Computer Algebra

Skriptum zur Vorlesung
von

Jacques Calmet

Institut für Algorithmen und Kognitive Systeme
Universität Karlsruhe

Das Skript wurde von Winfried Fakler übersetzt. Gegenüber der Version SS91 wurden zwei neue Kapitel, eines über mathematische Grundlagen und eines über formale Potenzreihen hinzugefügt und einige kleinere Umstellungen vorgenommen.

Viele haben sich durch Fehlerkorrektur und konstruktive Kritik an der Überarbeitung beteiligt: Winfried Fakler, Bernhard Garz, Karsten Homann, Martin Kreidler, Markus Michels, Holger Petersen, Werner Seiler, Indra Adiono Tjandra, Christoph Zenger u.v.a.

Eine Garantie auf Fehlerfreiheit bezüglich Rechtschreibung und Inhalt wird nicht übernommen. Für die Meldung verbliebener Fehler sind wir dankbar.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Computer-Algebra

Computer-Algebra steht für die symbolische (fehlerfreie) Manipulation mathematischer Objekte.

1. Die Objekte müssen dargestellt werden.

$$\frac{x^2-5}{x(x-1)^4} = \frac{-5}{x} + \frac{5}{x-1} + \frac{-5}{(x-1)^2} + \frac{6}{(x-1)^3} - \frac{4}{(x-1)^4}$$

$$\sqrt{9+4\sqrt{2}} = 1 + 2\sqrt{2}$$

   Hier ist das Hauptproblem die Bestimmung einer „adäquaten“ Darstellung und deren Eindeutigkeit.

2. Algorithmen, die über diesen mathematischen Bereichen untersucht werden.

$$\begin{array}{rcll} x^4+1 & = & (x+1)^4 & \text{über } GF(2) \\ x^4+1 & = & (x^-2)(x^+2) & \text{über } GF(5) \\ x^4+1 & = & (x^2-2x+2)(x^2+2x+2) & \text{über } GF(3) \end{array}$$

$$\int \frac{1}{1-x^4} dx = \frac{\log(x+1)}{4} + \frac{\arctan(x)}{2} - \frac{\log(1-x)}{4}$$

1.2 Computer-Algebra-Systeme

Ein Computer-Algebra-System besteht in der Regel aus

- Programmiersprache (immer)
- Sammlung von algebraischen Algorithmen (immer)
- Typsystem, „coercion“, überladen (optional)

Seit 1960 wurden verschiedene Computer-Algebra-Systeme entwickelt. An unserem Institut wird regelmäßig ein Praktikum angeboten, in dessen Rahmen einige dieser Systeme eingeführt werden.

Hier ein paar Bemerkungen zu einigen CA-Systemen: Eine ausführliche Liste findet sich in [AMS93]. Dort kann man auch nachlesen, wie welche Systeme zu bekommen sind (Viele sind Public Domain).

1. Allgemeine CA-Systeme

   Eines der ersten CA-Systeme war REDUCE, LISP-basiert und 1968 in Stanford entwickelt, legte es als erstes System wert auf Portabilität. Daraus resultiert seine große Verbreitung. Heute entspricht es vielleicht FORTRAN unter den Programmiersprachen. Als neuere kommerzielle Systeme sind Mathematica, das vor allem durch seine Graphikmöglichkeiten hervorragt und Maple bekannt. Ein modernes System ist AXIOM (aus SCRATCHPAD hervorgegangen), das als einziges System benutzerdefinierte algebraische Strukturen erlaubt und ein Typsystem hat. MACSYMA ist ein großes System, das am M.I.T. entwickelt wurde und die Entwicklung der CA-Systeme maßgeblich beeinflußte. SAC-2 ist durch seine umfangreiche und gut dokumentierte Algorithmensammlung bekannt. Viele dieser Algorithmen wurden für andere Systeme implementiert. PARSAC ist eine Weiterentwicklung für Parallelität. DERIVE ist kleines, für PCs geschriebenes CA-System, das trotz dieser Ausrichtung ein breites Spektrum an Möglichkeiten bietet.

2. Spezielle CA-Systeme

   Viele CA-Systeme wurden auch für spezielle Anwendungen konzipiert. Als Beispiele sollen hier CAYLEY und GAP für Gruppentheorie, LiE für Lie-Algebra, MACAULY für algebraische Geometrie, PARI für Zahlentheorie, LIE, DELiA für Differentialgleichungen sowie SHEEP und STENSOR für Tensorrechnung genannt werden.

1.3 Möglichkeiten von CA-Systemen

Die folgende Faktorisierung wurde von einer Cray Y-MP4/464 in drei Wochen berechnet:

$$\begin{aligned}
2^{463}+1 &= 2381705131778446589520242536874132581700120107002038199303870846751188\\
&\quad 1928998231515526283497886045162950663079941301185260618261664450478 09\\
&= 3 * 2356759188941953 * 768349662098580495261 07 * 88
\end{aligned}$$

Für die folgende Faktorisierung wurden weltweit 600 Workstations eingesetzt, die Berechnung bräuchte auf einem 1-MIPS-Rechner etwa 400 Jahre.

$$\begin{aligned}
10^{142}+1 &= 101 * 569 * 7669 * 3806238488714809 *\\
&\quad 771692651883350877868950850 4941 *\\
&\quad 9361138228751395032943162581149 0669 *\\
&\quad 825198826590619667087624834867194466392884304460 81
\end{aligned}$$

Auch das folgende Integral läßt sich auf CA-Systemen lösen:

$$\begin{aligned}
&\int \frac{2x^6+4x^5+7x^4-3x^3-x^2-8x-8}{(2x^2-1)^2\sqrt{x^4+4x^3+2x^2+1}}dx =\\
= \quad &\frac{\sqrt{x^4+4x^3+2x^2+1}(2x+1)x}{2(2x^2-1)} - \ln(x^2+4x+2)\\
&+\frac{1}{2}\ln(\sqrt{x^4+4x^3+2x^2+1}(\sqrt{2}+4)-2\sqrt{2}x-4\sqrt{2}-x-2)\\
&+\frac{1}{2}((25\sqrt{2}x+6\sqrt{2}+44x+38)\sqrt{x^4+4x^3+2x^2+1}\\
&+\sqrt{2}(-7x^3-36x^2-21x-10)-28x^3-130x^2-84x+2)\\
&+\frac{5}{2}\ln((16\sqrt{2}+13)\sqrt{x^4+4x^3+2x^2+1}-28\sqrt{2}x^2-32\sqrt{2}x+6\sqrt{2}-35x^2-26x+11)\\
&+\ln((2\sqrt{2}+1)\sqrt{x^4+4x^3+2x^2+1}-4\sqrt{2}x^2-16\sqrt{2}x-10\sqrt{2}+5x^2+20x+9)\\
&-\ln(2\sqrt{2}x+4\sqrt{2}-x^2-4x-6)-\frac{\ln(\sqrt{2}+2x)}{2}-\frac{5}{2}\ln(2x^2-1)
\end{aligned}$$

Im folgenden noch eine Beispielsitzung auf dem CA-System AXIOM:

```
factor 889432583716104169843O7

   (1)   65239 351053 3883587112921
                                                          Type: Factored Integer


m1:SquareMatrix(3,Integer):=[[1,2,1],[2,1,-2],[1,-2,4]]

         +1   2    1 +
         |           |
   (2)   |2   1   - 2|
         |           |
         +1  - 2   4 +
                                                    Type: SquareMatrix(3,Integer)

eigenvectors(m1)

   (3)
                                   + 0 +
                                   |   |
                                   |  1|
   [[eigval= 5,eigmult= 1,eigvec= [|- -|]],
                                   |  2|
                                   |   |
                                   + 1 +

                                              +%Q+
               2                              |  |
    [algrel= %Q  - %Q - 5,algmult= 1,algvec= [|2 |]]]
                                              |  |
                                              +1 +

Type: List Union(Record(algrel: Fraction Polynomial Fraction Integer,
algmult: Integer,algvec: List Matrix Fraction Polynomial
Fraction Integer), Record(eigval: Fraction Polynomial Fraction
Integer,eigmult: Integer, eigvec: List Matrix Fraction Polynomial
Fraction Integer))
```

```
m2:SquareMatrix(2,SquareMatrix(3,Integer)):=[[2,m1],[m1,3]]

          +   +2  0  0+     +1   2    1 ++
          |   |       |     |           ||
          |   |0  2  0|     |2   1   - 2||
          |   |       |     |           ||
          |   +0  0  2+     +1  - 2   4 +|
   (4)    |                              |
          |+1   2    1 +     +3  0  0+   |
          ||           |     |       |   |
          ||2   1   - 2|     |0  3  0|   |
          ||           |     |       |   |
          ++1  - 2   4 +     +0  0  3+   +
                       Type: SquareMatrix(2,SquareMatrix(3,Integer))

m3:=m2**2

          + +10   2    1 +   +5    10    5  ++
          | |            |   |              ||
          | |2   13   - 8|   |10   5    - 10||
          | |            |   |              ||
          | +1   - 8  25 +   +5   - 10   20 +|
   (5)    |                                  |
          |+5    10    5  +   +15   2    1 + |
          ||              |   |            | |
          ||10   5    - 10|   |2   18   - 8| |
          ||              |   |            | |
          ++5   - 10   20 +   +1   - 8  30 + +
                       Type: SquareMatrix(2,SquareMatrix(3,Integer))

trace(m3)

          +25   4     2  +
          |              |
   (6)    |4    31   - 16|
          |              |
          +2   - 16   55 +
                                      Type: SquareMatrix(3,Integer)
```

```
sum(k**3,k=1..n)


          4     3    2
         n  + 2n  + n
   (7)   -------------
               4
                                          Type: Fraction Polynomial Integer

sum(k**2,k=a..b)


           3     2         3     2
         2b  + 3b  + b - 2a  + 3a  - a
   (8)   -----------------------------
                       6
                                          Type: Fraction Polynomial Integer

limit(sum(1/(k*(k+2)),k=1..n),n=%plusInfinity)

         3
   (9)   -
         4

Type: Union(OrderedCompletion Fraction Polynomial Integer, Record(
leftHandLimit: Union(OrderedCompletion Fraction Polynomial Integer,
"failed"), rightHandLimit: Union(OrderedCompletion Fraction
Polynomial Integer,"failed")), "failed")

p:=x**9-x**4-x-1

          9    4
   (10)  x  - x  - x - 1
                                                   Type: Polynomial Integer

factor p

           4       5
   (11)  (x  + 1)(x  - x - 1)
                                          Type: Factored Polynomial Integer
```

```
pol:=x**4+2*x**3 -7*x**2 -8*x+31

          4     3     2
   (12)  x  + 2x  - 7x  - 8x + 31
                                                  Type: Polynomial Integer

factor pol

          4     3     2
   (13)  x  + 2x  - 7x  - 8x + 31
                                         Type: Factored Polynomial Integer

factor(pol::UnivariatePolynomial(x, AlgebraicNumber), [rootOf('a**2-5)])

           2                          2
   (14)  (x  + (- 2a + 1)x - a + 6)(x  + (2a + 1)x + a + 6)
                     Type: Factored UnivariatePolynomial(x,AlgebraicNumber)

factor(pol::UnivariatePolynomial(x, AlgebraicNumber),
[rootOf('a**2-5), rootOf('b**2+b+1)])

   (15)  (x - b - a)(x - b + a)(x + b - a + 1)(x + b + a + 1)
                     Type: Factored UnivariatePolynomial(x,AlgebraicNumber)

integrate((x+1)/(x*(x+log(x))**(3/2)),x)

             +----------+
           2\|log(x) + x
   (16)  - --------------
             log(x) + x
                    Type: Union(Expression Integer,List Expression Integer)

integrate(sinh(x)**4,x)

                      3            3
         cosh(x)sinh(x)  + (cosh(x)  - 4cosh(x))sinh(x) + 3x
   (17)  ---------------------------------------------------
                                  8
                    Type: Union(Expression Integer,List Expression Integer)
```

```
integrate(x/(3*x**6-10*x**5 -9*x**4+40*x**3-27*x**2+50*x-15),x)

   (18)
             +-+
          50\|5
       *
          log
                                                     2
                   34204427984397103365x  - 152966855668794828766x
                 +
                   171022139921985516825
              *
                  +-+
                 \|5
            +
                                                   2
              - 76483427834397414383x  + 342044279843971033650x
            +
              - 38241713917198707191 5
      +
              +-+           2
        (- 50\|5  - 90)log(x  - 5) + 81log(3x - 1) + 99log(x - 3)
      +
                8525x - 6774
        176atan(------------)
                6774x + 8525
   /
      10560
                   Type: Union(Expression Integer,List Expression Integer)

)set output tex on

differentiate(x*5*sin(x)**3/(x**2-1),x)

$$
{{{{\left( -{5 \  {x \sp 2}} -5 \right)} \  {{\sin \left( {x} \right)} \sp
3}}+{{\left( {{15} \  {x \sp 3}} -{{15} \  x} \right)} \  {\cos \left( {x}
\right)} \  {{\sin \left( {x} \right)} \sp 2}}} \over {{x \sp 4} -{2 \  {x
\sp 2}}+1}}
\leqno(19)
$$
```

```
                2           3       3                   2
         (- 5x  - 5)sin(x)  + (15x  - 15x)cos(x)sin(x)
   (19)  ----------------------------------------------
                             4     2
                            x  - 2x  + 1
                                            Type: Expression Integer
```

```
)set output tex off
)set output fortran on

differentiate(x*5*sin(x)**3/(x**2-1),x)

      T1=SIN(x)
      T2=x*x
      R20=((-5.*T2-5.)*T1**3+(15.*x**3-15.*x)*COS(x)*T1*T1)/(x**4-2.*T2+
     &1.)
```

```
                2           3       3                   2
         (- 5x  - 5)sin(x)  + (15x  - 15x)cos(x)sin(x)
   (20)  ----------------------------------------------
                             4     2
                            x  - 2x  + 1
                                            Type: Expression Integer
```

```
)set output fortran off
```

Kapitel 2

Normalformen und Datenstrukturen

2.1 Abstraktionsebenen

Es werden drei Abstraktionsebenen unterschieden:

(i) Die *Objekt*-Ebene ist die Abstraktionsebene, in der die Elemente eines Bereiches als einfache (primitive) Objekte angesehen werden. Eine bessere Bezeichnung wäre mathematische Ebene.

(ii) Die *Form*-Ebene ist die Abstraktionsebene, in der man sich damit beschäftigt, wie man Objekte in Termen aus einigen gewählten 'Grundsymbolen' darstellt, auch unter Beachtung, daß ein spezielles Objekt viele verschiedene gültige Darstellungen aus Termen der gewählten Symbole haben kann. In dieser Stufe wird beispielsweise zwischen den folgenden Darstellungen des gleichen bivariaten Polynoms aus dem Integritätsbereich $\mathbb{Z}[x, y]$ unterschieden:

$$a(x,y) = 12x^2y - 4xy + 9x - 3, \tag{2.1}$$
$$a(x,y) = (3x-1)(4xy+3), \tag{2.2}$$
$$a(x,y) = (12y)x^2 + (-4y+9)x - 3. \tag{2.3}$$

(iii) In der *Datenstruktur*-Ebene befaßt man sich mit der Speicherorganisation, die benutzt wird, ein Objekt in einer speziellen Form darzustellen. Das Polynom $a(x, y)$ in der Form (2.1) kann z.B. als eine verkettete Liste aus vier Elementen (eines für jeden Term) dargestellt werden, oder man kann $a(x, y)$ durch ein Array der Länge 6, das die ganzen Zahlen 12, 0, -4, 9, 0, -3 enthält als die Koeffizienten eines bivariaten Polynoms darstellen, was einen Vektor der Exponenten (2, 1), (2, 0), (1, 1), (1, 0), (0, 1), (0, 0) in dieser Reihenfolge impliziert, oder aber durch andere Datenstrukturen.

2.2 Hauptprobleme

2.2.1 Problem der Vereinfachung

Dieses Problem wird im Kapitel 9 vorgestellt.

2.2.2 Beliebig große Zwischenergebnisse

Will man beispielsweise prüfen, ob die beiden Polynome

$$\begin{aligned} A(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5 \\ B(x) &= 3x^6 + 5x^4 - 4x^2 - 9x + 21 \end{aligned}$$

teilerfremd sind, kann man dies wie im Beispiel aus Abschnitt 5.3.1 durch eine ggT-Berechnung mit dem euklidischen Algorithmus durchführen. Dabei sind die Daten zunächst sehr klein, weil 21 die größte auftretende ganze Zahl ist. Es zeigt sich aber, daß die letzte berechnete ganze Zahl 35 Ziffern hat. Da dieser letzte Schritt eine ganze Zahl zum Ergebnis hat, sind die beiden Polynome teilerfremd. Zu dieser Antwort (ja/nein Antwort) genügt 1 Bit, aber für die Zwischenergebnisse werden 117 Bits notwendig.
Diese und ähnliche derartige Tatsachen sind, wenn immer Zwischenergebnisse beliebig groß werden können als das „Anschwellen der Zwischenergebnisse“ (intermediate expression swell) bekannt. Dies ist eines der größten Probleme in der Computer Algebra.

2.2.3 Normale und kanonische Formen

2.2.3.1 Definition von ‘einheitsnormal’

Im folgenden sei D ein Integritätsbereich.

Man nennt $u \in D$ eine Einheit (oder invertierbar), falls u ein multiplikativ Inverses in D besitzt. Zwei Elemente a und b von D heißen assoziiert, wenn es eine Einheit

$u \in D^*$ gibt mit $a = ub$. Sind a und b assoziiert, so schreibt man $a \sim b$, andernfalls $a \not\sim b$. Daraus ergibt sich unmittelbar für alle $a, b \in D$

$$a \sim b \iff (a) = (b) \iff a|b \text{ und } b|a.$$

Die Relation 'assoziiert' bildet eine Äquivalenzrelation auf D und teilt daher den Integritätsbereich D in assoziierte Klassen ein. Für ein spezielles D muß ein Kriterium gefunden werden, um ein Element aus jeder assoziierten Klasse, als seinen kanonischen Repräsentanten auswählen zu können. Diese Elemente werden dann als *einheitsnormal* (unit normal) definiert.

Beispiele:

1. In $\mathbb{Z}$ werden die nichtnegativen ganzen Zahlen als einheitsnormal definiert. Hier bilden $\{0\}$, $\{1, -1\}$, $\{2, -2\}$, ... die assoziierten Klassen.

2. Sei F ein Körper. Jedes $a \in F$, $a \neq 0$ ist in F eine Einheit. Daher werden hier 0 und 1 als einheitsnormal definiert.

Für jeden Integritätsbereich gilt:

1. 0 ist einheitsnormal.

2. 1 ist das einheitsnormale Element für die assoziierte Klasse der Einheiten.

3. Sind $a, b \in D$ einheitsnormale Elemente, dann ist auch ab einheitsnormal in D.

4. Einheitsnormale Elemente werden definiert. Zum Beispiel

   Definition 2.1
   In einem beliebigen Polynomring $D[x]$ über D heißen Polynome mit einheitsnormalem Leitkoeffizient einheitsnormal.

   Definition 2.2
   In einem beliebigen Potenzreihenring $F[[x]]$ über einem Körper F heißen die Monome x^l $(l \geq 0)$ und die Null Potenzreihe 0 einheitsnormal.

Bemerkung:
Das Konzept 'einheitsnormaler Elemente' wird benötigt, um die Eindeutigkeit der gcd's sicherzustellen.

2.2.3.2 Definitionen für normale und kanonische Formen

Definition 2.3
Sei E eine Menge von Ausdrücken und $\sim$ eine Äquivalenzrelation auf E. Eine berechenbare Funktion $f: E \longrightarrow E$ heißt **Normalfunktion** *für $[E, \sim]$, wenn sie die folgenden beiden Eigenschaften*

(i) $f(a) \sim a$ *für alle* $a \in E$

(ii) $a \sim 0 \Longrightarrow f(a) \equiv f(0)$ *für alle* $a \in E$

erfüllt.

Definition 2.4
Sei $[E, \sim]$ wie oben. Eine **kanonische Funktion** *für $[E, \sim]$ ist eine normale Funktion $f: E \longrightarrow E$, die zusätzlich die Eigenschaft*

(iii) $a \sim b \Longrightarrow f(a) \equiv f(b)$ *für alle* $a, b \in E$

erfüllt.

Definition 2.5
Ist f eine Normalfunktion für $[E, \sim]$, dann wird ein Ausdruck $\tilde{a} \in E$ eine **Normalform** *genannt, falls $f(\tilde{a}) \equiv \tilde{a}$ gilt. Ist f eine kanonische Funktion für $[E, \sim]$, dann wird ein Ausdruck $\tilde{a} \in E$ eine* **kanonische Form** *genannt, falls $f(\tilde{a}) \equiv \tilde{a}$ gilt.*

Beispiel:
Sei E der Ring $\mathbb{Z}[x]$ der univariaten Polynome über den ganzen Zahlen. Die Normalfunktionen f_1 und f_2 seien wie folgt spezifiziert:

f_1: (i) multipliziere alle Produkte von Polynomen aus
(ii) sammle alle Terme gleichen Grades

f_2: (i) multipliziere alle Produkte von Polynomen aus
(ii) sammle alle Terme gleichen Grades
(iii) ordne die Terme neu in absteigender Reihenfolge ihrer Grade.

Dann ist f_1 eine normale Funktion, die nicht kanonisch ist und f_2 eine kanonische Funktion. Eine Normalform für Polynome in $\mathbb{Z}[x]$ bezüglich f_1 ist

$$a_1 x^{e_1} + a_2 x^{e_2} + \ldots + a_m x^{e_m} \quad \text{mit } e_i \neq e_j \text{ für } i \neq j.$$

Eine kanonische Form für Polynome in $\mathbb{Z}[x]$ bezüglich f_2 ist

$$a_1 x^{e_1} + a_2 x^{e_2} + \ldots + a_m x^{e_m} \quad \text{mit } e_i < e_j \text{ für } i > j.$$

Es ist offensichtlich, daß in einer Klasse E von Ausdrücken, in der eine Normalfunktion definiert ist, das Problem des Null-Vergleichs gelöst ist. Es gibt aber keine eindeutige Normalform für eine bestimmte Äquivalenzklasse in $E/\sim$, es sei denn die Äquivalenzklasse enthält 0. Eine kanonische Form dagegen liefert eine eindeutige Darstellung für jede Äquivalenzklasse in $E/\sim$, wie der folgende Satz zeigt.

Satz 2.6
Ist f eine kanonische Funktion für $[E, \sim]$, dann gelten die folgenden Eigenschaften:

(i) f ist idempotent (d.h. $f \circ f \equiv f$, wobei $\circ$ die Verkettung von Funktionen bedeutet)

(ii) $f(a) \equiv f(b)$ genau dann, wenn $a \sim b$

(iii) in jeder Äquivalenzklasse von $E/\sim$ existiert eine eindeutige kanonische Form.

Beweis:

(i) $f(a) \sim a$ für alle $a \in E$ nach Definition 2.1 (i)
$\Longrightarrow f(f(a)) \equiv f(a)$ für alle $a \in E$ nach Definition 2.4 (iii).

(ii) '$\Leftarrow$': Dies gilt nach Definition 2.4 (iii).
'$\Rightarrow$': Sei $f(a) \equiv f(b)$. Dann gilt

$$\begin{aligned} & a \sim f(a) \equiv f(b) \sim b \quad \text{nach Definition 2.1 (i)} \\ & \Longrightarrow a \sim b. \end{aligned}$$

(iii) 'Existenz': Sei a ein beliebiges Element einer beliebigen Äquivalenzklasse. Man definiert $\tilde{a} \equiv f(a)$. Dann ist

$$\begin{aligned} f(\tilde{a}) &\equiv f(f(a)) \\ &\equiv f(a) \quad \text{durch Idempotenz} \\ &\equiv \tilde{a}. \end{aligned}$$

'Eindeutigkeit': Angenommen, es seien $\tilde{a}_1$ und $\tilde{a}_2$ zwei kanonische Formen in der gleichen Äquivalenzklasse von $E/\sim$. Dann gilt

$$\begin{aligned} & \tilde{a}_1 \sim \tilde{a}_2 \\ & \Longrightarrow f(\tilde{a}_1) \equiv f(\tilde{a}_2) \quad \text{nach Definition 2.4 (iii)} \\ & \Longrightarrow \tilde{a}_1 \equiv \tilde{a}_2 \quad \text{nach Definition einer kanonischen Form.} \end{aligned}$$

□

2.3 Normalformen für Polynome

2.3.1 Darstellung multivariater Polynome

Eine erste Wahl, die man treffen muß, ist die Entscheidung zwischen einer rekursiven und distributiven Darstellung für ein $a(x_1, x_2, \ldots, x_\nu) \in D[x_1, \ldots, x_\nu]$.

- Rekursive Darstellung:

  $$a(x_1, \ldots, x_\nu) = \sum_{i=0}^{\deg_{x_1}(a(x_1,\ldots,x_\nu))} a_i(x_2, \ldots, x_\nu)x_1^i,$$

  d.h. $a(x_1, x_2, \ldots, x_\nu) \in D[x_2, \ldots, x_\nu][x_1]$, wobei die a_i rekursiv dargestellt sind in $D[x_3, \ldots, x_\nu][x_2] \ldots$,[1] beispielsweise

  $$a(x, y, z) = (3y^2 + (-2z^3)y + 5z^2)x^2 + (4)x + ((-6z+1)y^3 + 3y^2 + (z^4+1)). \quad (2.4)$$

- Distributive Darstellung:

  $$a = \sum_{s \in \mathbb{N}^\nu} a_s x^s$$

  mit $a_s \in D$. Also z.B.

  $$a(x, y, z) = 3x^2y^2 - 2x^2yz^3 + 5x^2z^2 + 4x - 6y^3z + y^3 + 3y^2 + z^4 + 1. \quad (2.5)$$

Eine andere zu treffende Entscheidung ist die Frage nach einer dünn bzw. dicht besetzten Darstellung. Diese Entscheidung hat damit zu tun, ob man Terme, die Null als Koeffizienten haben, explizit darstellt oder nicht. In der *dünn besetzten Darstellung* werden nur die tatsächlichen Terme (also Terme mit von Null verschiedenen Koeffizienten) dargestellt, während bei einer *dicht besetzten Darstellung* alle möglichen Terme eines Polynom mit festgelegtem Grad auftreten (egal, ob diese auch möglicherweise Null zum Koeffizienten haben). Eine natürliche Repräsentation univariater Polynome $\sum_{i=0}^n a_i x^i \in \mathbb{Z}[x]$ mit einem festgelegten maximalen Grad n unter Benutzung von Arrays wäre z.B. ein $(n+1)$-Array $(a_0, \ldots, a_n)$, was eine dicht besetzte Darstellung ist, da die Koeffizienten mit Null explizit gespeichert sind. Während es leicht möglich ist, dieses Beispiel auf eine entsprechende dicht besetzte Darstellung für multivariate Polynome (durch Einführen einer lexikographischen Ordnung auf dem Vektor für die Exponenten) zu verallgemeinern, hat sich solch eine Darstellung als äußerst unpraktisch erwiesen. Bringt eine Berechnung die Handhabung von Polynomen in ν Unbestimmten

[1] Dies ist möglich, weil $D[x_1, \ldots, x_\nu]$ isomorph ist zu $(\cdots(D[x_1])\cdots)[x_\nu]$.

mit maximalem Grad d für jede Unbestimmte mit sich, dann ist die Zahl der Koeffizienten, die gespeichert werden müssen, für jedes solche Polynom $(d+1)^\nu$. Man braucht kein ungewöhnlich großes Problem um beispielsweise $\nu = 5$ und $d = 15$ zu bekommen, wobei man dann für jedes Polynom mehr als eine Million Koeffizienten speichern muß. Wären bei einem praktischen Problem mit $\nu = 5$ und $d = 15$ nicht die meisten der über eine Million Koeffizienten Null, würden Berechnungen auf diesen Polynomen die Kapazität der heutigen Rechner übersteigen. (Und wer möchte sich schon mit Ausdrücken beschäftigen, die mehr als eine Million Terme enthalten!) Daher benutzen alle wichtigen Systeme zur symbolischen Berechnung die dünn besetzte Darstellung für multivariate Polynome.

Eine ähnliche Wahl muß dafür gemacht werden, ob man *Nullexponenten* speichert oder nicht. Schreibt man selbst Polynome wie (2.4) und (2.5) auf, so werden Monome mit Nullexponenten nicht explizit mit aufgenommen (man benutzt also auf natürliche Weise die dünn besetzte Darstellung für Polynome). Der Speicherplatzverlust, den man sich bei einer Wahl für die Abspeicherung der Nullexponenten einhandelt ist nicht übermäßig, während solch eine Wahl zu effizienteren Algorithmen zur Handhabung von Polynomen beitragen kann. Man findet sowohl Systeme, in denen die Nullexponenten gespeichert werden, als auch welche, in denen dies nicht der Fall ist.

Die verschiedenen Abstraktionsebenen zur Darstellung multivariater Polynome ergeben sich zu

$$
\begin{array}{c}
\text{Mathematische Ebene} \\
\Downarrow \\
\text{Form Ebene A: normale / kanonische Formen} \\
\Downarrow \\
\text{Form Ebene B: rekursive / distributive Darstellung} \\
\Downarrow \\
\text{Form Ebene C: dünn / dicht besetzte Darstellung} \\
\Downarrow \\
\text{Form Ebene D: Nullexponent Darstellung} \\
\Downarrow \\
\text{Datenstrukturebene.}
\end{array}
$$

2.3.2 Definitionen für Normalformen

Definition 2.7
Eine **erweiterte Normalform** *für Polynomausdrücke in einem Integritätsbereich* $D[x_1, \ldots, x_\nu]$ *kann durch die Normalfunktion*

f_1: *(i) multipliziere alle Produkte von Polynomen aus*
(ii) sammle alle Terme gleichen Grades

bestimmt werden.
Eine **erweiterte kanonische Form** *für Polynomausdrücke in einem Integritätsbereich* $D[x_1, \ldots, x_\nu]$ *kann durch die kanonische Funktion*

f_2: *wende erst* f_1 *an, dann*
(iii) ordne die Terme neu in absteigender Reihenfolge ihrer Grade.

bestimmt werden.

Definition 2.8
Eine **faktorisierte Normalform** *für Polynomausdrücke in einem Integritätsbereich* $D[x_1, \ldots, x_\nu]$ *kann durch die Normalfunktion*

f_3: *ist der Ausdruck in Produktform* $\prod_{i=1}^{k} p_i$, $p_i \in D[x_1, \ldots, x_\nu]$ *für* $i = 1, 2, \ldots, k$, *wobei kein* p_i *selbst in Produktform ist, dann ersetze den Ausdruck durch* $\prod_{i=1}^{k} f_2(p_i)$, *wobei* f_2 *die kanonische Funktion der Definition 2.7 ist und letzteres Produkt als Null aufgefaßt wird, falls einer der Faktoren Null ist.*

bestimmt werden.
Eine **faktorisierte kanonische Form** *für Polynomausdrücke in einem Integritätsbereich* $D[x_1, \ldots, x_\nu]$ *(es wird angenommen, daß D die ZPE-Eigenschaft besitzt) kann durch die kanonische Funktion*

f_4: *wende erst* f_3 *an; ist das Ergebnis von Null verschieden, dann bringe jedes* $f_2(p_i)$ *auf seine einheitsnormale Faktorisierung und sammle die Faktoren, um die einheitsnormale Faktorisierung des gesamten Ausdruckes zu erhalten (d.h. Eindeutigkeit durch Einführen einer vorbestimmten Ordnung auf den Faktoren erzwingen).*

bestimmt werden.

Beispiel:
Sei $a(x,y) \in \mathbb{Z}[x,y]$ der Ausdruck

$$a(x,y) = ((x^2 - xy + x) + (x^2+3)(x-y+1))((y^3 - 3y^2 - 9y - 5) + x^4(y^2+2y+1)).$$

Benutzt man die distributive Darstellung für Polynome, erhält man eine (mögliche) erweiterte Normalform durch Anwenden von f_1 aus Definition 2.7 auf $a(x,y)$ (diese ist abhängig von der Ordnung, in der der Multiplikationsalgorithmus die Terme erzeugt)

$$\begin{aligned} f_1(a(x,y)) &= 5x^2y^3 + 3x^2y^2 - 13x^2y - 10x^2 + 3x^6y + 2x^6 - xy^4 + 7xy^3 - 31xy \\ &\quad -x^5y^3 + 2x^5y^2 + 7x^5y - 20x + 4x^5 + x^3y^3 - 3x^3y^2 - 9x^3y - 5x^3 \\ &\quad +x^7y^2 + 2x^7y + x^7 - x^2y^4 - x^6y^3 - 3y^4 + 12y^3 + 18y^2 - 12y \\ &\quad -3x^4y^3 + 3x^4y - 15 + 3x^4 - 3x^4y^2 - 3xy^2. \end{aligned}$$

Die erweiterte kanonische Form erhält man durch Anwenden von f_2 aus Definition 2.7 auf $a(x,y)$

$$\begin{aligned} f_2(a(x,y)) &= x^7y^2 + 2x^7y + x^7 - x^6y^3 + 3x^6y + 2x^6 - x^5y^3 + 2x^5y^2 + 7x^5y + 4x^5 \\ &\quad -3x^4y^3 - 3x^4y^2 + 3x^4y + 3x^4 + x^3y^3 - 3x^3y^2 - 9x^3y - 5x^3 - x^2y^4 \\ &\quad +5x^2y^3 + 3x^2y^2 - 13x^2y - 10x^2 - xy^4 + 7xy^3 - 3xy^2 - 31xy - 20x \\ &\quad -3y^4 + 12y^3 + 18y^2 - 12y - 15. \end{aligned}$$

Das Anwenden von f_3 bzw. f_4 aus Definition 2.8 auf $a(x,y)$ führt auf die faktorisierte Normalform

$$f_3(a(x,y)) = (x^3 - x^2y + 2x^2 - xy + 4x - 3y + 3)(x^4y^2 + 2x^4y + x^4 + y^3 - 3y^2 - 9y - 5)$$

und die faktorisierte kanonische Form

$$f_4(a(x,y)) = (x-y+1)(x^2+x+3)(x^4+y-5)(y+1)^2.$$

2.4 Normalformen für rationale Funktionen und Potenzreihen

2.4.1 Rationale Funktionen

Existieren größte gemeinsame Teiler im Integritätsbereich D und wurde eine kanonische Form (d.h. eine eindeutige Darstellung) für die Elemente von D bestimmt, dann ergeben sich die folgenden Gemeinsamkeiten bei der Definition einer kanonischen Form für Elemente des Quotientenkörpers $Q(D)$: der Repräsentant $\frac{a}{b}$ von $[\frac{a}{b}] \in Q(D)$ ist kanonisch, falls

- $\gcd(a, b) = 1$
- b ist einheitsnormal in D (2.6)
- a und b sind kanonisch in D.

Definition 2.9
Eine **erweiterte kanonische Form** *für rationale Ausdrücke in einem Körper $D(x_1, \ldots, x_\nu)$ kann durch die kanonische Funktion*

f_5: *(i) [bilde den Hauptnenner] bringe den Ausdruck auf die Form $\frac{a}{b}$, wobei $a, b \in D[x_1, \ldots, x_\nu]$ ist, durch Ausführen der arithmetischen Operationen gemäß den Gleichungen*

$$\begin{aligned} \left(\frac{a}{b}\right) + \left(\frac{c}{d}\right) &= \frac{ad + bc}{bd} \\ \left(\frac{a}{b}\right)\left(\frac{c}{d}\right) &= \frac{ac}{bd}. \end{aligned}$$

(ii) [erfülle die gcd*-Bedingung aus (2.6), kürze den* gcd*] berechne $g = \gcd(a, b)$ und ersetze $\frac{a}{b}$ durch $\frac{a'}{b'}$, wobei $a = a'g$ und $b = b'g$.*

(iii) [erfülle die Einheitsbedingung aus (2.6), normalisiere durch Herausziehen der Einheiten] ersetze $\frac{a'}{b'}$ durch $\frac{a''}{b''}$, wobei $a'' = a'[u(b')]^{-1}$ und $b'' = b'[u(b')]^{-1}$.

(iv) [erfülle die kanonische Bedingung aus (2.6), bringe die Polynome auf kanonische Form] ersetze $\frac{a''}{b''}$ durch $\frac{f_2(a'')}{f_2(b'')}$, wobei f_2 die Funktion aus Definition 2.7 ist.

bestimmt werden.

Wie bei den Polynomen, kann es auch hier nützlich sein, nicht kanonische Formen (auch allgemeinere Formen, die weder kanonisch noch normal sind) für rationale Funktionen zu betrachten. Es werden hier keine formalen Definitionen von Normalformen für rationale Funktionen angegeben, jedoch kann man mögliche Normalformen wie folgt umreißen:

faktorisiert / faktorisiert: Zähler und Nenner sind in faktorisierter Normalform,

faktorisiert / erweitert: Zähler ist in faktorisierter Normalform und der Nenner ist in erweiterter kanonischer Form, *erweitert / faktorisiert:* Zähler ist in erweiterter kanonischer Form und der Nenner ist in faktorisierter Normalform.

2.4.2 Potenzreihen

2.4.2.1 Die TPS-Darstellung

Die Darstellung von Potenzreihen wirft das Problem auf, eine *endliche* Darstellung für einen *unendlichen* Ausdruck zu finden. Offensichtlich kann man Potenzreihen nicht direkt in einer analogen Form zu der erweiterten kanonischen Form für Polynome darstellen, weil es hier unendlich viele Terme gibt. Eine gewöhnliche Lösung dieses Problems ist das Benutzen einer *abgeschnittenen Potenzreihen* (truncated power series (TPS)) Darstellung, mit der eine Potenzreihe

$$a(x) = \sum_{k=0}^{\infty} a_k x^k \in D[[x]] \tag{2.7}$$

dargestellt wird als

$$\sum_{k=0}^{t} a_k x^k, \tag{2.8}$$

wobei t ein bestimmter *Abschneidungsgrad* (truncation degree) ist. Da nur endlich viele Terme dargestellt werden, sieht eine TPS, wie (2.8), genauso aus wie ein Polynom. Es muß jedoch eine Unterscheidung zwischen einem Polynom vom Grad t und einer TPS mit Abschneidungsgrad t vorgenommen werden, da Ergebnisse arithmetischer Operationen auf den beiden Objekttypen nicht identisch sind. Um diese Unterscheidung durchführen zu können, ist es angebracht, die Schreibweise

$$a(x) = \sum_{k=0}^{t} a_k x^k + O(x^{t+1}) \tag{2.9}$$

für die TPS Darstellung (vom Abschneidungsgrad t) für Potenzreihen (2.7) zu benutzen, wobei der Ausdruck $O(x^p)$ allgemein eine unbestimmte Potenzreihe $\alpha(x)$ mit $\mathrm{ord}[\alpha(x)] \geq p$ bezeichnet.

Die nicht exakte Natur der TPS Darstellung von Potenzreihen bringt ein Problem beim Erstellen von Normalformen mit sich. Wird in Definition 2.3 die Menge E der Ausdrücke als die Menge aller (unendlicher) Potenzreihen angesehen, so ist durch die Transformation einer Potenzreihe in ihre TPS Darstellung (mit bestimmtem Abschneidungsgrad t) die erste Eigenschaft für eine Normalfunktion verletzt. Es werden auch zwei verschiedene Potenzreihen, die mindestens bis zum Grad t die gleichen Koeffizienten haben, auf die gleiche TPS Darstellung abgebildet. Man kann aber auch einen mehr praktischen Standpunkt einnehmen und die Menge E als alle TPS Darstellungen der Form (2.9) mit bestimmtem Abschneidungsgrad t betrachten. Da hier nur Bereiche univariater Potenzreihen, die meist einen Körper aus rationalen Funktionen als Koeffizientenbereich haben, betrachtet werden, folgt sofort, daß (2.9) eine Normalform für die Menge E ist, wählt man nur eine Normalform für die Koeffizienten und eine kanonische Form für die Menge E, wenn man eine kanonische Form für die Koeffizienten wählt.

2.4.2.2 NTPS-Darstellungen

Die TPS Darstellung ist nicht die einzige Möglichkeit, eine unendliche Potenzreihe endlich darzustellen. Es gibt auch Darstellungen, die endlich und *exakt* sind. Zum Beispiel kann die Taylor-Entwicklung der Funktion e^x um den Punkt $x = 0$ geschrieben werden als

$$\sum_{k=0}^{\infty} \frac{1}{k!} x^k \tag{2.10}$$

(d.h. die Koeffizienten können berechnet werden), was eine exakte Darstellung einer vollständigen unendlichen Reihe ist, bei der nur eine endliche Zahl von Symbolen verwendet wird. Die Form (2.10) ist ein spezieller Fall für das, was man mit *nicht abgeschnittener Potenzreihen* (non-truncated power series (NTPS)) Darstellung einer Potenzreihe $a(x) \in D[[x]]$ bezeichnet, die die allgemeine Form

$$a(x) = \sum_{k=0}^{\infty} f_a(k) x^k \tag{2.11}$$

hat, wobei $f_a(k)$ eine bestimmte *Koeffizientenfunktion* ist (definiert für alle nichtnegativen ganzen Zahlen k), die den k-ten Koeffizienten berechnet. Durch Darstellen der Koeffizientenfunktion $f_a(k)$ ist die unendliche Potenzreihe $a(x)$ vollständig repräsentiert. Die Tatsache, daß man nur eine endliche Zahl von Koeffizienten jemals explizit *berechnen* kann, ist auf diesem Weg getrennt von der Frage der *Darstellung* und es ist – im Gegensatz zur TPS Darstellung – nicht nötig, eine maximale Koeffizientenanzahl vorzubestimmen, da man die Koeffizienten ja schließlich explizit berechnen kann.

Ist $a(x)$ die Potenzreihe (2.10), dann kann die Koeffizientenfunktion bestimmt werden durch

$$f_a(k) ::= \frac{1}{k!}. \tag{2.12}$$

Für eine praktische Implementierung der NTPS Darstellung ist es vorteilhaft, die explizit ausgerechneten Koeffizienten zu speichern, so daß man diese, wenn sie in der Berechnung später noch einmal benötigt werden, nicht neu zu berechnen braucht. Daher sollte zu einem bestimmten Berechnungszeitpunkt, wenn die ersten l Koeffizienten für $a(x)$ bereits explizit berechnet und in einer linearen Liste $a = (a_0, a_1, \ldots, a_{l-1})$ abgespeichert sind, die Spezifikation der Koeffizientenfunktion (2.12) umgeschrieben werden auf

$$f_a(k) ::= \textbf{ if } k < l \textbf{ then } a[k] \textbf{ else } \frac{1}{k!}, \tag{2.13}$$

wobei $a[k]$ ein Element der linearen Liste a bezeichnet. Initialisiert wird die Spezifikation (2.13) mit $l = 0$. Allgemein ist l die gegenwärtige *Länge* von a. Man sieht auch, daß vom Standpunkt einer effizienteren Berechnung aus, es besser wäre (2.13) umzuschreiben auf

$$f_a(k) ::= \textbf{ if } k = 0 \textbf{ then } 1 \textbf{ else } \textbf{ if } k < l \textbf{ then } a[k] \textbf{ else } \frac{f_a(k-1)}{k}, \tag{2.14}$$

wobei die Rekurrenz $a_k = \frac{f_a(k-1)}{k}$ (für $k > 0$) benutzt wird, um die Koeffizienten sukzessiv zu berechnen.

Die Spezifikation der Koeffizientenfunktion kann sehr viel komplexer werden, als es oben angedeutet ist. Sind $a(x)$ und $b(x)$ zwei Potenzreihen mit den Koeffizientenfunktionen $f_a(k)$ und $f_b(k)$, dann kann die Summe

$$c(x) = a(x) + b(x)$$

durch die Koeffizientenfunktion

$$f_c(k) ::= f_a(k) + f_b(k) \tag{2.15}$$

und das Produkt

$$d(x) = a(x)b(x)$$

durch die Koeffizientenfunktion

$$f_d(k) ::= \sum_{i=0}^{k} f_a(i) f_b(k-i) \tag{2.16}$$

spezifiziert werden. Sind $f_a(k)$ und $f_b(k)$ explizit Ausdrücke in k, so kann man die Koeffizientenfunktion $f_c(k)$ in (2.15) als einen expliziten Ausdruck in k schreiben, $f_d(k)$ in (2.16) kann man dagegen im allgemeinen nicht auf einen expliziten Ausdruck in k vereinfachen.

2.4.3 Erweiterte Potenzreihen

Es kann gezeigt werden, daß der Körper $F\langle x\rangle$ der erweiterten Potenzreihen[2]

$$a(x) = \sum_{k=m}^{\infty} a_k x^k$$

(extended power series (EPS)) über dem Koeffizientenkörper F mit dem Quotientenkörper[3] $F((x))$ über dem Ring der Potenzreihen $F[[x]]$ gleichgesetzt werden kann. Eine kanonische Form für den Quotientenkörper $F((x))$, die die Bedingungen (2.6) erfüllt, hat die Form

$$\frac{a(x)}{x^n} \qquad \text{mit } a(x) \in F[[x]],\ n \geq 0. \tag{2.17}$$

Damit kann man normale und kanonische Formen für EPS direkt aus den für die Darstellung gewöhnlicher Potenzreihen gewählten normalen bzw. kanonischen Formen mit einer zusätzlichen Information, die den Wert für n in (2.17) festlegt, erhalten.

