Algorithmen für Polynome
Vorlesung Wintersemester 2004/05

Prof. H.-G. Gräbe, Institut für Informatik,
http://www.informatik.uni-leipzig.de/~graebe

9. Februar 2005

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 $R = \mathbb{R}$ 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.

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 symmetriches 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);
```

$$
\begin{aligned}
&{x_4}^3 + (x_3 + x_2 + x_1 + x_1)\,{x_4}^2 \\
&\quad + \left({x_3}^2 + (x_2 + x_1 + x_1)x_3 + {x_2}^2 + (x_1 + x_1)x_2 + {x_1}^2 + {x_1}^2\right) x_4 \\
&\quad + {x_3}^3 + (x_2 + x_1 + x_1){x_3}^2 + \left({x_2}^2 + (x_1 + x_1)x_2 + {x_1}^2 + {x_1}^2\right)x_3 \\
&\qquad + {x_2}^3 + (x_1 + x_1){x_2}^2 + \left({x_1}^2 + {x_1}^2\right)x_2 + {x_1}^3 + {x_1}^3
\end{aligned}
$$

Polynomgröße wird bestimmt durch 2 Parameter: Gradschranke $d_n(f) = \deg_{x_n}(f)$ 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

$$\mathbf{x}^a < \mathbf{x}^b \implies \mathbf{x}^c \cdot \mathbf{x}^a < \mathbf{x}^c \cdot \mathbf{x}^b$$

gilt.

Begriffe: Leitterm $lt(f) \in T$, Leitkoeffizient $lc(f) \in k$, Leitmonom $lm(f) = lc(f) \cdot lt(f)$ (Bezeichnung in der Literatur nicht einheitlich).

Beispiel: Volles symmetrisches Polynom vom Grad 5 in x_1, x_2, x_3, x_4 siehe oben.

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

Auch in diesem Fall können wir die Polynome betrachten, deren Terme druch eine Gradschranke $\mathbf{d} = (d_1,\ldots,d_n)$ und deren Koeffizentenbitgröße durch eine Schranke t begrenzt ist.

1.3 Komplexitätsbetrachtungen

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

Komplexität wollen wir deshalb in folgendem distributiv-rekursiven Ansatz betrachten.

Ausgangsparameter ist der Polynomring $R = A[x_1,\ldots,x_m]$, wobei die Komplexität eines Polynoms $f \in R$ durch ein Tupel $(t; d_1,\ldots,d_m)$ charakterisiert wird, in dem t für die Koeffizientengröße (als z.B. deren Bitlänge) steht und $d_i = d_i(f)$ gilt.

Wir erhalten:

Bitlänge $L(t; d_1,\ldots,d_m) = O(t\,d_1\,\ldots\,d_m)$

Addition zweier solcher Polynome

$$C^+_R(t; d_1,\ldots,d_m) \sim C^+_A(t)\cdot(d_1\cdot\ldots\cdot d_m)$$

Multiplikation zweier solcher Polynome (klassisch)

$$C^*_R(t; d_1,\ldots,d_m) \sim C^*_A(t)\cdot(d_1^2\cdot\ldots\cdot d_m^2)$$

Für Grundbereiche mit konstanten Kosten für die Arithmetik erhalten wir daraus

$$
\begin{aligned}
C^+_R(1; d_1,\ldots,d_m) &\sim d_1\cdot\ldots\cdot d_m\\
C^*_R(1; d_1,\ldots,d_m) &\sim (d_1\cdot\ldots\cdot d_m)^2,
\end{aligned}
$$

für den Grundbereich $\mathbb{Z}$ und Koeffizienten mit einer Bitlänge kleiner als t dagegen

$$
\begin{aligned}
C^+_R(t; d_1,\ldots,d_m) &\sim t\cdot d_1\cdot\ldots\cdot d_m\\
C^*_R(t; d_1,\ldots,d_m) &\sim t^2\cdot(d_1\cdot\ldots\cdot d_m)^2.
\end{aligned}
$$

Wir können auf dieser Basis die auch durch vielfältige Erfahrungen mit konkreten Rechnungen bestätigte Regel formulieren, dass

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

1.4 Schnelle Multiplikationsverfahren

Wir betrachten dabei den rekursiven Ansatz der Multiplikation von zwei Polynomen $f, g \in R = A[x]$ vom Grad $\deg(f), \deg(g) < d$, wobei A ein Integritätsbereich ist, der selbst wieder ein Polynomring sein kann.

Die Karatsuba-Multiplikation

Idee: Sind f, g Polynome mit der Gradschranke $d = 2l$, 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 $< l$ und erhalten

$$f \cdot g = (f_1 g_1) x^{2l} + (f_1 g_2 + f_2 g_1) x^l + (f_2 g_2)$$

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

$$(f_1 g_2 + f_2 g_1) = (f_1 + g_2)(f_1 + g_2) - f_1 g_1 - f_2 g_2.$$

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

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

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

wenn man nur die Multiplikationen berücksichtigt und

$$C_{Karatsuba}(2\,l) = 3\,C_{Karatsuba}(l) + 6\,l,$$

wenn auch die Additionen berücksichtigt werden. In beiden Fällen erhält man

$$C_{Karatsuba}(d) = O(d^\alpha) \text{ mit } \alpha = \frac{\log 3}{\log 2} \approx 1.58$$

Dieser Exponent liegt unterhalb des Exponenten 2 aus dem klassischen Verfahren. Allerdings wird er in der Praxis meist nicht angewendet, weil man es dort überwiegend mit dünnen Polynomen 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 $< d$, das mit $O(d \log(d))$ Operationen 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. 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 solche Wert kann also durch eine einzige Multiplikation berechnet werden, wenn die Funktionswerte $a(\lambda)$ und $b(\lambda)$ berechnet sind. Für $n+1$ verschiedene Elemente $\lambda_0, \ldots, \lambda_n \in k$ lassen sich dann $h_i = c(\lambda_i)$ mit $n+1$ nichtskalaren Operationen berechnen. Nun gilt aber

$$\begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_n \end{pmatrix} = \begin{pmatrix} 1 & \lambda_0 & \lambda_0^2 & \ldots & \lambda_0^n \\ 1 & \lambda_1 & \lambda_1^2 & \ldots & \lambda_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_n & \lambda_n^2 & \ldots & \lambda_n^n \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix},$$

wobei die Übergangsmatrix $(\lambda_i^j)_{0 \le i,j \le n}$ eine van der Mondesche Matrix ist, also nichtsingulär.

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

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

(o.Bew.) 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 Diagonalmatrizen 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 n von einer solchen Reduktion nicht betroffen wird. Wir können das gesuchte Produkt also auch in S ausrechnen.

Für zwei Polynome $a, b \in S$ 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

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

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 1 *Das Produkt zweier Polynome vom Grad $< n$ mit komplexen Koeffizienten kann mit $O(n\, \log(n))$ 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. $\omega^k \ne 1$ für $0 < k < N$. Für einen Körper fallen diese beiden Begriffe zusammen.

Lemma 2 (Verallgemeinerter Kürzungssatz)
Für eine N-te Hauptwurzel ω gilt $\sum_{i=0}^{N-1}\omega^i = 0$ und damit

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

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

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

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 2 *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*

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

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 3 *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 enthält, so kann man das Produkt zweier Polynome $a, b \in A[X]$ mit $deg(ab) = n$ mit $O(n \log(n))$ arithmetischen Operationen berechnen.*

Bemerkung: 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 lineare 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;

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

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

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

```
rstep(M,1);
```

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

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.

Satz 4 *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 billigeres 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 5 *Über einem Grundbereich mit Einheitskostenarithmetik lassen sich die wichtigsten Fragestellungen der linearen Algebra für eine quadratische n-reihigen 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 also 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.

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

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.

```
u:=randmat(10,10^2,Dom::Rational);
u0:=rtriang(u);
map(u0,x->length(numer(x)));
```

$$\begin{pmatrix}
1 & 2 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 2 \\
0 & 1 & 3 & 4 & 4 & 3 & 3 & 3 & 3 & 3 \\
0 & 0 & 1 & 5 & 5 & 5 & 5 & 4 & 5 & 5 \\
0 & 0 & 0 & 1 & 6 & 7 & 6 & 7 & 5 & 7 \\
0 & 0 & 0 & 0 & 1 & 9 & 8 & 9 & 8 & 6 \\
0 & 0 & 0 & 0 & 0 & 1 & 10 & 9 & 10 & 9 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 12 & 12 & 12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 13 & 15 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

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.

