritt die folgende Sequenz von Resten erzeugen:

$$\begin{aligned} r_1 &= -15X^4 + 3X^2 - 9, \\ r_2 &= 15795X^2 + 30375X - 59535, \\ r_3 &= 1254542875143750X - 1654608338437500, \\ r_4 &= 12593338795500743100931141992187500 \end{aligned}$$

Dies zeigt, dass a, b relativ prim sind.

Mittels Reduzierung auf primitive Polynome liefert der o. g. Algorithmus die folgende Sequenz:

$$\begin{aligned} r_1 &= 5X^4 - X^2 + 3, \\ r_2 &= 13X^2 + 25X - 49, \\ r_3 &= 4663X - 6150, \\ r_4 &= 1, \end{aligned}$$

welche bezüglich der Koeffizientengröße des jeweiligen Restes der Pseudo-Division sicherlich optimal ist. Zu beachten ist allerdings, dass dies mit je einem Aufruf der pp*-Funktion, innerhalb derer die* cont*-Funktion zur Bestimmung des ggTs der Koeffizienten aufgerufen wird, in der Hauptschleife erkauft wird.*

2.4 Subresultanten-Algorithmus

Man sieht leicht am vorhergehenden Algorithmus, dass die rechenintensivste Operation zur Bestimmung eines ggTs zweier Polynome innerhalb der Hauptschleife liegt; bei jedem Schritt wird hier der ggT der Koeffizienten des Polynomes r bestimmt (s. pp-Funktion). Auf der anderen Seite ist zu beachten, dass ein Weglassen der obigen Funktion, welches mathematisch ohne Probleme möglich wäre, zu einer erheblichen Koeffizientenexplosion (s. voriges Beispiel) führen und somit ebenfalls den Algorithmus verlangsamen würde.

Der von *G. E. Collins* 1967 gefundene und von *W. S. Brown* sowie *J. F. Traub* verfeinerte Algorithmus geht einen Mittelweg, vermeidet die o. g. zusätzlichen Berechnungen in der Hauptschleife und führt hier eine Division des euklidischen Restes r durch einen speziellen Faktor durch. Der Algorithmus ist dabei wie folgt gegeben.

Algorithmus 2.6 (Subresultanten-Algorithmus) *Seien $u, v \in R[X]$ und R faktorieller Ring. Der Algorithmus bestimmt einen ggT von u und v, wobei lediglich Operationen in R ausgeführt werden.*

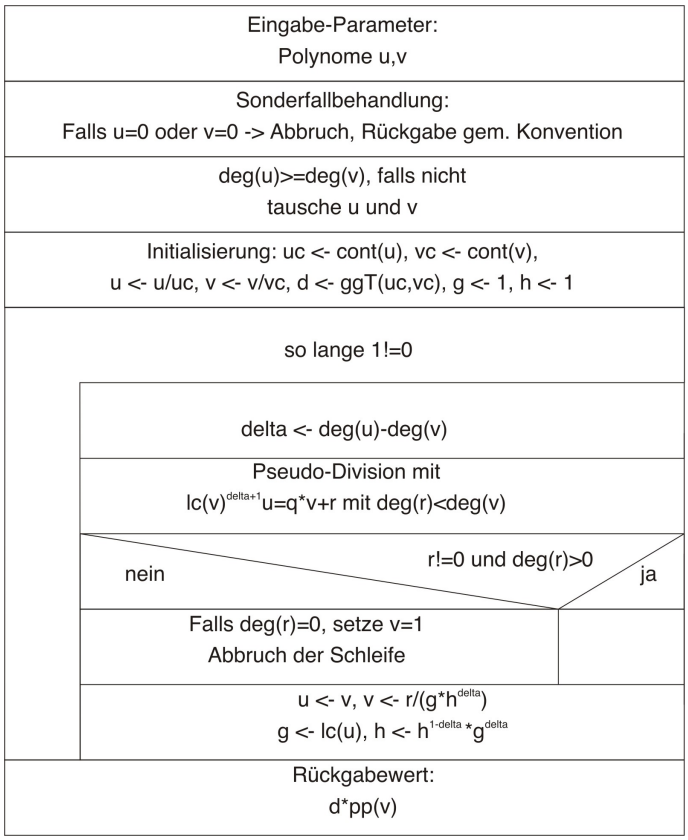

Diagramm 6: Subresultanten-Algorithmus
Sourcen-Code siehe subresultant.par*, Anhang S.124*

Bemerkung zur Laufzeit: Unter der Bedingung, dass für zwei Polynome $f, g \in \mathbb{Z}[X]$ mit $m = \deg(f)$ und $n = \deg(g)$ N eine obere Schranke für den Betrag der Koeffizienten ist, ist

$$N^{m+n}(m+1)^{\frac{n}{2}}(n+1)^{\frac{m}{2}}$$

eine obere Schranke für alle zur Laufzeit des Algorithmus auftretenden Koeffizienten. Die Anzahl der notwendigen Elementaroperationen ist in diesem

Fall von der Ordnung

$$\mathcal{O}(n^4(\ln(Nn))^2).$$

Der direkte Vergleich des obigen Algorithmus mit dem verallgemeinerten euklidischen Algorithmus 2.5 zeigt, dass sich die von beiden Algorithmen erzeugten Sequenzen von Polynomen nur um Faktoren unterscheiden. Damit ist, um die Richtigkeit des Algorithmus nachzuweisen, nur zu zeigen, dass der Faktor gh^δ, durch welchen der euklidische Rest r geteilt wird, ein Faktor des ggTs der Koeffizienten von r ist, d. h. , dass alle Operationen im faktoriellen Ring R liegen. *Knuth* liefert in [KNU] eine Beweisskizze an Hand eines Beispiels. Im Rahmen eines Vortrags zum AlZAGK-Seminar wurde dieser Beweis in allgemeiner Form von Prof. E. Oeljeklaus ausgeführt. Ein Beweis, der konsequent der Resultanten-/Subresultanten-Theorie folgt und somit auch die Namensgebung für den Algorithmus erklärt, findet sich in [GCL], S. 285-300. Der dort dargestellte Beweis folgt der im nun folgenden Schritt einzuführenden theoretischen Betrachtungsweise.

Definition 2.2 (Sylvester-Matrix) *Seien $f, g \in R[X]$, R faktorieller Ring und $f \neq 0 \neq g$ mit $f = \sum_{i=0}^{m} a_i X^i$ und $g = \sum_{j=0}^{n} b_j X^j$. Die Sylvester-Matrix ist dann definiert als die $(m+n) \times (m+n)$-Matrix M mit*

$$M = \begin{pmatrix}
a_m & a_{m-1} & a_{m-2} & \dots & a_1 & a_0 & 0 & 0 & \dots & 0 \\
0 & a_m & a_{m-1} & a_{m-2} & \dots & a_1 & a_0 & 0 & 0\dots & 0 \\
0 & 0 & a_m & a_{m-1} & a_{m-2} & \dots & a_1 & a_0 & 0\dots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \dots & 0 & a_m & a_{m-1} & \dots & a_2 & a_1 & a_0 \\
b_n & b_{n-1} & \dots & b_2 & b_1 & b_0 & 0 & 0 & \dots & 0 \\
0 & b_n & b_{n-1} & \dots & b_2 & b_1 & b_0 & 0 & \dots & 0 \\
0 & 0 & b_n & b_{n-1} & \dots & b_2 & b_1 & b_0 & \dots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \dots & 0 & b_n & b_{n-1} & \dots & b_2 & b_1 & b_0
\end{pmatrix},$$

wobei die Koeffizienten von f in $n = \deg(g)$ Reihen und die von g in $m = \deg(f)$ Reihen wiederholt werden.

Definition 2.3 (Resultante) *Die Resultante zweier Polynome $f, g \in R[X]$ ist definiert als*

$$\operatorname{res}(f,g) = \begin{cases} \det(M) & \text{für } f \neq 0 \text{ und } g \neq 0, \\ 0 & \text{für } f = 0 \text{ oder } g = 0, \\ 1 & \text{für } f, g \in R, \end{cases}$$

wobei M die zu f und g gehörige Sylvester-Matrix ist.

Bemerkung: Aus der Definition folgt unmittelbar, dass

$$\mathrm{res}(g,f) = (-1)^{mn}\mathrm{res}(f,g)$$

gilt.

Satz 2.3 *Seien $f,g \in R[X]$ mit $m = \deg(f) > 0$, $n = \deg(g) > 0$. Dann existieren Polynome $v,w \in R[X]$ mit $\deg(v) < n, \deg(w) < m$, so dass*

$$fv + gw = \mathrm{res}(f,g). \tag{2.2}$$

Beweis: Sei $\mathrm{res}(f,g) \neq 0$ angenommen, der Fall $\mathrm{res}(f,g) = 0$ ist unmittelbar klar. Man betrachte die $m+n$ linearen Gleichungen die durch

$$M \begin{pmatrix} X^{m+n-1} \\ \vdots \\ X^{m-1} \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} X^{n-1}f \\ \vdots \\ X^{m-1}g \\ \vdots \\ g \end{pmatrix}$$

gegeben sind. Löst man dieses System für die letzte Komponente mittels der *Kramerschen Regel*, so erhält man

$$\det(M) = \det \begin{pmatrix} a_m & a_{m-1} & \dots & a_1 & a_0 & 0 & \dots & 0 & X^{n-1}f \\ 0 & a_m & a_{m-1} & \dots & a_1 & a_0 & \dots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & 0 & a_m & a_{m-1} & \dots & a_2 & a_1 & f \\ b_n & b_{n-1} & \dots & b_1 & b_0 & 0 & \dots & 0 & X^{m-1}g \\ 0 & b_m & b_{n-1} & \dots & b_1 & b_0 & \dots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & 0 & b_m & b_{m-1} & \dots & b_2 & b_1 & g \end{pmatrix},$$

die Entwicklung nach der letzten Spalte liefert dann die Behauptung.

□

Als Korollar erhält man das sog. *Kriterium von Sylvester.*

Korollar 2.1 (Kriterium von Sylvester) *Seien $f,g \in R[X]$, R faktorieller Ring. Dann haben f,g genau dann nicht-triviale Teiler, wenn $\mathrm{res}(f,g) = 0$ gilt.*

Beweis: Falls $\mathrm{res}(f,g) \neq 0$ gilt, folgt, dass jeder Teiler von f und g auch die Resultante teilen muss. Da diese konstant ist, muss der gemeinsame Teiler den Grad 0 haben. Also existieren nur triviale Teiler. Andererseits folgt aus $\mathrm{res}(f,g) = 0$ mit Gleichung 2.2 $fv = -wg$. Falls es nun keine nicht-trivialen Teiler von f und g gäbe, müßte $g \mid v$ folgen, was aber wegen $\deg(v) < \deg(g)$ nicht möglich ist.

□

Satz 2.4 *Sei R ein faktorieller Ring, K der zugehörige Quotientenkörper und $\overline{K}$ ein algebraischer Abschluss von K. Seien weiter $f, g \in R[X]$ mit $m = \deg(f) > 0$, $n = \deg(g) > 0$ und seien α_i bzw. β_j die m bzw. n Nullstellen von f bzw. g über $\overline{K}$. Dann gilt*

$$\begin{aligned} \operatorname{res}(f,g) &= a_m^n \prod_{i=1}^{m} g(\alpha_i) \\ &= (-1)^{mn} b_n^m \prod_{j=1}^{n} f(\beta_j) \\ &= a_m^n b_n^m \prod_{1\le i\le m, 1\le j\le n} (\alpha_i - \beta_j). \end{aligned}$$

Beweis: Sei M die zu f, g gehörige Sylvester-Matrix mit $\operatorname{res}(f,g) = \det(M)$, und sei zunächst angenommen, dass die α_i, β_j alle verschieden sind. Dann betrachte man die $(n+m) \times (n+m)$ Vandermond-Matrix $V = (\nu_{i,j})$ mit

$$\nu_{i,j} = \begin{cases} \beta_j^{m+n-i} & \text{für } 1 \le j \le n \\ \alpha_{j-n}^{m+n-i} & \text{für } n+1 \le j \le n+m \end{cases}, \text{ also}$$

$$V = \begin{pmatrix} \beta_1^{m+n-1} & \dots & \beta_n^{m+n-1} & \alpha_1^{m+n-1} & \dots & \alpha_m^{m+n-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ 1 & \dots & 1 & 1 & \dots & 1 \end{pmatrix}.$$

Dann gilt zunächst $\det(V) \neq 0$, da die α_i, β_j nach Annahme alle verschieden sind. Weiter folgt für die Vandermond-Matrix V

$$\det(V) = \prod_{1\le i<j\le n} (\beta_i - \beta_j) \cdot \prod_{1\le i<j\le m} (\alpha_i - \alpha_j) \cdot \prod_{1\le i\le n, 1\le j\le m} (\beta_i - \alpha_j). \tag{2.3}$$

Betrachtet man nun das Produkt MV, so erhält man

$$MV = \begin{pmatrix} \beta_1^{n-1} f(\beta_1) & \dots & \beta_n^{n-1} f(\beta_n) & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ f(\beta_1) & \dots & f(\beta_n) & 0 & \dots & 0 \\ 0 & \dots & 0 & \alpha_1^{m-1} g(\alpha_1) & \dots & \alpha_m^{m-1} g(\alpha_m) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & g(\alpha_1) & \dots & g(\alpha_m) \end{pmatrix}.$$

Damit ist $\det(MV)$ als das Produkt zweier Blockdeterminanten gegeben, welche wieder die Form von Vandermond-Determinanten haben. Insgesamt ergibt sich daher

$$\det(MV) = f(\beta_1)\cdots f(\beta_n)g(\alpha_1)\cdots g(\alpha_m) \prod_{1\le i<j\le n} (\beta_i-\beta_j) \prod_{1\le i<j\le m} (\alpha_i-\alpha_j).$$

Wegen $det(V) \neq 0$ und $\det(M) = \operatorname{res}(f,g)$ folgt mit Gleichung 2.3

$$\operatorname{res}(f,g) \prod_{1\le i\le n, 1\le j\le m} (\beta_i - \alpha_j) = f(\beta_1)\cdots f(\beta_n)g(\alpha_1)\cdots g(\alpha_m).$$

Beachtet man zusätzlich, dass

$$a_m^n \prod_{1\le i\le n, 1\le j\le m} (\beta_i - \alpha_j) = f(\beta_1)\cdots f(\beta_n),$$

bzw. dass

$$\begin{aligned}(-1)^{mn} b_n^m \prod_{1\le i\le n, 1\le j\le m} (\beta_i - \alpha_j) &= b_n^m \prod_{1\le i\le n, 1\le j\le m} (\alpha_j - \beta_i) \\ &= g(\alpha_1)\cdots g(\alpha_m)\end{aligned}$$

gilt, dann erhält man die Behauptung des Satzes, falls die α_i, β_j alle verschieden sind.

Die allgemeine Behauptung folgert man (s. a. [HEA]) dadurch, dass man die α_i, β_j als verschiedene formale Variablen auffasst, die durch die Bedingung $f(\alpha_i) = 0$ und $g(\beta_i) = 0$ definiert sind. Dadurch wird gewährleistet, dass $\det(V) \neq 0$ gegeben ist. Somit behält obiger Beweis seine Gültigkeit, wenn im letzten Schritt die formalen Variablen durch die realen Nullstellen von f und g ersetzt werden.

□

Mit dem bisher Gezeigten ist es bereits möglich, den folgenden Algorithmus festzuhalten.

Algorithmus 2.7 (Bestimmung der Resultante) *Seien $u, v \in R[X]$ und R faktorieller Ring. Der Algorithmus bestimmt die Resultante von u und v, wobei lediglich Operationen in R ausgeführt werden.*

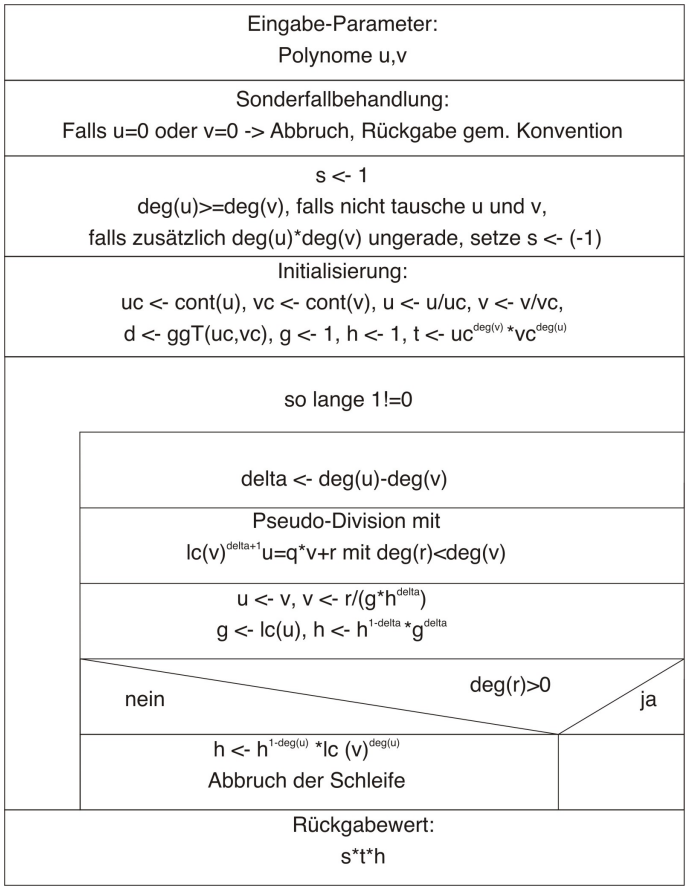

Diagramm 7: Algorithmus zur Bestimmung der Resultante
Sourcen-Code siehe subresultant.par*, Anhang S.124*

Beweis: Setze $u_0 = u$, $u_1 = v$ und sei u_i die vom Algorithmus erzeugte Sequenz von Polynomen. Ferner sei r_i der Rest der Pseudo-Division und t der Index für den der Grad von u_{t+1} 0 ist. Weiterhin gelten folgende Vereinbarungen im Zustand i:

$$d_i = \deg(u_i) \quad \text{und} \quad l_i = \mathrm{lc}(u_i),$$

g_i, h_i entsprechen den Bezeichnern g, h im Algorithmus. Im Fall $i = 0$ gilt also $g_0 = h_0 = 1$. Außerdem sei $\delta_i = d_i - d_{i+1}$. Aufgrund von Satz 2.4 gilt für $i \geq 1$ und den Fall, dass die β_j die Nullstellen von u_i über dem algebraischen

Abschluss des Quotientenkörpers K von R sind, die nachfolgende Aussage:

$$\begin{aligned}
\operatorname{res}(u_{i-1},u_i) &= (-1)^{d_{i-1}d_i} l_i^{d_{i-1}} \prod_{1\le j\le d_i} u_{i-1}(\beta_j) \\
&= (-1)^{d_{i-1}d_i} l_i^{d_{i-1}} \prod_{1\le j\le d_i} \frac{r_{i-1}(\beta_j)}{l_i^{\delta_{i-1}+1}} \\
&\qquad \text{mit } \operatorname{lc}(u_{i+1})^{\delta_i+1} u_i = u_{i+1}q + r_i \text{ und } u_i(\beta_j) = 0 \\
&= (-1)^{d_{i-1}d_i} l_i^{d_{i-1}-d_i(\delta_{i-1}+1)} \prod_{1\le j\le d_i} r_{i-1}(\beta_j) \\
&= (-1)^{d_{i-1}d_i} l_i^{d_{i-1}-d_i(\delta_{i-1}+1)} \frac{1}{l_i^{d_{i+1}}} \operatorname{res}(u_i, g_{i-1}h_{i-1}^{\delta_{i-1}} u_{i+1}) \\
&\qquad \text{mit } u_{i+1} = \frac{r_{i-1}}{g_{i-1}h_{i-1}^{\delta_{i-1}}} \text{ und Satz 2.4}
\end{aligned}$$

Mit der Definition der Resultante folgt für $c \in R$ unmittelbar $\operatorname{res}(f,cg) = c^{\deg(f)}\operatorname{res}(f,g)$. Somit gilt mit dem oben Gezeigten

$$\operatorname{res}(u_{i-1},u_i) = (-1)^{d_{i-1}d_i} l_i^{d_{i-1}-d_i(\delta_{i-1}+1)-d_{i+1}} (g_{i-1}h_{i-1}^{\delta_{i-1}})^{d_i} \operatorname{res}(u_i,u_{i+1}).$$

Nutzt man nun noch die beiden im Falle $i \ge 1$ gültigen Identitäten $g_i = l_i$ und $h_i = h_{i-1}^{1-\delta_{i-1}} g_i^{\delta_{i-1}}$ aus, so erhält man

$$\begin{aligned}
\operatorname{res}(u_{i-1},u_i) &= (-1)^{d_{i-1}d_i} \cdot \frac{g_{i-1}^{d_i} g_i^{d_{i-1}} (h_{i-1}^{\delta_{i-1}})^{d_i}}{g_i^{d_{i+1}} g_i^{d_i(\delta_{i-1}+1)}} \cdot \operatorname{res}(u_i,u_{i+1}) \\
&= (-1)^{d_{i-1}d_i} \cdot \frac{g_{i-1}^{d_i} (h_{i-1}^{\delta_{i-1}})^{d_i-1} h_{i-1}^{\delta_{i-1}}}{g_i^{d_{i+1}} g_i^{\delta_{i-1}(d_i-1)}} \cdot \operatorname{res}(u_i,u_{i+1}) \\
&= (-1)^{d_{i-1}d_i} \cdot \frac{g_{i-1}^{d_i} (h_{i-1}^{\delta_{i-1}})^{d_i-1} h_{i-1}^{\delta_{i-1}}}{g_i^{d_{i+1}} \left(\frac{h_i}{h_{i-1}^{1-\delta_{i-1}}}\right)^{d_i-1}} \cdot \operatorname{res}(u_i,u_{i+1}) \\
&= (-1)^{d_{i-1}d_i} \cdot \frac{g_{i-1}^{d_i} h_{i-1}^{d_{i-1}-1}}{g_i^{d_{i+1}} h_i^{d_i-1}} \cdot \operatorname{res}(u_i,u_{i+1}),
\end{aligned}$$

und mit der Tatsache, dass t der Index der letzten Iteration ist, erhält man nach t-maliger Anwendung der o. g . Beziehung

$$\mathrm{res}(u_0, u_1) = (-1)^{\sum_{i=0}^{t} d_i d_{i+1}} \frac{g_0^{d_1} h_0^{d_0 - 1}}{g_t^{d_{t+1}} h_t^{d_t - 1}} \mathrm{res}(u_t, u_{t+1}).$$

Zusammen mit $g_0 = h_0 = 1$ und $u_{t+1} \in R$ (d. h. $\mathrm{res}(u_t, u_{t+1}) = u_{t+1}^{d_t}$) erhält man

$$\mathrm{res}(u_0, u_1) = s_t h_t^{1-d_t} u_{t+1}^{d_t},$$

wobei s_i dem Bezeichner s aus dem Algorithmus im Zustand i entspricht. Damit liefert der Algorithmus das geforderte Resultat im Falle von primitiven Polynomen. Die Modifikation für beliebige Polynome folgert man direkt aus der Beziehung $\mathrm{res}(f, cg) = c^{\deg(f)} \mathrm{res}(f, g)$ für beliebiges $c \in R$.

□

Bemerkung zur Realisierung: Aufgrund der offensichtlichen gemeinsamen Struktur der beiden Algorithmen 2.6 und 2.7 wurden diese in angepaßter Form in einem Algorithmus realisiert, wobei die Realsierung im Wesentlichen der Berechnung eines ggTs zweier Polynome dient, optional wird dabei auch die Resultante bestimmt.

3 Allgemeines zur Polynom-Faktorisierung

Dieses Kapitel befaßt sich mit der allgemeinen Problematik der Faktorisierung von Polynomen über $\mathbb{F}_p$ oder $(\mathbb{Z}/p^k\mathbb{Z})[X]$. Dabei wird der klassische Weg zur Herausarbeitung eines Algorithmus zur Faktorisierung eingeschlagen. Zunächst wird ausführlich die Faktorisierung in $\mathbb{F}_p[X]$ behandelt, da diese mit Ihren Algorithmen die Grundlage fast aller Faktorisierungsverfahren für Polynome über $\mathbb{Z}$ bildet. Mit dem anschließend beschriebenen *Henselschen Lemma* und der daraus resultierenden Möglichkeit des Liftens einer Faktorisierung, sowie mit der Existenz der im Anhang aufgeführten Polynomialabschätzungen kann dann im nächsten Kapitel ein allgemeiner Algorithmus zur Faktorisierung von Polynomen über $\mathbb{Z}$ als logische Konsequenz der vorhergehenden Sätze und Algorithmen angegeben werden.

3.1 Faktorisierung von Polynomen über $\mathbb{F}_p$

Da $\mathbb{F}_p$ ein Körper ist, können wir o. B. d. A annehmen, dass $f \in \mathbb{F}_p[X]$ ein normiertes Polynom ist; gemäß Anhang ist $\mathbb{F}_p[X]$ faktorieller Ring, und es gelten die dort aufgeführten Eigenschaften und Sätze. Die Tatsache, dass an dieser Stelle nur endliche Grundkörper betrachtet werden, hat bereits zur Folge, dass alle Faktoren einer Faktorisierung in endlich vielen Schritten gefunden werden können.
Allgemein gliedert sich die Faktorisierung eines normierten, nicht-konstanten Polynoms in drei Hauptschritte, welche im Folgenden, noch schrittweise zu verfeinernden Algorithmus wiedergegeben werden.

Algorithmus 3.1 (Faktorisierung in $\mathbb{F}_p[X]$)
Sei $f \in \mathbb{F}_p[X]$, $n = \deg(f) \geq 1$, und es gelte $lc(f) = 1$. Der Algorithmus faktorisiert f als ein Produkt von irreduziblen Polynomen.

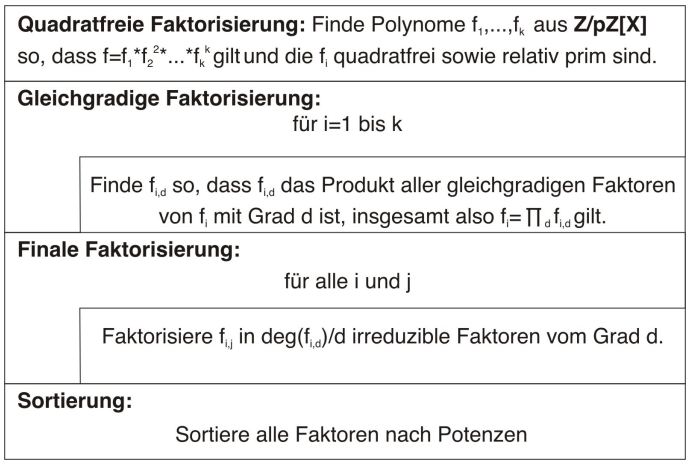

Diagramm 8: Faktorisierung in $\mathbb{F}_p[X]$

Bemerkung: Bei Cohen [COH] wird im letzten Schritt noch neben dem Sortieren der Faktoren ein Gruppieren aller identischen Faktoren vorgeschlagen; dies ist aufgrund der im Algorithmus gemachten Annahmen überflüssig. Denn wäre h für ein i Faktor von beispielsweise $f_{i,d}$ und $f_{i,e}$, dann könnte f_i nicht quadratfrei sein. Der Fall, dass h ein Faktor von $f_{i,d}$ und $f_{j,e}$ mit $i \neq j$ sein könnte, widerspricht der Tatsache, dass die f_i relativ prim sind.

Da einige der nachfolgenden Beweise Bezug auf den *Chinesischen Restsatz für Polynome* nehmen, sei dieser zunächst aufgeführt.

Satz 3.1 (Chinesischer Restsatz für Polynome) *Seien $f_1, \ldots, f_r$ Polynome aus $\mathbb{F}_p[X]$, und es gelte $f = f_1 \cdots f_r$, sowie $\mathrm{ggT}(f_i, f_j) = 1$ für alle $i \neq j$, $i, j \in \{1, \ldots, r\}$ (oder äquivalent f_i eindeutig, und irreduzibel in $\mathbb{F}_p[X]$), und seien weiter $g_1, \ldots, g_r$* beliebige *Polynome aus $\mathbb{F}_p[X]$. Dann existiert ein eindeutig bestimmtes Polynom $h \in \mathbb{F}_p[X]$ mit*

$$\deg(h) < \sum_{j=1}^{r} \deg(f_j)$$

und

$$h \equiv g_i \mod f_i \quad \textit{für} \quad 1 \leq i \leq r.$$

Bemerkung: *In anderen Worten bedeutet dies, dass die Abbildung*

$$\Phi : \mathbb{F}_p[X]/f \to \oplus_{i=1}^{r} \mathbb{F}_p[X]/f_i$$
$$\textit{mit}$$
$$t \mapsto [g_1, \ldots, g_r],$$

wobei $t \equiv g_i \mod f_i$ für $1 \leq i \leq r$ gilt, bijektiv ist.

Beweis: Mittels des erweiterten euklidischen Algorithmus bestimmt man Polynome m_i für $1 \leq i \leq r$ so, dass

$$m_i \prod_{j \neq i} f_j \equiv 1 \mod f_i$$

gilt. Setzt man nun

$$h := \sum_{i=1}^{r} g_i m_i \prod_{j \neq i} f_j,$$

dann gilt $h \equiv g_i \mod f_i$. Bleibt die Eindeutigkeit zu zeigen. Angenommen t wäre auch eine Lösung der Kongruenzen des Satzes, dann würde für alle $i \in \{1, \ldots, r\}$ gelten: $f_i \mid (t - h)$; also $t \equiv h \mod \prod_{i=1}^{r} f_i$.

□

3.1.1 Faktorisierung in Produkte quadratfreier Polynomen

Im einleitenden Abschnitt taucht innerhalb der Beschreibung des allgemeinen Algorithmus zur Faktorisierung bereits der Begriff des *quadratfreien Polynoms* auf, ohne näher spezifiziert zu sein. Für die Betrachtung der mathematischen Grundlagen sei dies an dieser Stelle nachgeholt.

Definition 3.1 *Sei $f \in R[X]$ und R faktorieller Ring. f ist* quadratfrei, *falls kein $g \in R[X]$ mit* $\deg(g) \geq 1$ *existiert, so dass $g^2 \mid f$ gilt.*
Eine quadratfreie Faktorisierung *von f ist gegeben, falls*

$$f = \prod_{i=1}^{k} f_i^i$$

mit f_i ist quadratfrei für $i \in \{1, \ldots, k\}$ und die $f_1, \ldots, f_k$ relative prim sind.

Mit der Bestimmung einer quadratfreien Faktorisierung von f erhält man also alle sich wiederholenden Faktoren, und das Problem der Faktorisierung reduziert sich auf Faktoren ohne Wiederholungen.
Sei nun $f = a_n X^n + \cdots + a_0 \in \mathbb{F}_p[X]$. Dann ist die formale Ableitung $f' = n a_n X^{n-1} + \cdots a_1$ genau dann Null, wenn $i a_i = 0$ für alle $i \in 0, \ldots, n$. Dies ist genau dann der Fall, wenn $p \mid i$ oder $a_i = 0$ für alle $i \in 0, \ldots, n$ gilt. Daraus folgert man direkt:

Lemma 3.1 *Sei $f \in \mathbb{F}_p[X]$,* $\deg(f) \geq 1$. *Dann gilt $f' = 0$ genau dann, wenn f Polynom in X^p ist.*

Die Grundlage der quadratfreien Faktorisierung bilden die nachfolgenden drei Sätze.

Satz 3.2 *Sei $f \in \mathbb{F}_p[X]$, f primitiv und* $\deg(f) \geq 1$. *Sei*

$$f = f_1^{e_1} f_2^{e_2} \cdots f_k^{e_k}$$

die eindeutige Faktorisierung in irreduzible Faktoren. f' sei die formale Ableitung von f. Dann gilt

$$\mathrm{ggT}(f, f') = f_1^{e_1 - \delta_1} \cdots f_k^{e_k - \delta_k}$$

mit

$$\delta_i = \begin{cases} 0 & , \textit{falls } e_i f_i' = 0, \textit{d. h. } f_i' = 0 \textit{ oder } p \mid e_i, \\ 1 & , \textit{sonst.} \end{cases}$$

Beweis: Sei $q = \prod_{2 \leq i \leq n} f_i^{e_i}$ und $r = \mathrm{ggT}(f, f')$. Dann ist $f = f_1^{e_1} q$ und $f' = f_1^{e_1} q' + e_1 f_1^{e_1 - 1} f_1' q$. Damit folgt $f_1^{e_1 - 1} \mid r$. Fall $f_1' = 0$ oder $p \mid e_1$ folgt offensichtlich auch $f_1^{e_1} \mid r$. Zeige nun $f_1^{e_1} \nmid r$ falls $p \nmid e_1$ und $f_1' \neq 0$.

Annahme: $f_1^{e_1} \mid r$, also $f_1^{e_1} \mid f'$, und d. h. $f_1^{e_1} \mid e_1 f_1^{e_1-1} f_1' q$. Nach Kürzung bedeutet dies $f_1 \mid e_1 f_1' q$. Da die f_i allerdings relativ prim sind, folgt $\mathrm{ggT}(f_1, q) = 1$, also $f_1 \mid e_1 f_1'$. Daher würde $\deg(f_1) \leq \deg(f_1')$ folgen, Widerspruch. Wegen der Symmetrie des Beweises folgt die Behauptung für alle f_i.

□

Sei nun $f = f_1^{e_1} \cdots f_k^{e_k}$ und e das Maximum der e_i mit $i \in \{1, \ldots, k\}$, und man definiere die Polynome s_i für alle i mittels

$$s_i = \begin{cases} \prod\{f_j | \quad e_j = i \text{ und } f_j' \neq 0\} & , \text{falls } p \nmid i \\ 1 & , sonst \end{cases},$$

sowie anschließend s über

$$s = \prod\{f_j^{e_j} | \quad p \mid e_j \text{ oder } f_j' = 0\}.$$

Dann gilt $f = (s_1 s_2^2 \cdots s_e^e) s$, und außerdem folgt aufgrund der Definition der s_i, $\mathrm{ggT}(s_i, s_i') = 1$ und $s' = 0$. Also gilt nach dem vorherigen Satz 3.2

$$\mathrm{ggT}(f, f') = s_2 s_3^2 \cdots s_e^{e-1} s,$$

also

$$\frac{f}{\mathrm{ggT}(f, f')} = s_1 s_2 \cdots s_e.$$

Abgesehen von der Behandlung des Sonderfalls, dass das Polynom f einen Faktor, welcher ein Polynom in X^P ist, hat, ist damit bereits der Algorithmus vorgezeichnet.

Satz 3.3 *Sei $f \in \mathbb{F}_p[X]$, dann gilt*

$$(f(X))^p = f(X^p).$$

Beweis: Seien $f_1, f_2 \in \mathbb{F}_p[X]$, dann gilt

$$\begin{aligned} (f_1 + f_2)^p &= f_1^p + \binom{p}{1} f_1^{p-1} f_2 + \cdots + \binom{p}{p-1} f_1 f_2^{p-1} + f_2^p \\ &= f_1^p + f_2^p, \end{aligned}$$

da die obigen Binomialkoeffizienten alle Vielfache von p sind. Mittels des kleinen Satzes von Fermat folgt dann schließlich für $f(X) = a_n X^n + \cdots + a_0$:

$$\begin{aligned} (f(X))^p &= (a_n X^n)^p + (a_{n-1} X^{n-1})^p + \cdots + (a_0)^p \\ &= a_n X^{np} + a_{n-1} X^{(n-1)p} + \cdots + a_0^p = f(X^p). \end{aligned}$$

□

Satz 3.4 *Sei $f \in \mathbb{F}_p[X]$. f' ist genau dann Null, falls ein Polynom $g \in \mathbb{F}_p[X]$ existiert mit $f = g^p$.*

Beweis: Falls $f = g^p$ ist, gilt $f' = pg^{p-1}g' = 0$. Falls nun umgekehrt $f' = 0$ gilt, so existiert für f eine Darstellung der Form $f = a_{kp}X^{kp} + a_{k-1}X^{(k-1)p} + \cdots + a_0$. Für $g = a_k^{1/p}X^k + a_{k-1}^{1/p}X^{(k-1)p} + \cdots + a_0^{1/p}$ folgt dann die Behauptung.

□

Damit haben wir folgenden Algorithmus:

Algorithmus 3.2 (Quadratfreie Faktorisierung) *Für ein normiertes Polynom $f \in \mathbb{F}_p[X]$ liefert der Algorithmus eine Faktorisierung in ein Produkt von Potenzen quadratfreier Polynome.*

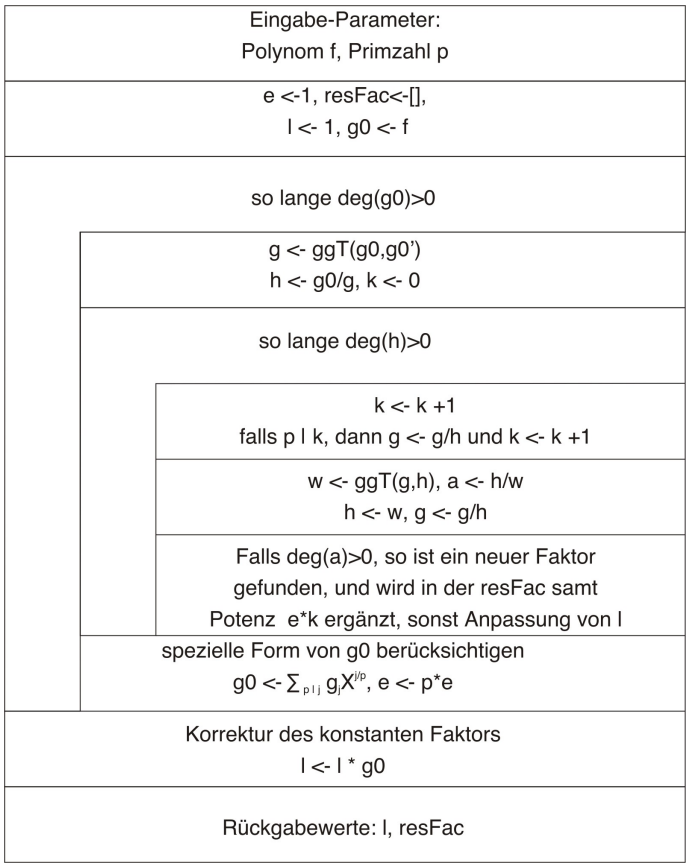

Diagramm 9: Quadratfreie Faktorisierung
Sourcen-Code siehe squarefree_fac.par, *Anhang S.130*

3.1.2 Gleichgradige Faktorisierung

Zunächst können wir nach dem vorherigen Abschnitt davon ausgehen, dass f ein quadratfreies Polynom ist, welches nun in Produkte von irreduziblen Faktoren gleichen Grades zerlegt werden soll. Um dies zu erreichen, betrachte man den folgenden Satz.

Satz 3.5 *Sei* $g \in \mathbb{F}_p[X]$, $d = \deg(g)$ *und* g *irreduzibel. Dann ist* $K = \mathbb{F}_p[X]/g\mathbb{F}_p[X]$ *ein Körper.*

Beweis: $\mathbb{F}_p[X]/g\mathbb{F}_p[X]$ ist offensichtlich ein Integritätsring, bleibt nur zu zeigen, dass $K^* = K\backslash\{0\}$ ist. Sei also $z \in K\backslash\{0\}$ und $h \in \mathbb{F}_p[X]$ mit $z = [h]_g$ (Restklassenbetrachtung). Da $z \neq 0$ ist, gilt $h \notin g\mathbb{F}_p[X]$, also $g \nmid h$ und somit $\mathrm{ggT}(g,h) = 1$. Daher existieren $v, w \in \mathbb{F}_p[X]$ mit $vg + hw = 1$. Es folgt $hw = 1 - vg \equiv 1 \mod g$ und damit

$$z[w]_g = [h]_g[w]_g = [hw]_g = [1_{\mathbb{F}_p}]_g = 1_K.$$

Also ist z Einheit in K.