2.5 Darstellung algebraischer Funktionen und Zahlen

Unter *algebraisch* versteht man hier in diesem Zusammenhang eine Lösung einer Polynomgleichung. So ist $y = \sqrt[3]{x^2 - 1}$ eine algebraische Funktion, da y sowohl

- eine Funktion

als auch

- Lösung von $y^3 - x^2 + 1 = 0$

ist. $\sqrt{2}$ ist eine algebraische Zahl, weil sie Zahl und (eine) Lösung der Gleichung $\alpha^2 - 2 = 0$ ist.

[2]Der Ring der formalen Potenzreihen $F[[x]]$ über einem Körper F kann auf den Körper $F\langle x\rangle$ der erweiterten formalen Potenzreihen

$$a(x) = \sum_{k=-\infty}^{\infty} a_k x^k$$

erweitert werden, wobei nur endlich viele $a_k \neq 0$ sind für $k < 0$, d.h. es gibt ein $m \in \mathbb{Z}$ (welches von $a(x)$ abhängt), so daß $a_k = 0$ für alle $k < m$. (Das Wort 'formal' wird benutzt, um den Datentyp zu beschreiben und um Konflikte mit Begriffen und Methoden aus der Analysis zu vermeiden.)

[3]Dies ist der Körper der formalen Laurent-Reihen, oder – wie bereits erwähnt – der Körper der erweiterten Potenzreihen.

Man spricht im allgemeinen von algebraischen Funktionen, selbst wenn es sich um eine algebraische Zahl handeln sollte. Jedes Radikal ist ein algebraischer Ausdruck. Die Umkehrung gilt jedoch nicht, wie der Mathematiker Abel bewies. Genauer, es kann die algebraische Zahl γ, die Lösung der Gleichung $\gamma^5+\gamma+1=0$ ist, nicht in Radikalen ausgedrückt werden.
Man kann damit 3 Klassen von algebraischen Funktionen unterscheiden:

(a) einfache Radikale, wie $\sqrt[3]{x^2-1}$, $\sqrt{2}$,

(b) einfache oder geschachtelte Radikale, welche auch Ausdrücke, wie $\sqrt{1+\sqrt{2}}$, $\sqrt[3]{\sqrt{2}+\sqrt[3]{x}}$ enthalten,

(c) allgemeine algebraische Ausdrücke, in denen auch Ausdrücke, wie die algebraische Zahl γ, die durch $\gamma^5+\gamma+1=0$ definiert ist, enthalten sind.

Jede Klasse ist in den nachfolgenden Klassen enthalten.

2.5.1 Einfache Radikale

Eine ziemlich offensichtliche Lösung zur Darstellung dieser ersten Klasse ist, jedes Radikal als eine Variable entweder in einem Polynom oder in einem rationalen Ausdruck zu betrachten. Ist α eine n-te Wurzel, so benutzt man nur die Potenzen $0,\ldots,n-1$ von α, die höheren Potenzen werden durch niedrigere ersetzt.

Diese Darstellung ist aus zwei Gründen nicht kanonisch.
Erstens: Es gibt ein Problem mit Brüchen, die algebraische Ausdrücke enthalten. Zum Beispiel stellen die beiden *verschiedenen* Ausdrücke $\frac{1}{\sqrt{2}-1}$ und $\sqrt{2}+1$ die gleiche Zahl dar. Eine Lösung dafür ist, zu verlangen, daß Wurzeln nur im Zähler auftreten dürfen (damit ist die Darstellung $\frac{1}{\sqrt{2}-1}$ verboten). Dies kann man für jede Wurzel α dadurch erreichen, daß man den Zähler n und den Nenner d des Ausdruckes mit einem Polynom d_1 multipliziert, so daß α im Produkt dd_1 nicht auftritt. Damit ist

$$\frac{n}{d}=\frac{nd_1}{dd_1}$$

und α ist aus dem Nenner entfernt. Man berechnet das Polynom d_1 mit Hilfe des erweiterten euklidischen Algorithmus angewandt auf d und p, wobei p das Polynom ist, das α definiert.[4]

[4] Dieser Algorithmus liefert die beiden Polynome d_1 und d_2, so daß $dd_1+pd_2=c$, wobei c nicht von α abhängt und es ist (da $p(\alpha)=0$) dann $dd_1=c$.

Mit dieser Forderung erhält man kanonische Formen, sofern die Radikale unabhängig voneinander sind. Doch diese Einschränkung hat zur Folge, daß die Darstellung nicht immer effizient ist. Es ist beispielsweise

$$\begin{aligned}\frac{1}{\sqrt{2}+\sqrt{3}+\sqrt{5}+\sqrt{7}} &= \frac{1}{215}(22\sqrt{3}\sqrt{5}\sqrt{7}-34\sqrt{2}\sqrt{5}\sqrt{7}-50\sqrt{2}\sqrt{3}\sqrt{7}\\ &\qquad +135\sqrt{7}+62\sqrt{2}\sqrt{3}\sqrt{5}-133\sqrt{5}-145\sqrt{3}+185\sqrt{2}).\end{aligned}$$

Daher wird meist die normale Darstellung benutzt, die durch Brüche in den Potenzen von α kleiner n gegeben ist.

Zweitens: Gegenseitige Abhängigkeit der Radikale. Dazu drei sehr offensichtliche Beispiele:

- man sollte nicht $\sqrt{1}$ konstruieren, da man mit 1 arbeitet;
- man kann mit $\sqrt{2}$ oder mit $\sqrt{8}$ rechnen, aber nicht mit beiden, da $\sqrt{8} = 2\sqrt{2}$ und diese Darstellung nicht kanonisch, nicht einmal normal ist;
- man kann nur mit zwei der drei Radikale $\sqrt{2}$, $\sqrt{3}$ und $\sqrt{6}$ arbeiten, da die Beziehung $\sqrt{2}\sqrt{3} = \sqrt{6}$ gilt.

Man könnte nun annehmen es reiche völlig aus sicher zu stellen, daß alle Radikanden (Zahlen, Polynome oder rationale Funktionen) eines Ausdrucks zueinander teilerfremd sind und, daß die Radikale nicht mehr vereinfacht werden können. Aber die Sache ist etwas komplizierter. Man betrachte dazu $\alpha = \sqrt[4]{-4}$. Als Folge der Faktorisierung $x^4+4 = (x^2-2x+2)(x^2+2x+2)$ ist es möglich, auf $\alpha^2 = 2\alpha - 2$ zu vereinfachen. Najid-Zejli hat einen Algorithmus (1985) entwickelt, der entscheidet, ob eine Beziehung zwischen nicht-geschachtelten Radikalen besteht.

2.5.2 Geschachtelte Radikale

Die hier auftretenden Probleme sind viel schwieriger als in der ersten Klasse. Das *erste* Problem, die Äquivalenz von Brüchen mit Ausdrücken, die Radikale enthalten, hat die gleichen Lösungen und Mängel wie in der ersten Klasse.
Das *zweite* Problem, die Abhängigkeit zwischen Radikalen, ist noch nicht zufriedenstellend gelöst.[5] (Es gibt dazu eine allgemeine Lösung, aber es ist die gleiche Lösung

[5] Borodin *et al.* und Zippel (1985) haben Algorithmen gefunden, die einige geschachtelte Radikale lösen können. Dies geschieht entweder dadurch, daß diese auf eine nicht-geschachtelte Weise geschrieben werden, oder durch beweisen, daß es keine derartige Vereinfachung gibt. Diese Algorithmen sind leider auf einige wenige Spezialfälle, wie Quadratwurzel mit zweistufiger Schachtelungstiefe, beschränkt.

wie für allgemeine algebraische Ausdrücke). Dieses Problem soll anhand der folgenden Identitäten (mit den Jahresangaben, in denen diese bekannt geworden sind!) aufgezeigt werden:

$$\begin{aligned}
\sqrt{9+4\sqrt{2}} &= 1+2\sqrt{2} && (1981)\\
\sqrt{5+2\sqrt{6}}+\sqrt{5-2\sqrt{6}} &= 2\sqrt{3} && (1985)\\
\sqrt{x+\sqrt{x^2-1}} &= \sqrt{\tfrac{x+1}{2}}+\sqrt{\tfrac{x-1}{2}} && (1985)\\
\sqrt{16-2\sqrt{99}+2\sqrt{55-10\sqrt{29}}} &= \sqrt{22+2\sqrt{5}}-\sqrt{11+2\sqrt{29}}+\sqrt{5} && (1974)\\
\sqrt[3]{\sqrt[5]{\tfrac{32}{5}}-\sqrt[5]{\tfrac{27}{5}}} &= \sqrt[5]{\tfrac{1}{25}}+\sqrt[5]{\tfrac{3}{25}}-\sqrt[5]{\tfrac{9}{25}}\\
&= \sqrt[5]{\tfrac{1}{25}}\left(1+\sqrt[5]{3}-\sqrt[5]{3^2}\right) && (1927).
\end{aligned}$$

2.5.3 Allgemeine algebraische Funktionen

Algebraische Funktionen wurden als Lösungen von bestimmten Polynomen definiert. Es sei daher α eine algebraische Funktion (oder Zahl), die als Lösung des Polynoms $p(\alpha)$ definiert ist. Es gibt nun zwei Fälle:

- p ist *nicht irreduzibel.* Dieser Fall ist noch nicht gelöst. Es treten hier schon bei ganz einfachen Berechnungen viele Probleme auf. Man betrachte beispielsweise das Polynom $p(\alpha)=\alpha^2-3\alpha+2$. Hierbei ist α eine algebraische Zahl vom Grad 2 und es sind aufgrund dessen die beiden algebraischen Zahlen $(\alpha-1)$ und $(\alpha-2)$ von Null verschieden. Ihr Produkt ergibt aber gerade das Polynom p, das Null ist.
  Dies ist im Prinzip nichts anderes als eine Verallgemeinerung des Problems, das schon vorher bei $\sqrt{1}$ aufgetaucht ist. Denn das Polynom, das $\sqrt{1}$ definiert, ist z.B. α^2-1, das nicht irreduzibel ist und $(\alpha-1)$ als Faktor hat.
  Dies ist nur ein Beispiel für eines der Probleme, die hierbei auftreten.[6] Das gleiche gilt erst recht für geschachtelte Radikale, d.h. die Definition wie „α ist Lösung (Wurzel) von $p(\alpha)$“ ist nur gerechtfertigt, wenn

- p ist *irreduzibel.* 'Gerechtfertigt' meint hier, daß das Benutzen von Polynomen in α (mit Grad kleiner als der Grad von p) eine kanonische Darstellung liefert. Ebenso ergeben rationale Funktionen (die nicht α im Nenner enthalten) eine kanonische Darstellung.

[6] Aber gerade diese Schwierigkeiten ergeben eine Faktorisierung dieser Polynome, was es möglich macht, reduzible Polynome zur Erzeugung algebraischer Erweiterungen zu verwenden.

Treten mehrere solche Wurzeln auf, müssen alle Polynome, die diese definieren, irreduzibel sein. Aber dies ist noch nicht hinreichend. Sie müssen nicht nur *für sich*, sondern auch *zusammen* irreduzibel sein.
Man betrachte beispielsweise α als eine Wurzel von $\alpha^5 - \alpha - 1$ und β als eine Wurzel von $\beta^5 + 5\beta^4 + 10\beta^3 + 10\beta^2 + 4\beta - 1$. Diese beiden Polynome sind irreduzibel über $\mathbb{Z}$. Wird aber das Polynom, das β definiert, als Polynom angesehen, dessen Koeffizienten von α abhängen, so gilt dies nicht mehr.
Um zu prüfen, ob ein System von Wurzeln α_i von Polynomen p_i gerechtfertigt ist, muß man p_1 über $\mathbb{Z}$ faktorisieren, dann p_2 über $\mathbb{Z}[\alpha_1]$, p_3 über $\mathbb{Z}[\alpha_1, \alpha_2]$, $\ldots$.
Hängt dabei ein p_i von einem α_j ab, so muß p_j vor p_i faktorisiert werden.
Dieses Verfahren wird sehr leicht sehr teuer und es ist sehr schwer eine Faktorisierung über algebraischen Körpern durchzuführen.[7]

Das Problem, eine kanonische Repräsentation für algebraische Ausdrücke zu finden, kann man formal durch die folgenden zwei Probleme beschreiben:

- Problem 1

  Sei X eine rationale Funktion mit rationalen Koeffizienten in endlich vielen Symbolen $a_1, a_2, \cdots, a_n$. Jedes dieser a_i erfülle eine Bedingung $P_i(a_i) = 0$, wobei die P_i Polynome sind, deren Koeffizienten auch rationale Funktionen in den Symbolen $a_1, \cdots, a_{i-1}$ sind. Kann man die Werte $a_1, \cdots, a_n$ charakterisieren, für die X Null wird?

- Problem 2

  Sei X eine rationale Funktion mit rationalen Koeffizienten in endlich vielen Symbolen $a_1, a_2, \cdots, a_n$. Jedes dieser a_i erfülle eine Bedingung $P_i(a_i) = 0$ und Bedingungen $Q_{ij}(a_i) > 0$, wobei die P_i und Q_{ij} Polynome sind wie im Problem 1. Kann man die Werte $a_1, \cdots, a_n$ charakterisieren, für die X Null, reell bzw. positiv wird?

Die Antwort ist in beiden Fällen ja. Wir haben Algorithmen, um diese Probleme zu lösen. Diese fundieren auf der Faktorisierung von Polynomen über algebraischen Zahlen oder der sogenannten dynamischen Auswertung (1. Problem) bzw. der zylindrischen algebraischen Zerlegung (2. Problem). Beide Methoden werden in der Vorlesung nicht behandelt.

[7] Anstelle von verschiedenen algebraischen Zahlen (oder Funktionen) kann man über einem Körper der Charakteristik 0 auch eine einzige algebraische Zahl – das sogenannte *primitive Element* – berechnen. Aber auch hier sind die Berechnungen schwierig und sehr teuer, so daß die Bedeutung des primitiven Elements mehr auf die Theorie (aber hier ist es sehr nützlich) beschränkt bleibt.

Diese Techniken sind jedoch sehr kompliziert und in der Praxis manchmal sehr ineffizient.

Das Problem stellt sich jetzt so dar:

- Problem 3

  Existiert unter den Vorraussetzungen von Problem 1 und 2 eine kanonische Form für X?

Die Antwort ist ja. Das heißt also, im Prinzip existiert eine kanonische Form für algebraische Ausdrücke, aber in der Praxis ist es nicht immer einfach die beste kanonische Repräsentation zu finden. Im besonderen bleibt das Problem, *die* kanonische Repräsentation automatisch zu finden, offen.

2.6 Darstellung transzendenter Funktionen

Transzendente Funktionen werden „durch ihre Namen" dargestellt (dies wird auch bei einigen Systemen mit 'kernel' bezeichnet). Ebenso kann man Zahlen wie $\sin 1$ oder $\sin \pi$ auf diese Weise darstellen.
Ausdrücke mit transzendenten Funktionen kann man als Polynome oder rationale Funktionen mit den kernels (also die transzendenten Funktionen) als Variablen betrachten. Das Problem ist nun die Vereinfachung. Man erhält sie über Termersetzungsregeln, die aus wohlbekannten Identitäten oder Gleichungen stammen, wie

$$\begin{aligned}
\sin(x+y) &= \sin(x)\cos(y)+\cos(x)\sin(y) \\
\sin(x)\cos(y) &= \frac{\sin(x+y)+\sin(x-y)}{2} \\
\log(xy) &= \log(x)+\log(y) \\
\exp(x+y) &= \exp(x)\cdot\exp(y) \\
\log(\exp(x)) &= x \\
\sin(\pi) &= 0 \\
&\cdots .
\end{aligned}$$

Die meisten Systeme erlauben benutzergesteuerte Vereinfachung durch Aufstellen von Regeln der obigen Art. Es gibt Systeme, wie REDUCE, die nicht erkennen, daß $\sin(2x) = \sin(x + x)$ ist, d.h. das Muster '$2x$' wird nicht erkannt, was zur Folge hat, daß z.B. die erste Identität (in REDUCE) zwei Regeln

```
FOR ALL X,Y LET SIN(X+Y) = SIN(X)*COS(Y)+COS(X)*SIN(Y);
FOR ALL X LET SIN(2*X) = 2*SIN(X)*COS(X);
```

erfordert.
Es können auch Regeln zueinander gegenseitig invers sein. Dies ist in REDUCE bei den ersten beiden Identitäten der Fall, d.h. die Vereinfachung endet in einer Schleife bzw. es wird eine Fehlermeldung ausgegeben.
Auch kann die Vereinfachung über Termersetzungsregeln sehr teuer werden und man ist nicht sicher, ob alle notwendigen Regeln gegeben sind (was durch die Benutzersteuerung nicht zu schlimm ist).

Ein idealer Vereinfacher ist daher eine Kombination von algorithmischer Vereinfachung und Vereinfachung durch Termersetzung.

Da bei der Termersetzung nur bekannte Regeln angewandt werden, obwohl es auch möglicherweise noch andere Regeln gibt, ist es wichtig Struktursätze zu entwickeln, die für bestimmte transzendente Bereiche alle möglichen Vereinfachungen exakt beschreiben. Der erste und derjenige, der den wichtigsten solchen Bereich abdeckt, ist der Struktursatz von Risch (1979). Die Prinzipien, auf denen dieser Satz beruht, waren bereits Liouville (1833) bekannt. Aber erst die Computer Algebra hat eine derartige explizite Form erforderlich gemacht.
Dieser Struktursatz[8] ist nicht ganz leicht zu verstehen, daher hier nur eine einfache Anwendung:

- eine Exponentialfunktion ist genau dann unabhängig von bereits eingeführten Exponentialfunktionen und Logarithmen, wenn dessen Argumente nicht als Linearkombination (mit rationalen Koeffizienten) der Logarithmen und den Argumenten der bereits eingeführten Exponentialfunktionen dargestellt werden können.

[8]

Struktursatz 2.10
Sei K ein Körper aus Konstanten und $\theta_1, \ldots, \theta_n$ algebraische, Exponential- (man schreibt dann $\theta_i = u_i = \exp v_i$) oder logarithmische ($\theta_i = v_i = \log u_i$) Funktionen, wobei jedes θ_i über $K(x, \theta_1, \ldots, \theta_{i-1})$ definiert ist. $K(x, \theta_1, \ldots, \theta_n)$ habe den Grundkörper K aus Konstanten. Dann gilt:

(i) Eine Exponentialfunktion θ_i ($\theta_i' = v_i'\theta_i$) ist genau dann transzendent über $K(x, \theta_1, \ldots, \theta_{i-1})$, wenn v_i nicht in der Form $c + \sum_{j=1}^{i-1} n_j v_j$ dargestellt werden kann, wobei $c \in K$ und die n_j rationale Zahlen sind.

(ii) Eine Logarithmusfunktion θ_i ($\theta_i' = u_i'/u_i$) ist genau dann transzendent über $K(x, \theta_1, \ldots, \theta_{i-1})$, wenn keine Potenz u_i^n von u_i in der Form $c \prod_{j=1}^{i-1} u_j^{n_j}$ dargestellt werden kann, wobei $c \in K$ und n und die n_j ganze Zahlen ($n \neq 0$) sind.

Bemerkung: Dieser Satz wird nur auf Funktionen angewandt. Er scheint zwar auch für Zahlen, die durch Exponentialfunktionen und Logarithmen definiert sind (z.B. $e = \exp(1)$) zu gelten, aber dies ist bisher nicht bewiesen.

(Solch eine Kombination würde bedeuten, daß die neue Exponentialfunktion ein Produkt aus Potenzen von Exponentialfunktionen und Argumenten von Logarithmen, die bereits eingeführt wurden, wäre.)

- eine Logarithmusfunktion ist genau dann unabhängig von bereist eingeführten Exponentialfunktionen und Logarithmen, wenn dessen Argumente nicht als Produkt (mit rationalen Exponenten) aus Exponentialfunktionen und Argumenten von Logarithmen, die bereits eingeführt wurden, ausgedrückt werden können.

2.7 Darstellung von Matrizen

Bei Matrizen kann man zwei verschiedene Arten von Berechnungen durchführen.
Eine Möglichkeit ist die *implizite* Berechnung, bei der die Matrizen selbst als Variablen betrachtet werden und man daher keine Information über die genaue Dimension und die Elemente der Matrix braucht. Ein Beispiel dafür ist die mathematische Sprechweise „Seien A und B zwei quadratische Matrizen derselben Dimension".
Die andere Möglichkeit sind *explizite* Berechnungen, bei der die Elemente einer Matrix exakt definiert werden (dies können neben Zahlen auch Polynome, rationale Funktionen oder sonstige symbolische Objekte sein).

Bei impliziten Berechnungen muß man beachten, daß die Matrizen *nicht-kommutative Variablen* sind, d.h. AB braucht nicht das gleiche wie BA zu sein. Dem tragen die verschiedenen Systeme unterschiedlich Rechnung. In MACSYMA werden beispielsweise zwei verschiedene Multiplikationen definiert. Dabei ist '$*$' kommutativ und '.' nicht kommutativ. In REDUCE kann man einen Operator M durch `NONCOM M` als nicht kommutativ erklären, was nichts anderes heißt als, daß ein kernel der mit `M` beginnt (wie `M(1)` oder `M(A,B)`) mit anderen nicht kommutativen kernels nicht kommutiert wird.

Der Rest dieses Abschnittes beschränkt sich auf explizite Matrixberechnungen. Hierbei wird zwischen dicht und dünn besetzter Darstellung unterschieden.

2.7.1 Dicht besetzte Matrizen

Matrizen können dargestellt werden als

- Array aus den Elementen der Matrix
- Vektor aus Vektoren (z.B. in Lisp-Dialekten, in denen kein Array implementiert ist)

- Liste von Listen (aber diese Darstellung ist weniger effizient).

Verwendet man die Bezeichnung $< \ldots >$ für Vektoren, so kann man die Matrix

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

darstellen durch $<< a\ b\ c >< d\ e\ f >< g\ h\ i >>$. Für dicht besetzte Matrizen stellt dies ein gutes Verfahren dar und wird daher von den meisten CA-Systemen benutzt.

Man benutzt zur Addition und Multiplikation dieser Matrizen die gleichen Algorithmen wie für numerische Matrizen. Daher kann man Matrizen der Größe n in $O(n^2)$ Operationen addieren und in $O(n^3)$ Operationen multiplizieren. Es gibt (asymptotisch) schnelle numerische Multiplikationsalgorithmen, wie der von Strassen ($O(n^{\log_2 7})$), Winograd … . Für numerische Matrizen sind diese Algorithmen für $n = 100$ um 18% schneller wie die herkömmlichen Verfahren. Wahrscheinlich sind diese schnellen Multiplikationsalgorithmen auch in der Computer Algebra möglich, aber bisher hat noch niemand versucht diese zu implementieren.
Für die Inversion und den mit ihr verwandten Problemen (Lösen von linearen Gleichungssystemen, Determinante berechnen) sind die numerischen Algorithmen unbrauchbar, weil sich die auftretenden Probleme unterscheiden. In der Computer Algebra gibt es keine Probleme mit der Stabilität, was bedeutet, daß jedes von Null verschiedene Element ein gutes Pivot-Element für die Gauß-Elimination ist. Damit ist dieses Problem hier einfacher wie bei numerischer Berechnung. Auf der anderen Seite gibt es hier das Problem beliebig großer Daten, sowohl bei Zwischenergebnissen, als auch beim Endergebnis.
Für die generische 3×3 Matrix A berechnet sich die Determinante zu

$$\det A = aei - afh - bdi + bfg + cdh - ceg$$

und die Inverse zu

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} ei - fh & -bi + ch & bf - ce \\ -di + fg & ai - cg & -af - cd \\ dh - eg & -ah + bg & ae - bd \end{pmatrix},$$

aber für eine generische 4×4 Matrix hat die Determinante bereits 24 Terme und die Inverse ist schon zu groß, um hier abgedruckt werden zu können. Die Daten für die Endergebnisse können also beliebig groß werden.
Aber dies gilt ebenso für Zwischenergebnisse, denn beispielsweise ist die Determinante einer Matrix vom Typ $\begin{pmatrix} M & M \\ M & M \end{pmatrix}$, wobei M irgendeine generische Matrix ist, gleich 0, doch die Berechnung erzeugt große Daten in den Zwischenergebnissen.

Ein anderes großes Problem ist die Division (besonders bei der Berechnung der Determinante). Nach der Cramerschen Regel ist die Determinante einer Matrix eine Summe von Produkten aus Elementen der Matrix. Sind die Elemente der Matrix aus einem Ring, so ist die Determinante ebenfalls aus diesem Ring. Aber das Eliminationsverfahren benötigt mehrere Divisionen, die vielleicht gar nicht zulässig sind.
Zum Beispiel hat die Matrix $\begin{pmatrix} 5 & 2 \\ 2 & 5 \end{pmatrix}$ mit Elementen aus dem Ring $\mathbb{Z}_{10}$ eine wohldefinierte Determinante (sie ist einfach 1), aber man kann in diesem Ring nicht durch 5 oder 2 teilen.
Selbst wenn die Division möglich ist, bedeutet dies, daß man mit Brüchen rechnen muß, was sehr kostspielig ist, da dazu gcd-Berechnungen notwendig werden (und diese sind oft nicht trivial).

Aus diesem Grund ist das Verfahren von Bareiss (1968) wichtig. Es ist eine Variante des Gaußschen Eliminationsverfahren, in der jede Division im Ring ein Ergebnis im Ring liefert und keine Brüche notwendig werden. Daher wird dieses Verfahren in der Computer Algebra dann oft eingesetzt, wenn der Ring der Matrixelemente die Division erlaubt (d.h. der Ring muß integer sein). Andernfalls benutzt man ein Verfahren (Sasaki und Murao (1982)), bei dem man mehrere neue Variablen zum Ring hinzufügt.

Die Cramersche Methode stellt die Determinante einer Matrix der Größe n als Summe von $n!$ Produkten aus n Elementen dar. Dies ist numerisch sehr ineffizient, da die Zahl der Operationen in $O(n(n!))$ liegt, anstatt in $0(n^3)$ wie für die Gauß-Elimination. In der Computer Algebra dagegen hängt die Effizienz von der Datengröße der Zwischenergebnisse ab. Bei diesem Verfahren ist diese für alle Zwischenergebnisse kleiner wie das Ergebnis selbst. Im Falle von Matrizen aus multivariaten Polynomen (dünn besetzte Darstellung) ist dieses Verfahren schneller als jedes Verfahren, das auf der Gauß-Elimination beruht.
Hat man Matrizen aus ganzen Zahlen oder univariaten Polynomen scheint das Verfahren von Bareiss das Effizienteste zu sein.

2.7.2 Berechnung der Determinante nach Bareiss

Bareiss hat eine ganze Familie von Verfahren zur Elimination ohne Brüche entwickelt (d.h. alle benötigten Divisionen sind exakt). Das einfachste solche Verfahren wird mit „Einschritt“ bezeichnet. Es basiert auf einer Verallgemeinerung der Identität von

Sylvester. Sei $a_{i,j}^{(k)}$ definiert als die Determinante

$$\begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k} & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k} & a_{2,j} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,k} & a_{k,j} \\ a_{i,1} & a_{i,2} & \cdots & a_{i,k} & a_{i,j} \end{vmatrix}.$$

Speziell für eine Matrix der Größe n ergibt dies die Determinante $a_{n,n}^{(n-1)}$. Die grundlegende Identität lautet

$$a_{i,j}^{(k)} = \frac{1}{a_{k-1,k-1}^{(k-2)}} \begin{vmatrix} a_{k,k}^{(k-1)} & a_{k,j}^{(k-1)} \\ a_{i,k}^{(k-1)} & a_{i,j}^{(k-1)} \end{vmatrix}.$$

Man kann also sicher sein, nach jeder Elimination durch das Pivotelement der vorhergehenden Elimination teilen zu können.

Zur Veranschaulichung des Verfahrens von Bareiss betrachte man die generische Matrix

$$\begin{vmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{vmatrix}.$$

Durch Elimination nach der ersten Zeile (ohne Division) erhält man die Matrix

$$\begin{vmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & b_{2,2}b_{1,1} - b_{2,1}b_{1,2} & b_{2,3}b_{1,1} - b_{2,1}b_{1,3} \\ 0 & b_{3,2}b_{1,1} - b_{3,1}b_{1,2} & b_{3,3}b_{1,1} - b_{3,1}b_{1,3} \end{vmatrix}.$$

Nach einer zweiten Elimination ergibt sich

$$\begin{vmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ 0 & b_{2,2}b_{1,1} - b_{2,1}b_{1,2} & b_{2,3}b_{1,1} - b_{2,1}b_{1,3} \\ 0 & 0 & \begin{array}{r} b_{1,1}(b_{3,3}b_{2,2}b_{1,1} - b_{3,3}b_{2,1}b_{1,2} - b_{3,2}b_{2,3}b_{1,1} \\ +b_{3,2}b_{2,1}b_{1,3} + b_{3,1}b_{2,3}b_{1,2} - b_{3,1}b_{2,2}b_{1,3}) \end{array} \end{vmatrix}.$$

Damit teilt $b_{1,1}$ alle Elemente der dritten Zeile.

2.7.3 Dünn besetzte Matrizen

Hier schließt die Computer Algebra dicht an die Methoden der numerischen Mathematik an. Gewöhnlich wird eine Darstellung benutzt, bei der jede Zeile (Spalte) als Liste aller von Null verschiedenen Elemente der Zeile (Spalte) mit einem Verweis auf die zugehörige Spalte (Zeile) gespeichert wird.

Die Addition solcher Matrizen ist ziemlich einfach. Dagegen ist die Multiplikation etwas schwieriger, da die linke Matrix Zeile für Zeile und die rechte Matrix Spalte für Spalte durchquert werden muß.
Der Berechnung der Determinante liegt die Cramer-Entwicklung zugrunde. Dazu gibt es noch mehrere Tricks, um die beste Richtung für die Entwicklung zu wählen und um bereits berechnete Ergebnisse zu benutzen.
Die Inverse einer dünn bestezten Matrix ist meist dicht besetzt und sollte nicht berechnet werden. Damit gibt es drei generelle Methoden zur Lösung eines linearen Gleichungssystemes $Ax = b$:

- Die x_i werden als Lösung in Termen von

  $$x_i = \sum A_{i,j} b_j$$

  dargestellt, wobei jedes $A_{i,j}$ ein Minor von A ist (d.h. eine Unterdeterminante von A, die durch Streichen der i-ten Spalte und j-ten Zeile der Matrix A entsteht). Diese Formel benötigt keine Berechnung von Inversen.
  Die Minoren kann man mit der Cramerschen Regel berechnen. Auch werden bei diesem Verfahren Zwischenergebnisse für weitere Berechnungen genutzt. Der Nachteil dabei ist, daß Speicherplatz für das Ablegen der wiederverwendbaren Zwischenergebnisse benötigt wird.

- Gaußsches Eliminationsverfahren zusammen mit einigen Tricks, die die Zahl der von Null verschiedenen Elemente in den Zeilen (oder Spalten) mit berücksichtigen. Die Matrix wird während der Berechnung immer dichter besetzt.
  Es gibt einen Bericht (Coppersmith und Davenport 1985 in Scratchpad-II) über eine Berechnung zur Lösung eines linearen Gleichungssystems aus 1061 Gleichungen in 739 Unbekannten, das dieses Verfahren benutzt (und dieses Verhalten tatsächlich aufweist).

- Seit kurzem gibt es auch an die Computer Algebra angepaßte numerische Verfahren, wie das Verfahren von Lanczos und die Methode der konjugierten Gradienten. Diese Verfahren sind asymptotisch besser, aber sie sind für „kleinere“ Probleme langsamer.

2.8 Datenstrukturen für multiprecision Zahlen und rationale Zahlen

2.8.1 Multiprecision Zahlen

Ein Digitalrechner bietet Hardwaremöglichkeiten zur Speicherung und Durchführung arithmetischer Operationen auf dem grundlegenden Datentyp, der normalerweise 'integer' (ganze Zahlen) genannt, hier aber mit *single-precision Zahlen* bezeichnet wird. Der Wertebereich einer single-precision Zahl ist durch die Zahl der verschiedenen Codes, die bei einem Maschinenwort, das meist die Länge 8, 16, 32, 36, 48 oder 64 Bits besitzt, gemacht werden können, beschränkt. So wird beispielsweise eine single-precision Zahl mit Vorzeichen und Wortlänge 16 auf den Wertebereich von $-2^{15}+1 = -32767$ bis $2^{15}-1 = +32767$ eingeschränkt. Diese eingeschränkte Darstellung von Objekten aus dem Integritätsbereich Z reicht für die Zwecke symbolischen Rechnens nicht aus.

Eine brauchbarere Darstellung für ganze Zahlen erhält man durch Aufsetzen einer Datenstruktur auf den Grunddatentyp 'single-precision Zahl'. Eine *multiprecision Zahl* ist eine lineare Liste $(d_0, d_1, \ldots, d_{l-1})$ von single-precision Zahlen, die den Wert

$$\sum_{i=0}^{l-1} d_i \beta^i$$

repräsentiert, wobei die Basis β vorbestimmt ist. Man beachte, daß eine single-precision Zahl entweder positiv oder negativ sein kann; eine positive multiprecision Zahl kann als Liste von positiven single-precision Zahlen und eine negative multiprecision Zahl als Liste von negativen single-precision Zahlen dargestellt werden. (Man könnte auch alternativ, statt das Vorzeichen der multiprecision Zahl in jedem Element der Liste zu speichern, andere Konventionen zur Speicherung des Vorzeichens einführen, aber diese Unterschiede sind hier nicht von Interesse).

Die *Basis* β könnte prinzipiell jede solche positive ganze Zahl größer 1 sein, so daß $\beta-1$ eine single-precision Zahl ist. Aus Effizienzgründen wird β jedoch als eine *große* solche Zahl gewählt. Die zwei üblichen Wahlen für β sind

(i) β so, daß $\beta-1$ die größte single-precision Zahl ist (z.B. $\beta = 2^{35}$, falls die Wortlänge 36 Bits mit Vorzeichen ist)

(ii) $\beta = 10^p$, wobei p so groß wie möglich gewählt wird, so daß $\beta - 1$ eine single-precision Zahl ist (z.B. $\beta = 10^{10}$, falls die Wortlänge 36 Bits ist).

Die *Länge* l der linearen Liste zur Darstellung der multiprecision Zahlen kann dynamisch (d.h. sie hängt von der speziellen darzustellenden Zahl ab) oder statisch (d.h.

von vorbestimmter fester Länge) sein, abhängig davon ob die lineare Liste mit Zeigerzuweisung (linked allocation) oder durch Anlegen eines Arrays (sequentielle Zuweisung, sequential allocation) implementiert ist.

2.8.2 Referenzierte und sequentielle Zuweisung

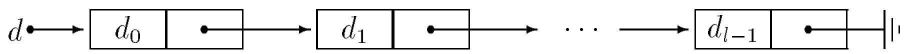
Eine übliche Methode zur Implementierung der Datenstruktur lineare Liste für multiprecision Zahlen ist die Benutzung verketteter (referenzierter) Listen, in der jeder *Knoten* von der Form

ZIFFER	ZEIGER

ist. Das ZIFFER-Feld enthält eine Ziffer zur Basis β (d.h. eine single-precision Zahl) und das ZEIGER-Feld einen Zeiger auf den nächsten Knoten (oder einen 'Listenende' Zeiger) der verketteten Liste. Damit wird die multiprecision Zahl $d = (d_0, d_1, \ldots, d_{l-1})$ mit dem Wert

$$d = \sum_{i=0}^{l-1} d_i \beta^i \tag{2.18}$$

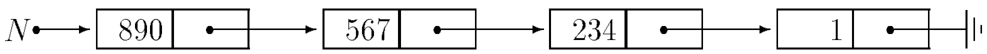
durch die verkettete Liste



dargestellt, wobei die Schreibweise 'Erde' den 'Listenende' Zeiger repräsentiert. Man beachte, daß die Ordnung, in der die Basis-β Ziffern d_i referenziert sind, in umgekehrter Reihenfolge zur konventionellen Methode der Zahlendarstellung steht. Für $\beta = 10^3$ wird die Dezimalzahl

$$N = 1234567890 \tag{2.19}$$

durch die verkettete Liste



dargestellt. Diese Anordnung entspricht der gewöhnlichen Schreibweise der Darstellung einer Zahl zur Basis β wie in (2.18) und – was noch viel wichtiger ist – sie entspricht der Reihenfolge in der, bei der Durchführung der Operationen Addition und Multiplikation auf den ganzen Zahlen, auf die Ziffern zugegriffen wird.

Ein andere Standardmethode zur Implementierung linearer Listen ist die Verwendung von sequentieller Zuweisung. Bei diesem Entwurf ist die Länge l der erlaubten multiprecision Zahlen eine vorbestimmte feste Zahl und es wird jeder multiprecision Zahl ein Array der Länge l (d.h. l aufeinanderfolgende Worte im Speicher) zugeteilt. Damit wird die multiprecision Zahl d aus (2.18) durch das Array

d
d_0
d_1
d_2
$\vdots$
d_{l-1}

dargestellt, wobei darauf zu achten ist, daß jede multiprecision Zahl mit allen l β-Ziffern ausgedrückt werden muß (falls nötig, durch Einführen von Nullen für Terme höherer Ordnung in (2.18)). Ist z.B. $\beta = 10^3$ und $l = 10$, so können ganze Zahlen, die eine Länge von 30 Dezimalziffern nicht überschreiten, dargestellt werden. Die Zahl N aus (2.19) ist dann durch das Array

N
890
567
234
1
0
0
0
0
0
0

gegeben.

2.8.3 Vor- und Nachteile

Es gibt einige wohlbekannte Faktoren, die die Wahl zwischen referenzierter und sequentieller Zuweisung beeinflussen. Sequentielle Zuweisung erfordert eine vorbestimmte Länge l, was auf zwei signifikante Nachteile führt:

- man muß für die benötigten ganzen Zahlen eine maximale Länge festlegen (was bei einer Überschreitung dieses Maximums immer zu einem Systemfehler führt)
- ein beträchtlicher Speicherplatz wird zum Ablegen der Null-Ziffern höherer Ordnung verschwendet (entsprechend vergeudet man auch Rechenzeit beim Zugriff auf diese irrelevanten Nullen).

Bei referenzierter Zuweisung werden diese Probleme vermieden, da man die irrelevanten Nullen höherer Ordnung nicht darstellt und die Länge einer Liste nur durch den dem System zugänglichen Speicher beschränkt ist.

Auf der anderen Seite bringen verkettete Listen ebenfalls wenigstens zwei Nachteile mit sich:

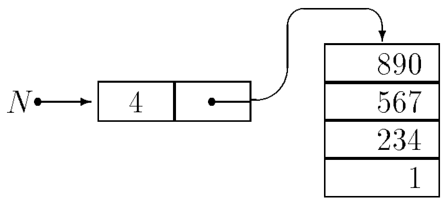
- ein beträchtlicher Speicherplatz wird zum Ablegen der Zeiger benötigt
- die erforderliche Zugriffszeit auf sukzessive Ziffern ist bedeutend größer als die für den Zugriff auf ein Array.

Diese beiden Nachteile scheinen besonders bedenklich bei dieser speziellen Anwendung verketteter Listen zu sein, weil die dazu nötigen Zeiger den Speicherbedarf für multiprecision Zahlen verdoppeln können und auch, weil die Ziffern der ganzen Zahlen *immer* in einer Folge gespeichert werden und auch so wieder auf sie zugegriffen wird. Trotzdem führen die Vorteile von *indefinite-precision Zahlen* (d.h. multiprecision Zahlen mit dynamisch bestimmter Länge l) zusammen mit den noch größeren Vorteilen von referenzierter Zuweisung zur Darstellung von Polynomen und allgemeinerer Funktionenklassen auf die Wahl von referenzierter Zuweisung bei den meisten CA-Systemen.

Eine dritte mögliche Implementation von multiprecision Zahlen ist die beschreibende Zuweisung (descriptor allocation), die nun versucht, die besten Eigenschaften der beiden obigen Implementierungen zu verbinden. Bei diesem Entwurf benutzt man einen Beschreibungsblock (descriptor block),

l	ZEIGER

d.h. einen Knoten, der die Länge l der darzustellenden multiprecision Zahl enthält und ein ZEIGER-Feld mit einem Verweis auf einem Array aus l β-Ziffern, dem Array-Block. Für $\beta = 10^3$ wird die Zahl N aus (2.19) mit diesem Entwurf als



dargestellt. Es scheint, daß hier alle Nachteile der beiden anderen Implementierungen vermieden sind. Trotzdem sind auch hiermit Probleme verbunden, die mit der Notwendigkeit dynamisch angelegter Arrays und einem komplizierten garbage collection Algorithmus zusammenhängen.

2.8.4 Rationale Zahlen

Der Körper $\mathbb{Q}$ der rationalen Zahlen ist der Quotientenkörper des Integritätsbereiches $\mathbb{Z}$ der ganzen Zahlen. Eine natürliche Darstellung der rationalen Zahlen ist daher das Paar (*Zähler, Nenner*), wobei sowohl der *Zähler* wie *Nenner* eine multiprecision Zahl ist. Die Grunddatenstruktur ist eine Liste der Länge 2 bei der jedes Element selbst wieder eine lineare Liste ist. Die Darstellung wird durch die gewöhnlichen Bedingungen auf den rationalen Zahlen kanonisiert.
Eine rationale Zahl kann man durch einen Knoten

ZEIGER1	ZEIGER2

darstellen, wobei ZEIGER1 ein Verweis auf den Zähler, eine multiprecision Zahl (diese entweder eine verketteter Liste oder ein Array), und ähnlich ZEIGER2 ein Verweis auf den Nenner ist. Im Falle einer sequentiellen Zuweisung der multiprecision Zahlen ist es auch möglich, zur Darstellung einer rationalen Zahl ein 2-dimensionales Array (l Zeilen und 2 Spalten) zu verwenden, da die Länge l von Zähler und Nenner eine feste Zahl ist.

2.9 Datenstrukturen für Polynome, rationale Funktionen und Potenzreihen

2.9.1 Formen und Datenstrukturen

Die Datenstruktur zur Darstellung multivariater Polynome wird von den Auswahlentscheidungen auf den verschiedenen – bereits bekannten – Abstraktionsebenen beeinflußt:

Ebene	**Auswahl**	**Auswirkung**
A	normale / kanonische Formen	keine Auswirkung auf die Datenstruktur
B	rekursive / distributive Darstellung	eng verwandt mit der Datenstruktur
C	dünn / dicht besetzte Darstellung	meist dünn besetzte Darstellung; spiegelt sich in Teilen der Datenstruktur wieder
D	Nullexponent Darstellung	die Wahl spiegelt sich auch hier in Teilen der Datenstruktur wieder

Es werden im folgenden die beiden Datenstrukturen

- verzeigerte Liste mit rekursiver, dünn besetzter und Nullexponent Darstellung
- Beschreibungsblock (Arrays) mit distributiver, dünn besetzter und Nullexponent Darstellung

beschrieben. Danach wird eine zusätzliche Struktur eingeführt, die es erlaubt, aufgesetzt auf einer dieser beiden Grunddatenstrukturen, eine faktorisierte Normalform zu implementieren.

2.9.2 Eine verzeigerte Listenstruktur

Bei der rekursiven Darstellung multivariater Polynome in erweiterter kanonischer Form wird ein Polynomring $D[x_1, \ldots, x_\nu]$ als Ring $D[x_2, \ldots, x_\nu][x_1]$ angesehen und diese Sichtweise rekursiv auf den 'Koeffizientenring' $D[x_2, \ldots, x_\nu]$ fortgesetzt. Dadurch wird ein Polynom $a(x_1, \ldots, x_\nu) \in D[x_1, \ldots, x_\nu]$ in der 'obersten Stufe' als ein univariates Polynom in x_1 betrachtet und kann durch eine verzeigerte Liste, in der jeder Knoten die Form

KOEF_ZEIGER	EXPONENT	NEXT_ZEIGER

hat, dargestellt werden. Jeder dieser Knoten gibt einen Polynomterm $a_i x_1^i$ mit $a_i \in D[x_2, \ldots, x_\nu]$ wieder, wobei das EXPONENT-Feld den Wert i (als single-precision Zahl) enthält, das KOEF_ZEIGER-Feld einen Zeiger auf den Koeffizienten a_i von x_1^i und das NEXT_ZEIGER-Feld einen Zeiger auf den nächsten Polynomterm (oder einen 'Listenende' Zeiger). Diese Darstellung wird rekursiv angewandt. Um den Namen der Unbestimmten zu bekommen, der auf jeder Stufe dieser Darstellung unterschieden wird, kann man einen 'Kopf Knoten'

VAR_ZEIGER	FIRST_ZEIGER

verwenden, wobei das VAR_ZEIGER-Feld einen Zeiger auf den Namen der Unbestimmten enthält und das FIRST_ZEIGER-Feld einen Zeiger auf den ersten Polynomterm (dieser speziellen Rekursionsstufe).