```
m0:=ntriang(m);
map(m0,length);
```

$$\begin{pmatrix}
4 & 5 & 5 & 5 \\
0 & 9 & 10 & 9 \\
0 & 0 & 18 & 17 \\
0 & 0 & 0 & 35
\end{pmatrix}$$

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

```
u0:=ntriang(u):
u0[1..4,1..6];
```

$$\begin{pmatrix} 35 & 73 & 83 & 66 & 84 & 60 \\ 0 & -1091 & -586 & -2162 & -1498 & 405 \\ 0 & 0 & -1315510 & 1733900 & -1529325 & -2195270 \\ 0 & 0 & 0 & -220360670950 & 775875860025 & -3373265572950 \end{pmatrix}$$

```
[ifactor(igcd(u0[k,i] $i=1..10)) $k=1..6];
```

$$\left[1, 1, 7 \cdot 5, 1091 \cdot 5^2 \cdot 7^2, 18793 \cdot 2^2 \cdot 5^4 \cdot 7^4 \cdot 1091^2, 4339 \cdot 19 \cdot 2^3 \cdot 5^8 \cdot 7^8 \cdot 1091^4 \cdot 18793^2\right]$$

Das systematische Auftreten des Faktors $35 = 7 \cdot 5$ und 1091 ist deutlich zu erkennen. Es handelt sich dabei jeweils um Pivotelemente, die in einem früheren `nstep` verwendet wurden.

Untersuchen wir dieses Phänomen näher. Wir verwenden dazu eine generische n-reihige Matrix A, 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;

A:=genmat(5);
```

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

```
A1:=nstep(A,1);
A2:=map(nstep(A1,2),normal);
[gcd(A2[k,i] $ i=1..5) $ k=1..5];
```

$$[1, 1, x_{11}, x_{11}, x_{11}]$$

Wir sehen, dass das erste Pivotelement x_{11} in in allen Einträgen von $A2[3..5, 3..5]$ als systematischer Faktor vorkommt und folglich ausgeteilt werden kann.

```
sysassign(A2[k,i],normal(A2[k,i]/A2[1,1])) $ i=3..5 $ k=3..5:
```

Ein typisches Element der daraus entstehenden Matrix hat die Gestalt

$$\texttt{A2}[4,5] = x_{11}x_{22}x_{45} - x_{11}x_{42}x_{25} - x_{12}x_{21}x_{45} + x_{12}x_{41}x_{25} + x_{21}x_{15}x_{42} - x_{22}x_{41}x_{15},$$

ist also die Determinante einer dreireihigen Teilmatrix der Ausgangsmatrix A. Führen wir mit dieser modifizierten Matrix $A2$ einen weiteren `nstep` aus und analysieren die Elemente der neuen Matrix $A3$.

```
A3:=map(nstep(A2,3),normal);
[gcd(A3[k,i] $ i=1..5) $ k=1..5];
```

$$[1, 1, 1, x_{12}x_{21} - x_{11}x_{22}, x_{12}x_{21} - x_{11}x_{22}]$$

Das Pivotelement $A2[2,2] = x_{11}x_{22} - x_{12}x_{21}$ kommt als gemeinsamer Faktor in allen Einträgen von A3[4..5,4..5] vor.

```
sysassign(A3[k,i],normal(A3[k,i]/A3[2,2])) $ i=4..5 $ k=4..5:
A3[4,5];
```

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

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

Der Allgemeingültigkeit dieser Aussage wollen wir nun auf den Grund gehen. Dazu bezeichnen wir mit $M_k(i,j)$ die Teilmatrix aus A, die aus den Elementen mit den Zeilennummern $1, 2, \ldots, k, i$ und den Spaltennummern $1, 2, \ldots, k, j$ besteht und mit $D_k(i,j)$ deren Determinante.

Die Struktur unserer Beispielrechnungen lässt folgenden Satz vermuten:

Satz 6 *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:

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

DTest(M,4,6,5);
expand(%);
```

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. Dieses Verfahren kann auch auf Matrizen mit Einträgen aus einem Polynomring $R = k[x_1, \ldots, x_m]$ angewendet werden, da dort ebenfalls eine exakte Division ausgeführt werden kann.

Für ganzzahlige Matrizen wächst die Bitlänge der Einträge der Matrizen, die in den verschiedenen Schritten im Bareissalgorithmus entstehen, nur noch linear: Haben die Einträge in der Ausgangsmatrix die Länge l, so wird im Schritt i zweimal das Produkt zweier Zahlen der Länge $i \cdot l$ berechnet (Aufwand: $2\,(il)^2$) und danach ein Faktor der Länge $(i-1)\,l$ abgespalten (Aufwand: $(i+1)(i-1)l^2$). Das Ergebnis hat dann die Bitlänge $(i+1)\,l$. Im Schritt i sind insgesamt $(n-i)^2$ solcher Operationen auszuführen, was sich zu einem Gesamtaufwand von $(n-i)^2(3i^2-1)l^2$ summiert. Für den Bareissalgorithmus erhalten wir damit eine Komplexität

$$\sum_{i=1}^{n-1} (n-i)^2(3i^2-1)l^2 = O(n^5\,l^2).$$

Satz 7 *Der Aufwand, eine n-reihige ganzzahlige Matrix, deren Einträge jeweils Bitlängen der Größenordnung l haben, mit dem Bareissverfahren zu triangulieren, ist von der Größenordnung $O(n^5\, l^2)$.*

Dasselbe gilt für Matrizen über einem Grundbereich $R = k[x]$ und dem Grad d dieser Polynome statt der Bitlänge l (k mit Einheitskosten vorausgesetzt).

Aufgabe 1 Zeigen Sie, dass der Aufwand für n-reihige Matrix mit Einträgen aus $R = k[x_1, \ldots, x_m]$ vom Grad $(d_1, \ldots, d_m)$ von der Größenordnung $O(n^{2m+3}\,(d_1 \cdot \ldots \cdot d_m)^2)$ ist.
(Hinweis: Zeigen Sie, dass die Polynome im Schritt i des Bareissverfahrens den Gradvektor $(i\,d_1, \ldots, i\,d_m)$ haben)

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 $O(n^3\, l(p)^2)$.
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. Diese Abschätzung liefert die folgende Schranke von Hadamard.

Satz 8 (Schranke von Hadamard) *Ist $A = ||a_{ij}||$ eine n-reihige Matrix, so gilt für die Determinante*

$$|det(A)| \le \prod_{i=1}^{n} \sqrt{\sum_{j=1}^{n} a_{ij}^2}.$$

Ist insbesondere A eine ganzzahlige Matrix mit Einträgen der Bitlänge l, so gilt für die Bitlänge der Determinante

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

Hierbei bedeutet $\widetilde{O}(m)$ (gelesen „soft Oh“), dass die Schranke bis auf logarithmische Faktoren zutrifft.

Der Beweis folgt etwa aus der Interpretation der Determinante als das Volumen des n-dimensionalen Parallelepipeds, das von den Zeilenvektoren von A aufgespannt wird und welches nicht größer als das Produkt der Längen dieser Vektoren ist. Zur selben Abschätzung kommt man auch über die Determinantendefinition selbst: $n!$ Summanden der Größe $n\,l$ ergeben eine Zahl der maximalen Größe $\log(n!) + n\,l = n(\log(n) + l)$.

Satz über Determinantenberechnung mit big prime einfügen.

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 9 (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 2 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: $x = 5y + 2$ wegen $x \equiv 2 \pmod 5$. Da außerdem noch $x = 5y + 2 \equiv 2 \pmod 3$ gilt, folgt $y \equiv 0 \pmod 3$, also $y = 3z$ und somit $x = 15z+2$. Schließlich muss auch $x = 15z+2 \equiv 1 \pmod 2$, also $z \equiv 1 \pmod 2$, d.h. $z = 2u + 1$ und somit $x = 30u + 17$ 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 $\texttt{CRA}(\,(x,x^2)\,:\, x \in \{2,3,5,7,11,13\})$.
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 $C_{CRA2} = O(k)$ und über alle Durchläufe $C_{CRA} = O(n^2)$.

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. Die folgende Funktion bestimmt die Determinante bis zu einer Schranke $M = 3234846615$ korrekt:

```
detmod:=proc(A,m)
  begin det(Dom::Matrix(Dom::IntegerMod(m))(A)) end;

moddet:=proc(A) local primes,u,i;
begin
  primes:=[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53];
  u:=[expr(detmod(A,primes[i])),primes[i]] $ i=1..nops(primes);
  i:=CRA(u);
  mods(i[1],i[2]);
end;
```

Nach der Schranke von Hadamard reicht es, $N = O(n\,(\log(n) + l))$ Primzahlen zu nehmen. Für die small primes Version der modularen Determinantenberechnung erhalten wir damit

Satz 10 *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 $O(n^3 \cdot (l+n) \cdot (l + \log(n))) = \widetilde{O}(n^3 \cdot (l+n) \cdot l)$.*

In der Tat, es sind zunächst $N = O(n\,(\log(n) + l))$ modulare Determinantenberechnungen durchzuführen. 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 eigentliche 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. Gesamtaufwand ist also $O(N \cdot (n^2(l+n) + N)) = O(N \cdot n^2(l+n))$.

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. Der entstehende Aufwand lässt sich 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 11 *Der Aufwand für eine modulare Version des Gaußalgorithmus auf einer n-reihigen ganzzahligen Matrix, deren Einträge eine Bitlänge der Größenordnung l haben, ist von der Größenordnung $O(n^4(log(n)+l)^2) = \widetilde{O}(n^4l^2)$.*

Das ist noch immer etwas besser als die Schranke $O(n^5l^2)$, die sich direkt für das Bareissverfahren ergeben hatte und mit den Kosten der modularen big primes Variante übereinstimmt.

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. $g \mid a,\ g \mid b$ und
2. $d \mid a,\ d \mid b \ \Rightarrow\ d \mid g$ 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 $g_1 \mid g_2$ und auch $g_2 \mid g_1$.
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. $a \mid k,\ b \mid k$ und
2. $a \mid d,\ b \mid d \ \Rightarrow\ k \mid d$ für alle $d \in D$

gilt.

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

Aufgabe 3 Zeigen Sie, dass in einem Integritätsbereich, in welchem gcd existieren und die Eigenschaft

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

erfüllt ist, auch lcm existieren und die Beziehung

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

gilt.

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 12 *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

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

aus, wodurch der Leitterm von r weggehoben wird.

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

Beispiel:

```
P:=Dom::UnivariatePolynomial(x);
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);

u:=divrem(P(f),P(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];
   print(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.