□

Bemerkung: K ist endlich, es gilt $\#K = p^d$, und somit gilt nach dem kleinen Satz von Fermat für alle $z \in K^*$ $z^{p^d-1} = 1$ bzw. $z^{p^d} = z$

Satz 3.6 *Sei* $f \in \mathbb{F}_p[X]$ *irreduzibel mit* $n = \deg(f)$*, und sei* $m \in \mathbb{N}^*$*. Dann gilt:*

$$f \mid X^{p^m} - X \text{ in } \mathbb{F}_p[X] \quad \Leftrightarrow \quad n \mid m.$$

Beweis: Sei $z \in K = \mathbb{F}_p[X]/f\mathbb{F}_p[X]$. Aufgrund der Bemerkung gilt $z^{p^n} = z^p = z$, also fortsetzbar auch $z^{p^{2n}} = (z^p)^{p^n} = z^{p^n}$, usw., so dass man für jedes $z \in K$ und jedes $k \in \mathbb{N}$ $z^{p^{kn}} = z$ erhält.
'⇐': Es gelte $n \mid m$, dann ist $m = kn$ für ein $k \in \mathbb{N}$ und $[X^{p^{kn}}]_f = [X]_f^{p^{kn}} = z^{p^{kn}} = z = [X]_f$, also $X^{p^m} \equiv X \mod f$ und damit $f \mid X^{p^m} - X$.
'⇒': Es gelte $f \mid X^{p^m} - X$. Euklidische Division über $\mathbb{Z}$ liefert $k, r \in \mathbb{N}$ mit $m = kn + r$ und $r < n$. Mit $z := [X]_f$ gilt

$$z = [X]_f = [X]_f^{p^m} = z^{p^m} = z^{p^{kn+r}} = (z^{p^{kn}})^{p^r} = z^{p^r}.$$

Sei $g \in K$, dann existieren Koeffizienten g_i mit $g = \sum_{i=0}^{n-1} g_i z^i$ und da p^m Potenz von $\mathrm{char}(K) = p^n$ ist folgt mit Satz 3.3 $(g(z))^{p^r} = g(z^{p^r}) = g(z)$. Also ist jedes $g \in K$ eine Nullstelle von $X^{p^r} - X \in \mathbb{F}_p[X] \subset K[X]$. Wäre $r \geq 1$, dann hätte $X^{p^r} - X$ $p^n = \#K > p^r = \deg(X^{p^r} - X)$ Nullstellen, Widerspruch. Damit ist $r = 0$, also $m = kn$.

□

Als direkte Folgerung ergibt sich der für den gesuchten Algorithmus entscheidende Satz.

Satz 3.7 *Sei $m \in \mathbb{N}^*$ und $g = X^{p^m} - X \in \mathbb{F}_p[X]$, dann ist g das Produkt der über $\mathbb{F}_p$ normierten, irreduziblen Polynome g_i mit* $\deg(g_i) \mid m$.

Beweis: Seien $g_1, \ldots, g_s \in \mathbb{F}_p[X]$ die normierten, irreduziblen Polynome, für die $\deg(g_i) \mid m$ $(1 \leq i \leq s)$ gilt. Nach dem Satz 3.6 sind die g_i Teiler von g; zudem gilt $\mathrm{ggT}(g_i, g_j) = 1$ für $i \neq j$. Insgesamt gilt also $g_1 \cdots g_s \mid g$. Aufgrund von $g' = -1$ ist $\mathrm{ggT}(g, g') = -1$, d. h. g quadratfrei. Also existieren paarweise verschiedene, irreduzible Polynome $h_1, \ldots, h_r$ mit $g = h_1 \cdots h_r$. Für jedes $i \in \{1, \ldots, r\}$ gilt nun $h_i \mid g$ und damit $\deg(h_i) \mid m$ nach Satz 3.6. Somit erhält man $h_i \in \{f_1, \ldots, f_s\}$, also $g = (h_1 \cdots h_r) \mid f_1 \cdots f_s$. D. h. $g = f_1 \cdots f_s$.

□

Algorithmus 3.3 (Gleichgradige Faktorisierung) *Für ein quadratfreies Polynom $f \in \mathbb{F}_p[X]$ liefert der Algorithmus eine Faktorisierung in ein Produkt gleichgradiger, irreduzibler Polynome.*

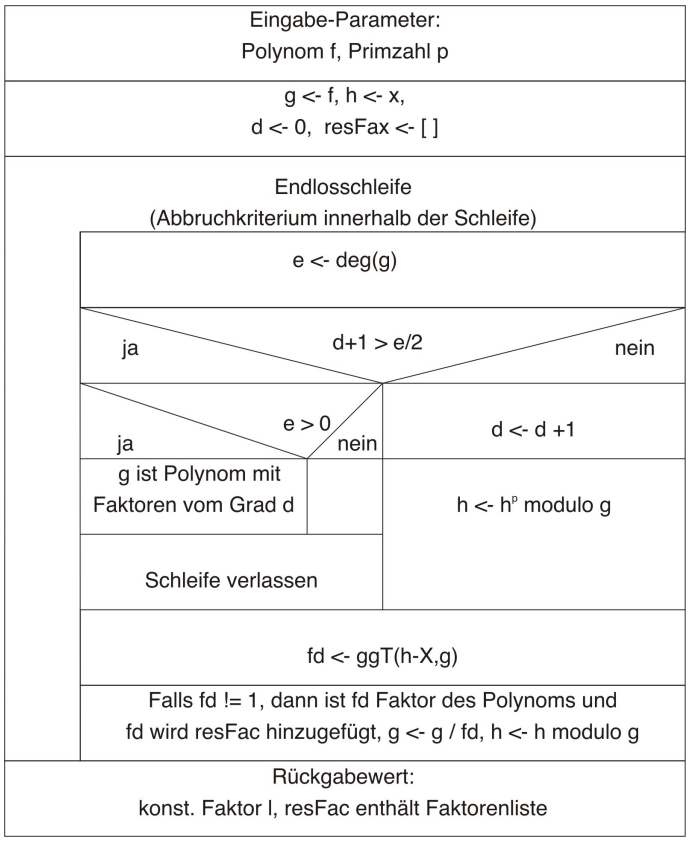

Diagramm 10: Gleichgradige Faktorisierung
Sourcen-Code siehe distinct_fac.par, *Anhang S.132*

3.1.3 Finale Faktorisierung

Unter Berücksichtigung der vorangehenden Algorithmen ist zum Schluss folgende Aufgabenstellung zu lösen: Gegeben ist ein quadratfreies Polynom $f \in \mathbb{F}_p[X]$, welches in gleichgradige, irreduzible Faktoren f_i mit $\deg(f_i) = d$ zerfällt. Daher wissen wir zunächst, dass

$$d \mid \deg(f)$$

gilt, und falls $\deg(f) = d$ ist, so ist f irreduzibel.
Ein Verfahren zur Lösung des obigen Problems stammt von Cantor und Zassenhaus, ein weiteres wurde von Berlekamp gefunden; beide Verfahren werden im nächsten Abschnitt realisiert.

3.2 Realisierungen zur Polynom-Faktorisierung modulo p

3.2.1 Finale Faktorisierung nach Cantor-Zassenhaus

Sei f mit Eigenschaften, wie im Abschnitt 3.1.3 beschrieben, gegeben. Man betrachte zunächst den Fall, dass $p \neq 2$ ist.

Lemma 3.2 *Sei $f \in \mathbb{F}_p[X]$ mit gen. Eigenschaften, dann gilt für alle Polynome $g \in \mathbb{F}_p[X]$ die folgende Identität:*

$$f = \mathrm{ggT}(f, g)\mathrm{ggT}(f, g^{\frac{p^d-1}{2}} + 1)\mathrm{ggT}(f, g^{\frac{p^d-1}{2}} - 1).$$

Beweis: Wie im Abschnitt 3.1.2 gesehen, sind alle Elemente von $\mathbb{F}_{p^d}$ Nullstellen des Polynoms $X^{p^d} - X$. Daher folgt, dass für jedes Polynom $g \in \mathbb{F}_p[X]$ das Polynom $g^{p^d} - g$ ebenfalls alle Elemente aus $\mathbb{F}_{p^d}$ als Nullstellen hat. D. h. insbesondere $X^{p^d} - X \mid g^{p^d} - g$. Mit Satz 3.7 weiß man, dass $X^{p^d} - X$ Vielfaches jedes irreduziblen Polynoms von Grad d ist, insbesondere also auch von f, da f quadratfreies Produkt irreduzibler Faktoren vom Grad d ist. Beachtet man weiter, dass

$$g^{p^d} - g = g(g^{p^d-1} - 1) = g(g^{\frac{p^d-1}{2}} + 1)(g^{\frac{p^d-1}{2}} - 1) \tag{3.1}$$

gilt, wobei die drei Faktoren der rechten Seite teilerfremd sind, so folgt direkt

$$f = \mathrm{ggT}(f, g^{p^d} - g) = \mathrm{ggT}(f, g)\mathrm{ggT}(f, g^{\frac{p^d-1}{2}} + 1)\mathrm{ggT}(f, g^{\frac{p^d-1}{2}} - 1).$$

□

Satz 3.8 *Sei $f \in \mathbb{F}_p[X]$, und f sei das Produkt von $r > 1$ quadratfreien, gleichgradigen, irreduziblen Polynomen $f_i \in \mathbb{F}_p[X]$ mit Grad d. Dann ist für jedes zufällige Polynom $h \in \mathbb{F}_p[X]$ der Menge der p^{rd} Polynome mit Grad$< rd$, die Wahrscheinlichkeit, dass*

$$\mathrm{ggT}(f, h^{\frac{p^d-1}{2}} - 1)$$

ein echter Faktor von f ist, gegeben durch

$$1 - \left(\frac{1}{2^r}\right)\left[\left(1 + \frac{1}{p^d}\right)^r + \left(1 - \frac{1}{p^d}\right)^r\right] > \frac{4}{9}.$$

Beweis: Aufgrund von Gleichung 3.1 gilt für $\left(\frac{p^d-1}{2}\right)$ Elemente von $\mathbb{F}_{p^d}$ $X^{\frac{p^d-1}{2}} = 1$. Außerdem hat man für die gleiche Anzahl $X^{\frac{p^d-1}{2}} = -1$, und im Falle $x = 0$ gilt $X^{\frac{p^d-1}{2}} = 0$. Mit dem Chinesischen Restsatz folgt, dass $\mathbb{F}_p[X]/f \cong \oplus_{i=1}^{r} \mathbb{F}_{p^d} = G$ gilt. Sei $g = (g_1, \ldots, g_r) \in G$, dann gilt

$$g^{\frac{p^d-1}{2}} = \left[g_1^{\frac{p^d-1}{2}}, \ldots, g_r^{\frac{p^d-1}{2}}\right], \tag{3.2}$$

und damit nehmen die Koeffizienten nur die Werte $-1, 0, 1$ an. Die Zahl der g so, dass $g^{\frac{p^d-1}{2}}$ keine Komponente mit 1 hat, ist

$$\left(\frac{p^d - 1}{2} + 1\right)^r,$$

und die Zahl der g, für die mindestens eine Komponente von $g^{\frac{p^d-1}{2}}$ 1 ist, ist

$$p^{rd} - \left(\frac{p^d - 1}{2} + 1\right)^r.$$

Der Fall, dass alle Komponenten 1 sind, kann

$$\left(\frac{p^d - 1}{2}\right)^r \text{-mal}$$

auftreten. Die Anzahl der Elemente g für die mindestens eine Komponente von $g^{\frac{p^d-1}{2}}$ 1 und mindestens eine von 1 verschieden ist, ist dann

$$p^{rd} - \left(\frac{p^d - 1}{2} + 1\right)^r - \left(\frac{p^d - 1}{2}\right)^r.$$

Für jedes g dieser Art ist $g - 1 \neq 0$, und $g - 1$ besitzt ein Inverses in G. In diesem Fall ist für das zugehörige $q \in \mathbb{F}_p[X]$ $(q^{\frac{p^d-1}{2}} - 1)$ weder relativ prim zu f, noch ist f ein Faktor. Daher ist

$$\mathrm{ggT}(f, q^{\frac{p^d-1}{2}} - 1)$$

ein echter Faktor von f. Für die Wahrscheinlichkeit folgt dann direkt

$$\frac{p^{rd} - \left(\frac{p^d-1}{2}+1\right)^r - \left(\frac{p^d-1}{2}\right)^r}{p^{rd}} = 1 - \left(\frac{1}{2^r}\right)\left[\left(1+\frac{1}{p^d}\right)^r + \left(1-\frac{1}{p^d}\right)^r\right].$$

Wegen $p^d \geq 3$ und mit $r = 2$ erhält man die letzte Abschätzung.

□

Damit ist die Wahrscheinlichkeit, dass bei 10 zufällig ausgewählten Polynomen der ggT nur ein trivialer Teiler von f ist, höchstens

$$\left(\frac{5}{9}\right)^{10} < \frac{3}{1000}.$$

Aus dieser Abschätzung zieht der folgende probabilistische Algorithmus von Cantor-Zassenhaus seine Daseinsberechtigung. Dabei handelt es sich bei der unten gezeigten Variante um den bei Cohen [COH] veröffentlichten rekursiven Algorithmus.

Algorithmus 3.4 (Cantor-Zassenhaus für $p > 2$) *Für ein quadratfreies Polynom $f \in \mathbb{F}_p[X]$, welches in gleichgradige, irreduzible Faktoren vom Grad d zerfällt, liefert der Algorithmus die Faktorisierung in diese irreduziblen Polynome.*

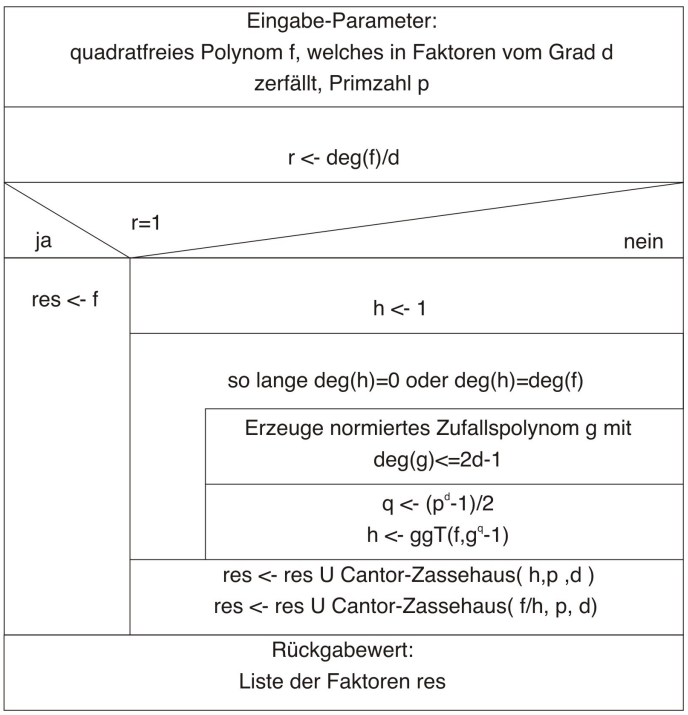

Diagramm 11: Finale Faktorisierung nach Cantor-Zassenhaus ($p > 2$)
Sourcen-Code siehe factor_czmodp.par*, Anhang S.138*

Der Fall $p = 2$ erfordert eine gesonderte Betrachtung, da hier die Gleichung 3.2 in dieser Form nicht gültig ist.

Lemma 3.3 *Sei f mit gen. Eigenschaften gegeben und $p = 2$, dann gilt für jedes Polynom $g \in \mathbb{F}_p[X]$*

$$f = \mathrm{ggT}(f, v \circ g)\mathrm{ggT}(f, v \circ g + 1)$$

mit $v = X + X^2 + X^4 + \ldots + X^{2^d-1}$.

Beweis: Mit $(v \circ g)^2 = g^2 + g^4 + \ldots + g^{2^d}$ folgt

$$(v \circ g)(v \circ g + 1) = (v \circ g)^2 + v \circ g = g^{2^d} - g \quad \text{wg. Charakteristik 2,}$$

und der restliche Beweis verläuft wie der zu Lemma 3.1.

□

Auch in diesem Fall kann wie in Satz 3.8 gezeigt werden, dass die Wahrscheinlichkeit, dass $\mathrm{ggT}(f, v \circ g)$ ein echter Teiler von f ist, nahe bei $\frac{1}{2}$ liegt. Daher wäre ein Abändern von Algorithmus 3.4 durch Tausch von $h \leftarrow \mathrm{ggT}(f, g^{\frac{p^d-1}{2}})$ gegen $h \leftarrow \mathrm{ggT}(f, v \circ g)$ innerhalb der Schleife problemlos möglich. Cohen führt allerdings auch eine Variante auf, die in diesem Fall ohne die Bestimmung eines Zufallspolynoms auskommt. Dieser Algorithmus sei im Nachfolgenden genannt.

Algorithmus 3.5 (Cantor-Zassenhaus für $p = 2$) *Für ein quadratfreies Polynom $f \in \mathbb{F}_p[X]$, welches in gleichgradige irreduzible Faktoren vom Grad d zerfällt, liefert der Algorithmus die Faktorisierung in diese irreduzible Polynome.*

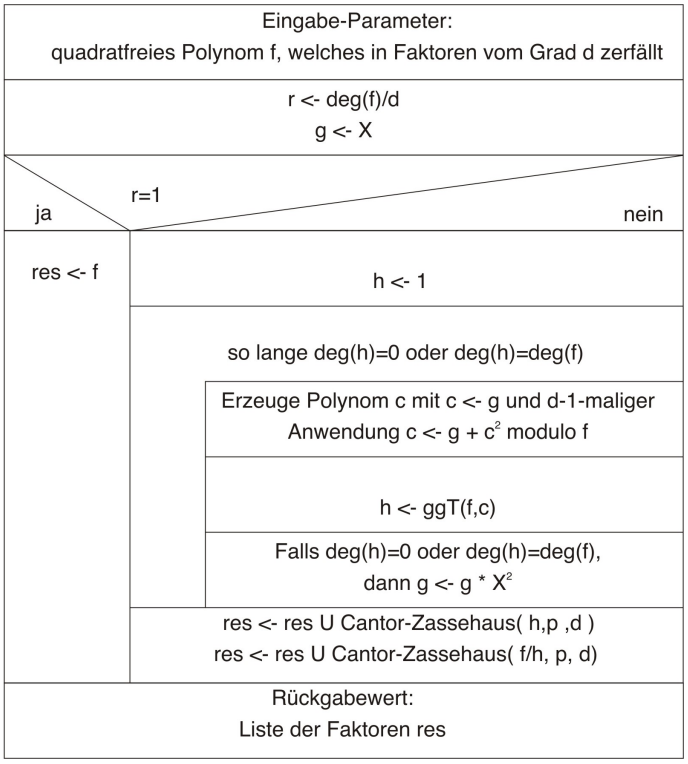

Diagramm 12: Finale Faktorisierung nach Cantor-Zassenhaus ($p = 2$)
Sourcen-Code siehe factor_czmodp.par, *Anhang S.138*

Bemerkung: Da der obige Algorithmus nicht direkt aus dem bisher gezeigten hervorgeht, bleibt die Korrektheit des Algorithmus zu zeigen. Aufgrund der Struktur des Algorithmus ist klar, dass dieser, falls er terminiert, eine Faktorisierung liefert. Ein Beweis für die Terminiertheit findet sich bei Cohen [COH], S. 129-130.

Bemerkung zur Rekursion: Die Rekursion in den beiden obigen Algorithmen kann durch das übliche Verfahren zum Entfernen einer Rekursion, die Quasi-Verwaltung eines Stacks, aufgehoben werden. Die beiden Algorithmen wurden daher in *PARI/GP* rekursionsfrei ausgeführt.

3.2.2 Faktorisierung nach Cantor-Zassenhaus

Gemäß der Beschreibung des Abschnitts 3.1 ergibt sich unter Verwendung des Verfahrens von Cantor-Zassenhaus zur finalen Faktorisierung der folgende Algorithmus.

Algorithmus 3.6 (Faktorisierung modulo p nach Cantor-Zassenhaus) *Für ein Polynom $f \in \mathbb{F}_p[X]$ liefert der Algorithmus die Faktorisierung in irreduzible Polynome gemäß Algorithmus 3.1. Das Verfahren zur quadratfreien Faktorisierung kann dabei wahlweise, je nach den Voraussetzungen über f durchlaufen werden oder nicht.*

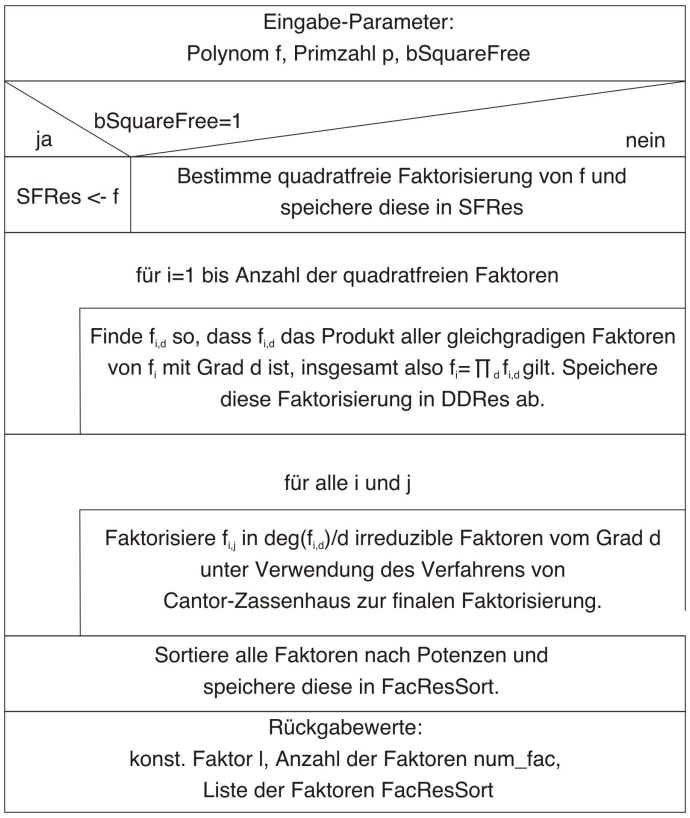

Diagramm 13: Faktorisierung modulo p nach Cantor-Zassenhaus
Sourcen-Code siehe factor_czmodp.par*, Anhang S.138*

3.2.3 Algorithmus von Berlekamp

Die Faktorisierung nach Berlekamp findet beim *LLL-Faktorisierungs-Algorithmus* Verwendung und soll zur Vorbereitung dessen zunächst an ihrer Anwendung orientiert dargestellt werden. Dabei dienen im Wesentlichen die Ausarbeitungen von Knuth[KNU] und Cohen[COH] als Vorlagen. Die Beweisführung ist hier den praktischen Gegebenheiten der Problemstellung angepaßt. Für eine ausführlichere, mehr theoretisch ausgelegte Herleitung sei hier auf die Diplomarbeit von Wenke Sietas [SIE], welche sich ausschließlich mit der *Faktorisierung über endlichen Körpern* befaßt, verwiesen.

Der hier beschriebene Algorithmus behandelt nur quadratfreie Polynome,

was aufgrund von Abschnitt 3.1.1 allerdings keine Einschränkung darstellt. Im Jahre 1967 entdeckte E. R. Berlekamp einen instruktiven Algorithmus für die Faktorisierung von Polynomen modulo p. Sei im Nachfolgenden f ein quadratfreies, normiertes Polynom mit $f \in \mathbb{F}_p[X]$. Gesucht ist die Faktorisierung von f in seine irreduziblen Faktoren f_i mit $i \in \{1, \ldots, r\}$, so dass $f = f_1 \cdots f_r$ gilt.
Die Hauptidee Berlekamps war es, den *Chinesischen Restsatz* (s. Satz 3.1,S. 28) für seine Belange einzusetzen. Dabei kommt der Satz in folgender Form zum Einsatz:
Sei $s = (s_1, \ldots, s_r)$ ein r-Tupel mit $s_i \in \mathbb{F}_p$, dann liefert der Chinesische Restsatz die Existenz eines eindeutig bestimmten Polynoms v, so dass

$$v \equiv s \mod f_1, \quad \ldots, \quad v \equiv s \mod f_r, \tag{3.3}$$

$$\deg(v) < \deg(f_1) + \cdots + \deg(f_r) = \deg(f) \tag{3.4}$$

gilt. Mit v ist damit auf einfache Weise eine Möglichkeit gegeben Faktoren von f zu bestimmen. Denn falls $r \geq 2$ ist, folgt mit 3.4 und $s_1 \neq s_2$

$$f_1 \mid \mathrm{ggT}(f, v - s_1) \quad \text{und} \quad f_2 \nmid \mathrm{ggT}(f, v - s_1).$$

Vorbereitend bleibt noch festzuhalten, dass wegen Satz 3.3 das Polynom v die Bedingung $v^p \equiv s_j^p \mod f_j = s_j \mod f_j \equiv v$ für $1 \leq j \leq r$ erfüllt und damit

$$v^p \equiv v \mod f \quad \text{mit} \quad \deg(v) < \deg(f) \tag{3.5}$$

gilt.
Für den weiteren Verlauf verwendet man die folgende allgemeine Tatsache für endliche Körper.

Satz 3.9 *Über dem Körper $\mathbb{F}_p$ gilt*

$$X^p - X = \prod_{s \in \mathbb{F}_p} (X - s).$$

Beweis: Sei $s \in \mathbb{F}_p$ beliebig. Nach dem kleinen Satz von Fermat gilt $s^p = s$, und d. h. s ist Nullstelle von $X^p - X$ oder anders ausgedrückt $(X - s) \mid (X^p - X)$. Da s beliebig war, gilt dies für alle s und somit $\prod_{s \in \mathbb{F}_p}(X - s) \mid (X^p - X)$. Da $\deg(\prod_{s \in \mathbb{F}_p}(X - s)) = p = \deg(X^p - X)$ und beides normierte Polynome sind, folgt die Gleichheit.

□

Direkt folgert man also:

Korollar 3.1 *Sei $g \in \mathbb{F}_p[X]$ beliebig, dann gilt*

$$g^p - g = \prod_{s \in \mathbb{F}_p} (g - s).$$

Mit dem Korollar gilt also für das oben betrachtete v die Gleichung

$$v^p - v = (v-0)(v-1)\cdots(v-(p-1)) \tag{3.6}$$

und aus der Gleichung 3.5 folgt

$$f \mid v^p - v. \tag{3.7}$$

D. h. jeder der irreduziblen Faktoren von f muss einen der p Faktoren der rechten Seite von Gleichung 3.6, die relativ prim sind, teilen. Für die irreduziblen Faktoren f_j mit $1 \leq j \leq r$ muss also

$$f_j \mid v^p - v = \prod_{s \in \mathbb{F}_p} (v - s)$$

gelten. Wegen $\mathrm{ggT}(v-s, v-t) = 1$ für $s \neq t$ folgt, dass bei gegebenem $j \in \{1, \ldots, r\}$ f_j für genau ein $s_j \in \mathbb{F}_p \quad v - s_j$ teilt.
Das bisher Gesagte bedeutet, dass alle Lösungen von Gleichung 3.5 die Bedingungen 3.3 und 3.4 erfüllen müssen.

Insgesamt kann bisher festgehalten werden

$$f_j \mid \mathrm{ggT}(f, v - s_j)$$

und damit

$$f \mid \prod_{j=1}^{r} (f, v - s_j) \mid \prod_{s \in \mathbb{F}_p} \mathrm{ggT}(v - s, f).$$

Offensichtlich gilt auch $\mathrm{ggT}(f, v - s) \mid f$ und zusammen mit $\mathrm{ggT}(v-s, v-t) = 1$ für $t \neq s$ folgt

$$f = \prod_{s \in \mathbb{F}_p} \mathrm{ggT}(v - s, f). \tag{3.8}$$

Jeder Faktor der rechten Seite von 3.8 hat aufgrund von 3.5 einen Grad $< \deg(f)$. D. h. es existieren mindestens zwei nicht-triviale Faktoren von f auf der rechten Seite. Damit ist durch 3.8 eine nicht-triviale Faktorisierung gegeben. Damit reduziert sich das Faktorisierungsproblem für das Polynom f auf die Bestimmung der Polynome v und einiger ggT-Berechnungen. Die Berechnung der v erfolgt dabei über ein System von linearen Gleichungen.

Sei $v = v_0 + v_1 X + \ldots + v_{n-1} X^{n-1}$, wobei die Koeffizienten v_i mit $i \in \{0, \ldots, n-1\}$ zu bestimmen sind. v ist aufgrund von Gleichung 3.5 bestimmt (s. o.). Nach Satz 3.3 hat man

$$v^p = v_0 + v_1 X^p + \ldots + v_{n-1} X^{(n-1)p}, \tag{3.9}$$

und euklidische Division von X^{ip} und f liefert $q_i, r_i \in \mathbb{F}_p[X]$ mit

$$X^{ip} = fq_i + r_i \quad \text{für} \quad 0 \leq i \leq n-1. \tag{3.10}$$

Ersetzt man anschließend in Gleichung 3.9 X^{ip} durch den jeweiligen Ausdruck der n euklidischen Divisionen, so erhält man die Darstellung

$$v^p = v_0 r_0 + \ldots + v_{n-1} r_{n-1} + \text{ Vielfache von } f$$

wobei $r_i = r_{i,0} + r_{i,1}X + \ldots + r_{i,n-1}X^{n-1}$ ist. Daraus folgt, dass f genau dann $v^p - v$ teilt, wenn f das Polynom

$$\begin{aligned} h &= v_0(r_0 - 1) + v_1(r_1 - x) + \ldots + v_{n-1}(r_{n-1} - x^{n-1}) \\ &= v_0 r_0 + \ldots + v_{n-1} r_{n-1} - (v_0 + v_1 X + \ldots + v_{n-1} X^{n-1}) \end{aligned}$$

teilt. Da $\deg(h) \geq n-1$ und $\deg(f) = n$, trifft dies genau dann zu, wenn h das Nullpolynom ist. Mit $h = 0$ und nach dem Sortieren der Koeffizienten nach $1, X, \ldots, X^{n-1}$ erhält man ein System von n linearen Gleichungen in den Unbekannten $v_0, \ldots, v_n$. Diese Unbekannten sind gemäß obiger Definition die Koeffizienten des gesuchten Polynoms v für das $f \mid v^p - v$ gilt.
Man setze nun

$$Q = \begin{bmatrix} r_{0,0} & r_{0,1} & \cdots r_{0,n-1} \\ r_{1,0} & r_{1,1} & \cdots r_{1,n-1} \\ & \vdots & \\ r_{n-1,0} & r_{n-1,1} & \cdots r_{n-1,n-1} \end{bmatrix},$$

dann gilt der nachfolgende Satz.

Satz 3.10 *$v \in \mathbb{F}_p[X]$ mit $v = v_0 + v_1 X + \ldots + v_{n-1}X^{n-1}$ ist Lösung von $v^p \equiv v \mod f$ genau dann, wenn*

$$(v_0, v_1, \ldots, v_{n-1})Q = (v_0, v_1, \ldots, v_{n-1}),$$

oder äquivalent

$$(v_0, v_1, \ldots, v_{n-1})(Q - I) = (0, \ldots, 0). \tag{3.11}$$

Für den Beweis des obigen Satzes wird zunächst noch folgendes Lemma benötigt.

Lemma 3.4 *Für jedes Polynom $g \in \mathbb{F}_p[X]$ mit $g = g_0 + g_1 X + \ldots + g_{n-1}X^{n-1}$ gilt modulo f*

$$gQ = g^p,$$

wobei g hier auch mit dem Vektor der Koeffizienten identifiziert wird.

Beweis: Mit der obigen Definition für Q gilt:

$$\begin{aligned}
(g(X))^P \mod f &= g(X^p) \mod f \quad \text{wg. Satz 3.3} \\
&= \sum_{i=0}^{n-1} g_i X^{ip} \mod f \\
&= \sum_{i=0}^{n-1} g_i \sum_{k=0}^{n-1} r_{i,k} X^k \mod f \quad \text{wg. Gleichung 3.10} \\
&= \sum_{k=0}^{n-1} (\sum_{i=0}^{n-1} g_i r_{i,k}) X^k \mod f \\
&= gQ.
\end{aligned}$$

□

Beweis zu Satz 3.10: Der Beweis des vorhergehenden Satzes folgt nun mit der Tatsache, dass die Bedingung 3.11 genau dann erfüllt ist,wenn

$$v(X) = \sum_{j=0}^{n-1} v_j X^j = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} v_k r_{k,j} X^j \equiv \sum_{k=0}^{n-1} v_k X^{p^k} = v(X^p) \equiv (v(X))^p \bmod f$$

gilt.

□

Setzt man nun $N = Q - I$, so kann mittels des Algorithmus B.1, S.99 eine Basis des Nullraums bestimmt werden. Somit ist die Bestimmung der Polynome v auf die Bestimmung des Nullraums von $Q - I$ zurückgeführt. Die letzten beiden Sätze liefern die noch fehlenden Grundlagen, um aus dem bisher Gesagten einen Algorithmus zur Faktorisierung erstellen zu können. Mittels Gleichung 3.8 kann man Faktoren bestimmen; der nächste Satz gibt Auskunft darüber, wann die Faktorisierung vollständig ist.

Satz 3.11 *Falls r, wie bisher, die Zahl der irreduziblen Faktoren f_j von f ist, dann gilt*

$$r = \textit{Dimension des Nullraums von } Q - I.$$

Beweis: Wie bereits gesehen, gilt $f \mid \prod_{s \in \mathbb{F}_p} \mathrm{ggT}(f, v - s)$ genau dann, wenn für jedes f_i ein $s_i \in \mathbb{F}_p$ existiert, so dass $f_i \mid (v - s_i)$. Seien $s_1, \ldots, s_r$ gegeben. Nach dem Chinesischen Restsatz existiert ein eindeutig bestimmtes Polynom $v \mod f$ mit

$$v \equiv s_i \mod f \quad \text{für} \quad 1 \leq i \leq r.$$

Es gibt p^r mögliche Kombinationen der s_i; nach einer der vorigen Bemerkungen gibt es genau p^r Lösungen von $v^p - v \equiv 0 \mod f$. Mit Satz 3.10 gilt zudem, dass v genau dann eine Lösung dieser Kongruenz ist, wenn

$$(v_0, \ldots, v_n)(Q - I) = (0, \ldots, 0)$$

gilt. Damit hat das System p^r Lösungen. Also muss r die Dimension des Nullraums von $Q - I$ sein.

□

Satz 3.12 *Sei $(b_1, \ldots, b_r)$ Basis des Nullraums von $Q - I$. Dann gilt: Für alle $1 \leq j < j' \leq r$ existiert ein $k \in 1, \ldots r$ und ein $s \in \mathbb{F}_p$ so, dass*

$$f_j \mid \mathrm{ggT}(f, b_k - s)$$

und

$$f_{j'} \nmid \mathrm{ggT}(f, b_k - s)$$

gilt. (Der Vektor b_i wird hier mit dem entsprechenden Polynom identifiziert.)

Beweis: Da r die Dimension des Nullraums von $Q - I$ ist, existiert ein Vektor v mit $v(Q - I) = 0$ und $v_j \neq v_{j'}$. Also existiert ein k mit $1 \leq k \leq r$, so dass

$$b_k \mod f_j \neq b_k \mod f_{j'} \tag{3.12}$$

gilt. Denn wäre dies nicht der Fall, d. h. also, würde für alle k die Gleichheit gelten, dann gäbe es zu jeder Lösung b von Gleichung 3.5, die ja Linearkombination der b_i ist, ein $s \in \mathbb{F}_p$ mit $b \equiv s \mod f_j$ und $b \equiv s \mod f_{j'}$. Dies steht im Widerspruch dazu, dass es eine Lösung mit $b \equiv 0 \mod f_j$ und $b \equiv 1 \mod f_{j'}$ gibt. Also gilt 3.12. Setzt man nun weiter $s := b \mod f_j \in \mathbb{F}_p$, dann gilt $f_j \mid b - s$ und $f_{j'} \nmid (b - s)$.

□

Damit ist der Aufbau des Algorithmus vorgezeichnet.

Algorithmus 3.7 (Berlekamp) *Für ein quadratfreies, normiertes Polynom $u \in \mathbb{F}_p[X]$ liefert der Algorithmus die Faktorisierung in irreduzible Polynome.*

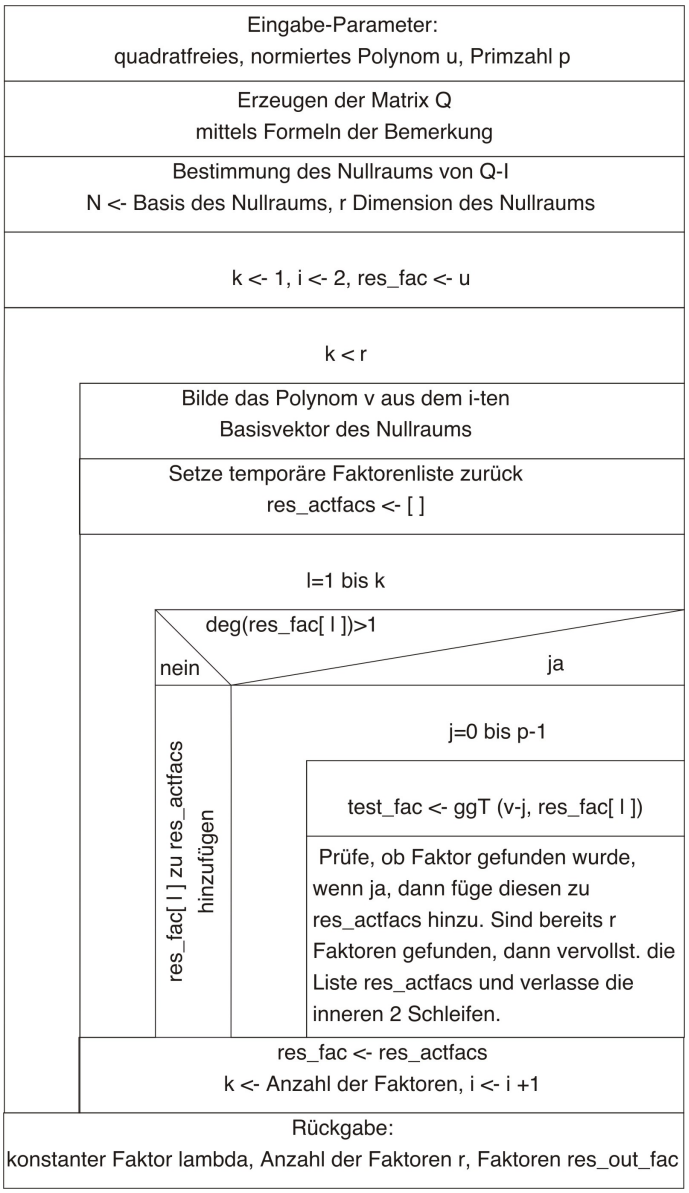

Diagramm 14: Faktorisierung nach Berlekamp
Sourcen-Code siehe berlekamp.par*, Anhang S.134*

Funktionsweise des Schleifenkonstrukts: Die äußere Schleife über k terminiert, wenn alle Faktoren gefunden sind. *res_fac* enthält am Ende die vollständige Faktorisierung von u in r irreduzible Faktoren. Dabei wird *res_fac* vor der Hauptschleife (über k) zunächst mit u initialisiert. Für jeden

Faktor mit Grad> 1 (Schleife über l) wird nun der ggT$(v[i] - s, res_fac[l])$ für alle $s \in \mathbb{F}_p$ (Schleife über j) bestimmt. Falls ein Faktor gefunden wurde, wird dieser zur Liste *res_fac* hinzugefügt und geprüft, ob bereits r Faktoren gefunden wurden. Falls dies der Fall ist, kann der Algorithmus beendet werden. Satz 3.12 garantiert dabei, dass dieses Verfahren zum Erfolg führt.

Bemerkung: Bestimmung der Matrix Q
Zur Bestimmung der Matrix Q existieren mehrere Verfahren, ein für nicht zu große p praktikables Verfahren sei hier genannt. Für nähere Betrachtung des Sachverhalts sei auf [KNU],[AKR],[COH] verwiesen.
In jedem Fall ist die Matrix $Q = (r_{i,j})_{i,j}$ durch die n Gleichungen

$$X^{ip} = fq_i + r_i \quad \text{mit} \quad i = 0, \ldots, n-1$$

bestimmt. Die erste Zeile ist immer durch $(1, 0, \ldots, 0)$ gegeben, da sie das Polynom $x^0 \mod f$ repräsentiert. Unter der Annahme, dass für ein k die Kongruenz

$$X^k \equiv r_{k,n-1}X^{n-1} + \ldots + r_{k,1}X + r_{k,0} \mod f$$

mit $f = X^n + c_{n-1}X^{n-1} + \ldots + c_1X + c_0$ gilt, dann folgt

$$\begin{aligned} X^{k+1} &\equiv r_{k,n-1}X^n + \ldots + r_{k,1}X^2 + r_{k,0}X \mod f \\ &\equiv r_{k,n-1}(-c_{n-1}X^{n-1} - \ldots - c_1X - c_0) \\ &\quad + r_{k,n-2}X^{n-1} + \ldots + r_{k,1}X^2 + r_{k,0}X \mod f \\ &\equiv r_{k+1,n-1}X^{n-1} + \ldots + r_{k+1,1}X + r_{k+1,0} \mod f, \end{aligned}$$

wobei

$$r_{k+1,j} = r_{k,j-1} - r_{k,n-1}c_j, \quad r_{k,-1} = 0.$$

Man erhält also eine einfache rekursive Formel zur Bestimmung der $r_{i,j}$.

