Algorithmen für Polynome
Vorlesung Sommersemester 2009

Prof. H.-G. Gräbe, Institut für Informatik,
http://bis.informatik.uni-leipzig.de/HansGertGraebe

7. Juli 2009

Dieser Kurs ist eine Abspaltung aus dem früher von mir gelesenen Kurz „Grundlegende Algorithmen der Computeralgebra“. Dort hatte ich sowohl Algorithmen für Zahlen als auch Algorithmen für Polynome betrachtet. Aus Zeitgründen waren die fortgeschrittenen Polynomalgorithmen, insbesondere die Faktorisierung von Polynomen, sehr kurz weggekommen.

Der erste Teil des früheren Kurses ist in erweiterter Form Gegenstand der VL „Algorithmen für Zahlen und Primzahlen“.

1 Polynome – ihre Darstellung und Arithmetik

Begriffe: Polynomring $R[x_1, \ldots, x_n]$, Koeffizientenbereich R (wollen wir immer als kommutativen Ring mit 1 und in den meisten Fällen als Integritätsbereich voraussetzen)

Schreibweise $\mathbf{x}^a$ für das Potenzprodukt $x_1^{a_1} x_2^{a_2} \cdot \ldots \cdot x_n^{a_n}$.
Termmonoid $T = T(x_1, \ldots, x_n) = \{\mathbf{x}^a : \ a_i \in \mathbb{N}\}$.

Dichte und dünne Darstellung

Komplexitätsbetrachtungen hängen vom Kostenmodell für den Grundbereich R ab.

Für Grundbereiche mit beschränkter Koeffizientenlänge $O(1)$ wie etwa $R = \mathbb{R}$ (`float`), $R = \mathbb{C}$ (`complex`) oder $R = \mathbb{Z}_m$ können wir Einheitskosten für die arithmetischen Operationen ansetzen, während für Grundbereiche mit unbeschränkter Koeffizientenlänge wie $R = \mathbb{Q}$, $R = \mathbb{Z}$ oder $R = k[x_1, \ldots, x_n]$ die Bitlänge der entsprechenden Koeffizienten zu berücksichtigen ist.

1.1 Rekursive Darstellung von Polynomen

Als *rekursive Darstellung* eines Polynoms

$$f(x_1, \ldots, x_n) \in R = k[x_1, \ldots, x_n] = k[x_1, \ldots, x_{n-1}][x_n] = R'[x_n]$$

bezeichnet man eine solche Darstellung, die f als Polynom in x_n mit Koeffizienten aus $R' = k[x_1, \ldots, x_{n-1}]$ betrachtet.

Beispiel: Volles symmetrisches Polynom vom Grad 3 in $x_1, \ldots, x_4$ (MuPAD)

```
h:=proc(d,vars) local u;
```

```
begin
  if d=0 then 1
  elif nops(vars)=1 then op(vars,1)^d
  else u:=[op(vars,i)$i=2..nops(vars)];
      expand(h(d,u)+op(vars,1)*h(d-1,vars))
  end_if
end_proc;

UPx1:=Dom::UnivariatePolynomial(x1,Dom::Integer);
UPx2:=Dom::UnivariatePolynomial(x2,UPx1);
UPx3:=Dom::UnivariatePolynomial(x3,UPx2);
UPx4:=Dom::UnivariatePolynomial(x4,UPx3);

vars:=[x1,x2,x3,x4];
uhu:=h(3,vars);
UPx4(uhu);
```

$$x_4^3 + (x_3 + x_2 + x_1)\, x_4^2 + \left(x_3^2 + (x_2 + x_1)\, x_3 + \left(x_2^2 + x_1\, x_2 + x_1^2\right)\right) x_4 \\ + \left(x_3^3 + (x_2 + x_1)\, x_3^2 + \left(x_2^2 + x_1\, x_2 + x_1^2\right) x_3 + \left(x_2^3 + x_1\, x_2^2 + x_1^2\, x_2 + x_1^3\right)\right)$$

Polynomgröße wird bestimmt durch 2 Parameter: Gradschranke $d_n(f) = \deg(f, x_n)$ und Koeffizentengröße $b(f)$, die sich rekursiv aus Gradschranken $d_i(f), i < n$ und der Größe $t(f)$ der „Grundkoeffizienten“ aus k ergibt.

Wir sprechen von der *Gradschranke* $\mathbf{d} = (d_1, \ldots, d_n)$, wenn $d_i(f) < d_i, i = 1, \ldots, n$, gilt.

Die Bitgröße eines solchen Polynoms entspricht etwa der Summe der Bitgrößen der einzelnen Koeffizienten und liegt für dichte Polynome in der Ordnung $O(t \cdot d_1 \cdot \ldots \cdot d_n)$, wobei t die durchschnittliche Bitgröße eines Koeffizienten angibt.

1.2 Distributive Darstellung

Als distributive Darstellung bezeichnet man die Darstellung eines Polynoms $f \in R$ als geordnete Kollektion (etwa als Feld oder Liste) von (Koeffizient-Term)-Paaren mit Koeffizenten aus k und Termen aus T.

Reihenfolge der Terme geht von einer *Termordnung* aus. Das ist eine irreflexive, lineare, transitive Relation zwischen den Termen aus T, die zusätzlich *monoton* ist, d.h. für die

$$\forall\, m_1, m_2, m \quad (m_1 < m_1 \;\Rightarrow\; m \cdot m_1 < m \cdot m_2)$$

gilt.

Beispiel: Volles symmetrisches Polynom vom Grad 5 in x_1, x_2, x_3, x_4, siehe oben, bzgl. der rein lexikographischen Termordnung zu $(x_1 > x_2 > x_3 > x_4)$ geordnet.

$$h_5(x_1, x_2, x_3, x_4) = \\ {x_1}^5 + {x_1}^4\, x_2 + {x_1}^4\, x_3 + {x_1}^4\, x_4 + {x_1}^3\, {x_2}^2 + {x_1}^3\, x_2\, x_3 + {x_1}^3\, x_2\, x_4 + {x_1}^3\, {x_3}^2 + {x_1}^3\, x_3\, x_4 + \\ {x_1}^3\, {x_4}^2 + {x_1}^2\, {x_2}^3 + {x_1}^2\, {x_2}^2\, x_3 + {x_1}^2\, {x_2}^2\, x_4 + {x_1}^2\, x_2\, {x_3}^2 + {x_1}^2\, x_2\, x_3\, x_4 + {x_1}^2\, x_2\, {x_4}^2 +$$

$$x_1{}^2\,x_3{}^3+x_1{}^2\,x_3{}^2\,x_4+x_1{}^2\,x_3\,x_4{}^2+x_1{}^2\,x_4{}^3+x_1\,x_2{}^4+x_1\,x_2{}^3\,x_3+x_1\,x_2{}^3\,x_4+x_1\,x_2{}^2\,x_3{}^2+$$
$$x_1\,x_2{}^2\,x_3\,x_4+x_1\,x_2{}^2\,x_4{}^2+x_1\,x_2\,x_3{}^3+x_1\,x_2\,x_3{}^2\,x_4+x_1\,x_2\,x_3\,x_4{}^2+x_1\,x_2\,x_4{}^3+x_1\,x_3{}^4+$$
$$x_1\,x_3{}^3\,x_4+x_1\,x_3{}^2\,x_4{}^2+x_1\,x_3\,x_4{}^3+x_1\,x_4{}^4+x_2{}^5+x_2{}^4\,x_3+x_2{}^4\,x_4+x_2{}^3\,x_3{}^2+x_2{}^3\,x_3\,x_4+$$
$$x_2{}^3\,x_4{}^2+x_2{}^2\,x_3{}^3+x_2{}^2\,x_3{}^2\,x_4+x_2{}^2\,x_3\,x_4{}^2+x_2{}^2\,x_4{}^3+x_2\,x_3{}^4+x_2\,x_3{}^3\,x_4+x_2\,x_3{}^2\,x_4{}^2+$$
$$x_2\,x_3\,x_4{}^3+x_2\,x_4{}^4+x_3{}^5+x_3{}^4\,x_4+x_3{}^3\,x_4{}^2+x_3{}^2\,x_4{}^3+x_3\,x_4{}^4+x_4{}^5$$

Auch in diesem Fall können wir die Polynome betrachten, deren Terme durch eine Gradschranke $\mathbf{d} = (d_1, \ldots, d_n)$ und deren Koeffizienten-Bitgröße durch eine Schranke t begrenzt sind. Für die Algorithmen dieser Vorlesung wird ausschließlich die rekursive Darstellung von Polynomen eine Rolle spielen.

1.3 Komplexitätsbetrachtungen

Ein Polynom f mit der Gradschranke $\mathbf{d}$ hat höchstens $D = D(f) = d_1 \cdot \ldots \cdot d_n$ Terme, also selbst eine maximale Bitlänge $L(f) = t \cdot D$. Das gilt sowohl für die rekursive als auch die distributive Darstellung.

Komplexität wollen wir deshalb in folgendem Ansatz betrachten. Wir gehen aus von einem Polynomring $R = A[x]$ und Polynomen $f \in R$ mit $\deg(f) < d$, wobei A selbst wieder ein kommutativer Ring mit Eins und in den meisten Anwendungsfällen ein Integritätsbereich ist, also nullteilerfrei. In diesem Fall können wir den Quotientenkörper $K = Q(A)$ bilden und $f \in A[x]$ auch als Polynom im Euklidschen Ring $K[x]$ betrachten. Dies wird von Fall zu Fall genauer anzumerken sein. Wir gehen davon aus, dass in A neben effektiven Ringoperationen $+$ und $*$ ein boolesches Prädikat `iszero` definiert ist, mit dem sich effektiv (mit konstanten Kosten $O(1)$) für $c \in A$ die Frage $c = 0$ entscheiden lässt.

Diesen Ansatz wenden wir rekursiv auf die Setzung $R = R_m,\ A = R_{m-1},\ x = x_m,\ d = d_m$ für $m = 1, \ldots, n$ an, wobei wir $A_0 = k$ als den *Grundbereich* bezeichnen. In den meisten Anwendungen gilt $k \in \{\mathbb{Z}, \mathbb{Z}_m, \mathbb{Q}\}$. Für die theoretischen Untersuchungen wollen wir annehmen, dass k ein Körper und A eine k-Algebra ist. Dies bedeutet, dass A neben der Ringstruktur auch noch die Struktur eines k-Vektorraums trägt. Die entsprechenden Vektorraum-Operationen (Addition, k-skalare Vervielfachung) sind sowohl in der rekursiven als auch der distributiven Darstellung der Polynome besonders einfach auszuführen.

Ein Polynom $f \in A[x],\ f \neq 0$ hat eine *eindeutige* Darstellung

$$f = c_p \cdot x^p + c_{p-1} \cdot x^{p-1} + \cdots + c_0$$

mit den *Koeffizienten* $c_i \in A$, wenn deren Darstellung eindeutig ist. Diese Eigenschaft der eindeutigen Darstellung vererbt sich von k auf die Polynomringe $R_1, \ldots R_n$ und wird als *kanonische rekursive Darstellung* des Polynoms f bezeichnet. Wir wollen insbesondere davon ausgehen, dass die Algorithmen Polynome in dieser kanonischen Darstellung zurückliefern, womit die Frage der Existenz eines effektiven Prädikats `iszero` auf die Existenz eines solchen Prädikats im Grundbereich k reduziert ist. Für die Grundbereiche $k \in \{\mathbb{Z}, \mathbb{Z}_m, \mathbb{Q}\}$ existieren kanonische Darstellungen der Elemente, insbesondere auch des Nullelements (über $\mathbb{Q}$ ist dies etwa die Darstellung $\frac{0}{1}$). Für allgemeinere Grundbereiche kann der Nulltest deutlich schwieriger sein bis hin zur algorithmischen Unentscheidbarkeit.

$p = \max(k : c_k \neq 0)$ bezeichnet man als den *Grad* $p = \deg(f)$ des Polynoms $f \neq 0$, c_p als dessen *Leitkoeffizienten*. Grad und Leitkoeffizient des Nullpolynoms sind unbestimmt.

Für ein Polynom $f \in R = A[x]$ vom Grad $\deg(f) < d$ erhalten wir die Bitlänge $L_R(f) \lesssim L_A(f) \cdot d$, wobei $L_A(f)$ die (je nach Komplexitätsmodell maximale bzw. durchschnittliche) Bitlänge der Koeffizienten von f angibt.

> $L_A(t;d) = O(t\,d)$ gibt dann eine Schranke an für die Bitlänge von Polynomen $f \in A[x]$ mit $\deg(f) < d,\ L_A(f) \le t$. Wir bezeichnen diese Klasse von Polynomen mit $\mathcal{C}_A(t;d)$.

Für $f \in k[x_1,\dots,x_n],\ f \neq 0$ mit der distributiben Darstellung $f = \sum_\alpha c_\alpha \mathbf{x}^\alpha$ bezeichnen wir den maximalen Grad $p_i = \deg_i(f) = \max(\alpha_i : c_\alpha \neq 0)$ der Variablen x_i, welche in einem Term von f vorkommt, als *x_i-Grad*.

Im rekursiven Ansatz $R = R_n,\ A = R_{n-1}$ betrachten wir die Klasse der Polynome $f \in k[x_1,\dots,x_n]$, welche durch das Tupel $(t;d_1,\dots,d_n)$ charakterisiert wird, wobei t für eine Schranke der Bitlänge der Koeffizienten aus dem Grundbereich steht und $d_i > \deg_i(f)$ gilt. Wir bezeichnen diese Klasse von Polynomen mit $\mathcal{C}_k(t;d_1,\dots,d_n)$.

Aus obiger Formel erhalten wir rekursiv

$$L(t;d_1,\dots,d_n) = O(t \cdot d_1\ \dots \cdot d_n)$$

als Schranke für die Bitlänge von Polynomen $f \in \mathcal{C}_k(t;d_1,\dots,d_n)$.

Wir wollen eine Längenfunktion l auf A als *additiv* bezeichnen, wenn $l(a_1 \cdot a_2) \sim l(a_1) + l(a_2)$ für $a_1, a_2 \in A$ gilt. Die Bitlänge auf $A = \mathbb{Z}$ sowie der Gradvektor $d(f) = (\deg_i(f),\ i = 1,\dots,n)$ auf $A = k[x_1,\dots,x_n]$ sind solche additiven Längenfunktionen.

Für die Komplexität der Addition zweier Polynome $f, g \in \mathcal{C}_A(t;d)$ ergibt sich

$$C_R^+(t;d) \le C_A^+(t) \cdot d$$

und rekursiv für $f, g \in \mathcal{C}_k(t;d_1,\dots,d_n)$

$$C_R^+(t;d_1,\dots,d_n) \le C_k^+(t) \cdot (d_1 \cdot \ldots \cdot d_n)$$

sowie für die (klassische) Multiplikation zweier Polynome $f, g \in \mathcal{C}_A(t;d)$

$$C_R^*(t;d) \le C_A^*(t) \cdot d^2$$

und rekursiv für $f, g \in \mathcal{C}_k(t;d_1,\dots,d_n)$

$$C_R^*(t;d_1,\dots,d_n) \le C_k^*(t) \cdot (d_1^2 \cdot \ldots \cdot d_n^2)$$

Für Grundbereiche mit beschränkter Elementlänge $t \in O(1)$ und damit konstanten Kosten für die Arithmetik erhalten wir daraus

$$C_R^+(1;d_1,\dots,d_n) \sim d_1 \cdot \ldots \cdot d_n$$
$$C_R^*(1;d_1,\dots,d_n) \sim (d_1 \cdot \ldots \cdot d_n)^2,$$

während für den Grundbereich $\mathbb{Z}$ und Koeffizienten mit einer Bitlänge kleiner als t für die Kosten der klassischen Verfahren $C_k^+(t) \sim t$ und $C_k^*(t) \sim t^2$ gilt, so dass sich insgesamt

$$C_R^+(t;d_1,\dots,d_n) \sim t \cdot d_1 \cdot \ldots \cdot d_n$$
$$C_R^*(t;d_1,\dots,d_n) \sim t^2 \cdot (d_1 \cdot \ldots \cdot d_n)^2$$

ergibt. Wir können auf dieser Basis die auch durch vielfältige Erfahrungen mit konkreten Rechnungen bestätigte Regel formulieren:

Der Aufwandszuwachs beim Übergang von einem Koeffizientenbereich mit konstanten Arithmetikkosten (`float` oder modular) zum Koeffizientenbereich $\mathbb{Z}$ kann mit dem Aufwandszuwachs bei der Vergrößerung der Anzahl der Variablen um 1 gleichgesetzt werden.

1.4 Schnelle Multiplikationsverfahren

Multiplizieren und Quadrieren

Bei den bisherigen Komplexitätsbetrachtungen für die Multiplikation haben wir das klassische Verfahren der termweisen Multiplikation zu Grunde gelegt. Die Aussagen lassen sich unmittelbar verfeinern, wenn die Faktoren verschiedenen Komplexitätsklassen $f \in \mathcal{C}_A(t;d)$, $g \in \mathcal{C}_A(t';d')$ angehören.

Wir wollen nun schnellere Multiplikationsverfahren für Polynome f, g aus derselben Komplexitätsklasse $\mathcal{C}_A(t;d)$ kennenlernen.

Dieser Fall ist etwa für die Berechnung von f^2 interessant. In der Tat stellt sich heraus, dass jedes schnelle Verfahren zum Quadrieren zu einem schnellen Verfahren der Multiplikation führt und umgekehrt. In der Tat, ist $M(t;d) = C^*_R(t;d)$ eine Schranke für das Multiplizieren in der Klasse $\mathcal{C}_A(t;d)$ und $Q(t;d) = C^Q_R(t;d)$ eine Schranke für das Quadrieren, so gilt einerseits $Q(t;d) \leq M(t;d)$, da Quadrieren durch einfaches Multiplizieren besorgt werden kann, andererseits aber auch $M(t;d) \leq 3 \cdot Q(t;d)$, denn das Produkt $f \cdot g$ kann aus

$$(f+g)^2 = f^2 + 2\,f\,g + g^2 \quad \Rightarrow \quad 2\,f\,g = (f+g)^2 - f^2 - g^2$$

durch drei Quadratberechnungen (und einige Additionen) in der Klasse $\mathcal{C}_A(t;d)$ (beachten Sie $f, g \in \mathcal{C}_A(t;d) \Rightarrow f + g \in \mathcal{C}_A(t;d)$) berechnet werden.

Satz 1 *Quadrieren und Multiplizieren „gleichgroßer“ Polynome sind zueinander äquivalente algorithmische Aufgaben.*

Nichtskalare Komplexität

Wir sehen an diesem Beispiel zugleich, dass es bei der Untersuchung der Gleichwertigkeit von Verfahren oftmals nicht auf die Zahl der zusätzlich erforderlichen Additionen in $R = A[x]$ oder A ankommt, da diese Operationen vergleichsweise „billig“ sind. Die Summe der Polynome $f, g \in R$ wird durch termweise Addition der Koeffizienten berechnet und liegt damit im k-Vektorraum, der von den Elementen f, g in der k-Algebra R aufgespannt wird. Rechenschritte, die sich durch solche k-linearen Kombinationen darstellen lassen, bezeichnen wir als *skalare Operationen*. Als *nichtskalare Komplexität* bezeichnet man jede Aufwandsrechnung, in welcher skalare Operationen nicht, sondern im Wesentlichen nur die ausgeführten Multiplikationen in A gezählt werden. Derartige Aussagen sind insbesondere dann interessant, wenn die auszuführenden Multiplikationen in A durch eine uniforme Komplexitätsschranke begrenzt sind, sich eine Schranke für die die Gesamtkomplexität also als Produkt aus dieser Schranke und der Zahl der auszuführenden Multiplikationen in A ergibt.

Wie oben hergeleitet unterscheiden sich die nichtskalaren Komplexitäten der Multiplikation und des Quadrierens in der Klasse $\mathcal{C}_A(t;d)$ höchstens um den Faktor 3.

Schnelles Quadrieren

Das Quadrat eines Polynoms $f \in R = A[x]$ vom Grad $\deg(f) < d = 2\,l$ kann bereits durch *drei* Multiplikationen von Polynomen vom Grad $< l$ ausgeführt werden. Wir zerlegen dazu $f = f_1 \cdot x^l + f_2$ mit $\deg(f_1), \deg(f_2) < l$ und berechnen

$$f^2 = \left(f_1 \cdot x^l + f_2\right)^2 = f_1^2 \cdot x^{2\,l} + 2\,f_1\,f_2 \cdot x^l + f_2^2$$

Es gilt also $Q(t; 2\,l) = 3 \cdot Q(t; l)$ plus skalare Operationen, denn das Ergebnis lässt sich aus den Teilergebnissen f_1^2, $f_1\,f_2$ und f_2^2 durch Gradshifts und Additionen berechnen.

Ist $A = k$ ein Körper mit Einheitskostenarithmetik, setzen wir $Q(1; d) = Q(d)$ für die Kosten des Quadrierens eines Polynoms $f \in k[x]$ vom Grad $\deg(f) < d$ und wählen m so, dass $2^{m-1} < d \le 2^m$ gilt, so erhalten wir

$$\begin{aligned} Q(d) \le Q\left(2^m\right) &= 3 \cdot Q\left(2^{m-1}\right) = 3^2 \cdot Q\left(2^{m-2}\right) = \cdots = 3^m \cdot q_k \\ &= \left(2^{\frac{\log(3)}{\log(2)}}\right)^m \cdot q_k = (2^m)^{\frac{\log(3)}{\log(2)}} \cdot q_k \sim d^{\frac{\log(3)}{\log(2)}} \end{aligned}$$

mit $q_k \sim O(1)$ als Kosten des Quadrierens in k.

Wir haben damit die folgende Aussage bewiesen:

Satz 2 *Zum Quadrieren eines Polynoms $f \in k[x]$ vom Grad $\deg(f) < d$ werden maximal $O\left(d^\alpha\right)$ Multiplikationen benötigt. Hier ist $\alpha = \frac{\log(3)}{\log(2)} \approx 1.58 < 2$.*

Die Komplexität des Quadrierens über einem Körper k mit Einheitskostenarithmetik ist also höchstens von der Ordnung $O\left(d^\alpha\right)$.

Die Karatsuba-Multiplikation

Zusammen mit obiger Äquivalenz von Multiplizieren und Quadrieren lässt sich diese Idee unmittelbar auf die Berechnung beliebiger Produkte von Polynomen $f, g \in k[x]$ mit gleicher Grad $\deg(f)$, $\deg(g) < d$ übertragen. Dieser Algorithmus wird als *Karatsuba-Multiplikation* bezeichnet.

Idee: Sind $f, g \in R = k[x]$ Polynome mit $\deg(f)$, $\deg(g) < d = 2\,l$, so zerlegen wir sie in

$$f = f_1 \cdot x^l + f_2, \quad g = g_1 \cdot x^l + g_2$$

mit Polynomen f_1, f_2, g_1, $g_2 \in R$ vom Grad kleiner l und erhalten

$$f \cdot g = (f_1\,g_1)\,x^{2l} + (f_1\,g_2 + f_2\,g_1)\,x^l + (f_2\,g_2) \tag{K.1}$$

Die drei Klammerausdrücke kann man mit *drei* Multiplikationen von Polynomen vom Grad kleiner l berechnen wegen

$$(f_1\,g_2 + f_2\,g_1) = (f_1 + g_2)\,(f_1 + g_2) - f_1\,g_1 - f_2\,g_2\,. \tag{K.2}$$

Die zusätzlichen Additionen sind von linearer Komplexität im Grad, also deutlich billiger.

Komplexität: Bezeichnet $C_{\text{Karatsuba}}(l)$ die Laufzeit für die Multiplikation zweier Polynome vom Grad $< l$ mit dem Karatsuba-Verfahren, so gilt

$$C_{\text{Karatsuba}}(2\,l) = 3\,C_{\text{Karatsuba}}(l),$$

wenn man nur die Multiplikationen berücksichtigt (nichtskalare Komplexität) und

$$C_{\text{Karatsuba}}(2\,l) = 3\,C_{\text{Karatsuba}}(l) + 8\,l,$$

wenn auch die Additionen[1] berücksichtigt werden. In beiden Fällen erhält man wie oben

$$C_{\text{Karatsuba}}(d) = O\left(d^{\alpha}\right) .$$

Allerdings wird dieses Verfahren in der Praxis selten angewendet, weil man es dort überwiegend mit dünnen Polynomen verschiedener Grade zu tun hat.

Die schnelle Fourier-Transformation

Von theoretischem Interesse ist ein noch schnelleres Verfahren zur Multiplikation von zwei Polynomen $a = \sum a_i x^i, b = \sum b_i x^i \in A[x]$ vom Grad $\deg(a), \deg(b) \le d$, das mit $O(d \log(d))$ Operationen aus A auskommt.

Die grundlegende Idee dieses Verfahrens besteht darin, die Polynome an genügend vielen Werten $\lambda \in A$ zu evaluieren und aus diesen Werten $c = a \cdot b$ durch Interpolation zu gewinnen.

Satz 3 *Ist A ein Körper, so gibt es genau ein Polynom $f(x) \in A[x]$ vom Grad $\deg(f) \le d$, so dass für vorgegebene voneinander verschiedene Argumente $\lambda_0, \ldots, \lambda_d \in A$ und Funktionswerte $h_0, \ldots, h_d \in A$ die Beziehung $f(\lambda_i) = h_i$ gilt.*

Beweis: Dieses Polynom kann über die Lagrange-Interpolationsformel

$$f = c_d\,x^d + c_{d-1}x^{d-1} + \cdots + c_0 = \sum_{i=0}^{n} \left(\prod_{j \neq i} \frac{x - \lambda_j}{\lambda_i - \lambda_j} \right) \cdot h_i$$

berechnet werden. □

Bemerkung: Die Formel bleibt gültig, wenn A ein Ring und $\lambda_0, \ldots, \lambda_d \in K$ aus einem Teilkörper $K \subset A$ gewählt sind.

Die Lagrange-Formel liefert das Polynom f nicht in seiner Normalform zurück, so dass die Frage steht, diese Normalform zu berechnen. Wir können diese Frage allgemein – ohne Rückgriff auf die Interpolationsformel – beantworten, denn es handelt sich um ein Problem der linearen Algebra. Zwischen dem Vektor der Koeffizienten $\mathbf{c} = \begin{pmatrix} c_0 & c_1 & \ldots & c_d \end{pmatrix}$ und dem Vektor der Funktionswerte $\mathbf{h} = \begin{pmatrix} h_0 & h_1 & \ldots & h_d \end{pmatrix}$ besteht der Zusammenhang

$$\begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_d \end{pmatrix} = \begin{pmatrix} 1 & \lambda_0 & \lambda_0^2 & \ldots & \lambda_0^d \\ 1 & \lambda_1 & \lambda_1^2 & \ldots & \lambda_1^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_d & \lambda_d^2 & \ldots & \lambda_d^d \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{pmatrix} ,$$

[1] Zwei Additionen in $\mathcal{C}_k(l)$, zwei Additionen in $\mathcal{C}_k(2\,l)$ in (K.2) sowie Additionen in Überlappungsbereichen der Länge $2\,l$ in (K.1).

wobei die Übergangsmatrix $U = (\lambda_i^j)_{0\le i,j\le d}$ eine van der Mondesche Matrix und damit nichtsingulär ist. Um den Koeffizientenvektor $\mathbf{c}$ und damit die kanonische Darstellung von f aus dem Vektor der Funktionswerte $\mathbf{h}$ zu berechnen, muss also nur U^{-1} bestimmt und anschließend $\mathbf{c} = U^{-1} \cdot \mathbf{h}$ berechnet werden. Die Berechnung von U^{-1} ist überdies nur einmal pro Stützstellen-Argumente-Tupel erforderlich.

Sind die Stützstellen $\lambda_0, \ldots, \lambda_d$ aus einem Teilkörper $K \subset A$ gewählt, so gilt $U \in GL(d+1, K)$ und U^{-1} kann über K bestimmt werden. Die Elemente $c_j \in A$ ergeben sich dann als K-lineare Kombinationen der Elemente $h_i \in A$. Dies kann interessant sein, wenn die Arithmetikkosten in K deutlich geringer sind als die Arithmetikkosten in A, etwa im Fall $A = k[x_1, \ldots, x_{n-1}]$ für $K = k$.

Wir wollen nun das Produkt $c = a \cdot b$ der Polynome $a, b \in A[x]$ vom Grad $\deg(a), \deg(b) \le d$ bestimmen, indem wir a, b an genügend vielen Werten $\lambda \in A$ evaluieren und aus diesen Werten $c = a \cdot b$ durch Interpolation gewinnen.

Für ein solches $\lambda \in A$ gilt

$$c(\lambda) = \sum c_l \lambda^l = \left(\sum a_i \lambda^i\right) \left(\sum b_j \lambda^j\right) = a(\lambda) \cdot b(\lambda)\,.$$

Jeder Wert $c(\lambda)$ kann also durch eine einzige Multiplikation berechnet werden, wenn die Funktionswerte $a(\lambda)$ und $b(\lambda)$ berechnet sind.

Unabhängige Berechnungen der $a(\lambda)$ nach dem Hornerschema benötigen d Multiplikationen pro Argument λ, also d^2 Multiplikationen, wenn die Argumente λ_i unabhängig voneinander gewählt werden. Damit haben wir allein durch diese vorbereitenden Kosten die Kosten der klassischen Multiplikation zweier Polynome erreicht und nichts gewonnen.

Für ein effizientes Multiplikationsverfahren können wir jedoch im bisher besprochenen Ansatz noch die Nullstellen λ_i genauer festlegen. Es stellt sich heraus, dass zur Multiplikation von Polynomen bis zum Grad d dafür N-te Einheitswurzeln mit $N > 2\,d$ besonders gut geeignet sind.