Kostenanalyse über Körpern mit Einheitskostenarithmetik

Zur Kostenanalyse betrachten wir zunächst den Fall, dass k ein Körper mit Einheitskostenarithmetik, also z. B. ein Restklassenkörper ist. Betrachten wir obige Polynome im Polynomring $k[x]$ über $k = \mathbb{Z}_{23}$, so ergibt sich nacheinander

```
P:=Dom::UnivariatePolynomial(x,Dom::IntegerMod(23));
Euklid(P(f),P(g));
```

$$2x^4 - 5x^2 - 8$$
$$-x^2 - 9x + 2$$
$$-8 + 10x$$
$$-4$$

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 genau wie bei den entsprechenden Algorithmen für ganze Zahlen

$C_{divrem} = O(\deg(g)\,(\deg(f) - \deg(g))$ für die Komplexität der Division mit Rest und

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

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

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

$$f(x) \sim_R cont(f) \cdot pp(f).$$

$pp(f)$ 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 $pp(f) := pp(r \cdot f)$ von f definieren. Ebenso erhalten wir einen Inhalt $cont(f) := \frac{cont(r \cdot f)}{r} \in K$.

Aufgabe 4 Zeigen Sie,

- dass $cont(f)$ und $pp(f)$ 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 $pp(pp(f)) \sim_R pp(f)$ 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. $\forall\ f,g \in R[x]\ cont(fg) \sim cont(f)cont(g)$,
2. $\forall\ f,g \in K[x]\ pp(fg) \sim pp(f)pp(g)$,
3. $\forall\ f,g \in R[x]\ g \mid f\ \Rightarrow\ cont(g) \mid cont(f),\ pp(g) \mid pp(f)$,
4. $\forall\ f \in R[x], r \in R\ cont(rf) \sim r \cdot cont(f),\ pp(rf) \sim pp(f)$,
5. $\forall\ f,g \in K[x]\ g \mid f$ *in* $K[x]\ \Rightarrow\ pp(g) \mid pp(f)$ *in* $R[x]$

und als zentraler Satz über die Berechnung polynomialer größter gemeinsamer Teiler

Satz 13 (Kompositionssatz) *Die gcd-Berechnung von $f,g \in R[x]$ kann man auf die in R und in $K[x]$ zurückführen.*

Es gilt

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

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

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

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

Damit spielt die gcd-Berechnung von univariaten Polynomen über einem Körper und dabei der Euklidsche Algorithmus eine zentrale Rolle. 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 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*

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

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 14 *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 als Matrix aufschreiben ergibt die *Sylvester-Matrix* S. Deren Determinante bezeichnet man als die *Resultante* $res(f, g)$.

Folgerung 2 *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 15 (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 3 *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 $t = -f/h$.

gcd von Familien von Polynomen

Rückführung auf die Bestimmung von $\gcd(f_1, \alpha_1 f_2 + \ldots \alpha_N f_N)$ und damit die praktischen Kosten eines probabilistischen Verfahrens auf der Basis des Kompositionssatzes:

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

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 $h = \gcd_{K[x]}(f,g)$.

3.5 Polynome in mehreren Veränderlichen

Die Berechnung des gcd von Polynomen in mehreren Veränderlichen $x_1, \ldots, x_n$ können wir nach dem Kompositionssatz auf die Berechnung des gcd der Koeffizienten in $R = k[x_1, \ldots, x_{n-1}]$ sowie die Berechnung des gcd univariater Polynome in $K[x = x_n]$ über dem Quotientenkörper $K = Q(R)$ zurückführen. Die erste Teilaufgabe nimmt auf die Aufgabenstellung rekursiv Bezug. Die zweite Teilaufgabe kann theoretisch mit Hilfe des Euklidschen Algorithmus gelöst werden, allerdings ist die Arithmetik in K dabei keine Einheitskostenarithmetik mehr.

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

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

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

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

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

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

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

Wiederum erhalten wir als Ergebnis, dass f und g relativ prim sind, stellen aber ein unverhältnismäßiges Wachstum der an der Rechnung beteiligten Koeffizienten fest. In jedem Schritt wurden 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 `rpoly`, die Zufallspolynome erzeugt, generiert werden.

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

P:=Dom::UnivariatePolynomial(x,Dom::Rational);
Euklid(P(rpoly(9,10)),P(rpoly(5,10)));
```

$$1193/243\,x^4 - 2617/243\,x^3 - 1301/243\,x^2 - 1132/243\,x + 553/243$$
$$11015190/1423249\,x^3 + 9351612/1423249\,x^2 + 1579014/1423249\,x - 2466207/1423249$$
$$91873569448/13869960075\,x^2 - 39334332613/27739920150\,x - 9881617807/9246640050$$
$$48860110585044675/11861245310014592\,x - 4808978687243925/11861245310014592$$
$$-21879733479287971994176/1912454761123820476867 5$$

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.

```
Euklid(P(rpoly(4,100)*(x-1)),P(rpoly(3,100)*(x-1)));
```

$$509875124/1071225\,x^2 - 193844783/357075\,x + 2866369/42849$$
$$-6283157125781025/1074267116008328\,x + 6283157125781025/1074267116008328$$

Der berechnete gcd unterscheidet sich von $(x-1)$ um ein $\mathbb{Q}$-skalares Vielfaches, wie aus der Theorie nicht anders zu erwarten ist. Dieser „eingeschleppte“ Faktor, der in allen Koeffizienten des Ergebnispolynoms enthalten ist, ist deutlich komplizierter als die Koeffizienten des endgültigen Ergebnisses der gcd-Berechnung über $\mathbb{Z}[x]$, die $\gcd = x - 1$ ergibt.

Da $h = gcd_{\mathbb{Z}[x]}(f(x), g(x))$ ein Teiler von $f(x)$ und $g(x)$ ist, also insbesondere $lc(h)$ die Leitkoeffizienten von f und g teilen muss, ist es kein Zufall, dass sich ein solcher gemeinsamer Faktor im Ergebnis wiederfindet. Dieser kann durch Normierung des Leitkoeffizienten auf 1 ausdividiert werden, muss also nicht mühsam durch eine Inhaltsberechnung ermittelt werden.

Normiert man den Leitkoeffizient auch aller Zwischenergebnisse, so ist zu sehen, dass dieser Effekt nur im Endergebnis auftritt.

```
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;
   print(r);
   f:=g; g:=r;
  end;
  f;
end;
```

```
nEuklid(P(rpoly(4,100)*(x-1)),P(rpoly(3,100)*(x-1)));
```

$$x^2 - 33410537/26247473\,x + 7163064/26247473$$
$$x - 1$$

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

Polynome in zwei Veränderlichen

Die Berechnung des gcd von Polynomen in mehreren Veränderlichen, also etwa in $\mathbb{Q}[x,y]$, wird nach dem allgemeinen Ansatz auf die Berechnung von gcd in $\mathbb{Q}(y)[x]$ sowie die Berechnung von gcd in $\mathbb{Q}[y]$ zurückgeführt. Für die erste der beiden Aufgaben kann wieder der Euklidsche Algorithmus eingesetzt werden.

Im folgenden Beispiel wird der gcd von $f_4 = \sum_{i=0}^{4}(y-1)^i\,x^i$ und $g_4 = \sum_{i=0}^{3}(y+1)^i\,x^i$ berechnet. Wir definieren dazu eine Funktion `gpoly`, mit der univariate Polynome erzeugt werden können, deren Koeffizienten sich einheitlich durch eine Formel beschreiben lassen.

```
gpoly:=proc(d,a) local i;
begin _plus(a(i)*x^i$i=0..d) end_proc;

P:=Dom::UnivariatePolynomial(x);

f:=gpoly(4,i->(y-1)^i);
g:=gpoly(3,i->(y+1)^i);
Euklid(P(f),P(g));
```

$$\frac{(-4\,y+2\,y^2+2)}{(y+1)}\,x^2 + \frac{(-4\,y+4\,y^2)}{(2\,y+y^2+1)}\,x + \frac{(-2\,y+12\,y^2+2\,y^3+y^4+3)}{(4\,y+6\,y^2+4\,y^3+y^4+1)}$$

$$\frac{(2\,y-y^2-1)}{2}\,x + \frac{(4\,y+6\,y^2+4\,y^3+y^4+1)}{(6\,y-6\,y^2+2\,y^3-2)}$$

$$\frac{(230\,y^2+1271\,y^4+1924\,y^6+611\,y^8+54\,y^{10}+y^{12}+5)}{(-4\,y+2\,y^2+12\,y^3-17\,y^4-8\,y^5+28\,y^6-8\,y^7-17\,y^8+12\,y^9+2\,y^{10}-4\,y^{11}+y^{12}+1)}$$

Auch hierbei sind gcd-Berechnungen zum Kürzen der Koeffizenten erforderlich, wobei für Polynome in mehreren Veränderlichen diese gcd-Berechnungen rekursiv aufgerufen werden mit polynomialen Koeffizienten von wachsendem Grad. Das führt natürlich zu nicht akzeptablen Laufzeiten, so dass wir, wie im Fall des nennerfreien Gauß-Algorithmus und des Bareiss-Verfahrens, eine „nennerfreie“ Version des Euklidschen Algorithmus untersuchen wollen, in der zunächst überhaupt nicht gekürzt wird, und dann gezielt nach Faktoren schauen wollen, die sich systematisch aus den Rechnungen austeilen lassen.

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

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

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

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

Beispiel:

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

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

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 $f(x), g(x) \in R[x], d(f) \geq d(g)$, so gibt es eindeutig bestimmte Polynome $q(x), r(x) \in R[x]$ mit $r = 0$ oder $d(r) < d(g)$ und

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

q nennt man den *Pseudoquotienten* $pquot(f, g)$ und r nennt man den *Pseudorest* $prem(f, g)$ der Polynome f und g.

Über $K[x]$ 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];
     f:=g; g:=r;
  end;
  f;
end;
```

Beispielrechnungen mit dieser Implementierung sind im Hinblick auf Vermeidung von Rechenaufwand auf den ersten Blick allerdings 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 wie bereits im Fall linearer Gleichungssysteme, dass sich diese 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 $d = d(f) \geq d(g)$. 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) $d(R_1) > d(R_2) > \ \ldots,\ > d(R_k)$.