Bemerkung: Laufzeit des Algorithmus
Bei [KNU] wird gezeigt, dass der Algorithmus für die Faktorisierung eines beliebigen, quadratfreien und normierten Polynoms f mit $n = \deg(f)$ $\mathcal{O}(n^3 + prn^2)$ Elementaroperationen benötigt

3.3 Linearer und Quadratischer Hensel Lift

Die beiden im Folgenden aufgeführten Sätze eröffnen die Möglichkeit gegebene Polynomkongruenzen über $\mathbb{Z}/p^k\mathbb{Z}$ in Kongruenzen modulo p^{k+s} mit $s \geq 1$ zu überführen. Das sog. *Hensel Lemma* geht auf eine Veröffentlichung von K. Hensel im Jahre 1908 zurück.

Satz 3.13 (Henselsches Lemma) *Sei* $f \in \mathbb{Z}[X]$ *mit* $\deg(f) \geq 1$ *und* p *Primzahl. Gelte weiter* $\mathrm{lc}(f) \neq 0 \mod p$. *Falls für zwei Polynome* $g_1, h_1 \in \mathbb{Z}[X]$

$$f \equiv g_1 h_1 \mod p \quad \textit{und} \quad \mathrm{ggT}(g_1, h_1) = 1 \mod p$$

gilt, dann existieren für alle $k \in \mathbb{N}$ *Polynome* $g_k, h_k \in \mathbb{Z}[X]$ *so, dass*

$$\begin{gathered} f \equiv g_k h_k \mod p^k, \quad \mathrm{ggT}(g_k, h_k) \equiv 1 \mod p, \\ \deg(h_k) = \deg(h_1) \quad \textit{und} \quad \mathrm{lc}(h_k) = \mathrm{lc}(h_1), \\ g_k \equiv g_1 \mod p \quad \textit{und} \quad h_k \equiv h_1 \mod p. \end{gathered}$$

Außerdem sind die Polynome g_k, h_k *eindeutig bestimmt modulo* p^k.

Um den Beweis für o. g. Satz durchführen zu können, benötigen wir einen Algorithmus zur Lösung einer Polynomgleichung über einem beliebigen endlichen Körper $\mathbb{F}_q$ mit $q = p^l$ $(l \geq 1)$.

Algorithmus 3.8 (Lösung einer Polynomgleichung in $\mathbb{F}_q[X]$) *Seien* $a, b, g, h \in \mathbb{F}_q[X]$ *und es gelte*

$$ag + bh = 1 \quad \textit{über} \quad \mathbb{F}_q,$$

dann bestimmt der gegebene Algorithmus Polynome $a', b' \in \mathbb{F}_q[X]$ *für die*

$$a'g + b'h = c \quad \textit{über} \quad \mathbb{F}_q \quad \textit{mit} \quad \deg(a') < \deg(h)$$

erfüllt ist. Der Ablauf gliedert sich in zwei Schritte.

i.) Mittels euklidischer Division *bestimmt man* $q, r \in \mathbb{F}_q[X]$ *so, dass* $ac = hq + r$ *mit* $\deg(r) < \deg(h)$ *gilt.*

ii.) Setze nun $a' = r$ *und* $b' = bc + gq$.

Beweis: Über $\mathbb{F}_q$ gilt $ag + bh = 1$. Daraus folgt durch Multiplikation mit c und unter Beachtung von i.): $c = acg + bch = rg + (bc + qg)h$. Die Bedingungen an a', b' sind aufgrund der euklidischen Division aus Schritt i.) trivialerweise erfüllt.

□

Beweis des Hensel Lemmas:
Mittels des *erweiterten Euklidischen Algorithmus* bestimmt man $a, b \in \mathbb{Z}[X]$ so, dass $ag_1 + bh_1 \equiv 1 \mod p$ gilt. Offensichtlich reicht es für den Beweis, eine Konstruktionsvorschrift für die im Satz genannten $g_j, h_j \in \mathbb{Z}[X]$ mit $j = 2, \ldots, k$ anzugeben. Seien die Polynome g_j, h_j mit den geforderten Eigenschaften für $j \leq k-1$ bereits konstruiert. Dann gilt insbesondere

$$f \equiv g_{k-1}h_{k-1} \mod p^{k-1},$$

es existiert also ein $c \in \mathbb{Z}[X]$ mit

$$f - g_{k-1}h_{k-1} = p^{k-1}c.$$

Unter Verwendung des vorhergehenden Algorithmus 3.8 berechne man $a', b' \in \mathbb{Z}[X]$ für die

$$a'g_1 + b'h_1 \equiv c \mod p \text{ gilt.}$$

Setze

$$\begin{aligned} g_k &= g_{k-1} + p^{k-1}b' \\ h_k &= h_{k-1} + p^{k-1}a'. \end{aligned}$$

Aufgrund der Konstruktionsvorschrift und Algorithmus 3.8 gilt

$$\deg(h_k) = \deg(h_{k-1}) \quad \text{und} \quad \mathrm{lc}(h_k) = \mathrm{lc}(h_{k-1})$$

und zudem

$$\begin{aligned} g_k &\equiv g_{k-1} \mod p^{k-1} \\ h_k &\equiv h_{k-1} \mod p^{k-1}. \end{aligned}$$

Offensichtlich sind g_k, h_k auch Polynome in $(\mathbb{Z}/p^k\mathbb{Z})[X]$ und über $\mathbb{Z}/p^k\mathbb{Z}$ ist

$$\begin{aligned} g_k h_k &= (g_{k-1} + p^{k-1}b')(h_{k-1} + p^{k-1}a') \\ &\equiv g_{k-1}h_{k-1} + p^{k-1}(g_{k-1}a' + h_{k-1}b') \\ &\quad \text{da } p^{2(k-1)}a'b' \equiv 0 \mod p^k \text{ wg. } 2k \geq k+1 \\ &\equiv g_{k-1}h_{k-1} + p^{k-1}c \\ &= f. \end{aligned}$$

□

Bemerkung: Der obige Beweis liefert uns damit direkt ein Verfahren, um eine Faktorisierung schrittweise von $\mathbb{F}_p$ nach $\mathbb{Z}/p^2\mathbb{Z}$, von $\mathbb{Z}/p^2\mathbb{Z}$ nach $\mathbb{Z}/p^3\mathbb{Z}$, ..., von $\mathbb{Z}/p^{k-1}\mathbb{Z}$ nach $\mathbb{Z}/p^k\mathbb{Z}$ zu liften.

Algorithmus 3.9 (Linearer Lift) *Für drei gegebene Polynome $f, g1, h1$, die die Eigenschaften des Henselschen Lemmas 3.13 für p erfüllen, liefert der Algorithmus die im Lemma genannten Polynome g_k, h_k bei gegebenem $k \in \mathbb{N}$.*

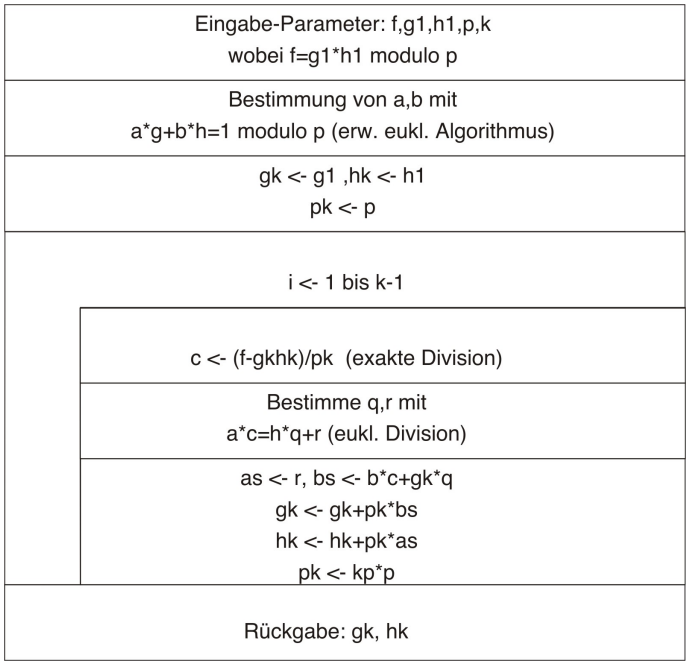

Diagramm 15: linearer Lift
Sourcen-Code siehe hensel.par*, Anhang S.127*

Beispiel:

Man betrachte das Polynom $f \in \mathbb{Z}[X]$ mit

$$f = 112x^4 + 58x^3 - 31x^2 + 107x - 66.$$

Für f gilt $f \equiv g_1 h_1 \mod 13$ mit

$$g_1 = 8x^2 + 12x + 10 \quad \text{und} \quad h_1 = x^2 + 9x + 9.$$

Diese Faktorisierung $\mod 13$ wollen wir nun zu einer Faktorisierung $\mod 13^2$ liften. Mit $a = x + 11$, $b = 5x + 11$ und $p = 13$ gilt

$$ag_1 + bh_1 \equiv 1 \mod p.$$

Gemäß dem in der Beweisführung beschriebenen Verfahren erhält man c durch

$$\begin{aligned} c &= (f - g_1 h_1)/p \\ &= ((112x^4 + 58x^3 - 31x^2 + 107x - 66) - \\ &\quad (8x^4 + 84x^3 + 190x^2 + 198x + 90))/13 \\ &= 8x^4 - 2x^3 - 17x^2 - 7x - 12 \\ &\equiv 8x^4 + 11x^3 + 9x^2 + 6x + 1 \mod 13. \end{aligned}$$

$a', b' \in \mathbb{Z}[X]$ mit $a' = x + 9$ und $b' = 8x^2 + 9x + 6$ lösen dann

$$\begin{aligned} a'g_1 + b'h_1 &= 8x^4 + 11x^3 + 9x^2 + 6x + 1 \\ &\equiv c \mod 13. \end{aligned}$$

Also erhalten wir

$$\begin{aligned} g_2 &= g_1 + pb' \\ &= 8x^2 + 12x + 10 + 13(8x^2 + 9x + 6) \\ &= 112x^2 + 129x + 88 \end{aligned}$$

und

$$\begin{aligned} h_2 &= h_1 + pa' \\ &= x^2 + 9x + 9 + 13(x + 9) \\ &= x^2 + 22x + 126. \end{aligned}$$

□

In der realen Anwendung kommt es natürlich häufig vor, dass man ein Produkt aus mehreren Faktoren liften möchte. Die nahe liegende Weise, dies zu tun, ist natürlich die wiederholte Anwendung des vorhergehenden Algorithmus auf Paare von Faktoren.

Sei $p \equiv p_1 p_2 \cdots p_r \mod p$. Im ersten Schritt wird nun $g_1 = p_1$ und $h_1 = p_2 \cdots p_r$ nach $\mathbb{Z}/p^k\mathbb{Z}$ mittels Algorithmus 3.9 geliftet. Zur Vorbereitung des nächsten Schrittes setzt man $p = \frac{p}{g_1} \mod p^k$. Im zweiten Schritt betrachtet man dann folgerichtig $g_1 = p_2$ und $h_1 = p_3 \cdots p_r$. Fortlaufende Anwendung des bisherigen Vorgehens führt nach $r - 1$ Schritten damit zum gewünschten Ergebnis.

Algorithmus 3.10 (Linearer Lift bei multiplen Faktoren) *Mit f wie im Satz 3.13, falls für $r \geq 2$ Polynome $p_1, \ldots, p_r \in \mathbb{Z}[X]$ gilt,*

$$p \equiv p_1 p_2 \cdots p_r \mod p$$

und falls diese zusätzlich alle relativ prim sind, so liefert der Algorithmus einen Lift der Faktorisierung modulo p^k mit $k \geq 1$ mit zum Satz 3.13 analogen Eigenschaften.

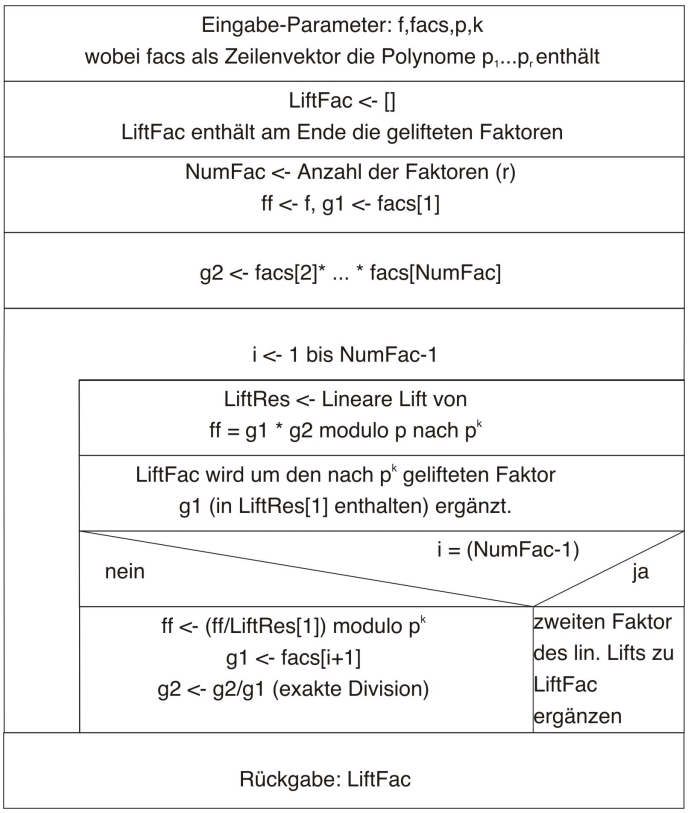

Diagramm 16: Linearer Lift bei multiplen Faktoren
Sourcen-Code siehe hensel.par*, Anhang S.127*

Der letzte Teil dieses Abschnittes befaßt sich mit dem *quadratischen Lift* nach *Zassenhaus*, dieser erweitert eine Faktorisierung $\mod p^k$ zu einer Faktorisierung $\mod p^{2k}$. Der Vorteil, den eine Kombination des linearen und des quadratischen Lifts bietet, ist augenscheinlich; sie ermöglicht das schnelle Vergrößern des Exponenten k.

Satz 3.14 (Quadratischer Lift) *Sei $f \in \mathbb{Z}[X]$ mit $\deg(f) \geq 1$, $j \in \mathbb{Z}$ mit $j \geq 1$ und p Primzahl. Gelte weiter $\mathrm{lc}(f) \neq 0 \mod p$. Falls Polynome $a_j, b_j, g_j, h_j \in \mathbb{Z}[X]$ existieren mit*

$$f \equiv g_j h_j \mod p^j \quad \textit{und} \quad a_j g_j + b_j h_j \equiv 1 \mod p^j,$$

dann existieren $a_{2j}, b_{2j}, g_{2j}, h_{2j} \in \mathbb{Z}[X]$ so, dass

$$\begin{gathered} f \equiv g_{2j} h_{2j} \mod p^{2j}, \\ g_{2j} \equiv g_j \mod p^j \quad \textit{und} \quad h_{2j} \equiv h_j \mod p^j, \\ a_{2j} g_{2j} + b_{2j} h_{2j} \equiv 1 \mod p^{2j}. \end{gathered}$$

Beweis: Sei

$$c_j = \frac{1}{p^j}(f - g_j h_j)$$

und sei $d \in \mathbb{Z}[X]$ so, dass

$$a_j g_j + b_j h_j = 1 + p^j d$$

erfüllt ist. Setzt man nun $a'_j = c_j a_j$, $b'_j = c_j b_j$ und $e = c_j d$, dann schreibt sich die zweite Formel als

$$a'_j g_j + b'_j h_j = c_j + p^j e.$$

Setze nun

$$\begin{aligned} g_{2j} &= g_j + p^j b'_j, \\ h_{2j} &= h_j + p^j a'_j. \end{aligned}$$

Dann gilt aufgrund der Definition bereits

$$g_{2j} \equiv g_j \mod p^j \quad \text{sowie} \quad h_{2j} \equiv h_j \mod p^j$$

und

$$f \equiv g_{2j} h_{2j} \mod p^{2j}$$

folgt aus

$$\begin{aligned} f - g_{2j} h_{2j} &= f - g_j h_j - p^j (b'_j h_j + a'_j g_j) - p^{2j} a'_j b'_j \\ &= p^j c_j - p^j (c_j - p^j e) - p^{2j} a'_j b'_j \\ &= p^{2j} (e - a'_j b'_j). \end{aligned}$$

Verbleibt die letzte Eigenschaft: Man betrachte dazu

$$\begin{aligned} a_j g_{2j} + b_j h_{2j} &= a_j (g_j + p^j b'_j) + b_j (h_j + p^j a'_j) \\ &= a_j g_j + b_j h_j + p^j (a_j b'_j + b_j a'_j) \\ &= 1 + p^j d'_j \text{ mit } d'_j = d + a_j b'_j + b_j a'_j \end{aligned}$$

Setzt man nun $a''_j = d'_j a_j$, $b''_j = d'_j b_j$ und $u = d'_j d$, dann gilt

$$a''_j g_j + b''_j h_j = d'_j + p^j u.$$

Mit $a_{2j} = a_j - p^j a''_j$, $a_{2j} = b_j - p^j b''_j$ folgt nach dem Bisherigen schließlich:

$$\begin{aligned} a_{2j}g_{2j} + b_{2j}h_{2j} &= (a_j - p^j a''_j)g_{2j} + (b_j - p^j b''_j)h_{2j} \\ &= (a_j g_{2j} + b_j h_{2j}) - p^j(a''_j g_{2j} + b''_j h_{2j}) \\ &= 1 + p^j d'_j - p^j(a''_j(g_j + p^j b'_j) + b''_j(h_j + p^j a'_j)) \\ &= 1 + p^j d'_j - p^j((a''_j g_j + b''_j h_j) + p^j(a''_j b'_j + b''_j a'_j)) \\ &= 1 + p^{2j} v \text{ mit } v = -(u + a''_j b'_j + b''_j a'_j). \end{aligned}$$

□

Bemerkung: Der quadratische Lift bietet zwar noch einigen Raum zur Optimierung der weiter hinten beschriebenen Faktorisierungsalgorithmen, würde allerdings auch einen zusätzlichen Algorithmus zur Bestimmung der optimalen Anzahl an quadratischen und linearen Lifts für eine Anwendung benötigen, daher geht der Algorithmus aus Gründen der Übersicht nicht in die programmtechnische Realisierung mit ein.

4 Der Algorithmus von Cantor-Zassenhaus

Nachdem im vorherigen Kapitel die Grundlagen für einen Algorithmus zur Faktorisierung von Polynomen über $\mathbb{Z}$ geschaffen wurden, kann nun das dafür notwendige Vorgehen beschrieben werden. Der hier gewählte Weg zur Bestimmung der Faktoren sei zunächst mit einem Beispiel motiviert.

Eine Grundlage des Vorgehens bildet dabei eine Tatsache, welche bereits in der Einleitung Erwähnung findet und die Existenz eines Algorithmus zur Faktorisierung garantiert.

Satz 4.1 (Mignotte-Schranke) *Seien* $f, h \in \mathbb{Z}[X]$, $h \mid f$ *mit* $\deg(h) \leq m \leq n = \deg(f)$, *dann gilt*

$$\|h\| \leq \binom{2m}{m}^{\frac{1}{2}} \|f\|.$$

Beweis: siehe Satz A.12, S. 95 und Korollar A.2, S. 96

□

Beispiel 4.1 *Sei die Faktorisierung des Polynoms*

$$f(X) = X^6 - 6X^4 - 2X^3 - 7X^2 + 6X + 1$$

über $\mathbb{Z}$ *gesucht. [f ist quadratfrei] Falls f nicht irreduzibel ist, muss es einen Faktor vom Höchstgrad 3 geben. Dieser Faktor sei* $h = h_m X^m + \ldots + h_1 + h_0$. *Mit einer bei [COH], S. 134 im Beweis zu Satz A.12 verwendeten Abschätzung erhält man*

$$|h_i| \leq \binom{m-1}{i} \|f\| + \binom{m-1}{i-1} |f_m| < 31 \quad \textit{für alle } i.$$

Wählt man nun p als die kleinste ganze Primzahl, die größer ist als das Zweifache der obigen Schranke, und die die Bedingung erfüllt, dass f quadratfrei modulo p ist, so ergibt sich $p = 67$. *Z. B. mittels des Algorithmus von Berlekamp zur Faktorisierung modulo p erhält man modulo* 67

$$f(X) = (X-4)(X-9)(X+29)(X+13)(X^2 - 29X + 30).$$

Da der konstante Term von f 1 ist, muss dies bis auf das Vorzeichen auch für jeden Faktor von f gelten, d. h. aufgrund der Faktorisierung modulo p erkennt man, dass f keinen Faktor vom Grad 1 hat. Ein Faktor vom Grad 2 kann, wg. $4 \cdot 9 = 36, 4 \cdot 29 = -18, 4 \cdot 13 = -15, 9 \cdot 29 = -7, 9 \cdot 13 = -17, 29 \cdot 13 = -25$ *ebenfalls nicht auftreten.*

Falls also f reduzibel ist, muss f zwei Faktoren vom Grad 3 haben und einer muss modulo p von $(X^2 - 29X + 30)$ geteilt werden. Mit obigem Argument bzgl. des konstanten Terms besteht modulo 67 nur die Möglichkeit

$$(X + 29)(X^2 - 29X + 30) = X^3 - 7X - 1.$$

Durch die Wahl von p (s. Bemerkung S. 56) ist dieses das einzige Polynom seiner Kongruenzklasse, welches die obige Schranke einhält. Falls f also von diesem Faktor geteilt wird, hat man bereits eine vollständige Faktorisierung von f, falls nicht, kann man folgern, dass f irreduzibel ist.
In der Tat erhält man

$$f(X) = (X^3 - 7X - 1)(X^3 + X - 1).$$

Die Irreduzibilität der Faktoren dritten Grades ist direkte Konsequenz der Betrachtung der konstanten Terme.

Aufgrund von Satz A.12 (s. Beweis des Satzes) wächst die Schranke der Koeffizienten eines Faktor erheblich mit dem Grad von f und der Größe der Koeffizienten von f. Das zu bestimmende p kann damit problemlos so groß werden, dass eine effiziente Faktorisierung modulo p nicht mehr möglich ist. An dieser Stelle kommt das Henselsche Lemma ins Spiel, d. h. man kann ein kleines p wählen und für ein passendes $e \in \mathbb{N}$ die Faktorisierung modulo p zu einer Zerlegung modulo p^e liften.

4.1 Allgemeines Vorgehen zur Faktorisierung

Das nachfolgend beschriebene Verfahren wurde von *E. R. Berlekamp, H. Zassenhaus* und *D. G. Cantor* entwickelt.

Gegeben sei ein Polynom $f \in \mathbb{Z}[X]$ mit $\deg(f) \geq 1$, dann erhält man die Faktorisierung von f über $\mathbb{Z}$ in irreduzible Faktoren durch folgende Schritte:

i.) Man assoziiert f durch seinen primitiven Anteil f_0 (siehe Lemma 2.2, S. 12), also $f_0 = \mathrm{pp}(f)$.

ii.) O. B. d. A. kann angenommen werden, dass f_0 quadratfrei ist. Sollte dies nicht der Fall sein (d. h. $\mathrm{res}(f_0, f_0') = 0$, siehe Korollar 2.1, S. 21), so setze man $g = \frac{f_0}{\mathrm{ggT}(f_0, f_0')}$. g ist dann offensichtlich quadratfrei, und man rechnet mit g anstelle von f_0 weiter. Da jeder irreduzible Faktor von $\mathrm{ggT}(f_0, f_0')$ auch g teilt, kann nach der Faktorisierung von g durch eine geringe Anzahl von Probedivisionen die komplette Faktorisierung von $f_0 = g \cdot \mathrm{ggT}(f_0, f_0')$ bestimmt werden.

iii.) Sei f_0 also als quadratfrei angenommen. Man wähle eine Primzahl $p \geq 2$, so dass f_0 quadratfrei über $\mathbb{F}_p$ ist. Dies ist nach Korollar 2.1, S. 21 dann der Fall, wenn $\mathrm{res}(f_0 \mod p, f_0' \mod p) \neq 0$ gilt.

iv.) Mit einem der vorgestellten Algorithmen bestimmt man anschließend eine vollständige Faktorisierung von $f_0 \mod p$.

v.) Da f_0 quadratfrei über $\mathbb{F}_p$ ist, kann die erhaltene Faktorisierung gem. dem Henselschen Lemma (siehe Satz 3.13, S. 47) zu einer Faktorisierung modulo p^e für ein passendes e geliftet werden (s. a. Bemerkung). Falls h ein Faktor von f_0 in $\mathbb{Z}[X]$ ist, dessen absolute Koeffizienten kleiner als $\frac{p^e}{2}$ sind, dann stimmt dieser mit einem Produkt der gelifteten, irreduziblen Faktoren überein.

vi.) Die Bestimmung der Faktorisierung von f_0 über $\mathbb{Z}$ besteht nun aus dem Testen aller Teiler modulo p^e als Teiler in $\mathbb{Z}[X]$.

Bemerkung zur Bestimmung der Zahl e: Falls $h = h_m X^m + \ldots + h_1 + h_0$ ein Faktor von f_0 mit $n = \deg(f_0)$ über $\mathbb{Z}$ ist, p eine gegebene Primzahl und $e \in \mathbb{N}$ so bestimmt, dass die Koeffizienten von h alle im Intervall von $[-\frac{p^e-1}{2}, \frac{p^e-1}{2}]$ liegen, dann muss $p^e \geq 2|b_i|$ für $i \in \{0, \ldots, m\}$ gelten.

Aufgrund von Satz A.12, S. 95 kann man ein passendes e als kleinste ganze Zahl bestimmen, welche die folgende Ungleichung erfüllt

$$p^e > 2\binom{[\frac{m}{2}]}{[\frac{m}{4}]}\|f_0\|.$$

Denn dann gilt nach Ungleichung A.6, S. 96 aus dem Beweis zu o. g. Satz

$$|h_i| \leq \binom{m}{i}\mathrm{M}(f_0) \leq \binom{m}{i}\|f_0\| \leq \binom{[\frac{m}{2}]}{[\frac{m}{4}]}\|f_0\| < \frac{p^e}{2}$$

für alle $i \in \{0, \ldots, m\}$, wobei $\mathrm{M}(f_0)$ das im Anhang A.3, S. 94 definierte Maß ist, $[b]$ die Gauß-Klammer der reellen Zahl b meint und

$$\max_{m \leq n/2} \binom{m}{[\frac{m}{2}]} = \binom{[\frac{m}{2}]}{[\frac{m}{4}]}$$

benutzt wird.

4.2 Quadratfreie Polynome über $\mathbb{Z}$

Insgesamt erhält man gemäß 4.1, S. 55, Schritte iii.) bis vi.). den folgenden Algorithmus.

Algorithmus 4.1 (Quadratfreie Faktorisierung) *Für ein quadratfreies Polynom $f \in \mathbb{Z}[X]$ liefert der Algorithmus die Faktorisierung in irreduzible Polynome gemäß der Beschreibung aus Abschnitt 4.1.*

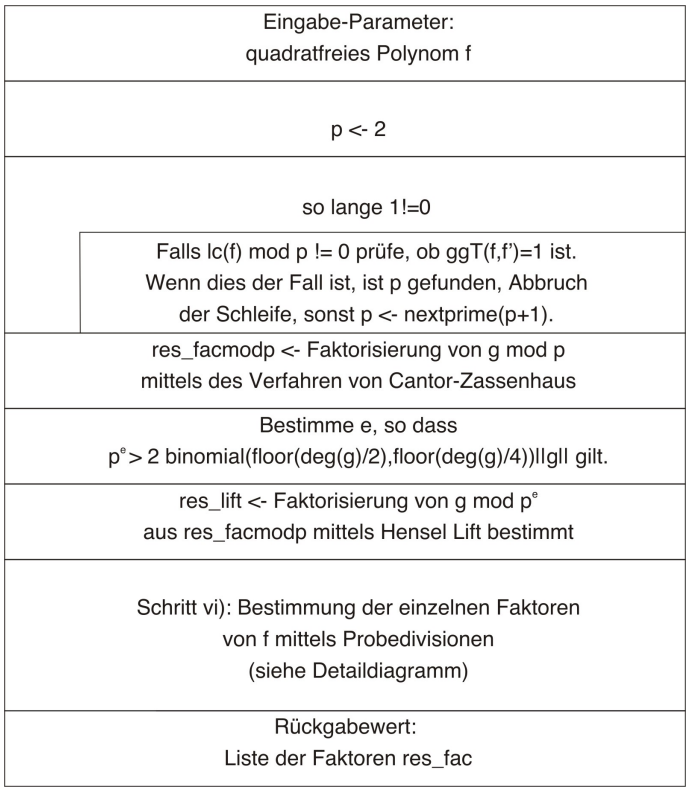

Diagramm 17: Quadratfreie Faktorisierung in $\mathbb{Z}[X]$
Sourcen-Code siehe factor_cz.par*, Anhang S.144*

Die Schritte iii.) bis v.) gehen direkt aus Abschnitt 4.1 hervor und können unmittelbar umgesetzt werden, lediglich Schritt vi.) bedarf weiterer Erläuterungen. Im Anschluss daran findet sich für diesen Schritt zusätzlich ein detailliertes Diagramm.

Nach dem Lift der Faktorisierung in Schritt v.) gilt

$$f \equiv \mathrm{lc}(f) \cdot f_1 \cdots f_r \mod p^e.$$

Unter Ausnutzung der Tatsache, dass jeder irreduzible Faktor von f über $\mathbb{Z}$ auch ein Faktor modulo p ist, welcher allerdings nicht notwendig irreduzibel

ist, folgt, dass Kombinationen von Faktoren in Probedivisionen verwendet werden müssen, um die irreduziblen Faktoren von f über $\mathbb{Z}$ zu finden.

Teilalgorithmus 4.1 (Faktorisierung – Schritt vi.) *) Schritt vi.) aus Abschnitt 4.1 wird durch folgendes Diagramm beschrieben.*

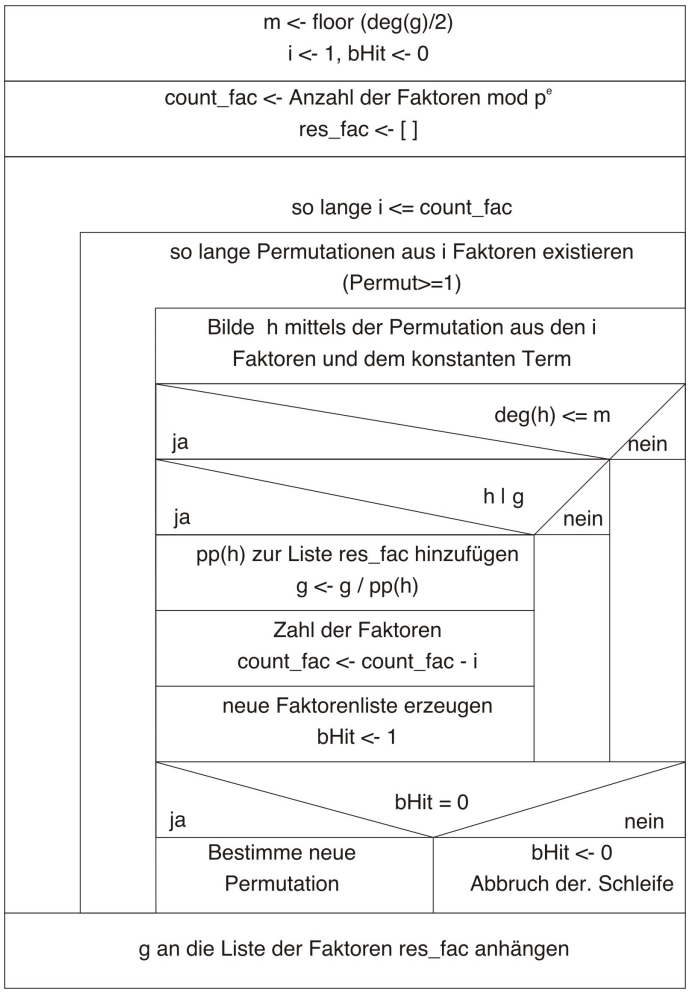

Diagramm 18: Quadratfreie Faktorisierung – Schritt vi.)
Sourcen-Code siehe factor_cz.par*, Anhang S.144*

Ausgehend von den r Faktoren $f_i \mod p^e$ bildet der Algorithmus alle Kombinationen von i Faktoren und für jedes so gebildete Polynom h wird getestet, ob $h \mid g$ gilt. Zwecks Laufzeitoptimierung wird dabei zuerst getestet, ob h die Gradbedingung für einen Faktor von g, $(\deg(h) \leq \left\lceil \frac{\deg(g)}{2} \right\rceil)$, erfüllt. Nur falls diese erfüllt und die Teilbarkeit der Höchstkoeffizienten gegeben ist, muss die Teilbarkeit von g durch h getestet werden. Ist ein irreduzibler

Faktor von f über $\mathbb{Z}$ gefunden, so werden die i Faktoren von der Liste der Faktoren entfernt, g entsprechend durch g/h ersetzt und die Betrachtung beginnt erneut. Bei diesem Vorgehen enthält g am Ende den letzten irreduziblen Faktor von f.

Bemerkung zur Laufzeit: Zur Erzeugung des jeweiligen Testpolynoms h wird die sog. *cardinality procedure* verwendet; d. h. es werden zu r Faktoren alle möglichen Kombinationen aus i Faktoren mit $i \in \{1, \ldots, [r/2]\}$ generiert. Denkbar wäre auch die sog. *degree procedure*, wobei alle Kombinationen von Faktoren gewählt werden, wo der Grad von $h \quad s$ mit $s \in \{1, \ldots, [\deg(g)/2]\}$ ist. Nach [LUT] S. 189 ist dieses jedoch aus Effizienzgründen zu vermeiden, da die mittlere Anzahl der benötigten Kombinationen im Fall der *degree procedure* von exponentieller Ordnung ist, bei Verwendung der *cardinality procedure* hingegen wird diese Anzahl von $(\deg(g))^2$ dominiert. Im schlechtesten Fall ist allerdings auch hier nur eine Laufzeit von exponentieller Ordnung zu erwarten.
Zusätzlich ist noch zu bedenken, dass zur Bestimmung der Faktorisierung modulo p ein probabilistischer Algorithmus eingesetzt wird.

4.3 Beliebige Polynome über $\mathbb{Z}$

Im Schritt von der Faktorisierung quadratfreier Polynome hin zur Faktorisierung beliebiger Polynome werden zwei wesentliche Tatsachen verwendet.

Einerseits kommt zum Tragen, dass jedes Polynom $f \in \mathbb{Z}[X]$ eine eindeutige Darstellung der Form

$$f = \mathrm{cont}(f)\mathrm{pp}(f)$$

hat, d. h. es reicht zur Faktorisierung einerseits, den primitiven Anteil zu betrachten, und andererseits, dass f nach Korollar 2.1, S. 21 genau dann quadratfrei ist, wenn $\mathrm{res}(f, f') \neq 0$ gilt. Falls f nicht quadratfrei ist, setzt man

$$g = \mathrm{ggT}(f, f') \quad \text{und} \quad f_1 = f/g.$$

Dann ist f_1 offensichtlich quadratfrei und kann mit dem Algorithmus von Cantor-Zassenhaus (Algorithmus 4.1, S. 57) faktorisiert werden. Beachtet man noch, dass jeder irreduzible Faktor von g ebenfalls Teiler von f_1 ist, so erhält man die Faktorisierung von f durch wenige Probedivisionen (siehe Hauptschleife des Algorithmus).

Bemerkung: Im Falle des *LLL*-Algorithmus wird eine nur geringfügig abgeänderte Variante zur Behandlung beliebiger Polynome verwendet; im unten stehenden Algorithmus muss lediglich zusätzlich das Ergebnis der Resultantenbestimmung an den Algorithmus zur Faktorisierung von quadratfreien Polynome weitergereicht werden.

Algorithmus 4.2 (Faktorisierung nach Cantor-Zassenhaus) *Unter Voraussetzung eines Algorithmus zur Faktorisierung quadratfreier Polynome (hier gegeben durch die Routine* `factor_czsf_calc`*), liefert dieser Algorithmus die Faktorisierung eines beliebigen Polynoms* $f \in \mathbb{Z}[X]$ *mit* $\deg(f) >= 1$ *in irreduzible Faktoren gemäß der Beschreibung aus 4.1, S. 55, Schritte i.) und ii.).*

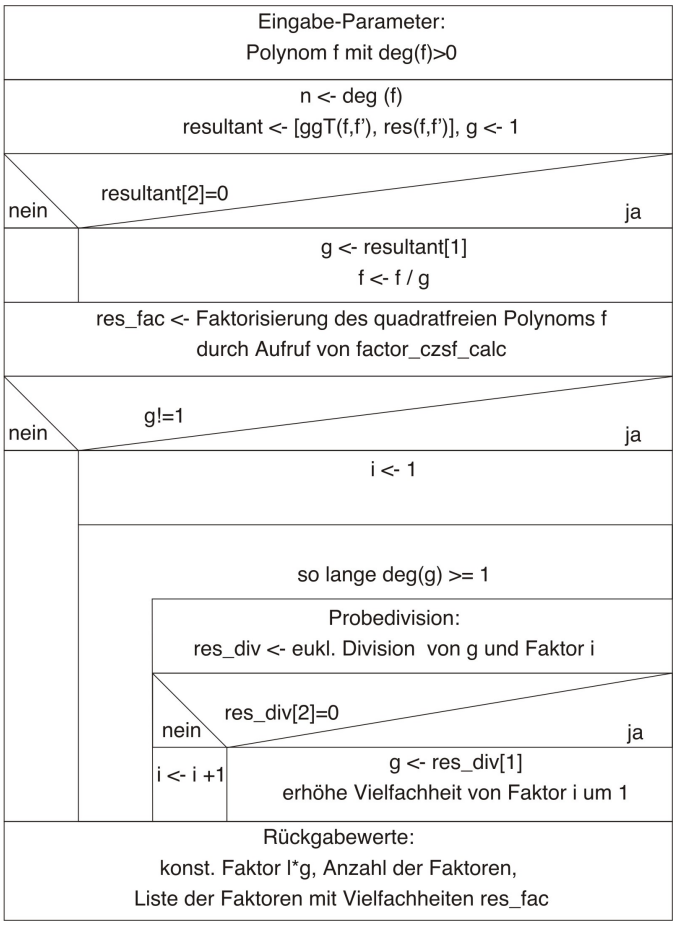

Diagramm 19: Faktorisierung in $\mathbb{Z}[X]$
Sourcen-Code siehe factor_cz.par*, Anhang S.144*

Bemerkung zur Laufzeit: Aufgrund von Algorithmus 4.1 ist die Laufzeit dieses Algorithmus von exponentieller Ordnung; im Mittel wird allerdings eine Laufzeit von zufälliger polynomieller Ordnung erzielt.