Betrachten wir zunächst den Fall $A = \mathbb{C}$ und wählen $\omega \in \mathbb{C}$ als eine solche primitive N-te Einheitswurzel. Für eine solche Einheitwurzel gilt der folgende

Lemma 1 (Kürzungssatz)

$$\sum_{j=0}^{N-1} \omega^{js} = \begin{cases} 0 & \text{für } s \not\equiv 0 \pmod N \\ N & \text{für } s \equiv 0 \pmod N \end{cases}$$

(Beweis eines allgemeineren Resultats später)

Die Abbildung

$$D_N :\ \mathbb{C}[x] \longrightarrow \mathbb{C}^N \qquad \text{via } f \mapsto (f(\omega^i), i = 0, \ldots, N-1)$$

ist ein Algebra-Homomorphismus, wenn man die rechte Seite mit der komponentenweisen Algebrastruktur versieht. Der Kern besteht aus all denjenigen Polynomen $f(x)$, die $\omega^i, i = 0, \ldots, N-1$, als Nullstellen besitzen, also Vielfache von $x^N - 1$ sind. Nach dem Isomorphiesatz ist also die Abbildung

$$D_N :\ S = \mathbb{C}[x]/(x^N - 1) \longrightarrow \mathbb{C}^N$$

ein Algebra-Isomorphismus von S in die Algebra der Vektoren der Größe N, in der Multiplikation mit $O(N)$ Operationen ausführbar ist.

Der Faktorring S, der aus $\mathbb{C}[x]$ durch Anwendung der zusätzlichen Reduktionsregel $x^N \mapsto 1$ entsteht, ist eine endlichdimensionale A-Algebra, wobei das Produkt zweier Polynome jeweils vom Grad d wegen $2\,d < N$ von einer solchen Reduktion nicht betroffen wird. Wir können das gesuchte Produkt also auch in S ausrechnen.

Für zwei Polynome $a, b \in S$, die jeweils durch ihre Koeffizientenvektoren gegeben sind, kann man dieses Produkt also als

$$a \cdot b = D_N^{-1}(D_N(a) \cdot D_N(b))$$

berechnen. Die lineare Abbildung D_N, die *(zyklische) diskrete Fouriertransformation*, wird dabei durch Multiplikation mit einer N-reihigen Matrix

$$DFT_N(\omega) = (\omega^{ij})_{0 \le i,j < N}$$

beschrieben. Es stellt sich heraus, dass man ein solches Produkt für diese spezielle Matrix besondes effizient berechnen kann: Für $N = 2M$ und $a(x) = \sum_{0 \le j < N} a_j\, x^j$ gilt

$$a(\omega^i) = \sum_{0 \le j < N} a_j \omega^{ij} = \left(\sum_{0 \le j < M} a_{2j} (\omega^2)^{ij} \right) + \omega^i \left(\sum_{0 \le j < M} a_{2j+1} (\omega^2)^{ij} \right)$$

und somit (mit $\omega^M = -1$)

$$DFT_N(\omega) \cdot a = \begin{pmatrix} DFT_M(\omega^2) & \Delta_M\, DFT_M(\omega^2) \\ DFT_M(\omega^2) & -\Delta_M\, DFT_M(\omega^2) \end{pmatrix} \cdot \begin{pmatrix} g_a \\ u_a \end{pmatrix}.$$

Dabei ist

$$\Delta_M := diag(1, \omega, \omega^2, \ldots, \omega^{M-1})$$

eine $M \times M$–Diagonalmatrix (der Twist-Faktoren), $g_a = (a_i)_{i \equiv 0 \pmod 2}$ der Vektor der Komponenten mit geradem Index und $u_a = (a_i)_{i \equiv 1 \pmod 2}$ der mit ungeradem Index. Um $DFT_N(\omega) \cdot a$ zu berechnen genügt es also, $G := DFT_M(\omega^2) \cdot g_a, U := DFT_M(\omega^2) \cdot u_a$ und $T := \Delta_M \cdot U$ zu berechnen. Dieser Ansatz wird allgemein als die *schnelle Fouriertransformation* bezeichnet. Eine Laufzeitanalyse ergibt nämlich das folgende Ergebnis: Bezeichnet $T(N)$ die arithmetischen Kosten zur Berechnung von $DFT_N(\omega) \cdot a$, so erhalten wir die Rekursionsformel

$$T(2\,M) \le 2\,T(M) + 3\,M - 1$$

und damit $T(N) \le 1.5\,N\,\log_2(N) - N + 1 = O(N\,\log(N))$. Da die inverse Matrix (nachrechnen!)

$$DFT_N(\omega)^{-1} = \frac{1}{N}(\omega^{-ij})_{0 \le i,j < N} = \frac{1}{N} DFT_N(\omega^{-1})$$

bis auf einen skalaren Faktor ebenfalls eine DFT-Matrix ist, kann man die inverse Fouriertransformation mit derselben Geschwindigkeit berechnen. Wir haben damit den folgenden Satz bewiesen

Satz 4 *Das Produkt zweier Polynome vom Grad d mit komplexen Koeffizienten kann mit $O(d\log(d))$ arithmetischen Operationen berechnet werden.*

Dieser Satz ist vor allem für numerische Anwendungen interessant, setzt er doch voraus, dass man mit komplexen Zahlen in Einheitskosten rechnen kann. Dies ist in einer exakten Arithmetik nicht möglich. Allerdings beruht der Ansatz im Wesentlichen allein auf der Eigenschaft, dass $\omega \in k$ eine N-te Einheitswurzel ist. Solche Elemente finden sich auch in anderen algebraischen Strukturen, in denen eine exakte Arithmetik existiert.

Definition 1 Sei N eine positive Zahl. Ein Element $\omega \in A$ heißt *N-te Hauptwurzel*, wenn $\omega^N = 1$ gilt und $1-\omega^k$ für alle $0 < k < N$ Nichtnullteiler in A ist.

Insbesondere ist dann ω eine *primitive* N-te Einheitswurzel, d.h. $\omega^k \neq 1$ für $0 < k < N$. Für einen Körper fallen diese beiden Begriffe zusammen.

Lemma 2 (Verallgemeinerter Kürzungssatz)
Für eine N-te Hauptwurzel ω gilt

$$\sum_{i=0}^{N-1} \omega^{im} = \begin{cases} N & \text{wenn } N \mid m \\ 0 & \text{sonst} \end{cases} \tag{1}$$

sowie für die Ideale in $A[X]$ (was nicht unbedingt ein Hauptidealring sein muss)

$$\bigcap_{i=0}^{N-1} Id(X-\omega^i) = Id\left(\prod_{i=0}^{N-1}(X-\omega^i)\right) \tag{2}$$

und damit auch

$$\prod_{i=0}^{N-1}(X-\omega^i) = X^N - 1 \tag{3}$$

Beweis:
(1) Multipliziere mit $(1-\omega^m)$.
(2) Zeige mit Induktion nach j

$$\bigcap_{i=0}^{j} Id(X-\omega^i) \subseteq Id\left(\prod_{i=0}^{j}(X-\omega^i)\right)$$

Setze dazu in

$$a(X)\prod^{j-1}(X-\omega^i) = b(X)\cdot(X-\omega^j)$$

$X = \omega^j$ und verwende, dass $1-\omega^i$ und ω (als Einheit) Nichtnullteiler sind, um zu sehen, dass ω^j Nullstelle von $a(X)$ ist.
(3) $X^N - 1$ liegt im Idealdurchschnitt, ist also ein Vielfaches der LHS. □

Satz 5 *Sei $N \in \mathbb{N}$ so gewählt, dass $N \cdot 1_A \in A$ eine Einheit in der kommutativen k-Algebra A ist, und sei $\omega \in A$ eine N-te Hauptwurzel. Dann ist*

$$\phi : A[X] \longrightarrow A^N \qquad \textit{via } f \mapsto (f(\omega^i))_{0\le i<N}$$

ein surjektiver Algebra-Homomorphismus mit dem Kern $Id(X^N - 1)$. Den induzierten A-Algebraisomorphismus

$$D_N : A[X]/Id(X^N - 1) \longrightarrow A^N$$

bezeichnet man als die zu ω gehörende diskrete Fourier-Transformation (DFT). *In Bezug auf die kanonischen Basen wird diese lineare Transformation beschrieben durch die Matrix*

$$DFT_N(\omega) := (\omega^{pq})_{0 \le p,q < N} \in Gl(N, A).$$

Die inverse Transformation wird durch die Matrix

$$DFT_N(\omega)^{-1} = \frac{1}{N} DFT_N(\omega^{-1})$$

gegeben. Für $N = 2^n$ gibt es darüber hinaus einen rekursiven Algorithmus, die schnelle Fourier-Transformation (FFT)*, der $DFT_N(\omega) \cdot a$ für einen beliebigen Eingabevektor $a \in A^N$ mit maximal $1.5\, N \log_2(N) - N + 1$ Additionen von Elementen aus A oder Multiplikationen mit Potenzen von ω berechnet.*

Beweis: ϕ ist offensichtlich ein Morphismus mit dem Kern

$$Ker\ \phi = \bigcap_{0 \le i < N} Id(X - \omega^i) = Id(X^N - 1)$$

nach obigem Lemma. Der oben analysierte Algorithmus berechnet dann auch im allgemeinen Fall die genannten Produkte. □

Damit ist auch die folgende Verallgemeinerung obigen Satzes über die totale Komplexität der Multiplikation zweier Polynome über einem allgemeinen Koeffizientenbereich richtig:

Satz 6 *Sei N die kleinste Zweierpotenz größer als n und A eine kommutative Algebra über einem Körper k mit $char(k) \neq 2$. Wenn A eine N-te Hauptwurzel ω enthält, so kann man das Produkt zweier Polynome $a, b \in A[X]$ mit $deg(a\,b) = n$ mit $O(n \log(n))$ arithmetischen Operationen berechnen.*

Ist l eine additive Längenfunktion auf A und gilt $L_A(a), L_A(b) \le t$, so liegt die Berechnung von $a, b \in A[X]$ mit $deg(a\,b) = n$ in der Komplexitätsklasse $O\left(n \log(n) \cdot C_A^+(t) + n \cdot C_A^*(t)\right)$, wobei $C_A^+(t)$ die Kosten einer skalaren Operation auf Elementen von A der Länge t angibt.

Der letzte Teil des Satzes ergibt sich, weil die N-te Hauptwurzel ω in den Rechnungen fixiert ist, deren Länge also mit $O(1)$ in die Komplexitätsberechnung eingeht.

Im Fall $char(k) = 2$ kann man eine dreireihige Fouriertransformation mit dem Ansatz $N = 3M$ verwenden und hat auf dieser Basis ebenfalls eine Polynom-Multiplikation mit $O(n \log(n))$ Operationen.

2 Algorithmen der linearen Algebra über einem Polynomring

In diesem kurzen Abschnitt wollen wir allgemeiner Fragestellungen der linearen Algebra über einem arithmetischen Grundbereich R betrachten wie etwa die Berechnung von Rang, Determinante oder der Inversen einer quadratischen Matrix oder das Lösen linearer Gleichungssysteme. Algorithmische Verfahren für all diese Aufgaben lassen sich auf den Gaußalgorithmus

zurück führen, wie aus dem Grundkurs Algebra bekannt ist. Diese Verfahren sind zugleich oft die effizientesten Verfahren zur Lösung der genannten Aufgaben.

Entsprechende Komplexitätsbetrachtungen hängen vom Kostenmodell für den Grundbereich R ab. Für $R = \mathbb{R}$ oder $R = \mathbb{Z}_m$ können wir Einheitskosten für die arithmetischen Operationen ansetzen, während für $R = \mathbb{Q},\ R = \mathbb{Z}$ oder $R = k[x_1, \ldots, x_n]$ die Bitlänge der entsprechenden Koeffizienten zu berücksichtigen ist.

2.1 Der Gaußalgorithmus unter Einheitskostenarithmetik

Erinnern wir uns, wie Matrizen Schritt für Schritt mit Hilfe von Pivotelementen auf Dreiecksform gebracht werden. Sei dazu M eine zufällige vierreihige Matrix, die wir mit MuPAD und folgender Funktion erzeugen:

```
export(linalg):
randmat:=proc(n,m,D) // n=size m=magnitude
  local r;
begin
  r:=random(m);
  Dom::Matrix(D)(n,n,(i,j)->r());
end_proc;

M4:=randmat(4,10^2,Dom::Float);
```

$$\begin{pmatrix} 41.0 & 56.0 & 95.0 & 23.0 \\ 24.0 & 93.0 & 19.0 & 26.0 \\ 50.0 & 6.0 & 70.0 & 35.0 \\ 5.0 & 16.0 & 36.0 & 66.0 \end{pmatrix}$$

Eine allgemeine Prozedur für Schritt i in diesem Verfahren hat folgende Gestalt:

```
rstep:=proc(A,i) local n,j,k;
begin
  n:=nrows(A);
  for k from i+1 to n do A[i,k]:=A[i,k]/A[i,i] end;
  A[i,i]:=1;
  for j from i+1 to n do
    for k from i+1 to n do
      A[j,k]:=A[j,k]-A[i,k]*A[j,i]
    end
  end;
  for j from i+1 to n do A[j,i]:=0 end;
  A
end;
```

Ein erster Triangulierungsschritt auf obiger Matrix liefert

```
M41:=rstep(M4,1);
```

$$\begin{pmatrix} 1 & 1.365853659 & 2.317073171 & 0.5609756098 \\ 0 & 60.21951218 & -36.60975610 & 12.53658536 \\ 0 & -62.29268295 & -45.8536586 & 6.95121951 \\ 0 & 9.170731705 & 24.41463414 & 63.19512195 \end{pmatrix}$$

Wollen wir die Matrix M vollständig triangulieren, so müssen wir diese Triangulierungsschritte für $i = 1, \ldots, n$ ausführen.

```
rtriang:=proc(A) local i;
begin
  for i from 1 to nrows(A) do A:=rstep(A,i) end;
  A;
end;
```

Abzählen zeigt, dass im Schritt i genau $(n-i)(n-i+1)$ Multiplikationen oder Divisionen auszuführen sind, insgesamt also

$$\sum_{i=1}^{n} (n-i)(n-i+1) = \frac{n^3}{3} - \frac{n}{3}$$

Multiplikationen.

Satz 7 *Der Gaußalgorithmus benötigt auf einer n-reihigen Matrix $O(n^3)$ Multiplikationen oder Divisionen, um die Matrix zu triangulieren.*

Der Gaußalgorithmus ist bei Einheitskostenarithmetik auch ein gutes Verfahren zur Determinantenberechnung, wenn man sich die verwendeten Pivotelemente in geeigneter Weise merkt:

```
rdet:=proc(A) local i,d,n;
begin
  d:=1; n:=nrows(A);
  for i from 1 to n do d:=d*A[i,i]; A:=rstep(A,i) end;
  d;
end;

rdet(M); det(M);
```

$$-14143595.0$$

Auf ähnliche Weise kann man die Inverse einer Matrix bzw. deren Rang bestimmen.

Satz 8 *Über einem Grundbereich mit Einheitskostenarithmetik lassen sich die wichtigsten Fragestellungen der linearen Algebra für eine quadratische n-reihige Matrix A (insbesondere Berechnung der Determinante und der Inversen) mit $O(n^3)$ Multiplikationen oder Divisionen lösen.*

2.2 Der Gaußalgorithmus über den rationalen Zahlen

Experimente mit zufälligen Matrizen und obiger Prozedur `rtriang` zeigen ein relativ unangenehmes Koeffizientenwachstum, d.h. der Rechenaufwand wird zum Ende des Algorithmus immer größer. Was kann man über dieses Wachstum aussagen?

Kostenabschätzung, wenn sich bei den rationalen Operationen nichts wegkürzt: In jedem Schritt verdoppelt sich die Bitlänge der entsprechenden Zahlen. Gesamtaufwand (klassische Multiplikation) damit von der Größe

$$\sum_{k=1}^{n-1}(n-k)^2(2^{k-1}l)^2 = \left(\frac{5\cdot 4^n}{27} - \frac{n^2}{3} - \frac{2\,n}{9} - \frac{5}{27}\right) l^2,$$

also exponentiell in der Anzahl der Zeilen der Matrix.

Diese Aussage ist unter der Annahme getroffen, dass sich unterwegs keine gemeinsamen Faktoren herauskürzen lassen. Schauen wir auf das Wachstum in realen Beispielen, so vermuten wir allerdings, dass dies geschehen kann.

```
M4:=randmat(4,10^5,Dom::Rational);
rtriang(M4);
```

Wir sehen, dass nach entsprechender Simplifikation der Grad von Zähler und Nenner der entsprechenden rationalen Ausdrücke im Gegensatz zu obiger Überlegung offensichtlich nur linear wächst.

```
M9:=randmat(9,10^2,Dom::Rational);
M91:=rtriang(M9);
map(M91,x->length(numer(x)));
```

$$\begin{pmatrix}
1 & 2 & 2 & 2 & 1 & 2 & 2 & 2 & 2\\
0 & 1 & 4 & 4 & 3 & 4 & 4 & 4 & 3\\
0 & 0 & 1 & 5 & 5 & 4 & 5 & 4 & 5\\
0 & 0 & 0 & 1 & 8 & 7 & 7 & 7 & 7\\
0 & 0 & 0 & 0 & 1 & 8 & 8 & 8 & 8\\
0 & 0 & 0 & 0 & 0 & 1 & 9 & 10 & 10\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 12 & 12\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 13\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Allerdings ist es schwierig, den zusätzlichen Aufwand für die gcd-Berechnung, der sich ja auch bei der Länge der Zwischenergebnisse bemerkbar macht, abzuschätzen. Bei genauerer Analyse stellt sich heraus, dass gewisse gemeinsame Faktoren „systematisch“ entstehen und deshalb auch ohne Rechnung bestimmt und wieder herausgekürzt werden können.

Um dieses Phänomen besser zu verstehen wollen wir zunächst folgende *nennerfreie Version des Gaußalgorithmus* studieren:

```
nstep:=proc(A,i) local n,j,k;
begin
  n:=nrows(A);
  for j from i+1 to n do
    for k from i+1 to n do
      A[j,k]:=A[j,k]*A[i,i]-A[i,k]*A[j,i]
    end
  end;
  for j from i+1 to n do A[j,i]:=0 end;
  A
end;

ntriang:=proc(A) local i;
begin
  for i from 1 to nrows(A) do A:=nstep(A,i) end;
  A;
end;
```

Betrachten wir wieder Beispiele mit zufälligen Matrizen, so erkennen wir hier deutlich das exponentielle Wachstum – von Zeile zu Zeile unterscheiden sich die Längen um einen Faktor 2.

```
M91:=ntriang(M9);
map(M91,length);
```

$$\begin{pmatrix}
2 & 2 & 2 & 2 & 2 & 0 & 2 & 2 & 2 \\
0 & 3 & 3 & 4 & 4 & 4 & 4 & 3 & 3 \\
0 & 0 & 7 & 7 & 7 & 7 & 7 & 7 & 6 \\
0 & 0 & 0 & 13 & 13 & 14 & 13 & 13 & 13 \\
0 & 0 & 0 & 0 & 26 & 26 & 26 & 26 & 25 \\
0 & 0 & 0 & 0 & 0 & 50 & 50 & 51 & 51 \\
0 & 0 & 0 & 0 & 0 & 0 & 101 & 101 & 101 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 201 & 201 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 402
\end{pmatrix}$$

Untersuchen wir nun, welche gemeinsamen Faktoren in den einzelnen Zeilen vorkommen:

```
M91[1..4,1..6];
```

$$\begin{pmatrix}
40 & 59 & 34 & 10 & 66 & 0 \\
0 & -451 & 774 & 1190 & -1834 & 3880 \\
0 & 0 & 3038280 & 2089160 & -3313360 & 9362480 \\
0 & 0 & 0 & 5689606014400 & -6475673758400 & 11147129593600
\end{pmatrix}$$

```
l:=[ igcd(M91[k,i] $i=1..nrows(M91)) $k=1..5 ]
```

$$[\,1,\ 1,\ 40,\ 5051200,\ 276859157245440000\,]$$

Das erste Pivotelement $40 = 2^3 \cdot 5$ taucht als gemeinsamer Faktor in der dritten Zeile auf, welche im dritten Umformungsschritt entstanden ist. Natürlich findet sich dieses Element damit

auch als gemeinsamer Faktor in immer höherer Potenz in jedem nachfolgenden Umformungsschritt, da in der Über-Kreuz-Multiplikation ja jeweils zwei Matrixelemente, die beide diesen gemeinsamen Faktor enthalten, miteinander multipliziert werden.

```
map(l,ifactor);
```

$$\left[1, 1, 2^3 \cdot 5, 2^6 \cdot 5^2 \cdot 7 \cdot 11 \cdot 41, 2^{12} \cdot 3 \cdot 5^4 \cdot 7^2 \cdot 11^2 \cdot 41^2 \cdot 3617\right]$$

Im vierten gcd ist neben dem erwarteten Faktor 40^2 mit $451 = 11 \cdot 41$ das Pivotelement des zweiten Umformungsschritts als weiterer gemeinsamer Faktor der Elemente des vierten Umformungsschritts enthalten. Mit den Kommandos

```
M91:=nstep(M9,1): M92:=nstep(M91,2);
M92a:=M92[3..9,3..9]/M92[1,1];
M93:=nstep(M92a,1): M93a:=M93[2..7,2..7]/M92[2,2];
M94:=nstep(M93a,1): M94a:=M94[2..6,2..6]/M93[1,1];
M95:=nstep(M94a,1): M95a:=M95[2..5,2..5]/M94[1,1];
M96:=nstep(M95a,1): M96a:=M96[2..4,2..4]/M95[1,1];
M97:=nstep(M96a,1): M97a:=M97[2..3,2..3]/M96[1,1];
M98:=nstep(M97a,1): M98a:=M98[2..2,2..2]/M97[1,1];
```

können wir die Umformungen schrittweise nachvollziehen. `M92a` ist die Teilmatrix der im zweiten Umformungsschritt entstandenen Matrix `M92` ab Zeile und Spalte 3, welche durch Ausdividieren des ersten Pivotelements $\texttt{M92}_{1,1}$ entsteht. Auf sie wird der nächste `nstep` angewendet und danach das zweite Pivotelement aus der Teilmatrix ab Zeile und Spalte 2 ausdividiert usw. Die Division geht in diesem Beispiel immer auf.

Wir wollen dieses Phänomen nun näher untersuchen. Wir verwenden dazu eine generische n-reihige Matrix G, führen `nstep` darauf genügend oft aus und untersuchen, ob die Elemente einer Zeile gemeinsame Faktoren enthalten. Solche gemeinsamen Faktoren, die in der generischen Situation auftreten, sind auch in allen speziellen Matrizen vorhanden. Es handelt sich um *systematische Faktoren*, die nicht in jedem Fall neu berechnet werden müssen. Wir können sie vor dem nächsten `nstep` aus den jeweiligen Zeilen der Matrix herausteilen, was die Bitgröße der Matrixelemente und damit den Rechenaufwand verringert.

```
genmat:=proc(n) // n=size
  begin Dom::Matrix()(n,n,(i,j)->(x.i).j);
end;

G:=genmat(5);
```

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \end{pmatrix}$$

```
G1:=nstep(G,1);
G2:=map(nstep(G1,2),expand);
```

`factor(G2[4,5])` zeigt, dass das Pivotelement x_{11} in allen Einträgen von `G2[3..5,3..5]` als systematischer Faktor vorkommt und folglich ausgeteilt werden kann.

```
G2a:=map(G2[3..5,3..5]/G1[1,1],normal);
```

Ein typisches Element `G2a[4,5]` der daraus entstehenden Matrix hat die Gestalt

$$x_{11}x_{22}x_{45} - x_{11}x_{42}x_{25} - x_{12}x_{21}x_{45} + x_{12}x_{41}x_{25} + x_{21}x_{15}x_{42} - x_{22}x_{41}x_{15}\,,$$

ist also die Determinante einer dreireihigen Teilmatrix der Ausgangsmatrix `G`. Führen wir mit dieser modifizierten Matrix `G2a` einen weiteren `nstep` aus und analysieren die Elemente der neuen Matrix `G3`.

```
G3:=map(nstep(G2a,1),expand): factor(G3[3,3]);
```

Das Pivotelement $\texttt{G2}[2,2] = x_{11}x_{22} - x_{12}x_{21}$ des zweiten Umformungsschritts kommt wieder als gemeinsamer Faktor in allen Einträgen von `G3[4..5,4..5]` vor.

```
G3a:=map(G3[2..3,2..3]/G2[2,2],normal): G3a[1,1];
```

$x_{11}x_{22}x_{33}x_{45} - x_{11}x_{22}x_{43}x_{35} - x_{11}x_{23}x_{32}x_{45} + x_{11}x_{23}x_{42}x_{35} + x_{11}x_{32}x_{25}x_{43} - x_{11}x_{33}x_{42}x_{25} - x_{12}x_{21}x_{33}x_{45} + x_{12}x_{21}x_{43}x_{35} + x_{12}x_{31}x_{23}x_{45} - x_{12}x_{31}x_{25}x_{43} - x_{12}x_{23}x_{41}x_{35} + x_{12}x_{41}x_{33}x_{25} + x_{21}x_{13}x_{32}x_{45} - x_{21}x_{13}x_{42}x_{35} - x_{21}x_{32}x_{15}x_{43} + x_{21}x_{15}x_{33}x_{42} - x_{13}x_{22}x_{31}x_{45} + x_{13}x_{22}x_{41}x_{35} + x_{13}x_{31}x_{42}x_{25} - x_{13}x_{32}x_{41}x_{25} + x_{22}x_{31}x_{15}x_{43} - x_{22}x_{41}x_{15}x_{33} - x_{31}x_{23}x_{15}x_{42} + x_{23}x_{32}x_{41}x_{15}$

Auch diese modifizierte Matrix hat als Elementeinträge Determinanten vierreihiger Teilmatrizen der Ausgangsmatrix G.

Der Allgemeingültigkeit dieser Aussage wollen wir nun auf den Grund gehen. Dazu bezeichnen wir mit $D_k(i,j)$ die Determinante der Teilmatrix aus G, welche aus den Elementen mit den Zeilennummern $1,2,\ldots,k,i$ und den Spaltennummern $1,2,\ldots,k,j$ gebildet wird. Die Struktur unserer Beispielrechnungen lässt folgenden Satz vermuten:

Satz 9 *Es gilt*

$$D_{k-1}(k,k)\cdot D_{k-1}(m,n) - D_{k-1}(m,k)\cdot D_{k-1}(k,n) = D_{k-2}(k-1,k-1)\cdot D_k(m,n).$$

Statt eines genauen mathematischen Beweises, den wir hier nicht führen wollen und der Eigenschaften von Determinanten verwendet, wollen wir die Formel mit MuPAD für verschiedene Werte von k testen. Dazu sind in jedem Fall nur (umfangreiche) polynomiale Ausdrücke zu normalisieren.

Wir definieren Prozeduren

```
submat:=proc(A,r,c) // r=rowlist, c=collist
  begin Dom::Matrix()(nops(r),nops(c),(i,j)->A[r[i],c[j]]) end_proc;

DDet:=proc(A,k,i,j)
  begin det(submat(A,[$1..k,i],[$1..k,j])) end_proc;

DTest:=proc(M,k,m,n)
```

```
begin
  (DDet(M,k-1,k,k)*DDet(M,k-1,m,n) - DDet(M,k-1,m,k)*DDet(M,k-1,k,n))
    - DDet(M,k-2,k-1,k-1)*DDet(M,k,m,n)
end_proc;
```

und testen die Vermutung für verschiedene Werte (k, m, n) und eine generische Matrix genügender Größe:

```
G:=genmat(6):
expand(DTest(G,3,4,5));
```

In allen Fällen erhalten wir nach mehr oder weniger langwierigen Rechnungen 0 als Ergebnis, was die Behauptung *für die konkreten Werte* je beweist.

Damit können wir in jedem Schritt des nennerfreien Gaußalgorithmus das im vorletzten Schritt verwendete Pivotelement wieder herausdividieren. Wir erhalten damit den folgenden *Bareiss-Algorithmus*:

```
bareiss:=proc(A)
  local n,p,i,j,k;
begin
  n:=nrows(A);
  for i from 1 to n-1 do
    if i<2 then p:=1 else p:=A[i-1,i-1] end;
    for j from i+1 to n do
      for k from i+1 to n do
        A[j,k]:=normal((A[j,k]*A[i,i]-A[i,k]*A[j,i])/p);
      end
    end;
    for j from i+1 to n do A[j,i]:=0 end;
  end;
  A
end;
```

Auf einer generischen Matrix als Eingabe enthält die Dreiecksform des Bareissverfahrens an der Stelle (i, j) den Eintrag $D_{i-1}(i, j)$. Insbesondere steht für eine quadratische Matrix A an der Stelle (n, n) von `bareiss(A)` die Determinante von A.

Zur Verallgemeinerung des Bareissalgorithmus auf eine beliebige k-Algebra A (ohne Nullteiler) benötigen wir auf A zusätzlich nur eine exakte Division

$$\texttt{ediv}_A(f,g) = \begin{cases} h & \text{mit } f = g \cdot h \\ \texttt{FAIL} & \text{sonst} \end{cases}$$

Eine solche Operation existiert auf allen Integritätsbereichen mit einer effektiven Teilbarkeitsrelation wie etwa $A = \mathbb{Z}$ oder $A = k[x_1, \ldots, x_m]$.

Letztere vererbt sich rekursiv von A auf $R = A[x]$ wie folgt:

```
ediv_R:=proc(f,g) local q,x,c,t;
```

```
  begin
  if iszero(f) then return(f) end_if;
  q:=0; x:=f::dom::variables[1];
  while ((not iszero(f)) and (t:=degree(f)-degree(g)) ≥ 0) do
    c:=ediv(lcoeff(f),lcoeff(g));
    if c = FAIL then return(FAIL) end_if;
    q:=q + c · x^t;
    f:=f − c · x^t · g;
  end_while;
  if not iszero(f) then return(FAIL) end_if;
  q;
end;
```

Dabei sind $\deg(q)+1$ Operationen $\texttt{ediv}_A$ sowie $(\deg(q)+1)\cdot(\deg(g)+1)$ A-Multiplikationen von Koeffizienten der Größe $L_A(q)$ und $L_A(g)$ zur Berechnung der Teilergebnisse $c\cdot x^t\cdot g$ auszuführen. Die Kosten der exakten Division in $k[x_1,\ldots,x_m]$ sind also mit denen der Probemultiplikation $f=q\cdot g$ vergleichbar. Dasselbe gilt in $\mathbb{Z}$.