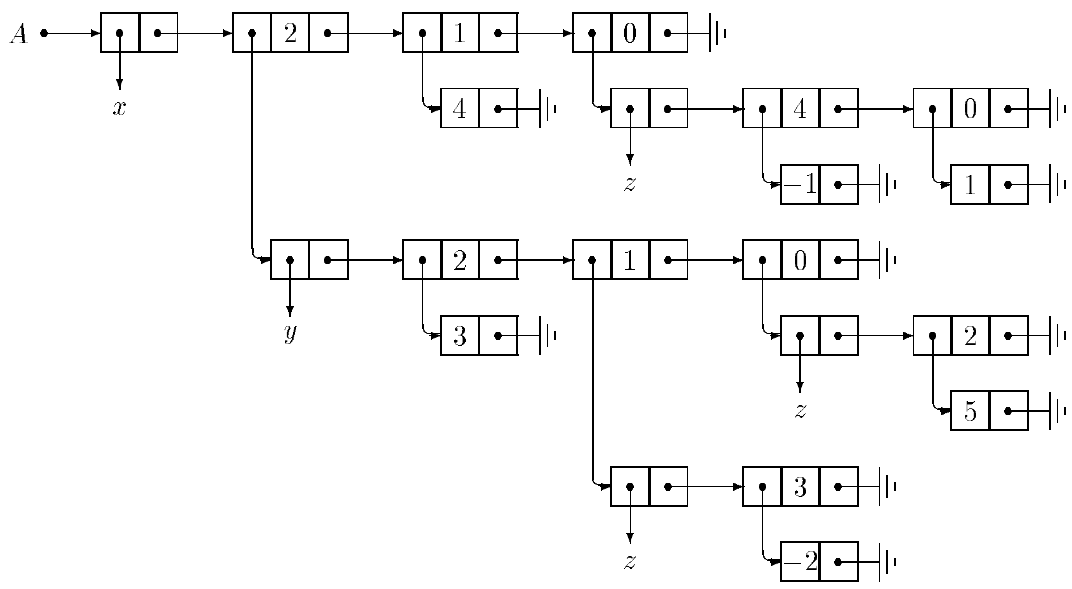
Beispiel:
Sei $D = \mathbb{Z}$ und das Polynom $A(x, y, z) \in \mathbb{Z}[x, y, z]$ gegeben durch

$$A(x, y, z) = 3x^2y^2 - 2x^2yz^3 + 5x^2z^2 + 4x - z^4 + 1.$$

Dann lautet die rekursive Darstellung

$$A(x, y, z) = (3y^2 + (-2z^3)y + 5z^2)x^2 + (4)x + (-z^4 + 1)$$

und die zugehörige verzeigerte Listenstruktur



.

2.9.3 Eine Beschreibungsblockstruktur

Bei der distributiven Darstellung multivariater Polynome in erweiterter kanonischer Form wird ein Polynom $a(x) \in D[x]$ als ein Polynom der Form

$$a(x) = \sum_{e \in \mathbb{N}^\nu} a_e x^e$$

angesehen, wobei $a_e \in D$, $x = (x_1, \ldots, x_\nu)$ der Vektor der Unbestimmten und jedes $e = (e_1, \ldots, e_\nu)$ der entsprechende Vektor der Exponenten ist. Genauer, ein Term $a_e x^e$ ist von der Form

$$a_e x_1^{e_1} x_2^{e_2} \cdots x_\nu^{e_\nu}.$$

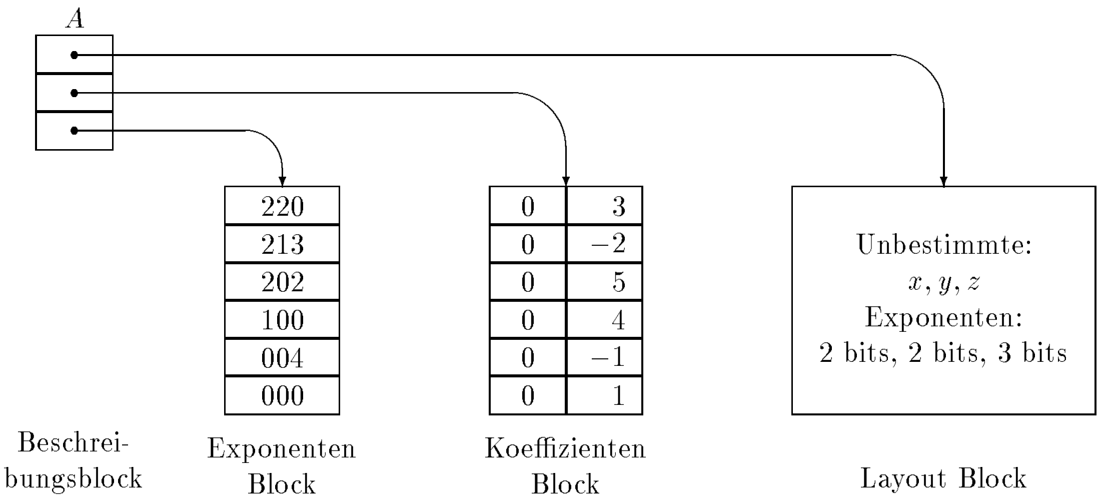
Damit kann man ein Polynom $a(x)$ durch Ablegen von drei Informations-*blöcken* darstellen:

- einem *Layoutblock*, der die Namen der Unbestimmten $x_1, \ldots, x_\nu$ beinhaltet,
- einem *Koeffizientenblock*, der die Liste aller von Null verschiedenen Koeffizienten a_e festhält und

- einem *Exponentenblock*, der die Liste der Vektoren der Exponenten $(e_1, \ldots, e_\nu)$ enthält, wobei jeder solche Vektor einem Koeffizienten a_e des Koeffizientenblockes entspricht.

Die Ordnung der ganzen Zahlen im Exponentenvektor entspricht der Ordnung der Unbestimmen, die im Layoutblock festgelegt ist.

Beispiel:
Sei $A(x, y, z) \in \mathbb{Z}[x, y, z]$ das Polynom des vorhergehenden Beispiels. Benutzt man zur Darstellung ganzer Zahlen die Basis $\beta = 10^3$ und die vorbestimmte Länge $l = 2$, dann erhält man die Beschreibungsblockstruktur



.

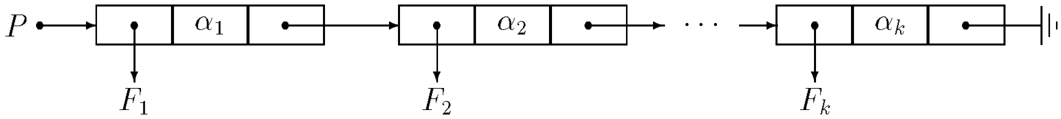
2.9.4 Implementierung einer faktorisierten Normalform

Die beiden Grunddatenstrukturen dieses Abschnittes wurden so beschrieben, wie sie zur Darstellung multivariater Polynome in erweiterter kanonischer Form angewandt werden. Es ist nicht schwer diese Grunddatenstrukturen zur Darstellung von Polynomen in nicht kanonischer Normalform, oder sogar in einer nicht normalen Form zu benutzen. Der Fall der faktorisierten Normalform wird hier kurz vorgestellt.

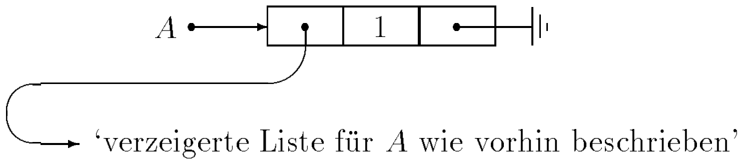
Ein Polynom in faktorisierter Normalform kann dargestellt werden als

$$P = \prod_{i=1}^{k} F_i^{\alpha_i}, \tag{2.20}$$

wobei α_i $(1 \leq i \leq k)$ eine positive ganze Zahl ist, F_i $(1 \leq i \leq k)$ ein Polynom in erweiterter kanonischer Form und $F_i \neq F_j$ für $i \neq j$. Benutzt man eine verzeigerte Listenstruktur, so kann das Polynom P in der Produktform (2.20) durch die verzeigerte Liste

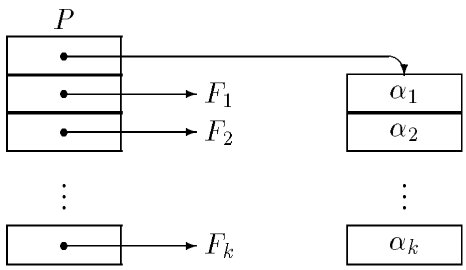


dargestellt werden, wobei jeder Faktor F_i $(1 \leq i \leq k)$ seinerseits durch eine verzeigerte Liste, wie vorhin für Polynome in erweiterter kanonischer Form beschrieben, dargestellt ist. In einem System, bei dem man diesen Entwurf benutzt, wird angenommen, daß alle Polynome in Produktform sind und, falls A ein einzelner Polynomfaktor in erweiterter kanonischer Form ist, wird dieser durch



dargestellt.

Ein ähnlicher Entwurf kann zur Darstellung der Produktform (2.20) durch eine Beschreibungsblockstruktur benutzt werden. Das Polynom P wird dabei durch einen *formalen Produktblock* dargestellt, der ein Array aus Zeigern auf jeden Faktor F_i $(1 \leq i \leq k)$ ist, deren Darstellung bereits beschrieben wurde und einem Zeiger auf ein entsprechendes Array der zugehörgen Potenzen α_i. Damit wird P durch



dargestellt, wobei jedes F_i seinerseits durch eine Beschreibungsblockstruktur für Polynome in erweiterter kanonischer Form dargestellt ist.

2.9.5 Rationale Funktionen

Ein Körper $D(x_1, \ldots, x_\nu)$ rationaler Funktionen besitzt die natürliche Darstellung als Quotientenkörper des Polynomringes $D[x_1, \ldots, x_\nu]$. Eine rationale Funktion wird damit als ein Paar (*Zähler, Nenner*) dargestellt, wobei *Zähler* und *Nenner* jeweils ein Polynom aus dem Ring $D[x_1, \ldots, x_\nu]$ darstellen. Benutzt man eine verzeigerte Listenstruktur oder eine Beschreibungsblockstruktur für Polynome, dann wird eine rationale Funktion durch einen Knoten

ZEIGER1	ZEIGER2

von Zeigern auf das Zähler- und Nennerpolynom dargestellt. Wie bereits in Abschnitt 2.4.1 vorgestellt wird eine rationale Funktion gewöhnlich durch teilerfremde Zähler und Nenner und einheitsnormalem Nenner dargestellt. Zusätzlich kann ein System auch mehrere (möglicherweise unabhängige) Wahlen von Normalformen für den Zähler und den Nenner bereitstellen.

Eine etwas andere Darstellung rationaler Funktionen erhält man durch einfache Verallgemeinerung der für Polynome beschriebenen formalen Produktdarstellung. Erlaubt man in dem formalen Produkt

$$P = \prod_{i=1}^{k} F_i^{\alpha_i},$$

daß die Potenzen α_i positive und negative ganze Zahlen annehmen dürfen, so hat man sofort eine Datenstruktur zur Darstellung rationaler Funktionen. Dabei ist keine Änderung in der Datenstruktur für die bereits eingeführte formale Produktdarstellung nötig.

2.9.6 Potenzreihen

Es werden hier nur univariate Potenzreihen aus einem Ring $D[[x]]$ betrachtet, deren Koeffizientenbereich D einer der vorher besprochenen Ringe ist (u.a. ganze Zahlen, rationale Zahlen, Polynome oder rationale Funktionen). Eine Datenstruktur zur Darstellung von Potenzreihen ist daher eine Erweiterung der bisherigen Datenstrukturen.

Wird die TPS Darstellung benutzt, so besitzt die TPS

$$\sum_{k=0}^{t} a_k x^k + O(x^{t+1})$$

die natürliche Darstellung als lineare Liste

$$(a_0, a_1, \ldots, a_t).$$

Diese lineare Liste kann leicht als verzeigerte Liste oder als ein Array von Zeigern auf die Koeffizienten a_k (mit geeigneter Darstellung) implementiert werden. Bei einer verzeigerten Liste eignet sich die dünn besetzte Darstellung (d.h. nur die von Null verschiedenen Terme werden gespeichert), während sich bei einem Array die dicht besetzte Darstellung anbietet. In der dicht besetzten Darstellung ist der Abschneidungsgrad t bereits implizit durch die Tatsache bestimmt, daß die lineare Liste aus $t+1$ Elementen besteht, wogegen in der dünn besetzten Darstellung der Wert von t als zusätzliche Information gespeichert werden muß.

Die nicht abgeschnittene Darstellung von Potenzreihen kann durch Benutzen einer ähnlichen Datenstruktur implementiert werden. Die Potenzreihe

$$a(x) = \sum_{k=0}^{\infty} f_a(k) x^k$$

wird als lineare Liste

$$(a_0, a_1, \ldots, a_{l-1}, f_a(k))$$

dargestellt, wobei die Zahl l der Koeffizienten, die explizit berechnet wurden, während der Berechnung ansteigen kann. Wieder kann man diese lineare Liste entweder durch eine verzeigerte Liste oder durch ein Array von Zeigern implementieren, mit dem einzigen Unterschied, daß das letzte Element nicht wie die anderen Zeiger auf die explizite Darstellung der Koeffizienten verweist, sondern auf die Darstellung der Koeffizientenfunktion $f_a(k)$. Dabei ist zu beachten, daß im allgemeinen die Darstellung der Koeffizientenfunktion $f_a(k)$ auch Ausdrücke einschließt, die sehr viel komplizierter sind, als die im Abschnitt 2.4.2.2 vorgestellten.

Darstellungen für erweiterte Potenzreihen erhält man durch eine Verallgemeinerung der Darstellung gewöhnlicher Potenzreihen. Wie bereits in Abschnitt 2.4.3 angedeutet, kann die Darstellung einer erweiterten Potenzreihe $a(x)$ mit Koeffizienten aus einem Körper F auch als Darstellung gewöhnlicher Potenzreihen mit einer zusätzlichen Information über die Potenz von x angesehen werden, durch die die gewöhnliche Potenzreihe 'dividiert' werden muß. Hat man daher eine bestimmte Datenstruktur für gewöhnliche Potenzreihen ausgewählt, so erhält man eine Datenstruktur für erweiterte Potenzreihen durch Hinzufügen einer ganzen Zahl (single-precision) zur gewählten Darstellung.

Kapitel 3

Mathematische Grundlagen

3.1 Grundlegende Mathematische Konzepte

Definition 3.1 *Monoid* $[M; \circ, 1]$

1.	$x \circ (y \circ z) = (x \circ y) \circ z$	*(Assoziativgesetz)*
2.	$1 \circ x = x \circ 1 = x$	*(Identität)*

Definition 3.2 *Gruppe* $[G; \circ, -1, 1]$

1. $[G; \circ; 1]$ *ist ein Monoid*
2. $\forall x \exists x^{-1} : x \circ x^{-1} = x^{-1} \circ x = 1$ *(Inverses Element)*

Definition 3.3 *Ring* $[R; +, -, 0, \circ, 1]$

1.	$[R; +, -, 0]$ *ist eine Gruppe*	*(additive Gruppe von R)*
2.	$[R; \circ, 1]$ *ist ein Monoid*	*(multiplikatives Monoid)*
3.	$a \circ (b + c) = a \circ b + a \circ c$	
	$(b + c) \circ a = b \circ a + c \circ a$	*(Distributivgesetze)*

3.1.1 Das Morphismen Konzept

Morphismen sind strukturerhaltende Abbildungen von einem algebraischen System zu einem anderen.

Definition 3.4 (Ähnliche Algebren) *A und A' heißen ähnlich, wenn sie sich entsprechende nullstellige, einstellige, zweistellige, ... Operationen haben.*

Notation: Die Mengen der Operationen werden mit Ω, Ω' bezeichnet.

Definition 3.5 (Morphismus) *$[A;\Omega]$ und $[A';\Omega']$ seien ähnliche Algebren. Eine Abbildung $\Phi : A \mapsto A'$ ist ein Morphismus (Homomorphismus) von $[A;\Omega]$ nach $[A';\Omega']$, wenn für alle $\omega \in \Omega$ und für alle $a_1, \ldots, a_n \in A$gilt:*

$$\Phi(\omega(a_1, \ldots, a_n)) = \omega'(\Phi(a_1), \ldots, \Phi(a_n))$$

wobei ω' die ω entsprechende Operation aus Ω' ist.

Die Abbildung erhält die sich entsprechenden Operationen in A und A'.

Definition 3.6 *Sei Φ ein Morphismus. Dann heißt*

- Φ *Isomorphismus* $:\Leftrightarrow \Phi$ *bijektiv*
- Φ *Epimorphismus* $:\Leftrightarrow \Phi$ *surjektiv*
- Φ *Monomorphismus* $:\Leftrightarrow \Phi$ *injektiv*

3.1.2 Eigenschaften von Morphismen

Definition 3.7 (Unteralgebra) *Eine Teilmenge A' der Ω-Algebra A heißt Unteralgebra von A ($A' \leq A$) genau dann, wenn A' unter allen Operationen $\omega \in \Omega$ abgeschlossen ist.*

Satz 3.1 *Seien A und A' Ω-Algebren und $\Phi : A \mapsto A'$ ein Morphismus. Dann gilt:*

1. $B \leq A \Rightarrow \Phi(B) \leq A'$
2. $C \leq A' \Rightarrow \Phi^{-1}(C) \leq A$.

Morphismen erhalten Unteralgebren (Sie bilden Unteralgebren auf Unteralgebren ab, sowohl in die eine als auch in die andere Richtung).

Definition 3.8 *$H \subseteq A$: $[H]$ ist die kleinste Unteralgebra von A, die H enthält; wenn $[H] = A$ dann ist A durch H erzeugt; H heißt Menge der Erzeugenden.*

Satz 3.2 *Seien A und A' Ω-Algebren, $\Phi : A \mapsto A'$ ein Morphismus und H Unteralgebra von A. Dann ist*

$$\Phi([H]) = [\Phi(H)]$$

Ein Morphismus erhält Erzeugende: Er bildet eine Menge von Erzeugenden einer Unteralgebra auf eine Menge von Erzeugenden der Bild-Unteralgebra ab.

Satz 3.3 *$\Phi, \Psi : A \mapsto A'$; H eine Menge von Erzeugenden von A.*

$$\Phi(x) = \Psi(x) \text{ für alle } x \in H \Rightarrow \Phi = \Psi$$

Ein Morphismus ist eindeutig durch die Bilder einer beliebigen Menge von Erzeugenden bestimmt.

3.2 Ringe, Integritätsbereiche, Körper

3.2.1 Ringe

Beispiele:

- $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}\ (+, -, 0, \cdot, 1)$
- $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$
- $R[[x]]$: formale Potenzreihen
  $\sum_{i=0}^{\infty} a_i x^i, a_i \in R$ (kommutativer Ring)
- $R[x]$: Polynomring
  $\sum_{i=0}^{n} a_i x^i, a_i \in R$ (kommutativer Ring)

Definition 3.9 (Charakteristik) *Die Charakteristik eines Ringes R, char R, ist die Ordnung von 1 in der additiven Gruppe von R.*

1. *char $R = m$ (endlich) wenn*
   (a) $m \circ 1 = \underbrace{1 + \cdots + 1}_{m-mal} = 0$
   (b) $n \circ 1 \neq 0$ *für* $1 \leq n < m$
2. *Falls es kein solches $m < \infty$ gibt, ist char $R = 0$ (per Definition).*

3.2.2 Ring-Morphismen

Wir betrachten die Ringe $[R; +, 0, \circ, 1]$ und $[R'; \oplus, 0', \otimes, 1']$.

$\Phi : R \mapsto R'$ ist ein Ring-Morphismus, wenn

1. $\Phi(a + b) = \Phi(a) \oplus \Phi(b)$
2. $\Phi(a \circ b) = \Phi(a) \otimes \Phi(b)$
3. $\Phi(1) = 1'$

Das homomorphe Bild eines (kommutativen) Ringes ist ein (kommutativer) Ring.

3.2.3 Integritätsbereiche, Körper

R sei ein kommutativer Ring.

Definition 3.10 (Nullteiler) *$a \neq 0 \in R$ ist ein Nullteiler, falls es ein $b \neq 0 \in R$ gibt mit $a \circ b = 0$.*

Definition 3.11 (Einheit) *$u \neq 0 \in R$ ist eine Einheit in R, falls u invertierbar ist, d.h. es gibt $v \in R$ mit $u \circ v = 1 (v = u^{-1})$.*
$U(R)$ ist die Menge aller Einheiten in einem Ring R.

Bemerkungen: $U(R)$ ist eine multiplikative Gruppe.
Ein Ringelement kann nicht gleichzeitig ein Nullteiler und eine Einheit sein.

Definition 3.12 (Integritätsbereich) *Ein Integritätsbereich ist ein nicht trivialer kommutativer Ring, der keine Nullteiler besitzt.*

Definition 3.13 (Körper) *Ein Körper ist ein nicht trivialer kommutativer Ring, in dem jedes Element $\neq 0$ eine Einheit ist.*

Ein Körper ist ein Integritätsbereich.
In einem Körper können wir addieren, subtrahieren, multiplizieren und durch Elemente $\neq 0$ dividieren.

Beispiele:

- Integritätsbereich: $\mathbb{Z}$(kein Körper, da ± 1 die einzigen Einheiten sind)
- Körper: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$

Satz 3.4 *1. p prim $\Rightarrow Z_p$ ist ein Körper*

2. m nicht prim $\Rightarrow Z_m$ ist ein Ring mit Nullteilern

Korollar 3.5 *1. Ein endlicher Integritätsbereich ist ein Körper.*

2. Falls ein Integritätsbereich eine Charakteristik $m > 0$ hat dann ist m prim.

3.2.4 Quotientenkörper

Sei D ein Integritätsbereich. Gesucht ist ein Körper F mit $D \subseteq F$

Definition 3.14 (Quotientenkörper) *Der kleinste Körper F mit $D \subseteq F$ heißt Quotientenkörper und wird mit $Q(D)$ bezeichnet.*

Bemerkung: Wenn D endlich, dann ist $Q(D) = D$. Für unendliches D gilt folgender Satz:

Satz 3.6 *$Q(D)$ kann konstruiert werden.*

$D^* := D \setminus \{0\}$. Analog zur Konstruktion der rationalen Zahlen als Quotient der ganzen Zahlen definieren wir die Relation $\sim$ auf $D \times D^*$ mit

$$(a, b) \sim (c, d) :\Leftrightarrow ad = bc$$

Wir setzen:

$$(a, b) \oplus (c, d) = (ad + cb, cd)$$
$$(a, b) \otimes (c, d) = (ac, bd)$$

$Q(D) = (D \times D^*)/\sim$ ist der kleinste Körper, der D enthält.

Proposition 3.2.1 *$D \cong D' \Rightarrow Q(D) \cong Q(D')$*

3.2.5 Polynome und formale Potenzreihen

R sei ein kommutativer Ring.

Definition 3.15 (Polynome)

$$R[x] := \{a(x) = \sum_{i=0}^{n} a_i x^i, a_i \in R, n \in N\}$$

Sei $a(x) = a_n x^n + \cdots + a_0$ *aus* $R[x], (a_n \neq 0)$. *Dann heißt* $\deg(a(x)) := n, \deg(0) := -\infty$ *der Grad des Polynoms,* $\operatorname{ldcf}(a(x)) := a_n$ *der führende Koeffizient und* a_0 *der konstante Term. Ein Polynom heißt normiert, falls* $a_n = 1$.

Definition 3.16 (Formale Potenzreihen(FPR))

$$R[[x]] := \{a(x) = \sum_{i=0}^{\infty} a_i x^i, a_i \in R\}$$

Sei $a(x) = a_n x^n + a_{n+1} x^{n+1} + \cdots$ *aus* $R[[x]], (a_n \neq 0)$. *Dann heißt* $\operatorname{ord}(a(x)) := n$, $\operatorname{ord}(0) := \infty$ *die Ordnung der FPR,* a_n *der Koeffizient der niedersten Ordnung und* a_0 *konstanter Term.*

Eigenschaften:

- D Integritätsbereich $\Rightarrow D[x]$ und $D[[x]]$ Integritätsbereiche.
- $a(x)$ ist Einheit in $D[x] \Leftrightarrow a(x) = a_0 \in D$ und a_0 ist eine Einheit.
- Die Einheiten von $F[x]$ (F ein Körper) sind die konstanten Polynome, die $\neq 0$ sind.
- $a(x) = a_0 + a_1 x + \cdots$ ist eine Einheit in $D[[x]] \Leftrightarrow a_0$ ist eine Einheit in D.
- $a(x) = a_0 + a_1 x + \cdots$ ist eine Einheit in $F[[x]] \Leftrightarrow a_0 \neq 0$

Bemerkung: $F[[x]]$ ist kein Körper. Wenn man jedoch die Menge $F\langle x\rangle$ der erweiterten formalen Potenzreihen betrachtet:

$$F\langle x\rangle := \{\sum_{i=-\infty}^{\infty} a_i x^i, a_i \in F, a_i \neq 0 \text{ für nur endlich viele } i < 0\}$$

so ist $F\langle x\rangle$ ein Körper und $F\langle x\rangle$ enthält $F[[x]]$ als Unterring.

Teilbarkeitseigenschaften von Polynomen

R sei ein beliebiger kommutativer Ring.
Seien $a(x), b(x) \in R[x]$, mit $b_n = \mathrm{ldcf}(b(x))$ eine Einheit in R, dann gibt es zwei eindeutig bestimmt Polynome $q(x)$ und $r(x)$ aus $R[x]$, so daß

$$a(x) = b(x) \cdot \underbrace{q(x)}_{Quotient} + \underbrace{r(x)}_{Rest}, \deg r(x) < \deg b(x)$$

Bemerkung: Falls $\deg(b(x)) > \deg(a(x))$ dann ist $q(x) = 0, r(x) = a(x)$.

Reste und Kongruenzen mod m(x)

Sei $m(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_0$ ein normiertes Polynom aus $R[x]$. Für alle $a(x)$ aus $R[x]$ gibt es $q(x)$ und $r(x)$ mit

$$a(x) = m(x)q(x) + r(x)(\deg r < \deg m)$$

Notation: $r_{m(x)}(a(x))$: Rest von $a(x)$ mod $m(x)$

Man kann nun die Äquivalenzrelation „mod $m(x)$" definieren:

$$a \equiv_{m(x)} b(x) :\Leftrightarrow m(x)|(a(x) - b(x))$$

(d.h.: $a(x)$ und $b(x)$ haben den selben Rest modulo $m(x)$)

Aus dieser Formulierung für Polynome ergibt sich:
Viele Eigenschaften von Polynomen (betrachtet als Funktionen) sind eng an die Eigenschaften der Koeffizienten gebunden (z.B. keine Nullteiler).

Beispiele:

- Wurzeln
- Reste $(x - \alpha)|a(x) \mapsto a(\alpha) = 0$
- deg $(a(x)) = n \Rightarrow a(x)$ hat höchstens n Wurzeln in D
- formale Ableitung
- vielfache Wurzeln
- Interpolation

3.2.6 Teilbarkeit im Integritätsbereich

Definition 3.17 (Teilbarkeit) *Für $a, b \in D : a|b :\Leftrightarrow$ es gibt ein $c \in D$ mit $b = a \cdot c$.*

Definition 3.18 (assoziiert) *$a, b \in D^*$ heißen assoziiert ($a \sim b$), falls $a = u \cdot b$, wobei u eine Einheit in D ist.*

Proposition 3.2.2

1. *$a \sim b \Leftrightarrow a|b$ und $b|a$.*
2. *$a \sim b, c \sim d \Rightarrow (a|c \Rightarrow b|d)$.*
3. *$\sim$ ist eine Äquivalenzrelation auf D^*.*

$\sim$ zerlegt D^* in Äquivalenzklassen von assoziierten Elementen.
Die $\sim$-Äquivalenzklasse, die a enthält, ist gegeben durch $[a] = \{ua | u \in U(D)\}$.

3.2.7 Euklidsche Integritätsbereiche

Ganze Zahlen und Polynome haben viele Ähnlichkeiten und gehören beide zur Kategorie der Euklidschen Integritätsbereiche.

Definition 3.19 (Euklidscher Integritätsbereich) *Ein Euklidscher Integritätsbereich ist ein Integritätsbereich D zusammen mit einer Funktion $d : D^* \mapsto N$ ($d(a)$ heißt der (Euklidsche) Grad von a), so daß gilt:*

1. *$d(ab) \geq d(a), (a, b \neq 0)$*
2. *Für alle a, b aus D mit $b \neq 0$ gibt es einen „Quotienten“ q und „Rest“ r in D, so daß $a = bq + r, d(r) < d(b)$ oder $r = 0$ (Teilbarkeitseigenschaft)*

Beispiele:
Euklidsche Integritätsbereiche:

1. $\mathbb{Z}$ mit $d(a) = |a|$.
2. F beliebiger Körper $d(a) = 1$, falls $a \neq 0$.
3. $F[x] d(a) = \deg a(x)$.

Nicht Euklidsche Integritätsbereiche:

1. $\mathbb{Q}$ mit $d(a) = |a|$, (denn $d(2 \circ \frac{1}{2}) = d(1) < d(2) = 2$).
2. $\mathbb{Z}[x]$ mit $d(a) = \deg(a(x))$ (Da die Koeffizienten des Quotienten nicht mehr in $\mathbb{Z}$ liegen).

3.2.8 Größter Gemeinsamer Teiler

Definition 3.20 (ggT) *Der ggT von $a, b \in D$, beide $\neq 0$, D Integritätsbereich, ist $g \in D$, so daß:*

1. *$g|a$ und $g|b$ (g ist gemeinsamer Teiler)*
2. *$c|a$ und $c|b \Rightarrow c|g$ (größter gemeinsamer Teiler)*

In Euklidschen Integritätsbereichen ist der ggT bis auf Assoziierte eindeutig bestimmt.

Satz 3.7 *Ein Euklidscher Integritätsbereich ist ein ggT-Bereich (d.h. der ggT existiert immer).*

Satz 3.8 *D sei ein Euklidscher Integritätsbereich, $a, b \in D(\neq 0)$. Dann besitzen a und b einen ggT g, der sich als Linearkombination von a und b darstellen läßt:*

$$g = sa + tb \text{ mit } s, t \in D$$

Satz 3.9 (Primfaktorzerlegung in Euklidschen Integritätsbereichen) *Jeder Euklidsche Integritätsbereich ist ein Ring mit eindeutiger (bis auf assoziierte) Primfaktorzerlegung, d.h. jede nicht Einheit läßt sich als endliches Produkt von Primelementen darstellen.*

3.3 Quotienten - Algebra

Ein fundamentales Problem in der Algebra: Finde alle homomorphen Bilder (einfacherer Versionen) einer gegebenen Ω-Algebra.

Quotienten-Algebra: Äquivalenzklassen einer Ω-Algebra modulo einer Äquivalenzrelation E (z.B.: eine Kongruenzrelation): A/E

3.3.1 Kongruenzrelationen

Definition 3.21 (Kongruenzrelation) *Eine Kongruenzrelation E auf einer Ω-Algebra ist eine Äquivalenzrelation auf A, die folgende Ersetzungseigenschaften besitzt: Für alle $\omega \in \Omega$ gilt:*

$$a_i E b_i (i = 1, \ldots, n) \Rightarrow \omega(a_1, \ldots, a_n) E \omega(b_1, \ldots, b_n)$$

Wenn die Operanden äquivalent sind, so sind es auch die Bilder.

Beispiel: $\mathbb{Z}_m$: die Kongruenzrelation mod m kann nun folgendermaßen auf einen beliebigen Ring R verallgemeinert werden:

$$a \equiv_m b \Leftrightarrow a - b = km, k \in R.$$

3.3.2 Homomorphiesätze für universelle Algebren

Kongruenzrelationen und homomorphe Bilder sind verschiedene Aspekte einer Idee: Abstraktion.

Satz 3.10 (Quotienten-Algebren) *A sei eine Ω-Algebra und E eine Kongruenzrelation auf A. Dann gilt:*

1. *Die Quotientenmenge A/E ist eine Ω-Algebra, indem man für $\omega \in \Omega$ auf A/E definiert:*
$$\omega([a_1], \ldots, [a_n]) := [\omega(a_1, \ldots, a_n)].$$
2. *A/E ist ein homomorphes Bild von A unter dem natürlichen Homomorphismus $\nu : a \mapsto [a]$.*

Lemma 3.11 *$\Phi : A \mapsto A'$ sei ein Ω-Algebren-Morphismus, dann ist*

$$aE_\Phi b :\Leftrightarrow \Phi(a) = \Phi(b)$$

eine Kongruenzrelation.

Notation: E_Φ wird oft als „Kern-Relation“ bezeichnet. ($aE_\Phi b \Leftrightarrow a - b \in$ Kern (Φ))

Satz 3.12 (Isomorphiesätze) *A und A' seien Ω-Algebren, A' ein homomorphes Bild von A unter $\Phi : A \mapsto A'$. Dann ist $A' \cong A/E_\Phi$, wobei $\Psi : A/E_\Phi \mapsto A'$ mit $[a] \mapsto \Phi(a)$ ein Isomorphismus ist.*

$$\begin{tikzcd}
A \arrow[rr, "\Phi"] \arrow[d, "\nu : a \mapsto {[a]}"'] & & A' \\
A/E_\Phi \arrow[urr, "\Psi : {[a]} \mapsto \Phi(a)"'] & &
\end{tikzcd}$$

Beispiel: $r_m : \mathbb{Z} \mapsto \mathbb{Z}_m$ mit $a \mapsto r_m(a)$ ist ein Ringepimorphismus. Für die „Kern-Äquivalenzrelation“ E_r gilt:

$$aE_r b \Leftrightarrow r_m(a) = r_m(b) \Leftrightarrow a \equiv_m b$$

also: $E_r = \equiv_m$, d.h. $Z/\equiv_m$ ist ein Quotientenalgebra von $\mathbb{Z}_m$

$$\begin{tikzcd}
\mathbb{Z} \arrow[r, "r_m"] \arrow[d, "\nu : a \mapsto {[a]}"'] & \mathbb{Z}_m \\
\mathbb{Z}/\equiv_m \arrow[ur, "\Psi : {[a]} \mapsto r_m(a)"'] &
\end{tikzcd}$$

3.4 Quotientenringe

3.4.1 Ideale und Quotienten-Ringe

Definition 3.22 *Ein Ideal I in einem Ring R ist eine nicht leere Teilmenge von R mit:*

1. $a, b \in I \Rightarrow a - b \in I$
2. $a \in I,\ r \in R \Rightarrow ar, ra \in I$

Bemerkungen: I ist eine additive Untergruppe von R.
Die Nebenklassen von I in R („$R \bmod I$“) bilden eine Partition von R mit

$$a + I = b + I \Leftrightarrow a - b \in I \Rightarrow aE_I b \Leftrightarrow a - b \in I.$$

R/I und R/E_I sind gleich, d.h. die Nebenklassen $a + I$ stimmen mit den Äquivalenzklassen $[a]_{E_I}$ überein.

Notation: $a \equiv b \pmod{I} :\Leftrightarrow aE_I b$

Satz 3.13 (Quotientenringe) *I sei ein Ideal in einem Ring R. Dann gilt:*

1. *Die Menge $R/I = \{a + I | a \in R\}$ der additiven Nebenklassen von I in R ist eine Quotientenalgebra (Quotientenring) mit den Operationen:*

$$(a + I) + (b + I) = (a + b) + I$$

$$-(a + I) = -a + I$$

$$0_{R/I} = 0 + I = I$$

$$(a + I)(b + I) = ab + I$$

$$1_{R/I} = 1 + I$$

2. *R/I ist ein homomorphes Bild von R unter dem natürlichen Homomorphismus: $a \mapsto a + I$.*

3. *R/I ist ein Ring (kommutativ, falls R es ist).*

Anwendungen:

- Quotientenring $\mathbb{Z}/(m)$
  $a + (m) = b + (m) \Leftrightarrow a \equiv_m b$
  $\mathbb{Z}/(m)$ und $\mathbb{Z}/\equiv_m$ stimmen überein
  Die Nebenklassen $a + (m) \in Z/(m)$ bilden Äquivalenzklassen.

- Quotientenring $R[x]/m(x)$

3.4.2 Hauptideale

R sei ein kommutativer Ring, $m \in R$. Dann ist die Menge aller Vielfachen von m in $R : (m) = \{km | k \in R\}$:

1. ein Ideal

2. das kleinste Ideal von R, das m enthält

(m) ist das Hauptideal, das durch m erzeugt wird.

3.4.3 Hauptideal-Integritätsbereiche

Definition 3.23 (Hauptideal-Integritätsbereiche (PID)) *Ein PID ist ein Integritätsbereich, in dem jedes Ideal ein Hauptideal ist.*

Falls D ein PID ist, $I \subseteq D \Rightarrow I = (a)$ mit $a \in D$

Satz 3.14 *Ein Euklidscher Integritätsbereich ist ein PID.*

Wie Euklidsche Integritätsbereiche haben auch PIDs die ggT-Eigenschaft:

$$a, b \in D \Rightarrow \text{ ggT } = sa + tb \text{ mit } s, t \in D$$

3.4.4 Euklidscher Algorithmus

Ein Euklidscher Integritätsbereich ist ein ggT-Bereich. Zu beliebigen a, b aus $D(a, b \neq 0)$ gibt es einen ggT g, der sich folgendermaßen darstellen läßt:

$g = sa + tb$, mit s, t aus D

Ziel:

1. Herleitung des Euklidschen Algorithmus, um g zu berechnen.
2. Herleitung des erweiterten Euklidschen Algorithmus, um auch s und t berechnen zu können.

Wir benötigen einige Eigenschaften des ggT.

Proposition 3.4.1 *In einem Euklidschen Integritätsbereich D gilt:*

1. ggT $(a, 0) \sim a$
2. *Falls $b \neq 0$ ist, gilt* ggT $(a, b) =$ ggT $(b, a \bmod b)$.

($\sim$ bedeutet hier: „a“ ist ein ggT von „a“ und 0)

GGT $(a, b; g)$

1. Initialisiere a_0, a_1
   $(a_0, a_1) := (a, b);$

2. Berechne den ggT (in einer Schleife)
   while $a_1 \neq 0$ **do**
   $\quad (a_0, a_1) := (a_1, a_0 \bmod a_1);$

3. Resultat
   $g := a_0;$

Beispiel: $\mathbb{Z}, a = 228, b = 612$

Iteration	a_0	a_1
0	228	612
1	612	228
2	228	156
3	156	72
4	72	12
5	12	0

$g = 12$

Beispiel: $\mathbb{Z}_5[x], a(x) = x^3 - 2, b(x) = 2x^2 - 3$

Iteration	a_0	a_1
0	$x^3 + 3$	$2x^2 + 2$
1	$2x^2 + 2$	$4x + 3$
2	$4x + 3$	0

$g(x) = 4x + 3 \sim x + 2$

Die Grundidee beim erweiterten Euklidsche Algorithmus ist die gleiche.
Wir betrachten $g = sa + bt$, führen neue Variablen ein $(a_0, a_1), (t_0, t_1), (s_0, s_1)$ und behaupten, daß folgende Relationen innerhalb der Schleife bestehen bleiben:

$$a_0 = s_0 a + t_0 b, \qquad a_1 = s_1 a + t_1 b$$

EUKLID $(a, b; g, s, t)$

1. Initialisiere
   $(a_0, a_1) := (a, b);$
   $(t_0, t_1) := (1, 0);$
   $(s_0, s_1) := (0, 1);$

2. Berechne
   while $a_1 \neq 0$ **do**
   $\quad q := a_0 \operatorname{div} a_1;$
   $\quad (a_0, a_1) := (a_1, a_0 - a_1 q);$
   $\quad (s_0, s_1) := (s_1, s_0 - s_1 q);$
   $\quad (t_0, t_1) := (t_1, t_0 - t_1 q);$

3. Resultat
   $g := a_0;$
   $s := s_0;$
   $t := t_0;$

Beispiel: $\mathbb{Z}, a = 228, b = 612$

Iteration	q	a_0	a_1	s_0	s_1	t_0	t_1
0	–	228	612	1	0	0	1
1	0	612	228	0	1	1	0
2	2	228	156	1	-2	0	1
3	1	156	72	-2	3	1	-1
4	2	72	12	3	-8	-1	3
5	6	12	0	-8	51	3	-19

$12 = -8 \cdot 228 + 3 \cdot 612$

Beispiel: $\mathbb{Z}_5[x], a(x) = x^3 + 3, b(x) = 2x^2 + 2$

Iteration	q	a_0	a_1	s_0	s_1	t_0	t_1
0	–	$x^3 + 3$	$2x^2 + 2$	1	0	0	1
1	$3x$	$2x^2 + 2$	$4x + 3$	0	1	1	$2x$
2	$3x + 4$	$4x + 3$	0	1	$2x + 1$	$2x$	$4x^2 + 2x + 1$

$4x + 3 = 1(x^3 + 3) + (2x)(2x^2 + 2)$

Der Euklidsche Algorithmus ist eine Anwendung der Arithmetik in Euklidschen Integritätsbereichen. Ein weiteres mächtiges Werkzeug ist die Berechnung durch homomorphe Bilder. Ein Prototyp hiervon ist der Algorithmus des Chinesischen Restsatzes.

Problem: Man finde die Lösungen eines Systems von n linearen Kongruenzen $u \equiv r_k \bmod m_k (k = 0, \ldots, n-1)$ über einem EuklidschenIntegritätsbereich. u läßt sich darstellen als: $u = s_0 + s_1 m_0 + s_2 m_0 m_1 + s_3 m_0 m_1 m_2$ Der Chinesische Restsatz-Algorithmus über einem Euklidschen Integritätsbereich:

CRA $(n, r_0, \ldots, r_{n-1}, m_0, \ldots, m_{n-1}; u)$

$M := 1;$
$u := r_0 \bmod m_0;$
for $k := 1$ **to** $n - 1$ **do**
   $M := M \cdot m_{k-1};$
   INVERSE $(M, m_k; c);$
   $s := [(r_k - (u \bmod m_k)) \cdot c] \bmod m_k;$
   $u := u + s \cdot M;$

Dabei läßt sich die Inverse mit dem erweiterten Euklidschen Algorithmus bestimmen:

INVERSE $(a, m; a') \equiv$ **EUKLID** $(a, m; \_, a', \_)$.

Beispiel:

$$\begin{array}{lcll} u & \equiv & 1 & \pmod 3 \\ u & \equiv & 3 & \pmod 5 \\ u & \equiv & 0 & \pmod 7 \\ u & \equiv & 10 & \pmod{11} \end{array}$$

k	r_k	m_k	M	c	s	u
0	1	3	1	–	–	1
1	3	5	3	2	4	13
2	0	7	15	1	1	28
3	10	11	105	2	8	868

$u = 868$ ist eine Lösung.

3.5 Körpertheorie

Notation: Im folgenden wird mit F der „Grundkörper“ und mit E der „Erweiterungskörper“ bezeichnet ($E \geq F$).

3.5.1 Körpererweiterung

Das Problem der Adjunktion einer Nullstelle: F sei ein Körper und $m(x)$ ein über F irreduzibles Polynom. Wie erweitert man F zu einem Körper, in dem $m(x)$ eine Nullstelle besitzt?

Beispiel: $p(x) = x^3 - 2$ ist ein Polynom über $\mathbb{Q}$.

In $\mathbb{Q}$ besitzt $p(x)$ keine Nullstelle, also hat $p(x)$ keinen linearen Faktor über $\mathbb{Q}$. Wenn man $\mathbb{Q}$ zu $\mathbb{R}$ erweitert ist $\sqrt[3]{2}$ eine Wurzel von $p(x)$.

Satz 3.15 *$m(x)$ sei ein über F irreduzibles Polynom (aus $F[x]$). Dann gilt: $E = F[x]/(m(x))$ ist eine Erweiterung von F, in der $m(x)$ ein Nullstelle besitzt.*

Beweis: (Kronecker) Theorie der Quotientenringe.

Satz 3.16 *Sei $f(x)$ ein Polynom vom Grad n über einem Körper F. F kann zu einem Körper E erweitert werden, in dem $f(x)$ n Nullstellen besitzt.*

Beweis: Induktion über n

Definition 3.24 (Zerfällungskörper) *Ein Zerfällungskörper über P von einem Polynom $f(x) \in F[x]$ ist ein Körper $E \geq F$, so daß:*

1. *alle Nullstellen von $f(x)$ in E liegen.*
2. *E ist die kleinste Erweiterung von F, für die (1.) gilt (d.h. kein echter Unterkörper von E ist eine Erweiterung von F, die alle Nullstellen von F enthält.)*

Satz 3.17 *Zu jedem Polynom $f(x) \in F[x]$ existiert ein Zerfällungskörper.*

3.5.2 Die Theorie der einfachen Körpererweiterungen

Definition 3.25 (algebraisch, transzendent) *Sei $\alpha \in E \geq F$. α heißt algebraisch über F, wenn α Nullstelle eines Polynoms über F ist: $f(\alpha) = 0$ für ein $f(x) \in F[x]$. Falls es kein solches $f(x)$ gibt, heißt α transzendent über F. $F(\alpha)$ heißt eine (einfache) algebraische oder transzendente Erweiterung von F.*

(einfach: es wird nur ein Element α zu F adjungiert, α heißt primitives Element von $F(\alpha)$.)