5 Der LLL-Algorithmus

Wie die Laufzeitbetrachtung zum Cantor-Zassenhaus-Algorithmus des vorherigen Kapitels gezeigt hat, kann der Algorithmus im ungünstigsten Fall eine Laufzeit von exponentieller Ordnung haben.
A. K. Lenstra, H. W. Lenstra und L. Lovász haben im Rahmen ihrer Forschung 1982 den sog. *LLL*-Algorithmus zur Faktorisierung von Polynomen über $\mathbb{Z}$ vorgestellt, dessen Zahl der nötigen Berechnungen von der Ordnung $\mathcal{O}(n^6 + n^5 log\|f\|)$ ist. Dabei sei $f \in \mathbb{Z}[X]$, man betrachte die euklidische Norm, und es gelte $n = \deg(f)$.
Der Algorithmus basiert auf dem *Algorithmus von Berlekamp* (s. Algorithmus 3.2.3, S.39) sowie dem *Henselschen Lemma* (s. Satz 3.13, S.47). Das Vorgehen kann dabei wie folgt kurz umrissen werden. Zu $f \in \mathbb{Z}[X]$ sucht man eine passende, möglichst kleine Primzahl p und berechnet einen irreduziblen Faktor g von f modulo p. Im nächsten Schritt sucht man einen irreduziblen Faktor h von f über $\mathbb{Z}$ welcher modulo p von g geteilt wird. Die Beziehung $g \mid h$ ist äquivalent dazu, dass h zu einem speziellen Gitter gehört und die Koeffizienten von h relativ klein sind.
Der Hauptansatz des Algorithmus besteht nun in der Bestimmung irreduzibler Faktoren von f durch wiederholte Anwendung des Basisreduktionsalgorithmus.
Die Beschreibung folgt im Aufbau, wie fast alle Veröffentlichungen, dem Orginaldokument [LLL], beginnend mit einem Abschnitt zu Gittern und reduzierten Basen, über einen größeren theoretischen Teil, welcher die Verbindung zwischen Gitterbasen und Polynomen schafft, hin zum Abschnitt mit der eigentlichen Beschreibung des Algorithmus.

5.1 Gitter und reduzierte Basen

Für einen Einstieg in die Materie sind zunächst einige Definitionen und Bemerkungen notwendig.

Definition 5.1 *Sei $n \in \mathbb{N}^*$ und $L \subset \mathbb{R}^n$. L heißt* Gitter*, falls eine Basis $\mathcal{B} = (b_1, \ldots, b_n)$ des reellen Vektorraums $\mathbb{R}^n$ existiert, so dass sich L darstellen läßt als*

$$L = \sum_{i=1}^{n} \mathbb{Z} b_i = \left\{ \sum_{j=1}^{n} r_j b_j \quad | \quad r_j \in \mathbb{Z}, j \in \{1, \ldots, n\} \right\}.$$

Zusätzlich sagt man auch, die Vektoren aus $\mathcal{B}$ bilden eine Basis oder erzeugen L. Die Zahl n ist der Gitterrang *und die* Determinante d(L) *ist definiert als*

$$\mathrm{d(L)} = |\det(b_1, \ldots, b_n)|.$$

Aufgrund dieser Definition ergibt sich unmittelbar eine erste Aussage bezüglich der Eigenschaften der Determinante eines Gitters.

Lemma 5.1 *Der Wert der Determinante* d(L) *eines Gitters* $L \subset \mathbb{R}^n$ *ist unabhängig von der gewählten Basis.*

Beweis: Seien $(b_1, \ldots, b_n)$ und $(b'_1, \ldots, b'_n)$ Basen von L. Aufgrund der Definition des Gitterbegriffes muss sich jedes b_i bzw. b'_i als ganzzahlige Linearkombination der $b'_1, \ldots, b'_n$ bzw. $b_1, \ldots, b_n$ darstellen lassen. Es existieren also Basiswechselmatrizen $U, V \in \mathbb{Z}^{n \times n}$ mit $B = B'U$ und $B' = BV$, falls man mit B, B' die Matrizen, die die Basisvektoren in den Spalten tragen, identifiziert. Damit ergibt sich $\det(B) = \det(B')\det(U)$ und $\det(B') = \det(B)\det(V)$. Also gilt $\det(U)\det(V) = 1$, was wegen $\det(U), \det(V) \in \mathbb{Z}$, $\mathrm{d(L)} = |\det(B)| = |\det(B')|$ zur Folge hat.

□

Ein Verfahren, welches man meist zunächst im Rahmen der Linearen Algebra kennenlernt, ist das sog. *Gram-Schmidt*-Verfahren zur Orthogonalisierung einer Basis eines Vektorraums. Da orthogonale Basen mit speziellen Eigenschaften die Grundlage des *LLL*-Algorithmus bilden, wird dieses Verfahren hier informell aufgeführt. $\Phi(x, y)$ sei das kanonische Skalarprodukt im reellen Vektorraum $\mathbb{R}^n$, also gelte für $x, y \in \mathbb{R}^n$

$$\Phi(x, y) = \sum_{i=1}^{n} x_i y_i,$$

und weiter für die euklidische Norm $\|x\|^2 = \Phi(x, x)$.

Satz 5.1 (Gram-Schmidt-Verfahren) *Sei* $\mathcal{B} = (b_1, \ldots, b_n)$ *Basis des* $\mathbb{R}^n$*. Dann liefern die unten gegebenen rekursiven Formeln eine orthogonale Basis* $\mathcal{B}' = (b_1^*, \ldots, b_n^*)$ *mit*

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \quad \textit{und} \quad \mu_{i,j} = \frac{\Phi(b_i, b_j^*)}{\Phi(b_j^*, b_j^*)}.$$

Beweis: siehe Lineare Algebra

□

Bemerkung: b_i^* ist die Projektion von b_i auf das orthogonale Komplement von $\sum_{j=1}^{i-1} \mathbb{R} b_j$.

5.1.1 Reduzierten Basen und ihre Eigenschaften

Definition 5.2 (reduzierte Basis) *Sei $\mathcal{B} = (b_1, \ldots, b_n)$ eine Basis des Gitters $L \subset \mathbb{R}^n$. $\mathcal{B}$ heißt* reduzierte Basis, *falls*

$$|\mu_{i,j}| \leq \frac{1}{2} \quad \textit{für} \quad 1 \leq j < i \leq n$$

und

$$\|b_i^* + \mu_{i,i-1} b_{i-1}^*\|^2 \geq \frac{3}{4} \|b_{i-1}^*\|^2 \quad \textit{für} \quad 1 \leq i \leq n$$

gelten.

Mittels des nachfolgenden Satzes ist es möglich, die Größe der Vektoren einer reduzierten Basis zu kontrollieren.

Satz 5.2 *Sei $\mathcal{B} = (b_1, \ldots, b_n)$ eine reduzierte Basis des Gitters L, und seien $b_1^*, \ldots, b_n^*$ die zugehörigen, mittels des Gram-Schmidt-Verfahrens erhaltenen, Vektoren. Dann gilt für alle i, j mit $1 \leq j \leq i \leq n$*

$$\|b_j\| \leq 2^{\frac{i-1}{2}} \|b_i^*\|.$$

Beweis: Da $b_1, \ldots, b_n$ als reduzierte Basis vorausgesetzt wird, erhält man aus der Bedingung

$$\frac{3}{4} \|b_{i-1}^*\|^2 \leq \|b_i^* + \mu_{i,i-1} b_{i-1}^*\|^2$$

mit $|\mu_{i,j}| \leq \frac{1}{2}$ für alle i, j mit $1 \leq j < i \leq n$, und da $\Phi(b_i^*, b_j^*) = 0$ für $i \neq j$, die Abschätzung

$$\frac{3}{4} \|b_{i-1}^*\|^2 \leq \|b_i^*\|^2 + \mu_{i,i-1}^2 \|b_{i-1}^*\|^2$$

also

$$\frac{1}{2} \|b_{i-1}^*\|^2 \leq \|b_i^*\|^2.$$

Induktiv gilt schließlich

$$\|b_j^*\|^2 \leq 2^{i-j} \|b_i^*\|^2$$

für alle i, j mit $1 \leq j \leq i \leq n$.
Beachtet man zusätzlich die Bildungsvorschrift der Basis $(b_1^*, \ldots, b_n^*)$ mit

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*,$$

so folgt

$$\begin{aligned}
\|b_i\|^2 &= \|b_i^*\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|b_j^*\|^2 \quad \text{wg. Orthogonalität} \\
&\leq \|b_i^*\|^2 + \sum_{j=1}^{i-1} \frac{1}{4} 2^{i-j} \|b_i^*\|^2 \\
&= (1 + \frac{1}{4}(2^i - 2))\|b_i^*\|^2 \\
&\leq 2^{i-1}\|b_i^*\|^2
\end{aligned}$$

und damit

$$\|b_j\|^2 \leq 2^{j-1}\|b_j^*\|^2 \leq 2^{j-1}2^{i-j}\|b_i^*\|^2 = 2^{i-1}\|b_i^*\|^2$$

für alle i, j mit $1 \leq j \leq i \leq n$.

□

Satz 5.3 (Hadmardsche Ungleichung) *Sei $\mathcal{B} = (b_1, \ldots, b_n)$ eine Basis des Gitters L. Dann gilt*

$$\mathrm{d(L)} \leq \prod_{i=1}^{n} \|b_i\|.$$

Beweis: Sei $b_1^*, \ldots, b_n^*$ die orthogonale Basis des Gitters, welche man mittels des Gram-Schmidt-Verfahrens aus $\mathcal{B}$ erhält. Dann gilt

$$\begin{aligned}
\mathrm{d(L)} &= \det(b_1^*, \ldots, b_n^*) \\
&= \|b_1^*\| \cdots \|b_n^*\| \quad \text{(wg. Orthogonalität)} \\
&\leq \prod_{i=1}^{n} \|b_i\| \quad \text{(wg. } \|b_i^*\| \leq \|b_i\|)
\end{aligned}$$

□

Als direkte Folgerung zu Satz 5.2 ergibt sich folgendes Korollar.

Korollar 5.1 *Sei $\mathcal{B} = (b_1, \ldots, b_n)$ eine reduzierte Basis des Gitters L, dann gilt*

$$\mathrm{d(L)} \leq 2^{\frac{n(n-1)}{4}} \mathrm{d(L)}, \tag{5.1}$$

sowie

$$\|b_1\| \leq 2^{\frac{n-1}{4}} \mathrm{d(L)}^{\frac{1}{n}}. \tag{5.2}$$

Bemerkung: Die Ungleichung 5.1 folgt dabei aus

$$\|b_1\|^{2n} \leq \prod_{i=1}^{n} 2^{i-1}\|b_i^*\|^2 = 2^{\frac{n(n-1)}{2}}\mathrm{d(L)}^2,$$

also

$$\|b_1\| \leq 2^{\frac{n-1}{4}}\mathrm{d(L)}^{\frac{1}{n}}.$$

Satz 5.4 *Sei $\mathcal{B} = (b_1, \ldots, b_n)$ eine reduzierte Basis des Gitters $L \subset \mathbb{R}^n$. Dann gilt*

$$\|b_1\| \leq 2^{\frac{n-1}{2}}\|x\|$$

für alle $x \in L, x \neq 0$.

Beweis: Da $x \in L$ ist, muss x, für den Fall, dass $b_1^*, \ldots, b_n^*$ wieder die zugehörige orthogonale Basis ist, zwei Darstellung der Form

$$x = \sum_{i=1}^{n} r_i b_i \quad \text{mit} \quad r_i \in \mathbb{Z}$$

und

$$x = \sum_{i=1}^{n} r_i' b_i^* \quad \text{mit} \quad r_i' \in \mathbb{Q}$$

besitzen. Sei $s = \max(\{i \mid r_i \neq 0 \text{ mit } 1 \leq i \leq n\})$, dann ist $r_s' = r_s$ aufgrund der Definition des Gram-Schmidt-Verfahrens, und es gilt

$$\|b_s^*\|^2 \leq r_s'^2\|b_s^*\|^2 \leq \|x\|^2.$$

Mit Satz 5.2 folgt

$$\|b_1\|^2 \leq 2^{s-1}\|b_s^*\|^2 \leq 2^{n-1}\|b_s^*\|^2,$$

also

$$\|b_1\|^2 \leq 2^{n-1}\|x\|^2.$$

□

Satz 5.5 *Sei $\mathcal{B} = (b_1, \ldots, b_n)$ eine reduzierte Basis des Gitters $L \subset \mathbb{R}^n$ und $x_1, \ldots, x_t \in L$ linear unabhängig. Dann gilt*

$$\|b_j\|^2 \leq 2^{n-1}\max(\|x_1\|^2, \ldots, \|x_t\|^2)$$

für alle $j \in \{1, \ldots, t\}$.

Beweis: Seien

$$x_j = \sum_{i=1}^{n} r_{i,j} b_i \quad \text{mit} \quad r_{i,j} \in \mathbb{Z}$$

und

$$s_j = \max(\{i \mid r_{i,j} \neq 0 \text{ mit } 1 \leq i \leq n\}).$$

Nach dem Beweis des vorherigen Satzes gilt dann

$$\|b_{s_j}^*\|^2 \leq \|x_j\|^2 \quad \text{für} \quad 1 \leq j \leq t.$$

O. B. d. A. kann man annehmen, das $s_1 \leq s_2 \leq \ldots \leq s_t$ gilt. Zusätzlich muss dann auch $j \leq s_j$ gelten, da sonst $x_1, \ldots, x_j \in \sum_{i=1}^{j-1} \mathbb{R} b_i$ im Widerspruch zur linearen Unabhängigkeit der x_i stünde. Damit gilt

$$\|b_j\|^2 \leq 2^{s_j - 1} \|b_{s_j}^*\|^2 \leq 2^{n-1} \|b_{s_j}^*\|^2 \leq 2^{n-1} \|x_j\|^2$$

für $1 \leq j \leq t$. Insgesamt gilt damit die Behauptung.

□

5.2 Basis-Reduktions-Algorithmus

Nachdem zahlreiche der interessanten Eigenschaften von reduzierten Basen gezeigt sind, stellt sich die Frage, wie man aus einer Gitterbasis eine reduzierte Basis erhält. Der Klärung dieses Sachverhalts dienen die nächsten drei Abschnitte.

5.2.1 Herleitung

Das Vorgehen zur Erlangung einer reduzierten Basis stellt sich wie folgt dar. Gegeben sei eine Basis $\mathcal{B} = (b_1, \ldots, b_n)$ des Gitters $L \subset \mathbb{R}^n$. Mittels des Gram-Schmidt-Verfahrens berechnet man zunächst eine orthogonale Basis $(b_1^*, \ldots, b_n^*)$. Die dabei anfallenden $\mu_{i,j}$ für $1 \leq j < i \leq n$ spielen eine entscheidende Rolle für den Algorithmus. Zur Laufzeit werden die Vektoren $b_1, \ldots, b_n$ wiederholt durch lineare Isomorphismen verändert, wobei nach jeder Änderung die b_i^* und $\mu_{i,j}$ aktualisiert werden.
Der Algorithmus geht iterativ vor; der aktuelle Schritt wird durch den Index k (s. a. die Hauptschleife im Diagramm) repräsentiert. $k = 1$ ist der Ausgangszustand, $k = 2$ ist der erste zu behandelnde Schritt.
Das Vorgehen ist dabei so angelegt, dass nach jedem Schritt die Bedingungen

$$|\mu_{i,j}| \leq \tfrac{1}{2} \quad \text{für} \quad 1 < j < i < k \tag{5.3}$$

$$\|b_i^* + \mu_{i,i-1} b_{i-1}^*\|^2 \geq \tfrac{3}{4} \|b_{i-1}^*\|^2 \quad \text{für} \quad 1 < i < k \tag{5.4}$$

erfüllt sind. Für $k = 2$ ist dies trivialerweise der Fall. Im Falle von $k = n + 1$ ist aufgrund von Definition 5.2 eine reduzierte Basis gegeben und der Algorithmus ist beendet.
Zunächst will man erreichen, dass

$$|\mu_{k,k-1}| \leq \frac{1}{2} \quad \text{für} \quad k > 1 \tag{5.5}$$

gilt. Sollte dies bei der Überprüfung nicht der Fall sein, so setze man $r = [\mu_{k,k-1}]$ ($[a]$ bezeichnet dabei die Rundung zur nächstgelegenen ganzen Zahl) und ersetzt die b_k durch $b_k - rb_{k-1}$. Offensichtlich müssen jetzt die $\mu_{k,j}$ für $j < k - 1$ angepasst werden. Wegen

$$\mu_{k,j} = \frac{\Phi(b_k, b_j^*)}{\Phi(b_j^*, b_j^*)}$$

folgt

$$\begin{aligned} \mu_{k,j} &\leftarrow \frac{\Phi(b_k, b_j^*)}{\Phi(b_j^*, b_j^*)} - r\frac{\Phi(b_{k-1}, b_j^*)}{\Phi(b_j^*, b_j^*)} \\ &= \mu_{k,j} - r\mu_{k-1,j}. \end{aligned}$$

Abschließend muss $\mu_{k,k-1}$ durch $\mu_{k,k-1} - r$ ersetzt werden. Die $b_1^*, \ldots, b_n^*$ bleiben unberührt. Die Bedingung 5.5 ist jetzt erfüllt.
Nun werden zwei Fälle unterschieden.

Fall 1: $k = 1$ oder $\|b_k^* + \mu_{k,k-1}b_{k-1}^*\|^2 \geq \frac{3}{4}\|b_{k-1}^*\|^2$
In diesem einfachen Fall muss nur noch

$$|\mu_{k,j}| \leq \frac{1}{2} \quad \text{für} \quad 1 \leq j < k - 1 \tag{5.6}$$

erreicht werden, für $j = k - 1$ ist die Bedingung bereits erfüllt. Falls 5.6 noch nicht erfüllt ist, verfahre man wie bei der Erfüllung der Ungleichung 5.5. Falls t der größte Index, für den $|\mu_{k,t}| > \frac{1}{2}$ gilt, ist, setze $r = [\mu_{k,t}]$, ersetze b_k durch $b_k - rb_t$ und passe $\mu_{k,j}$ mit $j < t$ an durch $\mu_{k,j} - r\mu_{t,j}$ und zum Schluss $\mu_{k,t} \leftarrow \mu_{k,t} - r$.
Die restlichen $\mu_{i,j}$ und alle b_i^* bleiben unverändert. Man wiederhole dies, bis die Bedingung 5.6 erfüllt ist.

Fall 2: $K \geq 2$ und $\|b_k^* + \mu_{k,k-1}b_{k-1}^*\|^2 \leq \frac{3}{4}\|b_{k-1}^*\|^2$
Um die zweite Ungleichung 5.4 zu erfüllen, wird nun gemäß der nahe liegenden Idee verfahren, die beiden Vektoren b_k und b_{k-1} zu tauschen. Natürlich müssen dann $\mu_{k,k-1}$, $\mu_{k-1,j}$, $\mu_{k,j}$, $\mu_{i,k-1}$ und $\mu_{i,k}$ für $j < k - 1$ und $i > k$ entsprechend abgeändert werden. Durch den Tausch wird b_k^* durch $b_k^* + \mu_{k,k-1}b_{k-1}^*$ ersetzt, und $\|b_{k-1}^*\|^2$ ist damit sicher um den Faktor $\frac{3}{4}$ kleiner, als der vorhergehende Betrag. Sei nun der Fall im Detail betrachtet. Sei

$(c_1, \ldots, c_n)$ die Basis die $\mathcal{B}$ ersetzen soll, und seien $\nu_{i,j}$ die Zahlen, die die $\mu_{i,j}$ ersetzen sollen. Mittels

$$c_{k-1} = b_k, \quad c_k = b_{k-1}, \quad c_i = b_i \quad \text{für} \quad i \neq k, k-1$$

realisiere man zunächst die neue Basis. Da c^*_{k-1} aufgrund des Gram-Schmidt-Verfahrens die Projektion von b_k auf das orthogonale Komplement von $\sum_{j=1}^{k-2} \mathbb{R} b_j$ ist, erhält man

$$c^*_{k-1} = b^*_k + \mu_{k,k-1} b^*_{k-1}$$

direkt aus der Formel zur Bestimmung von b^*_k. Um c^*_k zu bestimmen projiziert man b^*_{k-1} auf das orthogonale Komplement von $\mathbb{R} c^*_{k-1}$. Dann gilt

$$\nu_{k,k-1} = \frac{\Phi(b^*_{k-1}, c^*_{k-1})}{\Phi(c^*_{k-1}, c^*_{k-1})} \tag{5.7}$$

$$= \mu_{k,k-1} \frac{\|b^*_{k-1}\|^2}{\|c^*_{k-1}\|^2} \quad \text{wg. Orthogonalität} \tag{5.8}$$

und $c^*_k = b^*_{k-1} - \nu_{k,k-1} c^*_{k-1}$. Für $i \neq k-1, k$ gilt $c^*_i = b^*_i$. Bleiben noch die $\nu_{i,k-1}$ und $\nu_{i,k}$ für $i > k$ zu bestimmen. Durch Umstellen der bisherigen Gleichungen erhält man

$$b^*_{k-1} = \nu_{k,k-1} c^*_{k-1} + c^*_k$$

und

$$\begin{aligned} b^*_k &= c^*_{k-1} - \mu_{k,k-1} b^*_{k-1} \\ &= (1 - \mu_{k,k-1}\nu_{k,k-1}) c^*_{k-1} - \mu_{k,k-1} c^*_k \\ &= \frac{\|b^*_k\|^2}{\|c^*_{k-1}\|^2} c^*_{k-1} - \mu_{k,k-1} c^*_k. \end{aligned}$$

Durch Einsetzen in

$$b_i = b^*_i + \sum_{j=1}^{i-1} \mu_{i,j} b^*_j$$

folgert man schließlich

$$\begin{aligned} \nu_{i,k-1} &= \mu_{i,k-1}\nu_{k,k-1} + \frac{\|b^*_k\|^2}{\|c^*_{k-1}\|^2} \mu_{i,k} \\ \nu_{i,k} &= \mu_{i,k-1} - \mu_{i,k}\mu_{k,k-1} \end{aligned}$$

und

$$\nu_{k-1,j} = \mu_{k,j}, \quad \nu_{k,j} = \mu_{k-1,j} \quad \text{für} \quad 1 \leq j < k-1$$

sowie $\nu_{i,j} = \mu_{i,j}$ falls $1 \leq j < i \leq n$, $j, i \notin \{k-1, k\}$.

5.2.2 Darstellung des Algorithmus

Nach der Vorbetrachtung kann der Algorithmus insgesamt durch die beiden nachfolgenden Diagramme dargestellt werden.

Algorithmus 5.1 *Der Algorithmus bestimmt zu einer in den Spalten der Matrix A gegebenen Basis $\mathcal{B} = (b_1, \ldots, b_n)$ des Gitters $L \subset \mathbb{R}^n$ eine reduzierte Basis $\mathcal{B}' = (b'_1, \ldots, b'_n)$ gemäß Definition 5.2, S. 63.*

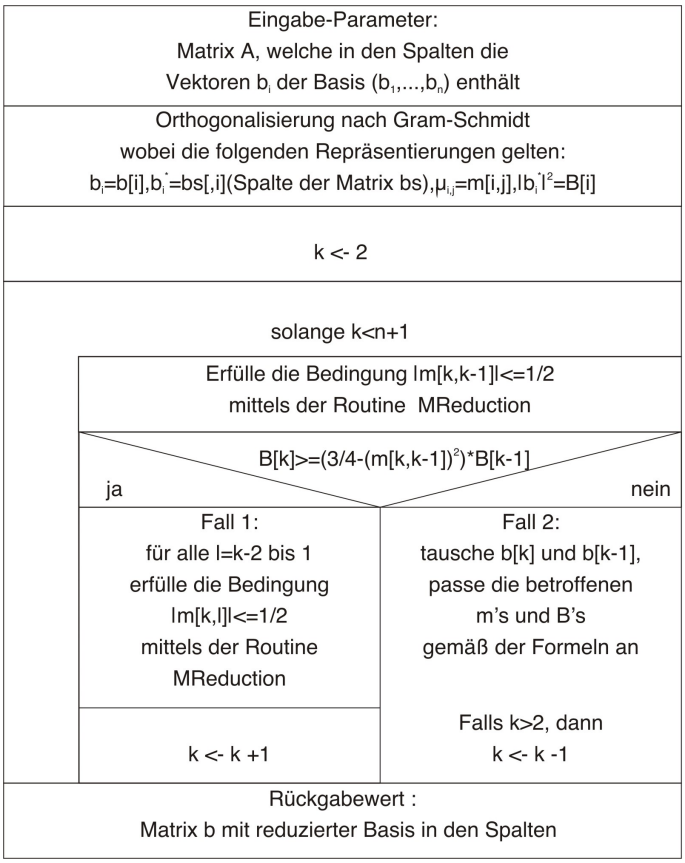

Diagramm 20: Basisreduktion
Sourcen-Code siehe basisreduction.par, *Anhang S.150*

Bemerkung: Der obige Algorithmus verwendet zur Anpassung der $\mu_{i,j}$ den Algorithmus `MReduction`.

Algorithmus 5.2 *Der Algorithmus führt einen der in Fall 1 beschriebenen Schritte zur Erreichung der Bedingung 5.6 durch und passt dabei die jeweiligen $\mu_{i,j}$ sowie b_i entsprechend der Beschreibung zu Fall 1 an.*

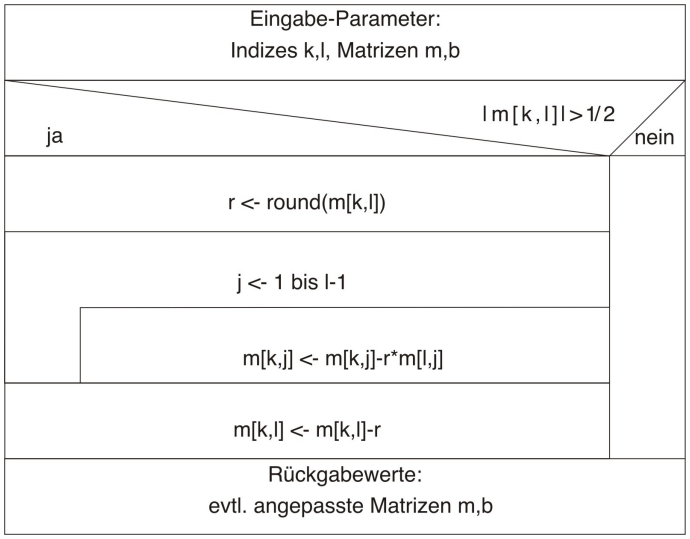

Diagramm 21: MReduction – Hilfsalgorithmus zur Basisreduktion
Sourcen-Code siehe basisreduction.par, *Anhang S.150*

5.2.3 Terminiertheit

Aufgrund der Herleitung des Algorithmus ist sichergestellt, dass dieser das gewünschte Ergebnis liefert, falls der Algorithmus terminiert.

Satz 5.6 *Der Algorithmus 5.1 zur Bestimmung einer reduzierten Basis terminiert.*

Beweis: Man setze

$$d_i := \begin{cases} \det(\Phi(b_j, b_l))_{1\le j,l\le i} & \text{für } 1 \le i \le n \\ 1 & \text{für } i = 0 \end{cases},$$

dann gilt

$$d_i = \prod_{j=1}^{i} \|b_j^*\|^2 \text{ für } 0 \le i \le n$$

und außerdem $d_n = d(L)^2$. Setzt man

$$D = \prod_{i=1}^{n-1} d_i, \tag{5.9}$$

dann ist $D > 0$, und D ändert sich lediglich, wenn die b_i^* sich ändern. Diese Änderung tritt nur im Fall 2 des Algorithmus ein. Beim Eintreten des

Fall 2 wird d_{k-1} gemäß Herleitung durch einen Faktor $< \frac{3}{4}$ reduziert, der Rest der d_i bleibt unverändert. Also wird D ebenfalls durch einen Faktor $< \frac{3}{4}$ verkleinert. Aufgrund der Beschaffenheit des Index k reicht es, um die Terminiertheit des Algorithmus zu zeigen, dass D eine positive untere Schranke besitzt. Daher setze man zunächst

$$\mathrm{m}(L) = \min\{\|x\|^2 \mid x \in L, x \neq 0\},$$

und es gilt $\mathrm{m}(L) > 0$. Für $i > 0$ interpretiere man d_i als das Quadrat der Determinante des Gitters U von Rang i, welches durch die Vektoren $b_1, \ldots, b_i$ im Vektorraum $\sum_{j=1}^{i} \mathbb{R}b_j$ erzeugt wird. Unter den so geschaffenen Voraussetzungen existiert nach einem von J. W. Cassels in [CAS] gezeigten Sachverhalt (s. Kapitel 1, Lemma 4 & Kapitel 2, Theorem 1) ein $x \in U$ mit

$$\|x\|^2 \leq \frac{4}{3}^{\frac{i-1}{2}} d_i^{\frac{1}{i}}$$

und damit $\frac{3}{4}^{\frac{i(i-1)}{2}} \mathrm{m}(L)^i \leq d_i$.

□

5.2.4 Laufzeitbetrachtung

Da im interessierenden Fall der Faktorisierung ganzzahliger Polynome aufgrund der Beschaffenheit der Basis auch die Koeffizienten der Gitterbasis ganzzahlig sind, wird die Laufzeitbetrachtung auf den Fall eines Gitters L mit $L \in \mathbb{Z}^n$ beschränkt.
Man erhält die Gültigkeit des folgenden Satzes.

Satz 5.7 *Sei $(b_1, \ldots, b_n)$ Basis des Gitters $L \subset \mathbb{Z}^n$, $B \in \mathbb{R}^+$ und $B \geq 2$ mit $\|b_i\|^2 \leq B$ für $1 \leq i \leq n$. Dann benötigt der Algorithmus höchstens $\mathcal{O}(n^4 \log(B))$ Elementaroperationen. Diese Operationen werden auf Ganzzahlen mit einer maximalen binären Länge von der Ordnung $\mathcal{O}(n \log(B))$ ausgeführt.*

Beweis: s. [LLL], S. 522-524

□

5.3 Faktoren und Gitter

Nachdem im vorherigen Abschnitt Gitter und reduzierte Basen eingeführt wurden, muss nun der Zusammenhang zwischen den Faktoren eines Polynoms und eben diesen Gitter hergestellt werden. Dieser Teil folgt im Wesentlichen den Schritten im 2. Abschnitt des Papiers von *A. K. Lenstra, H. W. Lenstra & L. Lovász* [LLL], wobei für die Beweisführung des Öfteren auf die Bücher von *Mignotte* [MIG] & [MUS] zurückgegriffen wird.

Zunächst treffen wir für den Verlauf dieses Abschnittes folgende Vereinbarungen: Sei $p \geq 2$ eine Primzahl und $k \in \mathbb{N}^*$. Wir betrachen die beiden Quotientenringe

$$\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} \quad \text{und} \quad \mathbb{Z}/p^k\mathbb{Z}.$$

$f \in \mathbb{Z}[X]$ sei ein Polynom vom Grad $n > 0$. Das Polynom $h \in \mathbb{Z}[X]$ mit $h = \sum_{i=0}^{l} a_i X^i$ und $l = \deg(h)$ habe die folgenden Eigenschaften:

(5.10) $a_l = 1$

(5.11) $(h \mod p^k) \mid (f \mod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$

(5.12) $(h \mod p)$ ist irreduzibel über $\mathbb{F}_p[X]$

(5.13) $(h \mod p)^2 \nmid (f \mod p)$ in $\mathbb{F}_p[X]$

Aus Formel 5.11 folgt insbesondere $(h \mod p) \mid (f \mod p)$ in $\mathbb{F}_p[X]$.

Satz 5.8 *Das Polynom f hat einen bis auf das Vorzeichen eindeutig bestimmten, irreduziblen Faktor $h_0 \in \mathbb{Z}[X]$ mit der Eigenschaft:*

$$(h \mod p) \mid (h_0 \mod p) \text{ in } \mathbb{F}_p[X].$$

Gilt weiter für ein Polynom $g \in \mathbb{Z}[X]$ $g \mid f$, so sind die folgenden Aussagen äquivalent:

(5.14) $(h \mod p) \mid (g \mod p)$ *in* $\mathbb{F}_p[X]$

(5.15) $(h \mod p^k) \mid (g \mod p^k)$ *in* $(\mathbb{Z}/p^k\mathbb{Z})[X]$

(5.16) $h_0 \mid g$ *in* $\mathbb{Z}[X]$

Bemerkung: Insbesondere gilt $(h \mod p^k) \mid (h_0 \mod p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$.

Beweis: Da jeder irreduzible Faktor von f in $\mathbb{Z}[X]$ auch ein nicht notwendigerweise irreduzibler Faktor von f in $\mathbb{F}_p[X]$ ist, aus 5.11 (wg. $a, b \in \mathbb{Z}$, und falls gilt $a \equiv b \mod p^k \Rightarrow a - b = cp^k = dp$ mit $c, d \in \mathbb{Z} \Rightarrow a \equiv b \mod p$) auch $h \mod p \mid f \mod p$ folgt, sowie ferner 5.12 gilt, folgt die Existenz eines in $\mathbb{Z}[X]$ irreduziblen Faktors h_0 mit $h \mod p \mid h_0 \mod p$. Eigenschaft 5.13 sichert die Eindeutigkeit bis auf das Vorzeichen.
(5.15)⇒(5.14): klar

(5.16)⇒(5.14): klar, da $(h_0 \mod p) \mid (g \mod p)$ und $h \mod p \mid h_0 \mod p$
(5.14)⇒(5.16): Aus (5.14) und (5.13) folgt

$$(h \mod p) \nmid (f/g \mod p) \text{ in } \mathbb{F}_p[X],$$

weswegen $h_0 \nmid f/g$ in $\mathbb{Z}[X]$ gelten muss, d. h. $h_0 \mid g$ in $\mathbb{Z}[X]$.

(5.14)⇒(5.15): Aus (5.12) folgt $\mathrm{ggT}(h, f/g) \equiv 1 \mod p$, also existieren $u, v \in \mathbb{Z}[X]$ mit

$$uh + vf/g \equiv 1 \mod p,$$

d. h. es existiert ein $w \in \mathbb{Z}[X]$ mit

$$uh + vf/g = 1 - pw. \tag{5.17}$$

Multipliziert man Gleichung 5.17 mit $(1 + pw + \cdots + (pw)^{k-1})g$ so erhält man wegen $(1 + pw + \cdots + (pw)^{k-1})(1 - pw) = 1 - p^k w^k$ die Existenz von $u', v' \in \mathbb{Z}[X]$ mit

$$u'h + v'f \equiv g \mod p^k.$$

Aufgrund von 5.11 ist damit die linke Seite durch $h \mod p^k$ teilbar; also muss dies auch für die rechte Seite gelten.

□

Für den weiteren Verlauf fixiere man ein $m \in \mathbb{N}^*$ mit $m \geq d = \deg(h)$ und setze

$$L = \left\{ q \in \mathbb{Z}[X] \mid \deg(q) \leq m \text{ und } (h \mod p^k) \mid (q \mod p^k) \text{ in } (\mathbb{Z}/p^k\mathbb{Z})[X] \right\}.$$

Unter Ausnutzung der Vektorraumisomorphie

$$\sum_{i=0}^{m} \mathbb{R}X^i \cong \mathbb{R}^{m+1}$$

mittels $\sum_{i=0}^{m} a_i X^i \mapsto (a_0, a_1, \ldots, a_m)$ gilt $L \subset \mathbb{R}^{m+1}$.

Lemma 5.2 *L ist ein Gitter im $\mathbb{R}^{m+1}$*

Beweis:

$$\mathcal{B} = (b_1, b_2, \ldots, b_{m+1}) \quad \text{mit} \quad b_i = \begin{cases} p^k X^{i-1} & \text{für} \quad 1 \leq i < d+1 \\ h X^{i-(d+1)} & \text{für} \quad d+1 \leq i \leq m+1 \end{cases}$$

bildet mit der o. g. Identifikation eine Basis des $\mathbb{R}^{m+1}$, und es gilt offensichtlich

$$L = \sum_{i=1}^{m+1} \mathbb{Z} b_i.$$

□

Bemerkung. Mit der Definition der Gitterdeterminante gilt damit

$$\det(L) = p^{kd} \quad \text{Notiz: } \mathrm{lc}(hX^{i-(d+1)}) = 1 \quad \text{für} \quad d \leq i \leq m$$

Sei nun h_0 wie in Satz 5.8.

Satz 5.9 *Sei b Element des Gitters L mit $b \neq 0$ und zusätzlich*

$$\|f\|^m \cdot \|b\|^n < p^{kd},$$

dann gilt

$$h_0 \mid b \quad \textit{in } \mathbb{Z}[X].$$

Insbesondere ist also $\mathrm{ggT}(f, b) \neq 1$.

Beweis: Sei $g = \mathrm{ggT}(f, b)$. Nach Satz 5.8 reicht es dann zu zeigen, dass $(h \mod p) \mid (g \mod p)$ über $\mathbb{F}_p[X]$ gilt.
Annahme: $(h \mod p) \nmid (g \mod p)$ *über* $\mathbb{F}_p[X]$
Es folgt also $\mathrm{ggT}(h \mod p, g \mod p) = 1$ in $\mathbb{F}_p[X]$, d. h. , da $(h \mod p)$ irreduzibel in $\mathbb{F}_p[X]$ ist, existieren $\lambda, \mu, \nu \in \mathbb{Z}[X]$, so dass gilt

$$\lambda h + \mu g = 1 - p\nu.$$

Im weiteren Verlauf wird sich herausstellen, dass dies zum gewünschten Widerspruch führt. Setze also $e = \deg(g)$ und $m' = \deg(b)$, dann ist $0 \leq e \leq m' \leq m \leq n$. Betrachtet man nun

$$L' = \{uf + vb + w \mid \begin{array}{l} u, v, w \in \mathbb{Z}[X], \deg(u) < m' - e, \\ \deg(v) < n - e, \deg(w) < e \end{array} \}$$

mit

$$\mathcal{B}' = \{1, X, \ldots, X^{e-1}\} \cup \{X^i f \mid 0 \leq i < m' - e\} \cup \{X^j b \mid 0 \leq j < n - e\}$$

als Basis. Die lineare Unabhängigkeit des Erzeugendensystems $\mathcal{B}'$ folgert man dabei wie folgt:
Sei $uf + vb = w$, dann gilt wegen $g = \mathrm{ggT}(f, b)$ die Beziehung

$$\mathbb{Z}[X] \cdot f + \mathbb{Z}[X] \cdot b = \mathbb{Z}[X] \cdot g$$

und mit $e = \deg(g)$ sowie $e \leq m' \leq n$ folgt $u = v = w = 0$. Die *Hadamardsche Ungleichung* (s. 5.3, S. 64) aus dem ersten Abschnitt liefert uns eine obere Grenze für die Determinante von L':

$$det(L') \leq \|f\|^{m'-e} \cdot \|b\|^{n-e} \leq \|f\|^m \cdot \|b\|^n < p^{kd}. \tag{5.18}$$

Zur Bestimmung einer unteren Grenze zeigen wir die folgende Implikation:

$$z \in L' \quad \text{und} \quad \deg(z) < e + d \quad \Rightarrow \quad \deg(z \mod p^k) < e. \tag{5.19}$$

Erfülle das Polynom z die obige Bedingung; mittels des *Euklidischen Algorithmus* erhält man

$$z = qg + r \quad \text{mit} \quad \deg(r) < e.$$

Multipliziert man nun Gleichung 5.3 mit

$$q(1 + p\nu + \cdots + (p\nu)^{k-1}),$$

so folgt die Existenz von $u', v' \in \mathbb{Z}[X]$ mit

$$u'h + v'qg \equiv q \mod p^k.$$

Weiter ist $b \in L'$, $(h \mod p^k) \mid (g \mod p^k)$ und daher mit o. g. Kongruenz

$$(h \mod p^k) \mid (q \mod p^k).$$

h hat aber wegen 5.10 Höchstkoeffizient 1 und $d = \deg(h)$, außerdem ist wegen $e = \deg(g)$ auch $\deg(q) < d$. Also muss $q = 0$ gelten und damit folgt $\deg(z \mod p^k) < e$.
Sei nun $(b_0, b_1, \ldots, b_{n+m'-e-1})$ eine Basis von L'. Aufgrund eines Theorems von *Cassels* aus der Elementarteilertheorie (siehe Anhang A.2) kann man $\deg(b_j) = j$ für alle j annehmen. Mit der Aussage 5.19 folgt dann aber, dass die Höechstkoeffizienten der b_i mit $e \leq i \leq edl - 1$ durch p^k teilbar sind und somit

$$\det(L') > p^{kd}. \tag{5.20}$$

Da sich 5.18 und 5.20 widersprechen, muss die Annahme falsch gewesen sein. Also gilt $h \mod p \mid g \mod p$ und nach Satz 5.8 $h_0 \mid b$ in $\mathbb{Z}[X]$.