Für eine additive Längenfunktion[2] auf A und Ausgangskoeffizienten der Länge l ergibt sich für die Länge l_k der Zwischenergebnisse in Stufe k des Bareissalgorithmus

$$l_1 = l,\ l_2 = 2\,l,\ l_k = 2\cdot l_{k-1} - l_{k-2} \text{ für } k>2$$

und somit $l_k = k\cdot l$. Das Koeffizientenwachstum ist also nur noch linear.

In Stufe k werden für jedes der $(n-k)^2$ neu zu berechnenden Matrixelemente zwei Multiplikationen von Elementen der Länge l_k ausgeführt und das so entstehende Element der Länge $2\,l_k$ durch das Pivotelement p der Länge l_{k-1} aus dem letzten Schritt geteilt. Bezeichnet $C_A^*(a_1,a_2)$ die Multiplikationskosten für zwei Elemente der Länge a_1 und a_2, so ergibt sich die Komplexität des Bareissalgorithmus auf einer n-reihigen Matrix mit Elementen aus A der Länge l zu

$$C_{\text{bareiss}}(A,n,l) = \sum_{k=1}^{n}(n-k)^2\ \left(2\cdot C_A^*(k\,l,k\,l) + C_A^*((k+1)\,l,(k-1)\,l)\right)$$

Für $A=\mathbb{Z}$ und die klassische Multiplikation gilt $C_{\mathbb{Z}}^*(a_1,a_2)\sim a_1\cdot a_2$ und folglich

$$C_{\text{bareiss}}(\mathbb{Z},n,t) = \sum_{k=1}^{n}(n-k)^2\ \left(3\,k^2-1\right)t^2 = O\left(n^5\,t^2\right)\ .$$

Eine schnelle Multiplikation von Elementen $c_1,c_2\in\mathbb{Z}$ mit $l(c_1),l(c_2)\le t$ verwendet ein ähnlichen Verfahren wie FFT für Polynome und liegt in der Komplexitätsklasse $\widetilde{O}(t)$. Hierbei bedeutet $\widetilde{O}(t)$ (gelesen „soft Oh“), dass die Schranke bis auf logarithmische Faktoren zutrifft. Für ein solches Verfahren gilt

$$C_{\text{bareiss}}(\mathbb{Z},n,t) = \sum_{k=1}^{n}(n-k)^2\ (3\,k+1)\,\widetilde{O}(t) = \widetilde{O}\left(n^4\,t\right)\ .$$

[2]Wir bezeichnen eine Längenfunktion l auf A als *additiv*, wenn $l(a_1\cdot a_2)\sim l(a_1)+l(a_2)$ für $a_1,a_2\in A$ gilt. Die Bitlänge auf $A=\mathbb{Z}$ sowie der Gradvektor $d(f)=(\deg_i(f),\ i=1,\ldots,m)$ auf $A=k[x_1,\ldots,x_m]$ sind solche additiven Längenfunktionen.

Satz 10 *Der Aufwand für den Bareissalgorithmus auf einer n-reihigen ganzzahligen Matrix mit Einträgen der Bitlängen t ist bei klassischer Multiplikation von der Ordnung $O(n^5\,t^2)$ und bei schneller Multiplikation von der Ordnung $\widetilde{O}\left(n^4\,t\right)$.*

Wendet man die oben hergeleitete Formel auf $R = k[x_1, \ldots, x_m]$ über einem Körper k mit Einheitskostenarithmetik und die (additive) Längenfunktion $(\deg_1, \ldots, \deg_m)$ an, so ergibt sich für die klassische Multiplikation die Komplexitätsabschätzung

$$C_{\text{bareiss}}(R, n, l) = \sum_{k=1}^{n} (n-k)^2 \left(2\,k^{2\,m} + ((k+1)\,(k-1))^m\right) D^2 = O\left(n^{2m+3}\,D^2\right)$$

mit $l = (d_1, \ldots, d_m)$ und $D = d_1 \cdot \ldots \cdot d_m$ und für die schnelle FFT-Multiplikation

$$C_{\text{bareiss}}(R, n, l) = \sum_{k=1}^{n} (n-k)^2\,\widetilde{O}\,(k^m\,D) = \widetilde{O}\left(n^{m+3}\,D\right)$$

Satz 11 *Der Aufwand für den Bareissalgorithmus auf einer n-reihigen Matrix mit Einträgen aus $R = k[x_1, \ldots, x_m]$ vom Grad $l = (d_1, \ldots, d_m)$ über einem Körper k mit Einheitskostenarithmetik ist bei klassischer Multiplikation von der Ordnung $O(n^{2m+3}\,D^2)$ und bei schneller FFT-Multiplikation von der Ordnung $\widetilde{O}\left(n^{m+3}\,D\right)$.*

2.3 Modulare Verfahren zur Determinantenberechnung

Ein anderer Zugang zur Vermeidung der intermediären Koeffizientenexplosion ist die Ausführung der Rechnungen in einem geeigneten Bildbereich mit apriori beschränkten Kosten und Rekonstruktion des gesuchten Ergebnisses aus den Ergebnissen im Bildbereich. Dabei macht man sich zu Nutze, dass für einen Ringhomomorphismus $\phi : R \to R'$ und eine Matrix $M \in Mat(n, R)$ die Beziehung $det(\phi(M)) = \phi(det(M))$ gilt, wobei auf der linken Seite die durch ϕ induzierte elementweise Abbildung $Mat(n, R) \to Mat(n, R')$ ebenfalls mit ϕ bezeichnet wurde.

Im Fall von ganzzahligen Matrizen, also $R = \mathbb{Z}$, werden dazu Rechnungen über Restklassenkörpern $\mathbb{Z}_p$ ausgeführt. Die Kosten der Determinantenberechnung über einem solchen Bereich hängen von der Größe von p ab und sind für den klassischen Gauß-Algorithmus (und klassische Multiplikation) von der Größenordnung $O(n^3\,l(p)^2)$ bzw. $\widetilde{O}(n^3\,l(p))$ mit schneller Multiplikation.

Zwei Zugänge sind prinzipiell möglich:

- **(big prime)** p wird so groß gewählt, dass das Ergebnis aus den modularen Ergebnis eindeutig rekonstruiert werden kann.
- **(small primes)** Es werden mehrere Primzahlen p von Wortgröße $(l(p) = 1)$ gewählt und das Ergebnis aus den verschiedenen modularen Ergebnissen rekonstruiert.

Für beide Verfahren benötigen wir eine Abschätzung über die Größe der Determinante einer ganzzahligen Matrix mit Einträgen vorgegebener Größe.

Satz 12 *Ist $A = ||a_{ij}||$ eine ganzzahlige n-reihige Matrix mit Einträgen der Bitlänge l, so gilt für die Bitlänge der Determinante*

$$l(det(A)) \leq n\,(\log(n) + l) = \widetilde{O}(n\,l).$$

Der Beweis folgt unmittelbar aus der Determinantendefinition: $n!$ Summanden S der Größe $l(S) \le n\,l$ ergeben eine Zahl der maximalen Größe

$$\log(n! \cdot S) \le \log(n!) + n\,l \le n\,\log(n) + n\,l = \widetilde{O}(n\,l)\,.$$

Satz 13 (Satz über Determinantenberechnung mit big prime)
Ist $A = ||a_{ij}||$ eine ganzzahlige n-reihige Matrix mit Einträgen der Bitlänge l, so kann $det(A)$ in der Zeit $\widetilde{O}(n^5\,l^2)$ (klassische Multiplikation) bzw. $\widetilde{O}(n^4\,l)$ (schnelle Multiplikation) berechnet werden.

Beweis: Wähle dazu eine genügend große Primzahl M (deren Bitlänge b größer als $\widetilde{O}(n\,l)$ ist) und berechne $det(A) \pmod M$. Der kleinste symmetrische Rest aus der Ergebnisklasse $det(A) \pmod M$ ist dann gleich $det(A)$. Die Kosten für diese Berechnung betragen klassisch $O(n^3\,b^2)$ und bei schneller Multiplikation $\widetilde{O}(n^3\,b)$. □

Der Chinesische Restklassensatz

Der zweite Ansatz (small primes) führt die modularen Rechnungen über Bereichen $\mathbb{Z}_p$ für mehrere Primzahlen p aus und rekonstruiert daraus das Ergebnis. Grundlage für dieses Vorgehen ist der Chinesische Restklassensatz.

Satz 14 (Chinesischer Restklassensatz) *Seien $m_1, \ldots, m_n$ paarweise teilerfremde natürliche Zahlen und $m = m_1 \cdot \ldots \cdot m_n$ deren Produkt. Das System von Kongruenzen*

$$\begin{aligned} x &\equiv x_1 \bmod m_1 \\ &\ldots \\ x &\equiv x_n \bmod m_n \end{aligned}$$

hat für jede Wahl von $(x_1, \ldots, x_n)$ genau *eine Restklasse $x \pmod m$ als Lösung.*
Anders formuliert, ist die natürliche Abbildung

$$P : \mathbb{Z}_m \to \mathbb{Z}_{m_1} \times \ldots \times \mathbb{Z}_{m_n} \qquad \textit{mit } [x]_m \mapsto ([x]_{m_1}, \ldots, [x]_{m_n})$$

ein Isomorphismus.

Beispiel: $P : \mathbb{Z}_{30} \to \mathbb{Z}_2 \times \mathbb{Z}_3 \times \mathbb{Z}_5$ bildet die Restklasse $[17]_{30}$ auf das Tripel $([1]_2, [2]_3, [2]_5)$ ab.

Beweis: Injektivität ist trivial, denn $x \equiv 0 \pmod{m_i}$ bedeutet $m_i \,|\, x$ und wegen der Teilerfremdheit auch $m \,|\, x$, also $x \equiv 0 \pmod m$. Die Surjektivität folgt nun wieder aus der Injektivität und der Gleichmächtigkeit der endlichen Mengen auf beiden Seiten des Pfeils. □

Der angegebene Beweis ist allerdings nicht konstruktiv. Für Anwendungen des Satzes brauchen wir auch eine algorithmische Lösung, die nicht alle Restklassen $\pmod m$ prüfen muss (Die Laufzeit eines solchen Verfahrens wäre $O(m)$, also exponentiell in der Bitlänge von m), sondern bei vorgegebenen $(x_1, \ldots, x_n)$ die Lösung x in akzeptabler Laufzeit findet.
Wir suchen also einen **Chinesischen Restklassen-Algorithmus**

$$\texttt{CRA}((x_1, m_1), (x_2, m_2), \ldots, (x_n, m_n)) \;\to\; (x, m)$$

zur Berechnung von x.

Betrachten wir diese Aufgabe zunächst an einem konkreten Beispiel:

Aufgabe 1 Gesucht ist eine Restklasse $x \pmod{30}$, so dass

$$x \equiv 1 \pmod 2,\ x \equiv 2 \pmod 3,\ x \equiv 2 \pmod 5$$

gilt.

Lösung: $x = 5\,y + 2$ wegen $x \equiv 2 \pmod 5$. Da außerdem noch $x = 5\,y + 2 \equiv 2 \pmod 3$ gilt, folgt $y \equiv 0 \pmod 3$, also $y = 3\,z$ und somit $x = 15\,z{+}2$. Schließlich muss auch $x = 15\,z{+}2 \equiv 1 \pmod 2$, also $z \equiv 1 \pmod 2$, d.h. $z = 2\,u + 1$ und somit $x = 30\,u + 17$ gelten. Wir erhalten als einzige Lösung $x \equiv 17 \pmod{30}$, also

$$\texttt{CRA}((1,2),(2,3),(2,5)) = (17,30).$$

Das folgende Vorgehen verallgemeinert diesen Ansatz und ist für unsere Zwecke am besten geeignet. Die Grundidee besteht darin, ein Verfahren

$$\texttt{CRA2}((x_1,m_1),(x_2,m_2)) \ \to\ (x,m)$$

zum Liften für zwei Argumente anzugeben und das allgemeine Liftungsproblem darauf rekursiv zurückzuführen:

$$\texttt{CRA}((x_1,m_1),(x_2,m_2),\ldots,(x_n,m_n)) = \texttt{CRA}(\texttt{CRA2}((x_1,m_1),(x_2,m_2)),(x_3,m_3),\ldots,(x_n,m_n)))$$

Vorbetrachtungen:

$$\begin{aligned} x \equiv x_1 &\pmod{m_1} \ \Rightarrow\ x = x_1 + c\cdot m_1 \\ x \equiv x_2 &\pmod{m_2} \ \Rightarrow\ c\cdot m_1 \equiv x_2 - x_1 \pmod{m_2} \end{aligned}$$

Die hier benötigte inverse Restklasse $m_1^{-1} \pmod{m_2}$ ergibt sich als Nebenprodukt des Erweiterten Euklidschen Algorithmus und ist als `1/a mod m` in MuPAD bereits implementiert, so dass sich `CRA2` und `CRA` wie folgt ergeben:

```
CRA2:=proc(a,b) local c;
begin
  c:=(b[1]-a[1]) * modp(1/a[2],b[2]) mod b[2];
  [a[1]+c*a[2],a[2]*b[2]];
end_proc;

CRA:=proc()
begin
  if args(0)<2 then args()
  elif args(0)=2 then CRA2(args())
  else CRA2(CRA(args(2..args(0))),args(1))
  end_if;
end_proc;
```

Beispiele:
1) `u:=CRA2([5,13],[2,11])`:
Lösung: Wegen $1 = 6 \cdot 2 - 11$, also $13^{-1} \equiv 2^{-1} \equiv 6 \pmod{11}$ ergibt sich $c = (2-5) \cdot 6 \equiv 4 \pmod{11}$ und $x \equiv 5 + 4 \cdot 13 = 57 \pmod{143}$.
2) Bestimmen Sie `CRA`$(\,(x, x^2) : x \in \{2,3,5,7,11,13\})$.
Lösung: Antwort mit MuPAD und obigen Funktionen:

```
u:=map([2,3,5,7,11,13],x->[x,x^2]);
v:=CRA(op(u));
```

$$v := [127357230, 901800900]$$

Probe:

```
map(u,x->v[1] mod x[2]);
```

$$[2,3,5,7,11,13]$$

Kostenbetrachtungen: Wegen der rekursiven Natur von `CRA` wollen wir bei der Analyse von `CRA2` $l(m_1) = k,\ l(m_2) = 1$ annehmen.

> Reduktionen von Zahlen der Länge k modulo m_2 mit Aufwand $O(k)$
> `ExtendedEuklid` mit Aufwand $O(1)$
> $x = x_1 + c \cdot m_1$ mit Aufwand $O(k)$

zusammen also $C_{\mathrm{CRA2}} = O(k)$ und über alle Durchläufe $C_{\mathrm{CRA}} = O(n^2)$.

Das modulare small primes Determinantenverfahren

Gegeben ist eine Matrix $A \in Mat(n, \mathbb{Z})$ mit Einträgen der Bitlänge l. Wie besprochen berechnen wir zunächst die Determinanten der Reduktionen $A_p \in Mat(n, \mathbb{Z}_p)$ für ausreichend viele Primzahlen $p \in \{p_1, \ldots, p_N\}$ von Wortlänge.

Daraus kann das Ergebnis nach dem CRT rekonstruiert werden, wenn M eine Schranke für $|det(A)|$ ist, also $-M < det(A) < M$ gilt, und $p_1 \cdot \ldots \cdot p_N \geq 2M$ gilt. $det(A)$ ist dann die vom Betrag her kleinste Restklasse des Liftungsproblems.

```
primes:=select([$1..70],isprime);
```

```
M9:=randmat(9,100,Dom::Integer);
u:=[expr(detmod(M9,p)),p]$p in primes;
```

$$[0,2],[0,3],[3,5],[4,7],[9,11],[2,13],[9,17],[0,19],[21,23],[21,29],[13,31],$$
$$[32,37],[23,41],[21,43],[19,47],[28,53],[53,59],[60,61],[1,67]$$

```
i:=CRA(u);
```

$$[53610444472937328, 78583215510802670558790 90]$$

Dasselbe Eregbnis erhält man bei direkter berechnung von `det(M9)`.

Mit der folgenden Funktion kann weiter experimentiert werden.

```
moddet:=proc(A,primes) local u,i;
begin
  u:=[expr(detmod(A,p)),p]$p in primes;
  i:=CRA(u);
  mods(i[1],i[2]);
end;
```

Für die small primes Version der modularen Determinantenberechnung erhalten wir folgende Aufwandsabschätzung:

Satz 15 *Der Aufwand, die Determinante einer n-reihigen ganzzahligen Matrix A, deren Einträge eine Bitlänge der Größenordnung l haben, mit dem modularen small primes Verfahren zu bestimmen, ist von der Größenordnung $\widetilde{O}(n^4\,l + n^3\,l^2)$.*

Beweis: Nach der Determinantenschranke reicht es, $N = \widetilde{O}(n\,l)$ verschiedene Primzahlen $m_1, \ldots, m_N$ zu nehmen. Für jeden Modul m_i entsteht ein Aufwand $O(n^2\,l)$ für die Bestimmung der Reste der Einträge von A und $O(n^3)$ für die modulare Determinantenberechnung, also zusammen $O(N \cdot n^2(l+n))$. Schließlich sind diese N Reste modulo verschiedener Primzahlen mit `CRA` zu liften, was einen weiteren Aufwand von $O(N^2)$ verursacht. Der Gesamtaufwand ist also $O(N \cdot (n^2(l+n) + N)) = O(N \cdot n^2(l+n))$. $\square$

Der Aufwand hängt hier von der Problemstellung ab. Ist $n \gg l$ (große Matrix mit Einträgen moderater Länge), so ergibt sich die Abschätzung $\widetilde{O}(n^4\,l)$. Ist dagegen $l \gg n$ (kleinere Matrix mit explodierten Einträgen), so ergibt sich die Abschätzung $\widetilde{O}(n^3\,l^2)$. Da die größe l^2 allein aus der CRA-Liftung herrührt, ist es in disem Fall sinnvoll, nach einem schnelleren Liftungsverfahren zu suchen.

Analog können wir die Berechnungen im Bareiss-Algorithmus in N Exemplaren $\mathbb{Z}_{m_i}$ statt in $\mathbb{Z}$ ausführen und die Ergebnisse mit CRA zum ganzzahligen Resultat zusammenfügen. Division durch das Pivotelement über $\mathbb{Z}$ wird dabei im modularen Bereich als Multiplikation mit dem Inversen ausgeführt. Hierbei sind wir das erste Mal damit konfrontiert, dass wir Reduktionen nach Primzahlen p vermeiden müssen, die eines der Pivotelemente teilen, da in diesem Fall 0^{-1} zu berechnen wäre. Es gibt aber nur überschaubar wenige solcher „schlechten Reduktionen“.

Wir wollen solche schlechten Reduktionen in der Aufwandsanalyse unberücksichtigt lassen. Der entstehende Aufwand lässt sich dann abschätzen durch $n^2\,l \cdot N$ für die modularen Reduktionen, $n^3 \cdot N$ für die verschiedenen modularen Bareiss-Berechnungen und $n^2 \cdot N^2$ für das Liften der $O(n^2)$ Einträge der Ergebnismatrix.

Satz 16 *Der Aufwand für eine modulare Version des Bareissalgorithmus auf einer n-reihigen ganzzahligen Matrix, deren Einträge eine Bitlänge der Größenordnung l haben, ist von der Größenordnung $\widetilde{O}(n^4 l^2)$.*

Das ist noch immer etwas besser als die Schranke $O(n^5 l^2)$ bei klassischer Multiplipation, die sich direkt für das Bareissverfahren ergeben hatte und mit den Kosten der modularen big primes Variante übereinstimmt, aber schlechter als die Schranke $\widetilde{O}(n^4 l)$ des Bareissverfahrens auf der Basis der schnellen FFT-Multiplikation. Mit einem schnelleren CRA-Liftungsverfahren können aber ähnliche Schranken erreicht werden.

Die modulare Version der `nstep`-Variante ist deutlich ungünstiger, da hier die Größe der ganzzahligen Einträge exponentiell wächst und damit auch exponentiell viele Reduktionen N zu berechnen wären, um das ganzzahlige Ergebnis aus den modularen zu rekonstruieren.

3 Polynomiale gcd-Berechnung

3.1 Allgemeine Teilbarkeitstheorie

Die Berechnung polynomialer größter gemeinsamer Teiler (gcd) ist eine der zentralen Aufgaben jedes Computeralgebrasystems. Hätte man keine solche Simplifikationsmöglichkeit zur Verfügung, so würde sich etwa bei der Addition rationaler Funktionen nach der Formel

$$\frac{a(x)}{b(x)} + \frac{c(x)}{d(x)} = \frac{a(x)d(x) + b(x)c(x)}{b(x)d(x)}$$

in jedem Schritt Zähler- und Nennergrad verdoppeln.

Bevor wir uns der gcd-Berechnung von Polynomen widmen, seien die wichtigsten Begriffe und Eigenschaften zur Teilbarkeit noch einmal zusammengestellt. Wir fixieren dazu einen Integritätsbereich D.

Definition 2 Für $a, b \in D$ ist a *ein Teiler von* b, wenn ein $c \in D$ mit $b = a \cdot c$ existiert. Wir schreiben $a \,|\, b$.

Gilt $a \,|\, b$ und $b \,|\, a$, so unterscheiden sich die beiden Elemente um einen in D invertierbaren Faktor $c \in D^*$. Solche Elemente heißen *zueinander assoziiert*. Wir schreiben $a \sim b$. $\sim$ ist eine Äquivalenzrelation.

Aus Sicht der Teilbarkeit kann man solche Elemente nicht unterscheiden. Ist z.B. $D = k[x]$ ein univariater Polynomring, also $D^* = k$ die Menge der konstanten Polynome, so sind Teilbarkeitsfragen nur bis auf einen solchen konstanten Faktor entscheidbar.

Definition 3 Ein Element $g \in D$ bezeichnet man als *größten gemeinsamen Teiler* (gcd) der Elemente $a, b \in D$, wenn

1. $g \,|\, a,\ g \,|\, b$ und
2. $d \,|\, a,\ d \,|\, b \ \Rightarrow\ d \,|\, g$ für alle $d \in D$

gilt.

Ein gcd muss nicht existieren und auch nicht eindeutig sein. Allerdings sind zwei gcd g_1 und g_2 von vorgegebenen Elementen zueinander assoziiert, denn aus der zweiten Bedingung folgt $g_1 \,|\, g_2$ und auch $g_2 \,|\, g_1$.

Analog kann man das kleinste gemeinsame Vielfache definieren:

Definition 4 Ein Element $k \in D$ bezeichnet man als *kleinstes gemeinsames Vielfaches* (lcm) der Elemente $a, b \in D$, wenn

1. $a \,|\, k,\ b \,|\, k$ und

2. $a \,|\, d,\ b \,|\, d \ \Rightarrow\ k \,|\, d$ für alle $d \in D$

gilt.

Auch lcm müssen nicht existieren, sind aber bei Existenz eindeutig bis auf assoziierte Elemente bestimmt.

Aufgabe 2 Zeigen Sie, dass in einem Integritätsbereich, in welchem gcd existieren und die Eigenschaft

$$f' \,|\, k,\ g' \,|\, k,\ \gcd(f', g') = 1 \ \Rightarrow\ (f' \cdot g') \,|\, k$$

erfüllt ist, auch lcm existieren und die Beziehung

$$f \cdot g \sim \gcd(f, g) \cdot \operatorname{lcm}(f, g)$$

gilt.

Ein Element $p \in D$ mit $p \notin D^*$ bezeichnet man als *irreduzibel*, wenn aus $p = a\,b$ folgt, dass $a \in D^*$ oder $b \in D^*$ gilt. Ein solches Element bezeichnet man als *prim*, wenn $p \,|\, a\,b \Rightarrow p \,|\, a$ oder $p \,|\, b$ gilt. Primelemente sind stets irreduzibel, die Umkehrung gilt jedoch nicht allgemein.

Ein Ring D heißt *faktoriell* oder UFD-Bereich[3], wenn jedes Element $u \in D$ mit $u \notin D^*$ eine Zerlegung $u = p_1^{a_1} \cdot \ldots \cdot p_m^{a_m}$ in Primelementpotenzen $(a_1, \ldots, a_m \in \mathbb{N}_{>0})$ hat und diese Zerlegung eindeutig ist bis auf die Reihenfolge der Faktoren und assoziierte Elemente.

In faktoriellen Ringen sind irreduzible Elemente prim und gcd sowie lcm existieren für beliebige Elemente.

3.2 Der Euklidsche Algorithmus für Polynome

Für univariate Polynome über einem Körper k kann der gcd mit Hilfe des Euklidschen Algorithmus bestimmt werden. Dieser beruht auf dem Satz über Division mit Rest für Polynome f, g:

Satz 17 *Sind $0 \neq f, g \in k[x]$ zwei nicht triviale Polynome, so gibt es eindeutig bestimmte Polynome $q, r \in k[x]$ mit $r = 0$ oder $\deg(r) < \deg(g)$ und*

$$f = g \cdot q + r\,.$$

Praktisch lässt sich diese Division mit Rest wie beim schriftlichen Dividieren von Zahlen ausführen; sie geht von $r := f$ aus und führt in jedem Schritt die Ersetzung

$$r \ \mapsto\ r - \frac{lc(r)}{lc(g)}\, g\, x^{\deg(r)-\deg(g)}$$

aus, wodurch der Leitterm von r weggehoben wird.

[3] UFD = Unique Factorization Domain; im deutschen ist auch die Bezeichnung ZPE-Ring = Ring mit eindeutiger Zerlegung in Prim-Elemente gebräuchlich.

```
divrem := proc(f,g) local x,c,q,r,d,P0;
begin
  P0:=domtype(g); x:=P0::mainvar();
  r:=f; q:=0;
  d:=degree(r)-degree(g);
  while not iszero(r) and d>=0 do
    c:=expr(lcoeff(r)/lcoeff(g))*x^d;
    q:=q+c; r:=r-P0(c*g);
    d:=degree(r)-degree(g);
  end;
  [P0(q),r];
end;
```

Beispiel:

```
f1:=(x^8+x^6-3*x^4-3*x^3+8*x^2+2*x-5);
g1:=(3*x^6+5*x^4-4*x^2-9*x+21);

P:=Dom::UnivariatePolynomial(x,Dom::Rational);
f:=P(f1); g:=P(g1);
u:=divrem(f,g);
```

$$\left[1/3x^2 - 2/9, -5/9x^4 + 1/9x^2 - 1/3\right]$$

```
P(f) - (u[1]*g+u[2]);
```

$$0$$

Auf dieser Basis ergibt sich der Euklidsche Algorithmus wie folgt:

```
Euklid := proc(f,g) local r;
begin
  while not iszero(g) do
   r:=divrem(f,g)[2];
   logdata(r);
   f:=g; g:=r;
  end;
  f;
end;
```

Selbst wenn die Ausgangspolynome nennerfreie Koeffizienten haben, entstehen im Laufe der fortgesetzten Division mit Rest gebrochene Koeffizienten, so dass die Voraussetzung an k ein Körper zu sein wesentlich ist. Die Funktion `logdata` ist dabei ein Eintrittspunkt, um die Zwischenergebnisse zu analysieren.

Normierte Zwischen- und Endergebnisse (mit $lc(r) = 1$) erhalten wir mit der folgenden Modifikation des ursprünglichen Algorithmus:

```
nEuklid := proc(f,g) local r;
```

```
begin
  while not iszero(g) do
   r:=divrem(f,g)[2];
   if not iszero(r) then r:=r/lcoeff(r) end_if;
   logdata(r);
   f:=g; g:=r;
  end;
  f;
end;
```

Kostenanalyse über Körpern mit Einheitskostenarithmetik

Zur Kostenanalyse betrachten wir zunächst den Fall, dass k ein Körper mit Einheitskostenarithmetik, also z. B. ein Restklassenkörper ist. Die Funktion `analyse` ist so konditioniert, dass sie die Zwischenergebnisse ausgibt.

Betrachten wir obige Polynome im Polynomring $k[x]$ über $k = \mathbb{Z}_{23}$, so ergibt sich nacheinander

```
logdata:=proc(r) begin print(expr(r)) end;
P:=Dom::UnivariatePolynomial(x,Dom::IntegerMod(23));
f:=P(f1); g:=P(g1);
Euklid(f,g);
```

$$
\begin{aligned}
2\,x^4 - 5\,x^2 - 8\\
-x^2 - 9\,x + 2\\
10\,x - 8\\
-4
\end{aligned}
$$

Also sind f und g relativ prim über $\mathbb{Z}_{23}[x]$.

Über einem solchen Körper mit Einheitskostenarithmetik können wir die Komplexität durch den Grad $\deg(f)$ und $\deg(g)$ der eingehenden Polynome messen und erhalten

$$C_{\text{divrem}} = O(\deg(g) \cdot \deg(q)) \text{ für die Berechnung von } (q, r) \text{ in } f = g \cdot q + r.$$

Für die Grade der Quotienten im Euklidschen Algorithmus ergibt sich

$$
\begin{aligned}
f = q_1 \cdot g + r_1 \quad &\Rightarrow \deg(q_1) = \deg(f) - \deg(g)\\
g = q_2 \cdot r_1 + r_2 \quad &\Rightarrow \deg(q_2) = \deg(g) - \deg(r_1)\\
&\dots\\
r_{k-2} = q_k \cdot r_{k-1} + r_k \quad &\Rightarrow \deg(q_k) = \deg(r_{k-2}) - \deg(r_{k-1})
\end{aligned}
$$

und damit für die Summe der Aufwendungen der sukzessiven Divisionen mit Rest

$$
\begin{aligned}
&\deg(g) \cdot \deg(q_1) + \deg(r_1) \cdot \deg(q_2) + \deg(r_2) \cdot \deg(q_3) + \dots\\
&\quad \le \deg(g) \cdot (\deg(q_1) + \deg(q_2) + \deg(q_3) + \dots) = \deg(g) \cdot \deg(f)\,.
\end{aligned}
$$

Damit ergibt sich

$C_{\gcd} = O(\deg(f) \cdot \deg(g))$ für die Komplexität des Euklidschen Algorithmus.