(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 $pp(R_k) = pp(\gcd_{K[x]}(f, g))$.

Setzen wir $\delta_i := d(R_{i-1}) - d(R_i) = d(Q_i)$, 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.

Bezeichnet q_i die durchschnittliche Größe der Koeffizienten von $Q_i(x)$ und l_i die durchschnittliche Größe der Koeffizienten von $R_i(x)$ (also eine Schranke für deren Bitlänge für $R = \mathbb{Z}$ und den Gradvektor für $R = k[x_1, \ldots, x_{n-1}]$), so gilt

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

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

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

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

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

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 Multiplikationskosten $C^*_R(u, v)$ ergeben sich zu

$C^*_R(u,v) = O(u \cdot v)$	für $R = \mathbb{Z}$ und Bitlängenschranken u, v
$C^*_R(u,v) = O(L(u) \cdot L(v))$	für $R = k[x_1, \ldots, x_{n-1}]$ und Gradvektorschranken u, v, wobei $L(u)$ für das Produkt der Gradschanken steht.

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

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

und analog für $R = k[x_1, \ldots, x_{n-1}],\ l_i \sim 2^i(d_1, \ldots, d_{n-1})$ und $d = d_n$

$$C_{EPRS}(n,d) \sim 4^{(n-1)\,d} \cdot (d_1 \cdot \ldots \cdot d_n)^2$$

2) Man könnte auch in jedem Schritt R_{i+1} durch seinen primitiven Teil ersetzen, also $\beta_i = cont(R_{i+1})$ 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:=rpoly(5,100); g:=rpoly(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. Die Koeffizientenpolynome in Zwischenschritten haben, wie wir sehen konnten, meist deutlich höheren Grad als die Koeffizienten der Ausgangspolynome (Gradexplosion). Selbst bei moderaten Annahmen über diese Gradexplosion ist die Laufzeit dieses Verfahrens exponentiell in der Koeffizientengröße und doppelt exponentiell in der Anzahl der Variablen, obwohl deutlich kleinere Zahlen im Vergleich zur Euklidschen PRS entstehen.

Es ergibt sich also die Frage, ob man ähnlich wie beim Bareiss-Algorithmus Faktoren, die man in früheren Schritten hineingesteckt hat, in späteren Schritten wieder herausdividieren kann, ohne in jedem Fall erst eine aufwändige gcd-Berechnung auf den Koeffizienten zu starten. Dies ist in der Tat möglich:

Satz 16 (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;
     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 5 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 17 (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);
     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 6 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

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

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 $C_R^*(u\, l, v\, l) = (uv)^{n-1} \cdot D^2$

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

Summieren wir den Aufwand der einzelnen Schritte, und berücksichtigen wegen $d = d_n$, dass $L(f) = d_1 \cdot \ldots \cdot d_n = D \cdot d$ gilt, so erhalten wir

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

Satz 18 *Sei $R[x_n]$ der univariate Polynomring über $R = k[x_1, \ldots, x_{n-1}]$ und einem Grundkörper k mit Einheitskostenarithmetik und $f, g \in R[x_n]$ zwei Polynome mit durch $(d_1, \ldots, d_n)$ beschränkten Gradvektoren.*

Zur Berechnung der Subresultanten-PRS werden dann $O((d_1 \cdots d_n)^2 (d_n)^{2n-2})$ Elementarmultiplikationen benötigt.

Aufgabe 7 Zeigen Sie, dass die entsprechende Komplexität über $R = \mathbb{Z}$ für Polynome mit Koeffizientenlängen, die durch $t(f), t(g) \leq b$ beschränkt sind, von der Größenordnung $O(b^2 d^4)$ und über $R = \mathbb{Z}[x_1, \ldots, x_{n-1}]$ von der Größenordnung $O((b \cdot d_1 \cdots d_n)^2 (d_n)^{2n})$ ist.

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

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

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

$h(x) := pp(gcd_{K[x]}(f(x), g(x)))$ 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 $R(x) = (r \cdot R_k(x))/lc(R_k) \in R[x]$ 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 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 Berechnungen von maximal $3d_n$ gcd's von Polynomen in $n-1$ Variablen auszuführen, deren Gradvektor durch $(d_1, \ldots, d_{n-1})$ beschränkt ist. Bezeichnet $C_{\text{gcd}}^{(n)}$ 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

$$C_{\text{gcd}}^{(n)} \lesssim 3d_n \cdot C_{\text{gcd}}^{(n-1)} + (d_1 \cdots d_n)^2 (d_n)^{2n-2}$$

und daraus rekursiv

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

Satz 19 *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 von der Größenordnung*

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

Insbesondere bestätigt diese Schranke, dass man die Variablen so anordnen sollte, dass $d_1 \geq d_2 \geq \ldots \geq d_n$ gilt.

Verwendet man für die gcd-Berechnung der Koeffizienten das probabilistische Verfahren für Polynomfamilien, so reduziert sich die Komplexitätsformel auf

$$C_{\gcd}^{(n)} \lesssim C_{\gcd}^{(n-1)} + (d_1 \cdots d_{n-1})^2 (d_n)^{2n}\,,$$

was aber auf ein ähnliches Ergebnis führt, da in diesem Ansatz die Rechenzeit durch die PRS-Berechnung dominiert wird. Die genaue Komplexität ist

$$C_{\gcd}^{(n)}(d_1,\ldots,d_n) = O\left(\sum_{i=1}^{n} (d_1 \ldots d_{i-1})^2 d_i^{2\,i}\right)\,.$$

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

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

Rufen wir die (wie folgt modifizierte[1]) Funktion `Euklid`

```
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;
   print(r);
   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(Q(f,p),Q(g,p)))]);
```

[1] Grund ist ein Bug in der MuPAD-Version 2.5

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

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(Q(f*(3*x+5),p),Q(g*(3*x+5),p)))]);
```

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

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.

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*. Für $f \in R[x]$ schreiben wir auch kurz $\overline{f}$ für $\phi(f)$.

Satz 20 (Reduktionssatz) *([4, thm 6.26])*
Sei $\overline{\ }: R[x] \to R'[x]$ *ein Reduktionsmorphismus,* $0 \neq \underline{f}, \underline{g} \in R[x]$ *mit* $\overline{lc(f)\,lc(g)} \neq 0$. *Sei weiter* $h = \gcd(f, g), f = h \cdot f', g = h \cdot g'$ *und* $h^* = \gcd(\overline{f}, \overline{g})$. *Dann gilt*

1. $\deg_x(\overline{f}) = \deg_x(f), \deg_x(\overline{g}) = \deg_x(g), \deg_x(\overline{h}) = \deg_x(h)$,
2. $lc(h) \mid \gcd(lc(f), lc(g))$,
3. $\deg_x(h^*) \geq \deg_x(h)$,
4. $\deg_x(h^*) = \deg_x(h) \Leftrightarrow \overline{r} \neq 0$, *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 $\overline{h} \,|\, \overline{f}, \overline{g}$ und somit $\overline{h} \,|\, h^*$ nach sich zieht.

Für die letzte Aussage halten wir zunächst die Beziehung

$$h^* = \gcd(\overline{f}, \overline{g}) = \overline{h} \cdot \gcd(\overline{f'}, \overline{g'})$$

fest. Die Grade der beiden Polynome h^* und h stimmen also genau dann überein, wenn $\deg_x(\gcd(\overline{f'}, \overline{g'})) = 0$ gilt, was nach obigem Satz gleichbedeutend mit $res(\overline{f'}, \overline{g'}) \neq 0$ ist. Nun ergibt sich aber die Resultante als Determinante der Sylvestermatrix, die aus den Koeffizienten von $\overline{f'}$ und $\overline{g'}$ zusammengestellt wird. Diese Koeffizienten ergeben sich wiederum als Reduktion der Koeffizienten von f' und g', so dass folglich $res(\overline{f'}, \overline{g'}) = \overline{r}$ gilt. □

Die Bedingung an die Leitkoeffizienten lässt sich noch leicht abschwächen. Andererseits hatten wir oben gesehen, dass im Fall $\overline{lc(f)} = \overline{lc(g)} = 0$ der Grad des modularen gcd in der Tat kleiner als erwartet sein kann.

Als erste Folgerung aus Satz 20 ergibt sich für primitive Polynome

Folgerung 4 *Sind $0 \neq f, g \in R[x]$ primitive Polynome und $\overline{f}, \overline{g}$ unter einer Reduktion (mit $\overline{lc(f)\, lc(g)} \neq 0$) 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.

Für $f \in R[x]$ bezeichnen wir wieder $\overline{f} = \phi_a(f) \in K[x]$ und können versuchen, aus genügend vielen Evaluierungsdaten $a \in K^{n-1}$ die Koeffizienten des gcd zweier Polynome zu bestimmen. Dies führt auf ein lineares Gleichungssystem in den unbestimmten Koeffizienten, wenn das Gradmuster des gcd bekannt ist.

Mit obigem Korollar können wir insbesondere schnell beweisen, dass zwei Polynome zueinander teilerfremd sind. Betrachten wir etwa die beiden Polynome

$$f(x,t) = \sum_{i=0}^{7} (t+i) \cdot x^i,$$

$$g(x,t) = \sum_{i=0}^{6} (t-i^2) \cdot x^i$$

im bivariaten Polynomring $\mathbb{Q}[t,x] = \mathbb{Q}[t][x]$ und die Evaluation, welche durch $t = 1$ induziert wird, so erhalten wir

$$\overline{f} = \sum_{i=0}^{7} (i+1) \cdot x^i = 8 \cdot x^7 + 7 \cdot x^6 + 6 \cdot x^5 + 5 \cdot x^4 + 4 \cdot x^3 + 3 \cdot x^2 + 2 \cdot x + 1,$$

$$\overline{g} = \sum_{i=0}^{6} (1-i^2) \cdot x^i = -35 \cdot x^6 - 24 \cdot x^5 - 15 \cdot x^4 - 8 \cdot x^3 - 3 \cdot x^2 + 1$$

Aus der Teilerfremdheit von $\overline{f}$ und $\overline{g}$ können wir auf die Teilerfremdheit der beiden Ausgangspolynome schließen.