□

Satz 5.10 *Unter Verwendung der Notation der vorangehenden Sätze und der Voraussetzung, dass $(b_1, b_2, \ldots, b_{m+1})$ eine reduzierte Basis von L ist, gilt:*

i.) $\|b_1\| < \left(\frac{p^{kd}}{\|f\|^m}\right)^{\frac{1}{n}} \quad \Rightarrow \quad \deg(h_0) \leq m$

ii.) *Falls zusätzlich*

$$p^{kd} > 2^{\frac{mn}{2}} \binom{2m}{m}^{\frac{n}{2}} \|f\|^{m+n}$$

und $\deg(h_0) \leq m$, *dann gilt*

$$\|b_1\| < \left(\frac{p^{kd}}{\|f\|^m}\right)^{\frac{1}{n}}.$$

Beweis:
zu i.) Wegen Satz 5.9 folgt $h_0 \mid b_1$ und mit $\deg(b_1) \leq m$ folgt die Behauptung.
zu ii.) Sei also $\deg(h_0) \leq m$, dann folgt $h_0 \in L$ (da $(h \mod p^k) \mid (h_0 \mod p^k)$). Da nach Satz 5.8 $h_0 \mid f$ gilt, folgt mit der *Schranke von Mignotte*(siehe Anhang A.3):

$$\|h_0\| \leq \binom{2m}{m}^{\frac{1}{2}} \|f\|.$$

Wendet man nun noch aus dem ersten Abschnitt Satz 5.4 auf $x = h_0$ an, so folgt

$$\|b_1\| \leq 2^{\frac{m}{2}} \|h_0\| \leq 2^{\frac{m}{2}} \binom{2m}{m}^{\frac{1}{2}} \|f\|,$$

und zusammen mit der Voraussetzung zu ii.) ergibt dies

$$\|b_1\| < \left(\frac{p^{kd}}{\|f\|^m}\right)^{\frac{1}{n}}.$$

□

Satz 5.11 *Mit gleicher Notation wie im Satz 5.10 und unter der Voraussetzung aus i.), sowie der angenommenen Existenz eines Index* $j \in \{1, \ldots, m+1\}$ *für den gilt:*

$$\|b_j\| < \left(\frac{p^{kd}}{\|f\|^m}\right)^{\frac{1}{n}}. \tag{5.21}$$

Sei ferner t *das Maximum dieser* j, *dann folgt*

$$\deg(h_0) = m + 1 - t \quad \textit{und} \quad h_0 = \mathrm{ggT}(b_1, \ldots, b_t)$$

und die Ungleichung 5.21 gilt für alle j *mit* $1 \leq j \leq t$.

Beweis:
Sei $J = \{j \in \{1, \ldots, m+1\} \mid$ Bedingung 5.21 ist erfüllt$\}$. Wegen Satz 5.9 folgt $h_0 \mid b_j$ für $j \in J$. Setze also

$$h_1 = \mathrm{ggT}(\{b_j \mid j \in J\}),$$

und damit folgt

$$h_0 \mid h_1 \quad \text{und} \quad \forall j \in J : h_1 \mid b_j,$$

sowie $\deg(b_j) \leq m$. Damit sind diese b_j Elemente der Menge

$$\mathbb{Z}h_1 + \mathbb{Z}h_1X + \cdots + \mathbb{Z}h_1X^{m-deg(h_1)}.$$

Da die b_j als Basiselemente linear unabhängig sind, folgt automatisch

$$\text{card}(J) \leq m + 1 - \deg(h_1).$$

Verwendet man nun wie im Beweis von Satz 5.10 die *Schranke von Mignotte* (siehe Anhang A.3) so folgt:

$$\|h_0\| \leq \binom{2m}{m}^{\frac{1}{2}} \|f\| \quad \text{für alle} \quad i \geq 0$$

Weiter gilt aufgrund der Definition des Gitters L für $i = 0, 1, \ldots, m - \deg(h_0)$ $h_0X^i \in L$ und damit nach dem Satz 5.5 über linear unabhängige Gitterelemente:

$$\|b_j\|^2 \leq 2^m \left(\binom{2m}{m}^{\frac{1}{2}} \|f\| \right)^2$$

für $1 \leq j \leq m + 1 - \deg(h_0)$. Mit der Voraussetzung i.) aus Satz 5.10 ergibt sich

$$\text{card}(J) \geq m + 1 - deg(h_0).$$

Also folgt insgesamt aus der oberen und der unteren Grenze für die Kardinalität von J

$$\deg(h_0) = \deg(h_1) = m + 1 - t \quad \text{mit} \quad J = \{1, \ldots, t\}.$$

Bleibt noch zu zeigen, dass h_1 bis auf das Vorzeichen äquivalent zu h_0 ist. Es reicht zu zeigen, dass h_1 primitiv ist. Setze $d_j = \text{cont}(b_j)$ für ein $j \in J$; dann ist das Polynom b_j/d_j teilbar durch h_0 und mit Satz 5.8 gilt $b_j/d_j \in L$. b_j ist allerdings Basiselement von L, und somit gilt $d_j = 1$. Also ist b_1 primitiv, und das Gleiche gilt auch für h_1 .

□

5.4 Faktorisierungsalgorithmus

Nachdem im vorherigen Abschnitt der theoretische Zusammenhang zwischen Gitter, reduzierten Basen und Polynomen hergestellt wurde, kann nun mit Hilfe der dort gezeigten Eigenschaften der sog. LLL-Algorithmus zur Faktorisierung schrittweise hergeleitet werden.

Sei für den Verlauf des Abschnittes $f \in \mathbb{Z}[X]$ mit $\deg(f) = n > 0$ und f primitiv. Um den zu beschreibenden Algorithmus einfach und überschaubar zu halten, betrachtet man zunächst zwei Hilfsalgorithmen.

5.4.1 Hilfsalgorithmus aux1

Gegeben sei Polynom $f \in \mathbb{Z}[X]$ mit $n = \deg(f) \geq 1$, eine Primzahl p, $k \in \mathbb{N}^*$, sowie ein Polynom $h \in \mathbb{Z}[X]$, welches die Bedingungen 5.10 bis 5.13 aus Abschnitt 5.3 erfüllt. Die Koeffizienten von h seinen reduziert modulo p^k. Zusätzlich sei ein $m \in \mathbb{N}^*$ mit $m \geq d$ gegeben, so dass

$$p^{kd} > 2^{\frac{mn}{2}} \binom{2m}{m}^{\frac{n}{2}} \|f\|^{m+n}.$$

Unter diesen Bedingungen sind die Voraussetzungen für Satz 5.10, S. 75 gegeben.

Hilfsalgorithmus 5.1 (aux1) *Seien f, h, m, p, k gegeben und erfüllen die eingehend beschriebenen Bedingungen. Dann entscheidet der Algorithmus, ob für das existierende Polynom h_0 $\deg(h_0) \leq m$ gilt. Falls dies der Fall ist, wird h_0 bestimmt.*

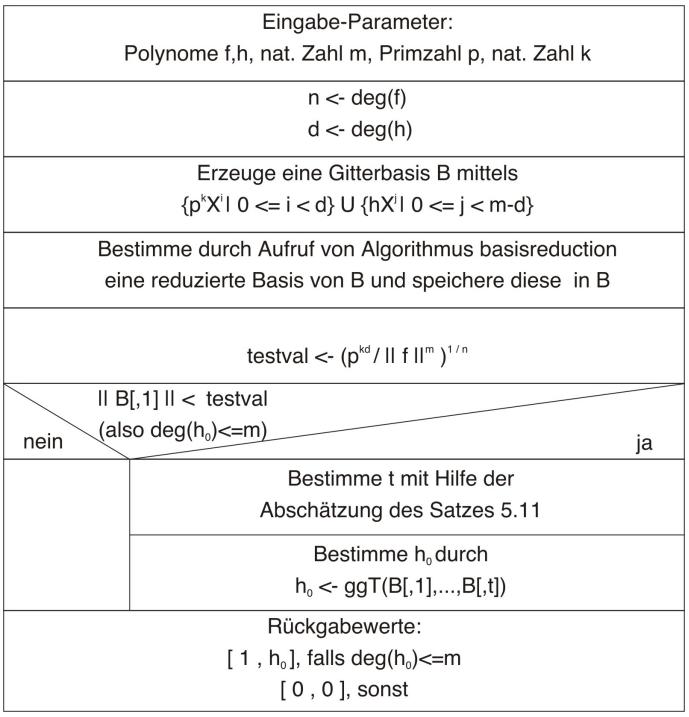

Diagramm 22: Hilfsalgorithmus aux1
Sourcen-Code siehe auxlll.par*, Anhang S.153*

Mit Hilfe von Lemma 5.2, S. 73 wird im Algorithmus zunächst eine Basis des Gitters L bestimmt. Durch Anwendung von Algorithmus 5.1, S. 69 erhält

man dann eine reduzierte Basis. Satz 5.10 und Satz 5.11 liefern direkt den Rückgabewert des Algorithmus.

Satz 5.12 (Laufzeit des Hilfsalgorithmus AUX1) *Die Zahl der zur Ausführung von* AUX1 *notwendigen Elementaroperationen ist von der Ordnung* $\mathcal{O}(m^4 k \log p)$. *Die binäre Länge der auftretenden Ganzzahlen ist maximal* $\mathcal{O}(mk \log p)$

Beweis: s. [LLL], S. 530; [MI2], S. 280

□

5.4.2 Hilfsalgorithmus aux2

Aufbauend auf dem Hilfsalgorithmus *aux1* aus 5.1 ergibt sich der zweite Hilfsalgorithmus *aux2*. Seien hier das Polynom $f \in \mathbb{Z}[X]$ mit $n = \deg(f) \geq 1$, eine Primzahl p, sowie ein Polynom $h \in \mathbb{Z}[X]$, welches die Bedingungen 5.10 bis 5.13 für $k = 1$ erfüllt, gegeben. Weiterhin seien die Koeffizienten von h reduziert modulo p.

Der Algorithmus arbeitet nun nach folgenden Schritten.

i.) Sei $d = \deg(h)$. Falls $d = n$ gilt, dann ist h_0 bereits durch f gegeben, und der Algorithmus bricht ab. Falls $d < n$ ist, bestimmt man die kleinste positive Zahl k, so dass

$$p^{kd} > 2^{\frac{n(n-1)}{2}} \binom{2(n-1)}{n-1}^{\frac{n}{2}} \|f\|^{2n-1}. \tag{5.22}$$

erfüllt ist.

ii.) Mittels Anwendung des linearen Hensel Lifts wird nun die Zerlegung modulo p zu einer Zerlegung modulo p^k geliftet. Dabei wird nach Satz 3.13, S. 47 h so geliftet, dass sich h mod p nicht verändert. Damit gilt Bedingung 5.11 für das oben bestimmte k und wg. der Voraussetzung sind auch 5.10, 5.12 und 5.13 erfüllt.

Für das weitere Vorgehen kann angenommen werden, dass die Koeffizienten von h modulo p^k reduziert sind.

iii.) Sei nun u die größte ganze Zahl für die $d \leq \frac{n-1}{2^u}$ gilt, und man führe den Hilfsalgorithmus *aux1* mit den Parametern f, h, p, k und $m = \left\lceil \frac{n-1}{2^u} \right\rceil, \left\lceil \frac{n-1}{2^{u-1}} \right\rceil, \ldots, \left\lceil \frac{n-1}{2} \right\rceil, n-1$ aus. Die Ausführung wird abgebrochen, so bald ein passendes h_0 gefunden ist. Da in der Ungleichung 5.22 für m der ungünstigste Fall $n-1$ gewählt wurde, sind aufgrund der Definition von u offentsichtlich die Voraussetzungen für den Algorithmus *aux1* für die obigen m erfüllt.

iv.) Falls keines der obigen m ein passendes h_0 liefert, gilt $\deg(h_0) > n-1$ und damit $h_0 = f$.

Hilfsalgorithmus 5.2 (aux2) *Seien f, h, p gegeben und erfüllen die oben beschriebenen Bedingung. Dann bestimmt der Algorithmus den irreduziblen Faktor $h_0 \in \mathbb{Z}[X]$ von f aus Satz 5.8, S. 72, für den $h \mod p \mid h_0 \mod p$ gilt.*

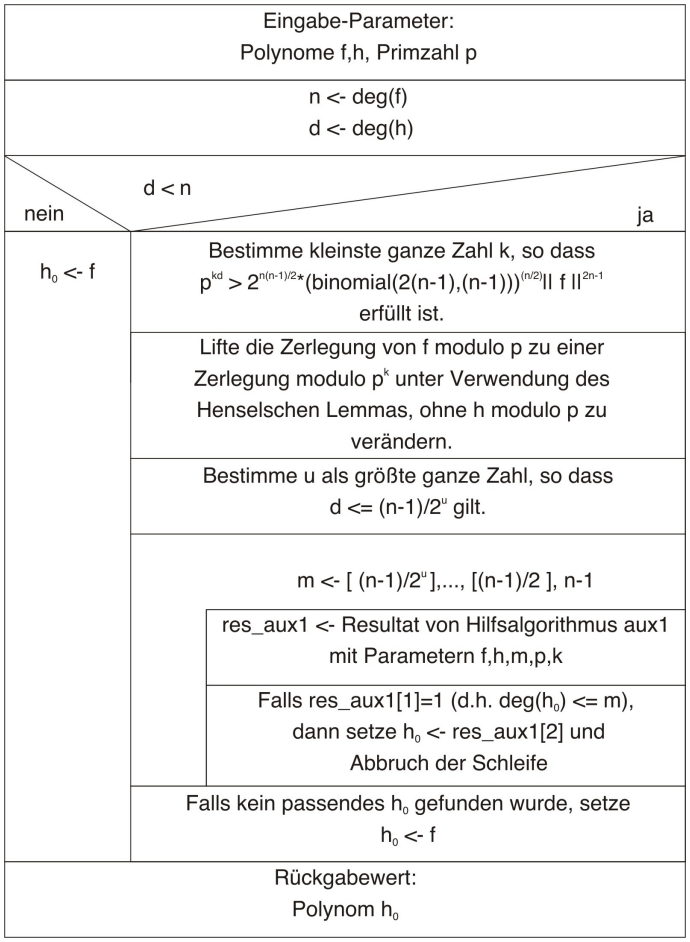

Diagramm 23: Hilfsalgorithmus aux1
Sourcen-Code siehe auxlll.par*, Anhang S.153*

Bemerkung zur Implementierung: Die in *PARI/GP* realisierte Version dieses Algorithmus verwendet nicht direkt die in Schritt i.) beschriebene Schranke zur Bestimmung der Zahl k. Da die Ungleichung 5.22 durch Annahme des ungünstigsten Falls $m = n-1$ aus der Bedingung in Satz 5.10 hervorgeht,

erreicht k sehr schnell Werte mit $k > 100$. Da p^k unmittelbar im Basisreduktionsalgorithmus in den einzelnen Komponenten der Vektoren aufgrund der Beschaffenheit der Basis des Gitters L auftritt, wird dieses Verfahren selbst für kleine p unpraktikabel aufgrund der Größe der verwendeten Zahlen.
Ausgehend von einem Vorschlag im Orginaldokument [LLL] wird hier m schrittweise für die Werte von d bis $n-1$ getestet bis h_0 gefunden ist. Trotz des im schlimmsten Falle gestiegenen Mehraufwandes hat sich eine signifikante Laufzeitverbesserung für die getesteten Polynome ergeben.

Satz 5.13 (Laufzeit des Hilfsalgorithmus AUX2) *Sei $h_0 \in \mathbb{Z}[X]$ mit $m_0 = \deg(h_0)$ irreduzibler Faktor von f, und sei h_0 durch den Algorithmus* AUX2 *bestimmt worden. Die Zahl der zur Bestimmung notwendigen Elementaroperationen ist von der Ordnung $\mathcal{O}(m_0(n^5 + n^4 \log \|f\| + n^3 \log p))$; die maximale binäre Länge ist $\mathcal{O}(n^3 + n^2 \log \|f\| + n \log p)$*

Beweis: s. [LLL], S. 531; [MI2], S. 280-282

□

5.4.3 Algorithmus LLL

Nachdem ein großer Teil der Berechnungen auf die beiden Hilfsalgorithmen ausgelagert ist, kann der Algorithmus zur Faktorisierung eines beliebigen Polynoms $f \in \mathbb{Z}[X]$ mit $\deg(f) \geq 1$ einfach formuliert werden. Wie bereits beim *Cantor-Zassenhaus*-Algorithmus geschehen, können dabei beliebige Polynome und quadratfreie, primitive Polynome unterschieden werden. Der erste Fall wird abgesehen von einer kleinen Anpassung wie in Abschnitt 4.3, S. 59 behandelt und wird daher im zweiten Teil dieser Beschreibung nur noch informell wiedergegeben.

Sei $f \in \mathbb{Z}[X]$ ein quadratfreies, primitives Polynom mit $n = \deg(f) \geq 1$. Nach dem Kriterium von Sylvester ist f genau dann quadratfrei, wenn $\operatorname{res}(f, f') \neq 0$ gilt.
Unter diesen Voraussetzungen bestimmt man eine Primzahl p, so dass

$$\operatorname{res}(f, f') \not\equiv 0 \mod p$$

gilt, damit ist gewährleistet, dass auch $f \mod p$ quadratfrei ist. Zudem gilt aufgrund der Darstellung der Resultante aus Satz 2.4, S. 22 auch $\deg(f \mod p) = n$. Mittels des Algorithmus von *Berlekamp* bestimmt man im nächsten Schritt die Faktorisierung von $f \mod p$. Aufgrund der beschriebenen Voraussetzungen gilt Bedingung 5.13, S. 72 für jeden so erhaltenen irreduziblen Faktoren $h \mod p$ von $f \mod p$.

Das weitere Vorgehen stellt sich nun wie folgt dar. Unter der Annahme, dass eine Zerlegung von f in $\mathbb{Z}[X]$ mit $f = f_1 f_2$ (, zu Beginn durch $f_1 = 1$ und

$f_2 = f$,) gegeben ist, so dass die vollständige Faktorisierung von f_1 in $\mathbb{Z}[X]$ und die von f_2 in $\mathbb{F}_p[X]$ bekannt sind, verfährt man wie folgt.
Falls $f_2 = \pm 1$ ist, so ist $f = \pm f_1$ komplett faktorisiert in $\mathbb{Z}[X]$, und der Algorithmus bricht ab. Falls hingegen $\deg(f_2) \geq 1$, dann wähle man o. B. d. A. den ersten Faktor h aus der aktuellen Liste der Faktoren der Faktorisierung von f_2 in $\mathbb{F}_p[X]$. Dabei kann angenommen werden, dass h normiert und die Koeffizienten modulo p reduziert sind. Mit f_2, h, p sind damit die Voraussetzungen für den Hilfsalgorithmus *aux2* gegeben, und man bestimmt den eindeutig bestimmten, irreduziblen Faktor h_0 von f_2 in $\mathbb{Z}[X]$ für den $h \mod p \mid h_0 \mod p$ gilt.
Abschließend ersetzt man f_1 durch $f_1 \cdot h_0$, f_2 durch f_2/h_0 und von der Liste der irreduziblen Faktoren modulo p werden all diejenigen entfernt, die $h_0 \mod p$ teilen. Diese Liste enthält dann wieder die Faktorisierung des aktuellen $f_2 \mod p$. Nun beginnt die Betrachtung für f_1, f_2 erneut.

Algorithmus 5.3 (LLL-SF) *Seien ein quadratfreies, primitives Polynom $f \in \mathbb{Z}[X]$ mit $n = \deg(f) \geq 1$ und die Resultante $\operatorname{res}(f, f')$ gegeben. Dann liefert der Algorithmus die vollständige Faktorisierung von f in irreduzible Faktoren.*

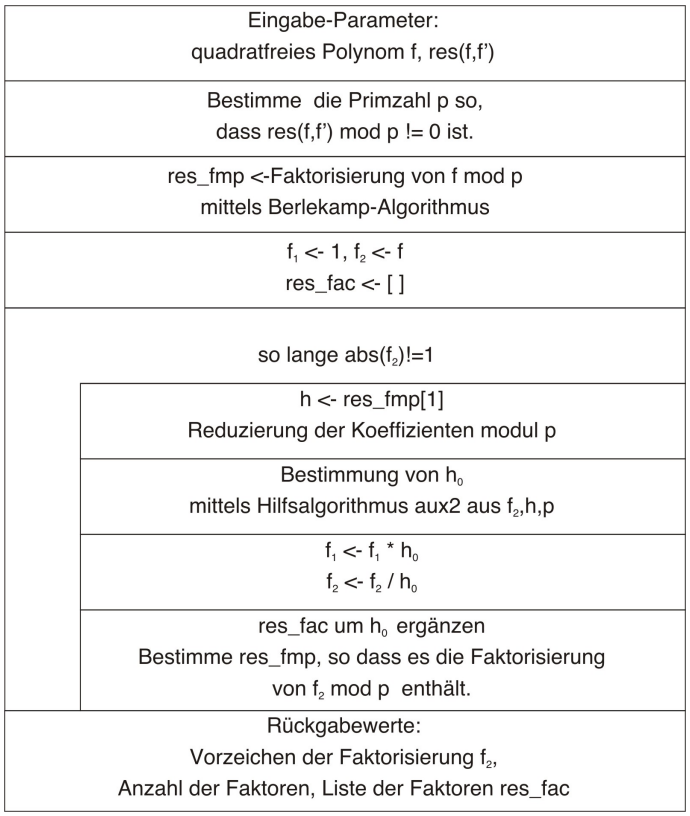

Diagramm 24: LLL-SF
Sourcen-Code siehe factor_lll.par*, Anhang S.155*

Sei nun der Fall eines beliebigen Polynoms $f \in \mathbb{Z}[X]$ mit $\deg(f) \geq 1$ betrachtet.

Algorithmus 5.4 (LLL) *Unter Voraussetzung des LLL-Algorithmus zur Faktorisierung eines primitiven, quadratfreien Polynoms (s.* `factor_lllsf_calc`*), liefert dieser Algorithmus die Faktorisierung eines beliebigen Polynoms $f \in \mathbb{Z}[X]$ mit $\deg(f) \geq 1$ in irreduzible Faktoren.*

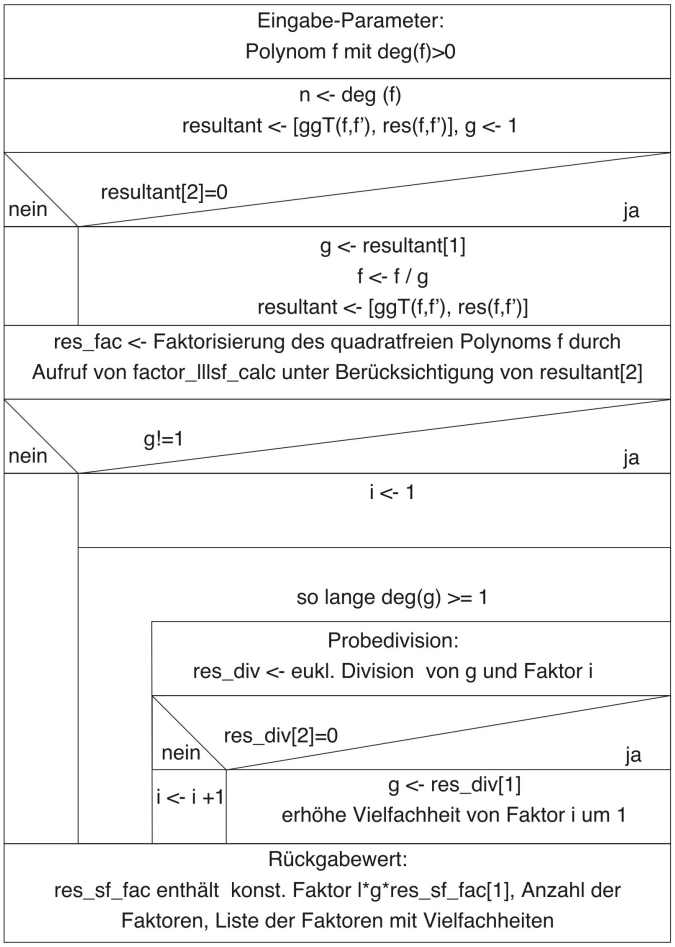

Diagramm 25: Faktorisierung in $\mathbb{Z}[X]$
Sourcen-Code siehe factor_lll.par*, Anhang S.155*

Satz 5.14 (Laufzeit des LLL-Algorithmus) *Der* LLL*-Algorithmus faktorisiert jedes normierte Polynom $f \in \mathbb{Z}[X]$ mit $n = \deg(f)$. Die Zahl der notwendigen Berechnungen ist von der Ordnung $\mathcal{O}(n^6 + n^5 \log \|f\|)$. Die maximale binäre Länge der dabei verwendeten Ganzzahlen ist $\mathcal{O}(n^3 + n^2 \log \|f\|)$.*

Beweis: s. [LLL], S. 531-533; [MI2], S. 280-282 □

6 Zusammenfassung

In den vorhergehenden Kapiteln werden zwei Algorithmen zur Faktorisierung von Polynomen über $\mathbb{Z}$,

- der Algorithmus von *Cantor-Zassenhaus* (Kapitel 4, S. 54),
- der *LLL*-Algorithmus (Kapitel 5, S. 61),

hergeleitet und parallel dazu durch den entsprechenden Sourcecode im Anhang begleitet. Es zeichnet sich dabei gut ab, wieviel Theorie investiert werden muss, um von der sehr schnell zu zeigenden Existenz eines Faktorisierungsalgorithmus (*Schranke von Mignotte*, s. Anhang A.3, S. 94), sowie dem daraus resultierenden Brute-Force-Algorithmus, zu einem Algorithmus zu gelangen, dessen Laufzeit im Mittel von polynomieller Ordnung (s. Cantor-Zassenhaus, S .60) ist, oder gar im schlechtesten Fall eine Laufzeit von polynomieller Ordnung (s. LLL, S .83) aufweist.
In beiden Fällen resultiert die zu investierende Theorie ebenfalls in zwar überschaubaren Hilfsalgorithmen, sie schafft aber dadurch jeweils einen umso komplexeren Gesamtalgorithmus.
Fast alle Quellen, der hier gezeigten Algorithmen, präsentieren die Algorithmen in Pseudo-Code, welcher in Ermangelung von Sourcecode, jedoch manchmal auch aufgrund zu unpräziser Formulierungen noch zu viel Spielraum zur Implementierung liefert. Der einzeln dargestellte Algorithmus mag hier problemlos lauffähig sein, ein Zusammenfügen zu einem komplexen Algorithmus wird hier aufgrund fehlender Konventionen und beispielsweise gerne vernachlässigter Konstanten leicht zu einer Herausforderung.
In dieses Bild passt auch, dass selbst *PARI/GP* in der aktuellen Version 2.1.5 eine falsche Faktorisierung im Falle des Polynoms

$$\begin{aligned} f(X) \;=\; & (-1)\cdot(x+1)\cdot \\ & (3x^9-5x^8+6x^7-9x^6+8x^5-9x^4+6x^3-5x^2+3x-3)\cdot \\ & (4x^9+4x^8-2x^7-3x^5+2x^4-x^3+4x^2-4)\cdot \\ & (x^9-x^8+x^7-2x^6+4x^5+3x^4-4x^3-x^2-x+5)\cdot \\ & (2x^9+2x^7+2x^6+4x^5-2x^2+x+2)\cdot \\ & (4x^{10}+3x^9-4x^8+3x^7+4x^6-5x^5+2x^4+2x^3-x^2+2x+2)\cdot \\ & (2x^{10}-4x^9-3x^8-2x^7-5x^6-4x^5+3x^4+3x^3+x^2-5x-3)\cdot \\ & (4x^{10}+4x^9+2x^8+4x^7+2x^6+2x^5+2x^4-x^3+2x^2-4x-3)\cdot \\ & (3x^{10}+2x^9+4x^8+5x^5+2x^4-3x^3-x^2+5x-3)\cdot \\ & (5x^{10}-x^9-3x^8-x^7+4x^6+4x^4-x^3-4x^2+5)\cdot \\ & (2x^{10}-x^9+2x^8-4x^7+x^6-x^5-2x^4-5x^3-4x^2-3x+4) \end{aligned}$$

liefert. Trotzdem muss natürlich die enorme Geschwindigkeit bei der Faktorisierung mittels *PARI/GP* herausgestellt werden, auch wenn diese sicher-

lich nicht als Entschuldigung dienen soll und kann. Laut Aussage der Dokumentation wird hier zur Faktorisierung eine Implementierung des *LLL*-Algorithmus verwendet. Dessen Geschwindigkeit steht zunächst scheinbar im Widerspruch zu der im Rahmen dieser Arbeit implementierten Variante, welche nahezu um Faktor 10 langsamer ist, als die gewählte Implementierung des Algorithmus von *Cantor-Zassenhaus*. Dies entspricht dem bei Cohen [COH] geschilderten Verhalten in der Praxis. Dass die Unterschiede hier so drastisch ausfallen, findet aber eine einfache Erklärung in der Größe der beim Basisreduktionsalgorithmus auftretenden Komponenten (s. Bemerkung zum Algorithmus 5.2, S. 80), sowie in der hohen Zahl an Funktionsaufrufen zwecks Komponentenzugriff. Eine Implementierung in einer Hochsprache, welche ein Compiler nutzt, verspricht hier wesentliche Besserung. Weiteres Optimierungspotential ist z. B. in der Reduzierung von Funktionsaufrufen durch Zusammenfassung von Routinen, in der problembezogenen Spezialisierung der allgemeinen Hilfsalgorithmen oder durch die maschinennahe Implementierung bestimmter Routinen gegeben.

A Beweisergänzungen

Im Rahmen der in den Hauptkapiteln geführten Beweise werden einige Sätze und Resultate als bekannt vorausgesetzt. Die gängigen Vorlagen zu den aufgeführten Algorithmen verweisen hier gerne auf die jeweiligen Orginaldokumente, was wegen der angestrebten Vollständigkeit bei der Beweisführung und dem Aufbau der Argumentation für diese Arbeit nicht zweckmäßig erscheint. Die dem jeweiligen Sachverhalt angepaßten Sätze und Beweise werden daher an dieser Stelle aufgeführt. Der Abschnitt A.1 liefert zudem einen schnellen Überblick über die Grundlagen zu Polynomringen.

A.1 Ergänzende Grundlagen zu Polynomen

A.1.1 Integritätsringe und euklidische Ringe

Definition A.1 (Integritätsring) *Ein* Integritätsring R *ist ein nullteilerfreier kommutativer Ring mit Einselement.*

Beispiele: $\mathbb{Z}$ und $K[X]$ mit K Körper.

Sei R im Weiteren Integritätsring. Seien $x, y \in R$. Man sagt x teilt y oder $x \mid y$, wenn ein $q \in R$ existiert mit $y = qx$, gilt dies nicht, so schreibt man $x \nmid y$. Ein Element u heißt *Einheit*, falls ein $v \in R$ existiert mit $uv = 1$. R^* ist die *Menge aller Einheiten*. Für $K[X]$ gilt $(K[X])^* = K^*$. Zwei Elemente $x, y \in R\backslash\{0\}$ heißen *assoziiert*, falls eine Einheit u existiert mit $x = uy$.

Satz A.1 *Seien $x, y \in R\backslash\{0\}$, dann gilt:*

$$x \mid y \text{ und } y \mid x \quad \Rightarrow \quad x, y \text{ sind assoziiert.}$$

Beweis: Es gibt $q_1, q_2 \in R$ so, dass $y = q_1 x$ und $x = q_2 y$ gilt. Daher ist $y = q_1 q_2 y$, also $0 = (q_1 q_2 - 1)y$. Da $y \neq 0$ und R nullteilerfrei ist, gilt $q_1 q_2 = 1$ und damit $q_1, q_2 \in R^*$.

□

Definition A.2 *Seien $x, y \in R$. $d \in R$ heißt größter gemeinsamer Teiler (*ggT*) von x, y, falls die beiden folgenden Bedingungen erfüllt sind:*

i.) $d \mid x$ *und* $d \mid y$

ii.) Gilt für ein $d' \in R$ $d' \mid x$ *und* $d' \mid y$, *so folgt* $d' \mid d$.

Im Falle von $x = y = 0$ ist Null der eindeutig bestimmte ggT. Sind d_1, d_2 ggT von x und y, so folgt wegen *ii.)*, dass d_1 und d_2 assoziiert sind. x, y heißen *teilerfremd*, wenn $\mathrm{ggT}(x, y) = 1$ gilt.

Definition A.3 (Euklidischer Ring) *Ein Integritätsring R heißt* euklidischer Ring, *falls es ein $\beta \in \mathrm{Abb}(R, \mathbb{N})$ gibt, für das gilt:*
Für je zwei $x, y \in R$ und $y \neq 0$ existiert eine Darstellung

$$x = qy + r \quad mit \quad q, r \in R$$

mit $r = 0$ oder $\beta(r) < \beta(y)$.

In anderen Worten, in *euklidischen Ringen* ist das Teilen mit Rest möglich.

Satz A.2 *$K[X]$ mit K Körper ist euklidischer Ring.*

Beweis: Sei $f \in K[X]$, dann definiert man $\beta(f) = \deg(f)$. (Entgegen der üblichen Konvention sei hier der Grad des Nullpolynoms als 0 definiert.) Man betrachte nun $f, g \in K[X]$ mit $g \neq 0$. Falls $\deg(g) = 0$ gilt, ist $g \in K^*$, und man kann f durch g ohne Rest teilen. Daher sei jetzt $m = \deg(g) > 0$. Mittels Induktion über $n = \deg(f)$ zeige man:

$$f = qg + r \quad \text{mit} \quad \deg(r) < m.$$

Für $n < m$ ist die Behauptung trivial; sei also $n \geq m$. Das Polynom $f_1 = f - \frac{f_n}{g_m} X^{n-m} g$ erfüllt $\deg(f_1) < n$, und damit existiert nach Induktionsvoraussetzung eine Darstellung

$$f_1 = q_1 g + r \quad \text{mit} \quad \deg(r) < m.$$

Somit ist $f = (\frac{f_n}{g_m} X^{n-m} + q_1)g + r$.

□

Satz A.3 *In euklidischen Ringen besitzen je zwei Elemente x, y einen ggT.*

Beweis: Im Fall $y = 0$ ist x der ggT. Sei daher $y \neq 0$ und $\beta \in \mathrm{Abb}(R, \mathbb{N})$ gemäß Definition A.3 gegeben. Durch Induktion über $\beta(y)$ erhält man :
Induktionsanfang: $\beta(y) = 0$, d. h. bei der euklidischen Division bleibt kein Rest; y ist ein ggT.
Induktionsschritt: Mittels euklidischer Division folgt

$$x = qy + r \quad \text{mit} \quad r = 0 \quad \text{oder} \quad \beta(r) < \beta(y).$$

Falls $r = 0$ ist, ist y ein ggT, sonst kann man die Induktionsvoraussetzung auf y, r anwenden und erhält $d = ggT(r, y)$ mit $d \mid r$ und $d \mid y$. Daraus folgt $d \mid x$. Andererseits folgt aus $d' \mid x$ und $d' \mid y$ auch $d' \mid r$ und damit $d' \mid d$ nach Voraussetzung über d. Also ist d ein ggT von x und y.

□

Bemerkung: Der obige Beweis liefert offensichtlich einen Algorithmus zur Bestimmung eines ggT, den *euklidischen Algorithmus*.

A.1.2 Idealtheorie und faktorielle Ringe

Die sog. Idealtheorie liefert die restlichen Grundlagen, die zur Betrachtung und Faktorisierung von Polynomen benötigt werden.

Definition A.4 *Sei $I \subset R$, R kommutativer Ring. I heißt* Ideal, *falls $I \neq \oslash$ additive Untergruppe von R ist und für alle $\lambda \in R$ gilt:*

$$x \in I \quad \Rightarrow \quad \lambda x \in I.$$

Bemerkung: $Rx = \{\lambda x \mid \lambda \in R\}$ mit $x \in R$ ist offenbar das kleinste Ideal, das x enthält; es wird das von x erzeugte *Hauptideal* (x) genannt. Allgemein ist offensichtlich auch $Rx_1 + \ldots + Rx_n$ für $x_1, \ldots, x_n \in R$ ein Ideal, kurz $(x_1, \ldots, x_n)$.
Falls R auch Integritätsring ist, so wird der Zusammenhang zwischen Idealtheorie und Teilbarkeitsbetrachtungen durch nachfolgenden Satz gegeben.

Satz A.4 *Sei R Integritätsring, dann gelten die folgende drei Aussagen.*

i.) *Für $x, y, \in R$ gilt:* $x \mid y \quad \Leftrightarrow \quad (y) \subset (x)$

ii.) $x, y, \in R$ *sind assoziiert* $\quad \Leftrightarrow \quad (x) = (y)$

iii.) $u \in R^* \quad \Leftrightarrow \quad (u) = R$

Beweis: i.) '$\Rightarrow$': Es existiert ein $q \in R$ mit $y = qx$, d. h. $\lambda y = (\lambda q)x$ für $\lambda \in R$ also $(y) \subset (x)$. '$\Leftarrow$': Wg. $(y) \subset (x)$ gilt insbes. für ein $\lambda \in R \quad y = \lambda x$, d. h. $x \mid y$.
ii.) Anwendung von i.)
iii.)'$\Rightarrow$': Sei $x \in R$ und $u \in R^*$, dann existiert ein $\lambda' \in R^*$ mit $\lambda' u = 1$, d. h. $(x\lambda')u = x$, also $R \subset (u)$. '$\Leftarrow$': klar

□

Korollar A.1 *Für $x_1, \ldots, x_n \in R$, R Integritätsring, gilt: $d \in R$ ist genau dann gemeinsamer Teiler der x_i, wenn $(x_1, \ldots, x_n) \subset (d)$ gilt.*

Definition A.5 (Hauptidealring) *Sei R Integritätsring. R heißt* Hauptidealring, *wenn jedes Ideal $I \subset R$ ein Hauptideal ist.*

Satz A.5 *Jeder euklidische Ring ist ein Hauptidealring.*

Beweis: Sei R euklidischer Ring mit einer Abb. β gem. Definition A.3. Das Nullideal ist offensichtlich ein Hauptideal. Sei also I Ideal mit $I \neq \{0\}$, dann ist $M = \{n \in \mathbb{N} \mid \exists \quad a \in I : \beta(a) = n\} \neq \oslash$. Sei $k = \min(M)$ und $a \in I \backslash \{0\}$ so gewählt, dass $\beta(a) = k$ gilt. Dann ist $I = (a)$, denn würde ein $b \in I$ und $b \notin (a)$, dann würden $q, r \in R$ existieren, so dass $b = qa + r$, $r \neq 0$ und $\beta(r) < \beta(a) = k$. Da $r = b - qa \in I$ wäre, würde man einen Widerspruch zu $k = \min(M)$ erhalten.

□

Folgerung: In Hauptidealringen existiert ein ggT. Denn falls $x_1, \ldots, x_r \in R$, R Hauptidealring und $I = (x_1, \ldots, x_r)$, dann existiert ein $d \in R$ mit $I = (d)$. Wegen Korollar A.1 ist dann d gemeinsamer Teiler. d ist ein ggT, denn für jeden anderen gemeinsamen Teiler d' gilt $d' \mid d$, da $d' \mid d \quad \Leftrightarrow \quad (d) \subset (d')$.

Satz A.6 *Sei R ein Hauptidealring, $x_1, \ldots, x_r \in R$ und d ein ggT der x_i. Dann existieren $\lambda_1, \ldots, \lambda_r \in R$ mit*

$$d = \lambda_1 x_1 + \ldots + \lambda_r x_r.$$

Beweis: unmittelbar mittels obiger Folgerung

□

Definition A.6 *Sei R Integritätsring.*

i.) $a \in R\backslash(R^* \cup \{0\})$ *heißt* irreduzibel, *wenn es keine Zerlegung $a = xy$ mit $x, y \in R\backslash R^*$ gibt.*

ii.) $p \in R\backslash(R^* \cup \{0\})$ *heißt* Primelement, *wenn für alle $a, b \in R\backslash\{0\}$ gilt: $p \mid ab \quad \Rightarrow \quad p \mid a$ oder $p \mid b$.*

Satz A.7 *Sei R Integritätsring. Dann gilt:*

i.) Jedes Primelement p ist irreduzibel.

ii.) Falls R sogar Hauptidealring ist, so ist jedes irreduzible a Primelement.