Dies ist zugleich eine Abschätzung für die Anzahl der auszuführenden arithmetischen Operationen.

Univariate Polynome über $\mathbb{Q}$ und Koeffizientenexplosion

Betrachten wir nun gcd-Berechnungen im Ring $\mathbb{Q}[x]$. Für das oben betrachtete Beispiel erhält man mit den Euklidschen Algorithmus

```
P:=Dom::UnivariatePolynomial(x,Dom::Rational);
f:=P(f1); g:=P(g1);
Euklid(f,g);
```

$$-\frac{5\,x^4}{9}+\frac{x^2}{9}-\frac{1}{3}$$

$$-\frac{117\,x^2}{25}-9\,x+\frac{441}{25}$$

$$-\frac{102500}{6591}+\frac{233150\,x}{19773}$$

$$-\frac{1288744821}{543589225}$$

f und g sind relativ prim, so dass das Ergebnis 1 lautet. In den Zwischenrechnungen stellen aber ein unverhältnismäßiges Wachstum der Koeffizienten fest, welches bei `Euklid` übrigens deutlich stärker ausfällt als bei der normierten Cariante `nEuklid`. In beiden Fällen werden in jedem Schritt außerdem eine Reihe von gcd-Berechnungen im Bereich der ganzen Zahlen ausgeführt, denn in den Ergebnissen liegen stets gekürzte Brüche als Koeffizienten vor.

Weitere Beispiele können mit der folgenden Funktion `randompoly`, die Zufallspolynome erzeugt, generiert werden.

```
randompoly := proc(d,N) local r;
begin r:=random(N); _plus((r()-(N div 2))*x^i $ i=0..d); end;
```

Um einen genaueren Überblick über die Größe der Koeffizienten zu bekommen, passen wir außerdem die Funktion `logdata` an, indem wir die mit `length` bestimmte maximale Bitlänge der Koeffizienten jedes Zwischenergebnisses r in einer globalen Liste `globList` aufzeichnen und nach der Rechnung auswerten.

```
initlog:=proc() begin globList:=[] end;
logdata:=proc(r)
begin
  if not iszero(r) then
    globList:=append(globList, max(map(coeff(r),x->length(x))))
  end_if;
end;
```

```
P:=Dom::UnivariatePolynomial(x,Dom::Rational);
f:=P(randompoly(15,100)); g:=P(randompoly(13,100));
initlog(): Euklid(f,g):
```

Mit `stats::reg` kann das Wachstum der Koeffizienten genauer analysiert werden.

```
logevaluate:=proc() local n,l,xList;
begin
n:=nops(globList)-1;
l:=globList[1..n];
xList:=[i$i=1..n]:
stats::reg(xList,l,a*x^b,[x],[a,b])
end:

logevaluate();
```

$$[[4.708786947, 1.921626295], 158.0518251]$$

```
initlog(): nEuklid(f,g): logevaluate();
```

$$[[9.693343246, 0.912787451], 19.80560479]$$

Während die Bitlänge der Koeffizienten im klassischen Algorithmus `Euklid` mit dem Grad d der Eingabepolynome etwa wie $O(d^2)$ wachsen, verzeichnet `nEuklid` in praktischen Experimenten nur lineares Wachstum $O(d)$ der Koeffizienten. In der Analyse ist der letzte Wert der Restsequenz ausgelassen, da er als gcd in der normierten Version Koeffizientenlängen hat, die mit den Koeffizientenlängen der Eingabepolynome vergleichbar sind, die Normierung also bereits das intermediäre Koeffizientenwachstum beseitigt.

Da zufällig gewählte Polynome fast immer teilerfremd sind, ergibt sich als Ergebnis ein Polynom vom Grad 0. Dasselbe Koeffizientenwachstum ist zu beobachten, wenn wir Beispiele mit nichttrivialem gcd konstruieren.

Diese Beobachtung bezeichnet man als *Koeffizientenwachstum in den Zwischenergebnissen* (intermediate coefficient swell).

Der Erweiterte Euklidsche Algorithmus

Der Euklidsche Algorithmus für univariate Polynome $f, g \in k[x]$ lässt sich wie über $\mathbb{Z}$ zu einer Variante erweitern, welche mit $h = \gcd(f, g)$ zusätzlich Kofaktoren $s, t \in k[x]$ mit $\deg(s) < \deg(g),\ \deg(t) < \deg(f)$ und $h = s \cdot f + t \cdot g$ berechnet.

```
ExtendedEuklid := proc(f,g) local P0,u1,u2,v1,v2,w1,w2,q;
begin
  P0:=domtype(g);
  u1:=P0(1); u2:=P0(0); v1:=P0(0); v2:=P0(1);
  while not iszero(g) do
   q:=divrem(f,g);
   w1:=u1-q[1]*v1; w2:=u2-q[1]*v2;
```

```
   f:=g; g:=q[2]; u1:=v1; v1:=w1; u2:=v2; v2:=w2;
  end;
  [f,u1,u2];
end;
```

3.3 Univariate Polynome über einem Integritätsbereich

In der rekursiven Darstellung hatten wir multivariate Polynome als univariate Polynome betrachtet, deren Koeffizienten selbst wieder Polynome sind, allerdings eine Variable weniger enthalten. Wir wollen diese Situation jetzt unter dem Aspekt von Teilbarkeitsfragen näher untersuchen. Sei also R ein Integritätsbereich, $D = R[x]$ der entsprechende univariate Polynomring und schließlich $K = Q(R)$ der Quotientenkörper von R. Wir wollen im Weiteren untersuchen, wie Teilbarkeitsfragen in R, in $R[x]$ und in $K[x]$ zusammenhängen, wobei wir davon ausgehen, dass in R gcd existieren und auch effektiv berechnet werden können. Offensichtlich ist $D^* = R^*$. Es gilt also $f(x) \sim g(x)$ für $f(x), g(x) \in D$, wenn sich diese beiden Polynome nur um einen Faktor $c \in R^*$ unterscheiden.

$K[x]$ ist als univariater Ring über einem Körper ein Euklidscher Ring, in dem man ähnlich wie über den ganzen Zahlen, eine Division mit Rest und darauf aufbauend den Euklidschen Algorithmus zur gcd-Berechnung zur Verfügung hat.

Definition 5 Für $f(x) = a_0 + a_1 x + \ldots + a_n x^n \in R[x]$ bezeichnet man

$a := \gcd(a_0, \ldots, a_n)$ den *Inhalt* $\mathrm{cont}(f)$ von f,
$\mathrm{pp}(f) := \frac{a_0}{a} + \frac{a_1}{a} x + \ldots + \frac{a_n}{a} x^n$ den *primitiven Teil* von f.
f heißt schließlich *primitiv*, wenn $\mathrm{cont}(f) \sim 1$ gilt.

Beide Größen sind nur bis auf Faktoren $c \in R^*$ eindeutig bestimmt und es gilt für $f(x) \in R[x]$

$$f(x) \sim_R \mathrm{cont}(f) \cdot \mathrm{pp}(f).$$

$\mathrm{pp}(f)$ ist ein primitives Polynom, denn $d \,|\, a_i/a,\ i = 0, \ldots, n$, bedeutet $d\,a \,|\, a_i$ und schließlich $d\,a \,|\, a$, also $d \,|\, 1$ für jeden gemeinsamen Teiler d der Koeffizienten des primitiven Teils.

Ist $f(x) \in K[x]$ ein Polynom mit rationalen Koeffizienten, so können wir durch Multiplikation mit einem geeigneten Nenner $r \in R$ erreichen, dass $r \cdot f(x) \in R[x]$ nennerfrei ist und damit auch den primitiven Teil $\mathrm{pp}(f) := \mathrm{pp}(r \cdot f)$ von f definieren. Ebenso erhalten wir einen Inhalt $\mathrm{cont}(f) := \frac{\mathrm{cont}(r \cdot f)}{r} \in K$.

Aufgabe 3 Zeigen Sie,

- dass $\mathrm{cont}(f)$ und $\mathrm{pp}(f)$ stets eindeutig bis auf eine Einheit $c \in R^*$ bestimmt sind,
- dass obige Defintion für $f(x) \in K[x]$ nicht vom gewählten Nenner r abhängt und
- dass $\mathrm{pp}(\mathrm{pp}(f)) \sim_R \mathrm{pp}(f)$ gilt.

Eine zentrale Rolle in der Teilbarkeit univariater Polynome spielt das folgende

Lemma 3 (Gauß-Lemma) *Über einem faktoriellen Ring R ist das Produkt zweier primitiver Polynome wieder ein primitives Polynom.*

Beweis: Seien $f = \sum_i a_i x^i$ und $g = \sum_j b_j x^j$ die beiden primitiven Polynome und $p \in R$ ein gemeinsamer Primteiler aller Koeffizienten von $f \cdot g$. Dann existieren Indizes i_0, j_0 mit

$$\begin{aligned} \forall\, i < i_0 \ &: \ p \,|\, a_i \text{ und } p \not| \, a_{i_0} \\ \forall\, j < j_0 \ &: \ p \,|\, b_i \text{ und } p \not| \, b_{j_0} \end{aligned}$$

Der Koeffizient $c_{i_0+j_0}$ vor $x^{i_0+j_0}$ in $f \cdot g$ hat dann die Form

$$a_0 b_{i_0+j_0} + a_1 b_{i_0+j_0-1} + \ldots + a_{i_0} b_{j_0} + \ldots + a_{i_0+j_0-1} b_1 + a_{i_0+j_0} b_0$$

Dabei sind die Summen $a_0 b_{i_0+j_0} + \ldots + a_{i_0-1} b_{j_0+1}$ wegen der ersten Faktoren und $a_{i_0+1} b_{j_0-1} + \ldots + a_{i_0+j_0} b_0$ wegen der zweiten Faktoren jeweils durch p teilbar. $a_{i_0} b_{j_0}$ als Produkt zweier nicht durch p teilbarer Faktoren ist dagegen nicht durch p teilbar, also $c_{i_0+j_0}$ auch nicht, im Gegensatz zur Annahme. $\square$

Aus diesem Satz ergeben sich sofort die folgenden weiteren Beziehungen

Folgerung 1

1. $\forall\ f, g \in R[x]\ \operatorname{cont}(fg) \sim \operatorname{cont}(f)\operatorname{cont}(g)$,
2. $\forall\ f, g \in K[x]\ \operatorname{pp}(fg) \sim \operatorname{pp}(f)\operatorname{pp}(g)$,
3. $\forall\ f, g \in R[x]\ g \,|\, f \ \Rightarrow\ \operatorname{cont}(g) \,|\, \operatorname{cont}(f),\ \operatorname{pp}(g) \,|\, \operatorname{pp}(f)$,
4. $\forall\ f \in R[x], r \in R\ \operatorname{cont}(rf) \sim r \cdot \operatorname{cont}(f),\ \operatorname{pp}(rf) \sim \operatorname{pp}(f)$,
5. $\forall\ f, g \in K[x]\ g \,|\, f$ *in* $K[x]\ \Rightarrow\ \operatorname{pp}(g) \,|\, \operatorname{pp}(f)$ *in* $R[x]$

sowie der folgende Satz über die Faktorzerlegung in Polynomringen

Folgerung 2 *Ist R ein faktorieller Ring, so auch $R[x]$.*

Beweis: $K = Q(R)$ sei der Quotientenkörper von R.

Ist $p \in K[x]$ mit $\deg(p) > 0$ prim in $K[x]$, so ist $\operatorname{pp}(p)$ prim in $R[x]$. In der Tat $\operatorname{pp}(p) \,|\, f \cdot g$ in $R[x] \Rightarrow p \,|\, f$ oder $p \,|\, g$ in $K[x] \Rightarrow \operatorname{pp}(p) \,|\, \operatorname{pp}(f)$ oder $\operatorname{pp}(p) \,|\, \operatorname{pp}(g)$ in $R[x]$ (wegen 5.) $\Rightarrow \operatorname{pp}(p) \,|\, f$ oder $\operatorname{pp}(p) \,|\, g$ in $R[x]$. Umgekehrt ist jedes prime $p \in R[x]$ mit $\deg(p) > 0$ primitiv, da sonst eine nichttriviale Zerlegung $p = \operatorname{cont}(p) \cdot \operatorname{pp}(p)$ existieren würde.

Ist $f \in R[x]$ ein Polynom und $f = r \cdot \prod p_i^{a_i}$ die Zerlegung in Primfaktoren in $K[x]$ mit $\deg(p_i) > 0$ und o. B. d. A. $p_i = \operatorname{pp}(p_i) \in R[x]$ prim und primitiv sowie $r \in K$, so ist mit Blick auf $\operatorname{pp}(f) \sim \prod p_i^{a_i}$ (wegen 2.) auch $r \sim \operatorname{cont}(f) \in R$. $f = r \cdot \prod p_i^{a_i}$ ist also eine Zerlegung in Primfaktoren in $R[x]$, wenn man $r \in R$ noch durch eine Primfaktorzerlegung ersetzt.

Die Eindeutigkeit der Primfaktorzerlegung ergibt sich aus der Eindeutigkeit der Zerlegung $f = \operatorname{cont}(f) \cdot \operatorname{pp}(f)$ sowie der Eindeutigkeit der Zerlegungen in R und in $K[x]$ (mit nachfolgendem Übergang zu primitiven Teilen). $\square$

Als zentraler Satz über die Berechnung polynomialer größter gemeinsamer Teiler ergibt sich daraus der folgende

Satz 18 (Kompositionssatz) *Die gcd-Berechnung von $f, g \in R[x]$ kann man auf die in R und in $K[x]$ zurückführen.*

Es gilt

$$\gcd_{R[x]}(f,g) = \gcd_R(\mathrm{cont}(f), \mathrm{cont}(g)) \cdot \mathrm{pp}(\gcd_{K[x]}(f,g)).$$

Beweis: Wir haben nur zu zeigen, dass die rechte Seite ein gcd von f und g ist. Diese rechte Seite besteht aus den Faktoren $r = \gcd_R(\mathrm{cont}(f), \mathrm{cont}(g)) \in R$ und $h = \mathrm{pp}(\gcd_{K[x]}(f,g)) \in R[x]$.

$r \,|\, \mathrm{cont}(f)$ und $h = \mathrm{pp}(h) \,|\, \mathrm{pp}(f)$, also ist $r\,h$ ein Teiler von $f \sim \mathrm{cont}(f)\,\mathrm{pp}(f)$ und analog von g.

Ist umgekehrt $d \in R[x]$ ein gemeinsamer Teiler von f und g, so gilt $\mathrm{cont}(d) \,|\, \mathrm{cont}(f)$, $\mathrm{cont}(g)$ und folglich $\mathrm{cont}(d) \,|\, \gcd(\mathrm{cont}(f), \mathrm{cont}(g))$. Wegen $d \,|\, f, g$ gilt $d \,|\, h$ wenigstens in $K[x]$ und damit wieder $\mathrm{pp}(d) \,|\, \mathrm{pp}(h) = h$ in R[x]. □

Beispiele: [4, 6.2.]

$R = \mathbb{Z}$,
$f_3 = 12\,x^3 - 28\,x^2 + 20\,x - 4$,
$g_3 = -12\,x^2 + 10\,x - 2$

```
P:=Dom::UnivariatePolynomial(x,Dom::Integer);
f:=P(f3); g:=P(g3);
P::content(f);
P::primpart(f);
```

In diesem Fall ergeben sich

$$\mathrm{cont}(f_3) = 4,\ \mathrm{cont}(g_3) = 2,$$
$$\mathrm{pp}\left(\gcd_{Q(R)[x]}(\mathrm{pp}(f), \mathrm{pp}(g))\right) = 3\,x - 1,\ \gcd_{R[x]}(f,g) = 6\,x - 2$$

$R = \mathbb{Z}_5[y]$,
$f_4 = (y^3 + 3\,y^2 + 2\,y)\,x^3 + (y^2 + 3\,y + 2)\,x^2 + (y^3 + 3\,y^2 + 2\,y)\,x + (y^2 + 3\,y + 2)$,
$g_4 = (2\,y^3 + 3\,y^2 + y)\,x^2 + (3\,y^2 + 4\,y + 1)\,x + (y + 1)$

```
Py:=Dom::UnivariatePolynomial(y,Dom::IntegerMod(5));
P:=Dom::UnivariatePolynomial(x,Py);
f:=P(f4); g:=P(g4);
P::content(f);
P::primpart(g);
```

In diesem Fall ergeben sich ($3 \equiv -2 \pmod 5$)

$$\mathrm{cont}(f_4) = y^2 - 2\,y + 2\,y,\ \mathrm{cont}(g_4) = y + 1,$$
$$\mathrm{pp}\left(\gcd_{Q(R)[x]}(\mathrm{pp}(f), \mathrm{pp}(g))\right) = y\,x + 1,\ \gcd_{R[x]}(f,g) = (y^2 + y)\,x + (y + 1)$$

Die gcd-Berechnung von univariaten Polynomen über einem Körper und damit der Euklidsche Algorithmus spielen eine zentrale Rolle bei der Berechnung von gcd in allgemeineren Strukturen. Es stellt sich allerdings heraus, dass der Euklidsche Algorithmus in seiner einfachen

Form für die direkte Anwendung in komplexeren Aufgabenstellungen nicht schnell genug ist. In den letzten 35 Jahren wurden eine Reihe von Verbesserungen gefunden, die im folgenden diskutiert werden sollen.

3.4 Die Resultante

Nach [4, 6.3.].

Für teilerfremde $f, g \in F[x]$ über einem Körper F mit $\deg(f) = m,\ \deg(g) = n$ lässt sich bekanntlich mit dem Erweiterten Euklidschen Algorithmus eine Darstellung

$$1 = s\,f + t\,g$$

mit geeigneten $s, t \in F[x]$ bestimmen. Ersetzen wir s durch das eindeutig bestimmte Polynom $s' = rem(s, g)$ mit $s' = s - q\,g$ und $\deg(s') < \deg(g)$, sowie $t' = t + q\,f$, so gilt offensichtlich immer noch

$$1 = s'\,f + t'\,g.$$

Wegen $t'\,g = 1 - s'\,f$ ist $\deg(t'\,g) \leq \deg(s) + \deg(f) < \deg(f) + \deg(g)$ und somit auch $\deg(t') < \deg(f)$.

Dieses eindeutig bestimmte Paar (s', t') mit $1 = s'\,f + t'\,g$ und $\deg(s') < \deg(g), \deg(t') < \deg(f)$ bezeichnet man als die *Bezout-Koeffizienten* von (f, g).

Lemma 4 *F ist ein Körper, $0 \neq f, g \in F[x]$ Polynome. Dann ist*

$$\gcd(f, g) \not\sim 1 \Leftrightarrow \exists s, t \in F[x] \setminus \{0\} \ \textit{mit}\ s\,f + t\,g = 0,\ \deg(s) < \deg(g), \deg(t) < \deg(f)$$

Dies können wir in Termini der „Linearkombinations-Abbildung“

$$\begin{aligned} \phi : F[x] \times F[x] &\longrightarrow F[x] \\ (s, t) &\longrightarrow s\,f + t\,g \end{aligned}$$

aufschreiben. $P_d = \{a \in F[x] \mid \deg(a) < d\}$ ist ein Vektorraum mit der Basis

$$\left(x^0, x^1, x^2, \ldots, x^{d-1}\right)$$

und ϕ induziert eine lineare Abbildung

$$\phi_0 : P_m \times P_n \longrightarrow P_{m+n}$$

zwischen Vektorräumen derselben endlichen Dimension $m+n$. Das Lemma kann nun wie folgt umformuliert werden.

Satz 19 *Seien $f, g \in F[x]$ Polynome vom Grad $\deg(f) = n, \deg(g) = m$. Dann gilt*

1. $\deg_x(\gcd(f, g)) = 0$, *d. h. (f, g) sind teilerfremd, genau dann, wenn ϕ_0 ein Isomorphismus ist.*
2. *Ist $\deg_x(\gcd(f, g)) = 0$, so gibt es eindeutig bestimmte $(s, t) \in P_m \times P_n$ mit $\phi_0(s, t) = 1$. Das sind die Bezout-Koeffizienten von (f, g).*

Abbildung in der gegebenen Basis

$$\left((x^i, 0)\,,\ i = 1, \ldots, m-1\right) \cup \left((0, x^j)\,,\ j = 1, \ldots, n-1\right)$$

als Matrix aufschreiben ergibt die *Sylvester-Matrix* S, in deren Spalten $i = 1, \ldots m$ die Koeffizienten von $x^{i-1} * f$ und in den Spalten $i = m+1, \ldots m+n$ die Koeffizienten von $x^{i-m-1} * g$ stehen. Deren Determinante bezeichnet man als die *Resultante* $\mathrm{res}(f, g)$.

Folgerung 3 *f, g wie gehabt.*

1. $\deg_x(\gcd(f, g)) = 0 \Leftrightarrow det(S) \neq 0$.
2. *Die Bezout-Koeffizienten von (f, g) ergeben sich als Lösung eines linearen Gleichungssystems mit Koeffizientenmatrix S.*

Kombination des allgemeinen Satzes über gcd mit diesen Ergebnissen liefert den folgenden

Satz 20 (Resultantenkriterium) *Sei R ein Integritätsbereich und $0 \neq f, g \in R[x]$. Dann ist $\deg_x(\gcd(f, g)) = 0$ genau dann, wenn $res(f, g) \neq 0$ in R gilt.*

In der Tat, $\deg_x(\gcd(f, g))$ kann über $F = Q(R)$ berechnet werden.

Da die zugehörigen Bezout-Koeffizienten über F aus einem linearen Gleichungssystem mit Koeffizientenmatrix S bestimmt werden können, ergibt sich aus der Cramerschen Regel weiter

Folgerung 4 *Sei R ein Integritätsbereich und $0 \neq f, g \in R[x]$. Dann gibt es $0 \neq s, t \in R[x]$ mit $s\,f + t\,g = res(f, g)$ und $\deg_x(s) < \deg_x(g),\ \deg_x(t) < \deg_x(f)$.*

Ist $h = \gcd(f, g)$ und $\deg_x(h) > 0$, so setze man $s = g/h$ und $t = -f/h$.

3.5 gcd von Familien von Polynomen

Es gilt auch hier der Kompositionssatz, so dass die gcd-Bestimmung zurückgeführt werden kann auf die Bestimmung des gcd der Inhalte der einzelnen Polynome sowie die Bestimmung von $h = \gcd(f_1, f_2, \ldots, f_N)$ in $K[x]$ über einem Körper K.

Letzteres kann mit einem probabilistischen Verfahren auf eine einzige gcd-Berechnung $h^* = \gcd(f_1, f_2 + \alpha_3 f_3 + \cdots + \alpha_N f_N)$ in $K[x]$ zurückgeführt werden, wobei $\alpha_3, \ldots, \alpha_N \in S$ uniform zufällig aus einer vorgegebenen Menge S ist.

Im Ergebnis gilt $h^* = h$ mit hoher Wahrscheinlichkeit. Generell gilt $h \mid h^*$, d. h. das berechnete h^* hat möglicherweise zu hohen Grad. Damit kann durch einfache Teilbarkeitstest $h^* \mid f_i$ festgestellt werden, ob $h^* = h$ gilt und anderenfalls mit einer weiteren Zufallsauswahl die Rechnung wiederholt werden. Details dazu finden sich im Buch [4, S. 166 ff.].

Berechnungen derartiger gcd treten vor allem bei der Bestimmung des Inhalts von Polynomen auf. Deren Kosten sind also mit den Kosten einer einfachen gcd-Berechnung vergleichbar. Die praktischen Kosten einer gcd-Berechnung nach dem Kompositionssatz auf der Basis eines solchen probabilistischen Verfahrens sind also von der Größenordnung

$$C_{\text{gcd in R[x]}}(b, d) = O\left(C_{\text{gcd in R}}(b) + C_{\text{gcd in R}}(b') + C_{\text{gcd in K[x]}}(b, d)\right)$$

Hier ist b eine Schranke für die Koeffizientengröße von $f, g \in R[x]$, d eine Schranke für den Grad und b' eine Schranke für die Koeffizientengröße von $h = \gcd_{K[x]}(f, g)$. Gewöhnlich dominieren die Kosten $C_{\text{gcd in K[x]}}(b, d)$ das gesamte Verfahren.

3.6 Polynomiale Restsequenzen

Seien dazu $f(x), g(x) \in R[x]$. Betrachten wir die Division mit Rest `divrem(f,g)`. In der klassischen Version über $K = Q(R)$ entsteht als Nenner eine Potenz von $lc(g)$, weil in jedem Schleifendurchlauf durch diesen Leitkoeffizienten geteilt wird. Um Divisionen zu vermeiden, modifizieren wir dazu den Ersetzungsschritt in der Division mit Rest zu

$$r \;\mapsto\; lc(g)\, r - lc(r)\, g\, x^{\deg(r)-\deg(g)}$$

Im Ergebnis erhalten wir eine Darstellung

$$lc(g)^l f(x) = q(x) \cdot g(x) + r(x),$$

wobei l die Anzahl der Ersetzungsschritte angibt. Eine entsprechende MuPAD-Prozedur hat folgendes Aussehen:

```
pdivrem1 := proc(f,g) local x,c,q,r,d,lg,P0;
begin
  P0:=domtype(g); x:=P0::mainvar();
  r:=f; q:=0; lg:=lcoeff(g);
  d:=degree(r)-degree(g);
  while not iszero(r) and d>=0 do
      c:=lcoeff(r)*x^d;
      q:=lg*q+c; r:=P0(lg*r-c*g);
      d:=degree(r)-degree(g);
  end;
  [P0(q),r];
end;
```

Beweis der Korrektheit durch Betrachtung der Schleifeninvariante

$$c = lc(r) \cdot x^d,\; r' = lc(g) \cdot r - c \cdot g,\; q' = lc(g) \cdot q + c,\; l = l + 1$$

Beispiel:

```
P:=Dom::UnivariatePolynomial(x,Dom::Rational);
f:=randompoly(5,100); g:=randompoly(3,100);

u:=pdivrem1(P(f),P(g));
```

Probe:

```
u[1]*P(g)+u[2] - lcoeff(P(g))^3*P(f);
```

Da in jedem Ersetzungsschritt der Grad von r um wenigstens 1 sinkt, gilt

$$l \le d := \deg(f) - \deg(g) + 1,$$

so dass wir im Weiteren für unsere theoretischen Untersuchungen l durch d ersetzen wollen, was durch eine entsprechende Nachjustierung in obiger Prozedur erfolgen kann:

```
pdivrem := proc(f,g) local x,c,q,r,d,lg,n,P0;
begin
  P0:=domtype(g); x:=P0::mainvar();
  r:=f; q:=0; lg:=lcoeff(g);
  d:=degree(r)-degree(g); n:=d+1;
  while not iszero(r) and d>=0 do
      c:=lcoeff(r)*x^d;
      q:=lg*q+c; r:=P0(lg*r-c*g);
      d:=degree(r)-degree(g); n:=n-1;
  end;
  [P0(lg^n*q),lg^n*r];
end;
```

Definition 6 Ist R ein Integritätsbereich und $f(x), g(x) \in R[x], \deg(f) \geq \deg(g)$, so gibt es eindeutig bestimmte Polynome $q(x), r(x) \in R[x]$ mit $r = 0$ oder $\deg(r) < \deg(g)$ und

$$lc(g)^{\deg(f)-\deg(g)+1} f(x) = q(x) \cdot g(x) + r(x).$$

q nennt man den *Pseudoquotienten* $pquot(f, g)$ und r nennt man den *Pseudorest* $prem(f, g)$ der Polynome f und g. Diese Funktionen stehen in MuPAD mit der Funktion `pdivide` zur Verfügung.

Über $K[x]$ mit $K = Q(R)$ sind der übliche Rest und der Pseudorest zueinander assoziierte Polynome. Also kann man bei der Berechnung von $\gcd_{K[x]}(f, g)$ mit Hilfe des Euklidschen Algorithmus die Restbildung durch Pseudorestbildung ersetzen. Die Folge der dabei entstehenden Reste bezeichnet man als die *Euklidsche Polynomiale Restsequenz*. Eine MuPAD-Implementierung sieht so aus:

```
eprs := proc(f,g) local r;
begin
  while not iszero(g) do
    r:=pdivrem(f,g)[2];
    print(r);
    f:=g; g:=r;
  end;
  f;
end;
```

Beispielrechnungen mit dieser Implementierung sind im Hinblick auf Vermeidung von Rechenaufwand nicht sehr ermutigend, denn die entstehenden Koeffizienten sind deutlich größer als die, welche bei der rationalen Version auftreten. Eine genauere Analyse der in den Restsequenzen auftretenden Koeffizienten zeigt deren exponentielles Wachstum. Wie bereits im Fall linearer Gleichungssysteme zeigt sich weiter, dass sich die Koeffizienten gewöhnlich aus vielen Faktoren zusammensetzen. Es entsteht also die Frage, ob man Faktoren finden kann, die in allen Koeffizienten eines Restpolynoms gemeinsam vorkommen. Diese könnte man dann ausdividieren, bevor die Rechnung fortgesetzt wird.

Diese Frage wollen wir nun studieren. Dazu werden wir zunächst eine geeignete Terminologie fixieren.

Definition 7 Seien $f(x), g(x) \in R[x]$ zwei Polynome mit $d = \deg(f) \geq \deg(g)$. Als *Polynomiale Restsequenz* (PRS) bezeichnet man eine Folge von Polynomen

$$R_0 = f,\ R_1 = g,\ R_2,\ R_3,\ \ldots,\ R_k \neq 0$$

mit

(1) $\deg(R_1) > \deg(R_2) > \ \ldots,\ > \deg(R_k)$.

(2) $\alpha_i \cdot R_{i-1}(x) = Q_i(x) \cdot R_i(x) + \beta_i \cdot R_{i+1}(x)$ mit geeigneten $\alpha_i, \beta_i \in R$ und $Q_i(x) \in R[x]$.

(3) $prem(R_{k-1}, R_k) = 0$

Wie oben gilt dann stets $\mathrm{pp}(R_k) = \mathrm{pp}(\gcd_{K[x]}(f, g))$.