```
f:=_plus((t+i)*x^i$i=0..7);
g:=_plus((t-i^2)*x^i$i=0..6);

UP:=x->Dom::UnivariatePolynomial(x);
Euklid(UP(x)(f),UP(x)(g));
Euklid(UP(x)(subs(f,t=1)),UP(x)(subs(g,t=1)));
```

Der Unterschied im Rechenaufwand zwischen beiden Varianten ist deutlich zu sehen.

3.8 Modulare gcd-Berechnung

3.8.1 Modulare bivariate gcd-Berechnung (big prime)

Betrachten wir als weiteres Beispiel die Berechnung des gcd für Polynome in zwei Veränderlichen x, y über einem Körper k mit Einheitskostenarithmetik. Seien dazu $0 \neq f, g \in R[x]$ zwei primitive Polynome in rekursiver Darstellung, also $R = k[y]$. 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 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* $\deg_y(h) \leq \deg_y(f)$.

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 $c \geq \deg_y(r)$ reduzieren und so das Gradwachstum begrenzen. Dies entspricht der Reduktion $\overline{\ }: R \mapsto R' = R/(p)$. 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 eine Einbettung $R'[x] \subset R[x]$ als Mengen.

Setzen wir zusätzlich voraus, dass p ein Primpolynom ist, wird R' ein Körper und die gcd-Berechnung in $R'[x]$ besonders einfach. Dazu müssen in R' insbesondere inverse Elemente berechnet werden können. 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[2], $h^* = \gcd_{R'[x]}(f, g)$ bestimmen und untersuchen, wann daraus Rückschlüsse auf $h = \gcd_{R[x]}(f, g)$ möglich sind. Wir verwenden im Weiteren die in Satz 20 eingeführten Bezeichnungen. Weiter seien f^*, g^* reduzierte Polynome mit $f \equiv f^* h^* \pmod p,\ g \equiv g^* h^* \pmod p$, die sich als Kofaktoren von h^* über $R'[x]$ berechnen lassen.

[2]Womit die Bedingung an die Leitkoeffizienten automatisch erfüllt ist.

Satz 20 (4) besagt, dass ein zu hoher x-Grad von h^* nur für $res(f', g') \equiv 0 \pmod p$ 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 unterscheiden sich h^* und h in $R'[x]$ (!) nur um den konstanten (aber unbekannten) Faktor $\alpha = lc(h)$, wenn wir davon ausgehen, dass h^* als gcd zweier univariater Polynome über dem Körper R' so normiert ist, dass $lc(h^*) = 1$ gilt:

$$\alpha\, h^* \equiv h \pmod p$$

h als reduziertes Polynom wäre damit gefunden. Da wir α aber nicht kennen, multiplizieren wir diese Gleichung mit $\beta = \gcd(lc(f), lc(g))$ durch. Wegen $\alpha \mid \beta$ und

$$\beta\, h^* \equiv \frac{\beta}{\alpha}\, h \equiv w \pmod p$$

ist das reduzierte Polynom w ein R-Vielfaches von h, wenn auch $\frac{\beta}{\alpha}h$ reduziert ist. 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 das reduzierte Polynom` $h^* = \gcd_{R'[x]}(f, g)$ `mit` $lc(h^*) = 1$
    `mit dem Euklidschen Algorithmus.`
  `Bestimme das reduzierte Polynom` $w \in R[x]$ `mit` $w \equiv \beta\, h^* \pmod p$`.`
`until` $w \mid \beta f,\ w \mid \beta g$`.`
`return` $pp_x(w)$`.`

Kommen wir zu einer Komplexitätsabschätzung dieses Verfahrens. 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 20 ist dies nur möglich, wenn $p \mid r = res(f', g')$ in R gilt. Aus der Gestalt der Sylvestermatrix erhalten wir $\deg(r) < 2\, d_1\, d_2$, so dass wir nach maximal $s = \frac{2\, d_1\, d_2}{c} < 2\, d_1$ Durchläufen auf eine gute Reduktion stoßen. Zufällige Wahl von p liefert mit Wahrscheinlichkeit (fast) 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 $O(c^2\, d_1^2) = O(d_1^2 d_2^2)$. Die Kosten der anderen

Schritte übersteigen diese Größenordnung nicht. Insbesondere ist der Aufwand für die abschließende Teilbarkeitsprüfung von dieser Größenordnung (klassische Multiplikation), denn die Gradvektoren sind durch $(2d_1, d_2)$ beschränkt.

Wir haben damit den folgenden Satz bewiesen.

Satz 21 *$f, g \in R[x]$ 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 (nahe) 1. Nach maximal $2\,d_1$ 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 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. Zusammen mit dem Reduktionsmorphismus $\mathbb{Z} \to \mathbb{Z}_p$ können wir auch hier $\mathbb{Z}_p[x]$ in $\mathbb{Z}[x]$ einbetten, indem wir jeden Koeffizienten (mod p) reduzieren, d.h. durch den eindeutig bestimmten Repräsentanten im Intervall $\left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$ ersetzen. Entsprechende Polynome wollen wir auch in diesem Fall als *reduziert* bezeichnen.

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 das reduzierte Polynom` $h^* = \gcd_{\mathbb{Z}_p[x]}(f, g)$ `mit` $lc(h^*) = 1$
      `mit dem Euklidschen Algorithmus in` $\mathbb{Z}_p[x]$`.`
  `Bestimme das reduzierte Polynom` $w \in \mathbb{Z}[x]$ `mit` $w \equiv \beta\, h^* \pmod p$`.`
`until` $w \,|\, \beta f,\ w \,|\, \beta g$`.`
`return` $pp_x(w)$`.`

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(Q(f,p),Q(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 $w = 6x + 3$ für $p \geq 13$ stabil gefunden wird.

Analog oben gilt mit dem Bezeichnungen von Satz 20 auch hier

Satz 22 *$f, g \in \mathbb{Z}[x]$ seien zwei primitive Polynome vom Grad $< d$ mit Koeffizienten, deren Bitgröße durch b beschränkt sei und p eine ausreichend große Primzahl der Bitgröße c.*

1. *p ist eine schlechte Reduktion $\Leftrightarrow p \,|\, res(f', g')$.*
2. *Die Hauptschleife von* `modularBigPrimeGCD(f,g)` *berechnet h für jede gute Reduktion korrekt.*
3. *Die Kosten für einen Durchlauf der Hauptschleife sind von der Größenordnung $O(c^2\, d^2)$.*

Mignottes Faktor-Schranke

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 $h \,|\, f$ auch $\deg_y(h) \leq deg_y(f)$ 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 $g := (x+1)^2 \cdot (x^2 - x + 1) = x^4 + x^3 + x + 1$ haben den gcd $x^2 + 2\,x + 1$.

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

Dazu definieren wir

- die **2-Norm** $\|f\|_2 := \sqrt{\sum |f_i|^2} \in \mathbb{R}_+$,
- die **1-Norm** $\|f\|_1 := \sum |f_i| \in \mathbb{R}_+$ und
- die **∞-Norm** $\|f\|_\infty := \max |f_i| \in \mathbb{R}_+$

wobei $|z| = \sqrt{z\overline{z}}$ den Betrag der komplexen Zahl z bezeichnet.
Offensichtlich gilt

$$\|f\|_\infty \le \|f\|_2 \le \|f\|_1 \le (n+1)\,\|f\|_\infty \quad \text{und auch} \quad \|f\|_2 \le \sqrt{n+1}\,\|f\|_\infty$$

Die Koeffizientengröße $b(f)$ als Bitlänge kann durch $\log(\|f\|_\infty)$ abgeschätzt werden. $\log(\|f\|_*)$ ist also bis auf einen Summanden $\log(n)$ für jede dieser Normen ein geeignetes Maß für die Koeffizientengröße.
Das Polynom f lässt sich über $\mathbb{C}$ in Linearfaktoren $f = f_n \prod (x - z_i)$ zerlegen. Für eine solche Nullstelle gilt $f = (x - z)g$.

Lemma 6

$$\|(x - z)g\|_2 = \|(\overline{z}x - 1)g\|_2$$

Beweis: [4, lemma 6.30.].
Wir definieren das *Landau-Maß* von f als $M(f) = |f_n| \prod \max(1, |z_i|)$. Für dieses Maß gilt $M(f\,g) = M(f)M(g)$.

Lemma 7 (Landau-Ungleichung) *Für $f \in \mathbb{C}[x]$ gilt $M(f) \le \|f\|_2$.*

Beweis: [4, thm 6.31.].

Lemma 8 *Für $h \in \mathbb{C}[x]$ und $\deg(h) = m$ gilt $\|h\|_1 \le 2^m M(h)$.*

Beweis: ([4, thm 6.32.]) OBdA ist $h = h_0 + h_1 x + \cdots + h_m x^m$ und $h_m = 1$ (die Ungleichung skaliert entsprechend), so dass wir weiter $h = \prod_{i=1}^{m} (x - z_i)$ annehmen können. Dann gilt

$$|h_i| = |e_{m-i}| \le \binom{m}{i} M(h),$$

wobei e_k die k-te elementarsymmetrische Summe der $(z_1, \ldots, z_m)$ bezeichnet. Aufsummieren beweist die behauptete Ungleichung. □

Folgerung 5 (Mignotte-Schranke) *Sind $f, g, h \in \mathbb{Z}[x]$ Polynome vom Grad n, m, k und $g\,h \mid f$ (in $\mathbb{Z}[x]$), so gilt*

$$\|g\|_\infty \|h\|_\infty \le \|g\|_2 \|h\|_2 \le \|g\|_1 \|h\|_1 \le 2^{m+k} \|f\|_2 \le \sqrt{n+1} \cdot 2^{m+k} \|f\|_\infty \tag{1}$$

$$\|h\|_\infty \le \|h\|_2 \le 2^k \|f\|_2 \le 2^k \|f\|_1 \tag{2}$$

$$\|h\|_\infty \le \|h\|_2 \le \sqrt{n+1} \cdot 2^k \|f\|_\infty \tag{3}$$

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 βf oder βg die Koeffizientenschranke $\|w\|_\infty \le B = \sqrt{d+1} \cdot 2^{d+2\,b}$. Wir können also $2B$ als untere Schranke für p und damit $c = O(d + b)$ wählen, wenn wir eine gute Reduktion in der Nähe dieser Schranke finden.
Die Kosten eines Durchlaufs der Hauptschleife von `modularBigPrimeGCD` für eine gute Reduktion dieser Größe ist dann $O(c^2\,d^2) = O((b^2 + d^2)d^2)$.

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 $\gcd_{\mathbb{Z}[x]}(f, g)$. Der Index p (etwa bei f_p) weist darauf hin, dass das entsprechende Polynom p-reduziert ist bzw. wurde.

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)`:=(`f^*, M`):=(`g^*, M`):=(0,1)`
`d:=`$\deg(f) + \deg(g) + 1$ `(* garantiert zu groß *)`
`repeat`
  `Wähle eine Primzahl` $p \not| lc(f) \cdot lc(g),\ (p, M) = 1$.
  `Berechne` p`-reduziertes Polynom` $h_p \equiv \gcd_{\mathbb{Z}_p[x]}(f_p, g_p) \pmod{p}$
    `mit` $lc(h_p) = 1$ `mit dem Euklidschen Algorithmus.`
  `if` $(\deg(h_p) < d)$ `then`
        `(* alle bisherigen` p `waren schlechte Reduktionen *)`
    `d:=`$\deg(h_p)$`,` (w, M)`:=(`f^*, M`):=(`g^*, M`):=(0,1)`
  `else if` $\deg(h_p) > d$ `then continue (*` p `ist schlechte Reduktion *)`