Beweis: i.) Annahme: p reduzibel, d. h. es existieren $x, y \in R\backslash R^*$ mit $p = xy$. Da p Primelement ist, folgt $p \mid x$ oder $p \mid y$. O. B. d. A. gelte $p \mid x$. Also sind p und x assoziiert, da offensichtlich $x \mid p$ gilt. Damit muß $y \in R^*$ gelten, was der Annahme widerspricht.
ii.) Sei $x, y \in R\backslash\{0\}$ mit $a \mid xy$, d. h. es existiert ein $b \in R$ mit $xy = ab$. Es ist zu zeigen, dass falls $a \nmid x$ folgt $a \mid y$. Wegen R Hauptidealring existiert ein d mit $(a, x) = (d)$, wobei $d \neq 0$ ist. Falls $d \notin R^*$ folgt wegen der Irreduzibilität von a, dass a und d assoziiert sind; damit folgt aus $d \mid x$ auch $a \mid x$ im Widerspruch zur Voraussetzung. Sei also $d \in R^*$. Es existieren $\alpha, \beta \in R$ mit $\alpha a + \beta x = 1 \quad \Rightarrow \quad y = \alpha a y + \beta x y = (\alpha y + \beta b)a$ und somit $a \mid y$.

□

Satz A.8 *Sei R Integritätsring. Dann sind folgende Aussagen äquivalent:*

i.) Zu jedem $a \in R\backslash(R^* \cup \{0\})$ existieren irreduzible Elemente $q_1, \ldots, q_r \in R$ mit $a = q_1 \cdots q_r$. Gilt für irreduzible $q_1, \ldots, q_r$ und $q'_1, \ldots, q'_s$ $q_1 \cdots q_r = q'_1 \cdots q'_s$, dann ist $r = s$, und ggf. durch Umindizierung kann erreicht werden, dass q_i und q'_i assoziiert sind.

ii.) Zu jedem $a \in R\backslash(R^* \cup \{0\})$ gibt es Primelemente $p_1, \ldots, p_r \in R$ mit $a = p_1 \cdots p_r$.

Beweis: "ii.) ⇒ i.):" klar, da jedes Primelement irreduzibel ist.
'i.) ⇒ ii.):" Man zeige, dass jedes irreduzible $q \in R$ unter diesen Bedingungen bereits Primelement ist. Es gelte also $q \mid ab$ mit $a, b \in R$. Die Sonderfälle $a = b = 0, a, b \in R^*$ sind offensichtlich klar. Sei also $c \in R\backslash(R^* \cup \{0\})$ und $ab = qc$. Dann gibt es irreduzible Elemente $q_1, \ldots, q_r, q'_1, \ldots, q'_s, q''_1, \ldots, q''_t$ mit

$$q_1 \cdots q_r q'_1 \cdots q'_s = q q''_1 \cdots q''_t.$$

Nach Voraussetzung gibt es ein $i \in \{1, \ldots, r\}$ mit $q \sim q_i$ oder ein $j \in \{1, \ldots, s\}$ mit $q \sim q_j$. D. h. $q \mid a$ oder $q \mid b$; also ist q Primelement.

□

Definition A.7 (faktorieller Ring) *Ein Integritätsring R heißt* faktorieller Ring *(UFD, Unique Factorization Domain), wenn R zusätzlich die folgende Eigenschaft hat:*
Zu jedem $a \in R$ mit $a \neq 0$ und $a \notin R^*$ gibt es Primelemente $p_1, \ldots, p_r \in R$ mit $a = p_1 \cdots p_r$. Diese Darstellung ist bis auf Permutation eindeutig.

Bemerkung: Aufgrund der obigen Definition für faktorielle Ringe gilt auch die Aussage bzgl. irreduzibler Elemente gemäß Satz A.8.

Definition A.8 (Teilerkette) *Seien $a_1, a_2, \ldots \in R$ mit R Hauptidealring. Die Folge $(a_i)_{i \geq 1}$ heißt* Teilerkette*, wenn für alle i gilt:*

$$a_i \in R\backslash\{0\} \quad \textit{und} \quad a_{i+1} \mid a_i.$$

Satz A.9 (Teilerkettensatz) *Sei R Hauptidealring und $a_1, a_2, \ldots$ Teilerkette in R. Dann wird die Folge $(a_i)_{i \geq 1}$* stationär*, d. h. es existiert ein $i_0 \in \mathbb{N}$, so dass für alle $i \geq i_0$ gilt: a_i ist assoziiert zu a_{i_0}.*

Beweis: Aufgrund von Satz A.4 liefert die Teilerkette eine Kette von Idealen $(a_1) \subset (a_2) \subset (a_3) \subset \ldots$. Somit ist $I := \bigcup_{i \geq 1}(a_i)$ Ideal in R. Da R Hauptidealring ist, existiert ein $a \in R$ mit $I = (a)$. Also existiert ein Index i_0 mit $a \in (a_{i_0})$ und somit $(a) = (a_i)$ für alle $i \geq i_0$.

□

Satz A.10 *Jeder Hauptidealring ist ein faktorieller Ring.*

Beweis:
a) Existenz der Primfaktorzerlegung: Sei $a \in R\backslash(R^* \cup \{0\})$. Da R Hauptidealring ist, fallen Primelemente und irreduzible Elemente zusammen. Falls a irreduzibel ist, ist nichts zu zeigen. Sei a also reduzibel, somit existiert eine Zerlegung $a = a_1 a_2$ mit $a_1, a_2 \in R\backslash(R^* \cup \{0\})$. Diese Faktoren sind entweder beide irreduzibel, oder man wendet das Verfahren so lange an, bis alle Faktoren irreduzibel sind. Aufgrund des Teilerkettensatzes muss dieses Verfahren nach endlich vielen Schritten zum Erfolg führen.

b) Eindeutigkeit der Primfaktorzerlegung: Annahme: Es existieren zwei Darstellungen

$$a = p_1 \cdots p_r \quad \text{und} \quad a = q_1 \cdots q_s \quad \text{mit} \quad r \geq s.$$

Induktion über r liefert dann die Behauptung.
Induktionsanfang: $r = 1 \quad \Rightarrow \quad s = 1$, also $p_1 = q_1$.
Induktionsschritt $(r - 1 \to r)$: p_r ist Primelement und $p_r \mid (q_1 \cdots q_s)$. Also teilt p_r einen der Faktoren q_j. Ggf. durch Permutation kann $p_r \mid q_s$ erreicht werden. Da q_s irreduzibel ist, sind p_r und q_s assoziiert, es existiert also ein $u \in R^*$ mit $q_s = u p_r$. Für $b := p_1 \cdots (u^{-1} p_{r-1}) = q_1 \cdots q_{s-1}$ gilt dann die Eindeutigkeit der Zerlegung nach Induktionsvoraussetzung.

□

Folgerung: $K[X]$ mit K Körper ist faktorieller Ring, da $K[X]$ nach Satz A.2 ein euklidischer Ring und somit ein Hauptidealring ist.

Bemerkung: Aufgrund der Definition des ggT und der Existenz einer bis auf Permutation eindeutigen Faktorisierung (wg. Definition A.7) ist ein ggT zweier Elemente ein gemeinsamer Faktor der Elemente, welcher von der größtmöglichen Anzahl von Primelementen geteilt wird. Der ggT ist bis auf Einheiten und Permutation der Primelemente eindeutig.

A.2 Argument von Cassels

Zusätzlich zum im Kapitel 5.1 eingeführten Begriff des *Gitters* benötigt man zunächst noch den Begriff des *Untergitters*. Die Definition folgt dabei der intuitiven Vorstellung.

Definition A.9 *Seien L, M Gitter in $\mathbb{R}^n$. L heißt* Untergitter *von M, falls jeder Gitterpunkt von L auch Gitterpunkt von M ist.*

Sei also L Untergitter von M und sei $a_1, \ldots, a_n$ bzw. $b_1, \ldots, b_n$ Basis von L bzw. M, dann existieren ganze Zahlen ν_{ij} mit

$$a_i = \sum_{j=1}^{n} \nu_{ij} b_j, \quad \text{da} \quad a_i \in M \quad \text{gilt.} \tag{A.1}$$

Wie bereits bei Einführung des Determinantenbegriffs im Zusammenhang mit Gittern gesehen, ist der Wert der Determinante eines Gitters unabhängig von der Wahl der Basis, und damit folgt unmittelbar, dass die Zahl

$$D = |\det(\nu_{ij})| = \frac{\mathrm{d}(L)}{\mathrm{d}(M)} \tag{A.2}$$

nur von der Wahl der Gitter L und M abhängt. Da $a_1, \ldots, a_n$ und $b_1, \ldots, b_n$ Gitterbasen sind, gilt $D > 0$. D heißt der *Index von L in M*. Löst man also die Gleichungen aus A.1 nach b_i auf und verwendet die Beziehung aus A.2 so erhält man

$$Db_i = \sum_{j=1}^{n} \omega_{ij} a_j \quad \text{mit} \quad \omega_{ij} \in \mathbb{Z},$$

oder anders ausgedrückt

$$DM \subset L \subset M \quad \text{wobei} \quad DM = \{Db \mid b \in M\}.$$

Satz A.11 *Sei L Untergitter von M. Zu jeder Basis $\mathcal{B} = (b_1, \ldots, b_n)$ von M kann man eine Basis $\mathcal{A} = (a_1, \ldots, a_n)$ von L der Gestalt finden, dass*

$$\begin{array}{lcllll} a_1 & = & \nu_{11} b_1 & & & \\ a_2 & = & \nu_{21} b_1 & + & \nu_{22} b_2 & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ a_n & = & \nu_{n1} b_1 & + & \ldots \quad + & \nu_{nn} b_n \end{array}$$

mit $\nu_{ij} \in \mathbb{Z}$ erfüllt ist.

Beweis: Da $Db_i \in L$ (s. o.) für alle i mit $1 \leq i \leq n$ gilt, existieren jedenfalls Punkte $a_i \in L$ mit

$$a_i = \nu_{11} b_1 + \cdots + \nu_{ii} b_i \quad \text{mit} \quad \nu_{ii} \neq 0.$$

Man wähle die a_i nun so, dass $|\nu_{ii}| \neq 0$ und *minimal* ist. Bleibt zu zeigen, dass $(a_1, \ldots, a_n)$ eine Basis von L ist.
Da $a_i \in L$ nach Konstruktion gilt, folgt für alle $\omega_i \in \mathbb{Z}$:

$$\omega_1 a_1 + \cdots + \omega_n a_n \in L. \tag{A.3}$$

Annahme: $c \in L$ und c habe nicht die Form aus A.3.
c kann aber wegen $c \in M$ sicherlich als

$$c = t_1 b_1 + \cdots + t_k b_k \quad \text{mit} \quad t_k \neq 0, t_k \in \mathbb{Z} \text{ und } 1 \leq k \leq n$$

dargestellt werden. Existieren mehrere solcher c, so wähle man das, für welches k kleinstmöglich ist. Da $\nu_{kk} \neq 0$ ist, kann man ein $s \in \mathbb{Z}$ so wählen, dass

$$|t_k - s\nu_{kk}| < |\nu_{kk}| \tag{A.4}$$

gilt. Damit folgt

$$c - sa_k = (t_1 - s\nu_{11}) b_1 + \cdots + (t_k - s\nu_{kk}) b_k \in L, \quad \text{da } c, a_k \in L,$$

und das Element hat nicht die Darstellung aus A.3, da c nach Annahme keine solche Darstellung besitzt; also gilt $t_k - \nu_{kk} \neq 0$, da k kleinstmöglich gewählt war. Damit widerspricht A.4 der Annahme, dass die ν_{kk} kleinstmöglich gewählt waren. D. h. es existiert kein c, welches sich nicht in der Form A.3 darstellen läßt.

□

Weitere Ausführungen zu diesem Thema finden sich bei [CAS]. Dort wird u. a. gezeigt, dass es unter den Voraussetzungen des obigen Satzes zu jeder Basis $\mathcal{A} = (a_1, \ldots, a_n)$ auch eine Basis $\mathcal{B}$ von M gibt, die das Gleichungssystem des Satzes erfüllt.

Bemerkung: Die im Beweis zu Satz 5.9 benötigte Aussage bezüglich Polynomen in $\mathbb{Z}[X]$ erhält man, indem man dort das Gitter

$$\Omega = \mathbb{Z} + \mathbb{Z}X + \cdots + \mathbb{Z}X^{n+m'-e-1}$$

wählt; dadurch wird L' zum Untergitter von Ω, und Satz A.11 liefert die benötigte Aussage.

A.3 Mignotte-Schranke für Polynome

Im Nachfolgenden werden zunächst Aussagen über Polynom mit komplexen Koeffizienten gemacht. Sei also

$$f = \sum_{i=0}^{n} a_i X^i \in \mathbb{C}[X]$$

mit $\deg(f) = n$ und $\|f\|$ die *Euklidische Norm* mit

$$\|f\| = \sqrt{\sum_{i=0}^{n} |a_i|^2}.$$

Lemma A.1 *Für $f \in \mathbb{C}[X]$ und $\alpha \in \mathbb{C}$ gilt*

$$\|(X-\alpha)f\| = \|(\overline{\alpha}X - 1)f\|.$$

Beweis:

$$\begin{aligned}
\|(X-\alpha)f\|^2 &= \|a_n X^{n+1} + \sum_{i=1}^{n}(a_{i-1} - \alpha a_i)X^i - \alpha a_0\|^2 \\
&= \|\sum_{i=0}^{n+1}(a_{i-1} - \alpha a_i)X^i\|^2 \quad \text{mit} \quad a_{-1} = a_{n+1} = 0 \\
&= \sum_{i=0}^{n+1}(a_{i-1}\overline{a_{i-1}} - \overline{\alpha}a_{i-1}\overline{a_i} - \alpha\overline{a_{i-1}}a_i + \alpha\overline{\alpha}a_i\overline{a_i}) \\
&= \sum_{i=0}^{n+1} a_{i-1}\overline{a_{i-1}} - \sum_{i=0}^{n+1}(\overline{\alpha}a_{i-1}\overline{a_i} + \alpha\overline{a_{i-1}}a_i) + \sum_{i=0}^{n+1}\alpha\overline{\alpha}a_i\overline{a_i} \\
&= \sum_{i=0}^{n+1} a_i\overline{a_i} - \sum_{i=0}^{n+1}(\alpha\overline{a_{i-1}}a_i + \overline{\alpha}a_{i-1}\overline{a_i}) + \sum_{i=0}^{n+1}\alpha\overline{\alpha}a_{i-1}\overline{a_{i-1}} \\
&= \sum_{i=0}^{n+1}(\alpha\overline{\alpha}a_{i-1}\overline{a_{i-1}} - \alpha\overline{a_{i-1}}a_i - \overline{\alpha}a_{i-1}\overline{a_i} + a_i\overline{a_i}) \\
&= \|\sum_{i=0}^{n+1}(\overline{\alpha}a_{i-1} - a_i)X^i\|^2 = \|(\overline{\alpha}X - 1)f\|^2
\end{aligned}$$

□

Definition A.10 *Sei $f \in \mathbb{C}[X]$ mit $\deg(f) = m$, Nullstellen $\alpha_1, \ldots, \alpha_m$ und Höchstkoeffizient a_m, dann ist das Maß von f definiert als*

$$M(f) = |a_m| \cdot \prod_{i=1}^{m} \max\{1, |\alpha_i|\}.$$

Lemma A.2 (Landau-Ungleichung) *Für $f \in \mathbb{C}[X]$ mit $\deg(f) = m$ gilt*

$$M(f) \leq \|f\|$$

Beweis: $\alpha_1, \ldots, \alpha_k$ seien die Nullstellen mit $|\alpha_i| > 1$. Dann ist

$$\mathrm{M}(f) = |a_m| \cdot |\alpha_1 \cdot \ldots \cdot \alpha_k|.$$

Betrachtet man nun das Polynom

$$g = a_m \cdot \prod_{i=1}^{k} (\overline{\alpha_i} X - 1) \cdot \prod_{i=k+1}^{m} (X - \alpha_i) = \sum_{j=0}^{m} b_j X^j, \tag{A.5}$$

dann folgt nach k-facher Anwendung von Lemma A.1 $\|g\| = \|f\|$, und wegen A.5 folgt außerdem $\mathrm{M}(g) = |b_m|$ mit $b_m = a_m(\overline{\alpha_1} \cdot \ldots \cdot \overline{\alpha_k})$, also

$$\mathrm{M}(f)^2 = \mathrm{M}(g)^2 = |b_m|^2 \leq \|g\|^2 = \|f\|^2.$$

□

Mit den bisherigen Ausführungen fällt es nun leicht, den Satz über die sog. *Mignotte-Schranke* zu beweisen. Ursprünglich handelt es sich um ein Theorem aus dem Artikel *An Inequality about Factors of Polynomials* von *Maurice Mignotte* [MI2]. Die hier verwendete Version folgt in der Herleitung einem Vorlesungsskript [DEC] und lautet:

Satz A.12 (Mignotte-Schranke) *Seien $f, h \in \mathbb{C}[X]$, $h \mid f$ mit $\deg(h) \leq m \leq n = \deg(f)$ und bezeichnen $\mathrm{lc}(f)$ bzw. $\mathrm{lc}(h)$ die Höchstkoeffizienten von f bzw. h, dann gilt*

$$\|h\| \leq \binom{2m}{m}^{\frac{1}{2}} \left| \frac{\mathrm{lc}(h)}{\mathrm{lc}(f)} \right| \|f\|.$$

Beweis: Sei h gegeben durch

$$h = \sum_{i=0}^{m} h_i X^i \quad \text{bzw. geschrieben als} \quad h = h_m \cdot \prod_{i=1}^{m} (X - a_i).$$

Nach dem *Wurzelsatz von Vieta* gilt für den Zusammenhang der Nullstellen von h mit den Koeffizienten h_i die folgenden Beziehung:

$$h_i = (-1)^{m-i} \cdot h_m \cdot \sum_{\substack{S \subset \{1,\ldots,m\} \\ \mathrm{card}(S) = m-i}} \prod_{j \in S} a_j.$$

Damit folgt

$$|h_i| \leq |h_m| \sum_S \prod_{j \in S} |a_j| \leq \binom{m}{i} \mathrm{M}(h). \tag{A.6}$$

Mittels eines kleinen Ausflugs in die Kombinatorik erhält man die Gültigkeit von

$$\sum_{i=0}^{m} \binom{m}{i}^2 = \binom{2m}{m}.$$

Dabei gibt die rechte Seite der Gleichung direkt die Anzahl aller m-elementigen Teilmengen einer $2m$-elementigen Gesamtmenge an. Dies ist in offensichtlicher Weise das Gleiche, wie aus zwei disjunkten m-elementigen Mengen die Anzahl der möglichen m-elementigen Teilmengen zu bestimmen. Also hat man:

$$\begin{aligned} \binom{2m}{m} &= \binom{m}{0}\binom{m}{m} + \binom{m}{1}\binom{m}{m-1} + \cdots + \binom{m}{m}\binom{m}{0} \\ &= \binom{m}{0}^2 + \binom{m}{1}^2 + \cdots + \binom{m}{m}^2. \end{aligned}$$

Beachtet man zusätzlich, dass mit der Definition von $\mathrm{M}(f)$ im Falle von $h \mid f$

$$\frac{\mathrm{M}(h)}{|\mathrm{lc}(h)|} \leq \frac{\mathrm{M}(f)}{|\mathrm{lc}(f)|}$$

gilt, so erhält man schließlich

$$\begin{aligned} \|h\|^2 = \sum_{i=0}^{m} |h_i|^2 &\leq \sum_{i=0}^{m} \binom{m}{i}^2 \mathrm{M}^2(h) = \binom{2m}{m} \mathrm{M}^2(h) \\ &\leq \underbrace{\binom{2m}{m} \left|\frac{h_m}{\mathrm{lc}(f)}\right|^2 \mathrm{M}^2(f)}_{h|f \text{ und Definition von M}} = \binom{2m}{m} \left|\frac{\mathrm{lc}(h)}{\mathrm{lc}(f)}\right|^2 \mathrm{M}^2(f), \\ &\leq \binom{2m}{m} \left|\frac{\mathrm{lc}(h)}{\mathrm{lc}(f)}\right|^2 \|f\|^2 \quad \text{wg. Landau-Ungleichung.} \end{aligned}$$

□

Korollar A.2 (Mignotte-Schranke) *Falls $h, f \in \mathbb{Z}[X]$, dann gilt sogar*

$$\|h\| \leq \binom{2m}{m}^{\frac{1}{2}} \|f\|.$$

Beweis: klar, wg. $|\mathrm{lc}(f)| \geq 1$ und $\mathrm{lc}(h) \mid \mathrm{lc}(f)$

□

Bemerkung: Im Falle von ganzzahligen Polynomen f, h mit $h \mid f$ liefert das Korollar aufgrund von $|h_i| \leq \|h\|$ eine Abschätzung für die Koeffizienten des Faktors. Zudem ist damit die Zahl der als Faktoren in Frage kommenden Polynome eine endliche Menge. Damit ist unmittelbar ein effektives Faktorisierungsverfahren gegeben, welches aber aufgrund der *Trial-And-Error*-Methode ebenso ineffizient ist.

Die in der Bemerkung angeführte Abschätzung kann aufgrund von Ungleichung A.6 natürlich noch verfeinert werden.

Korollar A.3 *Seien $f, h \in \mathbb{Z}[X]$ mit h wie im Beweis zu Satz A.12 und $h \mid f$. Dann gilt für die Koeffizienten von h die Abschätzung:*

$$|h_i| \leq \binom{m}{i} M(h) \leq \binom{m}{i} \|f\|.$$

Beweis: klar, wegen Ungleichung A.6, $\mathrm{M}(h) \leq \mathrm{M}(f)$ für ganzzahlige Polynome und Anwendung der *Landau-Ungleichung*

□

Beispiel: Sei $f, h \in \mathbb{Z}[X]$ mit

$$f = x^6+x^5+6x^4-5x^3+3x^2+2 \quad \text{und angenommen} \quad h \mid f \quad \text{mit} \quad \deg(h) \leq 3.$$

Dann ist aufgrund der Eigenschaften des Binomialkoeffizienten und mit Anwendung des Korollars A.3 eine Abschätzung für die Koeffizienten eines möglichen Faktors gegeben durch

$$|h_i| \leq \binom{3}{1} \|f\| = 3\sqrt{76} \approx 26,2.$$

Weitere Untersuchungen von f zeigen, dass dieses irreduzibel ist.

B Hilfsalgorithmen

Die nachfolgenden, in kurzer Form dargestellten, Algorithmen werden im Rahmen der beschriebenen Faktorisierungsalgorithmen verwendet. Da es sich hierbei in aller Regel um wohlbekannte Algorithmen handelt, erfolgt nur eine Darstellung der Funktionsweise inklusive Angabe des zugehörigen *Nassi-Schneidermann-Diagramms* . Der Sourcecode zu den hier beschriebenen Algorithmen befindet sich weiter hinten im Anhang.

B.1 Triangularisierung - Bestimmung des Nullraums

Gegeben sei eine $n \times n$ Matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$ mit $a_{i,j} \in \mathbf{K}$, wobei $\mathbf{K}$ irgendein Körper ist. Der Algorithmus liefert die maximale Zahl der linear unabhängige Vektoren $v_1, \ldots, v_r$, die in den ersten r Zeilen der $n \times n$ Matrix $BNull$ gespeichert sind und die Bedingung $v_1 A = v_2 A = \ldots = v_r A = 0$ erfüllen. Der Rang der Matrix A ist somit $n - r$. Die restlichen Zeilen von $BNull$ enthalten nur Nullen. Als *Nassi-Schneidermann-Diagramm* ergibt sich folgende Darstellung:

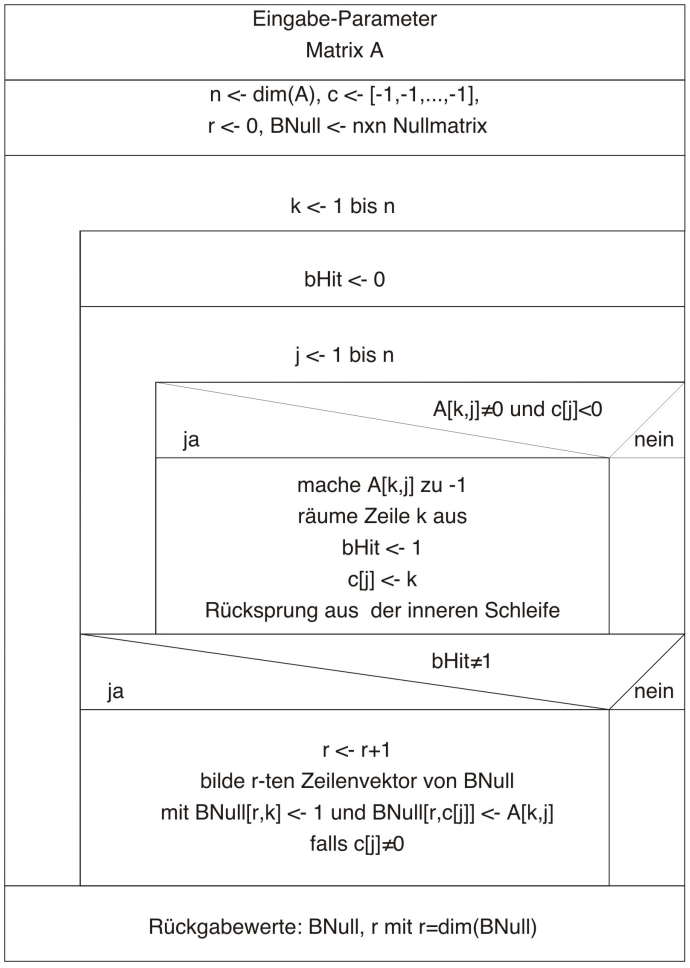

Diagramm 26: Nullspace-Funktion
Sourcen-Code siehe *nullspace.par*, Anhang S.122

Aus dem obigen Diagramm ist unmittelbar einsichtig, daß die Laufzeit des Algorithmus von der Ordnung $\mathcal{O}(n^3)$ ist.

B.2 Behandlung von Polynomen

Alle Routinen zur Behandlung von Polynomen, welche über die Grundrechenarten hinausgehen, werden an dieser Stelle aufgeführt und schaffen so die Grundlage für eine leichte Realisierung in anderen Programmiersprachen.

B.2.1 Funktion zur euklidischen Division (*euklid_division*)

Die *euklid_division*-Funktion erledigt die in Algorithmus 2.1 (S. 9) euklidische Division zweier Polynome u, v mit $\deg(u) \geq \deg(v)$ unter Zuhilfenahme des Pseudo-Divisions-Algorithmus 2.2 (S. 10).

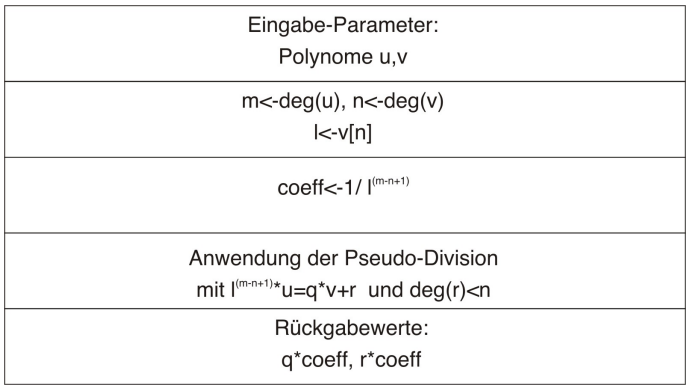

Diagramm 27: euklid_division-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.2 Vecnorm-Funktion (*vecnorm*)

Euklidische Norm eines Vektors v:

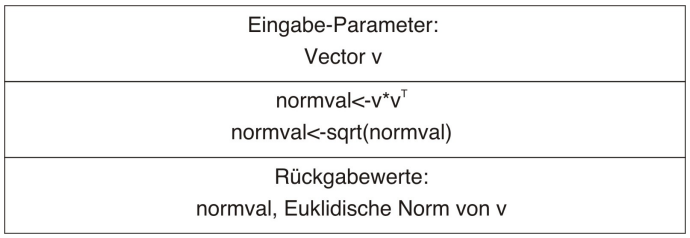

Diagramm 28: vecnorm-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.3 Content-Funktion (*cont*)

Die *Content*-Funktion dient der Berechnung des ggT der Koeffizienten eines Polynoms und findet bei der Bestimmung des primitiven Anteils eines Polynoms Verwendung. Für ein Polynom f über einem faktoriellen Ring gilt

$$f = \mathrm{cont}(f)\mathrm{pp}(f). \tag{B.1}$$

Damit ergibt sich der folgende Algorithmus:

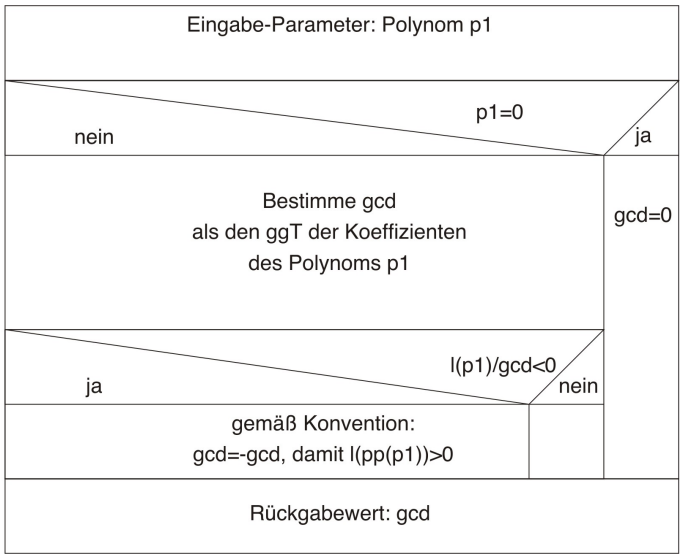

Diagramm 29: cont-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.4 Primitive-Part-Funktion (*pp*)

Die *pp*-Funktion bestimmt mittels der obigen Beziehung (B.1) den primitiven Anteil eines Polynoms f.

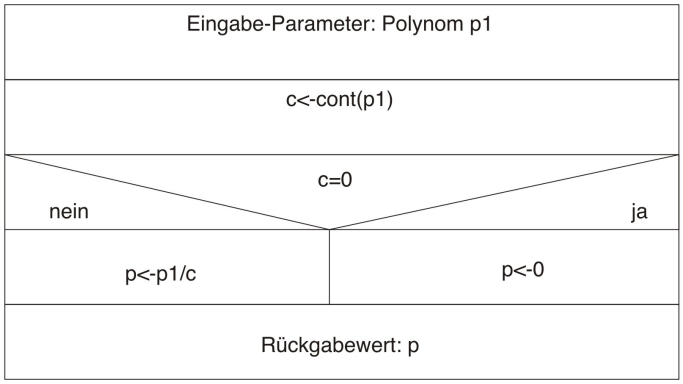

Diagramm 30: pp-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.5 PolCoeffMod-Funktion (*PolCoeffMod*)

PolCoeffMod dient der Umwandlung eines Polynoms $f \in \mathbb{Z}[X]$ in ein Polynom $f^* \in \mathbb{F}_p[X]$.

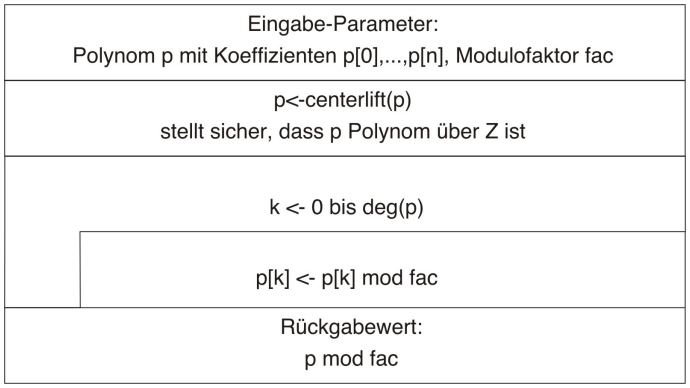

Diagramm 31: PolCoeffMod-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.6 PolRandom-Funktion (*PolRandom*)

Die *PolRandom*-Funktion generiert ein Zufallspolynom f über $\mathbb{Z}$ oder $\mathbb{F}_p$ vom Maximalgrad n mit optionaler Bedingung an den Höchstkoeffizienten.

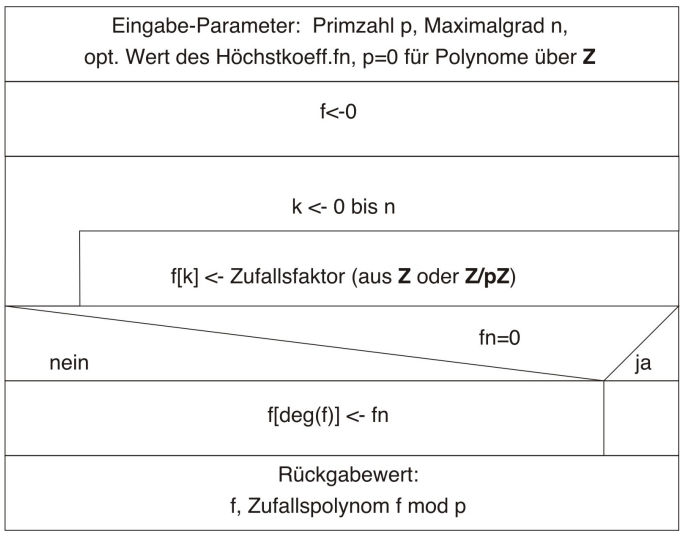

Diagramm 32: PolRandom-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.7 Derivative-Funktion (*derivative*)

Die formale Ableitung f' eines Polynoms f wird mittels der *Derivative*-Funktion bestimmt

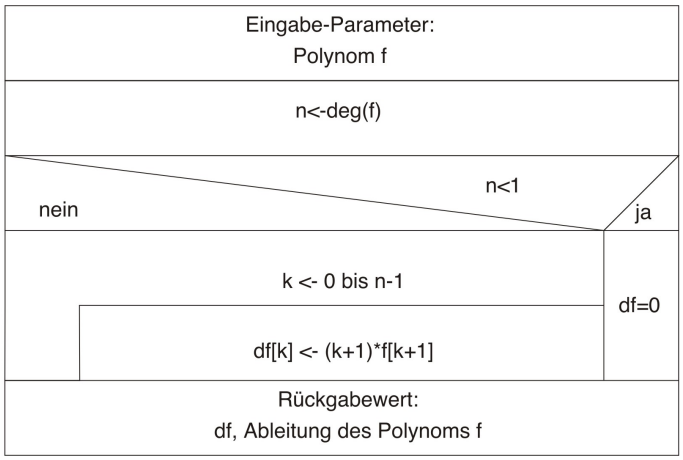

Diagramm 33: Derivative-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

B.2.8 Polnorm-Funktion (*polnorm*)

Die *polnorm*-Funktion dient der Bestimmung der euklidischen Norm eines Polynoms f.

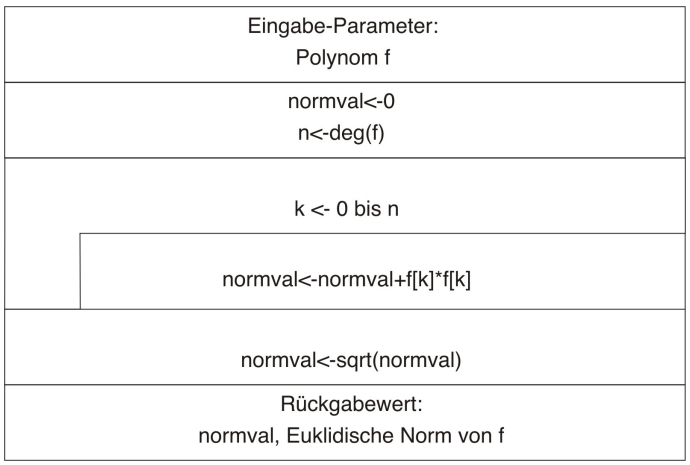

Diagramm 34: polnorm-Funktion
Sourcen-Code siehe *polynomutilities.par*, Anhang S.117

C Multipräzisions-Software-Pakete

Die Notwendigkeit für Multipräzisions-Software-Pakete resultiert einerseits aus der Schwäche heutiger Programmiersprachen bereits standardmäßig bestimmte Datentype darzustellen, Ganzzahlrechnungen nicht mit mehr als 64 Bit ausführen zu können und andererseits aus dem Verlangen nach einer mehr auf den zu betrachtenden Sachverhalt abgestimmten Benutzerschnittstelle.

Die Geschichte der Algebra-Software-Pakete geht zurück bis in die frühen 70er Jahre des vergangenen Jahrhundert. Hier sei nur ein Paket – *Macsyma* – genannt, welches sich auch heute noch in Projekten wiederfindet. *Macsyma* (MAC's SYmbolic MAnipulation System) wurde ursprünglich vom M. I. T. entwickelt und dann später kommerziell vom ESTSC (Energy Science & Technology Software Center) vertrieben. Eine mittlerweile unter *GNU Public License* stehende Variante (*MAXIMA*) kann unter http://maxima.sourceforge.net/ heruntergeladen werden. Aufgrund des Alters und der Entstehungsgeschichte handelt es sich um ein sehr ausgereiftes Paket, dessen Entwicklung allerdings nur noch sehr langsam voranschreitet und dessen Bekanntheitsgrad nur noch sehr gering ist.

In den 80er Jahren kamen mit dem Aufkommen der Personal Computer auch zunehmend Software-Pakete mit rein kommerziellem Charakter auf den Markt. In der Anfangsphase waren diese sicher weniger getestet als die in Forschungseinrichtungen entstandenen Vorgänger, wurden dafür aber um so aggressiver beworben. Aber auch diese Software blickt heute auf eine fast zwanzigjährige Vergangenheit zurück und hat sich aufgrund ihrer Leistungsfähigkeit und der großen Bekanntheit zu einem Quasistandard entwickelt. Die Rede ist hier von *Mathematica* aus dem Hause *Wolfram Research, Inc.* und von *Maple* von der *Universität Waterloo*.

Neben den bereits genannten Paketen existieren natürlich auch Resultate aus Forschungsprojekten an Universitäten, welche in aller Regel kostenlos weitergegeben werden. Da einer der Schwerpunkte der vorliegenden Arbeit in der vollständigen Realisierung der zu untersuchenden Algorithmen liegt und diese auch leicht und schnell für den Leser implementierbar sein sollen, kann die Wahl nur auf ein Software-Paket fallen, welches frei verfügbar und auf vielen Plattformen existent ist. Ein hoher Verbreitungs- und Bekanntheitsgrad ist ebenso wünschenswert.

Ein guter und einfach zu installierender Multipräzisions-Interpreter, welcher als Beigabe zum Buch von O. Forster [FOR] kommt, ist *Aribas*. Dieser schnelle, kompakte Interpreter lehnt sich an die *Pascal-Syntax* an und kommt somit dem geübten Programmierer sehr entgegen, bietet aber nur die wesentlichen

Multipräzisions-Routinen, ohne bereits spezielle Datentypen zu bieten.

Auch das *UBasic*-Paket ist frei verfügbar; wie der Name schon verrät, handelt es sich hier um ein BASIC-Derivat mit Multipräzisions-Eigenschaften. Viele Funktionen aus der algorithmischen Zahlentheorie sind in diesem rein in Assembler geschriebenen und daher sehr schnellen Interpreter bereits implementiert. Das Programm wird allerdings seit 2000 nicht mehr weiterentwickelt.

Das Software-Paket der Wahl, *PARI/GP*, vereint alle oben geforderten Eigenschaften, liegt zusätzlich auch noch im Quellcode vor, und ermöglich das Einbinden von Multipräzisions-Bibliotheken in eigene *C/C++*-Projekte. Es wurde im wesentlichen von H. Cohen (Autor von [COH]) entwickelt. Annähernd alle in seinem Buch behandelten Algorithmen sind hier bereits implementiert. Die Programmiersprache weist eine hochsprachenähnliche Syntax auf, die zwar bei einigen Kontrollstrukturen gewöhnungsbedürftig, aber nichtsdestotrotz schnell zu erlernen ist. Aufgrund der hervorragenden Dokumentation und wegen des ausführlichen Tutorials kann auch der nicht geübte Programmierer einen schnellen Zugang finden. Außerdem bietet das Paket bereits standardmäßig eine Vielzahl der zu untersuchenden Algorithmen an und ermöglicht so eine verfeinfachte Überprüfung der eigenen Algorithmen.

Abschließend bleibt festzuhalten, dass unabhängig von den weiter unten angegebenen Beschreibungen/Beurteilungen alle u. g. Softwarepakete von der software-technischen Seite der Aufgabenstellung dieser Diplomarbeit gewachsen gewesen wären. Eine Portierung von einer Interpreter-Sprache zur anderen ist hier mehr eine Frage des Fleißes.