Beispiele: $F = \mathbb{Q}, E = \mathbb{C}$

- $\sqrt[3]{2}$ ist algebraisch, da eine Nullstelle von $x^3 - 2 \in \mathbb{Q}[x]$.
- $i = \sqrt{-1}$ ist algebraisch, da Nullstelle von $x^2 + 1 \in \mathbb{Q}[x]$.
- e, π sind transzendent über $\mathbb{Q}$ (aber algebraisch über $\mathbb{R}$).
  [e : Beweis von Hermite (1873), π : Beweis von Lindeman (1822)]

Definition 3.26 (Minimalpolynom) *$\alpha \in E$ sei algebraisch über F. Das Minimalpolynom von α über F, $m_\alpha(x)$, ist das normierte Polynom vom kleinsten Grad in $F[x]$, für das α eine Nullstelle ist.*

Proposition 3.5.1 (Eigenschaften des Minimalpolynoms) *$m_\alpha(x)$ sei das Minimalpolynom von $\alpha \in F$:*

1. *$m_\alpha(x)$ ist eindeutig bestimmt.*

2. *$m_\alpha(x)$ ist irreduzibel über F.*

3. *für alle $f(x) \in F[x]$ gilt: $f(\alpha) = 0 \Rightarrow m_\alpha(x) | f(x)$.*

Proposition 3.5.2 (Charakterisierung des Minimalpolynoms) *Wenn $\alpha \in E$ Nullstelle eines normierten irreduziblen Polynoms $m(x)$ über F ist, dann ist $m(x)$ das Minimalpolynom von α über F.*

Satz 3.18 *Sei $\alpha \in E$ algebraisch über $F \leq E$ mit Minimalpolynom $m_\alpha(x)$ über F, dann gilt:*

$$F(\alpha) \cong F[x]/(m_\alpha(x))$$

Daraus ergibt sich ein Zusammenhang zwischen algebraischen Erweiterungen und Isomorphismen.

3.5.3 Multiplikative Gruppe eines endlichen Körpers

Eine wichtige Eigenschaft von endlichen Körpern ist, daß ihre multiplikative Gruppe zyklisch ist.

Lemma 3.19 *Seien a und b Elemente einer abelschen Gruppe und $O(a) = m, O(b) = n$ mit $(m, n) = 1$, dann gilt: $O(ab) = mn$.*

Satz 3.20 *G sei eine endliche abelsche Gruppe und m die Ordnung eines Elements maximaler Ordnung in G, dann gilt für alle $g \in G : O(g)|m$.*

Satz 3.21 *E sei ein endlicher Körper, dann ist E^*, die multiplikative Gruppe von E, zyklisch.*

(Zyklische Gruppe: G Gruppe, $a \in G : [a] = \{a^i | i \in \mathbb{Z}\} \Rightarrow [a]$ ist zyklische Untergruppe von G)

3.5.4 Endliche Körper und algebraische Erweiterungen

Problem: Beschreibung der Struktur einer beliebigen endlichen Erweiterung von einem gegebenen Grundkörper F.

Lemma 3.22 *E sei ein endlicher Körper mit n Elementen ($|E| = r$), dann ist jedes $\beta \in E^*$ eine Nullstelle von $(x^{n-1} - 1)$.*

Satz 3.23 *F sei ein endlicher Körper, E ein endlicher Erweiterungskörper von F, dann ist jedes $\beta \in E$ algebraisch über F.*

Satz 3.24 *F endlicher Körper, E endlicher Erweiterungskörper von $F \Rightarrow E$ ist algebraische Erweiterung von F.*

Satz 3.25 *F endlicher Körper, E endlicher Erweiterungskörper von $F \Rightarrow E \cong F[x]/m_\alpha(x)$, wobei $m_\alpha(x)$ das Minimalpolynom über F von einem primitiven Element $\alpha \in E$ ist.*

Korollar 3.26 *F sei ein endlicher Körper mit $|F| = q$ und E ein endlicher Erweiterungskörper von F, dann gilt $|E| = q^n$ für ein n aus $\mathbb{N}$.*

Korollar 3.27 *Jeder endliche Körper E der Charakteristik q besitzt $\mathbb{Z}_q$ als kleinsten Unterkörper ($\mathbb{Z}_q$ ist der Primkörper von E).*

Satz 3.28 *Jeder endliche Körper hat q^n Elemente mit q prim (die Charakteristik des Körpers) und einer ganzen Zahl n (der Grad des Minimalpolynoms über $\mathbb{Z}_q$ eines primitiven Elements des Körpers über $\mathbb{Z}_q$).*

Um das Minimalpolynom zu berechnen, benötigen wir folgende Informationen.

3.5.5 Eindeutigkeit und Existenz endlicher Körper

Lemma 3.29 *Seien $a_1(x), a_2(x), \ldots, a_L(x) \in F[x]$ paarweise verschiedene irreduzible Faktoren von $f(x) \in F[x]$, dann gilt:*

$$(\prod_{l=1}^{L} a_l(x)) | f(x)$$

Wir betrachten nun $GF(q^n)$ als Erweiterung von $GF(q)$ mit q prim.

Notation: $Q := q^n - 1$

Im folgenden soll die Faktorisierung von $x^Q - 1$ in irreduzible Faktoren über $GF(q^n)$ und $GF(q)$ untersucht werden.

Proposition 3.5.3 *Wenn $GF(q^n)$ die paarweise verschiedenen Elemente $\alpha_0 = 0, \alpha_1, \alpha_2, \ldots, \alpha_Q$ besitzt, dann besitzt $x^Q - 1$ über $GF(q)$ folgende Faktorisierung in irreduzible Polynome :*

$$x^Q - 1 = \prod_{i=1}^{Q}(x - \alpha_i)$$

Proposition 3.5.4 *Seien $m_1(x), \ldots, m_L(x)$ paarweise verschiedene Minimalpolynome über $GF(q)$ von den Elementen von $GF(q^n)^*$. Dann besitzt $x^Q - 1$ über $GF(q)$ folgende Faktorisierung in irreduziblen Polynomen:*

$$x^Q - 1 = \prod_{l=1}^{L} m_l(x)$$

Satz 3.30 (Eindeutigkeit von $GF(q^n)$) *Zwei beliebige Körper mit q^n Element sind zueinander isomorph.*

Der Beweis basiert auf der Eigenschaft der eindeutigen Faktorisierung von Euklidschen Integritätsbereichen.

Lemma 3.31 *In einem Integritätsbereich D mit $char(D) = q \neq 0$ (d.h. q ist Primzahl) gilt:*

$$(a + b)^q = a^q + b^q$$

Satz 3.32 (Existenz von $GF(q^n)$) *Für alle Primzahlen q und positive ganze Zahlen n gibt es einen Körper mit q^n Elementen.*

Satz 3.33 *In einem Galois-Körper der Charakteristik q besitzt jedes Element a genau eine q-te Wurzel $a^{1/q}$.*

Kapitel 4

Langzahlarithmetik

Bevor wir zur eigentlichen Arithmetik übergehen, kommt hier ein Einschub zum Thema Fourier-Transformation. Für eine ausführliche Diskussion siehe zum Beispiel [AHU74].

4.1 Diskrete Fourier-Transformation und Faltung

4.1.1 Die diskrete Fourier-Transformation und ihre Inverse

Die Fourier-Transformation wird meist über Körpern (gewöhnlich sogar über den komplexen Zahlen) betrachtet. Aus später ersichtlichen Gründen wird diese Sicht hier auf kommutative Ringe $(R, +, \cdot, 0, 1)$ mit Einselement (d.h. die Multiplikation ist kommutativ) erweitert. Zur schnellen Fourier-Transformation muß dann aber zusätzlich gefordert werden, daß die n-ten Einheitswurzeln in ihnen enthalten sind.

Definition 4.1
Ein Element $\omega \in R$ heißt **primitive n-te Einheitswurzel** *genau dann, wenn die Bedingungen*

(i) $\omega \neq 1$

(ii) $\omega^n = 1$

(iii) $\sum_{j=0}^{n-1} \omega^{jp} = 0$ *für* $1 \leq p < n$

erfüllt sind. Die Elemente $\omega^0, \omega^1, \ldots, \omega^{n-1}$ heißen n-te Einheitswurzeln.

Beispiele:

1. Im Körper $\mathbb{C}$, der komplexen Zahlen ist – wie in der Einführung erwähnt – $e^{2\pi i/N}$ eine primitive N-te Einheitswurzel.

2. In $\mathbb{Z}_{13}$ ist 8 eine primitive 4-te Einheitswurzel, was man leicht durch $8^2 = 12$, $8^3 = 5$ und $8^4 = 1$ nachrechnen kann.

Definition 4.2
Sei $\mathbf{a} = [a_0, a_1, ..., a_{n-1}]^T$ ein Spaltenvektor mit $a_i \in R$. Zusätzlich habe n ein multiplikativ inverses Element [1] *in R und es enthalte R eine primitive n-te Einheitswurzel ω. Weiter sei A eine $n \times n$-Matrix mit*

$$A[i,j] = \omega^{ij} \quad \textit{für } 0 \leq i, j < n.$$

Dann heißt der Vektor $F(\mathbf{a}) = A\mathbf{a}$, dessen i-te Komponente $b_i = \sum_{k=0}^{n-1} a_k \omega^{ik}$ $(0 \leq i < n)$ ist, die **diskrete Fourier-Transformierte** *von* **a** *und die zugehörige Abbildung diskrete Fourier-Transformation.*

Da die Matrix A nicht singulär ist, existiert auch deren Inverse A^{-1}. Die einfache Form dieser Umkehrmatrix wird im folgenden Lemma gegeben.

Lemma 4.3
Sei R ein kommutativer Ring (mit Einselement), der eine primitive n-te Einheitswurzel ω enthält und in dem n ein multiplikativ inverses Element besitzt. Sei A eine $n \times n$-Matrix mit $A[i,j] = \omega^{ij}$ für $0 \leq i, j < n$. Dann existiert A^{-1} und hat die Form

$$A^{-1} = \left(\frac{1}{n}\omega^{-ij}\right)_{0 \leq i,j < n}.$$

Beweis:
Sei δ_{ij} das Kroneckersymbol. Es genügt zu zeigen, daß $A \cdot A^{-1} = I_n$ ist, d.h. daß das ij-te Element in $A \cdot A^{-1}$ der Gleichung

$$\frac{1}{n}\sum_{k=0}^{n-1} \omega^{ik}\omega^{-kj} = \delta_{ij} \quad \text{für} \quad 0 \leq i, j < n \tag{4.1}$$

[1] Diese Forderung ist wichtig, um – wie in Lemma 4.3 zu sehen ist – die Existenz der inversen Fourier-Transformation zu sichern.

genügt.
Für $i = j$ vereinfacht sich die linke Seite von (4.1) auf

$$\frac{1}{n}\sum_{k=0}^{n-1}\omega^0 = 1.$$

Im Falle $i \neq j$ sei $q = i - j$. Dann vereinfacht sich die linke Seite von (4.1) auf

$$\frac{1}{n}\sum_{k=0}^{n-1}\omega^{qk}, \quad -n < q < n, \quad q \neq 0.$$

Für $q > 0$ erhält man

$$\frac{1}{n}\sum_{k=0}^{n-1}\omega^{qk} = 0,$$

da ω eine n-te Einheitswurzel ist. Ist $q < 0$, dann führt multiplizieren mit $\omega^{-q(n-1)}$, neuordnen der Terme in der Summe und das Ersetzen von q durch $-q$ auf

$$\frac{1}{n}\sum_{k=0}^{n-1}\omega^{qk}, \quad 0 < q < n,$$

was wiederum den Wert 0 hat, da ω eine n-te Einheitswurzel ist. Daraus folgt sofort (4.1). $\Box$

Der Vektor $F^{-1}(\mathbf{a}) = A^{-1}\mathbf{a}$, dessen i-te Komponente

$$\frac{1}{n}\sum_{k=0}^{n-1} a_k\omega^{-ik}, \quad 0 \leq i < n$$

ist, heißt die **inverse Fourier-Transformierte** von $\mathbf{a}$. Offensichtlich ist die inverse Transformation der Transformation von $\mathbf{a}$ wieder $\mathbf{a}$, d.h. es gilt

$$F^{-1}F(\mathbf{a}) = \mathbf{a}.$$

Wie bereits veranschaulicht steht die Fourier-Transformation in engem Zusammenhang mit der Polynomauswertung und der Interpolation. Hauptsächlich wird die Fourier-Transformation zur Faltung zweier Vektoren angewandt.

4.1.2 Polynominterpolation

Sei

$$p(x) = \sum_{i=0}^{n-1} a_i x^i$$

ein Polynom vom Grad $(n-1)$. Dieses Polynom kann auf zwei Arten eindeutig dargestellt werden. Entweder durch eine Liste von Koeffizienten $a_0, a_1, \ldots, a_{n-1}$ oder durch eine Liste von Werten an n verschiedenen Punkten $x_0, x_1, \ldots, x_{n-1}$. Das Berechnen der Koeffizienten-Darstellung eines Polynoms zu den Werten an den Stellen $x_0, x_1, \ldots, x_{n-1}$ heißt **Interpolation**.
Die Berechnung der Fourier-Transformierten eines Vektors $[a_0, a_1, ..., a_{n-1}]^T$ ist äquivalent zur Konvertierung der Koeffizienten-Darstellung des Polynoms $\sum_{i=0}^{n-1} a_i x^i$ in seine Werte-Darstellung an den Punkten $\omega^0, \omega^1, \ldots, \omega^{n-1}$. Ähnlich ist die inverse Fourier-Transformation äquivalent zur Polynominterpolation, wobei die Werte an den n-ten Einheitswurzeln gegeben sind.
Man könnte genauso eine Transformation definieren, die ein Polynom in einer Menge von anderen Punkten als den n-ten Einheitswurzeln auswertet. Beispielsweise könnte man die ganzen Zahlen $1, 2, \ldots, n$ wählen. Es zeigt sich jedoch, daß die Wahl von Potenzen von ω für die Auswertung und Interpolation geschickter ist. Im Kapitel 6 wird der Newtonsche Interpolationsalgorithmus zur Polynominterpolation vorgestellt. Mit der schnellen Fourier-Transformation kann diese beschleunigt werden.

4.1.3 Faltung

Zur Motivierung der Faltung betrachte man die Koeffizienten-Darstellung eines Polynoms. Das Produkt zweier Polynome vom Grad $(n-1)$

$$p(x) = \sum_{i=0}^{n-1} a_i x^i \quad \text{und} \quad q(x) = \sum_{j=0}^{n-1} b_j x^j$$

ist ein Polynom vom Grad $(2n-2)$

$$p(x)q(x) = \sum_{i=0}^{2n-2} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

Die Koeffizienten dieses Produktpolynoms sind genau die Komponenten der Faltung zweier Koeffizientenvektoren in der folgenden Definition, sofern man von der Komponente c_{2n-1} absieht, die Null ist.

Definition 4.4
Seien $\mathbf{a} = [a_0, \ldots, a_{n-1}]^T$ *und* $\mathbf{b} = [b_0, \ldots, b_{n-1}]^T$ *zwei Spaltenvektoren. Dann heißt der Vektor* $\mathbf{c} = [c_0, \ldots, c_{2n-1}]^T$ *mit*

$$c_i = \sum_{j=0}^{n-1} a_j b_{i-j} \qquad (a_k = b_k = 0 \;\; \textit{für} \;\; k < 0 \;\; \textit{oder} \;\; k \geq n)$$

die **Faltung** *von* $\mathbf{a}$ *und* $\mathbf{b}$ *und wird geschrieben als* $\mathbf{a} \star \mathbf{b}$.

Es ist also

$$c_o = a_0 b_0,\ c_1 = a_0 b_1 + a_1 b_0,\ c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0, \dots\ .$$

Man beachte, daß $c_{2n-1} = 0$ ist; dieser Term ist nur aus Symmetriegründen hinzugefügt.

Werden zwei Polynome $(n-1)$-ten Grades durch ihre Koeffizienten dargestellt, kann man deren Produkt in Koeffizienten-Darstellung durch die Faltung der beiden Koeffizienten-Vektoren berechnen. Sind dagegen $p(x)$ und $q(x)$ durch ihre Werte an den n-ten Einheitswurzeln dargestellt, genügt es jeweils die Wertepaare der entsprechenden Einheitswurzel zu multiplizieren, um deren Produkt zu berechnen. Diese Tatsache legt nahe, daß die Faltung zweier Vektoren $\mathbf{a}$ und $\mathbf{b}$ die inverse Transformation des komponentenweisen Produkts der beiden transformierten Vektoren ist; d.h. man kann eine Faltung auch durch die Fourier-Transformation, bilden des paarweisen Produkts und Invertierung berechnen. Das einzige Problem dabei ist, daß das Produkt zweier Polynome $(n-1)$-ten Grades im allgemeinen ein Polynom vom Grad $(2n-2)$ ist und zur Darstellung Werte an $2n-1$ verschiedenen Stellen erfordert. Diese technische Schwierigkeit wird im folgenden Satz dadurch gelöst, daß man die beiden Polynome als Polynome vom Grad $(2n-1)$ betrachtet und alle Koeffizienten von höheren Potenzen von x als $(n-1)$ auf Null setzt.

Satz 4.5 (Faltungssatz)
Seien

$$\mathbf{a} = [a_0, a_1, \dots, a_{n-1}, 0, \dots, 0]^T$$

und

$$\mathbf{b} = [b_0, b_1, \dots, b_{n-1}, 0, \dots, 0]^T$$

Spaltenvektoren der Länge $2n$,

$$F(\mathbf{a}) = [a'_0, a'_1, \dots, a'_{2n-1}]^T$$

und

$$F(\mathbf{b}) = [b'_0, b'_1, \dots, b'_{2n-1}]^T$$

ihre Fourier-Transformierten. Dann ist

$$\mathbf{a} \star \mathbf{b} = F^{-1}(F(\mathbf{a}) \cdot F(\mathbf{b})).$$

Beweis:
Da $a_i = b_i = 0$ für $n \leq i < 2n$, gilt für $0 \leq l < 2n$

$$a'_l = \sum_{j=0}^{n-1} a_j \omega^{lj} \qquad \text{und} \qquad b'_l = \sum_{k=0}^{n-1} b_k \omega^{lk},$$

also

$$a'_l b'_l = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} a_j b_k \omega^{l(j+k)}. \tag{4.2}$$

Sei $\mathbf{a} \star \mathbf{b} = [c_0, c_1, \ldots, c_{2n-1}]^T$ und $F(\mathbf{a} \star \mathbf{b}) = [c'_0, c'_1, \ldots, c'_{2n-1}]^T$.
Mit $c_p = \sum_{j=0}^{2n-1} a_j b_{p-j}$ gilt

$$c'_l = \sum_{p=0}^{2n-1} \sum_{j=0}^{2n-1} a_j b_{p-j} \omega^{lp}. \tag{4.3}$$

Das Vertauschen der Summen in (4.3) und substituieren von $p-j$ mit k führt auf

$$c'_l = \sum_{j=0}^{2n-1} \sum_{k=-j}^{2n-1-j} a_j b_k \omega^{l(j+k)}. \tag{4.4}$$

Da $b_k = 0$ für $k < 0$, kann man die Untergrenze der inneren Summe auf $k = 0$ setzen. Weiter kann man, da $a_j = 0$ für $j \geq n$, die Obergrenze der äußeren Summe auf $n-1$ herabsetzen. Die Obergrenze der inneren Summe ist dann mindestens n, unabhängig von j. Daher kann die Obergrenze durch $n-1$ ersetzt werden, weil $b_k = 0$ für $k \geq n$. Durch diese Abänderungen in (4.4) wird diese Gleichung identisch zu (4.2). Damit ist gezeigt, daß $F(\mathbf{a} \star \mathbf{b}) = F(\mathbf{a}) \cdot F(\mathbf{b})$ ist, woraus folgt, daß $\mathbf{a} \star \mathbf{b} = F^{-1}(F(\mathbf{a}) \cdot F(\mathbf{b}))$ gilt. $\square$

Die Faltung zweier Vektoren der Länge n ist ein Vektor der Länge $2n$. Dies erfordert das „Auffüllen" der beiden Vektoren $\mathbf{a}$ und $\mathbf{b}$ mit n Nullen im Faltungssatz. Um dies zu vermeiden, kann man *verhüllte Faltungen* benutzen.

4.1.3.1 Verhüllte Faltungen

Definition 4.6
Seien $\mathbf{a} = [a_0, \ldots, a_{n-1}]^T$ *und* $\mathbf{b} = [b_0, \ldots, b_{n-1}]^T$ *zwei Spaltenvektoren der Länge* n. *Dann heißt der Vektor*

- $\mathbf{c} = [c_0, \ldots, c_{n-1}]^T$ *mit* $c_i = \sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{n-1} a_j b_{n+i-j}$

  die **positiv verhüllte Faltung**

- $\mathbf{d} = [d_0, \ldots, d_{n-1}]^T$ *mit* $d_i = \sum_{j=0}^{i} a_j b_{i-j} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}$

  die **negativ verhüllte Faltung**

von $\mathbf{a}$ *und* $\mathbf{b}$.

Die verhüllten Faltungen werden im Kapitel 4 im Schönhage-Strassen Algorithmus zur schnellen Multiplikation ganzer Zahlen angewandt. Definition 4.6 führt auf den

Satz 4.7
Seien $\mathbf{a} = [a_0, \ldots, a_{n-1}]^T$ *und* $\mathbf{b} = [b_0, \ldots, b_{n-1}]^T$ *zwei Spaltenvektoren der Länge* n, ω *eine primitive* n*-te Einheitswurzel und* $\psi^2 = \omega$. *Zusätzlich habe* n *ein multiplikativ inverses Element.*

1. *Die positiv verhüllte Faltung von* $\mathbf{a}$ *und* $\mathbf{b}$ *ist durch*
$$F^{-1}(F(\mathbf{a}) \cdot F(\mathbf{b}))$$
gegeben.

2. *Sei* $\mathbf{d} = [d_0, d_1, ..., d_{n-1}]^T$ *die negativ verhüllte Faltung von* $\mathbf{a}$ *und* $\mathbf{b}$, $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$ *und* $\hat{\mathbf{d}}$ *seien jeweils* $[a_0, \psi a_1, \ldots, \psi^{n-1} a_{n-1}]^T$, $[b_0, \psi b_1, \ldots, \psi^{n-1} b_{n-1}]^T$ *und* $[d_0, \psi d_1, \ldots, \psi^{n-1} d_{n-1}]^T$. *Dann gilt*
$$\hat{\mathbf{d}} = F^{-1}(F(\hat{\mathbf{a}}) \cdot F(\hat{\mathbf{b}})).$$

Beweis:
Beachtet man, daß $\psi^n = -1$ ist, so verläuft der Beweis analog zum Beweis von Satz 4.5. □

4.2 Algorithmus zur schnellen Fourier-Transformation

Der Schlüssel zur Beschleunigung der Fourier-Transformation liegt in den beiden folgenden kleinen Beobachtungen über Einheitswurzeln.

Lemma 4.8
Sei $N = 2n$ *und* ω *eine primitive* N*-te Einheitswurzel. Dann gilt* $\omega^{k+n} = -\omega^k$ $(k = 0, \ldots, n-1)$.

Beweis:
Es gilt
$$(\omega^{k+n})^2 = (\omega^k)^2 \omega^N = (\omega^k)^2,$$
da ω eine N-te Einheitswurzel ist. Weiterhin ist $\omega^{k+n} \neq \omega^k$; sonst wäre $\omega^n = 1$, im Widerspruch dazu, daß ω eine *primitive* N-te Einheitswurzel ist. Also ist $\omega^{k+n} = -\omega^k$. □

Damit läßt sich das Problem der Berechnung der Fourier-Transformierten in zwei gleiche Teilprobleme zerlegen. Sei $[a_0, a_1, \ldots, a_{N-1}]^T$ der zu transformierende Spaltenvektor oder damit gleichbedeutend $a(x) = \sum_{i=0}^{N-1} a_i x^i$ das zu transformierende Polynom. Dieses kann man nun in

$$a(x) = b(y) + x\ c(y)$$

mit $\qquad y = x^2,\ b(y) = \sum_{i=0}^{n-1} a_{2i} y^i,\ c(y) = \sum_{i=0}^{n-1} a_{2i+1} y^i$

zerlegen und mit Lemma 4.8 an den N Potenzen von ω (den sogenannten Fourierpunkten) mit

$$\begin{aligned} a(\omega^k) &= b(\omega^{2k}) + c(\omega^{2k})\omega^k \\ a(\omega^{k+n}) &= b(\omega^{2k}) - c(\omega^{2k})\omega^k \end{aligned}$$

für $k = 0, \ldots, n-1$ auswerten.
Zur Übertragung dieser Zerlegung auf die beiden Teilprobleme benötigt man

Lemma 4.9
Sei $N = 2n$ und ω eine primitive N-te Einheitswurzel. Dann ist ω^2 eine primitive n-te Einheitswurzel.

Beweis:
Aus $(\omega^2)^n = \omega^N = 1$ folgt, daß ω^2 eine n-te Einheitswurzel ist. Weiter gilt $(\omega^2)^j \neq 1$ für $0 < j < n$. Sonst wäre $\omega^k = 1$ für ein k mit $0 < k < 2n = N$ im Widerspruch dazu, daß ω eine primitive N-te Einheitswurzel ist. Also ist ω^2 eine primitive n-te Einheitswurzel. $\square$

Vorausgesetzt $n = N/2$ ist gerade, kann die obige Zerlegung auf den beiden Teilproblemen $b(y)$ und $c(y)$ der Größe n an den n Fourierpunkten von ω^2 – was durch Lemma 4.9 sichergestellt ist – fortgesetzt werden, da nun wieder Lemma 4.8 erfüllt ist.
Dies kann man rekursiv fortsetzen, solange nur die Zahl der Auswertungspunkte gerade bleibt. Daher wählt man $N = 2^m$ (das m Rekursionen ermöglicht), um die Zerlegung solange durchzuführen bis das triviale Problem, die Auswertung eines Polynoms vom Grad 0, verbleibt.

Der folgende Algorithmus gilt in jedem kommutativen Ring $(R, +, \cdot, 0, 1)$, der die N-ten Einheitswurzeln enthält, zur Berechnung der Fourier-Transformierten $F(\mathbf{a})$ eines Vektors $\mathbf{a} = [a_0, a_1, \ldots, a_{N-1}]^T$.

rekursiver FFT-Algorithmus

Eingabe:
ganze Zahl $N = 2^m$,
$\mathbf{a} = [a_0, a_1, \ldots, a_{N-1}]^T$,
primitive N-te Einheitswurzel ω.

Ausgabe:
$\mathbf{A} = [A_0, A_1, \ldots, A_{N-1}]^T$ mit $A_k = \sum_{i=0}^{N-1} a_i(\omega^{-k})^i$

Hilfsdaten:
ganze Zahl $n = N/2$,
$\mathbf{b} = [b_0, b_1, \ldots, b_{n-1}]^T$, $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]^T$,
$\mathbf{B} = [B_0, B_1, \ldots, B_{n-1}]^T$, $\mathbf{C} = [C_0, C_1, \ldots, C_{n-1}]^T$.

```
procedure FFT(N, a, ω, A);

  if N = 1
    then
      { Basis }
        A_0 ← a_0
    else
    begin
      { Zerlegen }
        n ← N/2;
        b ← [a_0, ..., a_2i, ..., a_2n-2];
        c ← [a_1, ..., a_2i+1, ..., a_2n-1];
      { rekursive Aufrufe }
        FFT(n, b, ω^2, B);
        FFT(n, c, ω^2, C);
      { Kombinieren }
        for k ← 0 to n - 1 do
        begin
          A_k ← B_k + ω^k × C_k;
          A_k+n ← B_k - ω^k × C_k
        end
    end
```

Satz 4.10
Für $N = 2^m$, $\mathbf{a} = [a_0, a_1, \ldots, a_{N-1}]^T$ und einer primitiven N-ten Einheitswurzel ω über einem kommutativen Ring mit Einselement liefert die Prozedur FFT($N, \mathbf{a}, \omega, \mathbf{A}$) die Fourier-Transformierte $F(\mathbf{a}) = \mathbf{A}$.

Beweis:
Man nimmt $p(m)$ als Annahme von Satz 4.10. Dann ist der Korrektheitsbeweis der FFT gleichbedeutend mit dem Induktionsbeweis von $p(m)$ für alle $m \in \mathbb{N}$ (der hier übergangen wird). $\square$

Satz 4.11
Die Prozedur FFT benötigt

$$M(N) = (N/2)\log_2 N$$

Multiplikationen zur Berechnung der Fourier-Transformierten eines Vektors der Länge N.

Beweis:
Im 'else'-Teil der FFT-Prozedur werden

$$M(N) = 2M(N/2) + N/2$$

Multiplikationen benötigt ('$2M(N/2)$' entspricht den beiden rekursiven Aufrufen und '$N/2$' dem 'Kombinieren' Schritt), wodurch man mit $N = 2^m$ die Gleichung

$$M(2^m) = 2M(2^{m-1}) + 2^{m-1}$$

erhält. Iteriert man diese Gleichung m mal ergibt sich

$$M(2^m) = m2^{m-1} + M(1)2^m.$$

Es ist aber $M(1) = 0$ im 'then'-Teil (Basisfall) der FFT-Prozedur. Daher gilt $M(2^m) = 2M(2^{m-1})$ oder eben $M(N) = (N/2)\log_2 N$. □

Die schnelle inverse Fourier-Transformation ergibt sich nun direkt aus dem FFT-Algorithmus; denn ist ω eine primitive N-te Einheitswurzel, so auch trivialerweise $\omega^{-1} = \omega^{N-1}$ (Übung!). Es soll auch hier nochmal erwähnt sein, daß zusätzlich die Existenz von N^{-1} gefordert werden muß.

rekursiver InvFFT-Algorithmus

Eingabe:
ganze Zahl $N = 2^m$,
$\mathbf{A} = [A_0, A_1, \ldots, A_{N-1}]^T$,
primitive N-te Einheitswurzel ω.

Ausgabe:
$\mathbf{a} = [a_0, a_1, \ldots, a_{N-1}]^T$ mit $\sum_{i=0}^{N-1} a_i(\omega^k)^i = A_k$ $(k = 0, \ldots, N-1)$.

procedure InvFFT($N, \mathbf{A}, \omega, \mathbf{a}$);

begin
FFT($N, \mathbf{A}, \omega^{-1}, \mathbf{a}$)
$\mathbf{a} \leftarrow [N^{-1}a_0, N^{-1}a_1, \ldots, N^{-1}a_{N-1}]$
end

Es ist klar, daß auch dieser Algorithmus in $O(N \log N)$ Operationen ausgeführt wird.

Korollar 4.12
Die Faltung $\mathbf{a} \star \mathbf{b}$ *zweier Vektoren der Länge* N *kann in* $O(N \log N)$ *Operationen berechnet werden.*

Beweis:
Satz 4.5, 4.10 und 4.11. □

Korollar 4.13
Die positiv und negativ verhüllte Faltung von $\mathbf{a}$ *und* $\mathbf{b}$ *kann in* $O(N \log N)$ *Operationen berechnet werden.*

Bemerkung:
Satz 4.11 zeigt, daß durch die Teile-und-Herrsche Methode der rekursive FFT-Algorithmus asymptotisch um eine Größenordnung besser ist als die direkte Methode. Dennoch braucht diese Version – bedingt durch den Mehraufwand der rekursiven Aufrufe – nicht schneller zu sein. Eine wirklich schnellere iterative Version des FFT-Algorithmus ist ausführlich in „The Design and Analysis of Computer Algorithms“ von Aho, Hopcroft, Ullman beschrieben.

4.2.1 Exakte Faltung mit der FFT

In Anwendungen, in denen die Fourier-Transformation zur Vereinfachung der Berechnung der Faltung eingesetzt wird, ist meist ein exaktes Ergebnis notwendig. Man denke dabei beispielsweise an die Polynommultiplikation. Arbeitet man im Ring der reellen Zahlen, müssen diese auf Maschinenzahlen gerundet werden, was zwangsläufig auf Fehler führt. Diese Fehler kann man durch Rechnen in endlichen Körpern vermeiden. Die Schwierigkeit beim Benutzen endlicher Körper liegt darin, einen geeigneten Körper mit einer n-ten Einheitswurzel zu finden.
Man kann aber auch einen Ring $\mathbb{Z}_m$ benutzen und m so wählen, daß $\mathbb{Z}_m$ eine n-te Einheitswurzel ω besitzt. Es ist nicht von vorneherein klar, daß es zu einem gegebenen n ein ω und m derart gibt, daß ω eine n-te Einheitswurzel in $\mathbb{Z}_m$ ist. Ist n jedoch eine Potenz von 2, dann existiert immer ein geeignetes m. Im Satz 4.16 wird gezeigt, sind n und $\omega > 1$ Potenzen von 2, dann kann die Faltung in $\mathbb{Z}_{\omega^{n/2}+1}$ mit der Fourier-Transformation durchgeführt werden. Doch zuvor zwei vorbereitende Lemmata. In beiden sei $R = (S, +, \cdot, 0, 1)$ ein kommutativer Ring und $n = 2^k$, $k \geq 1$.

Lemma 4.14
Für alle $a \in S$ *gilt*

$$\sum_{i=0}^{n-1} a^i = \prod_{i=0}^{k-1} (1 + a^{2^i}).$$

Beweis:
Der Beweis ist eine Induktion über k. Der Fall $k = 1$ ist trivial.
Man beachte, daß

$$\sum_{i=0}^{n-1} a^i = (1+a) \sum_{i=0}^{n/2-1} (a^2)^i \tag{4.5}$$

gilt. Durch die Induktionsannahme und ersetzen von a durch a^2 erhält man

$$\sum_{i=0}^{n/2-1} (a^2)^i = \prod_{i=0}^{k-2} [1 + (a^2)^{2^i}] = \prod_{i=1}^{k-1} [1 + a^{2^i}]. \tag{4.6}$$

Die Behauptung ergibt sich durch Ersetzen der rechten Seite von (4.5) mit (4.6). □

Lemma 4.15
Sei $m = \omega^{n/2} + 1$ mit $\omega \in S$ und $\omega \neq 0$. Dann gilt

$$\sum_{i=0}^{n-1} \omega^{ip} \equiv 0 \bmod m \quad \textit{für} \quad 1 \leq p < n.$$

Beweis:
Nach Lemma 4.14 genügt es zu zeigen, daß $1 + \omega^{2^j p} \equiv 0 \bmod m$ für ein j mit $0 \leq j < k$. Sei $p = 2^s p'$, wobei p' ungerade. Sicherlich ist $0 \leq s < k$. Wählt man j so, daß $j + s = k - 1$, dann ist $1 + \omega^{2^j p} = 1 + \omega^{2^{k-1} p'} = 1 + (m-1)^{p'}$. Es ist aber $(m-1) \equiv -1 \bmod m$ und da p' ungerade, ist $(m-1)^{p'} \equiv -1 \bmod m$. Daraus folgt, daß $1 + \omega^{2^j p} \equiv 0 \bmod m$ für $j = k - 1 - s$. □

Satz 4.16
Seien n, ω positive Potenzen von 2 und $m = \omega^{n/2} + 1$. Dann besitzt n ein multiplikativ inverses Element modulo m in $\mathbb{Z}_m$ und ω ist in $\mathbb{Z}_m$ eine primitive n-te Einheitswurzel.

Beweis:
Da n eine Potenz von 2 und m ungerade ist, folgt, daß m und n teilerfremd sind. Also besitzt n ein multiplikativ inverses Element modulo m. [2] Aus $\omega \neq 1$, $\omega^n = \omega^{n/2}\omega^{n/2} \equiv (-1)(-1) = 1 \bmod (\omega^{n/2} + 1)$ folgt mit Lemma 4.15, daß ω eine primitive n-te Einheitswurzel in $\mathbb{Z}_m$ ist. □

Die Bedeutung dieses Satz liegt darin, daß er die Gültigkeit des Faltungssatzes auch für den Ring $\mathbb{Z}_{\omega^{n/2}+1}$ garantiert. Berechnet man die Faltung zweier Vektoren der Länge

[2] Dies ist ein Ergebnis aus der Zahlentheorie. Dort wird gezeigt, daß zu zwei teilerfremden Zahlen a und b zwei ganze Zahlen x und y existieren mit $ax + by = 1$. Dann ist $ax \equiv 1 \bmod b$. Setzt man $b = m$ und $a = n$ erhält man das gesuchte Ergebnis.

n mit ganzzahligen Komponenten und sind die Komponenten der Faltung aus dem Bereich von 0 bis $\omega^{n/2}$, dann hat man sicher die exakte Lösung erhalten. Liegen Komponenten der Faltung außerhalb dieses Bereiches, so sind sie korrekt modulo $\omega^{n/2} + 1$. Man kann aber trotzdem noch das exakte Ergebnis erzeugen, indem man mit mehreren verschiedenen ω's bzw. m's die Transformation wiederholt und darauf die chinesische Restetechnik (Kapitel 6) anwendet.

In diesem Kapitel werden Operationen auf Zahlen beliebiger Länge (d.h. unabhängig von irgendwelchen Registergrößen) betrachtet. Zur Vereinfachung wird hier mit ganzen Zahlen gearbeitet. Im folgenden wird eine ganze Zahl in der Darstellung zu einer Basis b ausgedrückt, wobei $b \geq 2$ eine feste ganze Zahl ist. Eine ganze Zahl a ist n-stellig in der Basis b Darstellung bzw. hat eine Länge $\leq n$, falls $a < b^n$ ist.

4.3 Die klassischen Algorithmen

In diesem Abschnitt werden Algorithmen zur

- Addition und Subtraktion zweier n-stelliger Zahlen mit einer n-stelligen Zahl und einem Übertrag als Ergebnis
- Multiplikation einer n-stelligen und einer m-stelligen Zahl mit einem $(m + n)$-stelligen Ergebnis
- Division einer $(m+n)$-stelligen Zahl durch eine n-stellige Zahl mit einem $(m+1)$-stelligen Teiler und einem n-stelligen Rest als Ergebnis

vorgestellt. Sie werden als die „klassischen Algorithmen“ bezeichnet, weil das Wort „Algorithmus“ jahrhundertelang nur in Verbindung mit diesen Verfahren gebraucht wurde.
Als bekannt vorausgesetzt werden die Grundoperationen

- Addition und Subtraktion zweier 1-stelliger Zahlen zu einer 1-stelligen Zahl und einem Übertrag als Ergebnis
- Multiplikation zweier 1-stelliger Zahlen zu einem 2-stelligen Ergebnis
- Division einer 2-stelligen Zahl durch eine 1-stellige Zahl zu einem 1-stelligen Teiler und Rest als Ergebnis.

Die Operationen beliebig langer Zahlen werden auf diese Grundoperationen zurückgeführt.

Der Additionsalgorithmus ist sehr einfach, aber es lohnt sich, diesen näher zu betrachten, da die gleichen Ideen in den anderen Algorithmen wieder auftreten.

Algorithmus A (Addition nicht negativer ganzer Zahlen)

Eingabe: Zwei nicht negative ganze Zahlen $(u_1 u_2 \ldots u_n)_b$ und $(v_1 v_2 \ldots v_n)_b$.

Ausgabe: $(w_0 w_1 w_2 \ldots w_n)_b$, wobei in w_0 der Übertrag steht.

A1. [Initialisierung]
Setze $j \leftarrow n$, $k \leftarrow 0$. (Die Variable j durchläuft alle Ziffern, k enthält den jeweiligen Übertrag.)

A2. [Addition der Ziffern]
Setze $w_j \leftarrow (u_j + v_j + k) \bmod b$ und $k \leftarrow \lfloor (u_j + v_j + k)/b \rfloor$. (D.h. k wird auf 1 oder 0 gesetzt, je nachdem, ob ein Übertrag $(u_j + v_j + k \geq b)$ auftritt oder nicht.)

A3. [Schleife über j]
Setze $j \leftarrow j - 1$. Falls $j > 0$ gehe zurück zu Schritt A2, andernfalls setze $w_0 \leftarrow k$ und terminiere.

Das Problem der Subtraktion ist dem der Addition sehr ähnlich, aber man sollte trotzdem die Unterschiede beachten.

Algorithmus S (Subtraktion nicht negativer ganzer Zahlen)

Eingabe: Zwei nicht negative ganze Zahlen $(u_1 u_2 \ldots u_n)_b \geq (v_1 v_2 \ldots v_n)_b$.

Ausgabe: nicht negative Differenz $(w_1 w_2 \ldots w_n)_b$.

S1. [Initialisierung]
Setze $j \leftarrow n$, $k \leftarrow 0$.

S2. [Subtraktion der Ziffern]
Setze $w_j \leftarrow (u_j - v_j + k) \bmod b$ und $k \leftarrow \lfloor (u_j - v_j + k)/b \rfloor$. (D.h. k wird auf -1 oder 0 gesetzt, je nachdem, ob ein Übertrag $(u_j - v_j + k < 0)$ auftritt oder nicht.)

S3. [Schleife über j]
Setze $j \leftarrow j-1$. Falls $j > 0$ gehe zurück zu Schritt S2, andernfalls terminiere. (Wenn der Algorithmus terminiert, gilt $k = 0$; $k = -1$ tritt genau dann auf wenn $v_1 \dots v_n > u_1 \dots u_n$ ist und dies wurde durch Voraussetzung ausgeschlossen.)

Das nächste Problem ist die Multiplikation. Hierbei werden die Ideen, die schon im Algorithmus A benutzt wurden, wieder verwendet.

Algorithmus M (Multiplikation nicht negativer ganzer Zahlen)

Eingabe: Zwei nicht negative ganze Zahlen $(u_1 u_2 \dots u_n)_b \geq (v_1 v_2 \dots v_n)_b$.

Ausgabe: das Produkt $(w_1 w_2 \dots w_{m+n})_b$. (Bei der herkömmlichen Papier und Bleistift Methode bildet man zuerst die partiellen Produkte $(u_1 u_2 \dots u_n) \times v_j$ für $1 \leq j \leq m$ und addiert dann diese – noch mit einem Skalierungsfaktor versehenen – Produkte; mit dem Computer dagegen ist es besser die Addition gleich mit der Multiplikation auszuführen.)

M1. [Initialisierung]
Setze $w_{m+1}, w_{m+2}, \dots, w_{m+n}$ auf Null und $j \leftarrow m$.

M2. [Multiplikation mit 0?]
Falls $v_j = 0$, setze $w_j \leftarrow 0$ und gehe zu Schritt M6.

M3. [Initialisiere i]
Setze $i \leftarrow n$, $k \leftarrow 0$.

M4. [Multiplikation und Addition]
Setze $t \leftarrow u_i \times v_j + w_{i+j} + k$, $w_{j+i} \leftarrow t \bmod b$ und $k \leftarrow \lfloor t/b \rfloor$. (Hier ist der Übertrag k immer im Bereich $0 \leq k < b$.)

M5. [Schleife über i]
Setze $i \leftarrow i-1$. Falls $i > 0$ gehe zurück nach Schritt M4, andernfalls setze $w_j \leftarrow k$.

M6. [Schleife über j]
Setze $j \leftarrow j-1$. Falls $j > 0$ gehe zurück nach Schritt M2, andernfalls terminiere.

Beispiel:
Multiplikation von $(914)_{10}$ mit $(84)_{10}$.

Schritt	i	j	u_i	v_j	t	w_1	w_2	w_3	w_4	w_5
M5	3	2	4	4	16	x	x	0	0	6
M5	2	2	1	4	05	x	x	0	5	6
M5	1	2	9	4	36	x	x	6	5	6
M6	0	2	x	4	36	x	3	6	5	6
M5	3	1	4	8	37	x	3	6	7	6
M5	2	1	1	8	17	x	3	7	7	6
M5	1	1	9	8	76	x	6	7	7	6
M6	0	1	x	8	76	7	6	7	7	6

Die Papier und Bleistift Methode zur Division benötigt ein gewisses „Raten“ der ausführenden Person. Zur Vermeidung des „Ratens“ ist ein genaueres Betrachten des Verfahrens notwendig.
Das Problem der Division einer $(m+n)$-stelligen Zahl durch eine n-stellige Zahl wird auf die Division einer $(n+1)$-stelligen Zahl u durch die n-stellige Zahl v zurückgeführt, wobei $0 \leq u/v < b$ ist. Der Rest r ist nach jedem Schritt kleiner als v, so daß $rb+$(nächste Stelle des Dividenden) als neues u für den folgenden Schritt genutzt werden kann. Will man beispielsweise 3142 durch 47 teilen, so teilt man erst 314 durch 47 und erhält 6 und den Rest 32. Dann teilt man 322 durch 47, erhält 6 und den Rest 40. Daher ist 66 der Teiler und der Rest ist 40.
Der Divisionsalgorithmus reduziert sich also auf folgendes Problem:

> Seien $u = (u_0u_1 \ldots u_n)_b$ und $v = (v_1v_2 \ldots v_n)_b$ nicht negative ganze Zahlen mit $u/v < b$. Finde einen Algorithmus zur Bestimmung von $q = \lfloor u/v \rfloor$. (Schreibt man $r = u - qv$, dann wird q durch $0 \leq r < v$ eindeutig bestimmt).