```
    Bestimme das p-reduzierte Polynom w_p ≡ βh_p (mod p)
    Lifte das Ergebnis: (w, M):=Lift((w, M), (w_p, p))
until w | βf, w | βg.
return pp(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. Fassen wir diese Aussagen im folgenden Satz zusammen

Satz 23 *$f, g \in \mathbb{Z}[x]$ seien zwei primitive Polynome.*

1. *p ist eine schlechte Reduktion $\Leftrightarrow p \,|\, res(f', g')$.*
2. *Beginnend mit der ersten guten Reduktion gilt $d = \deg(h)$ in der Hauptschleife von* `modularSmallPrimes(f,g)`*, so dass ab da alle Ergebnisse zur Berechnung von w beitragen.*
3. *Die Hauptschleife von* `modularSmallPrimes(f,g)` *terminiert nach endlich vielen Durchläufen.*

Zur Komplexität dieses Algorithmus

Wir werden den Algorithmus mit Reduktionen der Größe eines Computerworts durchlaufen. Die Hauptschleife terminiert spätestens dann, wenn alle s schlechten Reduktionen durchlaufen sind und der kumulierte Modul M groß genug ist. Nach k guten Reduktionen hat M die Bitlänge $k \cdot l(p) = O(k)$ und ist garantiert groß genug, wenn die Mignotte-Schranke $d + 2\,b$ überschritten ist.

Zur Bestimmung einer Schranke für die Anzahl der schlechten Reduktionen schätzen wir $r = |res(f', g')|$ ab. Da die Resultante eine Determinante ist liefert die Hadamard-Schranke

$$|r| \le ||f'||_2^{\deg(g')} ||g'||_2^{\deg(f')}$$

und $s \le l(r) \le 2d\,(2d+b)$, was allerdings eine sehr pessimistische Schranke ist. Wir können zunächst für eine genügende Anzahl c von Reduktionen den gcd in $\mathbb{Z}_p[x]$ bestimmen (Kosten: $O(c \cdot d^2)$) und die Liftungen nur für die erforderlichen $k = O(d+b)$ guten Reduktionen unter ihnen ausführen.

Zum Liften der i-ten guten Reduktion sind aus dem bisher kumulierten (Koeffizientenlänge $O(i)$) und dem neuen Ergebnis (Koeffizientenlänge $O(1)$) das neue kumulierte Ergebnis zu berechnen (Kosten: $O(d \cdot i)$), was über alle k Kumulationsschritte Kosten der Größe $O(d \cdot k^2)$ verursacht.

Den Terminationstest können wir ganz am Ende einmal ausführen, so dass wir noch Kosten der Größe $O(d^2 \cdot k^2)$ zu berücksichtigen haben. Verwenden wir an dieser Stelle ein schnelles Multiplikationsverfahren, so lassen sich diese Kosten bis auf $\widetilde{O}(d \cdot k)$ drücken.

Damit ergibt sich der folgende

Satz 24 *$f, g \in \mathbb{Z}[x]$ seien zwei primitive Polynome vom Grad $< d$ mit Koeffizienten, deren Bitgröße durch b beschränkt sei. Dann lassen sich die Kosten von* `modularSmallPrimesGCD` *durch $O(d^2(d^2+b^2))$ abschätzen.*

Mit hoher Wahrscheinlichkeit werden deutlich weniger ($\le O(b+d)$) schlechte Reduktionen durchlaufen, so dass der Flaschenhals dieses Algorithmus die Überprüfung der Testbedingung ist. Setzt man hier ein schnelles Multiplikationsverfahren ein, so ist die Laufzeit des Algorithmus von der Größenordnung $\widetilde{O}(d\,(d^2+b^2))$.

`modularSmallPrimesGCD` in dieser Version ist also besser als `modularBigPrimeGCD`.

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 $T(x_1, \ldots, x_n)$ 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.

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: (erwarteter) Grad des gcd
```

β:=gcd$(lc(f), lc(g))$

```
(w, M):=(f*, M):=(g*, M):=(0,1)
d:=lt(f) + lt(g)  (* garantiert zu groß *)
repeat
  Wähle eine Primzahl p ∤ lc(f)·lc(g),  (p, M) = 1.

  Berechne p-reduziertes Polynom h_p ≡ gcd_{S'}(f, g) (mod p)
    mit lc(h_p) = 1 mit dem Euklidschen Algorithmus.

  if (lt(h_p) < d) then
        (* alle bisherigen p waren schlechte Reduktionen *)
    d:=lt(h_p), (w, M):=(f*, M):=(g*, M):=(0,1)
  else if lt(h_p) > d then continue (* p ist schlechte Reduktion *)

  Bestimme das p-reduzierte Polynom w_p ≡ β h_p (mod p)