C.1 Bezugsquellen

Obwohl die Angabe von Internet-Adressen innerhalb gedruckter Literatur aufgrund des im ständigen Wandel befindlichen Mediums Internet meist nur wenig sinnvoll erscheint, werden hier die Bezugsquellen für die o. g. Software-Pakete genannt. Auch wenn diese Links schon nach kurzer Zeit nicht mehr aktuell sein sollten, so können diese doch meist noch als gute Ausgangspunkte für eine Suche via http://www.google.de verwendet werden.

Aribas	http://www.mathematik.uni-muenchen.de/~forster/
Maple	http://www.maplesoft.com/
Mathematic	http://www.wolfram.com/
MAXIMA	http://maxima.sourceforge.net/
MuPAD	http://www.mupad.de/schule+studium/
PARI/GP	http://pari.math.u–bordeaux.fr/
UBasic	http://www.rkmath.rikkyo.ac.jp/~kida/ubasic.htm

D PARI/GP

Um einen möglichst schnellen Einstieg in *PARI/GP* zu bieten, werden im Nachfolgenden eine Installation auf einem Standard-UNIX- bzw. auf einem *Windows*-System beschrieben. Abschließend werden zwei kleine Beispiele behandelt.

D.1 Unix-System

Nach dem Download der aktuellen Sourcen unter

http://www.gn-50uma.de/ftp/pari-2.1/pari-2.1.5.tar.gz

empfiehlt es sich, das Archiv an der gewünschter Stelle im System zu entpacken und den Installationshinweisen in der Datei *INSTALL.txt* zu folgen. Hier wird in wenigen Schritten eine komplette Installation auf einem Standard-System beschrieben. Sollen weitere Einstellungen vor der Übersetzung vorgenommen oder Pfade verändert werden, so sei dringend die Lektüre des sechsseitigen *Installation Guide for the UNIX Versions* angeraten.

http://www.gn-50uma.de/ftp/pari-2.1/manuals/INSTALL.pdf

D.2 Windows-System

Im Falle eines *Microsoft* Betriebssystems kommt sinnvollerweise nur der Download von sog. Executables, d. h. bereits übersetzten, lauffähigen Programmdateien, in Frage. Diese finden sich unter:

http://www.gn-50uma.de/ftp/pari/00index.html#exe

In Abhängigkeit von der Version des verwendeten Betriebssystem sollte nun nach den einfachen Instruktionen in der entsprechenden *README*-Datei verfahren werden

D.3 Start und Test

Eine erfolgreiche Installation vorausgesetzt kann *PARI/GP* nun zum ersten Mal gestartet werden. Bei Unix-Systemen erfolgt der Programmstart durch den Aufruf `gp` innerhalb einer *Shell*, bei *Windows* durch einen Doppelclick auf die Programmdatei. Anschließend sollte eine Befehlseingabezeile der Form `gp>` erscheinen.

D.3.1 Beispiel 1 - einfache Funktion

Durch die Eingabe von

```
gp > first_prime_div(x)=forprime(p=2,x,if(x%p==0,return(p)));
gp>
```

wird die Funktion *first_prime_div(x)* definiert; sie bestimmt iterativ den ersten Primteiler einer gegebenen Zahl x. Ein Aufruf erfolgt beispielsweise durch:

```
gp > first_prime_div(91)
%4 = 7
gp >
```

D.3.2 Beispiel 2 - Funktionen mittels Dateien

Funktionen wie die aus dem vorangehenden Beispiel können selbstverständlich auch innerhalb einer Datei definiert werden und dann mittels *PARI/GP* geladen werden. Um die Funktion *first_prime_div(x)* so umzusetzen, erzeuge man hierzu zunächst beispielsweise die Datei *first_prime.par* mit folgendem Inhalt:

```
\\ Funktion: first_prime_div
first_prime_div(x)=
{
  local(p);
  forprime(p=2,x,
           if(x%p==0,
              return(p);
             );
          );
}
```

Innerhalb von *PARI/GP* führt nun der Aufruf

```
gp > read("first_prime.par")
gp>
```

dazu, daß die Funktion dem System zur Verfügung gestellt wird und, wie oben bereits gesehen, aufgerufen werden kann.

D.3.3 Anmerkung

Eine komplette Übersicht über die in *PARI/GP* zur Verfügung stehenden Befehle und Funktionen liefert die *PARI/GP Reference Card*:

http://www.gn-50uma.de/ftp/pari-2.1/manuals/refcard.pdf.

Zum Softwarepaket gehört weiterhin ein umfangreiches Tutorial[PARI-Tut.], welches ausführlich alle Bereiche der Mathematik abdeckt, die mit *PARI/GP* behandelt werden können.

E Sourcecode

Die nachfolgenden Seiten führen alle im Rahmen dieser Diplomarbeit entwickelten Algorithmen in Form vollständiger *PARI/GP*-Programme auf. Wie bereits in Anhang C beschrieben, handelt es sich bei diesem Softwarepaket um eine sehr vielseitige Multipräzisionssoftware, die bereits eine hohe Grundfunktionalität bietet; die vorliegenden Sourcen machen allerdings im Wesentlichen nur von den Multipräzisionsdatentypen, der Polynomdarstellung sowie der einfachen Handhabung von Element aus $\mathbb{Z}/p^k\mathbb{Z}$ bzw. $(\mathbb{Z}/p^k\mathbb{Z})[X]$ mit $k \geq 1$ Gebrauch.
Die meisten Routinen wurden zwecks der gewünschten Gesamtübersicht und zu Gunsten einer einfachen Portierbarkeit in speziellen Implementierungen ausgeführt. Die vorausgesetzten mathematischen Algorithmen und Datentypen reduzieren sich daher auf ein Minimum:

- *Multipräzisionsdatentypen* mit den vier Grundrechenarten, sowie der Verwaltung von ein- und zweidimensionalen Feldern
- *einfache Vektor- und Matrizenoperationen*
- *Repräsentierung für Polynome*, welche natürlich auch durch einfache Vektoren erfolgen kann
- *Unterstützung der Modul-Funktion*

E.1 Verwendungen der Algorithmen

Nach dem Laden des sog. *Projectloaders* (s. Anhang E.2.6, S. 113) stehen sämtliche in dieser Arbeit implementierten Funktionen innerhalb der *PARI/GP*-Umgebung zur Verfügung.
Zahlreiche ausgezeichnete Routinen haben zusätzlich zum eigentlichen Algorithmus, der in der jeweiligen Quelldatei meist auf *_calc* endet, auch noch eine umschließende Routine, welche die Fehlerbehandlung und Parameterüberprüfung übernimmt. Zusätzlich steht zu diesen Funktionen eine *PARI/GP*-Hilfe zur Verfügung, welche nach der üblichen Konvention `? Funktionsname` aufgerufen wird. Die Hilfe zum linearen Hensel Lift erhält man also durch Eingabe von

```
gp>  ? lin_lift
```

nach dem Befehlsprompt.

E.2 Gesamtübersicht der Algorithmen

E.2.1 Alle Algortihmen auf eine Blick

Die nächsten drei Seiten liefern einen tabellarischen, alphabetisch sortierten Überblick über die implementierten Routinen.

Name	Parameter	Funktion	Resultat	Theorie	Sourcen
aux1	f,h,m,p,k	Hilfsalgorithmus 1 zum LLL-Algorihtmus	[bH0poss, h0]	S. 78	`auxlll.par`, S. 153
aux2	f,h,p	Hilfsalgorithmus 2 zum LLL-Algorihtmus	h0	S. 80	`auxlll.par`, S. 153
basisreduction	A	Bestimmung einer reduz. Basis B – Hauptroutine	B	S. 69	`basisreduction.par`, S. 150
basisreduction_calc	A	Bestimmung einer reduz. Basis B – ohne Fehlerbehandlung	B	S. 69	`basisreduction.par`, S. 150
cont	p1	ggT der Koeff. von $p1$	gdc_k	S. 101	`polynomutilities.par`, S. 117
derivative	f	Ableitung von f	dfv	S. 103	`polynomutilities.par`, S. 117
errorHandling	Error	standardisierte Fehlerausgabe	–	–	`polynomutilities.par`, S. 117
euklid_division	u,v	$u = qv + r$	[q,r]	S. 9	`polynomutilities.par`, S. 117
ext_gcd	a,b	$ua + vb = d$	[u,v,d]	S. 16	`polynomutilities.par`, S. 117
factor_berlekampsf	u,p	Faktorisierung eines quadratfreien Polynoms $f \mod p$ – Hauptroutine	[l,n,Faktoren]	S. 45	`berlekamp.par`, S. 134
factor_berlekampsf_calc	u,n,p	Faktorisierung eines quadratfreien Polynoms $f \mod p$ – ohne Fehlerbehandlung	[l,n,Faktoren]	S. 45	`berlekamp.par`, S. 134
factor_cantor_calc_p	f,p,d	Finale Faktorisierung $p \geq 3$	[l,n,Faktoren]	S. 36	`factor_czmodp.par`, S. 138
factor_cantor_calc_p2	f,p,d	Finale Faktorisierung $p = 2$	[l,n,Faktoren]	S. 38	`factor_czmodp.par`, S. 138
factor_cantorp_calc	f,p,d	Kombination der o. g. finalen Faktorisierungen	[l,n,Faktoren]	–	`factor_czmodp.par`, S. 138

Name	Parameter	Funktion	Resultat	Theorie	Sourcen
factor_cz	f	Faktorisierung eines Polynoms f – Hauptroutine	[l,n,Faktoren]	S. 60	`factor_cz.par`, S. 144
factor_cz_calc	f	Faktorisierung eines Polynoms f – ohne Fehlerbehandlung	[l,n,Faktoren]	S. 60	`factor_cz.par`, S. 144
factor_czmodp	f,p	Faktorisierung eines Polynoms f mod p – Hauptroutine	[l,n,Faktoren]	S. 39	`factor_czmodp.par`, S. 138
factor_czmodp_calc	f,p	Faktorisierung eines Polynoms f mod p – ohne Fehlerbehandlung	[l,n,Faktoren]	S. 39	`factor_czmodp.par`, S. 138
factor_czsf_calc	f,p,d	Faktorisierung eines quadratfreien Polynoms f	Faktoren	S. 60	`factor_cz.par`, S. 144
factor_distinct	f,p	gleichgradige Faktorisierung – Hauptroutine	[l,n,Faktoren]	S. 33	`distinct_fac.par`, S. 132
factor_distinct_calc	f,p	gleichgradige Faktorisierung – ohne Fehlerbehandlung	[l,n,Faktoren]	S. 33	`distinct_fac.par`, S. 132
factor_lll	f	Faktorisierung eines Polynoms f – Hauptroutine	[l,n,Faktoren]	S. 83	`factor_lll.par`, S. 155
factor_lll_calc	f	Faktorisierung eines Polynoms f – ohne Fehlerbehandlung	[l,n,Faktoren]	S. 83	`factor_lll.par`, S. 155
factor_lllsf_calc	f,n, resultant	Faktorisierung eines quadratfreien Polynoms f	[l,n,Faktoren]	S. 82	`factor_lll.par`, S. 155
factor_squarefree	f,p	quadratfreie Faktorisierung – Hauptroutine	[l,n,Faktoren]	S. 31	`squarefree_fac.par`, S. 130
factor_squarefree_calc	f,p	quadratfreie Faktorisierung – ohne Fehlerbehandlung	[l,n,Faktoren]	S. 31	`squarefree_fac.par`, S. 130

Name	Parameter	Funktion	Resultat	Theorie	Sourcen
lin_lift	f,g1,h1,p,k	linearer Lift für zwei Faktoren – Hauptroutine	[gk,hk]	S. 49	`hensel.par`, S. 127
lin_lift_calc	f,g1,h1,p,k	linearer Lift für zwei Faktoren – ohne Fehlerbehandlung	[gk,hk]	S. 49	`hensel.par`, S. 127
MReduction	k,l,m,b	Hilfsroutine zur Basisreduktion	[m,b]	S. 69	`basisreduction.par`, S. 150
multi_lin_lift	f,facs,p,k	linearer Lift für multiple Faktoren	Faktoren	S. 51	`hensel.par`, S. 127
nullspace	A	Basis des Nullraums von A – Hauptroutine	[BNull,r]	S. 99	`nullspace.par`, S. 122
nullspace_calc	A	Basis des Nullraums von A – ohne Fehlerbehandlung	[BNull,r]	S. 99	`nullspace.par`, S. 122
pp	p1	primitiver Anteil von $p1$	p1	S. 101	`polynomutilities.par`, S. 117
PolCoeffMod	p, fac	Polynom $p \mod fac$	p	S. 102	`polynomutilities.par`, S. 117
PolRandom	p,n,fn	Zufallspolynom max. Grad$\leq n$	p	S. 102	`polynomutilities.par`, S. 117
polnorm	f	euklidische Norm der Koeffizienten von f	normval	S. 103	`polynomutilities.par`, S. 117
projectloader	–	Laderoutine	–	S. 113	`projectloader.par`, S. 114
pseudo_division	u,v	$\mathrm{lc}(v)^{\deg(u)-\deg(v)+1}u = qv + r$	[q,r]	S. 10	`polynomutilities.par`, S. 117
subresultant_gcd	p1,p2, bCalcRes	Bestimmung des ggTs – Hauptroutine	gcd oder [gcd,Res]	S. 19	`subresultant.par`, S. 124
subresultant_gcd_calc	p1,p2, bCalcRes	Bestimmung des ggTs – ohne Fehlerbehandlung	gcd oder [gcd,Res]	S. 19	`subresultant.par`, S. 124
vecnorm	v	euklidische Norm des Vektors v	normval	S. 100	`polynomutilities.par`, S. 117

E.2.2 Modulstruktur des *Cantor-Zassenhaus*-Algorithmus

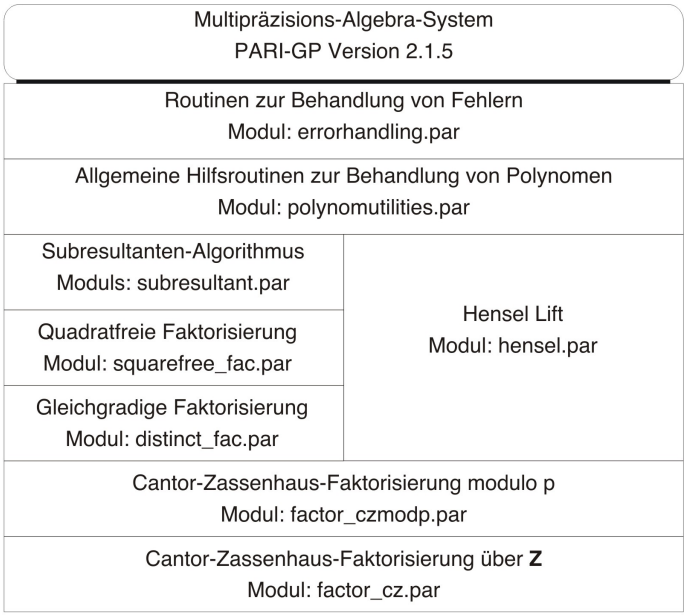

Diagramm 35: Modul-Abhängigkeiten im Cantor-Zassenhaus-Algorithmus

E.2.3 Modulstruktur des *LLL*-Algorithmus

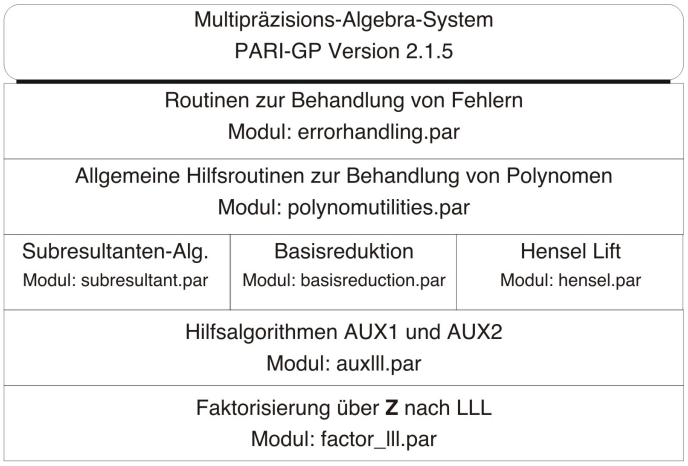

Diagramm 36: Modul-Abhängigkeiten im LLL-Algorithmus

E.2.4 Tests und Laufzeitverhalten

Im Hauptverzeichnis der zur Arbeit mitgelieferten CD befindet sich ein Verzeichnis *Tests*, welches alle zur Verifikation der erstellten Routinen verwendeten Testdaten enthält. Die Tests bestehen alle aus kurzen *PARI/GP*-Programmen und arbeiten nach zwei möglichen Mustern:

i.) Falls eine mit der zutestenden Routine vergleichbare Routine auch innerhalb von *PARI/GP* existiert, so wird diese zur Verifikation der Resultate verwendet.

ii.) Aufgrund fehlender Vergleichsroutinen werden anhand der Kenntnis des Algorithmus die Resultate überprüft.

Zur Durchführung von Laufzeitbetrachtungen stehen zwei Programme zur Verfügung, die an die jeweiligen Algorithmen angepaßt sind:

- `factor-cz-timing-test.par` für den Laufzeittest des Cantor-Zassenhaus-Algorithmus, bzw.
- `factor-lll-timing-test.par` für den Laufzeittest des LLL-Algorithmus.

Durch festes Setzen des Zufallsgenerators ist dabei gewährleistet, dass in beiden Tests die gleiche Sequenz zufälliger Polynome erzeugt wird.
Beide Programme geben am Ende eine kurze Übersicht über die Laufzeit sowie sonstige statistisch notwendige Daten. Alle Testprogramme werden durch einen direkten `read`-Aufruf gestartet.

E.2.5 Dokumentation

Bezüglich der Dokumentation der Algorithmen sei angemerkt, dass beginnend mit Kapitel E.3 des Anhangs das Unterkapitel das jeweilige Verzeichnis (genannt in Klammern) repräsentiert. Die Paragraphen entsprechen den im Unterverzeichnis enthaltenen Dateien; auch hier werden die jeweiligen Dateinamen der Quelldateien in Klammern angegeben.

E.2.6 Projectloader

Diese Routine dient lediglich als Hilfsroutine für das gesamte Projekt. Durch den PARI-Befehl

```
gp>  read("projectloader.par")
```

werden automatisch sämtliche für das Projekt benötigte Routinen nachgeladen. Alle in der Gesamtübersicht (E.2) aufgeführten Algorithmen stehen danach zur Verfügung. Sämtliche im folgenden abgedruckten Algorithmen können natürlich auch mit einzelnen `read`-Kommandos geladen werden; hierbei muss dann allerdings eigenverantwortlich auf die jeweiligen Abhängigkeiten geachtet werden.

Sourcen für Projectloader:

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

                        Projectloader

   ============================================================

   Beschreibung :   Diese Routine dient dazu, alle fuer  das
                    o.g. Projekt notwendigen   Routinen  und
                    Funktionen aus den   jeweiligen  Dateien
                    zu laden.  Nach Ausfuehrung dieser Datei
                    stehen fortan saemtliche Funktionen  zur
                    Verfuegung.
   ------------------------------------------------------------
   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :05.09.2003
   Datei        :projectloader.par
*/

loadfunc(fname)=
{
 print(concat("lade ",fname));
 read(fname);
}
\\-------------------------------------------------------------
print("");
print("");
print("Projectloader - Diplomarbeit Lars F. Paape 03/04");
print("");
print("      Faktorisierung von Polynomen ueber Z");
print("================================================");
print("");
\\
\\  Hilfsfunktionen
\\
print("Hilfsfunktionen werden geladen!");
loadfunc("Utilities/errorhandling.par");
loadfunc("Utilities/nullspace.par");
loadfunc("Utilities/polynomutilities.par");
loadfunc("Utilities/hensel.par");
loadfunc("Utilities/subresultant.par");
```

```
loadfunc("PreFactoring/squarefree_fac.par");
loadfunc("PreFactoring/distinct_fac.par");
print("");
\\
\\  Funktionen fuer den LLL-Algorithmus
\\
print("Funktionen fuer den LLL-Algorithmus werden geladen!");
loadfunc("Berlekamp/berlekamp.par");
loadfunc("LLL/basisreduction.par");
loadfunc("LLL/auxlll.par");
loadfunc("LLL/factor_lll.par");
print("");
\\
\\  Funktionen fuer den Algorithmus von Cantor-Zassenhaus
\\
print("Funktionen fuer den Algorithmus von");
print("Cantor-Zassenhaus werden geladen!");
loadfunc("Cantor-Zassenhaus/factor_czmodp.par");
loadfunc("Cantor-Zassenhaus/factor_cz.par");
print("");
\\-------------------------------------------------------
print("");
print("Gesamtfunktionalitaet steht zur Verfuegung!");
print("");
print("");
```

E.3 Hilfsalgorithmen (Utilities)

E.3.1 Fehlerbehandlung - errorhandling.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

         Allgemeine Hilfsroutinen zur Fehlerbehandlung
     und zur Behandlung von Hilfefunktionen

   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :12.09.2003
   Datei        :Utilities/errorhandling.par

*/
errorHandling(Error)=
{
  printp("Fehler: "+Error);
}
\\-------------------------------------------------------------
\\ Funktionshilfe
StdHelpMessage(stMsg)=
{
  stAdd="\nRoutine zur Diplomarbeit \"Faktorisierung von";
  stAdd=concat(stAdd," Polynomen mit ganzzahligen");
  stAdd=concat(stAdd," Koeffizienten\", Lars Fabian Paape");
  stAdd=concat(stAdd,", Universitaet Bremen, 2003.");
  return (concat(stMsg,stAdd));
}
\\-------------------------------------------------------------
```

E.3.2 Polynomroutinen - polynomutilities.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

  Allgemeine Hilfsroutinen zur Behandlung von Polynomen

   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :12.09.2003
   Datei        :Utilities/polynomutilities.par

   Achtung      :keine explizite Fehlerbehandlung;
                 es werden korrekte Eingabewerte/Typen erw.
*/
\\-------------------------------------------------------------
\\ Routine zur Bestimmung des ggT der Koeffizienten
\\ content of p1
cont(p1)=
{
  \\ lokale Variablen
  local(gcd_k,vec_pol,n,i);
  \\ Nullpolynom?
  if (p1==0,return(0););
  \\ Bestimmung des ggT
  vec_pol=Vec(p1);
  n      =poldegree(p1);
  gcd_k  =vec_pol[1];
  for (i=2,n+1 ,
        gcd_k=gcd(gcd_k,vec_pol[i]);
      );
  \\ Konvention zur Bestimmung des Leitkoeffizienten
  if (centerlift(vec_pol[1]/gcd_k)<0,
        return (-gcd_k);,
        return (gcd_k);
     );
}




\\-------------------------------------------------------
```

```
\\ Routine zur Bestimmung des primitiven Anteils von p1
\\ primitive part of p1
pp(p1)=
{
  local(c);
  c=cont(p1);
  if (c!=0,
      return (p1/c);,
      return (0);
     );
}
\\-----------------------------------------------------------
\\ Umwandlung eines Polynoms ueber Z oder Z/pZ nach Z/facZ
PolCoeffMod(p,fac)=
{
  local(polvec);
  polvec=Vec(p);
  polvec=Mod(centerlift(polvec),fac);
  return (Pol(polvec));
}
\\-----------------------------------------------------------
\\ Erzeugung eines zufaelligen Polynoms f ueber Z oder Z/pZ
\\ mit deg(f) <=n und Hoechstkoeff fn
PolRandom(p,n,fn=0)=
{
  local(polvec,i,g,dMax);
  polvec=vector(n+1);
  dMax=1;
  for(i=0,n,
      if (p>0,
          polvec[i+1]=Mod(random(p),p);
          ,
          polvec[i+1]=random();
         );
      if (polvec[i+1]!=0,
          dMax=i+1;
         );
     );
  if (fn!=0,
      polvec[dMax]=fn;
     );
  g=Polrev(polvec);
  return (g);
}
```

```
\\-----------------------------------------------------------
\\ Pseudo-Divisions-Algorithmus
\\ m=deg(u)>=deg(v)=n
pseudo_division(u,v)=
{
  \\ lokale Variablen
  local (q,r,e,d,m,mact,n,s);
  m=poldegree(u);
  n=poldegree(v);
  d=polcoeff(v,n);
  e=m-n+1;
  r=u;
  q=0;
  while (poldegree(r)>=n,
         mact=poldegree(r);
         s=polcoeff(r,mact);
         s*=x^(mact-n);
         q=d*q+s;
         r=d*r-s*v;
         e--;
        );
  d=d^e;
  q=q*d;
  r=r*d;
  return ([q,r]);
}
\\-----------------------------------------------------------
\\ Euklidischer Divisions-Algorithmus
\\ unter Verwendung der o.g. Pseudo-Division
euklid_division(u,v)=
{
  \\ Aus Gruenden der Verarbeitungsgeschw. obiger
  \\ pseudo-divisions-Funktion wird auf die
  \\ eukl. Division von PARI/GP zurueckgegriffen
  /*
  \\ lokale Variablen
  local(l,m,n,coeff,res);
  m=poldegree(u);
  n=poldegree(v);
  if(m<n,
     return([0,u]);
    );
  l=polcoeff(v,n);
  coeff=1/l^(m-n+1);
```

```
  res=pseudo_division(u,v);
  return ([res[1]*coeff,res[2]*coeff]);
  */
  return(divrem(u,v));
}
\\-----------------------------------------------------------
\\ Erweiterter Euklidischer Algorithmus nach Cohen, S.113
\\
ext_gcd(a,b)=
{
  \\ lokale Variablen
  local(u,d,v1,v3,t,r,q);
  \\ Sonderfallbehandlung
  if ((a==0)&&(b==0),
      return ([0,0,0]);
     );
  if ((a==0),
      return ([0,1,b]);
     );
  if ((b==0),
      return ([1,0,a]);
     );
  \\ Initialisierung
  u=1;
  d=a;
  v1=0;
  v3=b;
  while (v3!=0,
         r=d%v3;
         q=(d-r)/v3;
         t=u-v1*q;
         u=v1;
         d=v3;
         v1=t;
         v3=r;
    );
  \\ Bestimmung von v (exakte Division)
  v=(d-a*u)/b;
  return ([u,v,d]);
}
\\-----------------------------------------------------------
\\ Formale Ableitung eine Polynoms f
derivative(f)=
{
```

```
 local(i,n,dfv);
 n=poldegree(f);
 if (n<1,
     return(0);
     ,
     dfv=Vec(f);
     dfv=vecextract(dfv,concat("..",n));
     for(i=1,n, dfv[n+1-i]*=i;);
     return(Pol(dfv));
    );
}
\\-----------------------------------------------------------
\\ Euklidische Norm des Polynoms f
polnorm(f)=
{
  local(i,n,normval);
  n=poldegree(f);
  normval=0;
  for(i=0,n,
       normval+=sqr(polcoeff(f,i));
     );
  return (sqrt(normval));
}
\\-----------------------------------------------------------
\\ Euklidische Norm des Vectors v
vecnorm(v)=
{
  return(sqrt(v*v~));
}
```

E.3.3 Bestimmung des Nullraums - nullspace.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

   Teil:          Algorithmus zur Bestimmung
                  des Nullraums  nach Knuth
                  Band 2, 4.6.2, Algorithmus N

   ============================================================

   Autor         :Lars Fabian Paape
   Ort           :Universitaet Bremen
   Datum         :23.09.2003
   Datei         :Utilities/nullspace.par
*/
\\-------------------------------------------------------------
nullspace(A)=
{
 local(m,Error);
 \\ Fehlerbehandlung
 Error ="Argument falsch, nur nxn Matrizen  zulaessig!";
 if ((type(A)!="t_MAT"),
      errorHandling(Error),
      \\ else
      m=matsize(A);
      if(m[1]!=m[2],
         errorHandling(Error),
         \\ else
         return (nullspace_calc(A));
        );
     );
}


\\-------------------------------------------------------------
\\ Hilfeinformation
stMsg="[N,r]=nullspace(A): Bestimmung des";
stMsg=concat(stMsg," Nullraums der nxn-Matrix A. Die Routine");
stMsg=concat(stMsg," liefert eine Basis des Nullraums, sowie");
stMsg=concat(stMsg," die Dimension r des Nullraums.");

addhelp(nullspace,StdHelpMessage(stMsg));
```

```
\\-------------------------------------------------------------
\\ Triangularisierung
\\
nullspace_calc(A)=
{
  \\ lokale Variablen
  local(c,r,n,i,j,k,l,bHit,BNull);

  \\ Initialisierung
  n=(matsize(A))[1];
  r=0;
  c=vector(n,i,-1);
  BNull=matrix(n,n,i,j,0);
  for(k=1,n,
      bHit=0;
      for(j=1,n,
          if ((A[k,j]!=0)&&(c[j]<0),
              \\ Operation auf Spalte j
              A[,j]*=-1/A[k,j];
              \\ Ausraeumen der k-ten Zeile
              for(l=1,n,
                  if(l!=j,
                     A[,l]+=A[k,l]*A[,j];
                    );
                 );
              c[j]=k;
              bHit=1;
      break;
             );
         );
      if (!bHit,
          \\Vektor erzeugen
          r++;
          for(j=1,n,
              if(c[j]>-1,
                 BNull[r,c[j]]=A[k,j];
                );
             );
          BNull[r,k]=1;
         );
     );
  return([BNull,r]);
}
```

E.3.4 Subresultanten-Algorithmus - subresultant.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

               Implementierung des LLL-Algorithmus

   Teil:       Subresultanten Algorithmus nach Knuth

   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :20.10.2003
   Datei        :Utilities/subresultant.par
*/
subresultant_gcd(p1,p2,bCalcResultant=0)=
{
 local(pol,num);
 \\ Fehlerbehandlung
 Error ="Argument(e) falsch!";
 if (((type(p1)!="t_INT")&&(type(p1)!="t_POL")
      &&(type(p1)!="t_INTMOD"))||
     ((type(p2)!="t_INT")&&(type(p2)!="t_POL")
      &&(type(p2)!="t_INTMOD")),
     errorHandling(Error)
     ,
     return(subresultant_gcd_calc(p1,p2,bCalcResultant));
    );
}
\\-------------------------------------------------------------
\\ Hilfeinformation
stMsg="subresultant_gcd(p1,p2,{bCalcResultant=0}): Bestimmung ";
stMsg=concat(stMsg,"des ggT zweier Polynome mit ganzzahligen ");
stMsg=concat(stMsg,"Koeffizienten. Falls bCalcResultant 0 ist,");
stMsg=concat(stMsg,"  so wird der ggT zurueckgegeben, falls ");
stMsg=concat(stMsg,"nicht so erfolgt die Ausgabe in der ");
stMsg=concat(stMsg,"Form [ggT,Resultante].");

addhelp(subresultant_gcd,StdHelpMessage(stMsg));
\\-------------------------------------------------------------
subresultant_gcd_calc(u,v,bCalcResultant=0)=
{
  \\ lokale Variablen
```

```
local(d,lu,lv,w,g,h,delta,r,psd);
local(udeg,vdeg,wdeg,ludeg,lvdeg);
local(s,t,uc,vc,slv);

\\ Resultantenbestimmung
s  =1;

udeg =poldegree(u);
vdeg =poldegree(v);

\\ deg(u)>=deg(v) ?
if(udeg<vdeg,
   \\ Tausche Polynome
   w=u;       u=v;        v=w;
   wdeg=udeg; udeg=vdeg;  vdeg=wdeg;
   \\ Resultantenbestimmung
   if(((udeg%2)>0)&&((vdeg%2)>0), s=-1; );
  );
\\ Sonderfaelle: eines der Polynome ist das Nullpolynom
if ((v==0) ,
    if (bCalcResultant!=0,
       return ([u,0]);,   \\Ausgabe mit  Resultante
       return (u);        \\Ausgabe ohne Resultante
       );
   );
\\ eigentlicher Algorithmus
uc =cont(u);
vc =cont(v);
d     =gcd(uc,vc);
\\ primitive Polynome
lu =pp(u);
lv =pp(v);
g =1;
h =1;

\\ Resultantenbestimmung
t  =uc^vdeg*vc^udeg;
while(1,
      \\ Pseudo-Division
      ludeg=poldegree(lu);
      lvdeg=poldegree(lv);
      delta=ludeg-lvdeg;
      \\ Resultantenbestimmung
      if (bCalcResultant,
```

```
            \\ Polynomgrad ungerade?
            if(((ludeg%2)>0)&&((lvdeg%2)>0),s*=-1;);
           );
        \\ Bestimmung von r
        psd=pseudo_division(lu,lv);
        r=psd[2];
     \\ Ende ?
     if ((r==0)||(poldegree(r)==0),
            slv=lv;
    if((poldegree(r)==0),
               lv=1;
               );
            \\  Algorithmus terminiert
            \\  Resultantenbestimmung
            if (bCalcResultant,
                \\Ergebnis mit Resultante
                w=lv;
                lu=slv;
                slv=r/(g*h^delta);
                g=polcoeff(lu,poldegree(lu));
                h=h^(1-delta)*g^delta;
                h=h^(1-poldegree(lu))*\
                  polcoeff(slv,poldegree(slv))^poldegree(lu);
                return ([d*pp(w),s*t*h]);
                ,
                \\Ergebnis ohne Resultante
                \\
               return (d*pp(lv));
               );
           );
        \\ Anpassung des Rests
        lu=lv;
        lv=r/(g*h^delta);
        g=polcoeff(lu,poldegree(lu));
        h=h^(1-delta)*g^delta;
      );

}
```

E.3.5 Linearer Hensel Lift - `hensel.par`

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ===========================================================

              Implementierung des linearen Hensel Lifts

   ===========================================================

   Autor         :Lars Fabian Paape
   Ort           :Universitaet Bremen
   Datum         :03.09.2003
   Datei         :Utilities/hensel.par
*/
lin_lift(f,g1,h1,p,k)=
{
 \\ Fehlerbehandlung
 Error ="Argument(e) falsch!";
 if (((type(f)!="t_POL")&&(type(f)!="t_INT")
      &&(type(f)!="t_INTMOD"))||
     ((type(g1)!="t_POL")&&(type(g1)!="t_INT")
      &&(type(g1)!="t_INTMOD"))||
     ((type(g2)!="t_POL")&&(type(g2)!="t_INT")
      &&(type(g2)!="t_INTMOD"))
     ||(type(k)!="t_INT")||(k<=0)||(p<2),
     errorHandling(Error)
     ,
     \\else
     return (lin_lift_calc(f,g1,h1,p,k));
 );
}

\\-------------------------------------------------------------
\\ Hilfeinformation
stMsg="lin_lift(f,g1,h1,p,k): Hensel Lift ";
stMsg=concat(stMsg,"von mod p nach mod p^k ");


addhelp(lin_lift,StdHelpMessage(stMsg));
\\-------------------------------------------------------------
```

```
lin_lift_calc(f,g1,h1,p,k)=
{
  \\ lokale Variablen
  local(gk,hk,pk,as,bs,a,b,res_gcd,poldiv,q,r);
  \\ f,g1,h1 muessen aus Z[X] sein
  f =centerlift(f);
  g1=centerlift(g1);
  h1=centerlift(h1);
  \\ gkp, hkp sind gk,hk  mod  p
  gk=g1;
  hk=h1;
  pk=p;
  gkp=PolCoeffMod(gk,p);
  hkp=PolCoeffMod(hk,p);
  \\erw. Euklidischer Algorithmus zur Best. von a & b
  res_gcd=ext_gcd(gkp,hkp);
  \\falls ggT(gk,hk)=d!=1
  res_gcd*=1/res_gcd[3];
  a=centerlift(res_gcd[1]);
  b=centerlift(res_gcd[2]);
  \\ Hauptschleife
  for(i=1,k-1,
      \\Einzelschritt
      \\Bestimmung von c
      c=PolCoeffMod((f-gk*hk)/pk,p);
      \\ Bestimmung von as*g+bs*h=c mod p
      if (a*c!=0,
          poldiv=euklid_division(PolCoeffMod(a*c,p),hkp);
          ,
          \\else
          poldiv=[0,0];
         );
      q =poldiv[1];
      r =poldiv[2];
      as=centerlift(r);
      bs=b*c+gk*q;
      \\ k+1
      gk+=pk*centerlift(bs);
      hk+=pk*centerlift(as);
      pk*=p;
  );
  \\ Rueckgabewerte
  gk=centerlift(PolCoeffMod(gk,pk));
```

```
  hk=centerlift(PolCoeffMod(hk,pk));
  return([gk,hk]);
}
\\-------------------------------------------------------
\\   Algorithmus zum gleichzeitigen Liften
\\          mehrerer Faktoren
multi_lin_lift(f,facs,p,k)=
{
  \\ lokale Variablen
  local(i,LiftFac,NumFac,ff,g1,g2);
  \\ Initialisierung
  LiftFac=[];
  NumFac=matsize(facs)[2];
  \\ Bilden der ersten beiden Faktoren
  ff=f;
  g1=facs[1];
  g2=1;
  for(i=1,NumFac-1,
      g2*=facs[i+1];
     );

  for(i=1,NumFac-1,
      LiftRes=lin_lift(ff,g1,g2,p,k);
      \\ ein neuer Faktor ist geliftet
      LiftFac=concat(LiftFac,\
             [centerlift(PolCoeffMod(LiftRes[1],p^k))]);
      if(i==(NumFac-1),
         \\ letzten Faktor anfuegen
         LiftFac=concat(LiftFac,\
                [centerlift(PolCoeffMod(LiftRes[2],p^k))]);
         ,
         \\else
         \\ Anpassung der Faktoren
         ff=centerlift(PolCoeffMod(ff,p^k)\
           /PolCoeffMod(LiftRes[1],p^k));
         g1=facs[i+1];
         g2=g2/g1;
        );
     );
  \\ Rueckgabe der gelifteten Faktoren
  return (LiftFac);
}
```

E.4 Vorfaktorisierung (PreFactoring)

E.4.1 Quadratfreie Faktorisierung - squarefree_fac.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

   Implementierung des Cantor-Zassenhaus-Algorithmus

   Teil:         Algorithmus zur Faktorisierung
                 eines Polynoms modulo p in ein Produkt
                 quadratfreier Polynome
                 nach Cohen, 3.4.2, inkl. Anpassung
   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :08.11.2003
   Datei        :PreFactoring/squarefree_fac.par
*/
\\-----------------------------------------------------------
factor_squarefree(f,p)=
{
 local(f_modp,f_monp,l,res_fac);
 \\ Fehlerbehandlung
 Error ="Argument(e) falsch, f Polynom und p Primzahl!";
 if (((type(f)!="t_POL") &&(type(f)!="t_INT")
    &&(type(f)!="t_INTMOD"))
    ||(type(p)!="t_INT")||(!isprime(p)),
     errorHandling(Error);,
     \\ Berechnung der Faktorisierung
     if (poldegree(f)>0,
         f_modp=PolCoeffMod(f,p);
         l=polcoeff(f_modp,poldegree(f_modp));
         f_monp=Pol(Vec(f_modp)/l);
         res_fac=factor_squarefree_calc(f_monp,p);
         return ([l*res_fac[1],\
                 (matsize(res_fac[2]))[2],res_fac[2]]);
         ,
         \\ else
         return ([PolCoeffMod(f,p),0,[]]);
        );
    );
}
```

```
\\-------------------------------------------------------
\\ Hilfeinformation
stMsg="[l,n,[[n1,fac1],...,[nn,facn]]]=factor_";
stMsg=concat(stMsg,"squarefree(f,p): Faktorisierung von ");
stMsg=concat(stMsg,"f mod p in ein Produkt quadratfreier");
stMsg=concat(stMsg," Polynome(squarefree factorization).");

addhelp(factor_squarefree,StdHelpMessage(stMsg));
\\-------------------------------------------------------
\\ Achtung: Die nachfolgende Funktion geht davon aus,
\\          dass f normiert ist.
factor_squarefree_calc(f,p)=
{
  \\ lokale Variablen
  local(e,g,g0,h,k,w,a,j,l,resFac);

  \\ Initialisierung
  e     =1;
  resFac=[];
  l     =Mod(1,p);
  g0    =f;
  while (poldegree(g0)!=0,
         g=subresultant_gcd(g0,derivative(g0));
         h=g0/g;
         k=0;
         while(poldegree(h)!=0,
               k++;
               \\ p teilt k?
               if(Mod(k,p)==0,
                  g=g/h;
                  k++;
                );
               w=subresultant_gcd(g,h);
               a=h/w;
               h=w;
               g=g/h;
               if (poldegree(a)>0,
                   \\ neuer Faktor gefunden
                   resFac=concat(resFac,[[e*k,a]]);,
                   \\ else: Anpassung konst. Faktor
                   l=l*a;
                  );
             );
         \\ g0 hat jetzt spezielle Form
```