Setzen wir $\delta_i := \deg(R_{i-1}) - \deg(R_i) = \deg(Q_i)$, so kann man $\alpha_i = lc(R_i)^{\delta_i+1}$ wie bei der Pseudodivision setzen, was wir im Folgenden stets stillschweigend annehmen.

Zur Komplexität dieser Berechnungen: Nur aus dem Punkt (2) der Definition entstehen Kosten, und zwar (C^* bezeichnet die Multiplikationskosten in $R[x]$):

Skalieren von R_{i-1} mit α_i:	$C^*(\alpha_i, R_{i-1})$
Division mit Rest:	$C^*(Q_i, R_i)$
Finden des Faktors β_i:	$Kosten(\beta_i)$
Ausdividieren des Faktors β_i:	$C^*(\beta_i, R_{i+1})$

Die Gesamtkosten ergeben sich, wenn diese vier Posten für alle Schleifendurchläufe ($i = 1, \ldots, d$) addiert werden.

Die Kosten der Multiplikationen hängen von der Länge der entsprechenden Koeffizienten ab. Wir wollen wieder annehmen, dass auf R eine additive Längenfunktion[4] gegeben ist, bzgl. welcher q_i die durchschnittliche Größe der Koeffizienten von $Q_i(x)$ und l_i die durchschnittliche Größe der Koeffizienten von $R_i(x)$ bezeichnet. Dann gilt für $i \geq 1$

$$\begin{aligned} l(\alpha_i) &= (\delta_i + 1)\, l_i \\ l_{i+1} &= (\delta_i + 1)\, l_i + l_{i-1} - l(\beta_i) \\ q_i &= \delta_i\, l_i + l_{i-1} \end{aligned}$$

und im Normalfall, wenn alle $\delta_i = 1$ (so eine PRS heißt *normal*)

$$\begin{aligned} l_{i+1} &= 2\, l_i + l_{i-1} - l(\beta_i) \\ q_i &= l_i + l_{i-1} \\ \deg(R_i) &= d - i, \end{aligned}$$

woraus sich ergibt (C_R^* bezeichnet die Multiplikationskosten in R):

$$\begin{aligned} C^*(\alpha_i, R_{i-1}) &= (d - i + 2)\, C_R^*(2\, l_i, l_{i-1}) \\ C^*(\beta_i, R_{i+1}) &= (d - i)\, C_R^*(l(\beta_i), l_{i+1}) \\ C^*(Q_i, R_i) &= 2\,(d - i + 1)\, C_R^*(l_i + l_{i-1}, l_i) \end{aligned}$$

[4] Solche additiven Längenfunktionen sind zum Beispiel die Bitlänge für $R = \mathbb{Z}$ bzw. der Gradvektor für $R = k[x_1, \ldots, x_{n-1}]$.

Die Kosten hängen also wesentlich vom Wachstum der Koeffizienten von $R_i(x)$ ab, deren Größe l_i den Anfangsbedingungen $l_0 = l_1 = l$, $l_2 = (\delta_1 + 2)\, l$ genügen möge. Die Graddifferenz $\delta_1 = \deg(f) - \deg(g)$ spielt eine besondere Rolle unter den Graddifferenzen δ_i in einer PRS, da sie durch die Eingabegrößen f, g bestimmt ist.

1) $\beta_i = 1$ liefert die oben beschriebene *Euklidsche PRS*. Für den normalen Fall $\delta_i = 1$ ergibt sich deren Koeffizientenwachstum damit aus der Rekursion

$$l_0 = l_1 = l, l_{i+1} = 2l_i + l_{i-1} \text{ für } i > 0$$

Wir erhalten $2\, l_i \le l_{i+1} \le 3\, l_i$, also exponentielles Koeffizientenwachstum (die genaue Wachstumsgröße ergibt sich nach Auflösung der Rekursionsgleichung zu $l_i = \alpha^{i-1}\, l$ für $i \ge 1$ mit $\alpha = 1 + \sqrt{2}$).

Mit $l_i \sim 2^i\, l$ und $R = \mathbb{Z}$ (klassische Multiplikation) ergibt sich für die Kosten

$$C_{\mathrm{EPRS}}(d) = \sum_{i=1}^{d} ((d-i+2)\, 2^{2i} + 3\,(d-i+1)\, 2^{2i}) l^2 \sim \frac{76}{9} 4^d\, l^2 = O\left(4^d\, l^2\right)$$

bzw. $\widetilde{O}\left(2^d\, l^2\right)$ bei schneller Multiplikation in $\mathbb{Z}$.

2) Man könnte auch in jedem Schritt R_{i+1} durch seinen primitiven Teil ersetzen, also $\beta_i = \mathrm{cont}(R_{i+1})$ ausdividieren. Das wäre der größtmögliche Faktor. Wir verwenden dazu die Funktion `primpart`, welche der MuPAD-Datentyp `Dom::UnivariatePolynomial(x,R)` für Rechnungen über einem Grundbereich R zur Verfügung stellt.

```
pprs := proc(f,g) local r,P0;
begin
  P0:=domtype(g);
  while not iszero(g) do
    r:=pdivrem(f,g)[2];
    if not iszero(r) then r:=P0::primpart(r) end_if;
    print(r);
    f:=g; g:=r;
  end;
  f;
end;
```

Beispiele:

```
P:=Dom::UnivariatePolynomial(x,Dom::Integer);
f:=randompoly(5,100); g:=randompoly(3,100);
```

$$43\, x - 50\, x^2 - 39\, x^3 - 12\, x^4 - 37\, x^5 + 41$$
$$15\, x^2 - 6\, x + 41\, x^3 - 30$$

```
eprs(P(f),P(g));
pprs(P(f),P(g));
```

$$-4162372x^2 + 2587457x + 557581$$
$$427244858549977x - 425795935537343$$
$$-1821593858726480627916734841291523 52$$

$$4162372x^2 - 2587457x - 557581$$
$$6199051937x - 6178028983$$
$$1$$

Natürlich müsste bei dieser Berechnung des gcd der Koeffizienten von r die Funktion `pprs` rekursiv eingesetzt werden, was eine Aufwandsabschätzung schwierig macht, zumal nicht klar ist, ob der in den Beispielen regelmäßig zu beobachtende Kürzungseffekt generell auftritt. Das Verfahren ist also mglw. praktisch interessant, aber theoretisch schwer zu untersuchen.

Es ergibt sich damit die Frage, ob Faktoren gewisser Bauart systematisch auftreten. Insbesondere steht wegen der Analogie zu Verfahren der linearen Algebra, die bei der Betrachtung der Resultanten sichtbar geworden ist, die Frage, ob man ähnlich wie beim Bareiss-Algorithmus Faktoren, die man in früheren Schritten hineingesteckt hat, ohne aufwändige gcd-Berechnung in späteren Schritten wieder herausdividieren kann. Dies ist in der Tat möglich:

Satz 21 (Silvester 1853, Collins 1967)

$$\alpha_i = lc(R_i)^{\delta_i+1}, \qquad \beta_1 = 1, \beta_i = \alpha_{i-1} \text{ für } i > 1$$

liefert eine PRS, die reduzierte Polynomiale Rest-Sequenz*.*

Für einen Beweis dieses Satzes sei auf [1, ch. 7] bzw. [2] verwiesen. Von seiner Evidenz kann man sich an Hand von Rechnungen mit zufälligen Polynomen und der folgenden MuPAD-Implementierung überzeugen:

```
redprs := proc(f,g) local r,beta;
begin
  beta:=1;
  while not iszero(g) do
     r:=pdivrem(f,g)[2]/beta;
     print(r);
     beta:=lcoeff(g)^(degree(f)-degree(g)+1);
     f:=g; g:=r;
  end;
  f;
end;

redprs(P(f),P(g));
```

$$-4162372x^2 + 2587457x + 557581$$
$$6199051937x - 6178028983$$
$$-2213434272758$$

In allen Beispielen gehen die im Laufe der Rechnungen auszuführenden Divisionen auf. Die Rechnungen zeigen zugleich, dass mit diesem Ansatz offensichtlich ein großer Teil der systematisch auftretenden Koeffizienten bereits ausgeteilt wird.

Dies gilt allerdings nur für normale Restsequenzen, d.h. wenn $\delta_i = 1$ für alle $i > 1$ ist. In diesem Fall erhalten wir für das Koeffizientenwachstum

$$l_0 = l_1 = l, l_2 = 3\,l, \quad l_{i+1} = 2l_i + l_{i-1} - 2l_{i-1} = 2l_i - l_{i-1} \text{ für } i > 1,$$

also $l_i = (2i-1)\,l$. Die Koeffizientengröße wächst folglich linear.

Liegt dagegen eine Restsequenz mit größeren Gradsprüngen vor, so kann das Koeffizientenwachstum noch immer exponentiell sein.

```
redprs(P(subs(f,x=x^2)),P(subs(g,x=x^2)));
pprs(P(subs(f,x=x^2)),P(subs(g,x=x^2)));
```

$$-6996947332x^4 + 4349515217x^2 + 937293661$$
$$-729124334912677794832 04x^2 + 72665164271572099761436$$
$$-189821124513927317647524353510002824564645949014 4$$

$$4162372x^4 - 2587457x^2 - 557581$$
$$6199051937x^2 - 6178028983$$
$$1$$

Aufgabe 4 Leiten Sie für den Fall, dass alle $\delta_i = 2$ sind, eine Formel für l_i her und überzeugen Sie sich, dass diese exponentiell in i ist.

Ein noch besseres Verhalten zeigt die *Subresultanten-PRS*:

Satz 22 (Collins 1967, Brown/Traub 1971) $\alpha_i = lc(R_i)^{\delta_i+1}$ *und*

$$h_1 = 1, \quad h_i = lc(R_{i-1})^{\delta_{i-1}} \cdot h_{i-1}^{1-\delta_{i-1}} \quad (i > 1),$$
$$\beta_1 = 1, \quad \beta_i = lc(R_{i-1}) \cdot h_i^{\delta_i} \quad (i > 1)$$

definiert eine Polynomiale Restsequenz, die Subresultanten-PRS.

Für einen Beweis dieses Satzes sei ebenfalls auf [1, ch. 7] bzw. [2] verwiesen. Mit folgender MuPAD-Implementierung kann man sich überzeugen, dass wiederum alle Divisionen aufgehen:

```
sprs := proc(f,g) local r,lc,h,delta;
begin
  lc:=1; h:=1;
  while not iszero(g) do
     delta:=degree(f)-degree(g);
     r:=pdivrem(f,g)[2]/(lc*h^delta);
     print(r);
```

```
    lc:=lcoeff(g);
    h:=lc^delta*h^(1-delta);
    f:=g; g:=r;
  end;
  f;
end;
```

Eine Verbesserung gegenüber der reduzierten PRS wird höchsten für nicht normale Restsequenzen erreicht, da für normale Restsequenzen die Subresultanten-PRS mit der reduzierten PRS zusammenfällt. Im Gegensatz zu letzterer wachsen in der Subresultanten-PRS die Koeffizienten auch bei beliebigen Gradsprüngen nur linear.

```
sprs(P(subs(f,x=x^2)),P(subs(g,x=x^2)));
```

$$-6996947332x^4 + 4349515217x^2 + 937293661$$
$$-25802760209114564x^2 + 25715254854027676$$
$$-13721194015962666232246$$

Aufgabe 5 Zeigen Sie, dass in einer Subresultanten-PRS das Koeffizientenwachstum immer linear ist, indem Sie aus den Rekursionsbeziehungen $l_i = (\delta_1 + 2\delta_2 + \ldots + 2\delta_{i-1} + 2)\, l$ und $l(h_i) = l_i - 2l$ herleiten.

Hier ist die Komplexitätsabschätzung für normale reduzierte PRS, d.h. $\delta_i = 1$ und $l_i = (2i-1)\, l$ für alle $i > 1$. Wegen $l(\beta_i = lc(R_{i-1})^2) = 2\, l_{i-1} = (4i-6)\, l$ erhalten wir aus obiger Formel

$$C_{\mathrm{RedPRS}}(d) = \sum_{i=1}^{d} \big((d-i+2)\, C_R^*((4i-2)\, l, (2i-3)\, l) + (d-i)\, C_R^*((4i-6)\, l, (2i+1)\, l)$$
$$+ 2\,(d-i+1)\, C_R^*((4i-4)\, l, (2i-1)\, l)\big).$$

Für einen solchen Summanden s_i ergibt sich über $R = k[x_1, \ldots, x_{n-1}]$ für den Gradvektor $l = (d_1, \ldots, d_{n-1})$ mit $D = d_1 \cdot \ldots \cdot d_{n-1}$ wegen $C_R^*(u\, l, v\, l) = (uv)^{n-1} \cdot D^2$ (klassische Multiplikation)

$$\begin{aligned} s_i &= \big((d-i+2)\,((2i-3)(4i-2))^{n-1} + (d-i)\,((4i-6)(2i+1))^{n-1} \\ &\qquad + 2\,(d-i+1)\,((4i-4)(2i-2))^{n-1}\big)\, D^2 \\ &\sim 4 \cdot (8\, i^2)^{n-1}(d-i)\, D^2 \end{aligned}$$

Summieren wir den Aufwand der einzelnen Schritte, so erhalten wir

$$\sum_{i=1}^{d-1} s_i = O(d^{2n} D^2).$$

Mit schneller Multiplikation $C_R^*(u\, l, v\, l) = \widetilde{O}\left(u^{n-1} \cdot D\right)$ für $u \geq v$ erhalten wir

$$\begin{aligned} s_i &\sim \big((d-i+2)\,(4i-2)^{n-1} + (d-i)\,(4i-6)^{n-1} + 2\,(d-i+1)\,(4i-4)^{n-1}\big)\, D \\ &\sim 4 \cdot (4\, i)^{n-1}(d-i)\, D \end{aligned}$$

und

$$\sum_{i=1}^{d-1} s_i = \widetilde{O}(d^{n+1}D).$$

Satz 23 *Sei $R[x_n]$ der univariate Polynomring über $R = k[x_1, \ldots, x_{n-1}]$ und einem Grundkörper k mit Einheitskostenarithmetik, $f, g \in R[x_n]$ zwei Polynome mit durch $(d_1, \ldots, d_n)$ beschränkten Gradvektoren und damit $D = d_1 \cdot \ldots \cdot d_{n-1}$ eine obere Schranke für die Bitgröße $L(f)$ bzw. $L(g)$ eines Koeffizienten.*

Zur Berechnung der Subresultanten-PRS werden dann $O(D^2\, d_n^{2n})$ (klassisch) bzw. $\widetilde{O}(D\, d_n^{n+1})$ (schnelle Multiplikation) Elementarmultiplikationen benötigt.

Aufgabe 6 Zeigen Sie, dass die entsprechende Komplexität über $R = \mathbb{Z}$ für Polynome mit Koeffizientenlängen, die durch $t(f), t(g) \leq b$ beschränkt sind, für die klassische Multiplikation von der Größenordnung $O(b^2 d^4)$ und über $R = \mathbb{Z}[x_1, \ldots, x_{n-1}]$ von der Größenordnung $O(D^2\, d_n^{2n+2})$ mit $D = b \cdot d_1 \cdot \ldots \cdot d_{n-1}$ ist. Auch hier ist D eine obere Schranke für die Bitgröße $L(f)$ eines Koeffizienten von $f \in R[x_n]$.

Dies bestätigt die oben beschriebene Heuristik, dass die Berücksichtigung der Koeffizientenlänge der Hinzunahme einer zusätzlichen Variablen entspricht, ein weiteres Mal.

Dies ist allerdings noch nicht der Gesamtaufwand für die Berechnung des gcd, da nach der Formel

$$\gcd_{R[x]}(f, g) = \gcd_R(\mathrm{cont}(f), \mathrm{cont}(g)) \cdot \mathrm{pp}(\gcd_{K[x]}(f, g))$$

noch die Berechnung sowohl des Inhalts-gcd als auch des primitiven Teils aussteht.

$h(x) := \mathrm{pp}(\gcd_{K[x]}(f(x), g(x)))$ ergibt sich aus dem letzten nicht verschwindenden Rest $R_k(x)$ einer PRS durch Ausdividieren des Inhalts. Wir hatten gesehen, dass dieser Inhalt durch das intermediäre Koeffizientenwachstum eine beträchtliche Bitlänge erreichen kann, während das Ergebnis $h(x)$ als gemeinsamer Teiler von $f(x)$ und $g(x)$ nur moderate Koeffizientenlängen erwarten lässt. Es ergibt sich also wiederum die Frage, ob ein großer Teil der zusätzlichen Faktoren in $R_k(x)$ *a priori* bekannt ist und somit ohne weitere Rechnungen ausdividiert werden kann. Wegen $h(x) \mid \gcd_{R[x]}(f, g)$ gilt $lc(h) \mid r := \gcd_R(lc(f), lc(g))$ und wir können $R_k(x)$ zunächst durch das Polynom $R(x) = \frac{r}{lc(R_k)} \cdot R_k(x)) \in R[x]$ ersetzen, ehe wir den primitiven Teil berechnen. Die Koeffizienten von $R_k(x)$ sind für die Subresultanten-PRS durch einen Gradvektor $l_k = (2k-1) \cdot l$ beschränkt, die Koeffizienten von $R(x)$ wie r durch den Gradvektor $l = (d_1, \ldots, d_{n-1})$ selbst. Die Gesamtkosten für die maximal $d = d_n$ Koeffizientendivisionen kann man damit nach oben abschätzen durch

$$d \cdot C_R^*((2d-1)l, l) \sim d\,(2d-1)^{n-1}\, D^2,$$

was die Größenordnung der Kosten des PRS-Berechnung nicht übersteigt.

Es bleiben die Berechnungen von $\mathrm{cont}(f, g)$ sowie $\mathrm{pp}(R(x))$ auszuführen. Dazu sind maximal $3\, d_n$ gcd's von Polynomen in $n-1$ Variablen auszuführen, deren Gradvektor durch $(d_1, \ldots, d_{n-1})$ beschränkt ist. Mit der probabilistischen Methode der Berechnung des gcd von Familien von Polynomen lässt sich dieser Aufwand weiter senken.

Auch in der hier betrachteten Form mit maximal $3\, d_n$ gcd-Berechnungen ist er allerdings nicht entscheidend für die Laufzeit. Bezeichnet nämlich $C_{\gcd}^{(n)}$ die Komplexität der gcd-Berechnung

von Polynomen $f, g \in k[x_1, \ldots, x_n]$, deren Gradvektor durch $(d_1, \ldots, d_n)$ beschränkt ist, so erhalten wir für den Gesamtaufwand der gcd-Berechnung (klassische Multiplikation)

$$C^{(n)}_{\gcd} \lesssim 3d_n \cdot C^{(n-1)}_{\gcd} + (d_1 \cdot \ldots \cdot d_{n-1})^2 (d_n)^{2n}$$

und daraus rekursiv

$$C^{(n)}_{\gcd} \lesssim \left((d_n)^{2n-2} + 3(d_{n-1})^{2n-4} + 9(d_{n-2})^{2n-6} + \ldots\right)(d_1 \cdot \ldots \cdot d_n)^2$$

Für die schnelle Multiplikation erhalten wir

$$C^{(n)}_{\gcd} \lesssim 3d_n \cdot C^{(n-1)}_{\gcd} + d_1 \cdot \ldots \cdot d_{n-1}\,(d_n)^{n+1}$$

und daraus rekursiv

$$C^{(n)}_{\gcd} \lesssim \left((d_n)^{n} + 3(d_{n-1})^{n-1} + 9(d_{n-2})^{n-2} + \ldots\right)(d_1 \cdot \ldots \cdot d_n)$$

Satz 24 *Der Gesamtaufwand für die gcd-Berechnung von $f, g \in k[x_1, \ldots, x_n]$ über einem Körper k mit Einheitskostenarithmetik mit durch $(d_1, \ldots, d_n)$ beschränkten Gradvektoren über die Subresultanten-PRS ist bei klassischer Multiplikation von der Größenordnung*

$$O((d_1 \cdots d_n)^2 \cdot \max\{d_i^{2i-2},\ i = 1, \ldots, n\})$$

und bei schneller Multiplikation von der Größenordnung

$$\widetilde{O}((d_1 \cdots d_n) \cdot \max\{d_i^{i},\ i = 1, \ldots, n\})$$

Insbesondere bestätigen diese Schranken, dass man die Variablen so anordnen sollte, dass $d_1 \geq d_2 \geq \ldots \geq d_n$ gilt. Mit $O(d_1^2)$ für $n = 1$ und $O(d_1^2\, d_2^4)$ für $n = 2$ ergeben sich die früheren Komplexitätsabschätzungen für klassische Multiplikation als Spezialfälle.

Mit dem probabilistischen Verfahren für Polynomfamilien zur gcd-Berechnung der Koeffizienten erhält man dasselbe Ergebnis, da die Rechenzeit durch die PRS-Berechnung dominiert wird.

3.7 Rechnen in homomorphen Bildern

Die bisherigen Verfahren zur gcd-Berechnung zeichnen sich durch außerordentliches Wachstum der intermediären Daten aus. Dieser für viele symbolische Rechnungen typische Effekt lässt sich oft dadurch umgehen, dass man die entsprechenden symbolischen Rechnungen statt mit Unbestimmten mit konkreten Parameterwerten ausführt, d. h. in einem Bildbereich rechnet, und aus diesem Ergebnis z. B. durch Interpolationstechniken versucht, das entsprechende korrekte symbolische Ergebnis zu gewinnen.

Wir wollen diese Technik zunächst beispielhaft im Bereich $\mathbb{Z}[x]$ untersuchen und die Beziehung zwischen gcd-Berechnungen über $\mathbb{Z}$ und über $\mathbb{Z}_p$ studieren.

Die folgende MuPAD-Funktion Q bildet Polynome $f \in \mathbb{Z}[x]$ durch Koeffizientenreduktion auf Polynome $f_p \in \mathbb{Z}_p[x]$ ab

```
Qp:=proc(f,p)
begin Dom::UnivariatePolynomial(x,Dom::IntegerMod(p))(f) end_proc;
```

Rufen wir die wie folgt modifizierte Funktion `nEuklid`

```
nEuklid := proc(f,g) local P0,r;
begin
  P0:=domtype(g);
  while not iszero(g) do
   r:=P0::prem(f,g);
   if not iszero(r) then r:=r/lcoeff(r) end_if;
   f:=g; g:=r;
  end;
  f;
end;
```

mit den so transformierten Polynomen auf, so erhalten wir für unser erstes Beispiel

```
f:=x^8+x^6-3*x^4-3*x^3+8*x^2+2*x-5;
g:=3*x^6+5*x^4-4*x^2-9*x+21;
primes:=[2,3,5,7,11,13];
map(primes,p->[p,expr(nEuklid(Qp(f,p),Qp(g,p)))]);
```

$$\left[2, x + x^2 + 1\right],\ [3, 1],\ [5, 1],\ [7, x + 3],\ [11, 1],\ [13, 1]$$

Wir sehen an diesem Beispiel, dass für einige p auch f_p und g_p teilerfremd sind, während für andere p der modulare gcd zu hohen Grad hat. Diese Beobachtung wollen wir nun an Hand einer größeren Liste von Primzahlen systematisch vertiefen:

```
primes:=select([i$i=1..100],isprime);
map(primes,p->[p,expr(gcd(Q(f,p),Q(g,p)))]);
```

`gcd` erkennt dabei am Parametertyp, dass modulare gcd zu berechnen sind. Für unser erstes Beispiel erhalten wir stets gcd $= 1$ außer für $p = 2$ (gcd $= x^2 + x + 1$) und $p = 7$ (gcd $= x + 3$). Für Polynome, die in $\mathbb{Z}[x]$ nicht teilerfremd sind, ergibt sich folgendes Bild:

```
map(primes,p->[p,expr(gcd(Qp(f*(3*x+5),p),Qp(g*(3*x+5),p)))]);
```

$$\left[\left[2, x^3 + 1\right],\ [3, 1],\ [5, x],\ \left[7, x^2 + 5\right],\ [11, x + 9],\ [13, x + 6]\ \ldots\right]$$

In den meisten Fällen hat der modulare gcd auch den Grad 1, in einem Fall ($p = 3$) allerdings den Grad 0, in einigen anderen ($p = 2, 7$) einen höheren Grad. Aber auch die Ergebnisse im Grad 1 können nicht ohne Weiteres zur Bestimmung des gcd über $\mathbb{Z}$ herangezogen werden, da ihr Leitkoeffizient auf 1 normiert wurde, gcd $= 3\,x + 5$ aber den (in praktischen Rechnungen vorab unbekannten) Leitkoeffizienten 3 hat. Die Ergebnisrekonstruktion aus den modularen Resultaten ist also komplizierter als im Fall der modularen Determinantenberechnung. In diesem Fall genügt es allerdings, den Leitkoeffizienten des Ergebnisses auf das zu erwartende Ergebnis zu renormieren:

```
map(primes,p->[p,expr(3*gcd(Qp(f*(3*x+5),p),Qp(g*(3*x+5),p)))]);
```

$$\left[[2, x^3 + 1], [3, 0], [5, -2\,x], [7, 3\,x^2 + 1], [11, 3\,x + 5], [13, 3\,x + 5], \ldots\right]$$

Der Reduktionssatz

Wir wollen nun die zu untersuchende Situation allgemein mathematisch beschreiben. Seien R und R' zwei Integritätsbereiche, die durch einen Homomorphismus $\phi:\ R \to R'$ verbunden sind. Ein solcher Homomorphismus induziert durch seine Wirkung auf den Koeffizienten einen Ringhomomorphismus $\phi:\ R[x] \to R'[x]$, den wir mit demselben Symbol bezeichnen wollen. Einen solchen Morphismus bezeichnet man auch als *Reduktionsmorphismus*.

Satz 25 (Reduktionssatz) *([4, thm 6.26])*
Sei $\phi : R[x] \to R'[x]$ ein Reduktionsmorphismus, $0 \neq f, g \in R[x]$ mit $\phi\left(lc(f)\, lc(g)\right) \neq 0$. Sei weiter $h = \gcd(f,g)$, $f = h \cdot f'$, $g = h \cdot g'$ und $h^* = \gcd(\phi(f), \phi(g))$. Dann gilt

1. $\deg_x(\phi(f)) = \deg_x(f),\ \deg_x(\phi(g)) = \deg_x(g),\ \deg_x(\phi(h)) = \deg_x(h)$,
2. $lc(h) \,|\, \gcd(lc(f), lc(g))$,
3. $\deg_x(h^*) \geq \deg_x(h)$,
4. $\deg_x(h^*) = \deg_x(h) \Leftrightarrow \phi(r) \neq 0$, *wobei* $r = res(f', g') \in R$ *die Resultante der teilerfremden Polynome* f', g' *bezeichnet.*

Reduktionen, für welche $\deg_x(h^*) > \deg_x(h)$ *gilt, bezeichnet man als* schlechte Reduktionen *für* (f, g).

Beweis: Die ersten drei Beziehungen folgen sofort aus der Bedingung an die Leitkoeffizienten von f und g und $h \,|\, f, g$, was $\phi(h) \,|\, \phi(f), \phi(g)$ und somit $\phi(h) \,|\, h^*$ nach sich zieht.
Für die letzte Aussage halten wir zunächst die Beziehung

$$h^* = \gcd(\phi(f), \phi(g)) = \phi(h) \cdot \gcd(\phi(f'), \phi(g'))$$

fest. Die Grade der beiden Polynome h^* und h stimmen also genau dann überein, wenn $\deg_x(\gcd(\phi(f'), \phi(g'))) = 0$ gilt, was nach dem Resultantenkriterium 20 gleichbedeutend mit $\mathrm{res}(\phi(f'), \phi(g')) \neq 0$ ist. Nun ergibt sich aber die Resultante als Determinante der Sylvestermatrix, die aus den Koeffizienten von $\phi(f')$ und $\phi(g')$ zusammengestellt wird. Diese Koeffizienten ergeben sich wiederum als Reduktion der Koeffizienten von f' und g', so dass folglich $\mathrm{res}(\phi(f'), \phi(g')) = \phi(r)$ gilt. $\square$
Die Bedingung an die Leitkoeffizienten lässt sich noch leicht abschwächen. Andererseits hatten wir oben gesehen, dass im Fall $\phi\left(lc(f)\right) = \phi\left(lc(g)\right) = 0$ der Grad des modularen gcd in der Tat kleiner als erwartet sein kann.
Als erste Folgerung aus Satz 25 ergibt sich für primitive Polynome

Folgerung 5 *Sind $0 \neq f, g \in R[x]$ primitive Polynome und $\phi(f), \phi(g)$ unter einer Reduktion (mit $\phi\left(lc(f)\, lc(g)\right) \neq 0$) teilerfremd, so auch f und g.*

Neben der Koeffizientenbereichsreduktion $\mathbb{Z} \to \mathbb{Z}_p$ spielt auch $R = k[x_1, \ldots, x_{n-1}]$ und $x = x_n$ in Anwendungen eine wichtige Rolle. Ist K/k ein Erweiterungskörper und $a = (a_1, \ldots, a_{n-1}) \in K^{n-1}$, so können wir den durch $x_i \mapsto a_i, i = 1, \ldots, n-1$, induzierten Evaluierungshomomorphismus $\phi_a : R[x] \to K[x]$ betrachten.

3.8 Modulare gcd-Berechnung

Wir wollen nun diese Technik der Reduktion auf die Berechnung von gcd in $\mathbb{Z}[x]$ anwenden, indem wir Reduktionsmorphismen $\mathbb{Z} \to \mathbb{Z}_p$ für Primzahlen p heranziehen. Diese Situation hat mit Blick auf die Darstellung ganzer Zahlen in Positionssystemen bzgl. einer Basis β viel Ähnlichkeit mit der Berechnung von gcd in einem bivariaten Polynomring $S = k[y,x] = k[y][x] = R[x]$ mit $R = k[y]$ über einem Koeffizientenkörper k mit Einheitskostenarithmetik. Allein die gelegentlich auftretenden Überträge beim Rechnen mit ganzen Zahlen machen die Situation für $R = \mathbb{Z}$ unübersichtlicher.

In der Tat zeigt das Beispiel $f = (x+1)^2 = x^2+2\,x+1,\ h = (x+1)^2\,(x-1) = x^3+x^2-x-1$, dass es in $\mathbb{Z}[x]$ Polynome f, g, h mit $f \cdot g = h$ gibt, wo der Faktor f betragsmäßig größere Koeffizienten hat als das Produkt h.