  Lifte das Ergebnis: (w, M):=Lift((w, M), (w_p, p))
until w | βf,  w | βg.
return pp(w)
```

Kostenanalyse

Ist 2^c eine Schranke für die Bitlänge der Koeffizienten von $h = \gcd(f, g)$ 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 $O(c \cdot C_{\mathrm{mod-gcd}}(d_1, \ldots, d_n)$, 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 25 *Der Aufwand zur Berechnung von* `multivariateModularSmallPrimesGCD(f,g)` *ist unter obigen Annahmen von der Ordnung*

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

Auch diese Schranke kann durch Verwendung einer schnellen Multiplikation in der Testbedingung bis auf $\widetilde{O}(c^2 \cdot D + c \cdot C_{\mathrm{mod-gcd}}(d_1, \ldots, d_n))$ verbessert werden.

3.8.4 Berechnung des gemeinsamen gcd einer Familie von Polynomen

Zur Berechnung des Inhalts und des primitiven Teils ist die Bestimmung des gemeinsamen gcd mehrerer Koeffizienten erforderlich. Da dies in unserem rekursiven Ansatz ebenfalls Polynome sein können, wollen wir untersuchen, wie sich der gemeinsame gcd einer Familie von Polynomen effektiv berechnen lässt.

Satz 26 *Seien $f_1, \ldots, f_N \in R[x]$ Polynome über einem Integritätsbereich R und $K/Q(R)$ eine Erweiterung des Quotientenkörpers von R. Dann gilt*

$$\gcd(f_1, \ldots, f_N) = \gcd(f_1, \alpha_2\, f_2 + \cdots + \alpha_N\, f_N)$$

für fast alle $(\alpha_2, \ldots, \alpha_n) \in K^{N-1}$.

Beweis: Sei $g = \gcd(f_1, \ldots, f_N)$. Wir haben zu untersuchen, für welche $(\alpha_2, \ldots, \alpha_n) \in K^{N-1}$ die Polynome $h_1 = f_1/g$ und $h_2 = (\alpha_2\, f_2 + \cdots + \alpha_N\, f_N)/g$ teilerfremd sind. Das ist genau dann der Fall, wenn $res(h_1, h_2) \neq 0$.

Setzen wir die Frage mit unbestimmten Koeffizienten an, so ist $res(h_1, h_2) \in R[\alpha_2, \ldots, \alpha_n]$ ein nicht identisch verschwindendes Polynom (sonst wäre g nicht der gemeinsame gcd) in diesen Unbestimmten. Konkrete Werte für die α_i müssen also nur so gewählt werden, dass dieser polynomiale Ausdruck nicht verschwindet. □

Dieser Satz ist die Basis für einen probabilistischen Algorithmus zur Bestimmung des gemeinsamen gcd einer Familie von Polynomen mit derselben Laufzeit wie der binäre gcd-Algorithmus. Ist $h = \gcd(f_1, \alpha_2\, f_2 + \cdots + \alpha_N\, f_N)$ der berechnete gcd, so kann das Ergebnis durch Teilbarkeitsuntersuchung $h \mid f_i,\ i = 2, \ldots, N$ verifiziert werden, da $g \mid h$ nach Konstruktion gilt.

3.8.5 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 $\underline{\ }: R \longrightarrow K$ und $\underline{\ }: R[x] \longrightarrow K[x]$ als *Evaluationsmorphismen*.

Evaluation eines durch eine Gradschranke $d = (d_1, \ldots, d_{n-1})$ beschränkten Polynoms $f \in R$ in einem Punkt $a \in K^{n-1}$ kann am effienzientesten mit dem Hornerschema ausgeführt 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 $S_a = (a^\alpha, \alpha < d)$ bestimmt wird und dann dieser Vektor mit dem Koeffizientenvektor von f multipliziert wird. Die Vektoren S_a müssen dabei nur einmal berechnet werden und stehen 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 $N \leq D$ Evaluationspunkten und verschiedene Polynome $f, g, h, \ldots$ handelt es sich dabei immer um dasselbe $(N \times D)$-Gleichungssystem, so

dass das eine Gleichungssystem simultan mit verschiedenen rechten Seiten gelöst werden kann. Um die Lösbarkeit zu garantieren muss die Koeffizientenmatrix S des Systems maximalen Rang haben, d.h. wenigstens einer der N-Minore von S muss verschieden Null sein. Das ist für Punkte *in allgemeiner Lage* immer erfüllt, da das Verschwinden eines N-Minors eine polynomiale Bedingung vom Grad N 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 $\overline{lc(f) \cdot lc(g)} \neq 0$
    in genügend allgemeiner Lage.

  Berechne $w_a = \gcd_{K[x]}(\overline{f}, \overline{g})$ mit $lc(w_a) = \overline{\beta}$
    mit dem Euklidschen Algorithmus.

  if (deg(w_a) < d) then {d:=deg(w_a), M:={} }
        (* alle bisherigen a waren schlechte Reduktionen *)
  else if deg(w_a) > d then continue (* a ist schlechte Reduktion *)

  M:=M $\cup \{(a, w_a)\}$

until $|M| \geq D$.

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

If $w \mid \beta f$, $w \mid \beta g$ then return $pp(w)$ 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])

$$\begin{aligned} f &= x^3\,y\,(2\,y-1) + x^2\,y^3 + x\,(2\,y^4 - y^3 - 6\,y + 3) + y^2\,(y^3-3),\\ g &= x^3\,(2\,y-1) + x^2\,y\,(-y+1) + x\,(-y^3 + 4\,y - 2) + 2\,y^2 \end{aligned}$$

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 $\overline{f}, \overline{g}$ als Skalarprodukte der Koeffizienten mit S_a: $2\,d_n\,D$.
- Berechnung von $\gcd(\overline{f}, \overline{g})$: $O(d_n^2)$.

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 verschiedenen rechten Seiten gelöst werden. Dies ist mit einem simultanen Gauß-Verfahren in $O(D^2(D+d))$ Arithmetik-Schritten möglich. Die Kosten bis hierher sind also von der Größenordnung $O(D^3 + d_n\,D^2 + d_n^2\,D)$. Der Teilbarkeitstest zur Verifikation kostet im klassischen Ansatz $O(d_n^2\,D^2)$, in schneller Arithmetik $\widetilde{O}(d_n\,D)$. Übermäßige Kosten werden vor allem durch das Gaußverfahren verursacht, das durch eine bivariate Version wie folgt abgelöst werden kann.

fastModularEvalGCD

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

Ist $u \in R[y]$ ein Polynom vom Grad $\deg(u) = n$ und dessen Wert $u(c_i) = u_i, i = 0, \ldots, n$, an $n+1$ Stellen $c_i \in R$ bekannt, so kann u wie folgt (re)konstruiert werden: Ist $h \in R[y]$ ein Polynom vom Grad $< k$ mit $h(c_i) = u(c_i), i < k$ und $p = \prod_{i<k}(y - c_i) \in R[y]$, so ist

$$h'(y) = h(y) + \frac{u_k - h(c_k)}{p(c_k)} \cdot p(y)$$

ein Polynom vom Grad $\leq k$ mit $h(c_i) = u(c_i), i \leq k$. In der Tat, wegen $p(c_i) = 0$ ist $h'(c_i) = h(c_i) = u_i$ für $i < k$. Für $i = k$ ergibt sich $h'(c_k) = u_k$ unmittelbar. Schließlich gilt $\dfrac{u_k - h(c_k)}{p(c_k)} \in R$, denn das Polynom $u - h$ hat die Nullstellen $c_1, \ldots, c_{k-1}$ und ist mithin durch $p = \prod_{i<k}(y - c_i)$ in $R[y]$ teilbar.

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;
```

Sind $f, g \in R[y = x_{n-1}][x = x_n]$ zwei 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: $f, g \in R[y][x]$ primitive Polynome
Output: $h = \gcd_{R[y][x]}(f, g)$

β:=gcd$(lc_x(f), lc_x(g)) \in R[y]$
`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` $\overline{lc(f) \cdot lc(g)} \neq 0$.
`Berechne` $w_a = \gcd_{R[x]}(\overline{f}, \overline{g})$ `mit` $lc(w_a) = \overline{\beta}$ `rekursiv.`
`if` $(\deg(w_a) < d)$ `then {d:=`deg(w_a)`, M:={} }`
`(* alle bisherigen` a `waren schlechte Reduktionen *)`
`else if` $\deg(w_a) > d$ `then continue (*` a `ist schlechte Reduktion *)`
`M:=M` $\cup \{(a, w_a)\}$
`until` $|M| = d_{n-1}$.
`Lifte die` (a, w_a) `koeffizientenweise zu` $w \in R[y][x]$.
`If` $w \mid \beta f$, $w \mid \beta g$ `then return` $pp(w)$ `else return FAILED.`

Die Kosten $C^{(n)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_n)$ dieses Algorithmus ergeben sich ohne Berücksichtigung schlechter Reduktionen aus

- d_{n-1} rekursiven gcd-Berechnungen (Kosten je $C^{(n-1)}_{\mathrm{mod-gcd}}(d_1, \ldots, d_{n-2}, d_n)$)
- und den Kosten des Liftens der maximal d_n Koeffizienten von w.

  Die Werte der Koeffizienten an den Evaluationspunkten sind durch die Gradschranke $(d_1, \ldots, d_{n-2})$ beschränkte Polynome $u_i \in R$. Der Grad der rekonstruierten Koeffizienten in $R[y]$ ist durch d_{n-1} beschränkt, so dass die Kosten von `Interpolate` jeweils der Größenordnung $O\left(d^2_{n-1} C^*_R(d_1, \ldots, d_{n-2})\right)$ sind.

Klassische Multiplikation ergibt Lift-Kosten $O\left((d_1 \cdots d_{n-1})^2 \, d_n\right)$ und somit Gesamtkosten von $O\left((d_1 \cdots d_{n-1})^2 \, d_n\right)$.

Einsatz schneller Multiplikation führt auf die Schranke $\widetilde{O}(d_1 \cdots d_n) \, d_{n-1}$ für die Lift-Kosten und somit Gesamtkosten von $\widetilde{O}(d_1 \cdots d_n) \max(d_1, \ldots, d_{n-1})$.

4 Polynom-Faktorisierung

Dieses Kapitel hält sich weitgehend an [5].

4.1 Allgemeines

Grundbegriffe

R sei ein Integritätsbereich.