Die natürlichste Art, dieses Problem anzugehen, ist anhand der wichtigsten Ziffern von u und v eine Schätzung für q zu machen. Dazu nimmt man

$$\hat{q} = \min\left(\left\lfloor \frac{u_0b + u_1}{v_1} \right\rfloor, b-1\right).$$

Diese Formel sagt, man erhält $\hat{q}$ indem man die ersten beiden Ziffern von u durch die erste Ziffer von v teilt. Ist das Ergebnis größer oder gleich b, kann man es durch $(b-1)$ ersetzen. Daß diese Schätzung „gut“ ist, zeigt

Lemma 4.1
Mit den obigen Bezeichnungen gilt

(1) $\hat{q} \geq q$

(2) für $v_1 \geq \lfloor b/2 \rfloor$ ist $\hat{q} - 2 \leq q \leq \hat{q}$.

Beweis:
(1) Da $q \leq b-1$ ist die Aussage sicherlich richtig für $\hat{q} = b-1$. Anderfalls ist $\hat{q} = \lfloor (u_0 b + u_1)/v_1 \rfloor$, weil $\hat{q} v_1 \geq u_0 b + u_1 - v_1 + 1$. Daraus folgt

$$\begin{aligned} u - \hat{q}v \leq u - \hat{q}v_1 b^{n-1} &\leq u_0 b^n + \ldots + u_n \\ &\quad -(u_0 b^n + u_1 b^{n-1} - v_1 b^{n-1} + b^{n-1}) \\ &= u_2 b^{n-2} + \ldots + u_n - b^{n-1} + v_1 b^{n-1} < v_1 b^{n-1} \leq v. \end{aligned}$$

Da $u - \hat{q}v < v$ ist, muß $\hat{q} \geq q$ gelten.
(2) Angenommen es sei $\hat{q} \geq q + 3$. Es gilt

$$\hat{q} \leq \frac{u_0 b + u_1}{v_1} = \frac{u_0 b^n + u_1 b^{n-1}}{v_1 b^{n-1}} \leq \frac{u}{v_1 b^{n-1}} < \frac{u}{v - b^{n-1}}.$$

(Der Fall $v = b^{n-1}$ ist nicht möglich; denn für $v = (100\ldots0)_b$ ist $q = \hat{q}$.) Weiter folgt aus der Beziehung $q > (u/v) - 1$, daß

$$3 \leq \hat{q} - q < \frac{u}{v - b^{n-1}} - \frac{u}{v} + 1 = \frac{u}{v}\left(\frac{b^{n-1}}{v - b^{n-1}}\right) + 1$$

gilt. Daher ist

$$\frac{u}{v} > 2\left(\frac{v - b^{n-1}}{b^{n-1}}\right) \geq 2(v_1 - 1)$$

und wegen $b - 4 \geq \hat{q} - 3 \geq q = \lfloor u/v \rfloor \geq 2(v_1 - 1)$ erhält man $v_1 < \lfloor b/2 \rfloor$. □

Die Folgerung in (2) ist unabhängig von der Basis b. Die Bedingung $v_1 \geq \lfloor b/2 \rfloor$ ist eine Normierungsbedingung (Achtung, sie ändert den Rest!). Sie kann immer erreicht werden. Um zu sehen, daß v_1 groß genug ist, multipiziere man u und v mit $\lfloor b/(v_1+1) \rfloor$; dies ändert weder den Wert von u/v noch wächst die Stellenzahl von v.
Der folgende Algorithmus benutzt noch eine kleine verbesserte Wahl von $\hat{q}$ im Schritt D3, welche garantiert, daß $q = \hat{q}$ oder $\hat{q} - 1$.

Algorithmus D (Division nicht negativer ganzer Zahlen)

Eingabe: Zwei nicht negative ganze Zahlen $u = (u_1 u_2 \ldots u_{m+n})_b$ und $v = (v_1 v_2 \ldots v_n)_b$, wobei $v_1 \neq 0$ und $n > 1$.

Ausgabe: Der Teiler $\lfloor u/v \rfloor = (q_0 q_1 \ldots q_m)_b$ und der Rest $u \bmod v = (r_1 r_2 \ldots r_n)_b$.

D1. [Normierung]
Setze $d \leftarrow \lfloor b/(v_1 + 1) \rfloor$, $(u_0 u_1 \ldots u_{m+n})_b \leftarrow (u_1 u_2 \ldots u_{m+n})_b \times d$ und $(v_1 v_2 \ldots v_n)_b \leftarrow (v_1 v_2 \ldots v_n)_b \times d$.

D2. [Initialisiere j]
Setze $j \leftarrow 0$. (Die Schleife über j (Schritt D3 – D7) ist hauptsächlich eine Division von $(u_j u_{j+1} \dots u_{j+n})_b$ durch $(v_1 v_2 \dots v_n)_b$, um die Ziffer q_j zu erhalten.)

D3. [Berechne $\hat{q}$]
Falls $u_j = v_1$, setze $\hat{q} \leftarrow b - 1$; andernfalls setze $\hat{q} \leftarrow \lfloor (u_j b + u_{j+1})/v_1 \rfloor$. Prüfe, ob $v_2 \hat{q} > (u_j b + u_{j+1} - \hat{q} v_1) b + u_{j+2}$. Ist dies erfüllt, dann setze $\hat{q} \leftarrow \hat{q} - 1$ und wiederhole den Test. (Dieser Test bestimmt meist die Fälle, in denen $\hat{q}$ um eins zu groß ist und schließt alle Fälle, in denen $\hat{q}$ um zwei zu groß ist aus.)

D4. [Multiplikation und Subtraktion]
Ersetze $(u_j u_{j+1} \dots u_{j+n})_b$ durch $(u_j u_{j+1} \dots u_{j+n})_b - \hat{q} \times (v_1 v_2 \dots v_n)_b$. Falls $(u_j u_{j+1} \dots u_{j+n}) < 0$, dann setze $(u_j u_{j+1} \dots u_{j+n})_b \leftarrow (u_j u_{j+1} \dots u_{j+n})_b + b^{n+1}$ und behalte einen Übertrag von -1.

D5. [Prüfe Rest]
Setze $q_j \leftarrow \hat{q}$. Falls das Ergebnis in Schritt D4 negativ war, gehe zu Schritt D6; andernfalls gehe zu Schritt D7.

D6. [Korrektur (addiere zurück)]
Setze $q_j \leftarrow q_j - 1$ und $(u_j u_{j+1} \dots u_{j+n})_b \leftarrow (u_j u_{j+1} \dots u_{j+n})_b + (0 v_1 \dots v_n)_b$. (Ein Übertrag der links von u_j auftritt wird durch den Übertrag aus D4 gelöscht.)

D7. [Schleife über j]
Setze $j \leftarrow j + 1$. Falls $j \leq m$, gehe zurück zu D3.

D8. [Korrigiere Normierung]
$(q_0 q_1 \dots q_m)_b$ ist der gesucht Teiler. Den Rest erhält man durch teilen von $(u_{m+1} \dots u_{m+n})_b$ durch d.

4.4 Wie schnell kann man multiplizieren?

Die Standardmethode zur Multiplikation einer m-stelligen mit einer n-stelligen Zahl (siehe Algorithmus M im vorhergehenden Abschnitt) benötigt näherungsweise cmn Operationen, wobei c eine Konstante ist. Zur Multiplikation zweier n-stelliger Zahlen mit diesem Verfahren braucht man also $O(n^2)$ Operationen. Aber es geht auch schneller.

4.4.1 Algorithmus von Karatsuba

(Die spezielle Version mit $b = 2$ stammt von Karatsuba, 1962). Seien zwei $2n$-stellige Zahlen $u = (u_{2n-1} \dots u_1 u_0)_b$ und $v = (v_{2n-1} \dots v_1 v_0)_b$ gegeben. Diese kann man auch als

$$u = U_1 b^n + U_0 \quad \text{und} \quad v = V_1 b^n + V_0$$

schreiben, wobei $U_1 = (u_{2n-1} \dots u_n)_b$, $U_0 = (u_{n-1} \dots u_0)_b$ und $V_1 = (v_{2n-1} \dots v_n)_b$, $V_0 = (v_{n-1} \dots v_0)_b$ ist. Damit erhält man

$$\begin{aligned} uv &= (U_1 b^n + U_0)(V_1 b^n + V_0) \\ &= U_1 V_1 b^{2n} + (U_1 V_0 + U_0 V_1) b^n + U_0 V_0 \\ &= U_1 V_1 b^{2n} + [(U_1 - U_0)(V_0 - V_1) + U_1 V_1 + U_0 V_0] b^n + U_0 V_0 \\ &= U_1 V_1 (b^{2n} + b^n) + (U_1 - U_0)(V_0 - V_1) b^n + U_0 V_0 (b^n + 1). \end{aligned} \tag{4.7}$$

Diese Gleichung reduziert das Problem der Multiplikation zweier $2n$-stelligen Zahlen auf 3 Multiplikationen von n-stelligen Zahlen und einigen Shift- und Additionsoperationen. Der größte Vorteil liegt darin, daß Gleichung (4.7) rekursiv angewandt werden kann und dadurch das Verfahren schneller wird. Ist $T(n)$ die zur Multiplikation n-stelliger Zahlen benötigte Zeit, dann erhält man mit obigem Schema

$$T(2n) \leq 3T(n) + cn \tag{4.8}$$

für eine Konstante c, da die rechte Seite von (4.7) nur 3 Multiplikationen und einige Additionen und Shifts benutzt. Aus (4.8) ergibt sich durch Induktion ($n = 2^k$)

$$T(2^k) \leq c(3^k - 2^k), \quad k \geq 1,$$

falls c groß genug gewählt wird, so daß diese Ungleichung auch für $k = 1$ erfüllt ist. Damit gilt

$$\begin{aligned} T(n) \leq T(2^{\lceil \log_2 n \rceil}) &\leq c(3^{\lceil \log_2 n \rceil} - 2^{\lceil \log_2 n \rceil}) \\ &< 3c \cdot 3^{\log_2 n} = 3cn^{\log_2 3}. \end{aligned}$$

Dies zeigt, daß die Berechnungszeit für die Multiplikation für große n von der Ordnung $O(n^2)$ durch die rekursive Methode auf $O(n^{\log_2 3}) \approx O(n^{1.585})$ reduziert werden kann.

4.4.2 Ganzzahlige Multiplikation nach Schönhage-Strassen

Bei diesem Algorithmus wird die Basis auf 2 festgelegt. Die Idee der Halbierung der Zahlen kann verallgemeinert werden auf Partitionieren der Zahlen in b Blöcke von je l

Bits. Die b Blöcke werden als Koeffizienten eines Polynoms betrachtet. Um die Koeffizienten des Produktpolynoms zu berechnen, wertet man die Polynome an geeigneten Stellen aus, multipliziert die Werte und interpoliert. Werden Einheitswurzeln als Auswertungspunkte gewählt, kann man die Fourier-Transformation und den Faltungssatz anwenden. Setzt man b als eine Funktion von n und benutzt Rekursion, dann können zwei n-bit Zahlen in $O(n \log n \log\log n)$ Schritten multipliziert werden.

Der Einfachheit halber wird n auf eine Potenz von 2 beschränkt. (In den anderen Fällen füllt man die Zahlen einfach mit führenden Nullen auf, bis n eine Potenz von 2 ist). Ferner wird das Produkt zweier n-bit ganzer Zahlen modulo $(2^n + 1)$ berechnet. Um ein exaktes Ergebnis zu erhalten, muß man führende Nullen einführen und $2n$-bit ganze Zahlen modulo $(2^{2n} + 1)$ multiplizieren. Dadurch erhöht sich die Komplexität nur um einen konstanten Faktor.

Seien u und v binär dargestellte ganze Zahlen mit $0 \leq u, v \leq 2^n$. Ziel ist die Berechnung von $uv \bmod (2^n+1)$. (Man beachte, daß 2^n zur binären Darstellung $n+1$ Bits benötigt.) Ist entweder u oder v gleich 2^n, dann wird es durch -1 dargestellt; die Multiplikation wird einfach als ein Spezialfall behandelt, d.h. beispielsweise für $u = 2^n$ erhält man $uv \bmod (2^n + 1)$ durch berechnen von $(2^n + 1 - v) \bmod (2^n + 1)$.

Angenommen, es sei $n = 2^k$ und $b = 2^{k/2}$, falls k gerade und $b = 2^{(k-1)/2}$, falls k ungerade. Sei $l = n/b$. (Man beachte, daß $l \geq b$ und b ein Teiler von l ist.) Im ersten Schritt werden u und v in b Blöcke zu je l Bits

$$\begin{aligned} u &= u_{b-1}2^{(b-1)l} + \ldots + u_1 2^l + u_0 \\ v &= v_{b-1}2^{(b-1)l} + \ldots + v_1 2^l + v_0 \end{aligned}$$

unterteilt. Das Produkt von u und v ist gegeben durch

$$uv = y_{2b-2}2^{(2b-2)l} + \ldots + y_1 2^l + y_0 \tag{4.9}$$

mit

$$y_i = \sum_{j=0}^{b-1} u_j v_{i-j}, \quad 0 \leq i < 2b.$$

(Für $j < 0$ oder $j > b-1$ setzt man $v_j = u_j = 0$. y_{2b-1} ist immer 0 und wird nur aus Symmetriegründen eingeführt.)

Das Produkt uv kann mit Hilfe des Faltungssatzes berechnet werden. Multiplizieren mit der Fourier-Transformation benötigt aber $2b$ Multiplikationen. Durch die verhüllte Faltung kann die Zahl der Multiplikationen auf b reduziert werden. Aus diesem Grund berechnet man $uv \bmod (2^n + 1)$. Da $bl = n$, ist $2^{bl} \equiv -1 \bmod (2^n + 1)$. Daher gilt mit (4.9) und Lemma 4.2[3]

$$uv \equiv (w_{b-1}2^{(b-1)l} + \ldots + w_1 2^l + w_0) \bmod (2^n + 1),$$

[3]

wobei $w_i = y_i - y_{b+i}$ für $0 \leq i < b$.

Da

- das Produkt von 2 l-bit Zahlen kleiner als 2^{2l} ist,
- y_i und y_{b+i} Summen von $(i+1)$ bzw. $(b-(i+1))$ solchen Produkten sind,

liegt jeweils $w_i = y_i - y_{b+i}$ im Bereich von

$$-(b-1-i)2^{2l} < w_i < (i+1)2^{2l}.$$

Daher gibt es höchstens $b2^{2l}$ mögliche Werte für w_i. Können die w_i's $\bmod b2^{2l}$ berechnet werden, dann kann man $uv \bmod (2^n+1)$ in $O(b\log(b2^{2l}))$ in Schritten berechnen.

Um die w_i's $\bmod b2^{2l}$ zu berechnen, berechnet man w_i zweimal, einmal modulo b und ein zweites mal modulo $2^{2l}+1$. Seien also

$$w_i' = w_i \bmod b \qquad w_i'' = w_i \bmod (2^{2l}+1).$$

Da b eine Potenz von 2 und $2^{2l}+1$ ungerade ist, sind b und $2^{2l}+1$ teilerfremd. Damit können die w_i's aus den w_i''s und w_i'''s aus

$$w_i = (2^{2l}+1)((w_i' - w_i'') \bmod b) + w_i'' \text{ [4]}$$

berechnet werden und w_i liegt zwischen $-(b-1-i)2^{2l}$ und $(i+1)2^{2l}$. Die benötigte Zeit zur Berechnung für ein w_i aus w_i' und w_i'' ist $O(l+\log b)$ und für alle $O(bl+b\log b)$ oder $O(n)$.

Berechnung der w_i''s
Dazu nimmt man $u_i' = u_i \bmod b$ und $v_i' = v_i \bmod b$ und bildet daraus zwei teilbare $(3b\log b)$-bit Zahlen $\hat{u}$ und $\hat{v}$

$$\begin{aligned} \hat{u} &= u_{b-1}'00\ldots0u_{b-2}'00\ldots0u_{b-3}'\cdots00\ldots0u_0' \\ \hat{v} &= v_{b-1}'00\ldots0v_{b-2}'00\ldots0v_{b-3}'\cdots00\ldots0v_0', \end{aligned}$$

Lemma 4.2
Sei p eine positive ganze Zahl, $m = \omega^p + 1$ und $a = \sum_{i=0}^{l-1} a_i\omega^{pi}$, wobei $0 \leq a_i < \omega^p$ für jedes i. Dann gilt $a \equiv \sum_{i=0}^{l-1}(-1)^i a_i \pmod{m}$.

Beweis: Es ist $\omega^p \equiv -1 \pmod{m}$. □

[4] Seien p_1 und p_2 teilerfremd, $w \equiv q_1 \pmod{p_1}$ und $w \equiv q_2 \pmod{p_2}$ und es gelte $0 \leq w < p_1p_2$. Dann ist $w = p_2(p_2^{-1} \bmod p_1)(q_1 - q_2 \bmod p_1) + q_2$. Setze $p_1 = b$ und $p_2 = 2^{2l}+1$. Da b eine Potenz von 2 ist und $b \leq 2^{2l}$, ist b ein Teiler von 2^{2l}. Also ist das multiplikativ inverse Element von $(2^{2l}+1) \bmod b$ gleich 1.

wobei in jedem Block aus Nullen $2 \log b$ Nullen sind. Das Berechnen des Produktes $\hat{u}\hat{v}$ mit dem Algorithmus von Karatsuba benötigt höchstens $O((3b \log b)^{1.59})$ Schritte, d.h. weniger als $O(n)$. Es gilt $\hat{u}\hat{v} = \sum_{i=0}^{2b-1} y'_i 2^{(3 \log b)i}$, wobei $y'_i = \sum_{j=0}^{2b-1} u'_j v'_{i-j}$. Da $y'_i < 2^{3 \log b}$ ist, können die y'_i's leicht aus dem Produkt $\hat{u}\hat{v}$ gewonnen werden. Die Werte der w_i's mod$b$ können durch Berechnen von $(y'_i - y'_{b+i})$ mod b bestimmt werden.

Berechnung der w''_i's
Diese werden mittels der verhüllten Faltung bestimmt. Sei $\omega = 2^{4l/b}$ und $m = 2^{2l} + 1$. Mit Satz 4.16 besitzt ω und b ein multiplikativ inverses Element modulo m und ω ist eine primitive b-te Einheitswurzel. Daher ist die negativ verhüllte Faltung von $[u_0, \psi u_1, \ldots, \psi^{b-1} u_{b-1}]$ und $[v_0, \psi v_1, \ldots, \psi^{b-1} v_{b-1}]$ mit $\psi = 2^{2l/b}$ (ψ ist eine $2b$-te Einheitswurzel)

$$[(y_0 - y_b), \psi(y_1 - y_{b+1}), \ldots, \psi^{b-1}(y_{b-1} - y_{2b-1})] \bmod (2^{2l} + 1),$$

wobei $y_i = \sum_{j=0}^{b-1} u_j v_{i-j}$ für $0 \leq i \leq 2b - 1$ ist. Die w_i's modulo $(2^{2l} + 1)$ können durch geeignetes Shiften gewonnen werden.

Algorithmus (Ganzzahlige Multiplikation nach Schönhage-Strassen)

Eingabe: Zwei n-bit ganze Zahlen u und v, wobei $n = 2^k$.

Ausgabe: Das $(n + 1)$-bit Produkt von u und v modulo $2^n + 1$.

Vorgehensweise: Für kleines n berechne uv mod $2^n + 1$ mit einem gewöhnlichen Algorithmus. Für großes n setze $b = 2^{k/2}$, falls k gerade bzw. $b = 2^{(k-1)/2}$, falls k ungerade und $l = n/b$. Stelle u durch $\sum_{i=0}^{b-1} u_i 2^{li}$ und v durch $\sum_{i=0}^{b-1} v_i 2^{li}$ dar, wobei u_i und v_i ganze Zahlen zwischen 0 und $2^l - 1$ sind (d.h. sie sind Blöcke aus l Bit von u bzw. v).

1. Berechne die Fourier-Transformierten modulo $(2^{2l} + 1)$ von

$$U = [u_0, \psi u_1, \ldots, \psi^{b-1} u_{b-1}]^T \quad \text{und} \quad V = [v_0, \psi v_1, \ldots, \psi^{b-1} v_{b-1}]^T$$

mit $\psi = 2^{2l/b}$ und benutzen von ψ^2 als die die primitive b-te Einheitswurzel.

2. Bilde das paarweise Produkt, der im Schritt 1 berechneten Fourier-Transformierten modulo $(2^{2l}+1)$ unter rekursiver Anwendung dieses Algorithmus zur Berechnung jedes Produktpaares. (Ist eine der Zahlen 2^{2l} kann dies als einfacher Spezialfall behandelt werden).

3. Berechne die inverse Fourier-Transformation modulo $(2^{2l} + 1)$ vom Vektor der paarweisen Produkte aus Schritt 2. Das Ergebnis dieser Berechnung ist $[w_0, \psi w_1, \ldots, \psi^{b-1} w_{b-1}]^T$ mod $(2^{2l} + 1)$, wobei w_i der i-te Term der negativ verhüllten Faltung von U und V ist.Berechne $w''_i = w_i$ mod $(2^{2l} + 1)$ durch Multiplikation von $\psi^i w_i$ mit ψ^{-i} modulo $(2^{2l} + 1)$.

4. Berechne $w'_i = w_i \bmod b$ folgendermaßen:

(a) Setze $u'_i = u_i \bmod b$, $v'_i = v_i \bmod b$ für $0 \leq i < b$.

(b) Konstruiere $\hat{u}$ und $\hat{v}$, durch Einfügen von Blöcken aus je $2 \log b$ Nullen zwischen den u_i's und v_i's. Es ist dann $\hat{u} = \sum_{i=0}^{b-1} u'_i 2^{(3 \log b)i}$ und $\hat{v} = \sum_{i=0}^{b-1} v'_i 2^{(3 \log b)i}$.

(c) Berechne $\hat{u}\hat{v}$ mit dem Algorithmus von Karatsuba.

(d) Es ist das Produkt $\hat{u}\hat{v} = \sum_{i=0}^{2b-1} y'_i 2^{(3 \log b)i}$ mit $y'_i = \sum_{j=0}^{2b-1} u'_j v'_{i-j}$. Die w_i's modulo b gewinnt man aus $\hat{u}\hat{v}$ durch auswerten von

$$(y'_i - y'_{b+i}) \bmod b \quad \text{für} \quad 0 \leq i < b.$$

5. Berechne die w_i's exakt durch

$$w_i = (2^{2l} + 1)((w'_i - w''_i) \bmod b) + w''_i.$$

(w_i liegt zwischen $-(b-1-i)2^{2l}$ und $(i+1)2^{2l}$).

6. Berechne $\sum_{i=0}^{b-1} w_i 2^{li} \bmod (2^n + 1)$. Dies ist das gesuchte Ergebnis.

Analyseskizze
Die Schritte 5 und 6 liegen in $O(n)$ und werden daher ignoriert. Schritt 3 hat einen Aufwand von $O((3b \log b)^{1.59})$. Für große b gilt aber $(3b \log b)^{1.59} < b^2$. Daher wird auch dieser Schritt ignoriert. Die Analyse hängt stark von Schritt 1 bis 3 ab, wobei die zu erwartende Zeit dabei $O(bl \log b + bM(2l))$ ist. $M(n)$ ist die bei einer rekursiven Anwendung des Algorithmus benötigte Zeit zur Multiplikation zweier n-bit ganzer Zahlen. Eine genauere Analyse zeigt, daß

$$M(n) \leq c' n \log n \log \log n.$$

Daher benötigt der Algorithmus $O(n \log n \log \log n)$ Schritte.

Bemerkung:
Bis jetzt überprüft keine Implementation diese Komplexitätsanalyse. Die Frage bleibt offen, ob die Konstante sehr groß ist oder nicht.

4.5 Modulare Arithmetik

Die modulare Arithmetik ganzer Zahlen wird im Kapitel 6 ausführlich dargestellt. Deshalb werden hier nur die wesentlichen Ideen aufgezeigt.

Statt mit großen Zahlen zu rechnen, kann man auch mit mehreren kleinen Zahlen, genauer mit den Resten modulo gewisser Zahlen arbeiten.
Sei u eine ganze Zahl und $m_1, \ldots, m_r$ paarweise teilerfremde Zahlen (d.h. es ist der $\gcd(m_i, m_j) = 1$ für $i \neq j$). Man stellt u nun als Tupel dieser r Reste modulo m_i dar:

$$u \leftrightarrow (u_1, \ldots, u_r) = (u \bmod m_1, \ldots, u \bmod m_r).$$

Beispiel:
Seien $m_1 = 5$, $m_2 = 3$ und $m_3 = 2$. Dann ist

$$\begin{array}{rcl} 4 & \leftrightarrow & (4,1,0), \\ 7 & \leftrightarrow & (2,1,1). \end{array}$$

Die Idee ist nun statt der Zahlen u und v, die sie darstellenden Tupel komponentenweise zu addieren, subtrahieren und multiplizieren:

$$\begin{array}{rcl} (u_1, \ldots, u_r) + (v_1, \ldots, v_r) & = & ((u_1 + v_1) \bmod m_1, \ldots, (u_r + v_r) \bmod m_r), \\ (u_1, \ldots, u_r) - (v_1, \ldots, v_r) & = & ((u_1 - v_1) \bmod m_1, \ldots, (u_r - v_r) \bmod m_r), \\ (u_1, \ldots, u_r) \times (v_1, \ldots, v_r) & = & ((u_1 \times v_1) \bmod m_1, \ldots, (u_r \times v_r) \bmod m_r). \end{array}$$

Um beispielsweise zu beweisen, daß diese Multiplikationsdarstellung auch das Produkt uv repräsentiert, muß für jedes i gezeigt werden, daß

$$uv \bmod m_i = (u \bmod m_i)(v \bmod m_i) \bmod m_i.$$

Mit dem Divisionsalgorithmus erhält man

$$\begin{array}{rcl} u & = & xm_i + u_i, \quad 0 \leq u_i < m_i, \\ v & = & ym_i + v_i, \quad 0 \leq v_i < m_i. \end{array}$$

Damit ist

$$\begin{array}{rcl} uv \bmod m_i & = & ((xm_i + u_i)(ym_i + v_i)) \bmod m_i \\ & = & ((xym_i + xv_i + yu_i)m_i + u_i v_i) \bmod m_i \\ & = & (u_i v_i) \bmod m_i \\ & = & (u \bmod m_i)(v \bmod m_i) \bmod m_i. \end{array}$$

Die Addition und Subtraktion kann auf die gleiche Weise gezeigt werden. Diese drei Operationen sind sehr einfach. Nachteil dieser modularen Darstellung ist beispielsweise, daß sich der Vergleich zweier Zahlen oder die Entscheidung, ob die dargestellte Zahl positiv oder negativ ist, sehr schwierig gestaltet.

Die Codierung modulo $m_1, \ldots, m_r$, d.h. die Transformation von der normalen zur modularen Darstellung kann einfach durch je eine Division mit Rest $u_i = u \bmod m_i$ erfolgen. Schneller geht es mit einem Teile-und-Herrsche Verfahren (ähnlich der Fourier-Transformation), wenn r eine Potenz von 2 ist. Dann kann diese in $O(br \log r \log br \log\log br)$ Schritten berechnet werden, wobei b die Anzahl der Bits zur binären Darstellung des größten Modulus m_i ist.

Es bleibt noch die Frage der Eindeutigkeit dieser Darstellung und der Rücktransformation. Klärung schafft hier der chinesische Restesatz (Kapitel 6).

Auch kann die modulare Arithmetik benutzt werden, um große Zahlen schnell zu multiplizieren. Der Algorithmus stammt von A. Schönhage (vgl. Knuth Bd. II, S. 287). Die Zeit zur n-bit Multiplikation liegt in $O(n^{\log_3 6}) \approx O(n^{1.63})$.

4.6 Wie schnell kann man den gcd berechnen?

Bei der modularen Arithmetik oder beim Rechnen mit Brüchen wird der gcd oft gebraucht. Daher soll nun ein schnellerer Algorithmus als der Standard-Algorithmus vorgestellt werden. Der Standard-Algorithmus geht wie folgt

Algorithmus A (Euklidischer Algorithmus)

Eingabe: Zwei nicht negative ganze Zahlen u und v.

Ausgabe: größter gemeinsamer Teiler von u und v.

A1. [$v = 0$?]
Falls $v = 0$, terminiere mit u als Ausgabe.

A2. [Nimm $u \bmod v$.]
Setze $r \leftarrow u \bmod v$, $u \leftarrow v$ und $v \leftarrow r$. Gehe zurück zu A1. (Die Operationen in diesem Schritt verkleinern den Wert von v, lassen den $\gcd(u, v)$ jedoch unverändert.)

Beispiel:

$$\begin{aligned}
\gcd(40902, 24140) &= \gcd(24140, 16762) = \gcd(16762, 7378) \\
&= \gcd(7378, 2006) = \gcd(2006, 1360) = \gcd(1360, 646) \\
&= \gcd(646, 68) = \gcd(68, 34) = \gcd(34, 0) = 34.
\end{aligned}$$

Der ungünstigste Fall der gcd Berechnung tritt ein, wenn $\gcd(u, v) = 1$ ist. Und daß dies oft der Fall ist, zeigt

Satz 4.3 (Dirichlet Lejeune 1849, Cesàro 1881)
Für zwei zufällig gewählte Zahlen u und v ist die Wahrscheinlichkeit für $P(\gcd(u, v) = 1) = 6/\pi^2 = 0.60793$.

Ein ganz anderer gcd-Algorithmus, der primär auf binäre Arithmetik zugeschnitten ist, stammt von J. Stein 1961. Dieser neue Algorithmus benötigt keine Division. Er baut einzig auf den Operationen Subtraktion, testen, ob eine Zahl gerade oder ungerade ist und auf einem Rechtsshift (halbieren) auf. Der binäre gcd-Algorithmus basiert auf vier einfachen Eigenschaften positiver ganzer Zahlen u und v:

- Sind u und v gerade, dann ist $\gcd(u, v) = 2\gcd(u/2, v/2)$.
- Ist u gerade und v ungerade, dann gilt $\gcd(u, v) = \gcd(u/2, v)$.
- Wie beim euklidischen Algorithmus gilt $\gcd(u, v) = \gcd(u - v, v)$.
- Sind u und v ungerade, dann ist $u - v$ gerade und $|u - v| < \max(u, v)$.

Diese Eigenschaften ergeben sofort den

Algorithmus B (Binärer gcd-Algorithmus)

Eingabe: Zwei nicht negative ganze Zahlen u und v.

Ausgabe: größter gemeinsamer Teiler von u und v.

B1. [Finde Zweier-Potenzen.]
Setze $k \leftarrow 0$ und wiederhole, bis u und v nicht beide gerade sind, $k \leftarrow k + 1$, $u \leftarrow u/2$, $v \leftarrow v/2$.

B2. [Initialisierung.]
(Die Werte von u und v sind durch 2^k geteilt und mindestens einer der gegenwärtigen Werte ist ungerade.) Falls u ungerade, setze $t \leftarrow -v$ und gehe zu B4; andernfalls setze $t \leftarrow u$.

B3. [Halbiere t.]
(Hier ist t gerade und von Null verschieden.) Setze $t \leftarrow t/2$.

B4. [Ist t gerade?]
Falls t gerade, gehe zurück zu B3.

B5. [Neudefinition von $\max(u, v)$.]
Falls $t > 0$, setze $u \leftarrow t$; andernfalls setze $v \leftarrow -t$. (Der größere der beiden Werte von u und v ist durch $|t|$ ersetzt worden, außer vielleicht beim ersten Durchlauf dieses Schrittes.)

B6. [Subtraktion.]
Setze $t \leftarrow u - v$. Falls $t \neq 0$, gehe zurück zu B3. Andernfalls terminiere mit $u \cdot 2^k$ als Ausgabe.

Beispiel:
Sei $u = 40902$ und $v = 24140$, wie im vorhergehenden Beispiel. Schritt B1 ergibt $k \leftarrow 1$, $u \leftarrow 20451$ und $v \leftarrow 12070$. Dann wird t auf -12070 gesetzt und durch -6035 ersetzt. Weiter wird v durch 6035 ersetzt und die Berechnung setzt sich wie folgt fort:

u	v	t
20451	6035	$+14416,\ +7208,\ +3604,\ +1802,\ +901;$
901	6035	$-5134,\ -2567;$
901	2567	$-1666,\ -833;$
901	833	$+68,\ +34,\ +17;$
17	833	$-816,\ -408,\ -204,\ -102,\ -51;$
17	51	$-34,\ -17;$
17	17	0.

Die Ausgabe ist $17 \cdot 2^1 = 34$. Es sind bei diesem Algorithmus im Vergleich zum vorherigen Algorithmus mehr Schritte notwendig. Dafür ist aber jede Iteration einfacher, da keine Division benutzt wird.

Eine weitere Methode, den gcd zu berechnen ist in Aho, Hopcroft, Ullman Seite 309 beschrieben.

Der größte gemeinsame Teiler zweier ganzer Zahlen a und b hat eine sehr wichtige Eigenschaft. Es gibt nämlich stets zwei ganze Zahlen s und t, so daß

$$\gcd(a, b) = sa + tb$$

ist. In Kapitel 5 wird der erweiterte euklidische Algorithmus vorgestellt, der in jedem euklidischen Ring zusätzlich zum $\gcd(a, b)$ noch die beiden Zahlen s und t bestimmen kann. Anwendung findet dieser Algorithmus in der Berechnung von Inversen. Dies wird u.a. beim chinesischen Restealgorithmus in Kapitel 6 benötigt.

Kapitel 5

Polynom-Arithmetik

5.1 Polynomdivision

Die folgenden Berechnungen wurden mit Langzahldivision durchgeführt:

$$
\begin{array}{r|l}
 & x - y \quad R\, y \\
\hline
x + y & x^2 - y^2 + y \\
 & x^2 + yx \\
\cline{2-2}
 & \quad - yx - y^2 + y \\
 & \quad - yx - y^2 \\
\cline{2-2}
 & \qquad\qquad\qquad y
\end{array}
\qquad
\begin{array}{r|l}
 & - y + 1 + x \quad R - x \\
\hline
y + x & - y^2 + y + x^2 \\
 & - y^2 - xy \\
\cline{2-2}
 & \qquad y + xy + x^2 \\
 & \qquad y + x \\
\cline{2-2}
 & \qquad xy + x^2 - x \\
 & \qquad xy + x^2 \\
\cline{2-2}
 & \qquad\qquad\quad - x
\end{array}
$$

Dies zeigt, daß bei dieser Divisionsmethode das Ergebnis von der Wahl der Hauptvariablen abhängt; man erhält zwar äquivalente Ausdrücke, aber unterschiedliche Darstellungen.

Die Division

$$
\begin{array}{r|l}
 & \left(\frac{x+1}{x+2}\right) y \;+\; \frac{x+1}{(x+2)^2} \\ \hline
(x+2)y - 1 & (x+1)y^2 \;+\; x \\
 & (x+1)y^2 \;-\; \left(\frac{x+1}{x+2}\right) y \\ \cline{2-2}
 & \left(\frac{x+1}{x+2}\right) y \;+\; x \\
 & \left(\frac{x+1}{x+2}\right) y \;-\; \frac{x+1}{(x+2)^2} \\ \cline{2-2}
 & \frac{x+1}{(x+2)^2} \;+\; x
\end{array}
$$

zeigt, daß i. allg. weder der Quotient noch der Rest einer Polynomdivision wieder ein Polynom sein muß. In einem Körper ist die Division (außer durch 0) immer möglich. Jedoch ist $\mathbf{Z}[X,Y]$ (die Menge der bivariaten Polynome über den ganzen Zahlen) kein Körper. (Auch $\mathbf{Z}[X]$ ist kein Körper!)

5.1.1 Polynomdivision über einem Körper

Eingabe:

$$
\begin{aligned}
u(x) &= u_m x^m + \ldots + u_1 x + u_0 \\
v(x) &= v_n x^n + \ldots + v_1 x + v_0 \\
v_n &\neq 0, \quad m \geq n \geq 0
\end{aligned}
$$

Ausgabe:

$$
\begin{aligned}
q(x) &= q_{m-n} x^{m-n} + \ldots + q_0 \\
r(x) &= r_{n-1} x^{n-1} + \ldots + r_0 \\
u(x) &= q(x)v(x) + r(x), \quad \deg(r) < \deg(v).
\end{aligned}
$$

begin
   for $k \leftarrow m-n$ **step** -1 **to** 0 **do**
   begin
      $q_k \leftarrow u_{n+k}/v_n$
      for $j \leftarrow n+k-1$ **step** -1 **to** k **do**
         $u_j \leftarrow u_j - q_k v_{j-k}$
         { $\equiv u(x)$ durch $u(x) - q_k x^k v(x)$ zu ersetzen, }
         { u_{n+k} wird nicht mehr benötigt. }
   end
   for $i \leftarrow 0$ **to** $n-1$ **do** $r_i \leftarrow u_i$
end

Ist $l(v) = lc(v(x))$, dann benötigt man nur eine explizite Division von $l(v)$ und es wird die 2. Schleife genau $(m - n + 1)$-mal ausgeführt
Also falls $u(x)$ und $v(x)$ als Startwerte nur ganzzahlige Koeffizienten haben und man über den rationalen Zahlen arbeitet, dann sind nur die Nenner, die in den Koeffizienten von $q(x)$ und $r(x)$ vorkommen, Teiler von $l(v)^{m-n+1}$. Dies führt zu der Annahme, daß es immer möglich ist, Polynome $q(x)$ und $r(x)$ zu finden, so daß

$$l(v)^{m-n+1}u(x) = q(x)v(x) + r(x) \quad \deg(r) < n$$

$m = \deg(u)$ $\quad n = \deg(v)$ für beliebige Polynome $u(x)$, $v(x)$, $u(x) \neq v(x)$ gilt, unter der Vorausetzung $m \geq n$.

Um den Begriff der Division auf beliebige kommutative Ringe zu erweitern, benötigt man die Definition einer neuen Operation, die Pseudo-Division. Diese Operation verhält sich ähnlich wie die Division, nur die Ergebnisse (Pseudo-Quotient und Pseudo-Rest) sind immer Polynome. Zuerst definiert man das Konzept einer l-reduziblen Operation (gemäß Bonneau), das im nächsten Abschnitt benötigt wird. Im folgenden Diagramm (D ist eine Menge von Polynomen)

$$\begin{tikzcd}
D \arrow[r, "\phi"] \arrow[d, "\psi"'] & \phi(D) \arrow[d, "\psi"] \\
\psi(D) \arrow[r, "\phi"'] & \phi(\psi(D))
\end{tikzcd}$$

ist ϕ eine Operation auf Polynomen. Man sagt ϕ ist l-**reduzierbar**, falls ψ eine l-Auswertung ist (ψ läßt l Variablen unausgewertet) und das Diagramm kommutiert. Beispiele für 0-reduzierte Operationen sind exakte Division, Multiplikation, Addition und Subtraktion und für 1-reduzierte Operationen sind dies die Division mit Rest und der größte gemeinsame Teiler. Nun definiert man die

5.1.2 Pseudo-Division

Es seien $U(x)$ und $V(x)$ Polynome über einem kommutativen Ring mit Einselement. Sei m der Grad von U, n der von V und $lc(V)$ der Leitkoeffizient von V. Pseudo-Division ist die Operation, die Polynome $Q(x)$ und $R(x)$ – den Pseudo-Quotient und den Pseudo-Rest – so bestimmt, daß gilt

$$\begin{array}{ll} (i) & [lc(V)]^{m-n+1}U(x) = Q(x)V(x) + R(x) \quad \text{und} \\ (ii) & 0 \leq \text{ Grad von } R < n \end{array}$$

Man stellt fest, daß die Operation Polynome $q(x)$ und $r(x)$ zu finden, so daß $U(x) = q(x)V(x) + r(x)$ und $0 \leq$ Grad von $r < n$ gilt, falls die Menge der Koeffizienten keinen Körper bilden i. allg. nicht möglich ist, die Operation der Pseudo-Division dagegen ist in einem kommutativen Ring mit Einselement immer möglich. Ein Algorithmus für die Pseudo-Division ist der Algorithm R (Vol. 2, Section 4.6.1) von Knuth:

$$\begin{aligned} U(x) &= u_m x^m + u_{m-1} x^{m-1} \ldots + u_1 x + u_0 \\ V(x) &= v_n x^n + v_{n-1} x^{n-1} \ldots + v_1 x + v_0 \\ Q(x) &= q_{m-n} x^{m-n} + q_{m-n-1} x^{m-n-1} + \ldots + q_1 x + q_0 \\ R(x) &= r_{n-1} x^{n-1} + r_{n-2} x^{n-2} + \ldots + r_1 x + r_0 \end{aligned}$$

begin
  for $k \leftarrow m - n$ **step** -1 **to** 0 **do**
  begin
    $q_k \leftarrow u_{n+k} v_n^k$
    for $j \leftarrow n + k - 1$ **step** -1 **to** 0 **do**
      $u_j \leftarrow v_n u_j - u_{n+k} v_{j-k}$ $\{v_{-1} = v_{-2} = \ldots = 0\}$
  end
  for $i \leftarrow 0$ **to** $n - 1$ **do** $r_i \leftarrow u_i$
end Zu den Operationen Addition, Subtraktion, Multiplikation dicht besetzter Polynome gibt es wie für die Langzahlarithmetik grundlegende Algorithmen.

5.2 Berechnung von Potenzen (Binärmethode)

Problem: Effiziente Berechnung von x^n, bei gegebenem x und $n > 0$.

Beispiel: x^{16}.
Die einfache Methode $x * x * \cdots * x$ benötigt 15 Multiplikationen.
Bei Benutzung der Zwischenergebnisse braucht man nur 4 Multiplikationen: x^2, x^4, x^8, x^{16}.

Die gleiche Idee läßt sich auf jedes n anwenden:

(i) Schreibe n in Binärdarstellung (unterdrücke die linken 0'en)

(ii) Ersetze jede '1' durch SX

(iii) Ersetze jede '0' durch S

(iv) Streiche das SX, das ganz links steht

Das Ergebnis ist eine Regel um x^n zu berechnen:
S ist die Operation zum Quadrieren und X die Operation mit x zu multiplizieren.

$x^{23} \Longrightarrow$ (i) $n \equiv 10111$ (ii)-(iii) $\Longrightarrow$ SX S SX SX SX (iv) S SX SX SX $\Longrightarrow$ $x^2,\ x^4,\ x^5,\ x^{10},\ x^{11},\ x^{22},\ x^{23}$
Dies ist die **Binärmethode**.

Diese Methode setzt voraus, daß die Binärdarstellung von n von links nach rechts durchgegangen wird, obwohl es angenehmer ist dies von rechts nach links zu tun. Im folgenden Algorithmus wird dem Rechnung getragen.

Algorithmus Rechts-nach-Links Binärmethode zum Potenzieren ($n > 0$)

begin
  $N := n$; $Y := 1$; $Z := x$;
  while $N \neq 0$ **do**
    { Invariante: $Y \cdot Z^N = x^n$ }
    if N ungerade **then**
      $Y := Y \cdot Z$;
    end
    $Z := Z \cdot Z$
    $N := \lfloor \frac{N}{2} \rfloor$
  end
end

Beispiel: x^{23}.

bei Schleifenbedingung	N	Y	Z
1	23	1	x
2	11	x	x^2
3	5	x^3	x^4
4	2	x^7	x^8
5	1	x^7	x^{16}
6	0	x^{23}	x^{32}

5.3 Polynomiale Restfolge und gcd-Berechnung

Der Euklidische Algorithmus zur Bestimmung des gcd von zwei ganzen Zahlen u_0, u_1 berechnet die ganzzahlige Restfolge (integer remainder sequence IRS)

$$u_0,\ u_1,\ u_2, \ldots, u_{k-1},\ u_k = 0$$

mit $u_i = rem(u_{i-2}, u_{i-1})$ für $i \geq 2$. Dieses IRS kann als Idee zur eigentlichen Berechnung angesehen werden. Eine ähnliche Methode kann für Polynome entworfen werden, mit dem Hauptunterschied: Es wird die Pseudo-Division benutzt um den Pseudo-Rest zu bekommen.

Definition 5.1
Seien $F_1(x)$, $F_2(x)$ Polynome über einem Integritätsbereich und es gelte $\deg(F_1(x)) \geq \deg(F_2(x))$. *Eine polynomiale Restfolge (polynomial remainder sequence PRS) kann zur Berechnung des* $\gcd(F_1, F_2)$ *wie folgt konstruiert werden.*
Eine PRS von $F_1(x)$ und $F_2(x)$ ist:

$$F_1(x),\, F_2(x),\, \ldots,\, F_{k-1}(x),\, F_k(x) = 0$$

mit $F_i(x) = \alpha_i F_{i-2}(x) - Q_i(x) F_{i-1}(x)$ *und* $\deg(F_i) < \deg(F_{i-1})$, $i \geq 3$.

Der Wert für den $\gcd(F_1, F_2)$ stimmt mit dem Wert des ersten F_{k-1}, der konstruierten PRS, so daß $F_k(x) = 0$ gilt, überein.