```
        g0=0;
        forstep(j=0,poldegree(g),p,
                g0=g0+polcoeff(g,j)*x^(j/p);
               );
        e=p*e;
       );
  \\ Korrektur konst. Faktor
  l=l*g0;
  \\ Rueckgabewert
  return([l,resFac]);
}
```

E.4.2 Gleichgradige Faktorisierung - distinct_fac.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

   Implementierung des Cantor-Zassenhaus-Algorithmus

   Teil:         Algorithmus zur gleichgradigen Faktorisierung
                 eines Polynoms modulo p
                 nach Cohen, 3.4.3
   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :11.11.2003
   Datei        :PreFactoring/distinct_fac.par
*/
\\-----------------------------------------------------------
factor_distinct(f,p)=
{
 local(f_modp);
 \\ Fehlerbehandlung
 Error ="Argument(e) falsch, f Polynom und p Primzahl!";
 if ((type(f)!="t_POL") ||(type(p)!="t_INT")||(!isprime(p)),
         errorHandling(Error);,
         \\ Berechnung der Faktorisierung
         f_modp=PolCoeffMod(f,p);
         return (factor_distinct_calc(f_modp,p));
    );
}
\\-----------------------------------------------------------
\\ Hilfeinformation
```

```
stMsg="[l,n,[[d1,fac1],...,[dn,facn]]]=factor_distinct";
stMsg=concat(stMsg,"(f,p): Faktorisierung eines quadratfr.");
stMsg=concat(stMsg,"Polynoms f modulo p in Polynome gleich");
stMsg=concat(stMsg,"gradiger Faktoren (distinct degree)");
stMsg=concat(stMsg,"mit f=l*fac1*...*facn wobei faci norm.");
stMsg=concat(stMsg,"ist und nur Faktoren vom Grad di enth.");
addhelp(factor_distinct,StdHelpMessage(stMsg));
\\-----------------------------------------------------
\\ Achtung: f muss quadratfrei sein.
factor_distinct_calc(f,p)=
{
  \\ lokale Variablen
  local(g,h,d,e,resFac,fd,l,coe);
  \\ Initialisierung
  g=f;  h=Mod(1,p)*x;
  d=0;  resFac=[];  l=Mod(1,p);
  while(1,
         e=poldegree(g);
         if(d+1>e/2,
            \\ Ende?
            if(e>0,
               fd=g;
               coe=polcoeff(fd,e);
               l*=coe;
               resFac=concat(resFac,[[e,fd/coe]]);
              );
            break;
            ,
            \\ else
            d++;
            h=centerlift(Mod(h^p,g));
           );
         \\ Ausgabe des aktuellen Faktors
         fd=subresultant_gcd(h-Mod(1,p)*x,g);
         if(fd!=1,
            coe=polcoeff(fd,poldegree(fd));
            l*=coe;
            resFac=concat(resFac,[[d,fd/coe]]);
            g=g/fd;
            h=centerlift(Mod(h,g));
           );
       );
  return([l,matsize(resFac)[2],resFac]);
}
```

E.5 Berlekamp-Algorithmus (Berlekamp)

E.5.1 Faktorisierung modulo p - `berlekamp.par`

```
/* Faktorisierung von Polynomen mit rationalen Koeffizienten
   ========================================================

   Implementierung des LLL-Algorithmus

   Teil:         Algorithmus von Berlekamp zur
                 Faktorisierung eines quadratfreien
                 Polynoms modulo p nach Knuth, Band 2, 4.6.2
   ========================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :09.12.2003
   Datei        :Berlekamp/berlekamp.par
*/
\\-----------------------------------------------------------
factor_berlekampsf(u,p)=
{
 local(i,n,U,Error,U0);
 \\ Fehlerbehandlung
 Error ="Argument(e) falsch, u Polynom und p Primzahl!";
 if(((type(u)!="t_POL")&&(type(u)!="t_INT")&&
     (type(u)!="t_INTMOD"))||(type(p)!="t_INT")||
     (isprime(p)!=1),
    errorHandling(Error);
    ,
    if((type(u)=="t_POL"),
       \\ Generierung des Koeffizientenvektors modulo p
       n=poldegree(u);
       U=vector(n+1,i,Vec(u)[n+2-i]);
       U=Mod(centerlift(U),p);
       return (factor_berlekampsf_calc(U,n,p));
       ,
       \\ else
       return ([Mod(centerlift(u),p),0,[]]);
      );
    );
}
\\-----------------------------------------------------------
\\ Hilfeinformation
```

```
stMsg="[l,n,[[1,fac1],...,[1,facn]]=factor_berlekamp(u,p";
stMsg=concat(stMsg,"): Faktorisierung eines quadratfreien");
stMsg=concat(stMsg," Polynoms u modulo p in irreduzible");
stMsg=concat(stMsg," Faktoren unter Verwendung ");
stMsg=concat(stMsg," des Verfahrens von Berlekamp, ");
stMsg=concat(stMsg," u=l*fac1...facn.");

addhelp(factor_berlekampsf,StdHelpMessage(stMsg));
\\-------------------------------------------------------
\\ Ausgangspunkt ist ein quadratfreies Polynom u(x)
\\ welches hier nur  noch durch seinen Koeffizientenvektor
\\ repraesentiert wird.
\\
factor_berlekampsf_calc(u,n,p)=
{
  \\ lokale Variablen
  local(i,j,k,t,Q,N,a,E,EE);
  local(T,s,F,f_gcd,l);
  local(res_fac,lambda);

  \\ Initialisierung
  lambda=u[n+1];
  \\ Normierung
  u=lambda^-1*u;
  \\ Bestimmung der nxn-Matrix Q
  Q=Mod(matrix(n,n),p);
  \\ erster Vektor wird immer mit (1,0,0,0,...) angenommen
  Q[1,1]=Mod(1,p);
  \\ Hilfsvektor zur Bestimmung der r[i,j]
  a   =Mod(vector(n),p);
  a[1]=Mod(1,p);
  \\ Berechnung der restlichen Zeilen
  for(j=1,n-1,
     \\ Bestimmung der neuen Zeile (j-1)p->jp
     for(k=1,p,
         t=a[n];
         forstep(i=n-1,1,-1,
                 a[i+1]=a[i]-t*u[i+1];
                );
         a[1]=-t*u[1];
        );
     \\ neue Zeile in Q
     Q[j+1,]=a;
     );
```

```
\\ Q=Q-I
Q=Q-matid(n);
\\ Berechnung des Nullraums
N=nullspace(Q);

\\ Berechnung der Faktorisierung
r=N[2];
k=1;
res_fac=[Polrev(u)];
i=2;
while(k<r,
      v=Polrev(N[1][i,]);
      res_actfacs=[];
      for(l=1,k,
          if (poldegree(res_fac[l])>1,
              for(j=0,p-1,
                  test_fac=subresultant_gcd(v-j,res_fac[l]);
                  if(poldegree(test_fac)>=1,
                     test_fac=test_fac\
                      /polcoeff(test_fac,poldegree(test_fac));
                     res_actfacs=concat(\
                      res_actfacs,[test_fac]);
                     \\ Ziel schon erreicht?
                     if((matsize(res_actfacs)[2]+k-l)==r,
                        \\ Kopieren der k-l verbleibenden Faktoren
                        for(j=l+1,k,
                            res_actfacs=concat(\
                            res_actfacs,[res_fac[j]]);
                           );
                        break(2);
                       );
                    );
                );
              ,
              \\else
              res_actfacs=concat(res_actfacs,[res_fac[l]]);
             );
         );
       res_fac=res_actfacs;
       k=matsize(res_fac)[2];
       i++;
      );
```

```
  \\ Rueckgabewerte gem. verw. Konvention
  res_out_fac=[];
  for(i=1,r,
       res_out_fac=concat(res_out_fac,[[1,res_fac[i]]]);
     );
  return([lambda,r,res_out_fac]);
}
```

E.6 Cantor-Zassenhaus-Algorithmus (Cantor-Zassenhaus)

E.6.1 Polynom-Faktorisierung modulo p - factor_czmodp.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ===========================================================

   Implementierung des Cantor-Zassenhaus-Algorithmus

   Teil:         Algorithmus zur gleichgradigen Faktorisierung
                 eines Polynoms modulo p
                 nach Cohen, 3.4.6, modifiziert,
                 rekursionsfrei
   ===========================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :15.11.2003
   Datei        :Cantor-Zassenhaus/factor_czmodp.par
*/
\\-----------------------------------------------------------
factor_czmodp(f,p,bSquareFree=0)=
{
 local(f_modp);
 \\ Fehlerbehandlung
 Error ="Argument(e) falsch, f Polynom und p Primzahl!";
 if (((type(f)!="t_POL")&&(type(f)!="t_INT"))||
     (type(p)!="t_INT")||(!isprime(p)),
 errorHandling(Error);
 ,
         \\ Berechnung der Faktorisierung
         if (type(f)!="t_INT",
             f_modp=PolCoeffMod(f,p);
             return (factor_czmodp_calc\
                     (f_modp,p,bSquareFree));
             ,
             \\ else
             return ([f,0,[]]);
            );

    );
}
\\-----------------------------------------------------------
```

```
\\ Hilfeinformation
stMsg="[l,n,[[d1,fac1],...,[dn,facn]]=factor_czmodp(f,p";
stMsg=concat(stMsg,",{bSquareFree=0}): Faktorisierung");
stMsg=concat(stMsg," von f  modulo p in irreduzible");
stMsg=concat(stMsg," Faktoren in drei{zwei} Stufen");
stMsg=concat(stMsg," unter Verwendung des Verfahren von");
stMsg=concat(stMsg," Cantor-Zassenhaus, ");
stMsg=concat(stMsg," f=l*fac1^d1*...facn^dn.");

addhelp(factor_czmodp,StdHelpMessage(stMsg));
\\-------------------------------------------------------
\\   Algorithmus zur Faktorisierung eines beliebigen
\\                   Polynoms modulo p
factor_czmodp_calc(f,p,bSquareFree)=
{
  local(l,SFRes,DDRes,FacRes,FacResSort);
  local(CZRes,num_fac,i,j,k,dmax,ff,n);
  \\ Faktorisierung in quadratfreie Faktoren
  \\ schon quadratfrei?
  if (!bSquareFree,
      \\ quadratfreie Zerlegung bestimmen
      SFRes=factor_squarefree(f,p);
      ,
      \\ Ergebnis gem. factor_squarefree formatieren
      l=polcoeff(f,poldegree(f));
      SFRes=[l,1,[[1,f/l]]];
     );
  FacRes=[];
  l=SFRes[1];
  \\ Faktorisierung in Polynome gleichgradiger Faktoren
  num_fac=SFRes[2];
  dmax=0;
  for(i=1,num_fac,
      ff=SFRes[3][i][2];
      n =SFRes[3][i][1];
      DDRes=factor_distinct(ff,p);
      l*=DDRes[1]^n;
      \\ Umkopieren der Daten
      for(j=1,DDRes[2],
          \\ Cantor-Zassenhaus
          ff=DDRes[3][j][2];
          d =DDRes[3][j][1];
          if(d>dmax,
             dmax=d;
```

```
            );
          CZRes=factor_cantorp_calc(ff,p,d);
          l*=(CZRes[1])^n;
          for(k=1,CZRes[2],
              FacRes=concat(FacRes,[[n,CZRes[3][k]]]);
             );
         );
     );
  \\ Faktoren sortieren nach Grad der Faktoren
  FacResSort=[];
  \\ dmax ist der Maximalgrad eines Faktors
  num_fac=(matsize(FacRes))[2];
  for(i=1,dmax,
      for(j=1,num_fac,
          if (poldegree(FacRes[j][2])==i,
              FacResSort=concat(FacResSort,[FacRes[j]]);
             );
         );
     );
   return ([l,num_fac,FacResSort]);
}
\\-------------------------------------------------------
\\ Algorithmus von Cantor-Zassenhaus zur Bestimmung der
\\ Faktorisierung eines quadratfreien Polynoms mit p>=2
factor_cantorp_calc(f,p,d)=
{
  if(p==2,
     return(factor_cantor_calc_p2(f,d));
     ,
     \\ else
     return(factor_cantor_calc_p(f,p,d));
    );
}
\\-------------------------------------------------------
\\ Algorithmus von Cantor-Zassenhaus zur Bestimmung der
\\ Faktorisierung eines quadratfreien Polynoms mit p>=3
factor_cantor_calc_p(f,p,d)=
{
  \\ lokale Variablen
  local(r,k,m,resFac,maxDeg,g,l,PolPool,lam,coe);
  \\ Initialisierung
  resFac =[];
  PolPool=[];
  r      =poldegree(f)/d;
```

```
k      =0;
m      =(p^d-1)/2;
maxDeg =2*d-1;
\\ zur Normierung der Faktoren
lam    =1;
\\ Schleife so lange,
\\ bis alle r-Faktoren gefunden
while (k<r,
       if (poldegree(f)!=d,
           \\ Zufallspolynom g bestimmen
           \\ n=deg(g)<=maxDeg, gn=1
           g=PolRandom(p,maxDeg,1);
           h=subresultant_gcd(f,centerlift(Mod(g,f)^m)-1);
           l=poldegree(h);
           if((l>0)&&(l!=poldegree(f)),
              if (l==d,
                  \\ Faktor gefunden!
                  coe=polcoeff(h,poldegree(h));
                  lam=lam*coe;
                  resFac=concat(resFac,[h/coe]);
                  f=f/h;
                  k++;
                  if ((poldegree(f)==0)&&(k<r),
                     \\ PolPool enthaelt Faktoren
                     f=PolPool[1];
                     if(matsize(PolPool)[2]>1,
                        PolPool=vecextract(PolPool,"2..");
                        ,
                        PolPool=[];
                      );
                     );
                  ,
                  \\ else nur ein Produkt von Faktoren
                  PolPool=concat(PolPool,[f/h]);
                  \\ faktorisiere zunaechst h
                  f=h;
                 );
            );
           ,
           \\ else
           coe=polcoeff(f,poldegree(f));
           lam=lam*coe;
           resFac=concat(resFac,[f/coe]);
           k++;
```

```
            \\ noch Polynome zu faktorisieren?
            if ((k<r)&&(matsize(PolPool)[2]>0),
                f=PolPool[1];
                if(matsize(PolPool)[2]>1,
                   PolPool=vecextract(PolPool,"2..");
                   ,
                   PolPool=[];
                  );
               );
            );
         );
  \\ resFac enthaelt die Faktorisierung
  return ([lam,matsize(resFac)[2],resFac]);
}
\\-----------------------------------------------------------
\\ Algorithmus von Cantor-Zassenhaus zur Bestimmung der
\\ Faktorisierung eines quadratfreien Polynoms mit p=2
factor_cantor_calc_p2(f,d)=
{
  \\ lokale Variablen
  local(p,r,k,resFac,g,c,l,PolPool);
  \\ Initialisierung
  p      =2;
  resFac =[];
  PolPool=[];
  r      =poldegree(f)/d;
  k      =0;
  g      =Mod(1,p)*x;
  \\ Schleife so lange,
  \\ bis alle r-Faktoren gefunden
  while (k<r,
         if (poldegree(f)!=d,
             \\ Bestimmung von g
             c=g;
             for(i=1,d-1,c=centerlift(Mod(g+c^2,f)););
             h=subresultant_gcd(f,c);
             l=poldegree(h);
             if((l>0)&&(l!=poldegree(f)),
                if (l==d,
                    \\ Faktor gefunden!
                    resFac=concat(resFac,[h]);
                    f=f/h;
                    k++;
                    if ((poldegree(f)==0)&&(k<r),
```

```
                    \\ PolPool enthaelt Faktoren
                    f=PolPool[1];
                    if(matsize(PolPool)[2]>1,
                       PolPool=vecextract(PolPool,"2..");
                       ,
                       PolPool=[];
                      );
                    );
                  ,
                  \\ else nur ein Produkt von Faktoren
                  PolPool=concat(PolPool,[f/h]);
                  \\ faktorisiere zunaechst h
                  f=h;
                 );
               \\ g initialisieren
               g=Mod(1,p)*x;
               ,
               \\ else (l==0) or (l=deg(f))
               g=g*x^2;
              );
            ,
            \\ else
            resFac=concat(resFac,[f]);
            k++;
            \\ noch Polynome zu faktorisieren?
            if ((k<r)&&(matsize(PolPool)[2]>0),
                \\ g initialisieren
                g=Mod(1,p)*x;
                f=PolPool[1];
                if(matsize(PolPool)[2]>1,
                   PolPool=vecextract(PolPool,"2..");
                   ,
                   PolPool=[];
                  );
               );
          );
        );
  \\ resFac enthaelt die Faktorisierung
  \\ die 1 dient nur dem Einhalten der
  \\ Aufrufkonvention
  return ([Mod(1,2),matsize(resFac)[2],resFac]);
}
```

E.6.2 Polynom-Faktorisierung über $\mathbb{Z}$ - factor_cz.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

   Implementierung des Cantor-Zassenhaus-Algorithmus

   Teil:         Algorithmus zur Faktorisierung
                 eines ganzzahligen Polynoms
                 nach Cohen, 3.5.7, mittels
                 rekursionsfreiem Cantor-Zassenhaus-Verfahren
   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :27.11.2003
   Datei        :Cantor-Zassenhaus/factor_cz.par
*/
\\-----------------------------------------------------------
factor_cz(f)=
{
 \\ Fehlerbehandlung
 Error ="Argument falsch, f Polynom";
 if ((type(f)!="t_POL")&&(type(f)!="t_INT"),
 errorHandling(Error);
 ,
         \\ Berechnung der Faktorisierung
         if ((type(f)=="t_POL")&&(poldegree(f)>=1),
             \\ Faktorisierung
             return (factor_cz_calc(f));
             ,
             \\ else deg(f)<=0
             return([f,0,[]]);
            );
    );
}
\\-----------------------------------------------------------
\\ Hilfeinformation
stMsg="[l,n,[[d1,fac1],...,[dn,facn]]=factor_cz(f):";
stMsg=concat(stMsg,"Faktorisierung von f  mittels");
stMsg=concat(stMsg," Cantor-Zassenhaus-Verfahren wobei");
stMsg=concat(stMsg," f=l*fac1^d1*...facn^dn bzw. f=l, ");
stMsg=concat(stMsg," falls deg(f)<=0. ");
```

```
addhelp(factor_cz,StdHelpMessage(stMsg));
\\-------------------------------------------------------
\\   Algorithmus zur Faktorisierung eines quadratfreien
\\       primitiven  ganzzahligen Polynoms
factor_czsf_calc(g)=
{
  \\ lokale Variablen
  local(l,p,eval_gcd,gmodp,res_facmodp,S,res_simpfac,lcg,lch);
  local(m,e,i,k,j,res_lift,r,d,h,subset,per,a,res_fac,fac);
  \\ quadratfreie Faktorisierung mod p
  p=2;
  while(1,
        if (Mod(polcoeff(g,poldegree(g)),p)!=0,
            \\ Grad muss erhalten bleiben
            gmodp=PolCoeffMod(g,p);
            eval_gcd=subresultant_gcd\
                     (gmodp,derivative(gmodp));
            \\ p gefunden ?
            if(eval_gcd==1,break;);
           );
        p=nextprime(p+1);
       );
  \\ Faktorisierung von g mod p
  \\ g ist quadratfrei!
  res_facmodp=factor_czmodp(g,p,1);
  \\ Bestimmung der Schranke zur Abschaetzung
  m=floor(poldegree(g)/2);
  S=binomial(m,floor(poldegree(g)/4))*polnorm(g);
  \\ Bestimmung des Exponenten e so, dass
  \\ p^e>2*S*||g||
  \\ Schranke gem. Mignotte im Unterschied zur bei
  \\ Cohen verwendeten Variante!
  e=1;
  if (S!=0, e=round(log(2*S)/log(p)););
  if (p^e<=2*S, e++;);
  \\ Umkopieren der Faktoren, g ist quadratfrei!
  res_simpfac=[centerlift(res_facmodp[1])];
  for(i=1,res_facmodp[2],
      res_simpfac=concat(res_simpfac,[\
                  centerlift(res_facmodp[3][i][2])]);
     );
  \\ Falls nur ein Faktor vorhanden ist
  if (res_facmodp[2]==1,
      res_simpfac=concat(res_simpfac,[1]);
```

```
   );
\\ Lift der Faktorisierung nach mod p^e
res_lift= multi_lin_lift(g,res_simpfac,p,e);
r=res_facmodp[2];
\\ g= lc(g)*res_lif[1]*...*res_lift[r] mod p^e
\\ Finden der Faktoren
res_fac=[];
d=1;
\\ generieren eines Testpolynoms h
\\ i gibt Anzahl der Faktoren an
i=1;
count_fac=r;
bHit=0;
lcfac=res_lift[1];
res_lift=vecextract(res_lift,"2..");
while(i<=count_fac,
      \\ Initialisierung des Vektors
      a=vector(i,j,j);
      Permut=1;
      lcg=polcoeff(g,poldegree(g));
      while(Permut>=1,
            subset=[a[i]];
            forstep(j=i-1,1,-1,
                    subset=concat([a[j]],subset);
                   );
            \\ Bestimme nun h mittels subset
            h=lcfac;
            for(j=1,i,
                h*=res_lift[subset[j]];
               );
            h=centerlift(PolCoeffMod(h,p^e));
            \\ nur der primitive Anteil ist interessant
            h=pp(h);
            lch=polcoeff(h,poldegree(h));
            \\ gilt deg(h)<=m
            if((poldegree(h)<=m),
               \\ ist h Teiler von res_facmodp[1]*g?
               if((lcg%lch)==0,
                  if(g%h==0,
                     \\ Faktor gefunden
                     res_fac=concat(res_fac,h);
                     g=g/h;
                     lcg=lcg/lch;
                     m=floor(poldegree(g)/2);
```

```
                        \\ Faktorenzahl um i reduzieren
                        count_fac-=i;
                        resnew_lift=[];
                        for(j=1,r,
                            bFound=0;
                            for(k=1,i,
                                if (subset[k]==j,
                                    bFound=1;
                                    break;
                                   );
                               );
                            if(bFound==0,
                               resnew_lift=concat\
                                 (resnew_lift,[res_lift[j]]);
                              );
                           );
                        i=0;
                        r=count_fac;
                        res_lift=resnew_lift;
                        bHit=1;
                       );
                   );
               );
            if(bHit!=1,
               if(a[i]==r,
                  Permut--;
                  ,
                  Permut=i;
                 );
               if(Permut>=1,
                  forstep(j=i,Permut,-1,
                          a[j]=a[Permut]+j-Permut+1;
                          );
                 );
               ,
               bHit=0;
               break;
              );
           );
         i++;
      );
   res_fac=concat(res_fac,[g]);
   return(res_fac);
}
```

```
\\-------------------------------------------------------
\\   Algorithmus zur Faktorisierung eines beliebigen
\\                 ganzzahligen Polynoms
factor_cz_calc(f)=
{
  \\ lokale Variablen
  local(res_fac,res_sf_fac,n,resultant,g,i,res_div);
  \\ Initialisierung
  n=poldegree(f);
  resultant=subresultant_gcd(f,derivative(f),1);
  \\ Ist f quadratfrei ?
  \\
  g=1;
  if (resultant[2]==0,
      \\ nein
      g=resultant[1];
      f=f/g;
     );
 l=cont(f);
 f=pp(f);
 \\ f ist ab hier immer quadratfrei
 \\ und primitiv wg. s.o.
 res_sf_fac=factor_czsf_calc(f);
 \\ Faktorisierung umkopieren
 res_fac=[];
 for(i=1,matsize(res_sf_fac)[2],
     res_fac=concat(res_fac,[[1,res_sf_fac[i]]]);
    );
 \\  res_fac enthaelt nun Faktorisierung
 \\  des quadratfreien Anteils des urspr. f

 \\  g beruecksichtigen
 if (g!=1,
     \\ urspr. f war nicht quadratfrei
     i=1;
     while(poldegree(g)>=1,
           res_div=euklid_division(g,res_fac[i][2]);
           if(res_div[2]==0,
              \\ Faktor gefunden
              g=res_div[1];
              res_fac[i][1]+=1;
              ,
              \\ else
              \\ naechsten Faktor testen
```

```
              i++;
             );
          );
    );
 \\Rueckgabewert
 return([l*g,matsize(res_fac)[2],res_fac]);
}
```

E.7 LLL-Algorithmus (LLL)

E.7.1 Gitterbasis-Algorithmus - basisreduction.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

              Implementierung des LLL-Algorithmus

   Teil:            Basisreduktionsalgortihmus

   ============================================================

   Autor           :Lars Fabian Paape
   Ort             :Universitaet Bremen
   Datum           :28.09.2003
   Datei           :LLL/basisreduction.par
*/

basisreduction(A)=
{
 ASize  =matsize(A);
 Error  ="Matrixdimension falsch!";
 if ((ASize[1]-ASize[2]),
     errorHandling(Error);,
     \\else
     return (basisreduction_calc(A));
 );
}
\\-------------------------------------------------------------
\\ Hilfeinformation
stMsg="B=basisreduction(A): Bestimmung einer reduzierten";
stMsg=concat(stMsg," Basis B aus einer in Form einer nxn-");
stMsg=concat(stMsg," Matrix A gegebenen Gitterbasis.");

addhelp(basisreduction,StdHelpMessage(stMsg));
\\-------------------------------------------------------------
\\Routine zur Basisreduktion
basisreduction_calc(A)=
{
  \\lokale Variablen
  local(n,m,mh,B,b,bv,bs,BH,mh2);
  local(i,j,k,l,H);
```

```
 \\Initialisierung
 n =(matsize(A))[1];

 \\Gram-Schmidt-Orthogonalisierung
 m =matrix(n,n);
 b =A;
 bs=A;
 B =vector(n);

\\Orthogonalisierung
for(i=1,n,
    for (j=1,i-1,
         m[i,j]=(b[,i]~*bs[,j])/B[j];
         bs[,i]=bs[,i]-m[i,j]*bs[,j];
        );
    B[i]=bs[,i]~*bs[,i];
   );
\\Hauptschleife
k=2;
while (k<n+1,
       \\Bedingung abs(m[k,k-1])<=1/2 erfuellen
       H=MReduction(k,k-1,m,b);
       m=H[1];
       b=H[2];
       \\ Unterscheidung von zwei Faellen
       if (B[k]>=(3/4-m[k,k-1]^2)*B[k-1],
           \\ Fall 1
           \\ es muss nur die Bedingung fuer
           \\ die m's sichergestellt werden
           forstep(l=k-2,1,-1,
                   H=MReduction(k,l,m,b);
                   m =H[1];
                   b=H[2];
                  );
           k++;
           ,
           \\else:
           \\ Fall 2 b[k] und b[k-1]
           \\ muessen gem. Formel getauscht werden,
           \\ die m's,B's muessen angepasst werden
           mh      =m[k,k-1];
           BH      =B[k]+mh^2*B[k-1];
           m[k,k-1]=mh*B[k-1]/BH;
           B[k]    =B[k-1]*B[k]/BH;
```

```
            B[k-1]  =BH;
            \\Berechnung der b,m

            \\tausche  b[k],b[k-1]
            bv     =b[,k];
            b[,k]  =b[,k-1];
            b[,k-1]=bv;

            \\Anpassung der m's
            for(j=1, k-2,
                mh2     =m[k,j];
                m[k,j]  =m[k-1,j];
                m[k-1,j]=mh2;
               );
            for(i=k+1,n,
                mh2     =m[i,k-1]-mh*m[i,k];
                m[i,k-1]=m[i,k]+m[k,k-1]*mh2;
                m[i,k]=mh2;
               );
            if(k>2,k--;);
           );
 );
 return(b);
}
\\-------------------------------------------------------------
\\ Diese Routine stellt sicher, dass die Bedingung
\\ m[k,l]<=1/2 erfuellt ist und passt entsprechend auch
\\ die vorherigen m's an.
MReduction(k,l,m,b)=
{
  \\lokale Variable
  local(r);

  if (abs(m[k,l])>1/2,
      r=round(m[k,l]);
      b[,k]-=r*b[,l];
      for(j=1,l-1,
          m[k,j]-=r*m[l,j];
         );
      m[k,l]-=r;
     );
  return([m,b]);
}
```

E.7.2 Hilfsalgorithmen - auxlll.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ============================================================

              Implementierung des LLL-Algorithmus

   Teil:          Hilfsalgorithmen AUX1 und AUX2 zur direkten
                  Implementierung des LLL-Algorithmus
   ============================================================

   Autor         :Lars Fabian Paape
   Ort           :Universitaet Bremen
   Datum         :17.12.2003
   Datei         :LLL/auxlll.par

   Bemerkung     :
   Fuer diesen Algorithmen ist keine zusaetzlich Fehler-
   behandlung notwendig, da ein Aufruf nur innerhalb des
   LLL-Algorithmus erfolgt.
*/
\\-------------------------------------------------------------
\\ Rueckgabewert: [bH0poss, h0]
\\ bH0poss=1, falls h0 bestimmt werden konnte, sonst 0
aux1(f,h,m,p,k)=
{
  \\ lokale Variablen
  local(i,j,d,t,n,B,pk,h0,testval);
  \\ Matrix B der Gitterbasis erstellen
  B=matrix(m+1,m+1);
  d=poldegree(h);
  n=poldegree(f);
  pk=p^k;
  \\ erster Teil der Basis
  for(i=0,d-1,
      B[i+1,i+1]=pk;
     );
  \\ zweiter Teil der Basis
  hvec=Vec(h);
  for(j=0,m-d,
     for(i=0,d,
         B[i+j+1,d+j+1]=hvec[d-i+1];
        );
     );
```

```
  \\ Bestimmung der reduzierten Basis
  B=basisreduction(B);
  \\ Bestimmung des Polynoms h0
  testval=(pk^d/polnorm(f)^m)^(1/n);
  if (vecnorm(B[,1]~)<testval,
      \\ Bestimmung von t
      t=1;
      for(j=2,m+1,
          if (vecnorm(B[,j]~)<testval,
              t++;
              ,
              break;
             );
         );
      \\ Bestimmung von h0 mittels wiederholter Anwendung
      \\ des Subresultanten-Algorithmus
      h0=Polrev(B[,1]);
      for (i=2,t,
           h0=subresultant_gcd(h0,Polrev(B[,i]));
          );
      return([1,h0]);
      ,
      \\ else: deg(h0)>m
      \\ kein weiteres Ergebnis
      return([0,0]);
     );
}
\\-------------------------------------------------------------
aux2(f,h,p)=
{
  \\ lokale Variablen
  local(d,n,k_val,k,res_lift,g,m,res_aux1,mtest,i,h0,u);

  d=poldegree(h);
  n=poldegree(f);
  mtest=d;
  if(d<n,
     \\ Berechne k
     while(mtest<n,
           k_val=2^(n*mtest/2)*(\
       binomial(2*(mtest),(mtest)))^(n/2)*polnorm(f)^(n+mtest);
           k=1;
           while((p^(k*d))<=k_val,k++;);
           \\ Anpassung von h,
```

```
            \\ so dass  (h mod p^k) | (f mod p^k)
            \\ Realisierung mittels Hensel Lift
            \\ Berechne Rest mod p
            g=centerlift(PolCoeffMod(f,p)/PolCoeffMod(h,p));
            res_lift=lin_lift(f,g,h,p,k);
            \\die Koeffizienten sind red. mod p^k
            h=centerlift(\
            PolCoeffMod(res_lift[2],p^k));
            \\ Berechne ganzzahliges u,
            \\ so dass d<=(n-1)/2^u
            u=0;
            while(d<=((n-1)/2^u),u++;);
            u--;
            forstep(i=u,0,-1,
                    \\ Berechnung von m
                    m=floor((n-1)/(2^i));
                    \\ Algorithmus aux1
                    res_aux1= aux1(f,h,m,p,k);
                    \\ Ergebnis von Algorithmus 1 ?
                    if(res_aux1[1]==1,
                       h0=res_aux1[2];
                       \\return(h0);
                       break(2);
                      );
                  );
            h0=f;
            mtest++;
           );
     ,
     \\else: d==n
     h0=f;
    );
    return(h0);
}
```

E.7.3 Hauptalgorithmus - factor_lll.par

```
/* Faktorisierung von Polynomen mit ganzzahligen Koeffizienten
   ===========================================================

   Implementierung des LLL-Algorithmus

   Teil:         Algorithmus zur Faktorisierung
                 eines ganzzahligen Polynoms nach LLL
```

```
   ============================================================

   Autor        :Lars Fabian Paape
   Ort          :Universitaet Bremen
   Datum        :14.12.2003
   Datei        :LLL/factor_lll.par
*/
\\-----------------------------------------------------------
factor_lll(f)=
{
 \\ Fehlerbehandlung
 Error ="Argument falsch, f Polynom";
 if ((type(f)!="t_POL")&&(type(f)!="t_INT"),
     errorHandling(Error);
     ,
     \\ Berechnung der Faktorisierung
     if ((type(f)=="t_POL")&&(poldegree(f)>=1),
         \\ Faktorisierung:
         \\ Es wird ein nicht konst.
         \\ Polynom erwartet
         return(factor_lll_calc(f));
         ,
         \\ else deg(f)<=0
         return([f,0,[]]);
        );
    );
}
\\-----------------------------------------------------------
\\ Hilfeinformation
stMsg="[l,n,[[d1,fac1],...,[dn,facn]]=factor_lll(f):";
stMsg=concat(stMsg,"Faktorisierung von f  mittels");
stMsg=concat(stMsg," des LLL-Verfahrens wobei");
stMsg=concat(stMsg," f=l*fac1^d1*...facn^dn bzw. f=l, ");
stMsg=concat(stMsg," falls deg(f)<=0. ");

addhelp(factor_lll,StdHelpMessage(stMsg));
\\-----------------------------------------------------------
\\   Algorithmus zur Faktorisierung eines quadratfreien
\\   primitiven Polynoms f mit deg(f)>=1
factor_lllsf_calc(f,n,resultant)=
{
  \\ lokale Variablen
  local(res_fac,f1,f2,res_facmodp);
  local(res_fmp,p,h,h0,i,res_hlp,h0p);
```

```
  \\ Bestimmung von p
  p=2;
  while((resultant%p)==0,
        p=nextprime(p+1);
       );
  \\ Faktorisierung mod p
  res_facmodp =factor_berlekampsf(f,p);
  \\ Umkopieren der Faktoren
  res_fmp=[];
  for(i=1,res_facmodp[2],
      res_fmp=concat(res_fmp,[res_facmodp[3][i][2]]);
     );
  \\ Faktorisierung ueber Z
  f1=1;
  f2=f;
  res_fac=[];
  while(abs(f2)!=1,
        \\ Bestimme h
        \\ Waehle immer den ersten Faktor aus der Liste
        h=res_fmp[1];
        \\ Berechne h0
        h=centerlift(PolCoeffMod(h,p));
        h0=aux2(f2,h,p);
        \\ Anpassung von f1,f2
        f1*=h0;
        f2=f2/h0;

        \\ Faktorenliste anpassen
        res_fac=concat(res_fac,[[1,h0]]);
        \\
        res_hlp=[];
        h0p=PolCoeffMod(h0,p);
        for(i=1,matsize(res_fmp)[2],
            if((h0p%res_fmp[i])!=0,
               res_hlp=concat(res_hlp,[res_fmp[i]]);
              );
           );
        res_fmp=res_hlp;
       );
  \\ f ist komplett faktorisiert
  \\ Rueckgabewert
  return([f2,matsize(res_fac)[2],res_fac]);
}
\\-------------------------------------------------------
```

```
\\   Algorithmus zur Faktorisierung eines
\\   ganzzahligen Polynoms f mit deg(f)>=1
factor_lll_calc(f)=
{
  \\ lokale Variablen
  local(res_sf_fac,n,resultant,g,i,res_div);
  \\ Initialisierung
  n=poldegree(f);
  resultant=subresultant_gcd(f,derivative(f),1);
  \\ Ist f quadratfrei ?
  g=1;
  if (resultant[2]==0,
      g=resultant[1];
      f=f/g;
      \\ neue Resultante notwendig
      resultant=subresultant_gcd(f,derivative(f),1);
     );
 l=cont(f);
 f=pp(f);
 \\ f ist ab hier immer quadratfrei
 \\ und primitiv wg. s.o.
 res_sf_fac=factor_lllsf_calc(f,n,resultant[2]);
 \\  res_sf_fac enthaelt nun Faktorisierung des
 \\  quadratfreien Anteils von f.
 \\  g beruecksichtigen:
 if (g!=1,
     i=1;
     while(poldegree(g)>=1,
           res_div=euklid_division(g,res_sf_fac[3][i][2]);
           if(res_div[2]==0,
              \\ Faktor gefunden
              g=res_div[1];
              res_sf_fac[3][i][1]+=1;
              ,
              \\ else
              \\ naechsten Faktor testen
              i++;
             );
          );
    );
 \\Rueckgabewert
 res_sf_fac[1]*=l*g;
 return(res_sf_fac);
}
```

Literatur

[AKR] Alkiviadis G. Akritas, *Elements of Computer Algebra with Application*, S. 101-154,S. 215-329, John Wiley & Sons, Inc., 1989

[CAS] J. W. S. Cassels, *An Introduction to the Geometry of Numbers*, Springer-Verlag, 1971

[CCS] A. M. Cohen, H. Cuypers & H. Sterk, *Some Tapas of Computer Algebra*, S. 78-90, Springer-Verlag, 1999

[COH] Henri Cohen, *A Course in Computational Algebraic Number Theory*, Springer-Verlag, 2000

[DEC] Wolfram Decker, *Skript: Elementare Grundlagen der Computeralgebra*, Vorlesung Sommersemester 2001, Universität des Saarlandes

[FOR] Otto Forster, *Algorithmische Zahlentheorie*, Vieweg & Sohn, 1996

[FUS] G. Fischer & R. Sacher, *Einführung in die Algebra*, Teubner Studienbücher Mathematik, 1983

[GCL] K. O. Geddes, S. R. Czapor & G. Labahn, *Algorithms for Computer Algebra*, Kluwer Academic Publishers, 1995

[HEA] A. D. Healy,*Resultants, Resolvents and the Computation of Galois Groups*, Math 250a - Higher Algebra, Havard University, 2001, http://www.alexhealy.net/papers/math250a.pdf

[KNU] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, S. 418-455, Addison-Wesley, 1981

[LAN] Serge Lang, *Undergraduate Algebra - Second Edition*, Springer-Verlag, 1990

[LLL] A. K. Lenstra, H. W. Lenstra & L. Lovász, *Factoring Polynomials with Rational Coefficients*, Mathematische Annalen 261, S. 515-534, Springer-Verlag, 1982

[LUT] H. W. Lenstra & R. Tijdeman, *Computational Methods in Number Theory*,Part I, S. 169-195, Mathematisch Centrum Amsterdam, 1982

[MIG] Maurice Mignotte, *Mathematics for Computer Algebra*, Springer-Verlag, 1996

[MI2] Maurice Mignotte, *An Inequality about Factors of Polynomials*, Math. Comp. 28, S. 1153-1157, 1974

[MUS] M. Mignotte & D. Ştefănescu, *Polynomials - An Algorithmic Approach*, Springer-Verlag, 1999

[POH] Michael E. Pohst, *Computational Algebraic Number Theory*, Birkhäuser-Verlag, 1993

[ROS] H. E. Rose, *A Course in Number Theory*, Clarendon Press Oxford, 1988

[SCH] Friedrich Schwarz, *Skript: Computeralgebra*, Vorlesung Wintersemester 1999/2000, Universität-Gesamthochschule Paderborn, http://www–math.uni–paderborn.de/∼fritz/CompAlgebra.html

[PARI-HAV] William A. Stein, *Math 124 – Programming in Pari I & II*, Fall 2001, Harvard University, http://modular.fas.harvard.edu/edu/Fall2001/124/lectures/

[PARI-Tut.] C. Batut, K. Belabas, D. Bernadi, H. Cohen & M. Olivier, *A Tutorial for PARI/GP*, November 2000, http://pari.math.u–bordeaux.fr/

[PARI-User] C. Batut, K. Belabas, D. Bernadi, H. Cohen & M. Olivier, *User's Guide to PARI/GP*, November 2000, http://pari.math.u–bordeaux.fr/

[SIE] Wenke Sietas, *Faktorisierung von Polynomen über endlichen Körpern*, Universität Bremen, Januar 1999, http://alzagk.math.uni–bremen.de/Diplom/Diplwsietas.ps.gz