Definition 8 $0 \neq p \in R$ heißt *Primelement*, wenn

$$q \,|\, p \quad \Rightarrow \quad q \sim 1 \text{ oder } q \sim p$$

gilt.
Für $0 \neq a \in R$ bezeichnet man eine Zerlegung

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

mit $0 < a_i \in \mathbb{N},\ \varepsilon \sim 1$ sowie Primelementen $p_i \in R$ als *Primfaktorzerlegung.*

Eine solche Primfaktorzerlegung wird als *eindeutig* bezeichnet, wenn sie eindeutig bis auf Reihenfolge und assoziierte Elemente ist. Genauer, ist

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

eine andere Zerlegung, so muss $r = s$ gelten und eine Permutation $\pi \in S_r$ existieren, so dass $a_i = b_{\pi(i)}$ und $p_i \sim q_{\pi(i)}$ gilt.
Primfaktorzerlegungen müssen weder existieren noch eindeutig sein. Im Allgemeinen wird noch eine subtile Unterscheidung zwischen Primelementen (im engeren Sinne) und irreduziblen Elementen getroffen, auf die hier nicht eingegangen werden soll.

Definition 9 Ein Ring R heißt *ZPE-Ring* (oder UFD), wenn in R jedes Element $a \neq 0$ eine eindeutige (in obigem Sinne) Primfaktorzerlegung besitzt.

ZPE-Ring ist eine Abkürzung für „Ring mit eindeutiger Zerlegung in Prim-Elemente“ oder (umgekehrt gelesen) „Ring mit Eindeutiger Primfaktor-Zerlegung“, was dem englischen Unique Factorization Domain entspricht. Solche Ringe werden auch als faktorielle Ringe bezeichnet.

Ist R ein ZPE-Ring, so auch $R[x]$. Der auf Gauss zurückgehende Beweis soll hier nicht geführt werden. 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 $cont(f) \in R$ und von $pp(f) \in K[x]$ mit $K = Q(R)$: Ist $pp(f) = \varepsilon_1 \prod_i f_i^{a_i}$ eine so skalierte Faktorisierung über $K[x]$, dass alle $f_i \in R[x]$ primitiv sind und $cont(f) = \varepsilon_2 \prod_j r_j^{b_j}$ die Faktorisierung in R, so ist das Produkt der beiden eine Faktorisierung von f.

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

$$a^p \equiv a \pmod p \quad \text{für alle } a \in \mathbb{Z}_p.$$

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

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

lautet.

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

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

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

Mit $(\mathbb{Z}_p[x])_d$ bezeichnen wir die Menge der Polynome $f \in \mathbb{Z}_p[x]$ mit $f = 0$ oder $\deg(f) < d$. Diese Menge ist ein Vektorraum der Dimension d über $\mathbb{Z}_p$ und enthält somit genau p^d Elemente.

4.2 Quadratfreie Faktorisierung

Sei wieder D ein ZPE-Ring.

Definition 10 Ein Polynom $f \in D$ heißt *quadratfrei*, wenn jeder Faktor in der Primzerlegung von f mit der Multiplizität 1 vorkommt.

Satz 27 *Sei K ein Körper mit $char(K) = 0$ oder $K = \mathbb{Z}_p$.*
$f \in K[x]$ ist genau dann dann quadratfrei, wenn $\gcd(f, f') \sim 1$ *gilt.*

Beweis: Enthält $f = b^2 c$ einen quadratischen Faktor, so gilt $b \mid f' = b\,(2\,b'\,c + b\,c')$.
Ist umgekehrt $d \mid \gcd(f, f')$ ein gemeinsamer Primteiler und $f = d\,c$, so gilt $d \mid f' = d'\,c + d\,c'$ und somit $d \mid d'\,c$. Als Primteiler muss d einen der beiden Faktoren teilen. Für $d \mid c$ ist f nicht quadratfrei. $d \mid d'$ 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. □
Verallgemeinert für $char(K) = 0$ auf mehrere Variablen: $f \in K[x_1, \ldots, x_n]$ ist genau dann quadratfrei, wenn $\gcd(f, \partial_1 f, \ldots \partial_n f) \sim 1$.

Definition 11 Die (eindeutig bestimmte) Zerlegung $f = \prod_i b_i^i$ mit quadratfreien b_i bezeichnet man als *quadratfreie Faktorsierung* von f.

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]
```

Als Schleifeninvarianten haben wir dabei

$$f = \prod_{i \geq k} b_i^{i-k+1}, \quad g = \gcd(f, f') = \prod_{i>k} b_i^{i-k}, \quad f/g = \prod_{i \geq k} b_i$$

woraus die Korrektheit des Verfahrens unmittelbar folgt.
Über $\mathbb{Z}_p$ muss der Fall $f' = 0$ abgefangen werden. Dies kann mit einer rekursiven Variante von `sqrfreeFactorization` umgesetzt werden.

sqrfreeFactorization(f) /* rekursive Variante */

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

```
if f' = 0 then
  Finde g mit f(x) = g(x)^p
  return sqrfreeFactorization(g)^p
g=gcd(f,f');
if deg(g) = 0 then return [f]; /* ist selbst quadratfrei */
else return sqrfreeFactorization(g)*sqrfreeFactorization(f/g)
```

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

Voraussetzung: $a(x) = a_1(x) \cdot \ldots \cdot a_r(x) \in \mathbb{Z}_p[x]$ quadratfrei und $d = \deg(a(x))$.
Wir schreiben analog zu Restklassen in $\mathbb{Z}$ für Polynome $f(x), g(x), h(x) \in \mathbb{Z}_p[x]$

$$f(x) \equiv g(x) \pmod{h(x)} \quad \Leftrightarrow \quad h(x) \,|(f(x) - g(x))$$

Wie im Fall des Chinesischen Restklassensatzes beweist man, dass für vorgegebene Polynome $s_1(x), \ldots, s_r(x) \in \mathbb{Z}_p[x]$ das System von Kongruenzen

$$\begin{aligned} s(x) &\equiv s_1(x) \pmod{a_1(x)} \\ &\ldots \\ s(x) &\equiv s_r(x) \pmod{a_r(x)} \end{aligned} \tag{$*$}$$

eine $(\mathrm{mod}\ a_1(x)\cdot\ldots\cdot a_r(x) = a(x))$ eindeutig bestimmte Lösung hat, welche durch die zusätzliche Bedingung, dass $s(x)$ bzgl. $a(x)$ reduziert sein soll, also $s(x) \in (\mathbb{Z}_p[x])_d$ gilt, auch als Polynom eindeutig bestimmt ist.

Wir konzentrieren uns auf die Lösungen 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 $a_i \,|\, b$ und $a_j \not| b$ gilt.

Sei

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

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 9 *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);
```

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

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;
```

$$1,\ x + x^2 + 2,\ 2\,x^2 + x^3 + 1$$

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 (mod 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)}
while size(F)< r do
  Wähle f(x) ∈ F, s(x) ∈ B und c ∈ Z_p.
  Bestimme g(x) = gcd(f(x), s(x) - c).
  if 0 < deg(g(x)) < deg(f(x)) then F:=F \ {f(x)} ∪ {g(x), f(x)/g(x)}.
return F.
```

Zu jedem Zeitpunkt ist $\prod_{f(x)\in F} f(x) = a(x)$ und je zwei Faktoren a_i, a_j lassen sich durch eine Wahl von $s(x) \in B$ und $c \in \mathbb{Z}_p$ trennen, so dass schließlich alle Faktoren separiert worden sind.

Wegen $r \leq d$ 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

$$s^{(2)}(x) = 2x + 3x^2 + x^3,\ s^{(3)}(x) = x^4 - 2x^2$$

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));
```

$$\left[25\,x + x^2 + 34,\ x + 12,\ x^2 + 2\right]$$

was genau die Faktorzerlegung von f ergibt.

4.4 Faktorisierung in $\mathbb{Z}[x]$

Kronecker-Faktorisierung

Siehe etwa [3, S. 242 ff]. In diesem Kurs nicht behandelt.

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 $p \not| \operatorname{res}(a, a')$ 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 10 (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 $f^{(k)}(x), g^{(k)}(x) \in \mathbb{Z}[x]$, für die

$$\begin{aligned} f^{(k+1)}(x) &\equiv f^{(k)}(x) \pmod{p^k}, \\ g^{(k+1)}(x) &\equiv g^{(k)}(x) \pmod{p^k} \end{aligned}$$

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

$$a(x) - f^{(k)}(x)\, g^{(k)}(x) \equiv \left(f^{(k)}(x)\, V(x) + g^{(k)}(x)\, U(x)\right) p^k \pmod{p^{k+1}}$$

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

$$f(x)\, s(x) + g(x)\, t(x) \equiv 1 \pmod{p}$$

die entsprechenden Liftungen

$$f^{(k+1)}(x) = f^{(k)}(x) + \Big(a^{(k)}(x)\,t(x) \pmod{f(x)}\Big)\cdot p^k,$$
$$g^{(k+1)}(x) = g^{(k)}(x) + \Big(a^{(k)}(x)\,s(x) \pmod{g(x)}\Big)\cdot p^k$$

wobei die letzte Reduktion erreicht, dass $\deg(f^{(k+1)}) = \deg(f),\ \deg(g^{(k+1)}) = \deg(g)$ gilt. □

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 Bezoutschranke` B `für` $a(x)$.
`Wähle Primzahl` p `mit` $p \not| \operatorname{res}_x(a, a')$.

```
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 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. Für große r ergibt sich eine kombinatorische Explosion der Anzahl der zu untersuchenden Partitionen, die im schlechtesten Fall die Größenordnung 2^r hat. 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.