5.3.1 Der euklidische Algorithmus

Ist der Wert α_i in der Definition 5.1 durch f_{i-1}^{m-n+1} gegeben, wobei f_{i-1} der Leitkoeffizient von $F_{i-1}(x)$ ist, $m = \deg(F_1)$ und $n = \deg(F_2)$, dann heißt die PRS eine euklidische PRS und der Algorithmus, der den $\gcd(F_1, F_2)$ berechnet, heißt der euklidische gcd-Algorithmus.

Beispiel:

$$\begin{aligned} F_1(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5 \\ F_2(x) &= 3x^6 + 5x^4 - 4x^2 - 9x + 21 \end{aligned}$$

Bei Verwendung der Pseudo-Division erhält man folgende PRS:
(1) 1 0 1 0 −3 −3 8 2 −5
(2) 3 0 5 0 −4 −9 21
(3) −15 0 3 0 −9
(4) 15795 30375 −59535
(5) 1254542875143750 −1654608338437500
(6) 12593338795500743100931141992187500.
Da der letzte Schritt eine ganze Zahl liefert, sind die beiden Polynome teilerfremd.

Es werden nur die Koeffizienten der Polynome gespeichert.
Hauptbemerkung: Die PRS zeigt exponentielles Wachsen der Koeffizienten (Verdoppelung der Größe mit jedem Schritt!). Dies wirkt sich sehr nachteilig auf die Kosten dieses Algorithmus aus.

Verwendung rationaler Arithmetik:
Eine Möglichkeit den Wachstum der Koeffizienten einzudämmen ist, rationale Koeffizienten und rationale Arithmetik zur Division zu verwenden, um die PRS zu bestimmen. Damit wird die PRS zu:

(1) $1\ 0\ 1\ 0\ -3\ -3\ \ 8\ 2\ -5$

(2) $3\ 0\ 5\ 0\ -4\ -9\ 21$

(3) $-\frac{5}{9}\ 0\ \frac{1}{9}\ 0\ -\frac{1}{3}$

(4) $\frac{117}{25}\ -9\ \frac{441}{25}$

(5) $\frac{233150}{6591}\ -\frac{102500}{2197}$

(6) $\frac{1288744821}{543589225}$

Das Ergebnis-Polynom ist wieder eine Konstante. Das Anwachsen der Koeffizienten ist eingeschränkt, dafür entstehen aber neue Kosten zur Berechnung der gcd's in den Koeffizienten.

5.3.2 Normierte und primitive Polynome

Definition 5.2
Normiertes Polynom: Leitkoeffizient ist 1.
Primitives Polynom: alle Koeffizienten sind teilerfremd, $(cont(P) = 1)$
Inhalt eines Polynoms P: $\mathrm{cont}(P) = gcd(Koeffizienten)$.
Primitiver Teil von P: $\mathrm{pp}(P) = \dfrac{P}{\mathrm{cont}(P)}$

$$\Longrightarrow P = \mathrm{cont}(P)\,\mathrm{pp}(P).$$

Es lassen sich auch gcd-Algorithmen konstruieren, in denen alle Polynome (a) normiert oder zu (b) primitiven Polynomen umgeformt werden.

- Der erste Algorithmus benötigt für jedes Element der PRS einen Divisionschritt und ist daher nur in Verbindung mit rationaler Arithmetik sinnvoll. Ist dies gegeben, erhält man folgende PRS:

  (1) $1\ 0\ 1\ 0\ -3\ -3\ \ 8\ 2\ -5$

  (2) $1\ 0\ \frac{5}{3}\ 0\ \frac{4}{3}\ -3\ 7$

  (3) $1\ 0\ -\frac{1}{5}\ 0\ -\frac{3}{5}$

  (4) $1\ \frac{25}{13}\ -\frac{49}{13}$

(5) $1 - \frac{6150}{4663}$
(6) 1.

Die zusätzlichen Kosten können in vielen Fällen zu groß sein.

- Der zweite Algorithmus liefert eine PRS mit vernünftigem Koeffizientenwachstum, wie an dem gleichen Beispiel zu sehen ist:
  (1) 1 0 1 0 −3 −3 8 2 −5
  (2) 3 0 5 0 −4 −9 21
  (3) 5 0 −1 0 3
  (4) 13 25 −49
  (5) 4663 −6150
  (6) 1.

  Dies erreicht man durch Wegkürzen des Inhalts in jedem Schritt der PRS Konstruktion. Aber die zusätzliche Berechnung kann sehr teuer sein.

5.3.3 Reduzierte polynomiale Restfolge

Im letzten Algorithmus (der Primitive gcd-Algorithmus) wurde in jedem Schritt die zusätzliche Operation, durch den Inhalt des Restes von F_{i-2} und F_{i-1} zu dividieren, hinzugefügt. Ist β_i so ein Teiler, so kann der wesentliche Schritt des Algorithmus zur Berechnung der PRS dargestellt werden als

$$\beta_i F_i = \alpha_i F_{i-2} - Q_i F_{i-1},$$

wobei α_i der Faktor ist, der bei der Pseudo-Division gebraucht wird und β_i ist der Inhalt des Pseudo-Restes.

Man kann nun einen neuen Algorithmus einführen, falls für β_i angenommen wird, daß

$$\beta_3 = 1, \quad \beta_i = \alpha_{i-1}, \text{ für } i \geq 4$$

gilt. Diese Wahl der β_i führt auf den sogenannten **reduzierten** gcd-**Algorithmus** von Collins, bei dem sich die folgende PRS ergibt:
(1) 1 0 1 0 −3 −3 8 2 −5
(2) 3 0 5 0 −4 −9 21
(3) −15 0 3 0 −9
(4) 585 1125 −2205
(5) −18885150 24907500
(6) 527933700

5.3.4 Subresultante polynomiale Restfolge

Collins hat (1967) noch eine andere Wahl für die Werte der β_i vorgeschlagen, mit denen sich der sogenannte Subresultante gcd-Algorithmus ergibt. Die Werte der β_i sind folgendermaßen definiert:
Sei $\delta_i = n_i - n_{i+1}$, wobei n_i der Grad von $F_i(x)$ aus der PRS ist.

$$\begin{aligned} \beta_3 &= (-1)^{\delta_1+1} \\ \beta_i &= (-1)^{\delta_{i-2}+1} f_{i-2} h_{i-2}^{\delta_{i-2}} \quad \text{für } i > 3, \end{aligned}$$

wobei f_{i-2} der Leitkoeffizient von $F_{i-2}(x)$ ist und

$$h_2 = f_2^{\delta_1}, \qquad h_i = f_i^{\delta_{i-1}} h_{i-1}^{1-\delta_{i-1}} \quad (i > 2).$$

Ist $\delta_i = 1$ für alle i, dann heißt die PRS **normal**, andernfalls anormal. Es ist leicht zu sehen, daß der anormale Fall zu einem schnelleren Anwachsen der Koeffizienten führt. Dieser Algorithmus liefert in unserem Beispiel folgende PRS:

(1) $1\ 0\ 1\ 0\ -3\ -3\ \ 8\ 2\ -5$
(2) $3\ 0\ 5\ 0\ -4\ -9\ 21$
(3) $-15\ 0\ 3\ 0\ -9$
(4) $65\ 125\ -245$
(5) $9326\ -12300$
(6) 260708

Der Subresultante gcd-Algorithmus führt zu weniger gcd-Berechnungen der Koeffizienten.

Subresultante gcd-**Algorithmus**

Eingabe: $u(x), v(x)$ von Null verschiedene Polynome aus einem faktoriellen Ring (ZPE-Ring) $S[x]$. { Es wird angenommen, daß Algorithmen zur Berechnung des gcd von Elementen aus S und zur Division von a durch b in S für $b \neq 0$ und a ein Vielfaches von b existieren. }

Ausgabe: $\text{gcd}(u(x), v(x))$

1. [Reduziere auf Primitive] $d \leftarrow gcd(\text{cont}(u), \text{cont}(v))$,
   $(u(x), v(x)) \leftarrow (\text{pp}(u(x)), \text{pp}(v(x)))$,
   $g \leftarrow h \leftarrow 1$.

2. [Pseudo-Division] $\delta \leftarrow deg(u) - deg(v)$
   Berechne den Rest $r(x)$ mit dem Pseudo-Divisions-Algorithmus.
   Falls $r(x) = 0$, dann gehe nach (4)
   Falls $deg(r(x)) = 0$, dann $v(x) \leftarrow 1$, gehe nach (4)

3. [Berichtige den Rest] $u(x) \leftarrow v(x)$, $v(x) \leftarrow \dfrac{r(x)}{gh^\delta}$
   (Alle Koeffizienten von $r(x)$ sind Vielfache von gh^δ)
   $g \leftarrow lc(u)$, $h \leftarrow h^{1-\delta} g^\delta$, gehe nach (2).

4. [Binde Inhalt an] liefere $d \cdot \mathrm{pp}(v(x))$ zurück $\square$

Vergleiche

- Alle diese gcd-Algorithmen profitieren auf eine sehr natürliche Weise von der Größe des gcd. In diesen Fall wird die gcd-Berechnung kurz und das Anwachsen der Koeffizienten ist annehmbar.
- Für gcd = 1 weisen alle obigen Verfahren einen gewissen Grad von Ineffizienz auf. Trotzdem können einige Vergleiche gemacht werden:
  - Der euklidische Algorithmus hat einen großen Koeffizientenwachstum.
  - Der reduzierte gcd-Algorithmus ist nie besser als der Subresultante gcd-Algorithmus.
  - Der Primitive gcd-Algorithmus ist im Fall einer normalen PRS manchmal besser als der Subresultante gcd-Algorithmus.

Wegen der letzten Bemerkung, hier noch ein Entwurf für den euklidischen Primitive-PRS Algorithmus:

euklidischer Primitive-PRS Algorithmus

Eingabe: $a(x), b(x) \in D[x]$ mit D ein ZPE-Ring

Ausgabe: $g(x) = \mathrm{gcd}(a(x), b(x))$

$c(x) \leftarrow \mathrm{pp}(a(x))$; $d(x) \leftarrow \mathrm{pp}(b(x))$
while $d(x) \neq 0$ **do**
  $r(x) \leftarrow \mathrm{prem}[c(x), d(x)]$
  $c(x) \leftarrow d(x)$
  $d(x) \leftarrow \mathrm{pp}(r(x))$
$\gamma \leftarrow \mathrm{gcd}(\mathrm{cont}(a(x)), \mathrm{cont}(b(x)))$
$g(x) \leftarrow \gamma c(x)$; **return** $g(x)$ $\square$

Beispiel:

$$\begin{aligned} a(x) &= 48x^3 - 84x^2 + 42x - 36 \\ b(x) &= -4x^3 - 10x^2 + 44x - 30 \end{aligned}$$

nach Iteration	$r(x)$	$c(x)$	$d(x)$
0	-	$8x^3 - 14x^2 + 7x - 6$	$2x^3 + 5x^2 - 22x + 15$
1	$-68x^2 + 190x - 132$	$2x^3 + 5x^2 - 22x + 15$	$34x^2 - 95x + 66$
2	$4280x - 6420$	$34x^2 - 95x + 66$	$2x - 3$
3	0	$2x - 3$	0

Aus $\gamma = \gcd(6, 2)$ ergibt sich nun $g(x) = 2(2x - 3) = 4x - 6$.

5.3.5 gcd-Berechnung multivariater Polynome

Der obige Algorithmus kann zum Berechnen von gcd's in multivariaten Polynomringen herangezogen werden. Der Algorithmus wird auf jede Unbestimmte angewandt. Um dies am folgenden Beispiel auszutesten, braucht man sich nur daran erinnern, daß

$$a(x) = \underbrace{U(a(x))}_{\text{Einheits-Teil von } a} \operatorname{cont}(a(x))\operatorname{pp}(a(x))$$

gilt.

Beispiel: $a(x, y), b(x, y) \in \mathbf{Z}[x, y]$ (x wird als Hauptvariable gewählt)

$$\begin{aligned} a(x, y) &= (-30y)x^3 + (90y^2 + 15)x^2 - (60y)x + (45y^2) \\ b(x, y) &= (100y - 140)x^2 - (250y^2 - 350y)x - (150y^3 - 210y^2) \end{aligned}$$

Der erste Schritt verlangt, den Einheits-Teil und den Inhalt von jedem Polynom zu beseitigen. Dies fordert eine rekursive Anwendung des Algorithmus, um die gcd's in $\mathbf{Z}[y]$ zu berechnen.

$$\begin{aligned}
U(a(x,y)) &= -1 \qquad U(b(x,y)) = 1 \\
\mathrm{cont}(a(x,y)) &= \gcd(30y,\, -(90y^2+15),\, 60y,\, 45y^2) = 15 \\
\mathrm{cont}(b(x,y)) &= \gcd(100y-140,\, 250y^2-350y,\, 150y^3-210y^2) \\
&= 50y-70 \\
\text{somit } \mathrm{pp}(a(x,y)) &= (2y)x^3-(6y^2+1)x^2+(4y)x-(3y^2) \\
\mathrm{pp}(b(x,y)) &= (2)x^2-(5y)x-(3y^2)
\end{aligned}$$

nach Iteration	$r(x)$	$c(x)$	$d(x)$
0	-	$(2y)x^3-(6y^2+1)x^2+(4y)x-(3y^2)$	$(2)x^2-(5y)x-3y^2$
1	$(2y^3+6y)x-(6y^4+18y^2)$	$(2)x^2-(5y)x-3y^2$	$x-(3y)$
2	0	$x-(3y)$	0

Aus $\gamma = \gcd(15,\, 50y-70)$ ergibt sich $g(x) = 5(x-(3y)) = 5x-15y$.

5.3.6 Der erweiterte euklidische Algorithmus

In einem euklidischen Ring D kann der gcd g zweier Elemente $a, b \in D$ (nicht beide gleich Null) immer in der Form

$$g = sa + tb$$

dargestellt werden. Mit dem folgenden Algorithmus kann man neben g auch noch s und t berechnen.

Algorithmus (Erweiterter euklidischer Algorithmus)

Eingabe: $a, b \in D$, nicht beide Null.

Ausgabe: $g, s, t \in D$ mit $g = \gcd(a,b) = sa + tb$.

begin
$(a_0, a_1) \leftarrow (a, b)$;
$(s_0, s_1) \leftarrow (1, 0)$;
$(t_0, t_1) \leftarrow (0, 1)$;
{Schleifeninvariante $a_0 = s_0 a + t_0 b \quad a_1 = s_1 a + t_1 b$}
while $a_1 \neq 0$ **do**
begin
$q \leftarrow a_0 \text{ div } a_1$;
$(a_0, a_1) \leftarrow (a_1, a_0 - a_1 \cdot q)$;
$(s_0, s_1) \leftarrow (s_1, s_0 - s_1 \cdot q)$;
$(t_0, t_1) \leftarrow (t_1, t_0 - t_1 \cdot q)$
end;
$g \leftarrow a_0$;
$s \leftarrow s_0$;
$t \leftarrow t_0$
end

Beispiel 1:
Der erweiterte euklidische Algorithmus liefert mit den Eingabewerten $a = 228$ und $b = 612$ über $\mathbb{Z}$ die folgenden Werte:

Schritt	q	a_0	a_1	s_0	s_1	t_0	t_1
0	—	228	612	1	0	0	1
1	0	612	228	0	1	1	0
2	2	228	156	1	-2	0	1
3	1	156	72	-2	3	1	-1
4	2	72	12	3	-8	-1	3
5	6	12	0	-8	51	3	-19 .

Es wird also $g = \gcd(228, 612) = 12$, $s = -8$ und $t = 3$ ausgegeben. Dieses Ergebnis kann leicht durch $12 = -8 \cdot 228 + 3 \cdot 612$ nachgeprüft werden.

Beispiel 2:
Sei $a(x) = x^3 + 3$ und $b(x) = 2x^2 + 2$ über $\mathbb{Z}_5[x]$. Da $\mathbb{Z}_5$ ein Körper und damit $\mathbb{Z}_5[x]$ ein euklidischer Ring ist, kann der Algorithmus angewendet werden:

Schritt	q	a_0	a_1	s_0	s_1	t_0	t_1
0	—	$x^3 + 3$	$2x^2 + 2$	1	0	0	1
1	$3x$	$2x^2 + 2$	$4x + 3$	0	1	1	$2x$
2	$3x + 4$	$4x + 3$	0	1	$2x + 1$	$2x$	$4x^2 + 2x + 1$

Daraus erhält man $g = 4x + 3 \sim x + 2 = \gcd(x^3 + 3, 2x^2 + 2)$, $s = 1$ und $t = 2x$. Die Ausgabe des Algorithmus besagt, daß der gcd

$$g = 4x + 3 = 1(x^3 + 3) + (2x)(2x^2 + 2)$$

aus $a(x)$ und $b(x)$ linear kombiniert werden kann.

Kapitel 6

Homomorphismen und Chinesischer Restealgorithmus

Ein wichtiges Konzept der algebraischen Ringtheorie ist das der Homomorphismen, mit dem ein Ring auf einen anderen Ring abgebildet wird. Dieses Konzept hat sich als ein unverzichtbares Hilfsmittel zur Entwicklung von effizienten Algorithmen für verschiedene Probleme in symbolischem Rechnen herausgestellt. Die grundlegende Methode in diesen Algorithmen ist, einen gegebenen Bereich auf einen oder mehrere einfachere Bereiche (homomorphe Bilder) abzubilden, die gewünschte Berechnung in diesen Bereichen vorzunehmen und dann aus diesen Ergebnissen das Ergebnis für den Originalbereich zu rekonstruieren. In diesem Kapitel wird die algebraische Theorie der Homomorphismen eingeführt und dann der ganzzahlige und polynomiale Chinesische-Reste-Algorithmus zur Ausführung der oben gemeinten 'Rekonstruktion' entwickelt.

6.1 Ringmorphismen

Definition 6.1
Sei S eine Algebra. Eine Teilmenge S' von S heißt **Unteralgebra**, *wenn S' unter den auf S definierten Operationen abgeschlossen ist.*

Es wird hier folgende Terminologie für Unteralgebren der obigen speziellen Algebren benutzt. Ist S ein (kommutativer) Ring, ein Integritätsbereich, ZPE-Ring oder ein euklidscher Ring dann heißt eine Unteralgebra von S *Unterring*. Ist S ein Körper, so heißt eine Unteralgebra von S *Unterkörper*. Es ist klar, daß, falls eine Teilmenge S' einer beliebigen Algebra S unter allen auf S definierten Operationen abgeschlossen ist, automatisch alle Axiome, die in S gelten, auch auf die Unteralgebra S' übertragbar sind.

Insbesondere ist ein Unterring eines kommutativen (Integritäts-, ZPE-, euklidschen) Ringes selbst wieder ein kommutativer (Integritäts-, ZPE-, euklidscher) Ring.

6.1.1 Morphismen

Bei der Betrachtung der Abbildungen zwischen zwei Ringen R und R' wird hier die Konvention, gleiche Schreibweise für Operationen in R und R' zu benutzen, angewandt. D.h. $+$ bezeichnet die Addition in R oder in R', die Multiplikation wird in R und R' durch Nebeneinanderstellung ohne ein Operationszeichen ausgedrückt und '0' bzw. '1' stehen (abhängig vom Kontext) für das additive bzw. multiplikative neutrale Element in R oder R'. Diese Konvention eignet sich besonders für den Fall, wenn einer der beiden Ringe R, R' ein Unterring des anderen ist.

Definition 6.2
Sind R und R' Ringe, dann heißt die Abbildung $\phi : R \longrightarrow R'$ ein **Ringmorphismus**, *falls gilt:*

(i) $\phi(a + b) = \phi(a) + \phi(b)$ *für alle* $a, b \in \mathbf{R}$

(ii) $\phi(ab) = \phi(a)\phi(b)$ *für alle* $a, b \in \mathbf{R}$

(iii) $\phi(1) = 1$.

Das allgemeine (universelle Algebra-)Konzept eines *Morphismus* zwischen zwei Algebren ist das einer Abbildung, die alle Operationen die auf den Algebren definiert sind, erhält. In der Definition 6.2 sieht man, die Eigenschaften (i) – (iii) stellen sicher, daß drei der Ringoperationen erhalten werden, dagegen werden die Operationen Negation und 'erhalte 0' nicht erwähnt. Das kommt daher, weil die beiden zusätzlichen Eigenschaften

$$\begin{aligned} &\phi(0) = 0 \\ &\phi(-a) = -\phi(a) \quad \text{für alle } a \in \mathbf{R} \end{aligned}$$

einfache Folgerungen aus den Ringaxiomen und den Eigenschaften (i) – (iii) sind. Ähnlich ergibt sich, falls R und R' Körper sind, mit der zusätzlichen einstelligen Operation der Inversion, daß der Ringmorphismus aus der Definition 6.2 ein *Körpermorphismus* ist, weil sich die zusätzliche Eigenschaft

$$\phi(a^{-1}) = [\phi(a)]^{-1} \quad \text{für alle } a \in \mathbf{R} \setminus \{0\}$$

als Konsequenz der Körperaxiome und der Eigenschaften (i) – (iii) ergibt. Daraus wird auch verständlich, weshalb im Folgenden, wenn auf ein Morphismus verwiesen wird, immer auf den Ringmorphismus, wie in Definition 6.2 erklärt, bezug genommen wird. Die Morphismen sind bezüglich ihrer Eigenschaften als Funktionen klassifiziert. Ist $\phi : R \longrightarrow R'$ ein Morphismus, dann heißt er *Monomorphismus*, falls ϕ injektiv (one-to-one), *Epimorphismus*, falls ϕ surjektiv (onto) und ein *Isomorphismus*, falls ϕ bijektiv (one-to-one and onto) ist. Der klassische Ausdruck *Homomorphismus* ist in den meisten Fällen nur ein Synonym für den moderneren Ausdruck 'Morphismus', der im Kontext universeller Algebren benutzt wird. Im gewöhnlichen Gebrauch wird der Ausdruck 'Homomorphismus' mit einem Epimorphismus gleichgesetzt, insbesondere heißt R', falls $\phi : R \longrightarrow R'$ ein Epimorphismus ist, das *homomorphe Bild* von R.

Ein Monomorphismus $\phi : R \longrightarrow R'$ heißt *Einbettung* von R in R', wenn die Abbildung $\phi : R \longrightarrow \phi(R)$ auf die Bildmenge

$$\phi(R) = \{r' \in R' : \phi(r) = r' \text{ für ein } r \in R\}$$

ein Isomorpismus ist – der Ring R' enthält R (korrekter, eine isomorphe Kopie von R) als Unterring. Ein Epimorphismus $\phi : R \longrightarrow R'$ heißt *Projektion* von R auf das homomorphe Bild R'. Es ist klar, daß in dieser Terminologie für jeden Morphismus $\phi : R \longrightarrow R'$, die Bildmenge $\phi(R)$ ein homomorphes Bild von R ist. Eine wichtige Eigenschaft der Morphismen ist, daß das homomorphe Bild eines (kommutativen) Ringes selbst wieder ein (kommutativer) Ring ist. Jedoch muß das homomorphe Bild eines Integritätsbereiches nicht notwendig wieder ein Integritätsbereich sein.

Beispiele:

1. Sei D ein Integritätsbereich und F_D sein Quotientenkörper. Die Abbildung $\phi : D \longrightarrow F_D$,die definiert ist durch

$$\phi(a) = [a/1] \quad \forall a \in D,$$

   ist ein Monomorphismus. D.h. ϕ ist eine Einbettung von D in F_D und man nennt F_D eine *Erweiterung* von D (die kleinste Erweiterung des Integritätsbereiches D auf einen Körper).

2. $\mathbf{Z}$: Ganze Zahlen, $\mathbf{Z}_6$: Menge der ganzen Zahlen modulo 6
   Die Abbildung $\phi : \mathbf{Z} \longrightarrow \mathbf{Z}_6$ ist definiert durch

$$\phi(a) = \text{rem}(a, 6) \quad \forall a \in \mathbf{Z} \quad \text{(rem: remainder)}.$$

   Damit ist ϕ ein Epimorphismus und eine Projektion von $\mathbf{Z}$ auf das homomorphe Bild $\mathbf{Z}_6$. $\mathbf{Z}_6$ ist ein kommutativer Ring, weil $\mathbf{Z}$ ebenfalls einer ist. Dagegen ist $\mathbf{Z}_6$ kein Integritätsbereich, obwohl $\mathbf{Z}$ einer ist.

6.1.2 Modulo- und Auswertungshomomorphismus

Zwei besondere Arten von Homomorphismen haben viele praktische Anwendungen in der Computer-Algebra.

6.1.2.1 Modulo-Homomorphismen

(Eine Verallgemeinerung des letzten Beispieles.)

$$\phi_m : \mathbf{Z}[x_1, \ldots, x_\nu] \longrightarrow \mathbf{Z}_m[x_1, \ldots, x_\nu]$$

ist ein Homomorphismus, der für eine feste Zahl $m \in \mathbf{Z}$ durch folgende zwei Regeln definiert ist:

$$\begin{aligned} \phi_m(x_i) &= x_i, \quad \text{für } 1 \le i \le \nu, \\ \phi_m(a) &= \text{rem}(a,\, m), \quad \text{für alle Koeffizienten } a \in \mathbf{Z}. \end{aligned}$$

Mit anderen Worten, ein Modulo-Homomorphismus ϕ_m ist eine Projektion von $\mathbf{Z}[x_1, \ldots, x_\nu]$ auf $\mathbf{Z}_m[x_1, \ldots, x_\nu]$, die man einfach durch Ersetzen jedes Koeffizienten eines Polynoms $a(x) \in \mathbf{Z}[x_1, \ldots, x_\nu]$ mit seiner 'modulo m' - Darstellung erhält. Natürlich bleibt ϕ_m für den Fall $\nu = 0$ wohldefiniert, denn dann ist er einfach eine Projektion von $\mathbf{Z}$ auf $\mathbf{Z}_m$.

Beispiel:
In $\mathbf{Z}[x,\, y]$ seien $a(x, y)$ und $b(x, y)$ die Polynome

(1) $a(x, y) = 3x^2y^2 - x^2y + 5x^2 + xy^2 - 3xy$,
(2) $b(x, y) = 2xy + 7x + y^2 - 2$.

Der Modulo-Homomorphismus ϕ_5 bildet diese zwei Polynome auf die beiden folgenden Polynome in $\mathbf{Z}_5[x,\, y]$ ab:

$\phi_5(a(x, y)) = 3x^2y^2 + 4x^2y + xy^2 + 2xy$,
$\phi_5(b(x, y)) = 2xy + 2x + y^2 + 3$.

Ähnlich bildet der Modulo-Homomorphismus ϕ_7 (1) - (2) auf die folgenden Polynome in $\mathbf{Z}_7[x,\, y]$ ab:

$\phi_7(a(x, y)) = 3x^2y^2 + 6x^2y + 5x^2 + xy^2 + 4xy$,
$\phi_7(b(x, y)) = 2xy + y^2 + 5$.

6.1.2.2 Auswertungshomomorphismen

Ein Auswertungshomomorphismus

$$\phi_{x_i-\alpha}: D[x_1,\ldots,x_\nu] \longrightarrow D[x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_\nu]$$

ist definiert für eine bestimmte Unbekannte x_i und ein festes Element $\alpha \in D$, so daß für jedes Polynom $a(x_1,\ldots,x_\nu) \in D[x_1,\ldots,x_\nu]$ gilt:

$$\phi_{x_i-\alpha}(a(x_1,\ldots,x_\nu)) = a(x_1,\ldots,x_{i-1},\alpha,x_{i+1},\ldots,x_\nu).$$

Mit anderen Worten, ein Auswertungshomomorphismus $\phi_{x_i-\alpha}$ ist eine Projektion von $D[x_1,\ldots,x_\nu]$ auf $D[x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_\nu]$, die man einfach durch Substituieren des Wertes $\alpha \in D$ für die i-te Unbekannte x_i erhält. Deshalb kann die Schreibweise $\phi_{x_i-\alpha}$ als 'substituiere α für x_i' gelesen werden. (Die besondere Wahl der Schreibweise $\phi_{x_i-\alpha}$ für einen Auswertungshomomorphismus wird deshalb gemacht, daß der Index $x_i-\alpha$ mit dem Index m zur Notation ϕ_m für einen Modulo-Homomorphismus korrespondiert.)

Beispiel:
Es seien

$$\begin{aligned} a(x) &= 7x+5, \\ b(x) &= 2x-3 \end{aligned}$$

im Ring $\mathbf{Z}[x]$ gegeben. Angenommen man möchte das Produkt

$$c(x) = a(x)b(x)$$

der beiden Polynome bestimmen.
Besser wie direktes multiplizieren der beiden Polynome im Ring $\mathbf{Z}[x]$ wäre, $\mathbf{Z}[x]$ auf homomorphe Bilder $\mathbf{Z}_p$ zu projizieren und die (einfachere) Multiplikation im Körper $\mathbf{Z}_p$ durchzuführen. Zum Beispiel bildet der zusammengesetzte Homomorphismus

$$\phi_{x-0}\phi_5 : \mathbf{Z}[x] \longrightarrow \mathbf{Z}_5,$$

$a(x)$und $b(x)$ folgendermaßen ab:

$$\begin{aligned} a(x) &\xrightarrow{\phi_5} 2x \xrightarrow{\phi_{x-0}} 0, \\ b(x) &\xrightarrow{\phi_5} 2x+2 \xrightarrow{\phi_{x-0}} 2. \end{aligned}$$

Somit ist das Produkt im homomorphen Bild $\mathbf{Z}_5$ $0 \times 2 = 0$. Bei Benutzung der Standardnotation für die 'mod p' - Arithmetik ergibt dies

$$c(0) \equiv 0 (\mathrm{mod}\ 5).$$

Ähnlich führt das Anwenden des zusammengesetzten Homomorphismus $\phi_{x-1}\phi_5$ auf

$$\begin{aligned} a(x) &\xrightarrow{\phi_5} 2x \xrightarrow{\phi_{x-1}} 2, \\ b(x) &\xrightarrow{\phi_5} 2x+2 \xrightarrow{\phi_{x-1}} 4. \end{aligned}$$

Das Produkt in $\mathbf{Z}_5$ ist $2 \times 4 = 3$ und damit

$$c(1) \equiv 3 (\bmod\ 5).$$

Genauso findet man

$$c(2) \equiv 4 (\bmod\ 5)$$

bei Verwenden des zusammengesetzten Homomorphismus $\phi_{x-2}\phi_5$. Wenn man zusätzlich das Tripel

$$\phi_{x-0}\phi_7, \quad \phi_{x-1}\phi_7, \quad \phi_{x-2}\phi_7$$

zusammengesetzter Homomorphismen anwendet, wobei jeder $\mathbf{Z}[x]$ auf $\mathbf{Z}_7$ projiziert, erhält man

$$\begin{aligned} c(0) &\equiv 6 (\bmod\ 7), \\ c(1) &\equiv 2 (\bmod\ 7), \\ c(2) &\equiv 5 (\bmod\ 7). \end{aligned}$$

Der obige Prozeß ist nur brauchbar, wenn man den Homomorphismus invertieren kann, um das Polynom $c(x) \in \mathbf{Z}[x]$ aus den gegebenen Informationen der Bilder von $c(x)$ im Körper $\mathbf{Z}_p$ zu rekonstruieren. Der inverse Prozeß zieht das Konzept der Interpolation und der chinesischen Restetechnik, das später in diesem Kapitel vorgestellt wird, nach sich. Kurz, da $\deg(c(x)) = \deg(a(x)) + \deg(b(x)) = 2$ bekannt ist, ist das Polynom $c(x)$ vollständig durch die Werte an den 3 Punkten bestimmt. Bei Verwendung der obigen Informationen erhält man durch Interpolation

$$\begin{aligned} c(x) &\equiv 4x^2 + 4x (\bmod\ 5), \\ c(x) &\equiv 3x + 6 (\bmod\ 7). \end{aligned}$$

Damit sind die Bilder von $c(x)$ in $\mathbf{Z}_5[x]$ und in $\mathbf{Z}_7[x]$ bekannt. Nun kann man

$$c(x) = c_2x^2 + c_1x + c_0 \in \mathbf{Z}[x]$$

mit Hilfe der chinesischen Restetechnik bestimmmen. Ist z.B.

$$c_2 \equiv 4 (\bmod\ 5) \quad \text{und} \quad c_2 \equiv 0 (\bmod\ 7)$$

bekannt, kann man berechnen, daß

$$c_2 \equiv 14 (\bmod\ 35)$$

(wobei $35 = 5 \times 7$) gilt. Man erhält schließlich:

$$c(x) = 14x^2 - 11x - 15 \in \mathbf{Z}[x].$$

6.2 Charakterisierung der Morphismen

Definition 6.3
Sei R ein kommutativer Ring. Eine nichtleere Teilmenge I von R heißt **Ideal***, falls gilt:*

(i) $a - b \in I$ *für alle* $a, b \in I$,

(ii) $ar \in I$ *für alle* $a \in I$ *und für alle* $r \in R$.

Zwei sehr spezielle Ideale eines beliebigen kommutativen Ringes sind die Teilmengen $\{0\}$ und R, da die Eigenschaften (i) und (ii) der Definition 6.3 klar erfüllt sind. Man nennt $\{0\}$ das *Nullideal* und R das *universelle Ideal.* Unter einem *echten Ideal* versteht man ein, von den beiden trivialen Idealen verschiedenes Ideal. Man beachte, die Teilmenge $\{0\}$ ist kein Unterring von R bezüglich der Definition 6.1, da sie unter der nullstelligen Operation 'wähle 1' die auf R definiert ist, nicht abgeschlossen ist (d.h. in $\{0\}$ ist das multiplikative neutrale Element von R nicht enthalten). Dies ist eine charakteristische Eigenschaft von Idealen, die im folgenden Satz formuliert ist.

Satz 6.4
Jedes echte Ideal I eines kommutativen Ringes R ist, bis auf die nullstellige Operation 'wähle 1', abgeschlossen unter allen auf R definierten Ringoperationen (d.h. $1 \notin I$).

Beweis:
Es ist leicht nachzuprüfen, daß die Eigenschaft (i) der Definition 6.3 garantiert, daß I abgeschlossen ist unter den Operationen $+$ (zweistellig), $-$ (einstellig) und 'wähle 0' (nullstellig). (Eigenschaft (i) wird exakt gebraucht, weil es ausreicht die Abgeschlossenheit in bezug auf die drei 'Gruppen'-Operationen zu sichern). Ebenso ist trivial, daß (ii) die Abgeschlossenheit von I unter der Multiplikation sicher stellt. Ist $1 \in I$, dann gilt mit (ii) $r \in I$ für alle $r \in R$ – d.h. es ist $I = R$, so daß I kein echtes Ideal ist. $\square$

Die entscheidende Eigenschaft eines Ideals I ist – abgesehen von der Abgeschlossenheitseigenschaft des Satzes 6.4 – die 'erweiterte Abgeschlossenheits'-Eigenschaft (ii) der Definition 6.3, die sichert, daß I unter der Multiplikation abgeschlossen ist bezüglich *jedem Element des Ringes R.*

Beispiel:
Im Integritätsbereich **Z** ist die Teilmenge

$$\langle m\rangle = \{mr : r = 0, \pm 1, \pm 2, \ldots\}$$

für ein festes $m \in \mathbf{Z}$ ein Ideal, genauer ein durch m *erzeugtes Ideal.* Z.B. ist das Ideal $\langle 4 \rangle$, die Menge

$$\langle 4 \rangle = \{0, \pm 4, \pm 8, \pm 12, \ldots\}.$$

Beispiel:
Im Polynomring $\mathbf{Q}[x]$ ist die Teilmenge

$$\langle p(x) \rangle = \{p(x)a(x) : a(x) \in \mathbf{Q}[x]\}$$

für ein festes Polynom $p(x) \in \mathbf{Q}[x]$ ein Ideal, genauer ein durch $p(x)$ *erzeugtes Ideal.* Z.B., ist das Ideal $\langle x - \alpha \rangle$ für ein festes $\alpha \in \mathbf{Q}$ die Menge aller Polynome über $\mathbf{Q}$, die $x - \alpha$ als Faktor haben (d.h. Polynome $a(x)$, für die $a(\alpha) = 0$ gilt).

Beispiel:
In dem bivariaten Polynomring $\mathbf{Z}[x, y]$ ist die Teilmenge

$$\begin{aligned} \langle p_1(x,y),\, p_2(x,y) \rangle \;=\; & \{p_1(x,y)a_1(x,y) + p_2(x,y)a_2(x,y) : \\ & a_1(x,y),\, a_2(x,y) \in \mathbf{Z}[x,y]\} \end{aligned}$$

für feste Polynome $p_1(x,y)$, $p_2(x,y) \in \mathbf{Z}[x,y]$ ein Ideal, genauer ein durch $p_1(x,y)$ *und* $p_2(x,y)$ *erzeugtes Ideal.* Z.B. ist das Ideal $\langle x, y \rangle$ die Menge aller bivariaten Polynome über $\mathbf{Z}$ mit 0 als konstantem Term. Man beachte auch, daß $\langle y - \alpha \rangle$ für festes $\alpha \in \mathbf{Z}$ ein Ideal in $\mathbf{Z}[x,y]$ ist, das alle bivariaten Polynome über $\mathbf{Z}$ enthält, die $y - \alpha$ als Faktor haben (d.h. Polynome $a(x,y)$, für die $a(x,\alpha) = 0$ gilt). Auch $\langle m \rangle$ ist für festes $m \in \mathbf{Z}$ ein Ideal in $\mathbf{Z}[x,y]$, das alle bivariaten Polynome enthält, deren ganzzahligen Koeffizienten Vielfache von m sind.

Definition 6.5
Ein Ideal I in einem kommutativen Ring R heißt **endlich erzeugt**, *wenn I durch die Menge $\langle a_1, \ldots, a_n \rangle$ aller Linearkombinationen von n Elementen $a_1, \ldots, a_n \in R$ dargestellt werden kann.*

Definition 6.6
Ein Ideal I in einem kommutativen Ring R heißt **Hauptideal**, *wenn I durch die Menge $\langle a \rangle$ aller Vielfachen eines einzigen Elementes $a \in R$ dargestellt werden kann.*

Definition 6.7
Ein Integritätsbereich D heißt **noetherscher Integritätsbereich**, *wenn jedes Ideal in D endlich erzeugt ist.*

Definition 6.8
Ein Integritätsbereich D heißt **Hauptidealring**, *wenn jedes Ideal in D ein Hauptideal ist.*

Definition 6.9
Es seien I und J zwei Ideale in einem noetherschen Integritätsbereich D und die beiden Ideale gegeben durch $I = \langle a_1, \ldots, a_n\rangle$, $J = \langle b_1, \ldots, b_m\rangle$ mit den Elementen $a_i \in D\,(1 \leq i \leq n)$, $b_j \in D\,(1 \leq j \leq m)$.

(i) *Die* **Summe** *der Ideale I und J in D ist das Ideal $\langle I, J\rangle = \langle a_1, \ldots, a_n, b_1, \ldots, b_m\rangle$. Man beachte, daß das Ideal $\langle I, J\rangle$ aus alle möglichen Summen $a + b$ mit $a \in I$ und $b \in J$ besteht.*

(ii) *Das* **Produkt** *$I \cdot J$ der Ideale I und J in D ist das Ideal, das durch alle Elemente $a_i b_j$ mit a_i ein Basiselement von I und b_j ein Basiselement von J erzeugt wird. Daher kann das Produkt geschrieben werden als*

$$\begin{aligned} I \cdot J &= \langle a_1 b_1, a_1 b_2, \ldots, a_1 b_m, a_2 b_1, a_2 b_2, \ldots, a_2 b_m, \\ &\quad \ldots, a_n b_1, a_n b_2, \ldots, a_n b_m\rangle. \end{aligned}$$

(iii) *Die* **i-te Potenz** *eines Ideales I in D (i eine positive ganze Zahl) ist, wie folgt, rekursiv über die Produktterme von Idealen definiert:*

$$\begin{aligned} I^1 &= I, \\ I^i &= I \cdot I^{i-1} \quad \text{für } i \geq 2. \end{aligned}$$

Definition 6.10
Seien R und R' kommutative Ringe und $\phi : R \longrightarrow R'$ ein Morphismus. Der **Kern** *K des Morphismus ϕ ist die Menge*

$$K = \phi^{-1}(0) = \{a : a \in R \ \text{ und } \phi(a) = 0\}.$$

Satz 6.11
Seien R und R' kommutative Ringe. Der Kern K eines Morphismus $\phi : R \longrightarrow R'$ ist ein Ideal in R.

Beweis:
Die Menge K ist wegen $\phi(0) = 0$ nicht leer. Sind $a, b \in K$, dann ist

$$\phi(a - b) = \phi(a) - \phi(b) = 0 - 0 = 0,$$

so daß $a - b \in K$ gilt, das die Eigenschaft (i) der Definition 6.3 beweist. Ähnlich gilt auch (ii), weil für $a \in K$ und $r \in R$ gilt

$$\phi(ar) = \phi(a)\phi(r) = 0 \cdot \phi(r) = 0,$$

also $ar \in K$. □

Der folgende (wichtige) Satz zeigt, daß ein homomorphes Bild eines kommutativen Ringes, bis auf Isomorphie, durch das Ideal aus Elementen, die auf 0 abgebildet werden (d.h. durch den Kern), vollständig bestimmt ist.

Satz 6.12 (Charakterisierungssatz)
Sei R ein kommutativer Ring und K ein Ideal in R. Sind $\phi_1 : R \longrightarrow R'$ und $\phi_2 : R \longrightarrow R'$ zwei Morphismen, die beide den Kern K haben, dann ist die Korrespondenz zwischen den beiden homomorphen Bildern $\phi_1(R)$ und $\phi_2(R)$

$$(\phi_1(a)\ ,\ \phi_2(a)) \quad \textit{für alle } a \in R$$

ein Isomorphismus.