Für $f, g \in k[y][x]$ mit $f \cdot g = h$ gilt dagegen stets $\deg_y(f) \le \deg_y(h)$. Wir wollen deshalb zunächst die Situation für bivariate Polynome $f, g \in k[y][x]$ über einem Koeffizientenkörper k mit Einheitskostenarithmetik studieren.

3.8.1 Modulare bivariate gcd-Berechnung

Seien k ein Körper mit Einheitskostenarithmetik, $R = k[y]$ und $0 \neq f, g \in R[x]$ zwei primitive Polynome in rekursiver Darstellung. R ist ein Euklidscher Ring, in welchem der Satz von der Division mit Rest (für univariate Polynome) gilt. Damit lassen sich in R gcd mit dem Euklidschen Algorithmus berechnen. gcd-Berechnungen in $R[x]$ können dagegen aufwändig werden, da die y-Grade der Koeffizienten intermediärer Polynome – wie auch die ganzzahligen Koeffizienten in den bisherigen Beispielen – die Tendenz haben zu wachsen.

Für die Endergebnisse der gcd-Berechnung gilt allerdings die folgende triviale Schranke für das Maximum der y-Grade der Koeffizienten:

Lemma 5 *Sei* $R = k[y],\ h, f \in R[x]$ *und* $h \mid f$. *Dann gilt* $\deg_y(h) \le \deg_y(f)$.

Beweis: Betrachte h, f als Elemente von $k[x][y]$. $\square$

Kennen wir eine Schranke c für die y-Grade der Ergebnisse, so können wir alle intermediären Ergebnisse modulo eines Polynoms $p = y^c - r(y) \in R$ mit $c > \deg_y(r)$ reduzieren und so das Gradwachstum begrenzen. Dies entspricht der Reduktion $\phi : R \to R' = R/(p)$. Rechnerisch bedeutet dies, die algebraische Ersetzungsrelation $y^k \mapsto r(y)$ auf die Koeffizienten eines Polynoms $f \in R[x]$ anzuwenden. Ist $\deg_y(f) < c$, so ändert sich das Polynom bei dieser Reduktion nicht. Wir nennen deshalb ein solches Polynom *reduziert*. Dies definiert zugleich eine Liftungsabbildung $\psi : R'[x] \to R[x]$, bzgl. welcher $\psi(\phi(f)) \equiv f \pmod p$ für $f \in R[x]$ und $\psi(\phi(f)) = f$ für reduzierte Polynome gilt.

Setzen wir zusätzlich voraus, dass p ein Primpolynom ist, wird R' ein Körper und die gcd-Berechnung in $R'[x]$ kann mit `nEuklid` ausgeführt werden. Dazu müssen in R' insbesondere inverse Elemente berechnet werden. Das kann über den Erweiterten Euklidschen Algorithmus erfolgen: Ist $b \in R, b \not\equiv 0 \pmod p$, so liefert die Darstellung $1 = \gcd(b,p) = u \cdot b + v \cdot p$ ein Polynom $u \in R,\ \deg_y(u) < c$ mit $b \cdot u \equiv 1 \pmod p$. Additive Arithmetikoperationen in R' sind also mit dem Aufwand $O(c)$, multiplikative Arithmetikoperationen einschließlich der Inversenbildung mit dem Aufwand $O(c^2)$ (klassische Arithmetik) ausführbar.

Wir wollen p von so hohem Grad wählen, dass f, g reduziert sind[5], $h^* = \gcd(\phi(f), \phi(g))$ bestimmen und untersuchen, wann daraus Rückschlüsse auf $h = \gcd(f, g)$ möglich sind. Wir verwenden die im Reduktionssatz eingeführten Bezeichnungen. Weiter seien f^*, g^* reduzierte Polynome mit $\phi(f) = \phi(f^*)\, h^*,\ \phi(g) = \phi(g^*)\, h^*$, die sich als Liftungen von Kofaktoren von h^* berechnen lassen.

Satz 25 (4) besagt, dass ein zu hoher x-Grad von h^* nur für $\phi\,(\mathrm{res}(f', g')) = 0$ zu erwarten ist. Da wir f' und g' aber erst mit h kennen, können wir diese Ausnahmen nicht apriori ausfiltern. Wir wiederholen deshalb die Rechnungen mit zufällig gewählten Primpolynomen $p \in R$ ausreichend hohen Grades c, bis $\deg_x(h^*) = \deg_x(h)$ gilt. Dann gilt $\phi(h) = \phi(\alpha)\, h^*$ mit $\alpha = lc(h)$, wenn wir davon ausgehen, dass h^* als gcd zweier univariater Polynome über dem Körper R' so normiert ist, dass $lc(h^*) = 1$ gilt.

Ist h zugleich reduziert, so wäre h wegen $h = \psi(\phi(h))$ gefunden. Da wir α aber nicht kennen, berechnen wir $w = \psi\,(\phi(\beta)\, h^*)$ mit $\beta = \gcd(lc(f), lc(g))$. Dann gilt

$$\phi(w) = \phi\left(\frac{\beta}{\alpha}\, h\right) = \phi(\beta)\, h^*$$

Da wegen $\alpha \,|\, \beta$ das Polynom $\frac{\beta}{\alpha}\, h$ in $R[x]$ liegt, gilt $w = \frac{\beta}{\alpha}\, h$, wenn beide Polynome reduziert sind. Wegen $\deg_y(h) \le \deg_y(f)$ ist $c > \deg_y(f) + \deg_y(\beta)$ eine dafür hinreichende Bedingung.

Der folgende Algorithmus liefert also für das erste Polynom p, für welches $\deg_x(h^*) = \deg_x(h)$ gilt, das gesuchte Ergebnis.

modularBivariateGCD(f,g)

Input: $R = k[y],\ f, g \in R[x]$ primitive Polynome,
$\deg_x(f), \deg_x(g) < d_1,\ \deg_y(f), \deg_y(g) < d_2$

Output: $h = \gcd(f, g) \in R[x]$

β `:=` $\gcd(lc_x(f), lc_x(g)) \in R$`;`
`repeat`
  `Wähle zufällig ein irreduzibles Polynom` $p = y^c - r(y) \in R$`,`
    `vom Grad` $c = d_2 + \deg(\beta) > \deg_y(r)$`.`
  `Berechne` $h^* = \gcd(\phi(f), \phi(g))$ `in` $R'[x]$ `mit` $lc(h^*) = 1$
    `mit dem Euklidschen Algorithmus.`
  `if` $\deg(h^*) = 0$ `then return` 1 `(* Polynome sind teilerfremd *)`
  `Bestimme das reduzierte Polynom` $w = \psi\,(\phi(\beta)\, h^*) \in R[x]$`.`
`until` $w \,|\, \beta f,\ w \,|\, \beta g$`.`
`return` $\mathrm{pp}_x(w)$`.`

Beispiel:

```
f:=(x*y+y+1)*(x^3+y^2-1);
g:=(x*y+y+1)*(x^2+y^3-1);
```

[5]Damit ist auch die Bedingung an die Leitkoeffizienten für das Vorliegen einer guten Reduktion automatisch erfüllt.

```
Qy:=Dom::UnivariatePolynomial(y,Dom::Rational);
Qx:=Dom::UnivariatePolynomial(x,Qy);

fx:=Qx(f);
gx:=Qx(g);
```

$$f = y\,x^4 + x^3\,(y+1) + x\,\left(y^3 - y\right) + \left(y^3 + y^2 - y - 1\right)$$
$$g = y\,x^3 + x^2\,(y+1) + x\,\left(y^4 - y\right) + \left(y^4 + y^3 - y - 1\right)$$

In diesem Fall ist also $\beta = y$ und $c = d_2 + 1 = 6$, so dass wir das irreduzible Polynom $p = y^6 - y + 1$ verwenden können.
Aus verfahrenstechnischen Gründen verwenden wir die Variable t für die Rechnungen in R'. Die Abbildung $\phi : R = \mathbb{Q}[y] \to R' = \mathbb{Q}[t]/(t^6 - t + 1)$ wird also durch $y \to t$ induziert.

```
RR:=Dom::AlgebraicExtension(Dom::Rational, t^6-t+1,t);
Qxx:=Dom::UnivariatePolynomial(x,RR);
fxx:=Qxx(subs(f,y=t));
gxx:=Qxx(subs(g,y=t));
```

Bereits die Berechnung des ersten Rests unterscheidet sich in $R[x]$ und $R'[x]$:

```
rx:=Qx::prem(fx,gx);
rxx:=Qxx::prem(fxx,gxx);
```

$$\left(-y^6 + y^3\right)\,x^2 + \left(-y^6 + y^2\right)\,x + \left(y^5 + y^4 - y^3 - y^2\right)$$
$$\left(t^3 - t + 1\right)\,x^2 + \left(t^2 - t + 1\right)\,x + \left(t^5 + t^4 - t^3 - t^2\right)$$

`rx/lcoeff(rx)` kann nicht berechnet werden, weil die anderen Koeffizienten in R nicht durch den Leitkoeffizienten teilbar sind. Über R' als Körper kann dagegen der Leitkoeffizient invertiert werden.

```
rxx/lcoeff(rxx)
```

$$x^2 + \left(-\frac{4}{7}\,t^5 - \frac{1}{7}\,t^4 - \frac{2}{7}\,t^3 + \frac{3}{7}\,t^2 - \frac{1}{7}\,t + \frac{9}{7}\right)\,x + \left(\frac{8}{7}\,t^5 + \frac{2}{7}\,t^4 - \frac{3}{7}\,t^3 + \frac{1}{7}\,t^2 + \frac{2}{7}\,t - \frac{11}{7}\right)$$

Damit können wir den normierten Euklidschen Algorithmus `nEuklid` anwenden

```
hxx:=nEuklid(fxx,gxx);
```

und erhalten die Restefolge

$$x^2 + \left(-\frac{4}{7}\,t^5 - \frac{1}{7}\,t^4 - \frac{2}{7}\,t^3 + \frac{3}{7}\,t^2 - \frac{1}{7}\,t + \frac{9}{7}\right)\,x + \left(\frac{8}{7}\,t^5 + \frac{2}{7}\,t^4 - \frac{3}{7}\,t^3 + \frac{1}{7}\,t^2 + \frac{2}{7}\,t - \frac{11}{7}\right)$$
$$x + \left(-t^5 + 2\right)$$

Multiplikation mit $\phi(\beta) = t$ ergibt bereits einen brauchbaren Wert für $\phi(w)$

`t*hxx`

$$t\,x + (t+1)$$

aus dem sich durch Liften $w = \psi(t\,x + (t+1)) = y\,x + (y+1)$, ein gemeinsamer Teiler von $\beta\,f$ und $\beta\,g$, ergibt. Es gilt also

$$\gcd(f,g) = \mathrm{pp}_x(w) = y\,x + (y+1)\,.$$

Kommen wir zu einer Komplexitätsabschätzung dieses Verfahrens, wobei wir $c = 2\,d_2$ als sichere obere Schranke annehmen wollen.

Zunächst schätzen wir die Zahl der Durchläufe der `repeat`-Schleife ab. Haben wir nach s Durchläufen noch immer kein verwertbares Ergebnis gefunden, muss jedesmal $\deg_x(h^*) > \deg_x(h)$ gewesen sein. Nach Satz 25 ist dies nur möglich, wenn $\phi(r) = 0$ für $r = \mathrm{res}(f', g')$ gilt. Aus der Gestalt der Sylvestermatrix erhalten wir $\deg(r) < 2\,d_1\,d_2$, so dass wir nach maximal $s = \frac{2\,d_1\,d_2}{c} = d_1$ Durchläufen auf eine gute Reduktion stoßen. Zufällige Wahl von p liefert mit Wahrscheinlichkeit 1 eine gute Reduktion.

Die Kosten für den Euklidschen Algorithmus für Polynome vom Grad $< d_1$ unter Einheitskosten sind von der Größenordnung $O(d_1^2)$. Im Fall der Rechnungen in $R'[x]$ sind die Arithmetikkosten nicht durch $O(1)$, sondern durch $O(c^2)$ uniform beschränkt, so dass die Kosten für die Berechnung von h^* die Größenordnung $O(c^2\,d_1^2) = O(d_1^2 d_2^2)$ haben. Die Kosten der anderen Schritte übersteigen diese Größenordnung nicht. Insbesondere ist der Aufwand für die abschließende Teilbarkeitsprüfung von derselben Größenordnung (klassische Multiplikation), denn die Gradvektoren sind durch $(2d_1, d_2)$ beschränkt.

Wir haben damit den folgenden Satz bewiesen.

Satz 26 *$f, g \in R[x]$ seien zwei primitive Polynome, deren Größe durch den Gradvektor (d_1, d_2) beschränkt ist.*

1. *Die Hauptschleife von* `modularBivariateGCD(f,g)` *berechnet $h = \gcd(f,g)$ für jede gute Reduktion korrekt.*
2. *Die Wahrscheinlichkeit, dass bereits die erste Reduktion gut ist, ist gleich 1, da es nur endlich viele schlechte Reduktionen gibt. Nach maximal d_1 Durchläufen wird garantiert eine gute Reduktion erreicht.*
3. *Die Kosten für einen Durchlauf der Hauptschleife sind von der Größenordnung $O(d_1^2 d_2^2)$, die Kosten im schlechtesten Fall – $O(d_1)$ schlechte Reduktionen – also $O(d_1^3 d_2^2)$.*

Die Komplexitätsabschätzung ignoriert noch die Kosten, die erforderlich sind, um ein zufälliges Primpolynom zu finden und dessen Primpolynom-Eigenschaft zu verifizieren.

Gegenüber den klassischen Verfahren mit einer Komplexität von (wenigstens) $O(d_1^2 d_2^4)$ (für $d_1 \geq d_2$) ergibt sich, unbeachtlich der letzten Frage, ein deutlicher Vorteil.

3.8.2 Modulare gcd-Berechnung über $\mathbb{Z}$ (big prime)

Ähnlich können wir im Fall univariater ganzzahliger Polynome verfahren. Seien $0 \neq f, g \in \mathbb{Z}[x]$ zwei primitive Polynome und p eine genügend große Primzahl. Zum Reduktionsmorphismus

$\phi : \mathbb{Z} \to \mathbb{Z}_p$ gibt es auch hier eine inverse Abbildung $\psi : \mathbb{Z}_p \to \mathbb{Z}$ mit $\psi(\phi(f)) = f$ für reduzierte Polynome $f \in \mathbb{Z}[x]$. Ein Polynom heißt dabei *reduziert*, wenn jeder Koeffizient im Intervall $\left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$ liegt.

modularBigPrimeGCD(f,g)

Input: $f, g \in \mathbb{Z}[x]$ primitive Polynome
Output: $h = \gcd(f, g) \in \mathbb{Z}[x]$

β := $\gcd(lc(f), lc(g)) \in \mathbb{Z}$;
repeat
  Wähle zufällig eine genügend große Primzahl $p \in \mathbb{Z}$.
  Berechne $h^* = \gcd(\phi(f), \phi(g))$ mit $lc(h^*) = 1$
        mit dem Euklidschen Algorithmus in $\mathbb{Z}_p[x]$.
  if $\deg(h^*) = 0$ then return 1 (* Polynome sind teilerfremd *)
  Bestimme das reduzierte Polynom $w = \psi\left(\phi(\beta)\, h^*\right) \in \mathbb{Z}[x]$.
until $w \,|\, \beta f,\ w \,|\, \beta g$.
return $\mathrm{pp}_x(w)$.

Beispiel:

$$f = 12x^3 + 6x^2 + 14x + 7,\ g = 30x^3 + 15x^2 + 2x + 1$$

Der gcd dieser beiden Polynome ist $2x + 1,\ \beta = 6$.

```
reducedpoly:=proc(f,m)
begin _plus(mods(expr(coeff(f,i)),m)*x^i $ i=0..degree(f)) end;

Check:=proc(beta,f,g,p) local h,w;
begin
  h:=gcd(Qp(f,p),Qp(g,p));
  w:=reducedpoly(beta*h,p);
  [p,w,normal(beta*f/w),normal(beta*g/w)];
end;

f:= 12*x^3 + 6*x^2 + 14*x + 7;
g:= 30*x^3 + 15*x^2 + 2*x + 1;
primes:=select([$5..50],isprime);
map(primes,p->Check(6,f,g,p));
```

Die Ergebnisse zeigen, dass $w = 6\,x + 3$ für $p \geq 13$ stabil gefunden wird. $p = 11$ ist eine schlechte Reduktion, da $\deg(h^*) > \deg(h)$ ist, $p = 5$ und $p = 7$ sind zu klein, so dass $\phi(\beta)\, h^*$ nicht zu w geliftet wird.

Satz 27 *$f, g \in \mathbb{Z}[x]$ seien zwei primitive Polynome vom Grad $< d$ mit Koeffizienten, deren Bitgröße durch b beschränkt sei und p eine ausreichend große Primzahl der Bitgröße c. Mit den Bezeichnungen aus dem Reduktionssatz gilt*

1. *p ist eine schlechte Reduktion $\Leftrightarrow p \,|\, \mathrm{res}(f', g')$. Die Anzahl schlechter Reduktionen der Größenordnung c ist also durch $\widetilde{O}\left(\frac{b\,d}{c}\right)$ beschränkt.*

2. *Die Hauptschleife von* `modularBigPrimeGCD(f,g)` *berechnet h für jede gute Reduktion korrekt.*

3. *Die Kosten für einen Durchlauf der Hauptschleife und damit die bei zufälliger Wahl von p wahrscheinlichen Kosten sind von der Größenordnung $O(c^2\,d^2)$.*

4. *Im schlechtesten Fall muss man die Rechnungen mit $\widetilde{O}\left(\frac{b\,d}{c}\right)$ Primzahlen p wiederholen, bis h gefunden ist, was Kosten der Größenordnung $\widetilde{O}\left(b\,c\,d^3\right)$ verursacht.*

Die Abschätzung der Anzahl schlechter Reduktionen ergibt sich aus einer Abschätzung von $l = |\mathrm{res}(f', g')|$ als Determinante einer ganzzahligen, maximal $2\,d$-reihigen quadratischen Matrix mit Einträgen der Bitlänge b. Wie früher gezeigt, ist deren Bitlänge $\log(l) = \widetilde{O}(b\,d)$.

Die Polynome $f, g, \frac{\beta}{\alpha}h, \alpha f', \alpha g'$ sind allesamt Teiler von βf oder βg, woraus im bivariaten Fall eine Schranke für c bestimmt werden konnte, da für $w \,|\, \beta\, f$ auch $\deg_y(w) \le deg_y(\beta\, f)$ erfüllt war.

Die triviale Vermutung, dass dies für Koeffizienten von $\gcd(f, g)$ über $\mathbb{Z}$ auch gilt, d. h. dass sie nie größer sind als die von f und g, trifft nicht zu, wie das folgende Beispiel von Davenport und Trager zeigt:

Beispiel: $f := (x+1)^2 \cdot (x-1) = x^3+x^2-x-1$ und $g := (x+1)^2 \cdot (x^2-x+1) = x^4+x^3+x+1$. Es gilt $\gcd(f, g) = x^2 + 2\,x + 1$.

Wir benötigen also eine Abschätzung des möglichen Wachstums der Koeffizientengröße der Faktoren eines vorgegebenen Polynoms $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{Z}[x]$.

Diese liefert der folgende Satz

Satz 28 (Mignotte-Schranke) *Sind $f, h \in \mathbb{Z}[x]$ Polynome vom Grad $n = \deg(f)$ und $k = \deg(h)$ und gilt $h \,|\, f$ (in $\mathbb{Z}[x]$), so ist*

$$\|h\|_\infty \le \sqrt{n+1} \cdot 2^k \, \|f\|_\infty$$

wobei $\|f\|_\infty = max\,(|a_i|)$ für $f = \sum a_i\, x^i$ die Maximum-Norm des Polynoms f bezeichnet.

Beweis: [4, cor 6.33.].

Kehren wir zur Analyse des Algorithmus `modularBigPrimeGCD` zurück. Ist die Koeffizientengröße der Polynome $f, g \in \mathbb{Z}[x]$ vom Grad $\le d$ durch $\|f\|_\infty, \|g\|_\infty \le 2^b$ beschränkt, so ergibt die Mignotte-Schranke für jeden Teiler w von $\beta\, f$ oder $\beta\, g$ die Koeffizientenschranke $\|w\|_\infty \le B = \sqrt{d+1} \cdot 2^{d+2\,b}$. Wir können also $2\,B$ als untere Schranke für p und damit $c = l(\beta) + d + b \le d + 2\,b$ (statt vergleichbar $s = 2\,b$ im bivariaten Fall) wählen und werden eine gute Reduktion in der Nähe dieser Schranke finden.

Die Kosten eines Durchlaufs der Hauptschleife von `modularBigPrimeGCD` für eine gute Reduktion dieser Größe und damit die bei zufälliger Wahl von p wahrscheinlichen Kosten sind also von der Größenordnung $O(c^2\,d^2) = O(b^2\,d^2 + d^4)$, die Komplexität des gesamten Algorithmus im schlechtesten Fall ist von der Größenordnung $\widetilde{O}(b^2\,d^3 + b\,d^4)$.

3.8.3 Modulare gcd-Berechnung über $\mathbb{Z}$ (small primes)

Statt den gcd bzgl. einer großen Primzahl p gleich vollständig zu bestimmen, können wir auch versuchen, die Ergebnisse der Rechnungen bzgl. mehrerer kleiner Primzahlen zu vergleichen und Ergebnisse mit dem Chinesischen Restklassen-Algorithmus zu liften.

Bezeichne `Lift((`h_1, m_1`),(`h_2, m_2`))=(`$h, m_1\,m_2$`)` den Algorithmus, der CRA2 koeffizientenweise auf $h_1, h_2 \in \mathbb{Z}[x]$ anwendet und so das eindeutig bestimmte bzgl. m reduzierte Polynom $h \in \mathbb{Z}[x]$ berechnet, für welches $h \equiv h_1 \pmod{m_1},\ h \equiv h_2 \pmod{m_2}$ gilt.

```
CRA2:=proc(a,m1,b,m2) local c;
begin
  c:=(b-a) * modp(1/m1,m2) mod m2;
  mods(a+c*m1,m1*m2);
end_proc;

Lift:=proc(g,m1,h,m2) local d;
begin
  d:=max(degree(g),degree(h));
  _plus(CRA2(coeff(g,i), m1, coeff(h,i), m2)*x^i $i=0..d)
end;
```

Dann berechnet der folgende Algorithmus für zwei primitive Polynome $f, g \in \mathbb{Z}[x]$ deren größten gemeinsamen Teiler $\gcd_{\mathbb{Z}[x]}(f, g)$. Wir schreiben kurz $f_p = \phi_p(f)$ für $f \in \mathbb{Z}[x]$ und den natürlichen Reduktionsmorphismus $\phi_p : \mathbb{Z}[x] \to \mathbb{Z}_p[x]$.

modularSmallPrimesGCD(f,g)

Input: $f, g \in \mathbb{Z}[x]$ primitive Polynome
Output: $h = \gcd_{\mathbb{Z}[x]}(f, g)$

```
local
  M: kumulierter Modul
  p: aktuelle Primzahl
  (w, M) etc.: Lift der modularen Ergebnisse
  d: (erwarteter) Grad des gcd
```

β:=gcd$(lc(f), lc(g))$
(w, M):=(0,1) (* Reset *)
d:=deg(f) + deg(g) + 1 (* garantiert zu groß *)

```
repeat
  Wähle eine Primzahl p mit lc(f)·lc(g) ≢ 0 (mod p), gcd(p, M) = 1.
  Berechne h_p = gcd(f_p, g_p) in Z_p[x] mit lc(h_p) = 1
    mit dem Euklidschen Algorithmus.
  if deg(h_p) > d then continue (* p ist schlechte Reduktion *)
  if (deg(h_p) < d) then { d:=deg(h_p), (w, M):=(0,1) }
```

```
      (* Reset: alle bisherigen p waren schlechte Reduktionen *)
    if d = 0 then return 1 (* Polynome sind teilerfremd *)
    Bestimme das p-reduzierte Polynom w' ∈ ℤ[x] mit w'_p = β_p h_p.
    Lifte das Ergebnis: (w, M):=Lift((w, M), (w', p))
until w | βf, w | βg.
return pp_x(w)
```

Betrachten wir noch einmal das Beispiel

$$f = 12x^3 + 6x^2 + 14x + 7,\ g = 30x^3 + 15x^2 + 2x + 1$$

aus dem letzten Abschnitt. Für $p = 5$ und $p = 7$ wurden die gcd noch nicht korrekt bestimmt. Aus beiden Ergebnissen lässt sich aber $6x + 3$ rekonstruieren:

```
Lift(x-2,5,3-x,7);
```

$$6x + 3$$

In der Hauptschleife von `modularSmallPrimesGCD` wird das Ergebnis so lange kumuliert, wie die Erwartung über den Grad $d = \deg(h)$ nicht erschüttert wird. Ist d zu hoch, so kann die Terminationsbedingung aus Gradgründen nicht erfüllt sein. Stoßen wir auf eine Reduktion, die zu einem gcd von kleinerem als dem bisher erwarteten Grad d führt, so waren alle bisher betrachteten Reduktionen schlecht und wir können die entsprechenden Ergebnisse vergessen. Ist der Grad des gcd in der aktuellen Reduktion größer als d, so ist diese Reduktion schlecht und wir können sie nicht zur Verbesserung des Ergebnisses verwenden. Nach endlich vielen Schritten landen wir auf dem korrekten Grad und berücksichtigen von da ab nur noch gute Reduktionen für die Ergebnisakkumulation.

Die Kosten der Berechnung von h_p setzen sich zusammen aus $O(d\,b)$ (Berechnung der Reduktionen f_p und g_p) und $O(d^2)$ (Berechnung von $\gcd(f_p, g_p)$). Bestimmen wir zunächst h_p für genügend viele Werte von p, so dass darunter $c = b + 2\,d$ gute Reduktionen sind, und liften dann nur die guten Reduktionen zum Endergebnis, so entstehen einmalig weitere Kosten der Größe $O(d\,c^2)$. Der abschließende Teilbarkeitstest $h \,|\, \beta\, f$ in $\mathbb{Z}[x]$ verursacht Kosten $C^*_{\mathbb{Z}[x]}\,((c,d),(c,d))$, also $O(c^2\,d^2)$ mit klassischer Multiplikation.

Fassen wir diese Aussagen im folgenden Satz zusammen

Satz 29 *$f, g \in \mathbb{Z}[x]$ seien zwei primitive Polynome, b, c, f', g' wie oben.*

1. *p ist eine schlechte Reduktion $\Leftrightarrow p \,|\, \operatorname{res}(f', g')$. Die Zahl schlechter Reduktionen p von Wortgröße $O(1)$ ist beschränkt durch $\widetilde{O}(b\,d)$.*

2. *Beginnend mit der ersten guten Reduktion gilt $d = \deg(h)$ in der Hauptschleife von* `modularSmallPrimes(f,g)`*, so dass ab da alle weiteren gelifteten Ergebnisse zur Berechnung von w beitragen.*

3. *Die Hauptschleife der Prozedur* `modularSmallPrimes(f,g)` *terminiert nach spätestens c guten Reduktionen.*

4. *Der Algorithmus terminiert also bei zufälliger Wahl der p in der Regel nach c Durchläufen und verursacht maximal Kosten der Größenordnung $O(c\,(b+d)\,d + d\,c^2 + d^2\,c^2) = O(b^2\,d^2 + d^4)$ (klassische Multiplikation).*

5. *Im schlechtesten Fall terminiert das Verfahren erst nach $\widetilde{O}(b\,d+c) = \widetilde{O}(b\,d)$ Durchläufen der Hauptschleife, was Kosten der Größenordnung $\widetilde{O}(b\,d^2\,(b+d) + d\,c^2 + d^2\,c^2) = \widetilde{O}(b^2\,d^2 + d^4)$ verursacht.*

Die Hauptkosten verursacht hier nur die Probedivision am Schluss, mit welcher die Korrektheit des Ergebnisses verifiziert wird. Ohne diese entsteht nur ein Aufwand der Größenordnung $O(c\,(b+d)\,d + d\,c^2) = O(b^2\,d + d^3)$. `modularSmallPrimesGCD` ist deshalb im praktischen Einsatz besser als `modularBigPrimeGCD`, obwohl beide Laufzeitverhalten derselben Größenordnung aufweisen. Allein für den schlechtesten Fall schneidet `modularSmallPrimesGCD` besser ab, da hier der Aufwand zum Verwerfen der schlechten Reduktionen wegen der geringeren Bitgröße von p geringer ist.

gcd-Berechnung in $\mathbb{Z}[x_1, \ldots, x_n]$

In obigem Algorithmus haben wir „einfache“ gcd-Berechnungen über $\mathbb{Z}_p[x]$ zu einer gcd-Berechnung in $\mathbb{Z}[x]$ zusammengefügt. Nehmen wir an, dass wir über ein ähnlich effizientes Verfahren zur multimodularen gcd-Berechnung in $S' = \mathbb{Z}_p[x_1, \ldots, x_n]$ verfügen, so können wir dieses nach demselben Schema für die gcd-Berechnung über $S = \mathbb{Z}[x_1, \ldots, x_n]$ verwenden.

Wir fixieren dazu eine Termordnung auf $T = T(x_1, \ldots, x_n)$ bzgl. welcher alle Leitkoeffizienten, Leitterme etc. zu nehmen sind. Der gcd in S' kann ein echtes Vielfaches oder ein skalares Vielfaches des gcd in S sein. Im ersten Fall ist nicht der Grad zu groß, sondern der Leitterm.