Beweis:
Angenommen ϕ_1 und ϕ_2 haben den Kern K. Die oben gemeinte Korrespondenz kann formal wie folgt spezifiziert werden. Für ein beliebiges Element $\alpha \in \phi_1(R)$ ist die Urbildmenge von α die Menge

$$\phi_1^{-1}(\alpha) = \{a \in R : \phi_1(a) = \alpha\}.$$

Man definiert die Abbildung

$$\psi : \phi_1(R) \longrightarrow \phi_2(R)$$

durch

$$\psi(\alpha) = \phi_2(\phi_1^{-1}(\alpha)) \quad \text{für alle } \alpha \in \phi_1(R), \tag{6.1}$$

wobei gefordert wird, daß das Bild der Menge $\phi_1^{-1}(\alpha)$ unter ϕ_2 ein *einzelnes* Element in $\phi_2(R)$ ist. Um dies zu sehen, beachte man, daß für zwei Elemente $a, b \in R$ aus der Menge $\phi_1^{-1}(\alpha)$ $\quad a - b \in K$ (der Kern von ϕ_1) gilt, weil

$$\phi_1(a - b) = \phi_1(a) - \phi_1(b) = \alpha - \alpha = 0.$$

Da K auch der Kern von ϕ_2 ist, folgt hieraus mit

$$\phi_2(a - b) = \phi_2(a) - \phi_2(b) = 0,$$

daß

$$\phi_2(a) = \phi_2(b)$$

ist. Damit wird durch (6.1) eine gültige Abbildung von $\phi_1(R)$ nach $\phi_2(R)$ definiert und es ist klar, daß ψ die obige Korrespondenz spezifiziert. Man kann (6.1) berechnen, indem man ein beliebiges $a \in \phi_1^{-1}(\alpha)$ als Urbild von α nimmt und $\psi(\alpha) = \phi_2(a)$ setzt. Nun fordert man, daß ψ ein Isomorphismus ist. ψ erfüllt die Eigenschaften (i) – (iii) der Definition 6.2, da diesen die Morphismen ϕ_1 und ϕ_2 genügen. Denn sind für $\alpha, \beta \in \phi_1(R)$ $a \in \phi_1^{-1}(\alpha)$ und $b \in \phi_1^{-1}(\beta)$ die jeweiligen Urbilder von α und β, dann ist das Element $a + b \in R$ ein Urbild von $\alpha + \beta \in \phi_1(R)$, weil

$$\phi_1(a + b) = \phi_1(a) + \phi_1(b) = \alpha + \beta$$

ist; ähnlich ist $ab \in R$ ein Urbild von $\alpha\beta \in \phi_1(R)$. Daher sind die Eigenschaften (i) und (ii) mit

$$\begin{aligned} \psi(\alpha + \beta) &= \phi_2(\phi_1^{-1}(\alpha + \beta)) = \phi_2(a + b) \\ &= \phi_2(a) + \phi_2(b) = \psi(\alpha) + \psi(\beta) \end{aligned}$$

und

$$\begin{aligned} \psi(\alpha\beta) &= \phi_2(\phi_1^{-1}(\alpha\beta)) = \phi_2(ab) \\ &= \phi_2(a)\phi_2(b) = \psi(\alpha)\psi(\beta) \end{aligned}$$

gegeben. Um Eigenschaft (iii) zu prüfen, beachte man, daß $1 \in R$ ein Urbild von $1 \in \phi_1(R)$ ist, weil $\phi_1(1) = 1$ (d.h. ϕ_1 ist ein Morphismus) gilt und daher

$$\psi(1) = \phi_2(\phi_1^{-1}(1)) = \phi_2(1) = 1$$

(weil ϕ_2 ein Morphismus ist). Damit ist gezeigt, daß ψ ein Morphismus ist. Es ist leicht zu sehen, daß die Abbildung ψ surjektiv ist, da die Abbildungen

$$\phi_1 : R \longrightarrow \phi_1(R) \ \text{ und } \phi_2 : R \longrightarrow \phi_2(R)$$

surjektiv sind. Um zu sehen, daß ψ injektiv ist, seien $a, b \in R$ die möglichen Urbilder von $\alpha, \beta \in \phi_1(R)$ (d.h. $\alpha = \phi_1(a)$ und $\beta = \phi_1(b)$) und man nimmt an es gelte $\psi(\alpha) = \psi(\beta)$. Dann gilt

$$\begin{aligned} &\phi_2(\phi_1^{-1}(\alpha)) = \phi_2(\phi_1^{-1}(\beta)) \\ &\quad \Rightarrow \phi_2(a) = \phi_2(b) \\ &\quad \Rightarrow a - b \in K \ \ \text{(der Kern von } \phi_2) \\ &\quad \Rightarrow \phi_1(a) = \phi_1(b) \ \ \text{(weil } K \text{ der Kern von } \phi_1 \text{ ist)} \\ &\quad \Rightarrow \alpha = \beta. \end{aligned}$$

Also ist die Abbildung ψ injektiv und definiert einen Isomorphismus zwischen $\phi_1(R)$ und $\phi_2(R)$. □

6.2.1 Restklassenringe

Sei R ein kommutativer Ring und I ein Ideal in R. Nun wird gezeigt, wie man ein homomorphes Bild $\phi(R)$ derart konstruieren kann, daß I der Kern des Morphismus ϕ ist. Ist $\phi : R \longrightarrow R'$ ein Morphismus mit Kern I, dann gilt

$$\phi(a) = \phi(b) \quad \text{genau dann, wenn } a - b \in I.$$

Deshalb definiert man die folgende *Kongruenzrelation* auf R:

$$a \equiv b \quad \text{genau dann, wenn } a - b \in I. \tag{6.2}$$

Es ist bereits gezeigt, daß die Kongruenzrelation $\equiv$ eine Äquivalenzrelation auf R ist und R in Äquivalenzklassen, die sogenannten *Restklassen*, unterteilt. Es ist leicht zu zeigen, daß für alle Elemente $a \in R$, jedes Element der Menge

$$a + I = \{a + c \, : \, c \in I\}$$

zur gleichen Restklasse bezüglich der Kongruenzrelation $\equiv$ gehört, also $a \in a + I$ und vielmehr, falls b in der gleichen Restklasse wie a liegt (d.h. falls $a \equiv b$), dann auch $b \in a + I$ gilt. Deshalb ist a genau in der Restklasse $a + I$ enthalten.
Die Menge aller Restklassen bezüglich der in (6.2) definierten Kongruenzrelation $\equiv$ ist die *Quotientenmenge*

$$R/I = \{a + I : \, a \in R\}$$

(gelesen als 'R modulo dem Ideal I'). Sind a und b in derselben Restklasse (d.h. $a \equiv b$), dann sind $a + I$ und $b + I$ zwei Repräsentanten für das gleiche Element der Quotientenmenge R/I. Addition und Multiplikation auf der Quotientenmenge R/I werden, durch die auf R definierten Operationen, wie folgt definiert:

$$(a + I) + (b + I) \;=\; (a + b) + I, \tag{6.3}$$
$$(a + I)\,(b + I) \;=\; (ab) + I. \tag{6.4}$$

Macht man von der Tatsache gebrauch, daß I ein Ideal ist, kann für die Operationen Addition und Multiplikation auf der Quotientenmenge R/I ihre Wohldefiniertheit von (6.3) – (6.4) im Sinne der Repräsentantenunabhängigkeit gezeigt werden.

Beispiel:
Im Integritätsbereich $\mathbb{Z}$, ist $\langle m \rangle$, für ein festes $m \in \mathbb{Z}$, ein Ideal. Damit ist der Restklassenring $\mathbb{Z}/\langle m \rangle$ ein homomorphes Bild von $\mathbb{Z}$ und $\langle m \rangle$ der Kern des kanonischen Homomorphismus $\phi : \mathbb{Z} \longrightarrow \mathbb{Z}/\langle m \rangle$. Angenommen m ist positiv, dann sind die Elemente von $\mathbb{Z}/\langle m \rangle$ gegeben durch:

$$\mathbb{Z}/\langle m \rangle = \{0 + \langle m \rangle, \, 1 + \langle m \rangle, \ldots, \, m - 1 + \langle m \rangle\}.$$

Gewöhnlich wird $\mathbb{Z}/\langle m\rangle$ als $\mathbb{Z}_m$ (der Ring der ganzen Zahlen modulo m) geschrieben und seine Elemente einfach durch

$$\{0, 1, \ldots, m-1\}$$

dargestellt. Der kanonische Homomorphismus ist genau der Modulo-Homomorphismus $\phi_m : \mathbb{Z} \longrightarrow \mathbb{Z}_m$.

6.2.2 Kongruenz-Arithmetik

Es ist hilfreich, eine formale *Kongruenz-Schreibweise* einzuführen, die bei Umformungen auf den Restklassen in dem homomorphen Bild R/I eines Ringes R gebraucht wird. Man erinnere sich, falls I ein Ideal in einem kommutativen Ring R ist, dann bilden die Restklassen (d.h. die Äquivalenzklassen) den Restklassenring R/I. Die Restklassen sind bestimmt durch die auf R definierte Kongruenzrelation $\equiv$

$$a \equiv b \quad \text{genau dann, wenn } a - b \in I.$$

Diese Relation wird gelesen als 'a kongruent b modulo I' und man schreibt dafür

$$a \equiv b (\text{mod } I).$$

Für den speziellen Fall, daß I ein Hauptideal $\langle q\rangle$ für ein festes Element $q \in R$ ist, schreibt man meistens (mod q) für (mod $\langle q\rangle$).

Um auch bestimmte Gleichungen, die die Kongruenzrelation $\equiv$ enthalten, lösen zu können, sind zusätzlich zu den Eigenschaften der Äquivalenzrelation noch einige Eigenschaften von $\equiv$ nützlich.
Für einen kommutativen Ring R und ein Ideal I in R gelten folgende Beziehungen. Seien $a, b, c, d \in R$. Falls $a \equiv b(\text{mod } I)$ und $c \equiv d(\text{mod } I)$, dann ist

$$\begin{aligned} a + c &\equiv b + d(\text{mod } I) && (6.5)\\ a - c &\equiv b - d(\text{mod } I) && (6.6)\\ ac &\equiv bd(\text{mod } I). && (6.7) \end{aligned}$$

Satz 6.13 (Reziproker Algorithmus)
Ist $\langle q\rangle$ ein Ideal im euklidschen Ring D und $a \in D$ teilerfremd zu q (d.h. $ggT(a, q) = 1$), dann existiert ein Element $a^{-1} \in D$, so daß gilt

$$aa^{-1} \equiv 1(\text{mod } q).$$

Beweis:
Da D ein euklidischer Ring ist, kann der erweiterte euklidische Algorithmus auf $a, q \in D$ angewandt werden und man erhält zwei Elemente $s, t \in D$ mit

$$sa + tq = 1.$$

Hier geht die Voraussetzung $\gcd(a, q) = 1$ ein.
Nun gilt $sa - 1 \in \langle q \rangle$ bzw. $sa \equiv 1 (\mathrm{mod}\ q)$. Damit ist $a^{-1} = s$ das gesuchte Inverse.
Um die Kongruenz der letzten Aussage des Satzes zu zeigen, nimmt man an, es gelte $aa^{-1} \equiv 1 (\mathrm{mod}\ q)$. Dann ist $aa^{-1}-1 \in \langle q \rangle$, also $\phi_q(aa^{-1}-1) = 0$, das auf $\phi_q(a)\phi_q(a^{-1}) = 1$ führt – d.h. $\phi_q(a^{-1})$ ist das multiplikativ Inverse von $\phi_q(a)$ in $D/\langle q \rangle$. Für die andere Richtung nimmt man an, $\phi_q(a)$ habe das multiplikativ inverse Element $b \in D/\langle q \rangle$. Dann gibt es ein Element $b \in D$ mit $\phi_q(b) = b$. Aus $\phi_q(a)\phi_q(b) = 1$ folgt $\phi_q(ab - 1) = 0$ bzw. $ab - 1 \in \langle q \rangle$ bzw. $ab \equiv 1 \,(\mathrm{mod}\ q)$. □

Nun ist es möglich, die gesuchte Eigenschaft für Kongruenzrelationen aufzustellen. Für einen euklidschen Ring D und das vom festen Element $q \in D$ erzeugte Ideal $\langle q \rangle$ gilt die Eigenschaft:

> Zu $a, b \in D$ mit a und q teilerfremd, existiert ein Element $a^{-1} \in D$, welches das Inverse (mod q) zu a ist und ein Element $x \in D$, so daß gilt (6.8)
>
> $$x \equiv a^{-1} b \,(\mathrm{mod}\ q)$$
>
> ist eine Lösung der Kongruenzgleichung
>
> $$ax \equiv b \,(\mathrm{mod}\ q).$$

6.3 Chinesischer-Restealgorithmus für ganze Zahlen

Nun geht es um die Entwicklung von Algorithmen zur Invertierung von Homomorphismen. Die Grundidee dieser 'Invertierungs'-Algorithmen ist, daß unter bestimmten Bedingungen ein Element a aus einem Ring R aus seinen Bildern $\phi_{I_i}(a)$, $i = 1, 2, \ldots$ rekonstruiert werden kann, falls die homomorphen Bilder R/I_i von R in einer 'bestimmten Anzahl' bekannt sind.

6.3.1 Das chinesische-Resteproblem

Man erinnere sich, daß für ein festes $m \in \mathbb{Z}$ der Modulo-Homomorphismus $\phi_m : \mathbb{Z} \longrightarrow \mathbb{Z}_m$, der den Ring $\mathbb{Z}$ auf den endlichen Ring $\mathbb{Z}_m$ (den Ring der 'Ganzen

Zahlen modulo m') abbildet, spezifiziert ist durch

$$\phi_m(a) = \mathrm{rem}(a,\, m), \quad \text{für alle } a \in \mathbb{Z}. \tag{6.9}$$

In der Kongruenz-Schreibweise für $a \in \mathbb{Z}$ und $\phi_m(a) = \bar{a} \in \mathbb{Z}_m$ ist dies

$$a \equiv \bar{a}(\bmod m).$$

Das klassische mathematische Problem, als *Chinesisches Resteproblem* bekannt, kann folgendermaßen formuliert werden:

> Finde zu gegebenen *Moduli* $m_0, m_1, \ldots, m_n \in \mathbb{Z}$ und den korrespondierenden *Resten* $u_i \in \mathbb{Z}_{m_i}$, $0 \le i \le n$ eine ganze Zahl $u \in \mathbb{Z}$, so daß gilt
>
> $$u \equiv u_i (\mathrm{mod}\; m_i),\ 0 \le i \le n.$$

Satz 6.14 (Chinesischer Restesatz)
Seien die Moduli $m_0, m_1, \ldots, m_n \in \mathbb{Z}$ paarweise teilerfremd – d.h. der

$$\gcd(m_i, m_j) = 1 \quad \text{für } i \neq j,$$

und $u_i \in \mathbb{Z}_{m_i}$, $i = 0, 1, \ldots, n$ $n+1$ spezifizierte Reste. Dann existiert für ein beliebiges $a \in \mathbb{Z}$ eine eindeutig bestimmte Zahl $u \in \mathbb{Z}$, die folgende Bedingungen erfüllt:

$$\begin{aligned} a &\le u < a + m, \quad \text{mit } m = \prod_{i=0}^{n} m_i, && (6.10)\\ u &\equiv u_i(\mathrm{mod}\; m_i),\ 0 \le i \le n. && (6.11) \end{aligned}$$

Beweis:

Eindeutigkeit:
Seien $u, v \in \mathbb{Z}$ zwei ganze Zahlen, die die Bedingungen (6.10) und (6.11) erfüllen. Nutzt man die Tatsache, daß $\equiv$ eine Äquivalenzrelation ist, erhält man aus (6.11)

$$\begin{aligned} u &\equiv v(\mathrm{mod}\; m_i), \quad \text{für } i = 0, 1, \ldots, n\\ &\Rightarrow u - v \in \langle m_i \rangle, \quad \text{für } i = 0, 1, \ldots, n\\ &\Rightarrow u - v \in \langle m \rangle \ \text{ mit } m = \prod_{i=0}^{n} m_i \end{aligned}$$

wobei im letzten Schritt ausgenutzt wurde, daß, weil die Moduli $m_0, m_1, \ldots, m_n$ paarweise teilerfremd sind, ein ganze Zahl, die ein Vielfaches von jedem m_i ist, auch ein Vielfaches des Produktes m sein muß. Aus (6.10) folgt

$$|u - v| < m$$

also $u - v = 0$, weil 0 das einzige Element aus dem Ideal $\langle m \rangle$ ist, das einen Betrag kleiner als m hat. Daher ist $u = v$.

Existenz:
u durchlaufe die m verschiedenen Werte in den Grenzen von (6.10). Man betrachte die korrespondierenden $(n+1)$-Tupel $(\phi_{m_0}(u), \phi_{m_1}(u), \ldots, \phi_{m_n}(u))$, wobei die ϕ_{m_i} die in (6.9) definierten Modulo-Homomorphismen sind. Nach der Eindeutigkeit können keine zwei der $(n+1)$-Tupel identisch sein, also haben die $(n+1)$-Tupel m verschiedene Werte. Da der endliche Ring $\mathbb{Z}_{m_i}$ genau m_i Elemente enthält, gibt es genau $m = \prod_{i=0}^{n} m_i$ verschiedene $(n+1)$-Tupel $(\nu_0, \nu_1, \ldots, \nu_n)$, so daß $\nu_i \in \mathbb{Z}_{m_i}$. Jedes mögliche $(n+1)$-Tupel kommt genau einmal vor, daher muß es einen Wert von u in dem gegebenen Bereich geben mit

$$(\phi_{m_0}(u), \phi_{m_1}(u), \ldots, \phi_{m_n}(u)) = (u_0, u_1, \ldots, u_n). \quad \square$$

Es ist wichtig zu beachten, in welchem Sinne die Lösung des chinesischen Resteproblems eindeutig ist. Zu gegebenen $n+1$ Resten $u_i \in \mathbb{Z}_{m_i}$ $(0 \leq i \leq n)$ und korrespondierenden $n+1$ Moduli (paarweise teilerfremd) hat das chinesische Resteproblem unendlich viele ganzzahlige Lösungen, aber mit der Eigenschaft (6.10) des Satzes 6.14 $(a = 0)$ ist die Lösung eindeutig, sofern man sich auf den Bereich von $0 \leq u < m$ beschränkt. Daher sagt man die Lösung ist *eindeutig modulo m*. Mit anderen Worten, zu gegebenen $u_i \in \mathbb{Z}_{m_i}$ $(0 \leq i \leq n)$ hat das System von Kongruenzen (6.11) keine eindeutige Lösung im Ring $\mathbb{Z}$, aber sehr wohl im Ring $\mathbb{Z}_m$ mit $m = \prod_{i=0}^{n} m_i$.

Unterschiedliche Wertauswahl für die ganze Zahl a im Satz 6.14 führt zu verschiedenen Darstellungen für den Ring $\mathbb{Z}_m$. Die Wahl $a = 0$ führt zu der gewohnten *positiven Darstellung* für $\mathbb{Z}_m$ als

$$\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$$

(wobei angenommen wird, daß m positiv ist). In der Praxis werden alle Moduli $m_0, m_1, \ldots, m_n$ und m ungerade positive ganze Zahlen sein.
Eine andere nützliche Darstellung ist die *symmetrische Darstellung* von $\mathbb{Z}_m$

$$\mathbb{Z}_m = \{-\frac{m-1}{2}, \ldots, -1, 0, 1, \ldots, \frac{m-1}{2}\}.$$

Die Wertauswahl für die ganze Zahl a im Satz 6.14, die auf die symmetrische Darstellung von $\mathbb{Z}_m$ führt ist

$$a = -\frac{m-1}{2}.$$

Der Beweis für Satz 6.14 ist nicht konstruktiv, da es sehr unpraktisch ist, zur Bestimmung der Lösung u einfach alle Elemente des Ringes $\mathbb{Z}_m$ zu testen, wenn m eine größere ganze Zahl ist. Im weiteren wird ein effizienter Algorithmus zur Lösung des chinesischen Resteproblems entwickelt.

6.3.2 Algorithmus von Garner

Der Algorithmus, der allgemein zur Lösung des chinesischen Resteproblems benutzt wird, ist nach H. L. Garner benannt, der eine Version dieses Algorithmus in den späten 50er Jahren entwickelt hat. Zu gegebenen positiven, paarweise teilerfremden Moduli $m_i \in \mathbb{Z}$ $(0 \le i \le n)$ und den korrespondierenden Resten $u_i \in \mathbb{Z}_{m_i}$ $(0 \le i \le n)$ soll das eindeutig bestimmte $u \in \mathbb{Z}_m$ $(m = \prod_{i=0}^{n} m_i)$, welches das System von Kongruenzen (6.11) erfüllt, berechnet werden. Der Schlüssel zu Garners Algorithmus ist, die Lösung $u \in \mathbb{Z}_m$ in der *gemischten Basisdarstellung (mixed radix)*

$$u = \nu_0 + \nu_1(m_0) + \nu_2(m_0 m_1) + \ldots + \nu_n(\prod_{i=0}^{n-1} m_i) \qquad (6.12)$$

auszudrücken, wobei $\nu_k \in \mathbb{Z}_{m_k}$ für $k = 0, 1, \ldots, n$.
Die gemischte Basisdarstellung (6.12) ist bedeutungslos in der obigen Allgemeinheit, weil Addition und Multiplikation in (6.12) für Unformungen im Ring $\mathbb{Z}_m$ auftreten, aber jeder *gemischte Basiskoeffizient* ν_k in einem anderen Ring $\mathbb{Z}_{m_k}$ liegt. Um (6.12) die nötige Aussagekraft zu geben, fordert man, daß alle Ringe $\mathbb{Z}_{m_k}$ $(0 \le k \le n)$ und $\mathbb{Z}_m$ in einer der beiden folgenden *konsistenten Darstellungen* repräsentiert wird:

(i) Jeder Ring $\mathbb{Z}_{m_k}$ $(0 \le k \le n)$ und $\mathbb{Z}_m$ ist in seiner positiven Darstellung repräsentiert, oder

(ii) Jeder Ring $\mathbb{Z}_{m_k}$ $(0 \le k \le n)$ und $\mathbb{Z}_m$ ist in seiner symmetrischen Darstellung repräsentiert (wobei angenommen wird, daß jedes m_k ungerade ist).

Die natürliche Identifikation von Elementen aus einem Ring $\mathbb{Z}_{m_k}$ mit Elementen des größeren Ringes $\mathbb{Z}_m$ ergibt nun die gewünschte Interpretation von (6.12). Es kann gezeigt werden, daß jedes $u \in \mathbb{Z}_m$ in der Form (6.12) darstellbar ist und falls eine der konsistenten Darstellungen (i) oder (ii) benutzt wird, die Koeffizienten ν_k $(0 \le k \le n)$ eindeutig bestimmt sind. Es sollte beachtet werden, daß bei positiver konsistenter Darstellung (i) (6.12) eine Verallgemeinerung der bekannten Tatsache ist, daß jede ganze Zahl u in den Grenzen von $0 \le u < \beta^{n+1}$ eindeutig in der Darstellung

$$u = \nu_0 + \nu_1\beta + \nu_2\beta^2 + \ldots + \nu_n\beta^n$$

zur Basis β repräsentiert werden kann, wobei $0 \le \nu_k < \beta$ (d.h. $\nu_k \in \mathbb{Z}_\beta$) und $\beta > 1$ eine positive ganze Zahl (d.h. $u \in \mathbb{Z}_{\beta^{n+1}}$) ist.

Beispiel:
Sei $m_0 = 3$, $m_1 = 5$ und $m = m_0 m_1 = 15$. In der positiven konsistenten Darstellung hat $u = 11 \in \mathbb{Z}_{15}$ die eindeutige gemischte Basisdarstellung

$$11 = \nu_0 + \nu_1(3)$$

mit $\nu_0 = 2 \in \mathbb{Z}_3$ und $\nu_1 = 3 \in \mathbb{Z}_5$. In der symmetrischen konsistenten Darstellung hat $\tilde{u} = -4 \in \mathbb{Z}_{15}$ die eindeutige gemischte Basisdarstellung

$$-4 = \tilde{\nu_0} + \tilde{\nu_1}(3)$$

mit $\tilde{\nu_0} = -1 \in \mathbb{Z}_3$ und $\tilde{\nu_1} = -1 \in \mathbb{Z}_5$. Man beachte, $u = 11$ und $\tilde{u} = -4$ sind einfach zwei verschiedene Darstellungen für das gleiche Element in $\mathbb{Z}_{15}$, aber die korrespondierenden Koeffizienten ν_1 und $\tilde{\nu_1}$ sind *nicht* einfach zwei verschiedene Darstellungen für das gleiche Element in $\mathbb{Z}_5$.

Schreibt man die Lösung u des Systems von Kongruenzen (6.11) in der gemischten Basisdarstellung (6.12), so ist es leicht, Formeln für die ν_k $(0 \leq k \leq n)$ in (6.12) zu bestimmen. Offensichtlich folgt aus (6.12), daß

$$u \equiv \nu_0 (\mathrm{mod}\ m_0)$$

und damit, daß der Fall $i = 0$ des Systems von Kongruenzen (6.11) erfüllt ist, falls ν_0 so gewählt wird, daß gilt:

$$\nu_0 \equiv u_0 (\mathrm{mod}\ m_0). \tag{6.13}$$

Allgemein gilt für $k \geq 1$, sind die Koeffizienten $\nu_0, \nu_1, \ldots, \nu_{k-1}$ bestimmt, dann kann mit (siehe (6.12))

$$u \equiv \nu_0 + \nu_1(m_0) + \ldots + \nu_k(\prod_{i=0}^{k-1} m_i)(\mathrm{mod}\ m_k)$$

der Fall $i = k$ des Systems von Kongruenzen (6.11) erfüllt werden, indem ν_k so gewählt wird, daß gilt:

$$\nu_0 + \nu_1(m_0) + \ldots + \nu_k(\prod_{i=0}^{k-1} m_i) \equiv u_k (\mathrm{mod}\ m_k).$$

Wendet man die Eigenschaften (6.5) - (6.8) an, um die Kongruenzgleichung für die ν_k zu lösen, erhält man für $k \geq 1$

$$\nu_k \equiv \left(u_k - [\nu_0 + \nu_1(m_0) + \ldots + \nu_{k-1}(\prod_{i=0}^{k-2} m_i)]\right)\left(\prod_{i=0}^{k-1} m_i\right)^{-1} \quad (\mathrm{mod}\ m_k), \tag{6.14}$$

wobei die auftretende Inverse gültig ist, weil $\prod_{i=0}^{k-1} m_i$ teilerfremd zu m_k ist. Es sei noch bemerkt, daß es, nach der Wahl einer konsistenten Darstellung, eine *eindeutige* ganze Zahl $\nu_0 \in \mathbb{Z}_{m_0}$ gibt, die (6.13) $(\nu_0 = u_0 \in \mathbb{Z}_{m_0})$ erfüllt und ähnlich gibt es für $k = 1, 2, \ldots, n$ eine *eindeutige* ganze Zahl $\nu_k \in \mathbb{Z}_{m_k}$, die (6.14) erfüllt.

Algorithmus 1 Chinesischer Restealgorithmus von Garner

begin

comment Berechne zu gegebenen paarweise teilerfremden positiven Moduli $m_i \in \mathbb{Z}$ $(0 \leq i \leq n)$ und den korrespondierenden Resten $u_i \in \mathbb{Z}_{m_i}$ $(0 \leq i \leq n)$ die eindeutige ganze Zahl $u \in \mathbb{Z}_m$ mit $m = \prod_{i=0}^{n} m_i$, so daß

$$u \equiv u_i (\bmod\ m_i),\ i = 0, 1, \ldots, n.$$

Konsistente Darstellung für die $\mathbb{Z}_{m_i}$ $(0 \leq i \leq n)$ und $\mathbb{Z}_m$ wird vorausgesetzt.

comment Schritt 1: Bestimme die Inversen unter der Benutzung einer Prozedur reciprocal(a, q), die $a^{-1} (\bmod\ q)$ berechnet. Beachte, daß dieser Schritt gesondert vorberechnet werden sollte, falls der Algorithmus wiederholt mit der gleichen Menge $\{m_i\}$ von Moduli benutzt wird.

for $k \leftarrow 1$ **until** n **do**
  begin
    $product \leftarrow \phi_{m_k}(m_0)$;
    for $i \leftarrow 1$ **until** $k - 1$ **do**
      $product \leftarrow \phi_{m_k}(product \times m_i)$;
    $\gamma_k \leftarrow$ reciprocal$(product, m_k)$
  end;

comment Schritt 2: Berechne die gemischten Basiskoeffizienten $\{v_k\}$.

$v_0 \leftarrow u_0$;
for $k \leftarrow 1$ **until** n **do**
  begin
    $temp \leftarrow \phi_{m_k}(v_{k-1})$;
    for $j \leftarrow k - 2$ **step** -1 **until** 0 **do**
      $temp \leftarrow \phi_{m_k}(temp \times m_j + v_j)$;
    $v_k \leftarrow \phi_{m_k}((u_k - temp) \times \gamma_k)$
  end;

comment Schritt 3: Konvertiere von gemischter Basisdarstellung zur Standartdarstellung.

$u \leftarrow v_n$;
for $k \leftarrow n - 1$ **step** -1 **until** 0 **do**
      $u \leftarrow u \times m_k + v_k$
end.

6.3.3 Details zur Implementation des Algorithmus von Garner

Der Algorithmus von Garner ist formal im Algorithmus 1 beschrieben. Einige Details der Implementation bedürfen weiterer Erläuterungen. Es ist wichtig zu beachten, daß in den gebräuchlichen Anwendungen dieses Algorithmus die Moduli $m_i (0 \leq i \leq n)$ single-precision Zahlen sind (meist large single-precision) und daher die Reste u_i $(0 \leq i \leq n)$ ebenfalls. Die berechnete ganze Zahl u ist eine multiprecision Zahl und auch die Liste der Reste $(u_0, u_1, \ldots, u_n)$ kann einfach als eine andere Darstellung der multiprecision Zahl u angesehen werden. Der Algorithmus 1 ist so angelegt, daß typische Operationen auf multiprecision Zahlen, bis auf den letzten Schritt, vollständig vermieden sind. Insbesondere wird die Schreibweise ϕ_{m_k} benutzt, gewissermaßen um Konsistenz mit der mathematischen Bedeutung als ein Modulo-Homomorphismus zu erhalten, doch hier noch eine genauere algorithmische Spezifikation:

$\phi_{m_k}(Ausdruck)$ bedeutet 'setze *Ausdruck* im Ring $\mathbb{Z}_{m_k}$ ein'.

Genauer, es bedeutet, falls *Ausdruck* in eine Folge binärer Operationen zerlegt wird, das Zwischenergebnis *jeder* binären Operation zuerst modulo m_k reduziert und dann erst mit der Auswertung von *Ausdruck* fortgefahren wird. Auf diese Weise kann garantiert werden, daß jede im Algorithmus 1 vorkommende Variable (außer u) eine single-precision Variable und sogar, daß jede Operation in Schritt 1 und 2 eine Operation auf single-precision Zahlen ist. (Man beachte, sind a und b single-precision Zahlen, dann wird z.B. die Operation $\phi_{m_k}(a \times b)$ zunächst meistens durch eine gewöhnliche Ganzzahl-Multiplikation $a \times b$, die eine double-precission Zahl c ergibt, gefolgt von einer Ganzzahl-Division zur Bestimmung von $\mathrm{rem}(c, m_k)$, berechnet).
Die Zahlen ν_k, $k = 1, 2, \ldots, n$, die (6.14) erfüllen, werden in Schritt 2 durch Auswerten der rechten Seite von (6.14) im Ring $\mathbb{Z}_{m_k}$ berechnet. Die in (6.14) vorkommenden Inversen

$$\gamma_k = (\prod_{i=0}^{k-1} m_i)^{-1} (\bmod\ m_k),\ k = 1, 2, \ldots, n$$

werden alle in Schritt 1 berechnet. Man beachte, daß ein Verfahren zur Implementation der Prozedur

$$\mathrm{reciprocal}(a, q)$$

um $a^{-1}(\bmod\ q)$ für teilerfremde a und q zu berechnen, im Beweis des Satzes 6.13 angegeben ist, nämlich durch Anwenden des erweiterten euklidischen Algorithmus auf $a, q \in \mathbb{Z}$, der Zahlen s und t liefert, so daß

$$sa + tq = 1$$

gilt, woraus man $\phi_q(s) = rem(s, q)$ als das gewünschte Inverse im Ring $\mathbb{Z}_q$ erhält. Die Berechnung der Inversen $\{\gamma_k\}$ ist absichtlich separat vom Rest der Berechnungen im Algorithmus, weil die $\{\gamma_k\}$ nur von den Moduli $\{m_i\}$ abhängen. Typische Anwendungen des Algorithmus von Garner in einem System benutzen die modulo Darstellung der multiprecision Zahlen, die Moduli $\{m_i\}$ sind fest gegeben, so daß Schritt 1 aus dem Algorithmus heraus genommen werden kann und die Inversen $\{\gamma_k\}$ als vorberechnete Konstanten in den Algorithmus gegeben werden. Es ist auch beachtenswert, daß es Situationen gibt, in denen Schritt 1 und 3 aus dem Algorithmus heraus gelöst werden. Z.B. wenn man zwei multiprecision Zahlen a und b, die in ihrer modulo Darstellung repräsentiert sind vergleichen will, reicht es aus, ihre (single-precision) gemischten Basiskoeffizienten zu berechnen und diese zu vergleichen.

Schließlich benötigt Schritt 3 noch eine Rechtfertigung. Wird eine konsistente Darstellung für $\mathbb{Z}_{m_k}$ $(0 \leq k \leq n)$ und $\mathbb{Z}_m$ verwendet, dann ist die gemischte Basisdarstellung (6.12) für $u \in \mathbb{Z}_m$ eindeutig. Aber es ist noch nicht gezeigt, daß die Operationen in (6.12) im Ring $\mathbb{Z}$ anstatt im Ring $\mathbb{Z}_m$ auch das gewünschte eindeutige $u \in \mathbb{Z}_m$ liefern – also daß es in Schritt 3 nicht notwendig ist die **for**-Schleifen-Anweisung zu schreiben als

$$u \leftarrow \phi_m(u \times m_k + v_k).$$

Um dies zu prüfen, beachte man, daß aus (6.12) folgt, sind $|\nu_k| \leq \frac{m_k - 1}{2}$ für $k = 0, 1, \ldots, n$ (hier symmetrische konsistente Darstellung), dann zeigt

$$\begin{aligned} |u| &\leq \frac{m_0 - 1}{2} + \frac{m_1 - 1}{2}(m_0) + \ldots + \frac{m_n - 1}{2}(\prod_{i=0}^{n-1} m_i) \\ |u| &\leq \frac{1}{2}[(\prod_{i=0}^{n} m_i) - 1], \end{aligned}$$

daß u im richtigen Bereich liegt. Ähnlich, falls $0 \leq \nu_k \leq m_k - 1$ für $k = 0, 1, \ldots, n$ (d.h. falls die positive konsistente Darstellung benutzt wird), dann ist klar $u \geq 0$ und auf gleiche Weise wie oben zeigt

$$u \leq (\prod_{i=0}^{n} m_i) - 1$$

daß u im richtigen Bereich liegt. Schließlich leistet Schritt 3 die Auswertung von (6.12) unter Benutzung der Methode der geschachtelten Multiplikation:

$$u = \nu_0 + m_0(\nu_1 + m_1(\nu_2 + \ldots + m_{n-2}(\nu_{n-1} + m_{n-1}(\nu_n))\ldots)).$$

Beispiel:

Angenommen, die single-precision Zahlen auf einem speziellen Computer wären beschränkt auf den Bereich $-100 < a < 100$ (d.h. two-digit integer). Man betrachte die

Moduli der drei größten single-precision Zahlen, die ungerade und paarweise teilerfremd sind:

$$m_0 = 99;\ m_1 = 97;\ m_2 = 95.$$

Dann ist $m = m_0 m_1 m_2 = 912285$. Bei Gebrauch der symmetrischen konsistenten Darstellung ist der Bereich der ganzen Zahlen in $\mathbb{Z}_{912285}$

$$-456142 \leq u \leq 456142.$$

Nun betrachte man das Problem, u zu bestimmen mit

$$\begin{aligned} u &\equiv 49(\bmod 99) \\ u &\equiv -21(\bmod 97) \\ u &\equiv -30(\bmod 95). \end{aligned}$$

Mit dem Algorithmus 1 ergeben sich in Schritt 1 folgende Inverse:

$$\begin{aligned} \gamma_1 &= m_0^{-1}(\bmod m_1) = 99^{-1}(\bmod 97) = 2^{-1}(\bmod 97) = -48 \\ \gamma_2 &= (m_0 m_1)^{-1}(\bmod m_2) = 8^{-1}(\bmod 95) = 12. \end{aligned}$$

Als Ergebnis von Schritt 2 erhält man folgende gemischte Basiskoeffizienten für u:

$$\nu_0 = 49;\ \nu_1 = -35;\ \nu_2 = -28.$$

An dieser Stelle hat u die gemischte Basisdarstellung

$$u = 49 - 35(99) - 28(99)(97).$$

Schließlich liefert die Konversion in Schritt 3 unter Benutzung der 'multiprecision' Arithmetik

$$u = -272300.$$

6.4 Polynom-Interpolationsalgorithmus

Nun betrachtet man den entsprechenden Umkehrprozeß zu den Auswertungshomomorphismen. Man erinnere sich daran, daß man in erster Linie an Homomorpismen $\phi_{\langle I,p\rangle}$ interessiert ist, die den multivariaten Polynomring $\mathbb{Z}[x_1, \ldots, x_\nu]$ auf den euklidischen Ring $\mathbb{Z}_p[x_1]$ projizieren (oder vielleicht auf den Körper $\mathbb{Z}_p$). In der Schreibweise $\phi_{\langle I,p\rangle}$ bezeichnet p eine Primzahl, I den Kern eines multivariaten Auswertungshomomorphismus und $\phi_{\langle I,p\rangle}$ den zusammengesetzten Homomorpismus $\phi_I \phi_p$ mit den Definitionsbereichen

$$\phi_p : \mathbb{Z}[x_1, \ldots, x_\nu] \longrightarrow \mathbb{Z}_p[x_1, \ldots, x_\nu] \tag{6.15}$$

und

$$\phi_I : \mathbb{Z}_p[x_1, \ldots, x_\nu] \longrightarrow \mathbb{Z}_p[x_1] \tag{6.16}$$

(das homomorphe Bild in (6.16) könnte auch $\mathbb{Z}_p$ sein). Der Umkehrprozeß zu Homomorpismen der Form (6.15) ist der chinesische Restealgorithmus aus dem vorhergehenden Abschnitt. (Man beachte, daß der chinesische Restealgorithmus von Garner im polynomialen Fall Koeffizient für Koeffizient angewandt werden kann, wenn die Polynome in erweiterter kanonischer Form ausgedrückt sind). Der Umkehrprozeß zu Homomorpismen der Form (6.16) ist das Problem der Polynominterpolation.

6.4.1 Das Problem der Polynominterpolation

Die Umkehrung multivariater Auswertungshomomorphismen der Form (6.16) kann nacheinander in jeder Unbestimmten durchgeführt werden, indem man in natürlicher Weise ϕ_I als eine Komposition univariater Auswertungshomomorphismen ansieht. Deshalb genügt es, die Umkehrung univariater Auswertungshomomorphismen der Form

$$\phi_{x-\alpha_i} : D[x] \longrightarrow D$$

zu betrachten, wobei D im allgemeinen ein multivariater Polynomring über dem Körper $\mathbb{Z}_p$ und $\alpha_i \in \mathbb{Z}_p$ ist. Es ist für die Berechnung wichtig, daß α_i im Körper $\mathbb{Z}_p$ liegt.

Die Entwicklung eines Algorithmus zur Polynominterpolation verläuft genau parallel zur Entwicklung zu Garner's Algorithmus für das ganzzahlige chinesische Resteproblems. Betrachtet man dies aus einem genügend abstrakten Standpunkt (Ringtheorie), so wird klar, daß die beiden Prozesse identisch sind. Insbesondere erhält man durch Umschreiben des ganzzahligen chinesischen Resteproblems die folgende Beschreibung des Polynominterpolationsproblems:

> Sei $D[x]$ ein Polynomring (in x und möglicherweise in noch weiteren Unbestimmten) über dem Koeffizientenkörper $\mathbb{Z}_p$. Zu gegebenen *Moduli* $x-\alpha_0, x-\alpha_1, \ldots, x-\alpha_n$ mit $\alpha_i \in \mathbb{Z}_p$, $0 \le i \le n$ und den entsprechenden *Resten* $u_i \in D$, $0 \le i \le n$ finde man ein Polynom $u(x) \in D[x]$, so daß gilt:

$$u(x) \equiv u_i (\text{mod } x - \alpha_i),\ 0 \le i \le n. \tag{6.17}$$

Man beachte, daß in diesem Fall die Kongruenzen (6.17) gewöhnlich in der dazu äquivalenten Form

$$u(\alpha_i) = u_i,\ 0 \le i \le n \tag{6.18}$$

dargestellt werden. Die Elemente $\alpha_i \in \mathbb{Z}_p$ $(0 \le i \le n)$ heißen meist *Auswertungspunkte* oder *Interpolationspunkte*. Um, wie im Falle des ganzzahligen chinesischen Resteproblems, die Existenz einer Lösung garantieren zu können, muß noch die zusätzliche

Bedingung, daß die Moduli $\{x - \alpha_i\}$ paarweise teilerfremd sind, gefordert werden. Doch es gilt klar,

$$\gcd(x - \alpha_i, x - \alpha_j) = 1, \quad \text{genau dann, wenn } \alpha_i \neq \alpha_j,$$

so daß sich die zusätzliche Bedingung auf die ziemlich offensichtliche Bedingung reduziert, daß die Moduli $\{x - \alpha_i\}$ *verschieden* sein müssen (d.h. die Auswertungspunkte $\{\alpha_i\}$ müssen verschieden sein). Genau wie beim ganzzahligen chinesischen Resteproblem, ist die Lösung des Polynominterpolationsproblems nur eindeutig modulo $\prod_{i=0}^{n}(x - \alpha_i)$, was bedeutet, daß die Lösung eindeutig ist, falls sie auf einen Grad kleiner $n + 1$ beschränkt wird.

Der folgende Satz zeigt die obigen Existenz und Eindeutigkeitsergebnisse in einem allgemeineren Zusammenhang, in dem D ein beliebiger Integritätsbereich ist und die Auswertungspunkte $\{\alpha_i\}$ beliebige verschiedene Punkte aus D sind. Daher erlaubt dieser Satz, daß die Lösung $u(x)$ in $F_D[x]$ liegt, statt in $D[x]$, wobei F_D den Quotientenkörper über dem Integritätsbereich D bezeichnet. Danach wird ein effizienter Algorithmus zur Lösung des Polynominterpolationsproblems entwickelt. Es ist offensichtlich, daß mit den obigen speziellen Voraussetzungen die Lösung $u(x)$ in $D[x]$ liegt, weil die einzigen Divisionen, die vorkommenden, Divisionen im Koeffizientenkörper $\mathbb{Z}_p$ sind.

Definition 6.15
Seien

$$\begin{aligned} u_0 &= a_0 + a_1\alpha_0 + \ldots + a_n\alpha_0^n \\ u_1 &= a_0 + a_1\alpha_1 + \ldots + a_n\alpha_1^n \\ &\vdots \\ u_n &= a_0 + a_1\alpha_n + \ldots + a_n\alpha_n^n, \end{aligned}$$

dann heißt die Matrix

$$\mathbf{V} = \begin{pmatrix} 1 & \alpha_0 & \alpha_0^2 & \ldots & \alpha_0^n \\ 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^n \\ \vdots & & & & \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^n \end{pmatrix}$$

Vandermonde Matrix. *Man hat also* $V\mathbf{a} = \mathbf{u}$.

Satz 6.16
Sei D ein beliebiger Integritätsbereich, die $\alpha_i \in D$, $i = 0, 1, \ldots, n$ $\quad n + 1$ verschiedene Elemente in D sowie die $u_i \in D$ $i = 0, 1, \ldots, n$ $\quad n + 1$ beliebige Werte in D. Dann existiert ein eindeutiges Polynom $u(x) \in F_D[x]$ (F_D der Quotientenkörper von D) mit

(i) $\deg[u(x)] \leq n$

(ii) $u(\alpha_i) = u_i,\ 0 \leq i \leq n.$

Beweis:
Mit Bedingung (i) kann man $u(x)$ in der Form

$$u(x) = a_0 + a_1 x + \ldots + a_n x^n$$

schreiben, wobei die Koeffizienten $a_i \in F_D$ $(0 \leq i \leq n)$ zu bestimmen sind. Bedingung (ii) wird dann zu dem linearen System

$$V\mathbf{a} = \mathbf{u}$$

der Ordnung $(n+1)$, wobei V die *Vandermonde Matrix* ist mit dem (i,j)ten-Eintrag α_i^j $(i,j = 0,1,\ldots,n)$. $\mathbf{u}$ ist der Vektor mit dem $i$ten-Eintrag u_i $(i = 0,1,\ldots,n)$ und $\mathbf{a}$ der Vektor aus Unbekannten mit dem i-ten-Eintrag a_i $(i = 0,1,\ldots,n)$. Aus Sätzen der linearen Algebra folgt, daß dieses lineare System im Körper F_D gelöst werden kann und die Lösung eindeutig ist, falls $\det(V) \neq 0$. Durch Anwenden der klassischen Formel für die Vandermonde Determinante

$$\det(V) = \prod_{0 \leq i < k \leq n} (\alpha_k - \alpha_i)$$

sieht man, daß $\det(V) \neq 0$, weil die Elemente $\alpha_0, \alpha_1, \ldots, \alpha_n \in D$ verschieden sind. □

6.4.2 Der Newtonsche Interpolationsalgorithmus

Der Beweis von Satz 6.16 ist konstruktiv, da man lineare Gleichungen über dem Quotientenkörper eines Integritätsbereiches lösen kann. Doch kann die Lösung des Interpolationsproblems mit einem Algorithmus berechnet werden, der viel weniger Aufwand erfordert wie das Lösen eines linearen Gleichungssystemes. Der Algorithmus, der hier zur Polynominterpolation entwickelt wird, geht zurück auf Newton (17. Jahrhundert). Wie beim Algorithmus von Garner, liegt der Schlüssel zur Entwicklung darin, die Lösung $u(x) \in F_D[x]$ in der folgenden gemischten Basisdarstellung (die auch manchmal *Newton Form* oder *dividierte Differenzen Form* heißt) auszudrücken

$$u(x) = \nu_0 + \nu_1(x-\alpha_0) + \nu_2(x-\alpha_0)(x-\alpha_1) + \ldots + \nu_n \prod_{i=0}^{n-1}(x-\alpha_i), \tag{6.19}$$

wobei die *Newton Koeffizienten* $\nu_k \in F_D$ $(0 \leq k \leq n)$ zu bestimmen sind. Die Rechtfertigung für diese gemischte Basisdarstellung ist die Tatsache aus der linearen Algebra, daß jede Menge von Polynomen $m_k(x) \in F_D[x]$, $k = 0, 1, \ldots, n$ mit $\deg[m_k(x)] = k$ eine *Basis* für Polynome in x vom Grad n über dem Körper F_D bilden; in diesem Fall wählt man $m_0(x) = 1$, $m_k(x) = \mathbb{D} \prod_{i=0}^{k-1}(x - \alpha_i)$ für $k = 0, 1, \ldots, n$.

Der Newtonsche Interpolationsalgorithmus kann für die allgemeinen Voraussetzungen des Satzes 6.16 entwickelt werden. In diesem Fall sind die Newton Koeffizienten $\{\nu_k\}$ in (6.19) Quotienten von Elementen aus D (die sogenannten *dividierten Differenzen*). Jedoch wird der Algorithmus hier, unserem Interesse entsprechend, mit den Voraussetzungen, die in der Einleitung zu (6.17) gegeben sind, entwickelt. Dabei tritt kein Quotient von Elementen (Polynomen) aus D auf, da die einzigen vorkommenden Divisionen, Divisionen (d.h. Multiplikationen mit Inversen) im Koeffizientenkörper $\mathbb{Z}_p$ des Polynomringes D sind.