Die folgende Modifikation liefert also den gesuchten gcd in S. lt, lc und pp beziehen sich dabei auf die distributive Darstellung der Polynome bzgl. der auf T fixierten Termordnung.

multivariateModularSmallPrimesGCD(f,g)

Input: $f, g \in S = \mathbb{Z}[x_1, \ldots, x_n]$ primitive Polynome
Output: $h = \gcd_S(f, g)$

`local`
  M`: kumulierter Modul`
  p`: aktuelle Primzahl`
  (w, M) `etc.: Lift der modularen Ergebnisse`
  $d \in T$`: (erwarteter) Leitterm des gcd`

β`:=` $\gcd(lc(f), lc(g))$
(w, M)`:= (0,1)`
`d:=` $lt(f) + lt(g)$ `(* garantiert zu groß *)`
`repeat`
  `Wähle eine Primzahl` p `mit` $lc(f) \cdot lc(g) \not\equiv 0 \pmod{p}$, $(p, M) = 1$.
  `Berechne` $h_p = \gcd(f_p, g_p)$ `in` $\mathbb{Z}_p[x_1, \ldots, x_n]$ `mit` $lc(h_p) = 1$
    `mit dem Euklidschen Algorithmus.`

```
    if lt(h_p) > d then continue (* p ist schlechte Reduktion *)
    if lt(h_p) < d then { d:=lt(h_p), (w,M):=(0,1) }
          (* alle bisherigen p waren schlechte Reduktionen *)
    if d = 1 ∈ T then return 1 (* Polynome sind teilerfremd *)
    Bestimme das p-reduzierte Polynom w' ∈ ℤ[x_1,...,x_n] mit w'_p = β_p h_p.
    Lifte das Ergebnis: (w,M):=Lift((w,M),(w',p))
until w | βf, w | βg.
return pp(w)
```

Kostenanalyse

Ist c wieder eine Schranke für die Bitlänge der Koeffizienten von Teilern von f und g in S, $(d_1, \ldots, d_n)$ eine Schranke für die Gradvektoren von f und g und vernachlässigen wir den Aufwand, der durch schlechte Reduktionen entsteht, so erhalten wir eine ähnliche Abschätzung wie oben. Die Schleife wird höchstens c mal durchlaufen, bis die notwendige Liftgenauigkeit garantiert erreicht wird. Die dazu auflaufenden Kosten für modulare gcd-Berechnungen sind $O\left(c \cdot C_{\mathrm{mod-gcd}}(d_1, \ldots, d_n)\right)$, wobei $C_{\mathrm{mod-gcd}}(d_1, \ldots, d_n)$ die Kosten einer gcd-Berechnung in S' für Polynome mit den angegebenen Gradschranken bezeichnet.

Die abschließenden Lift-Kosten berechnen sich wie oben, allerdings ist die Anzahl der zu liftenden Koeffizienten von der Größenordnung $D = d_1 \cdot \ldots \cdot d_n$, so dass die Kosten dieses Schritts von der Größenordnung $O(c^2 \cdot D)$ sind.

Führen wir die Probemultiplikation in der Testbedingung erst dann aus, wenn wir uns ganz sicher sind, entstehen (bei konventioneller Multiplikation) einmalig weitere Kosten der Größenordnung $O(c^2\, D^2)$.

Satz 30 *Der Aufwand zur Berechnung von* `multivariateModularSmallPrimesGCD(f,g)` *ist unter obigen Annahmen von der Ordnung*

$$O(c^2 \cdot D^2 + c \cdot C_{\mathrm{mod-gcd}}(d_1, \ldots, d_n))$$

3.8.4 gcd-Berechnung in $k[x_1, \ldots, x_n]$

Es ist noch die Frage offen geblieben, wie man den gcd multivariater Polynome über einem Körper $k = \mathbb{Z}_p$ berechnen kann. Dies erfolgt durch eine andere Anwendung modularer Techniken, nämlich den Übergang von multivariaten zu univariaten Polynomen durch Evaluierung.

Sei dazu k ein Körper mit Einheitskostenarithmetik, $R = k[x_1, \ldots, x_{n-1}]$, $x = x_n$ und $S = R[x]$. Die Idee ist, für $x_1, \ldots, x_{n-1}$ spezielle Werte aus k oder einer Erweiterung K/k einzusetzen, den gcd der so reduzierten Polynome in $K[x]$ zu bestimmen, und aus diesen Ergebnissen den originalen gcd zu rekonstruieren. Für $a = (a_1, \ldots, a_{n-1}) \in K^{n-1}$ bezeichnen wir die entsprechenden Reduktionsmorphismen $\phi_a : R \to K$ und $\phi_a : R[x] \to K[x]$ als *Evaluationsmorphismen.*

Ist $f \in R$ ein durch die Gradschranke $d = (d_1, \ldots, d_{n-1})$ beschränktes Polynom, so kann der Funktionswert $f(a)$ in einem Punkt $a \in K^{n-1}$ am effizientesten mit dem Hornerschema berechnet werden, welches maximal $D' = d_1 \cdot \ldots \cdot d_{n-1}$ Multiplikationen in K benötigt.

Ähnliche Laufzeit $O(D')$, allerdings mehr Speicherplatz wird benötigt, wenn zunächst die Funktionswerte der D' Monome aus $S_a = (a^\alpha,\, \alpha < d)$ bestimmt werden[6] und dann dieser Vektor mit dem Koeffizientenvektor von f multipliziert wird. S_a muss dabei nur einmal pro $a \in K^{n-1}$ berechnet werden und steht dann für die Evaluation weiterer Polynome zur Verfügung.

Die umgekehrte Aufgabe, aus Reduktionen (a, f_a) an vorgegebenen Evaluationspunkten $a \in K^{n-1}$ ein zugehöriges Polynom $f \in R$ mit diesem Evaluationsverhalten zu konstruieren, bezeichnet man als *Interpolation*. Ist für f wieder eine Gradschranke $d = (d_1, \ldots, d_{n-1})$ bekannt, so kann f mit D' unbestimmten Koeffizienten angesetzt und ein lineares Gleichungssystem gelöst werden. In der Regel werden dafür D' Evaluationspunkte benötigt, die nicht „zu speziell“ liegen dürfen, damit das entsprechende Gleichungssystem ein Lösung besitzt.

Für eine vorgegebene Menge $M \subset K^{n-1}$ von D' Evaluationspunkten und verschiedene Polynome $f, g, h, \ldots$ handelt es sich dabei immer um dasselbe $(D' \times D')$-Gleichungssystem, welches simultan mit verschiedenen rechten Seiten gelöst werden kann. Um die Lösbarkeit zu garantieren, muss die Koeffizientenmatrix des Systems maximalen Rang haben, d. h. deren Determinante muss verschieden Null sein. Das ist für Punkte *in allgemeiner Lage* immer erfüllt, da das Verschwinden der Determinante eine polynomiale Bedingung auf die Koordinaten der Punkte ist. Wir werden M Schritt für Schritt aufbauen, so dass S jeweils maximalen Rang hat. Die Zeilen der Matrix S sind gerade die Vektoren S_a, $a \in M$.

Das Gerüst des Algorithmus orientiert sich an `modularSmallPrimesGCD`.

modularEvalGCD(f,g)

Input: $f, g \in R[x]$ primitive Polynome
Output: $h = \gcd_{R[x]}(f, g)$

β:=gcd$(lc(f), lc(g))$
`M:={} (* Menge der "nützlichen" Evaluationspunkte *)`
`d:=`deg(f) + deg(g) + 1
`(* erwarteter Grad des gcd; hier garantiert zu groß *)`

`repeat`
`Wähle ein` $a \in K^{n-1}$ `mit` $\phi_a(lc(f) \cdot lc(g)) \neq 0$
`in genügend allgemeiner Lage.`
`Berechne` $w_a = \gcd(\phi_a(f), \phi_a(g))$ `in` $K[x]$ `mit` $lc(w_a) = \phi_a(\beta)$
`mit dem Euklidschen Algorithmus.`
`if` deg$(w_a) > d$ `then continue (* a ist schlechte Reduktion *)`
`if` $(\deg(w_a) < d)$ `then { d:=`deg(w_a)`, M:={} }`
`(* alle bisherigen a waren schlechte Reduktionen *)`
`if` $d = 0$ `then return` 1 `(* Polynome sind teilerfremd *)`
`M:=M` $\cup \{(a, w_a)\}$
`until` $|M| = D'$.

[6] Durch kluge Organisation der Berechnung können die $D' = |S_a|$ Werte durch D' Multiplikationen je eines bereits berechneten Werts und eines α_i bestimmt werden.

`Lifte die` $(a, w_a) \in M$ `koeffizientenweise zu` $w \in R[x]$`.`

`If` $w \mid \beta f,\ w \mid \beta g$ `then return` $\mathrm{pp}_x(w)$ `else return FAILED.`

Wir wollen diesen Algorithmus zunächst an einem Beispiel studieren. Wir betrachten die Polynome

$$f = x^3 y + 2 x^2 y^2 + x^2 + x y^3 + 2 x y + y^2,$$
$$g = x^3 y - 2 x^2 y^2 + x^2 + x y^3 - 2 x y + y^2$$

im Ring $S = \mathbb{Q}[x, y] = \mathbb{Q}[y][x]$ und führen die Haputschleife für veschiedene Evaluationspunkte $a \in \mathbb{Q}$ aus. Wegen $\beta = y$ muss $a \neq 0$ gewählt werden. Die Matrix S ist eine van der Mondesche Matrix, so dass die Rangbedingung automatisch erfüllt ist.

```
RP:=Dom::UnivariatePolynomial(x,Dom::Rational);

Check:=proc(beta,f,g,a)
begin subs(beta,y=a)*gcd(RP(subs(f,y=a)),RP(subs(g,y=a))) end;

f:=x^3*y + 2*x^2*y^2 + x^2 + x*y^3 + 2*x*y + y^2;
g:=x^3*y - 2*x^2*y^2 + x^2 + x*y^3 - 2*x*y + y^2;

Check(y,f,g,a) $ a=1..4;
```

$$x + 1, 2 x + 1, 3 x + 1, 4 x + 1$$

Die Interpolation der Koeffizienten ergibt $w = y x + 1$, was in diesem Fall auch bereits der gcd ist. Formal kann diese Interpolation durch die folgende Funktion `Lift` ausgeführt werden, die als Parameter zwei gleich lange Listen der Evaluationspunkte und -werte übergeben bekommt, das zugehörige lineare Gleichungssystem aufstellt, löst und daraus das entsprechende Polynom in $\mathbb{Q}[y]$ zusammenstellt.

```
Lift:=proc(ev,v) local g,i,sys,sol;
begin
  g:=y->_plus(a.i*y^(i-1) $ i=1..nops(ev));
  sys:=[g(ev[i])=v[i] $ i=1..nops(ev)];
  sol:=solve(sys,[a.i $ i=1..nops(ev)]);
  subs(g(y),sol[1]);
end;
```

Betrachten wir als weiteres Beispiel die folgenden Polynome ([5, S. 95])

$$f = x^3 y (2 y - 1) + x^2 y^3 + x (2 y^4 - y^3 - 6 y + 3) + y^2 (y^3 - 3),$$
$$g = x^3 (2 y - 1) + x^2 y (-y + 1) + x (-y^3 + 4 y - 2) + 2 y^2$$

Hier ist $\beta = 2y - 1$. Evaluation liefert

```
Check(2*y-1,f,g,a) $ a=1..4;
```

$$x+1,\ 3\,x+4,\ 5\,x+9,\ 7\,x+16$$

und Liften der Koeffizienten

```
ev:=[1,2,3,4];
Lift(ev,[1,3,5,7]);
Lift(ev,[1,4,9,16]);
```

$$2\,y-1,\ y^2$$

also $w = (2\,y-1)\,x + y^2$.

Kostenanalyse

Wir wollen annehmen, dass die Arithmetikkosten in K ebenfalls konstant sind. Während eines Durchlaufs der Hauptschleife fallen dann folgende Kosten an:

- Auswahl von $a \in K^{n-1}$ und Berechnung von S_a, so dass S (weiterhin) maximalen Rang hat: $O(D')$
- Berechnung der Evaluationen $\phi_a(f), \phi_a(g)$ als Skalarprodukte der Koeffizienten mit S_a: $O(d_n\,D')$.
- Berechnung von $\gcd(\phi_a(f), \phi_a(g))$: $O(d_n^2)$.

Vernachlässigen wir wieder den Aufwand, der für schlechte Reduktionen entsteht, so wird diese Schleife $O(D')$ mal durchlaufen. Im abschließenden Liften muss ein lineares Gleichungssystem mit der D'-reihigen Koeffizientenmatrix S und d_n verschiedenen rechten Seiten gelöst werden. Dies ist mit einem simultanen klassischen Gauß-Verfahren in $O(D'^2(D'+d_n))$ Arithmetik-Schritten möglich.

Die Kosten bis hierher sind also von der Größenordnung $O(D'^3 + d_n\,D'^2 + d_n^2\,D')$. Der Test zur Verifikation der Teilbarkeit verursacht Kosten der Größenordnung $O(d_n^2\,D'^2) = O(D^2)$ (klassische Multiplikaton). Für $D' \gg d_n$ dominieren in diesem Ansatz die Kosten des Gauß-verfahrens, was durch die simultane Evaluation in allen $n-1$ Variablen $x_1, \ldots, x_{n-1}$ verursacht wird. Im nächsten Paragrafen wird eine bivariate Version mit besserem Laufzeitverhalten beschrieben.

fastModularEvalGCD

Wir reduzieren dazu die Variablenzahl durch Evaluierung nur in der Variablen $y = x_{n-1}$ und den Ansatz $R[x] \to R'[x]$ mit $R' = k[x_1, \ldots, x_{n-2}]$. Der Evaluationsmorphismus ist $\phi_a : R = R'[y] \to R'$ mit $f(y) \mapsto f(a)$ und $a \in k$ (oder aus einer Erweiterung).

Ist $u \in R'[y]$ ein Polynom vom Grad $\deg(u) = d$, dessen Werte $u(c_i) = u_i \in R'$ an $d+1$ Stellen $c_i \in k$, $i = 0, \ldots, n$ bekannt sind, so kann u – ähnlich wie beim Newtonverfahren zum Chinesischen Restesatz – iterativ wie folgt gefunden werden:

Ist $h \in R'[y]$ ein Polynom vom Grad $< l$ mit $h(c_i) = u(c_i)$ für $i < l$ und $p = \prod_{i<l}(y - c_i) \in k[y] \subset R'[y]$, so ist

$$h'(y) = h(y) + \frac{u_l - h(c_l)}{p(c_l)} \cdot p(y)$$

ein Polynom vom Grad $< l+1$ mit $h(c_i) = u(c_i)$ für $i < l+1$. In der Tat, wegen $p(c_i) = 0$ ist $h'(c_i) = h(c_i) = u_i$ für $i < l$. Für $i = l$ ergibt sich $h'(c_l) = u_l$ unmittelbar. $h'(y) \in R'[y]$ ergibt sich daraus, dass $p(y) \in k[x]$ und damit $p(c_l) \in k$ in R' invertierbar ist.

Der folgende Interpolations-Algorithmus bestimmt ein solches u:

```
Interpolate:=proc(ev,v) local h,p,i;
begin
  h:=0; p:=1;
  for i from 1 to nops(ev) do
    h:=h+(v[i]-subs(h,y=ev[i]))/subs(p,y=ev[i])*p;
    p:=p*(y-ev[i]);
  end;
  expand(h);
end;
```

Kosten: Im Schritt l ist $p(y) = \sum_{i<l} p_i\, y^i \in k[y]$ mit $r_l = (u_l - h(c_l))\ p(c_l)^{-1} \in R'$ zu multiplizieren und zu $h(y)$ hinzuzufügen. Dies entspricht l skalaren Multiplikationen $p_i \cdot r_l,\ i < l$ und ebensovielen Additionen in R'. Die Kosten eines Iterationsschritts sind also von der Größenordnung $O(l\, D'')$, wobei D'' eine Schranke für die Bitlänge der Elemente $u_i \in R'$ ist. Damit entstehen für die vollständige Interpolation Gesamtkosten der Größenordnung $O(d^2\, D'')$.

Sind $f, g \in R'[y = x_{n-1}][x = x_n]$ zwei (bzgl. x) primitive Polynome, deren Gradvektor durch die Schranke $(d_1, \ldots, d_n)$ beschränkt ist, so berechnet der folgende Algorithmus den zugehörigen gcd.

fastModularEvalGCD(f,g)

Input: $f, g \in R[x]$ primitive Polynome
Output: $h = \gcd_{R[x]}(f, g)$

β:=gcd$(lc_x(f), lc_x(g)) \in R = R'[y]$
M:={} (* Menge der "nützlichen" Evaluationspunkte *)
d:=deg$(f) + \deg(g) + 1$
(* erwarteter Grad des gcd; hier garantiert zu groß *)

repeat
  Wähle ein $a \in k$ mit $\phi_a\,(lc(f) \cdot lc(g)) \neq 0$.
  Berechne $w_a = \gcd(\phi_a(f), \phi_a(g))$ in $R'[x]$ mit $lc(w_a) = \phi_a(\beta)$ rekursiv.
  if $\deg(w_a) > d$ then continue (* a ist schlechte Reduktion *)
  if $(\deg(w_a) < d)$ then { d:=deg(w_a), M:={} }
        (* alle bisherigen a waren schlechte Reduktionen *)
  if $d = 0$ then return 1 (* Polynome sind teilerfremd *)
  M:=M $\cup \{(a, w_a)\}$
until $|M| = d_{n-1}$.

`Lifte die` $(a, w_a) \in M$ `koeffizientenweise zu` $w \in R[x]$`.`

`If` $w \,|\, \beta\, f,\ w \,|\, \beta\, g$ `then return` $\mathrm{pp}_x(w)$ `else return FAILED.`

Ist wieder $D' = d_1 \cdot \ldots \cdot d_{n-1}$ und $D = D'\, d_n$, so ergeben sich die Kosten $C^{(n)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_n)$ dieses Algorithmus – wieder ohne die Berücksichtigung schlechter Reduktionen – aus

- Berechnungen der Reduktionen $\phi_a(f)$, $\phi_a(g)$ (Kosten: $d_n\, O(D') = O(D)$) sowie rekursive Berechnung von w_a (Kosten $C^{(n-1)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_{n-2}, d_n)$) für $|M| = d_{n-1}$ verschiedene $a \in k$.
- Kosten der Berechnung der maximal d_n Koeffizienten von $w = \sum_{l<d_n} c_l(y)\, x^l$.

  Die Werte der Koeffizienten $c_l(y)$ an den Evaluationspunkten a sind durch die Gradschranke $2\,(d_1, \ldots, d_{n-2})$ von $\beta\, f$ beschränkte Polynome $c_{la} = c_l(a) \in R'$. Der Grad der zu rekonstruierenden Koeffizienten $c_l(y) \in R = R'[y]$ von w ist durch $2\, d_{n-1}$ beschränkt, so dass die Kosten von `Interpolate` für jeden dieser d_n Koeffizienten von der Größenordnung $O\left(2^n\, (d_1 \cdot \ldots \cdot d_{n-2})\, d_{n-1}^2\right)$ sind.

Damit ergibt sich für die Gesamtkomplexität die rekursive Formel

$$C^{(n)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_n) = d_{n-1} \cdot C^{(n-1)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_{n-2}, d_n) + O\,(d_{n-1} \cdot D)$$

Auflösung dieser Rekursionsbeziehung führt auf die Abschätzung

$$C^{(n)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_n) = O\,(D \cdot (d_1 + \cdots + d_{n-1}))\ .$$

Damit dominieren auch in diesem Fall die Kosten $O(D^2)$ der abschließenden Probedivision die Kosten des gesamten Verfahrens:

Satz 31 *k sei ein Körper mit Einheitskostenarithmetik, $R' = k[x_1, \ldots, x_{n-2}]$, $R = R'[x_{n-1}]$ und $f, g \in R[x]$ primitive Polynome, deren Grade durch den Gradvektor $(d_1, \ldots, d_n)$ beschränkt sind.*

Die durchschnittlichen (ohne Berücksichtigung schlechter Reduktionen) Kosten des Algorithmus `fastModularEvalGCD(f,g)` *sind – bei klassischer Multiplikation – von der Größenordnung $O\left((d_1 \cdot \ldots \cdot d_n)^2\right)$.*

Ist k genügend groß, so ergibt die zufällige Wahl von $a \in k$ fast immer eine gute Reduktion. Für die Reduktion $\mathbb{Z}[x_1, \ldots, x_n] \to \mathbb{Z}_p[x_1, \ldots, x_n]$ kann das im praktisch relevanten Rechnungen durch die Wahl von p in der Größe eines Computerworts erreicht werden. Ist k bereits gegeben, etwa für eine gcd-Berechnung in $k[x_1, \ldots, x_n]$, so muss oft in einer ausreichend großen Erweiterung K/k gerechnet werden, womit die hier als $O(1)$ angesetzten Arithmetikkosten in k entsprechend angepasst werden müssen.

Eine Abschätzung der Komplexität im schlechtesten Fall hängt von einem genaueren Überblick über die Zahl der schlechten Reduktionen ab. Für

$$f, g \in \mathbb{Z}[x_1, \ldots, x_n],\ d = \max(d_1, \ldots, d_n) \text{ sowie } L(b) = b \cdot d_1 \cdot \ldots \cdot d_{n-1}$$

nennt Loos [2] folgende Schranken für Verfahren, die auf klassischer Multiplikation beruhen:

- $d\,L(b)$ ist eine obere Schranke für die Anzahl der schlechten Reduktionen
- Der modulare gcd-Algorithmus hat eine Laufzeit der Größenordnung $O(d^{2n+1}\,L(b)^2)$.
- SPRS hat eine Laufzeit der Größenordnung $O(d^{2n+2}\,L(b)^2)$.
- (Collins 1971) gibt einen modularen Resultantenalgorithmus an mit Laufzeit $O(d^{2n+1}L(b) + d^{2n}L(b)^2)$.

4 Polynom-Faktorisierung

Dieses Kapitel hält sich weitgehend an [5].

4.1 Allgemeines

Bereits früher im Kurs wurden die folgenden Begriffe eingeführt:

Definition 8 Sei R ein Integritätsbereich.
Die Zerlegung

$$a = \varepsilon\, p_1^{a_1} \cdot \ldots \cdot p_r^{a_r}$$

von $0 \neq a \in R$ mit $0 < a_i \in \mathbb{N}$, $\varepsilon \sim 1$ sowie Primelementen $p_i \in R$ bezeichnet man als als *Primfaktorzerlegung.*
Die Zerlegung ist *eindeutig*, wenn für jede andere Zerlegung

$$a = \varepsilon' q_1^{b_1} \cdot \ldots \cdot q_s^{b_s}$$

gilt: $r = s$ und es existiert eine Permutation $\pi \in S_r$ mit $a_i = b_{\pi(i)}$ und $p_i \sim q_{\pi(i)}$.
R heißt *faktoriell*, *UFD-Bereich* oder *ZPE-Ring*, wenn jedes Element $0 \neq a \in R$ eine eindeutige Zerlegung in Primelemente besitzt.

ZPE-Ring ist eine Abkürzung für „Ring mit eindeutiger **Z**erlegung in **P**rim-**E**lemente“ oder (umgekehrt gelesen) „Ring mit **E**indeutiger **P**rimfaktor-**Z**erlegung“, was dem englischen Unique Factorization Domain entspricht.

Ist R ein ZPE-Ring, so auch $R[x]$, was wir als Folgerung des Gauß-Lemmas bereits früher bewiesen haben. Insbesondere folgt aus dieser Implikation, dass Polynomringe $k[x_1, \ldots, x_n]$ über einem Körper k ZPE-Ringe sind, da Körper trivialerweise ZPE-Ringe sind.

Generell besteht ein enger Zusammenhang zwischen der Faktorisierung eines Polynoms $f \in R[x]$, den Faktorisierungen von $\mathrm{cont}(f) \in R$ und von $\mathrm{pp}(f) \in K[x]$ mit $K = Q(R)$:

- Ist $p \in R[x]$ primitiv (also $p = \mathrm{pp}(p)$) und prim in $K[x]$, so ist p auch prim in $R[x]$:
  In der Tat, ist $p = a \cdot b$ eine Zerlegung in $R[x]$, so gilt $p \,|\, a$ oder $p \,|\, b$ in $K[x]$, nach der Folgerung aus dem Gauß-Lemma $p = \mathrm{pp}(p) \,|\, \mathrm{pp}(a) \,|\, a$ oder $p = \mathrm{pp}(p) \,|\, \mathrm{pp}(b) \,|\, b$ in $R[x]$.
- Ist $\mathrm{pp}(f) = \varepsilon_1 \prod_i f_i^{a_i}$ eine derart skalierte Faktorisierung über $K[x]$, dass alle $f_i \in R[x]$ primitiv sind und $\mathrm{cont}(f) = \varepsilon_2 \prod_j r_j^{b_j}$ die Faktorisierung in R, so ist das Produkt der beiden eine Faktorisierung von f.

Wir können unser Studium der Faktorisierungsverfahren also wie im Fall der gcd-Berechnung auf die Faktorisierung in univariaten Polynomringen $K[x]$ über einem Körper K beschränken.

Einige Identitäten in $\mathbb{Z}_p[x]$

Im Polynomring $\mathbb{Z}_p[x]$ über dem Körper $\mathbb{Z}_p$ (p also prim) gelten eine Reihe interessanter Beziehungen:

(a) Nach dem kleinen Satz von Fermat gilt $a^p = a$ für alle $a \in \mathbb{Z}_p$.

(b) Damit ist jedes $a \in \mathbb{Z}_p$ Nullstelle des Polynoms $x^p - x$, dessen Faktorisierung also gerade durch

$$x^p - x = \prod_{a \in \mathbb{Z}_p} (x - a)$$

gegeben.

(c) Für Polynome $A, B \in \mathbb{Z}_p[x]$ gilt $(A + B)^p = A^p + B^p$.
Dies folgt sofort aus dem Binomischen Satz und $\binom{p}{k} \equiv 0 \pmod p$ für $1 \le k \le p-1$.

(d) Für $f(x) = \sum_i c_i x^i \in \mathbb{Z}_p[x]$ gilt $f' = 0$ genau dann, wenn $c_i = 0$ für alle $i \not\equiv 0 \pmod p$.
f hat dann die Gestalt $f(x) = \sum_j a_j x^{pj} = g(x^p)$ mit $a_j = c_{pj}$ und $g(x) = \sum_j a_j x^j$.
Wegen (a) gilt $a_j^p = a_j$ und mit (c) auch

$$f' = 0 \quad \Leftrightarrow \quad f(x) = g(x^p) = g(x)^p.$$

4.2 Quadratfreie Faktorisierung

Sei R ein ZPE-Ring.

Definition 9 Ein Polynom $f \in R$ heißt *quadratfrei*, wenn jeder Faktor in der Primzerlegung von f mit der Multiplizität 1 vorkommt.

Lemma 6 *Sei K ein Körper mit $char(K) = 0$ oder $K = \mathbb{Z}_p$.*
$f \in K[x]$ ist genau dann quadratfrei, wenn $\gcd(f, f') \sim 1$ gilt.

Beweis: Enthält $f = b^2 c$ einen quadratischen Faktor, so gilt $b \,|\, f' = b\,(2\,b'\,c + b\,c')$.
Ist umgekehrt $d \,|\, \gcd(f, f')$ ein gemeinsamer Primteiler und $f = d\,c$, so gilt $d \,|\, f' = d'\,c + d\,c'$ und somit $d \,|\, d'\,c$. Als Primteiler muss d einen der beiden Faktoren teilen. Für $d \,|\, c$ ist f nicht quadratfrei. $d \,|\, d'$ geht aus Gradgründen nicht, wenn $d' \neq 0$ ist.
$d' = 0$ ist nur im Fall $K = \mathbb{Z}_p$ möglich. Dann ist aber $d(x) = g(x)^p$ und f ebenfalls nicht quadratfrei. □
Der Satz kann in $char(K) = 0$ auf Polynome in mehreren Variablen verallgemeinert werden: $f \in K[x_1, \ldots, x_n]$ ist genau dann quadratfrei, wenn $\gcd(f, \partial_1 f, \ldots \partial_n f) \sim 1$.

Definition 10 Die (eindeutig bestimmte) Zerlegung $f = \prod_i b_i^i$ mit quadratfreien, paarweise teilerfremden b_i bezeichnet man als *quadratfreie Faktorisierung* von f.

b_i ist dabei das Produkt der Primelemente, die in einer Primfaktorzerlegung von $f \in K[x]$ genau mit Vielfachheit i vorkommen. Der folgende Algorithmus berechnet eine solche Zerlegung im Fall $char(K) = 0$.

sqrfreeFactorization(f)

Input: $f \in K[x]$, $char(K) = 0$.
Output: Quadratfreie b_i mit $f = \prod_i b_i^i$

```
c=[]
repeat
  g=gcd(f,f'); c=append(c,f/g); f=g
until deg(f) = 0
return [(c[i]/c[i+1]) for i in 1..length(c)-1]
```

Zu Beginn der k-ten Iteration ist $f = \prod_{i \geq k} b_i^{i-k+1}$. Berechnung von f' liefert

$$f' = \sum_{j \geq k} \prod_{i \geq k, i \neq j} b_i^{i-k+1} \cdot (j-k+1)\, b_j^{j-k}$$

Sei p ein gemeinsamer Primfaktor von f unf f'. Dann gilt $p \,|\, b_i$ für ein i. p kommt damit in allen Summanden $j \neq i$ der Zerlegung von f' in der genauen Potenz $i-k+1$ und im Summanden $j = i$ in der genauen Potenz $i-k$ vor, insgesamt also in der genauen Potenz $i-k$. Damit hat $g = \gcd(f, f')$ die Faktorzerlegung $g = \prod_{i>k} b_i^{i-k}$, was die Korrektheit des angegebenen Verfahrens beweist.

Quadratfreie Zerlegung über $\mathbb{Z}_p$. Über $\mathbb{Z}_p$ muss kann $j-k+1 \equiv 0 \pmod p$ sein, weshalb ein größerer Aufwand getrieben werden muss.

Ist $g = \gcd(f, f') \sim 1$, so ist f nach dem Lemma quadratfrei und mit $f = b_1$ die quadratfreie Zerlegung gefunden.

Ist g ein echter Teiler von f, so können quadratfreie Zerlegungen $\prod b_i^i$ von g und $\prod c_i^i$ von f/g berechnet werden. Deren Faktoren müssen in ihrer Gesamtheit allerdings nicht teilerfremd sein. $d_{ij} = \gcd(b_i, c_j)$, $i, j > 0$, und $d_{i0} = b_i / \prod_{j>0} d_{ij}$ sowie $d_{0j} = c_j / \prod_{i>0} d_{ij}$ ist eine Zerlegung in paarweise teilerfremde quadratfreie Faktoren, aus denen sich quadratfreie Zerlegungen von g und f/g und damit auch von f zusammenstellen lassen.

Ist $f = g$, so muss $f' = 0$ gewesen sein und es gilt $f(x) = h(x^p) = h(x)^p$. Aus einer verallgemeinerten quadratfreien Zerlegung von h kann dann eine von f berechnet werden.

Bei der Suche nach Algorithmen zu Faktorzerlegung in $\mathbb{Z}[x]$ oder $\mathbb{Z}_p[x]$ können wir also im Weiteren voraussetzen, dass f bereits in ein Produkt quadratfreier Faktoren zerlegt ist, und uns auf die Frage der Bestimmung der Faktorisierung quadratfreier Polynome bechränken.

4.3 Faktorisierung in $\mathbb{Z}_p[x]$

$a(x) \in \mathbb{Z}_p[x]$ sei ein quadratfreies Polynom vom Grad $d = \deg(a)$, dessen Faktorisierung $a(x) = a_1(x) \cdot \ldots \cdot a_r(x)$ in verschiedene (nicht bekannte) Primpolynome $a_i(x)$ vom (ebenfalls nicht bekannten) Grad $d_i = \deg(a_i)$ in $\mathbb{Z}_p[x]$ zu berechnen ist. Es gilt $d = d_1 + \cdots + d_r$.

Sei $A = \mathbb{Z}_p[x]/(a)$, $A_i = \mathbb{Z}_p[x]/(a_i)$ und $\pi : \mathbb{Z}_p[x] \to A$ die natürliche Projektion. Es gilt die folgende Verallgemeinerung des Chinesischen Restklassensatzes: Der natürliche Ringhomomorphismus

$$\phi_0 : \mathbb{Z}_p[x] \to A_1 \times A_2 \times \cdots \times A_r$$

hat den Kern $ker(\phi) = (a)$, so dass

$$\phi : A \to A_1 \times A_2 \times \cdots \times A_r$$

ein injektiver Ringhomomorphismus ist. Auf beiden Seiten stehen endlich dimensionale $\mathbb{Z}_p$-Vektorräume der gleichen Dimension $\dim_k(A) = d$, $\dim_k(A_1 \times A_2 \times \cdots \times A_r) = d_1 + \cdots + d_r$, so dass ϕ sogar ein Isomorphismus ist.

Zu jedem $g \in A$ gibt es ein eindeutig bestimmtes Polynom $f \in \mathbb{Z}_p[x]$ vom Grad $\deg(f) < d$, für welches $\pi(f) = g$ gilt. Mit $(\mathbb{Z}_p[x])_d$ bezeichnen wir den d-dimensionalen $\mathbb{Z}_p$-Vektorraum dieser Polynome. Dann können wir die eben ausgeführten Überlegungen wie folgt fassen:

Folgerung 6 *Zu vorgegebenen Elementen $s_i(x) \in \mathbb{Z}_p[x]$, $i = 1, \ldots, r$, gibt es stets ein eindeutig bestimmtes $s(x) \in (\mathbb{Z}_p[x])_d$ mit*

$$s(x) \equiv s_i(x) \pmod{a_i(x)} \qquad \textit{für alle } i = 1, \ldots, r. \tag{*}$$

Wir konzentrieren uns auf Lösungen $s(x) \in (\mathbb{Z}_p[x])_d$ von (*) mit konstanten Polynomen $s_i(x) = s_i \in \mathbb{Z}_p$. Aus einer solchen Lösung mit $s_i \neq s_j$ ergibt sich mit $b = \gcd(a(x), s(x) - s_i)$ eine Faktorisierung $a = b \cdot (a/b)$ in nicht triviale Faktoren, da wegen $s(x) - s_i \equiv s_j - s_i \pmod{a_j(x)}$ a_i ein gemeinsamer Teiler von b, a_j, $j \neq i$, dagegen kein Teiler von b und damit als ein Teiler von a/b ist.

Natürlich sind die $a_1(x), \ldots, a_r(x)$ unbekannt und wir können die beschriebenen Rechnungen praktisch nicht ausführen. Wir wollen deshalb eine invariante Beschreibung von

$$S = \left\{ s(x) \in (\mathbb{Z}_p[x])_d \,:\, s(x) \text{ ist Lösung von (*) für rechte Seiten } s_1, \ldots, s_r \in \mathbb{Z}_p \right\}$$

herleiten. Da sich zu jeder Wahl der rechten Seiten in (*) genau eine Lösung ergibt, enthält S genau p^r Elemente und hat sogar die Struktur eines (dann r-dimensionalen) $\mathbb{Z}_p$-Vektorraums: Ist $s(x)$ die Lösung zu $(s_1, \ldots, s_r)$, so ist offensichtlich $\alpha\, s(x)$ die Lösung zu $(\alpha\, s_1, \ldots, \alpha\, s_r)$.

Polynome aus S lassen sich wie folgt charakterisieren:

Ist $s(x) \in S$ und $s(x) \equiv s_i \pmod{a_i(x)}$, so gilt $s(x)^p \equiv s_i^p = s_i \pmod{a_i(x)}$, damit $s(x)^p \equiv s(x) \pmod{a_i(x)}$ für $i = 1, \ldots, r$ und schließlich

$$s(x)^p \equiv s(x) \pmod{a(x)}.$$

Gilt umgekehrt $s(x)^p \equiv s(x) \pmod{a(x)}$, also (wegen 4.1.(b))

$$a(x) \,|\, (s(x)^p - s(x)) = \prod_{c \in \mathbb{Z}_p} (s(x) - c),$$

so muss jeder der Primfaktoren a_i einen der Faktoren $(s(x) - c)$ teilen, woraus $s(x) \in S$ folgt. Wir haben damit folgendes Lemma bewiesen:

Lemma 7 *Es gilt $S = \left\{ s(x) \in (\mathbb{Z}_p[x])_d \,:\, s(x)^p \equiv s(x^p) \equiv s(x) \pmod{a(x)} \right\}$.*

Dies ist zugleich eine Charakterisierung von S, welche ohne Kenntnis der Zerlegung $a(x) = a_1(x) \cdot \ldots \cdot a_r(x)$ auskommt. Die Menge S kann bestimmt werden, indem $s(x) = c_0 + c_1 x + \cdots + c_{d-1}x^{d-1}$ mit unbestimmten Koeffizienten angesetzt wird und aus der Beziehung

$$s(x^p) - s(x) \equiv c_1 \cdot NF(x^p - x) + \cdots + c_{d-1} \cdot NF(x^{p(d-1)} - x^{d-1}) = 0 \pmod{a(x)}$$

durch Koeffizientenvergleich ein homogenes lineares Gleichungssystem mit d-reihiger quadratischer Koeffizientenmatrix Q extrahiert wird. Dabei ist $NF(f(x))$ die reduzierte Normalform von $f(x) \in \mathbb{Z}_p[x]$ modulo $a(x)$ und $NF(x^0 - 1) = 0$ berücksichtigt. In der Spalte i $(i = 0, \ldots, d-1)$ von Q stehen also die Koeffizienten von $NF(x^{p\,i} - x^i) \in (\mathbb{Z}_p[x])_d$.

Der Rang der Matrix Q ist gerade $d-r$, eine Basis von S besteht aus r Polynomen $s^{(1)}(x) = 1$, $s^{(2)}(x), \ldots, s^{(r)}(x)$.

Beispiel:

```
p:=3;
R:=Dom::UnivariatePolynomial(x,Dom::IntegerMod(p));
f:=R(x^5+x^3+2*x^2+x+2);
```

Die Berechnung von `gcd(f,f')` zeigt zunächst, dass für das angegebene Polynom die Voraussetzung der Quadratfreiheit erfüllt ist. Als Ergebnis haben wir laut MuPAD

```
factor(f);
```

$$\left(x^2 + x - 1\right)\,\left(x^3 - x^2 + 1\right)$$

zu erwarten.

Stellen wir dazu zunächst obiges Gleichungssystem auf und finden eine Basis des Nullraums:

```
Q:=linalg::transpose(Dom::Matrix(Dom::IntegerMod(p))(
   [ [coeff(R::rem(x^(i*p)-x^i,f),j) $ j=0..degree(f)-1]
          $ i=0..degree(f)-1]));
linalg::nullspace(Q);
```

Wir lesen $r = 2$ ab und $s^{(2)}(x) = x^3 + 2x^2$.

$f(x)$ muss also in 2 Faktoren zerfallen. Diese finden wir, indem wir $s^{(2)}(x)$ zum Spalten von $f(x)$ verwenden:

```
s:=R(x^3+2*x^2);
[gcd(f,s-c) $ c=0..2];
```

$$\left[1,\ x^2 + x - 1,\ x^3 - x^2 + 1\right]$$

Dies demonstriert zugleich auch das allgemeine Vorgehen in diesem nach Berlekamp benannten Algorithmus.

BerlekampFactorization(a)

Input: $a(x) \in \mathbb{Z}_p[x]$ quadratfrei.

Output: Faktorzerlegung $a(x) = a_1(x) \cdot \ldots \cdot a_r(x) \in \mathbb{Z}_p[x]$

`Bilde die Matrix` Q `mit den Koeffizienten von` $x^{pi} - x^i \pmod{a(x)}$

```
    als Spalten.
Finde eine Basis C = {c^(1) = (1 0 ...0)^T, ..., c^(r)} des Nullraums von Q.
if r=1 then return {a(x)}.
Bilde aus C die Menge B:={s^(2)(x), ... s^(r)(x)} der Probepolynome.

F:={a(x)}
for s in B do /* (1) */
  for f in F do /* (2) */
    G:={}; F:=F \ {f(x)}
    for c from 0 to p - 1 do /* (3) */
      Bestimme g(x) = gcd(f(x), s(x) - c).
      if 0 < deg(g(x)) < deg(f(x)) then
        G:=append(G,g); f=f/g;
        if f ~ 1 then break; (* f komplett zerlegt *)
      end_for;
      F=join(F,G);
      if |F| = r then return F;
        (* Zerlegung in r Faktoren gefunden *)
    end_for;
  end_for;
```

Zu jedem Zeitpunkt (2) ist $\prod_{f(x)\in F} f(x) = a(x)$. Zum an der Stelle (2) ausgewählten Polynom f werden nacheinander Faktoren $g \,|\, f$ gefunden, für die zugleich $g \,|\, s - c$ gilt. Da $s - c$ für verschiedene $c \in \mathbb{Z}_p$ zueinander teilerfremd sind, sind es die g untereinander auch. Da f ein Teiler von a und a ein Teiler von $\prod_{c\in\mathbb{Z}_p} (s(x) - c)$ ist, wird f dabei komplett in Faktoren $f = \prod g_\alpha$ aufgespalten, wobei g_α paarweise teilerfremd sind und $g_\alpha \,|\, s(x) - c_\alpha$ für verschiedene $c_\alpha \in \mathbb{Z}_p$ gilt.

Je zwei Faktoren a_i, a_j werden durch eine Wahl von $s(x) \in B$ und $c \in \mathbb{Z}_p$ getrennt. In der Tat, es gibt ein $s' \in S$ mit $s' \equiv 0 \pmod{a_i}$ und $s' \equiv 1 \pmod{a_j}$. Dieses Element lässt sich als Kombination der Basiselemente $s \in B$ darstellen, so dass es ein $s \in B$ geben muss mit $s \equiv c_1 \pmod{a_i},\ s \equiv c_2 \pmod{a_j},\ 0 \le c_1, c_2 < p,\ c_1 \ne c_2$. Dann wird aber $a_i \cdot a_j$ von diesem s separiert. Dies zeigt, dass der Algorithmus terminiert.

Zur **Abschätzung der Laufzeit** ergibt sich folgende Rechnung:

- Zunächst sind die $NF(x^{pi} - x^i)$ zu berechnen (Kosten $O(p\,d + d^3)$, wenn zunächst $f \equiv x^p \pmod{a(x)}$ und dann $f^i \pmod{a(x)},\ i = 1, \ldots, d-1$, berechnet wird).
- Daraus ist die Matrix Q zu extrahieren und deren Nullraum zu bestimmen (Kosten $O(d^3)$).
- Schließlich sind $n\,r$ gcd's in $\mathbb{Z}_p[x]$ zu bestimmen (Kosten jeweils $O(d^2)$).

Damit ergibt sich eine Gesamtlaufzeit von $O\left(d^3 + p\,r\,d^2\right)$.

Wegen $r \le d$ liefern für große p fast alle $c \in \mathbb{Z}_p$ ein triviales $g(x)$. Nach dem Resultantenkriterium ist $\gcd(f(x), s(x) - c)$ genau für Nullstellen c des Polynoms $p(y) = res_x(f(x), s(x) - y) \in \mathbb{Z}_p[y]$ nicht trivial, so dass c noch gezielter gesucht werden kann.

Beispiel:

```
p:=37;
R:=Dom::UnivariatePolynomial(x,Dom::IntegerMod(p));
f:=R(x^5+3*x^3+x^2+2*x+2);
```

f ist quadratfrei wie die Berechnung von $\gcd(f, f')$ zeigt.

```
Q:=linalg::transpose(Dom::Matrix(Dom::IntegerMod(p))(
   [[coeff(R::rem(x^(i*p)-x^i,f),j)$ j=0..degree(f)-1]$ i=0..degree(f)-1]));
linalg::nullspace(Q);
```

Wir lesen $r = 3$ ab und erhalten als Menge der Probepolynome

$$s^{(2)}(x) = 2\,x + 3\,x^2 + x^3,\ s^{(3)}(x) = x^4 - 2\,x^2$$

Wir bestimmen die Resultante als Polynom über $\mathbb{Z}_p$ und deren Nullstellen

```
s2:=R(2*x+3*x^2+x^3);
res:=polylib::resultant(expr(f),expr(s2)-y,x);
solve(res,[y],Domain=Dom::IntegerMod(p));
```

und erhalten als Kandidaten $c \in \{8, 12, 31\}$. Diese drei Werte spalten f in die Faktoren

```
map([8,12,31], c->gcd(f,s2-c));
```

$$\left[x^2 - 12\,x - 3,\ x + 12,\ x^2 + 2\right]$$

was genau die Faktorzerlegung von f ergibt.

4.4 Faktorisierung in $\mathbb{Z}[x]$ – Der Berlekamp-Zassenhaus-Algorithmus

Ähnlich wie im Fall der gcd-Berechnung wird die Faktorisierung in $\mathbb{Z}[x]$ auf die Faktorisierung in $\mathbb{Z}_p[x]$ für eine geeignete Primzahl zurückgeführt. Da nicht klar ist, wie die Faktoren in den Faktorzerlegungen für unterschiedliche p einander zugeordnet sind, wird allerdings ein anderer Liftungsansatz verwendet, der zunächst Faktorisierungen $\pmod p$ zu Faktorisierungen $\pmod{p^k}$ für immer größere $k \in \mathbb{N}$ liftet.

Dieser Zugang der *p-adischen Approximation* der Faktorzerlegung in $\mathbb{Z}[x]$ geht davon aus, dass in einem gewissen Sinne Korrekturterme der Form $c\,p^k$ für Rechnungen $\pmod p$ als „klein“ der Ordnung k zu betrachten sind. Dieser Ansatz lässt sich formal weiter bis zu einer p-adischen Analysis treiben.

Zur Bestimmung der Faktorzerlegung eines Polynoms in $\mathbb{Z}[x]$ können wir uns zunächst wieder auf quadratfreie Polynome beschränken.

Ist $a(x) \in \mathbb{Z}[x]$ ein solches quadratfreies Polynom und p eine Primzahl, bzgl. welcher die Quadratfreiheit erhalten bleibt (also $\operatorname{res}(a, a') \not\equiv 0 \pmod p$ nach dem Resultantensatz), so können wir zunächst eine Faktorisierung

$$a(x) \equiv a_1(x) \cdot \ldots \cdot a_r(x) \pmod p$$

$\pmod p$ bestimmen. Ist $r = 1$, so ist $a(x) \in \mathbb{Z}[x]$ ebenfalls irreduzibel, denn jede Faktorzerlegung in $\mathbb{Z}[x]$ induziert eine solche in $\mathbb{Z}_p[x]$. Anderenfalls finden sich eine Aufspaltung $a(x) \equiv f(x)\,g(x) \pmod p$ in über $\mathbb{Z}_p$ teilerfremde Faktoren. Eine solche Aufspaltung kann zu einer Aufspaltung $\pmod{p^k}$ für höhere k angehoben werden:

Lemma 8 (Hensel-Lemma) *Ist*

$$a(x) \equiv f(x) \cdot g(x) \pmod{p}$$

eine Zerlegung von $a(x)$ in zwei über $\mathbb{Z}_p$ teilerfremde Faktoren, so gibt es für jedes $k > 0$ Polynome $f^{(k)}(x), g^{(k)}(x) \in \mathbb{Z}[x]$, so dass

$$f^{(k+1)}(x) \equiv f^{(k)}(x) \pmod{p^k}, \quad g^{(k+1)}(x) \equiv g^{(k)}(x) \pmod{p^k},$$

gilt, also

$$f^{(k+1)}(x) = f^{(k)}(x) + df^{(k)}(x) \cdot p^k, \quad g^{(k+1)}(x) = g^{(k)}(x) + dg^{(k)}(x) \cdot p^k,$$

für Korrekturpolynome $df^{(k)}(x), dg^{(k)}(x)$, die in $\mathbb{Z}_p[x]$ berechnet und dann geliftet werden können, und

$$a(x) \equiv f^{(k)}(x) \cdot f^{(k)}(x) \pmod{p^k}$$

sowie $\deg(f) = \deg(f^{(k)}), deg(g) = \deg(g^{(k)})$ gilt.
Diese Polynome sind $\pmod{p^k}$ eindeutig bestimmt.

Beweis: Für $k = 1$ ist die Bedingung erfüllt. Wir suchen

$$f^{(k+1)}(x) = f^{(k)}(x) + U(x) \cdot p^k,\ g^{(k+1)}(x) = g^{(k)}(x) + V(x) \cdot p^k$$

mit zu bestimmenden Polynomen $U(x), V(x) \in \mathbb{Z}[x]$, so dass

$$a(x) \equiv f^{(k+1)}(x)\, g^{(k+1)}(x) \pmod{p^{k+1}}$$

gilt. Dies ist äquivalent zu

$$a(x) - f^{(k)}(x)\, g^{(k)}(x) \equiv \Big(f^{(k)}(x)\, V(x) + g^{(k)}(x)\, U(x)\Big)\, p^k \pmod{p^{k+1}}$$

bzw.

$$a^{(k)}(x) = \frac{a(x) - f^{(k)}(x)\, g^{(k)}(x)}{p^k} \equiv f(x)\, V(x) + g(x)\, U(x) \pmod{p}$$

wegen $f^{(k)}(x) \equiv f(x) \pmod{p}$, $g^{(k)}(x) \equiv g(x) \pmod{p}$ und $a(x) - f^{(k)}(x)\, g^{(k)}(x) \equiv 0 \pmod{p^k}$.
Da $f(x), g(x) \in \mathbb{Z}_p[x]$ teilerfremd sind, liefert die Bezout-Zerlegung

$$f(x)\, s(x) + g(x)\, t(x) \equiv 1 \pmod{p} \tag{B}$$

die entsprechenden Liftungen

$$df^{(k)} = a^{(k)}(x)\, t(x) \pmod{f(x)}, \quad dg^{(k)} = a^{(k)}(x)\, s(x) \pmod{g(x)}$$

wobei die Rechnungen in $\mathbb{Z}_p[x]$ ausgeführt werden und die letzte Reduktion erreicht, dass $\deg(f^{(k+1)}) = \deg(f),\ \deg(g^{(k+1)}) = \deg(g)$ gilt.

Kennt man $df^{(k)}$ und $dg^{(k)}$, so gilt

$$\begin{aligned} a^{(k+1)}(x) &= \frac{a - f^{(k+1)}\,g^{(k+1)}}{p^{k+1}} = \frac{a - \left(f^{(k)} + df^{(k)}\cdot p^k\right)\,\left(g^{(k)} + dg^{(k)}\cdot p^k\right)}{p^{k+1}} \\ &= \frac{\left(a^{(k)} - f^{(k)}\,dg^{(k)} - g^{(k)}\,df^{(k)}\right)\cdot p^k + O(p^{k+2})}{p^{k+1}} \\ &\equiv \frac{a^{(k)} - f^{(k)}\,dg^{(k)} - g^{(k)}\,df^{(k)}}{p} \pmod p \end{aligned}$$

und $a^{(k+1)} \in \mathbb{Z}_p[x]$ lässt sich unmittelbar berechnen. $\square$

Komplexität: Nach der Berechnung der Bezout-Zerlegung lassen sich nacheinander

$$a^{(k)} \quad\longrightarrow\quad df^{(k)},\ dg^{(k)} \quad\longrightarrow\quad a^{(k+1)}$$

durch einfache arithmetische Operationen mit Polynomen vom Grad $\deg(a) < d$ über (mod p^2) bestimmen. Die Kosten pro Iteration sind aolso von der Größenordnung $O(d^2)$, wenn man nur die Inkremente $df^{(k)},\ dg^{(k)}$ berechnet und auf die (mit Blick auf die Multiplikationen mit p^k teure) Berechnung der $f^{(k)},\ g^{(k)}$ verzichtet.

Beispiel: $a(x) = x^4 + x^2 + 1$. Wegen $\operatorname{res}(a, a') = 144$ können die Primzahlen $p \in \{2, 3\}$ nicht zum Faktorisieren verwendet werden. Wir setzen also $p = 5$.

```
p:=5; a:=x^4+x^2+1;
R:=Dom::UnivariatePolynomial(x,Dom::IntegerMod(p));
factor(R(a));
```

$$\left(x + x^2 + 1\right)\,\left(4\,x + x^2 + 1\right)$$

Wir setzen also $f = x^2{+}x{+}1,\ g = x^2{+}4x{+}1$ und bestimmen die zugehörigen Bezout-Faktoren $s = 2x + 3,\ t = 3x + 3$ in $\mathbb{Z}_5[x]$

```
f:=x^2+x+1; g:=x^2+4*x+1; s:=2*x+3; t:=3*x+3;
R(f*s+g*t);
```

Für die Korrekturterme erster Ordnung erhalten wir nacheinander

```
a1:=expand(a-f*g)/p;
u1:=expr(R::rem(R(a1*t),R(f))); f1:=f+p*u1;
v1:=expr(R::rem(R(a1*s),R(g))); g1:=g+p*v1;
```

$$a_1 = -x - x^2 - x^3,\ u_1 = 0,\ v_1 = 4x, g_1 = x^2 + 24x + 1$$

Wegen $a_2 = \dfrac{a - f_1\,g_1}{p^2} = a_1$ wiederholen sich die Rechnungen und wir erhalten

$$f^{(k)} = f, \qquad g^{(k)} = g + 4x\left(5 + 5^2 + \cdots + 5^k\right).$$

In diesem Beispiel ist die Antwort einfach zu finden, wenn wir auf das kleinste symmetrische Restesystem normieren. Wir starten dann von der Zerlegung

$$x^4 + x^2 + 1 \equiv (x + x^2 + 1)\,(-x + x^2 + 1) \pmod 5$$

die bereits eine Identität in $\mathbb{Z}[x]$ ist und damit die gesuchte Faktorzerlegung liefert.

Beispiel: $a = x^4 - 98\,x^2 + 1$. Die Berechnung der Resultante

```
polylib::resultant(a,diff(a,x),x);
```

zeigt, dass $p \notin \{2,3,5\}$ beachtet werden muss, damit sich die Voraussetzung der Quadratfreiheit auf $\mathbb{Z}_p[x]$ überträgt. Wir wählen $p = 7$.

```
p:=7; a:=x^4 - 98*x^2 + 1;
R:=Dom::UnivariatePolynomial(x,Dom::IntegerMod(p));
factor(R(a));
```

$$(x^2 + 3x + 1)\,(x^2 - 3x + 1)$$

Wir setzen also $f = x^2 + 3x + 1,\ g = x^2 - 3x + 1$ und bestimmen die zugehörigen Bezout-Faktoren $s = x - 3,\ t = -x - 3$ in $\mathbb{Z}_7[x]$. Für die Korrekturterme erster Ordnung erhalten wir nacheinander

```
f:=x^2+3*x+1; g:=x^2-3*x+1; s:=x-3; t:=-x-3;
a1:=expand(a-f*g)/p;
u1:=expr(R::rem(R(a1*t),R(f)));
v1:=expr(R::rem(R(a1*s),R(g)));
```

Zu den berechneten Korrekturtermen $u_1 = x \pmod 7,\ v_1 = 6x \pmod 7$ nehmen wir die Liftungen $u_1 = x,\ v_1 = -x \in \mathbb{Z}[x]$ und erhalten mit den Approximationen $f_1 = x^2 + 10x + 1,\ g_1 = x^2 - 10x + 1$ bereits die Faktorzerlegung von $a(x)$.

Dieses Verfahren funktioniert offensichtlich immer dann, wenn die Faktorzerlegung $a(x) \equiv f^{(1)}(x) \cdot g^{(1)}(x) \pmod p$ über $\mathbb{Z}_p[x]$ von einer Faktorzerlegung $a(x) = f(x) \cdot g(x)$ induziert wird. Wir müssen dazu das Liften nur bis zu einem k treiben, für welches $p^k > 2B$ gilt, wobei B die Mignotte-Schranke für die Koeffizientengröße von Faktoren von $a(x)$ ist.

Jede Faktorzerlegung $a(x) = f(x) \cdot g(x)$ in $\mathbb{Z}[x]$ induziert eine solche in $\mathbb{Z}_p[x]$. Die Umkehrung gilt leider nicht:

Beispiel: $x^4 + 1$ ist irreduzibel in $\mathbb{Z}[x]$, zerlegt sich aber in zwei quadratische Faktoren in $\mathbb{Z}_p[x]$ für jede Primzahl p.

```
f:=p->Dom::UnivariatePolynomial(x,Dom::IntegerMod(p))(x^4+1);
map([2,3,5,7,11,13,17],p->factor(f(p)));
```

$$(x+1)^4,\ (x^2+x+2)\,(x^2-x+2),\ (x^2+2)\,(x^2+3),\ (x^2+3\,x+1)\,(x^2-3\,x+1),$$
$$(x^2+3\,x+10)\,(x^2-3\,x+10),\ (x^2+5)\,(x^2+8),\ (x+2)\,(x+8)\,(x+9)\,(x+15)$$

Wir können die Faktorisierungsaufgaben über $\mathbb{Z}[x]$ deshalb darauf reduzieren, irreduzible Polynome zu erkennen und für reduzible Polynome $a(x)$ einen nichttrivialen Faktor $f(x)$ zu finden. Dann können wir denselben Algorithmus auf $f(x)$ und $g(x) = a(x)/f(x)$ rekursiv anwenden, bis eine Zerlegung in irreduzible Faktoren von $a(x) \in \mathbb{Z}[x]$ gefunden ist.

BerlekampZassenhausFactorization(a)

Input: $a(x) \in \mathbb{Z}[x]$ quadratfrei.

Output: `isIrreducible` oder $a(x) = f(x) \cdot g(x) \in \mathbb{Z}[x]$ mit $0 < \deg(f), \deg(g) < \deg(a)$

`Berechne die Mignotteschranke` B `für` $a(x)$`.`

```
Wähle Primzahl p mit res_x(a, a') ≢ 0 (mod p).
Bestimme die Faktorzerlegung F = {a_1, ..., a_r} von a(x) ∈ Z_p[x].
if r = 1 then return isIrreducible.

for each S ⊂ {1, ..., r} do
  Lifte ∏_{i∈S} a_i und ∏_{i∉S} a_i
      zu reduzierten (mod p) Polynomen f = f^(1)(x), g = g^(1)(x) in Z[x].
  Bestimme die Bezoutfaktoren s, t ∈ Z_p[x] von (f, g).
  for k = 2 while p^k < 2B do
    Bestimme die reduzierten (mod p^k) Liftungen f^(k)(x), g^(k)(x).
    if a(x) = f^(k)(x) g^(k)(x) then return f^(k)(x) g^(k)(x).
  /* an dieser Stelle ist klar, dass dieses S keine Liftung nach Z hat */
  continue

/* an dieser Stelle ist klar, dass kein S eine Liftung nach Z hat */
return isIrreducible.
```

Dieses Verfahren ist effizient, wenn r klein ist, da die Kosten für das Hensel-Lifting pro Paar (f, g) von der Ordnung $O(d^2 \log(B))$ für $\deg(a) < d$ sind. Für große r ergibt sich eine kombinatorische Explosion der Anzahl der zu untersuchenden Partitionen, die im schlechtesten Fall die Größenordnung 2^r hat. Dies gilt etwa für in $\mathbb{Z}[x]$ irreduzible Polynome, die sich in $\mathbb{Z}_p[x]$ in Linearfaktoren zerlegen lassen; dann ist $r = d$. Hier kann es hilfreich sein, die Gradmuster der Faktorzerlegungen von $a(x) \in \mathbb{Z}_q[x]$ für weitere Primzahlen $q \neq p$ zu bestimmen, um Partitionen S, die aus bereits Gradgründen nicht funktionieren können, von vornherein auszuschließen.

Eine konsequente Umsetzung dieser Idee führt zum LLL-Algorithmus von Lenstra, Lenstra und Lovácz, der in [3, Kap. 4.3] beschrieben ist.

Literatur

[1] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Acad. Publisher, 2 edition, 1992.

[2] R. Loos. Generalized polynomial remainder sequences. In B. Buchberger, G.E. Collins, and R. Loos, editors, *Computer Algebra – Symbolic and Algebraic Computation*, pages 115–137. Springer, Wien, 1982.

[3] M. Mignotte and D. Stefanescu. *Polynomials*. Springer, 1999.

[4] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge Univ. Press, 1999.

[5] F. Winkler. *Polynomial algorithms in computer algebra*. Texts and Monographs in Symbolic Computation. Springer, Wien, 1996.