Schreibt man die Lösung $u(x)$ in der Newton Form (6.19) und verwendet die Bedingnung (6.18), erhält man Formeln für die Newton Koeffizienten $\{\nu_k\}$ $(0 \leq k \leq n)$. Mit (6.19) ist es offensichtlich, daß

$$u(\alpha_0) = \nu_0$$

gilt und damit der Fall $i = 0$ der Bedingnung (6.18) erfüllt ist, wenn ν_0 gewählt wird als

$$\nu_0 = u_0. \tag{6.20}$$

Allgemein gilt für $k \geq 1$, sind die Newton Koeffizienten $\nu_0, \nu_1, \ldots, \nu_{k-1}$ bestimmt, gilt mit (6.19)

$$u(\alpha_k) = \nu_0 + \nu_1(\alpha_k - \alpha_0) + \ldots + \nu_k \prod_{i=0}^{k-1}(\alpha_k - \alpha_i).$$

Also ist der Fall $i = k$ der Bedingnung (6.18) erfüllt, falls ν_k so gewählt wird, daß

$$\nu_0 + \nu_1(\alpha_k - \alpha_0) + \ldots + \nu_k \prod_{i=0}^{k-1}(\alpha_k - \alpha_i) = u_k.$$

Da die $\alpha_i \in \mathbb{Z}_p$ $(0 \leq k \leq n)$ kann man im Körper $\mathbb{Z}_p$ die Inverse zu dem von Null verschiedenen Element $\prod_{i=0}^{k-1}(\alpha_k - \alpha_i) \in \mathbb{Z}_p$ berechnen, indem man wieder einmal den erweiterten euklidischen Algorithmus anwendet (das geht, da jede von Null verschiedene ganze Zahl in $\mathbb{Z}_p$ teilerfremd (in $\mathbb{Z}$) zur Primzahl p ist). Löst man die ν_k, erhält man für $k \geq 1$:

$$\begin{aligned} \nu_k \;\; = \;\; & \left(u_k - [\nu_0 + \nu_1(\alpha_k - \alpha_0) + \ldots + \nu_{k-1} \prod_{i=0}^{k-2}(\alpha_k - \alpha_i)]\right) \\ & \cdot \left(\prod_{i=0}^{k-1}(\alpha_k - \alpha_i)\right)^{-1}. \end{aligned} \tag{6.21}$$

Es ist wichtig zu beachten, daß die u_k $(0 \leq k \leq n)$ und ν_k $(0 \leq k \leq n)$ i.a. multivariate Polynome aus D mit Koeffizienten aus $\mathbb{Z}_p$ sind und die gesamte Arithmetik in den Koeffizienten, die in der Gleichung (6.21) auftritt im Körper $\mathbb{Z}_p$ ausgeführt wird.

Algorithmus 2 Newtonscher Interpolationsalgorithmus

begin

> **comment** Sei $D = \mathbb{Z}_p[y]$ ein Polynomring in $\nu \geq 0$ Unbekannten $y = (y_1, \ldots, y_\nu)$ über dem endlichen Körper $\mathbb{Z}_p$ ($D = \mathbb{Z}_p$ im Fall $\nu = 0$). Berechne zu gegebenen verschiedenen Auswertungspunkten $\alpha_i \in \mathbb{Z}_p$ $(0 \leq i \leq n)$ und den entsprechenden Werten $u_i \in D$ $(0 \leq i \leq n)$ das eindeutige Polynom $u(x) \in D[x]$, so daß $\deg[u(x)] \leq n$ und $u(\alpha_i) = u_i$, $i = 0, 1, \ldots, n$ gilt.

> **comment** Schritt 1: Bestimme die erforderlichen Inversen unter Benutzung einer Prozedur reciprocal(a, q), die $a^{-1}(\mathrm{mod}\ q)$ berechnet.

for $k \leftarrow 1$ **until** n **do**
  begin
    $product \leftarrow \phi_p(\alpha_k - \alpha_0)$;
    for $i \leftarrow 1$ **until** $k - 1$ **do**
      $product \leftarrow \phi_p(product \times (\alpha_k - \alpha_i))$;
    $\gamma_k \leftarrow$ reciprocal$(product, p)$
  end;

> **comment** Schritt 2: Berechne die Newton Koeffizienten $\{v_k\}$.

$v_0 \leftarrow u_0$;
for $k \leftarrow 1$ **until** n **do**
  begin
    $temp \leftarrow v_{k-1}$;
    for $j \leftarrow k - 2$ **step** -1 **until** 0 **do**
      $temp \leftarrow \phi_p(temp \times (\alpha_k - \alpha_j) + v_j)$;
    $v_k \leftarrow \phi_p((u_k - temp) \times \gamma_k)$
  end;

> **comment** Schritt 3: Konvertiere von der Newton Form zur Standarddarstellung.

$u \leftarrow v_n$;
for $k \leftarrow n-1$ **step** -1 **until** 0 **do**
$\quad u \leftarrow \phi_p(u \times (x - \alpha_k) + v_k)$
end.

Beispiel:
Man berechne das Polynom $u(x,y) \in \mathbb{Z}_{97}[x,y]$ mit maximalem Grad 2 in x und maximalem Grad 1 in y, das durch die folgenden Werte im Körper $\mathbb{Z}_{97}$ bestimmt ist

$$\begin{aligned} u(0,0) &= -21 & \qquad u(0,1) &= -30 \\ u(1,0) &= 20 & u(1,1) &= 17 \\ u(2,0) &= -36 & u(2,1) &= -31. \end{aligned}$$

Zuerst rekonstruiert man das Bild von $u(x,y)$ in $\mathbb{Z}_{97}[x,y]/\langle x-0\rangle$ (d.h. für den Fall $x=0$). In der Schreibweise vom Algorithmus 2 heißt das $D = \mathbb{Z}_{97}$, $\alpha_0 = 0$, $\alpha_1 = 1$, $u_0 = -21$, $u_1 = -30$ und man berechnet ein Polynom $u(0,y) \in \mathbb{Z}_{97}[y]$ (d.h. die Unbestimmte x im Algorithmus 2 ist nun y). Schritt 1 ist in diesem Fall trivial:

$$\gamma_1 = (\alpha_1 - \alpha_0)^{-1} (\mathrm{mod}\ 97) = 1^{-1} (\mathrm{mod}\ 97) = 1.$$

Schritt 2 berechnet die folgenden Newton Koeffizienten für $u(0,y)$:

$$\nu_0 = -21 \qquad \nu_1 = -9$$

und man findet daher in Schritt 3

$$u(0,y) = -21 - 9(y-0) = -9y - 21.$$

Ähnlich findet man durch Rekonstruieren der Bilder von $u(x,y)$ in $\mathbb{Z}_{97}[x,y]/\langle x-1\rangle$ und $\mathbb{Z}_{97}[x,y]/\langle x-2\rangle$

$$\begin{aligned} u(1,y) &= -3y + 20 \\ u(2,y) &= 5y - 36. \end{aligned}$$

Für den multivariaten Schritt wendet man Algorithmus 2 mit $D = \mathbb{Z}_{97}[y]$, $\alpha_0 = 0$, $\alpha_1 = 1$, $\alpha_2 = 2, u_0 = u(0,y)$, $u_1 = u(1,y)$, $u_2 = u(2,y)$ an und berechnet das Polynom $u(x,y) \in \mathbb{Z}_{97}[x][y]$. In Schritt 1 werden in diesem Fall die folgenden Inversen berechnet:

$$\begin{aligned} \gamma_1 &= (\alpha_1 - \alpha_0)^{-1} (\mathrm{mod}\ 97) = 1^{-1} (\mathrm{mod}\ 97) = 1 \\ \gamma_2 &= [(\alpha_2 - \alpha_0)(\alpha_2 - \alpha_1)]^{-1} (\mathrm{mod}\ 97) = 2^{-1} (\mathrm{mod}\ 97) = -48. \end{aligned}$$

Schritt 2 berechnet die folgenden Newton Koeffizienten:

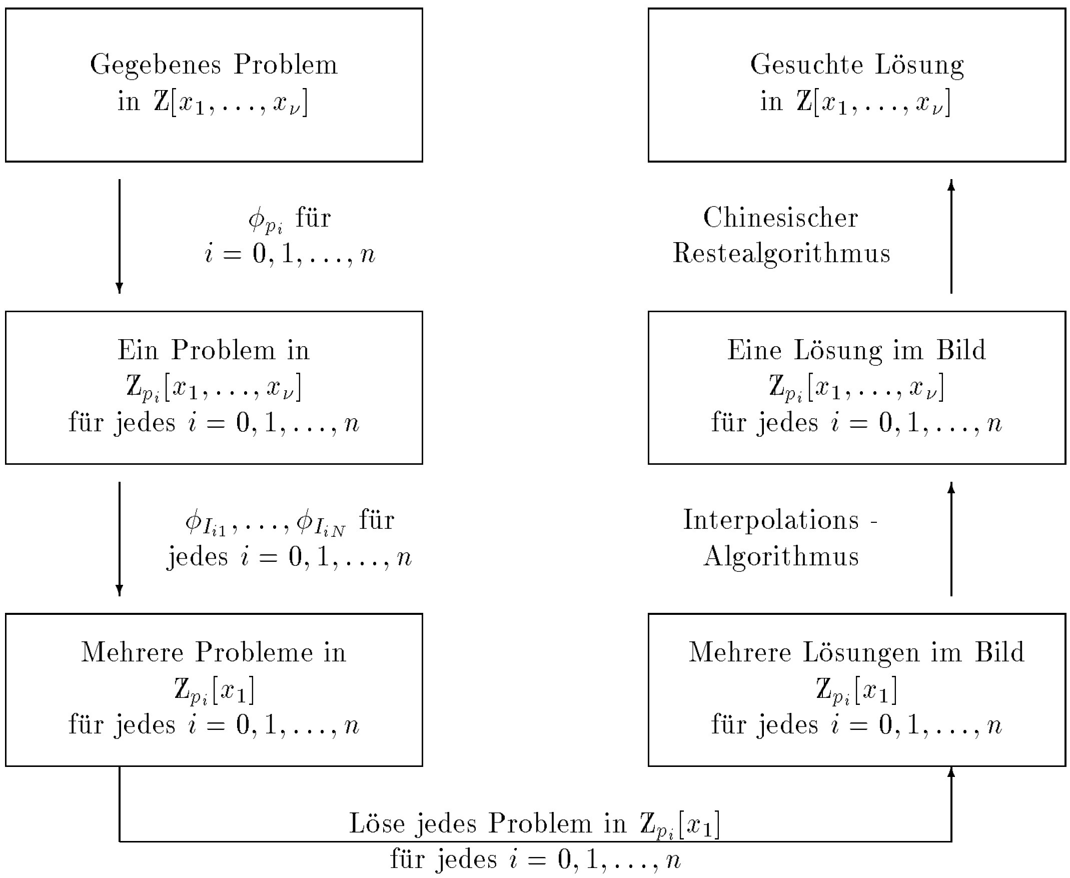
$$\begin{aligned} \nu_0 &= -9y - 21 \\ \nu_1 &= 6y + 41 \\ \nu_2 &= y. \end{aligned}$$

Schließlich findet man in Schritt 3

$$\begin{aligned} u(x,y) &= (-9y-21)+(6y+41)(x-0)+y(x-0)(x-1) \\ &= x^2y+5xy+41x-9y-21, \end{aligned}$$

welches das gesuchte Polynom im Ring $\mathbb{Z}_{97}[x,y]$ ist.

Zum Schluß noch ein *Homomorphismen-Diagramm*, das die Berechnungsschritte zur Lösung eines Problems veranschaulicht, wenn Methoden mit Homomorphismen angewandt werden.



Kapitel 7

Iteration von Newton und Hensel-Konstruktion

Ziel dieses Kapitels ist es, Techniken zu studieren, um Modulo- und Auswertungshomomorphismen zu invertieren, die auf $\mathbb{Z}[x_1, \ldots, x_\nu]$ definiert sind. Die Methoden basieren auf der Iteration von Newton zur Lösung von Polynomgleichungen. Diese benötigt nur *ein* Bild der Lösung in $\mathbb{Z}_p[x_1]$, um die gesuchte Lösung in $\mathbb{Z}[x_1, \ldots, x_\nu]$ zu rekonstruieren. Ein besonders wichtiger Fall ist die Hensel-Konstruktion.

7.1 p-adische und Ideal-adische Darstellungen

7.1.1 p-adische Darstellung

Problem: Invertierung des Modulo-Homomorphismus

$$\phi_p : \mathbb{Z}[x] \longrightarrow \mathbb{Z}_p[x].$$

Der Startpunkt ist, eine andere Darstellung für ganze Zahlen und Polynome zu betrachten, die p-adische Darstellung

$$u = u_0 + u_1 p + u_2 p^2 + \ldots + u_n p^n \tag{7.1}$$

mit p ungerade positive Primzahl, n so, daß $p^{n+1} > 2|u|$ ist, $u_i \in \mathbb{Z}_p$ $(0 \le i \le n)$.

Man kann die p-adische Darstellung entweder unter Benutzung der positiven oder der symmetrischen Darstellung von $\mathbb{Z}_p$ entwickeln. Mit der positiven Darstellung ist (7.1) die gewöhnliche Basis p Darstellung der nichtnegativen ganzen Zahl u.

Prozedur zur Bestimmung der p-adischen Darstellung einer ganzen Zahl u
Aus (7.1) ergibt sich $u \equiv u_0 (\mathrm{mod}\ p)$. Mit der Modulo-Abbildung $\phi_p(a) = \mathrm{rem}(a, p)$ erhält man

$$u_0 = \phi_p(u). \tag{7.2}$$

Der nächste p-adische Koeffizient ist u_1: $u - u_0$ muß durch p teilbar sein. Aus (7.1) folgt, daß $\frac{u - u_0}{p} = u_1 + u_2 p + \ldots + u_n p^{n-1}$. Daher ist

$$u_1 = \phi_p(\frac{u - u_0}{p}).$$

Die Weiterführung dieser Methode führt auf:

$$u_i = \phi_p \left(\frac{u - [u_0 + u_1 p + u_2 p^2 + \ldots + u_{i-1} p^{i-1}]}{p^i} \right) \quad i = 1, 2, \ldots, n, \tag{7.3}$$

wobei die Division durch p^i garantiert eine *exakte* Division ist.

Beispiel:
$u = -272300$. Mit der Wahl von $p = 97$ ergeben sich die p-adischen Koeffizienten zu

$$\begin{aligned} u_0 &= \phi_p(u) = -21 \\ u_1 &= \phi_p(\frac{u - u_0}{p}) = 6 \\ u_2 &= \phi_p(\frac{u - [u_0 + u_1 p]}{p^2}) = -29. \end{aligned}$$

Versucht man u_3 zu berechnen, erhält man $\frac{u - [u_0 + u_1 p + u_2 p^2]}{p^3} = 0$. Also ist man fertig und es ist

$$-272300 = -21 + 6(97) - 29(97)^2.$$

Wie für gemischte Basisdarstellung, kann auch das Konzept der p-adischen Darstellung leicht auf Polynome

$$u(x) = \sum_e u_e x^e \in \mathbb{Z}[x]$$

erweitert werden. Seien p und n so gewählt, daß $p^{n+1} > 2u_{\max} = 2 \max_e |u_e|$. Werden die u_e in ihrer p-adischen Darstellung

$$u_e = \sum_{i=0}^{n} u_{e,i} p^i, \quad \text{mit } u_{e,i} \in \mathbb{Z}_p$$

ausgedrückt, erhält man

$$u(x) = \sum_e \left(\sum_{i=0}^{n} u_{e,i} p^i \right) x^e = \sum_{i=0}^{n} \left(\sum_e u_{e,i} x^e \right) p^i.$$

Dies ist die **p-adische Polynomdarstellung von u**. Die allgemeine Form lautet

$$u(x) = u_0(x) + u_1(x)p + \ldots + u_n(x)p^n, \tag{7.4}$$

$u_i(x) \in \mathbb{Z}_p[x]$, $i = 0, 1, \ldots, n$. Die Formeln (7.2) und (7.3) bleiben gültig.

Beispiel:
$u(x) = 14x^2 - 11x - 15 \in \mathbb{Z}[x]$. Sei $p = 5$.

$$\begin{aligned} u_0(x) &= \phi_p(u(x)) = -x^2 - x \\ u_1(x) &= \phi_p(\frac{u(x) - u_0(x)}{p}) = -2x^2 - 2x + 2 \\ u_2(x) &= \phi_p(\frac{u(x) - [u_0(x) + u_1(x)p]}{p^2}) = x^2 - 1. \end{aligned}$$

Da $u_3(x) = 0$, ist

$$u(x) = (-x^2 - x) + (-2x^2 - 2x + 2)5 + (x^2 - 1)5^2.$$

7.1.2 Approximation

Konzept der Approximation verbunden mit der p-adischen Polynomdarstellung.
Die gewöhnliche Kongruenz-Schreibweise $a(x) \equiv b(x) \bmod q$ in $\mathbb{Z}[x]$ bedeutet in bezug auf ein Hauptideal $\langle q \rangle$ in $\mathbb{Z}[x]$

$$a(x) - b(x) \in \langle q \rangle$$

(d.h. $(a(x) - b(x))$ ist ein Vielfaches von q). Bei Benutzung der Kongruenzrelation sieht man, daß man für Polynome in der p-adischen Polynomdarstellung (7.4)

$$u(x) \equiv u_0(x) \,(\bmod\ p)$$

erhält und etwas allgemeiner

$$u(x) \equiv u_0(x) + u_1(x)p + \ldots + u_{k-1}(x)p^{k-1} \,(\bmod\ p^k) \quad 1 \leq k \leq n + 1.$$

Definition 7.1
Sei $a(x) \in \mathbb{Z}[x]$ ein Polynom. Ein Polynom $b(x) \in \mathbb{Z}[x]$ heißt eine ***p-adische Approximation n-ter Ordnung*** *von $a(x)$, falls*

$$a(x) \equiv b(x) \,(mod\ p^n).$$

Der **Fehler** *bei der Approximation von $a(x)$ durch $b(x)$ ist $a(x) - b(x) \in \mathbb{Z}[x]$.*

7.1.3 Multivariate Taylor-Reihendarstellung

Sei

$$\phi_I : \mathbb{Z}_p[x_1, \ldots, x_\nu] \longrightarrow \mathbb{Z}_p[x_1] \tag{7.5}$$

mit Kern $I = \langle x_2 - \alpha_2, \ldots, x_\nu - \alpha_\nu \rangle$ für $\alpha_i \in \mathbb{Z}_p$ $(2 \leq i \leq \nu)$.
Schlüsselidee: man wähle eine geeignete Darstellung für die Lösung.
In diesem Fall ist die 'Lösung' ein multivariates Polynom $\tilde{u} = u(x_1, \ldots, x_\nu) \in \mathbb{Z}_p[x_1, \ldots, x_\nu]$ und der 'erste Term' von $\tilde{u}$ ist ein univariates Polynom $u^{(1)} \in \mathbb{Z}_p[x_1]$ mit

$$u^{(1)} = \phi_I(\tilde{u}). \tag{7.6}$$

Man beachte, daß

$$u^{(1)} = u(x_1, \alpha_2, \ldots, \alpha_\nu)$$

gilt. Zur vorhergehenden Darstellung korespondierend, nehme man an, man habe für die Lösung $\tilde{u}$ eine Darstellung der Form

$$\tilde{u} = u^{(1)} + \Delta u^{(1)} + \Delta u^{(2)} + \Delta u^{(3)} + \ldots \tag{7.7}$$

gewählt, wobei der erste Term durch (7.6) gegeben ist. Um die restlichen Terme zu bestimmen, betrachte man den 'Fehler' $e^{(1)} = \tilde{u} - u^{(1)}$ und beachte dabei, daß man aus (7.6) leicht $\phi_I(e^{(1)}) = 0$ erhält, also

$$e^{(1)} \in I. \tag{7.8}$$

Da jedes Element aus dem Ideal I als Linearkombination der Basiselemente von I dargestellt werden kann, geht (7.8) über in

$$e^{(1)} = \sum_{i=2}^{\nu} c_i(x_i - \alpha_i), \quad \text{mit } c_i \in \mathbb{Z}_p[x_1, \ldots, x_\nu]. \tag{7.9}$$

Für den ersten 'Korrekturterm' $\Delta u^{(1)}$ in der Darstellung (7.7) wählt man die linearen Terme aus dem Fehlerausdruck (7.9) definiert durch

$$\Delta u^{(1)} = \sum_{i=2}^{\nu} u_i(x_1)(x_i - \alpha_i). \tag{7.10}$$

Die Koeffizienten $u_i(x_1) \in \mathbb{Z}_p[x_1]$ sind gegeben durch

$$u_i(x_1) = \phi_I(c_i),\ 2 \leq i \leq \nu. \tag{7.11}$$

Man beachte, daß $\Delta u^{(1)} \in I$ gilt. An dieser Stelle ist die 'Approximation' von $\tilde{u}$

$$u^{(2)} = u^{(1)} + \Delta u^{(1)}$$

durch (7.6) und (7.10) definiert. Man betrachte den neuen Fehlerterm

$$e^{(2)} = \tilde{u} - u^{(2)} = e^{(1)} - \Delta u^{(1)}.$$

Durch Anwenden von (7.9) und (7.10) erhält man

$$e^{(2)} = \sum_{i=2}^{\nu} (c_i - u_i(x_1))(x_i - \alpha_i).$$

Damit gilt

$$c_i - u_i(x_1) \in I,\ 2 \leq i \leq \nu,$$

weil aus (7.11) leicht $\phi_I(c_i - u_i(x_1)) = 0$ folgt, was impliziert, daß

$$e^{(2)} \in I^2. \tag{7.12}$$

Um die letzte Aussage zu verstehen, erinnere man sich an die Definition der i-ten Potenz eines endlich erzeugten Ideals I: I^2 wird von allen Produkt-Paaren der Basiselemente von I erzeugt, I^3 von allen Produkt-Tripeln .... In unserem Fall $I = \langle x_2 - \alpha_2, \ldots, x_\nu - \alpha_\nu \rangle$ sind die Basiselemente lineare Terme. Für den speziellen Fall $\nu = 3$ erhält man z.B.:

$$\begin{aligned}
I &= \langle x_2 - \alpha_2, x_3 - \alpha_3 \rangle \\
I^2 &= \langle (x_2 - \alpha_2)^2, (x_2 - \alpha_2)(x_3 - \alpha_3), (x_3 - \alpha_3)^2 \rangle \\
I^3 &= \langle (x_2 - \alpha_2)^3, (x_2 - \alpha_2)^2(x_3 - \alpha_3), \\
&\quad (x_2 - \alpha_2)(x_3 - \alpha_3)^2, (x_3 - \alpha_3)^3 \rangle \\
&\ \ \vdots \\
I^i &= \langle (x_2 - \alpha_2)^i, (x_2 - \alpha_2)^{i-1}(x_3 - \alpha_3), \ldots, \\
&\quad (x_2 - \alpha_2)(x_3 - \alpha_3)^{i-1}, (x_3 - \alpha_3)^i \rangle.
\end{aligned}$$

Daher ist

$$e^{(2)} = \sum_{i=2}^{\nu} \sum_{j=i}^{\nu} c_{ij}(x_i - \alpha_i)(x_j - \alpha_j)$$

mit $c_{ij} \in \mathbb{Z}_p[x_1, \ldots, x_\nu]$. Der nächste Korrekturterm in der Darstellung (7.7) ist der Term $\Delta u^{(2)} \in I^2$ definiert durch

$$\Delta u^{(2)} = \sum_{i=2}^{\nu} \sum_{j=i}^{\nu} u_{ij}(x_1)(x_i - \alpha_i)(x_j - \alpha_j), \tag{7.13}$$

wobei die Koeffizienten $u_{ij}(x_1) \in \mathbb{Z}_p[x_1]$ gegeben sind durch

$$u_{ij}(x_1) = \phi_I(c_{ij}),\ 2 \leq i \leq j \leq \nu.$$

Damit hat man die 'Approximation' für $\tilde{u}$

$$u^{(3)} = u^{(2)} + \Delta u^{(2)} = u^{(1)} + \Delta u^{(1)} + \Delta u^{(2)}$$

durch (7.6), (7.10) und (7.13) definiert. Auf diese Weise fortfahrend kann man zeigen, daß

$$e^{(3)} \in I^3$$

wobei $e^{(3)} = \tilde{u} - u^{(3)}$ und weiter den nächsten Korrekturterm $\Delta u^{(3)} \in I^3$ der Form

$$\Delta u^{(3)} = \sum_{i=2}^{\nu} \sum_{j=i}^{\nu} \sum_{k=j}^{\nu} u_{ijk}(x_1)(x_i - \alpha_i)(x_j - \alpha_j)(x_k - \alpha_k)$$

definieren für Koeffizienten $u_{ijk}(x_1) \in \mathbb{Z}_p[x_1]$. Dieses Verfahren wird schließlich terminieren, weil die Lösung $\tilde{u}$ ein Polynom ist. Bezeichnet d den *Totalgrad* von $\tilde{u}$ und ist $\tilde{u}$ ein Element des Ringes $\mathbb{Z}_p[x_1][x_2, \ldots, x_\nu]$ (d.h. ein Polynom in den Unbestimmten $x_2, \ldots, x_\nu$), dann erhält man mit

$$u^{(d+1)} = u^{(1)} + \Delta u^{(1)} + \ldots + \Delta u^{(d)}$$

daß

$$e^{(d+1)} = \tilde{u} - u^{(d+1)} = 0$$

ist. Damit ist $u^{(d+1)}$ das gewünschte Polynom. Dies muß so sein, weil jeder Korrekturterm $\Delta u^{(k)} \in I^k$ den Totalgrad k (bezüglich $x_2, \ldots, x_\nu$) hat.

Die Darstellung (7.7), die gerade für ein Polynom $\tilde{u} \in \mathbb{Z}_p[x_1, \ldots, x_\nu]$ entwickelt wurde heißt *multivariate Taylor-Reihendarstellung* bezüglich dem Ideal $I = \langle x_2 - \alpha_2, \ldots, x_\nu - \alpha_\nu \rangle$. Die allgemeine Form lautet

$$\begin{aligned} u(x_1, \ldots, x_\nu) \quad = \quad & u(x_1, \alpha_2, \ldots, \alpha_\nu) + \sum_{i=2}^{\nu} u_i(x_1)(x_i - \alpha_i) \\ & + \sum_{i=2}^{\nu} \sum_{j=i}^{\nu} u_{ij}(x_1)(x_i - \alpha_i)(x_j - \alpha_j) \\ & + \sum_{i=2}^{\nu} \sum_{j=i}^{\nu} \sum_{k=j}^{\nu} u_{ijk}(x_1)(x_i - \alpha_i)(x_j - \alpha_j)(x_k - \alpha_k) + \ldots . \end{aligned} \tag{7.14}$$

Die Zahl der Terme ist hier endlich, der letzte Term enthält d geschachtelte Summationen, wobei d der Totalgrad von $u(x_1, \ldots, x_\nu)$ bezüglich der Unbestimmten $x_2, \ldots, x_\nu$ ist.

7.1.4 Ideal-adische Darstellung und Approximation

Die multivariate Taylor-Reihendarstellung (7.14) eines Polynoms $u(x) \in \mathbb{Z}_p[x]$ kann als eine direkte Verallgemeinerung der p-adischen Polynomdarstellung angesehen werden. Man erinnere sich, daß die p-adische Polynomdarstellung eines Polynoms $\tilde{u} = u(x) \in \mathbb{Z}[x]$ in der Form

$$\tilde{u} = u^{(1)} + \Delta u^{(1)} + \Delta u^{(2)} + \ldots + \Delta u^{(n)}$$

ausgedrückt werden kann, wobei

$$\begin{aligned} u^{(1)} &= u_0(x) \in \mathbb{Z}[x]/\langle p \rangle \\ \Delta u^{(k)} &= u_k(x) p^k \in \langle p \rangle^k, \quad \text{für } k = 1, 2, \ldots, n. \end{aligned}$$

Man beachte hier, daß $\mathbb{Z}[x]/\langle p \rangle = \mathbb{Z}_p[x]$ und $\langle p \rangle^k = \langle p^k \rangle$ gilt. Weiter gilt die Eigenschaft, daß der Koeffizient $u_k(x)$ im Ausdruck für $\Delta u^{(k)}$ als ein Vielfaches des Basiselementes des Ideals in dem es liegt,

$$u_k(x) \in \mathbb{Z}[x]/\langle p \rangle, \; 1 \leq k \leq n$$

erfüllt. Im p-adischen Falle kann man eine Folge p-adischer Approximationen der Ordnung $k+1$ definieren

$$u^{(k+1)} \in \mathbb{Z}[x]/\langle p \rangle^{k+1}, \quad \text{für } k = 1, 2, \ldots, n,$$

wobei

$$u^{(k+1)} = u^{(1)} + \Delta u^{(1)} + \Delta u^{(2)} + \ldots + \Delta u^{(k)}.$$

Beim Definieren des k-ten Elements der Folge ergibt sich die Approximation $u^{(k)} \in \mathbb{Z}[x]/\langle p \rangle^k$ und man erhält die neue Approximation $u^{(k+1)} \in \mathbb{Z}[x]/\langle p \rangle^{k+1}$ durch Addieren des Termes $\Delta u^{(k)} \in \langle p \rangle^k$. Diese Addition

$$u^{(k+1)} = u^{(k)} + \Delta u^{(k)}$$

ist eine Addition in dem größeren Ring $\mathbb{Z}[x]/\langle p \rangle^{k+1}$, sie ist erlaubt durch die natürliche Einbettung des Ringes $\mathbb{Z}[x]/\langle p \rangle^k$ in den größeren Ring $\mathbb{Z}[x]/\langle p \rangle^{k+1}$. Daher liegen die sukzessiven p-adischen Approximationen $u^{(1)}, u^{(2)}, u^{(3)}, \ldots$ von $\tilde{u} \in \mathbb{Z}[x]$ in einer Folge von Teilmengen wachsender Größe von $\mathbb{Z}[x]$

$$\mathbb{Z}[x]/\langle p \rangle \subset \mathbb{Z}[x]/\langle p \rangle^2 \subset \mathbb{Z}[x]/\langle p \rangle^3 \subset \cdots \subset \mathbb{Z}[x].$$

Beachtet man, daß ein Polynom $\tilde{u} \in \mathbb{Z}[x]$ eine endliche p-adische Polynomdarstellung hat, so ist klar, daß für ein $k = n$ $\mathbb{Z}[x]/\langle p \rangle^{n+1}$ groß genug ist, das Polynom $\tilde{u}$ zu enthalten.

Die multivariate Taylor-Reihendarstellung (7.14) kann auf eine abstrakte äquivalente Weise bezüglich dem Ideal I angesehen werden, statt dem obigen Ideal $\langle q\rangle$. Damit ergibt sich

$$\mathbb{Z}_p[x]/I \subset \mathbb{Z}_p[x]/I^2 \subset \mathbb{Z}_p[x]/I^3 \subset \cdots \subset \mathbb{Z}_p[x].$$

Wie im p-adischen Falle gilt, da die multivariate Taylor-Reihendarstellung für $\tilde{u}$ endlich ist, daß es einen Index $k = d$ gibt, für den $\mathbb{Z}_p[x]/I^{d+1}$ groß genug ist das Polynom $\tilde{u}$ zu enthalten.

Angesichts der geschlossenen Korrespondenz mit der p-adischen Darstellung von Polynomen heißt die multivariate Taylor-Reihendarstellung (7.14) eines Polynoms $u(x) \in \mathbb{Z}_p[x]$ auch die **Ideal-adische Darstellung** von $u(x)$ bezüglich dem Ideal $I = \langle x_2 - \alpha_2, \ldots, x_\nu - \alpha_\nu\rangle$. Das oben erwähnte Konzept der Approximation wird in der folgenden Definition präzisiert.

Definition 7.2
Sei D ein noetherscher Integritätsbereich und I ein Ideal in D. Zu einem Element $a \in D$ heißt das Element $b \in D$ eine **Ideal-adische Approximation n-ter Ordnung** *für a bezüglich des Ideales I, falls*

$$a \equiv b (\mathrm{mod}\ I^n).$$

Der **Fehler** *bei der Approximation von a durch b ist das Element $a - b \in D$.*

7.2 Iteration nach Newton für $f(u) = 0$

7.2.1 Lineare p-adische Iteration

Das Ziel ist, ein Verfahren zur Invertierung des Modulo-Homomorphismus $\phi_p : \mathbb{Z}[x] \longrightarrow \mathbb{Z}_p[x]$ zu konstruieren, korrespondierend zum chinesischen Restealgorithmus. In einem neuen Anlauf nimmt man an, daß nur eine Primzahl p benötigt wird und das Bild $u_0(x) \in \mathbb{Z}_p[x]$ der gesuchten Lösung $u(x) \in \mathbb{Z}[x]$ bekannt ist. In der Terminologie des vorhergehenden Abschnittes ist $u_0(x)$ eine p-adische Approximation erster Ordnung (oder erster Stufe) für $u(x)$ und der erste Term in der p-adischen Polynomdarstellung von $u(x)$. Es wird nun ein Verfahren entwickelt, das sukzessiv die Approximation k-ter Ordnung

$$u_0(x) + u_1(x)p + \ldots + u_{k-1}(x)p^{k-1} \in \mathbb{Z}_{p^k}[x]$$

für $k = 1, 2, \ldots, n+1$ berechnet. Die Approximation $(n+1)$-ter Ordnung, die im Ring $\mathbb{Z}_{p^{n+1}}[x]$ liegt, ist dann die gesuchte Lösung $u(x) \in \mathbb{Z}[x]$ (vorausgesetzt n wurde

groß genug gewählt). Dieser allgemeine Prozeß heißt *lifting*, d.h. das Bild $u_0(x) \in \mathbb{Z}_p[x]$ der Lösung $u(x)$ wird in den größeren Ring $\mathbb{Z}[x]$ „hochgehoben".
Der Lifting-Vorgang enthält mehr Information über die Lösung $u(x)$ wie das einzelne Bild $u_0(x)$. Es wird angenommen, daß die zusätzliche Information in Form einer oder mehrerer Gleichungen (meist nichtlinear) dargestellt werden kann, die $u(x)$ erfüllen muß. Für jetzt wird angenommen, daß für die Lösung $u = u(x)$ bekannt ist, daß sie

$$f(u) = 0 \tag{7.15}$$

erfüllt, wobei $f(u) \in \mathbb{Z}[x][u]$ – d.h. $f(u)$ ist ein Polynom in u mit Koeffizienten aus dem Ring $\mathbb{Z}[x]$. Die Idee dieses neuen Zuganges ist, eine iterative Methode zu bekommen, die die gegebene p-adische Approximation erster Ordnung $u_0(x)$ sukzessive in p-adische Approximationen höherer Ordnung der Lösung $u(x)$ von (7.15) verbessert. Der iterative Vorgang endet, falls (7.15) eine polynomiale Lösung $u(x)$ hat, also – in der obigen Schreibweise – falls die p-adische Approximation der Ordnung $n+1$ von $u(x)$ selbst $u(x)$ ist.

Man erinnere sich an das klassische Newton-Verfahren zur Lösung einer nichtlinearen Gleichung der Form (7.15) mit der traditionellen analytischen Voraussetzung, in der $f(u)$ eine differenzierbare reellwertige Funktion einer reellen Veränderlichen u ist. Bezeichnet $u^{(k)}$ eine Approximation der Lösung $\tilde{u}$ und entwickelt man die Funktion $f(u)$ in eine Taylor-Reihe um den Punkt $u^{(k)}$, so erhält man

$$f(u) = f(u^{(k)}) + f'(u^{(k)})(u - u^{(k)}) + \frac{1}{2} f''(u^{(k)})(u - u^{(k)})^2 + \dots \quad .$$

Setzt man $u = \tilde{u}$, ergibt die linke Seite Null; und nimmt man nur die linearen Terme der Taylor-Reihe so erhält man näherungsweise

$$0 \approx f(u^{(k)}) + f'(u^{(k)})(\tilde{u} - u^{(k)}).$$

Löst man nach $\tilde{u}$ auf und bezeichnet dies als neue Approximation $u^{(k+1)}$, dann ergibt sich die Newtonsche Iterationsformel:

$$u^{(k+1)} = u^{(k)} - \frac{f(u^{(k)})}{f'(u^{(k)})}$$

(wobei $f'(u^{(k)}) \neq 0$ benötigt wird). Die Iteration muß mit einem Schätzwert $u^{(1)}$ gestartet werden. Unter Benutzung von Techniken der reellen Analysis kann gezeigt werden, falls u 'dicht genug' an der Lösung $\tilde{u}$ von $f(u) = 0$ liegt und $f'(u) \neq 0$ ist, daß die oben angegebene unendliche Iteration (quadratisch) gegen die Lösung $\tilde{u}$ konvergiert. Nun wird eine ähnliche Iterationsformel für den Polynomansatz entwickelt, mit zwei signifikanten rechentechnischen Vorteilen gegenüber dem traditionellen analytischen Fall: (i) es genügt die p-adische Approximation erster Ordnung zur *Konvergenzgarantie*, und (ii) die Iteration ist *endlich*.

Man hofft die Polynomgleichung – *unter der Annahme, daß es eine Lösung* $\tilde{u} = u(x) \in \mathbb{Z}[x]$ *gibt* – zu lösen, bei gegebener p-adischer Approximation erster Ordnung $u_0(x) \in \mathbb{Z}_p[x]$ von $\tilde{u}$. (Man beachte, daß eine beliebige Polynomgleichung der Form (7.15) i.a. keine Polynomlösung haben muß, aber hier wird ein Kontext angenommen in dem bekannt ist, daß eine Polynomlösung existiert). Durch Schreiben der Lösung in ihrer p-adischen Polynomdarstellung

$$\tilde{u} = u_0(x) + u_1(x)p + \ldots + u_n(x)p^n \tag{7.16}$$

hofft man, die p-adischen Polynomkoeffizienten $u_i(x) \in \mathbb{Z}_p[x]$ für $i = 1, 2, \ldots, n$ ($u_0(x)$ ist gegeben) zu bestimmen. $u^{(k)}$ bezeichnet die p-adische Approximation k-ter Ordnung von $\tilde{u}$, gegeben durch die ersten k Terme von (7.16), also $u^{(1)} = u_0(x)$ und allgemein

$$u^{(k)} = u_0(x) + u_1(x)p + \ldots + u_{k-1}(x)p^{k-1},\ 1 \le k \le n+1.$$

Man möchte eine Iterationsformel, die im Schritt k zur gegebenen p-adischen Approximation k-ter Ordnung $u^{(k)}$ den p-adischen Polynomkoeffizienten $u_k(x) \in \mathbb{Z}_p[x]$ berechnet und auf die p-adischen Approximation $(k+1)$-ter Ordnung führt

$$u^{(k+1)} = u^{(k)} + u_k(x)p^k,\ 1 \le k \le n. \tag{7.17}$$

Dazu benötigt man das Konzept der Taylorentwicklung im Sinne der folgenden Sätze:

Satz 7.3
Sei $a(x) \in D[x]$ ein univariates Polynom über einem beliebigen Integritätsbereich D. Dann gilt im Polynomring $D[x][y] = D[x,y]$

$$a(x+y) = a(x) + a'(x)y + b(x,y)y^2$$

für ein gewisses Polynom $b(x,y) \in D[x,y]$.

Satz 7.4
Sei $a(x,y) \in D[x,y]$ ein bivariates Polynom über einem beliebigen Integritätsbereich D. Dann gilt im Polynomring $D[x,y][\xi,\eta] = D[x,y,\xi,\eta]$

$$\begin{aligned} a(x+\xi, y+\eta) &= a(x,y) + a_x(x,y)\xi + a_y(x,y)\eta + b_1(x,y,\xi,\eta)\xi^2 \\ &\quad + b_2(x,y,\xi,\eta)\xi\eta + b_3(x,y,\xi,\eta)\eta^2 \end{aligned}$$

für gewisse Polynome $b_i(x,y,\xi,\eta) \in D[x,y,\xi,\eta],\ i = 1,2,3.$

Das Polynom $f(u) \in D[u]$ mit $D = \mathbb{Z}[x]$ hat die folgende Taylorentwicklung

$$\begin{aligned} f(u^{(k)} + u_k(x)p^k) &= f(u^{(k)}) + f'(u^{(k)})u_k(x)p^k \\ &\quad + g(u^{(k)}, u_k(x)p^k)[u_k(x)]^2 p^{2k} \end{aligned} \tag{7.18}$$

für ein $g(u, w) \in D[u, w]$. Nun benutzt man die Eigenschaften (Addition, Subtraktion, Multiplikation) der Kongruenzrelation. Als direkte Konsequenz dieser Eigenschaften folgt, falls I ein beliebiges Ideal in einem kommutativen Ring R und $h(x) \in R[x]$ ein beliebiges Polynom über R ist, daß für $a, b \in R$ gilt

$$a \equiv b(\mathrm{mod}\ I) \Rightarrow h(a) \equiv h(b)(\mathrm{mod}\ I). \tag{7.19}$$

Da $u^{(k)} \equiv \tilde{u}(\mathrm{mod}\ p^k)$ erhält man durch Anwenden der Eigenschaft (7.19) und der Tatsache, daß $f(\tilde{u}) = 0$ ist

$$f(u^{(k)}) \equiv 0(\mathrm{mod}\ p^k).$$

Ähnlich gilt

$$f(u^{(k)} + u_k(x)p^k) \equiv 0(\mathrm{mod}\ p^{k+1}),$$

falls durch (7.17) die p-adische Approximation $(k+1)$-ter Ordnung $u^{(k+1)}$ definiert ist. Daher kann man in (7.18) durch p^k dividieren und erhält

$$\begin{aligned} \frac{f(u^{(k)} + u_k(x)p^k)}{p^k} &= \frac{f(u^{(k)})}{p^k} + f'(u^{(k)})u_k(x) \\ &\quad + g(u^{(k)}, u_k(x)p^k)[u_k(x)]^2 p^k. \end{aligned}$$

Wendet man nun den Modulo-Homomorphismus ϕ_p an und beachtet, daß die linke Seite ein Vielfaches von p ist, erkennt man, daß der gesuchte p-adische Polynomkoeffizient $u_k(x) \in \mathbb{Z}_p[x]$ die Gleichung

$$0 = \phi_p\left(\frac{f(u^{(k)})}{p^k}\right) + \phi_p(f'(u^{(k)}))u_k(x) \in \mathbb{Z}_p[x]$$

erfüllen muß. Schließlich folgt — da $u^{(k)} \equiv u^{(1)}(\mathrm{mod}\ p)$ für alle $k \geq 1$ — mit der Eigenschaft (7.19), daß

$$f'(u^{(k)}) \equiv f'(u^{(1)})(\mathrm{mod}\ p).$$

Genügt nun die gegebene Approximation erster Ordnung $u^{(1)}$ der Bedingung

$$f'(u^{(1)}) \not\equiv 0 \quad (\mathrm{mod}\ p),$$

dann ist der gesuchte p-adische Polynomkoeffizient gegeben durch

$$u_k(x) = -\frac{\phi_p\left(\dfrac{f(u^{(k)})}{p^k}\right)}{\phi_p(f'(u^{(1)}))} \in \mathbb{Z}_p[x]. \tag{7.20}$$

Die in (7.20) auftretende Division ist im Polynomring $\mathbb{Z}_p[x]$ exakt, falls die Gleichung (7.15) eine Polynomlösung besitzt. Die Iterationsformel (7.17) bildet zusammen mit der *linearen Aktualisierungsformel* (7.20) das sogenannte *lineare p-adische Newtonverfahren.* Man beachte, daß die Berechnung von $f(u^{(k)})$ in der Formel (7.20) im Ring $\mathbb{Z}[x]$ durchgeführt werden muß, gefolgt von der exakten Division durch p^k in $\mathbb{Z}[x]$, bevor der Modulo-Homomorphismus ϕ_p angewandt werden kann.

Beispiel:
Man betrachte das Problem, das Polynom $u(x) \in \mathbb{Z}[x]$ zu bestimmen, das die Quadratwurzel des Polynoms

$$a(x) = 36x^4 - 180x^3 + 93x^2 + 330x + 121 \in \mathbb{Z}[x]$$

ist (unter der Annahme, daß $a(x)$ quadratisch ist). Dann kann $u(x)$ als Lösung der Polynomgleichung

$$f(u) = a(x) - u^2 = 0$$

ausgedrückt werden. Wählt man $p = 5$, so muß die p-adische Approximation erster Ordung $u^{(1)} = u_0(x) \in \mathbb{Z}_5[x]$ eine Quadratwurzel von $\phi_5(a(x))$ in $\mathbb{Z}_5[x]$ sein. Es ist

$$\phi_5(a(x)) = x^4 - 2x^2 + 1,$$

das klar die Quadratwurzel

$$u^{(1)} = u_0(x) = x^2 - 1 \in \mathbb{Z}_5[x]$$

hat. Um das lineare p-adische Newtonverfahren anzuwenden, beachte man zuerst, daß

$$\phi_5(f'(u^{(1)})) = \phi_5(-2u^{(1)}) = -2x^2 + 2$$

ist. Damit führt

$$\begin{aligned} u_1(x) &= -\frac{\phi_5\left(\dfrac{f(u^{(1)})}{5}\right)}{(-2x^2+2)} = -\frac{\phi_5\left(\dfrac{35x^4 - 180x^3 + 95x^2 + 330x + 120}{5}\right)}{(-2x^2+2)} \\ &= -\frac{(2x^4 - x^3 - x^2 + x - 1)}{(-2x^2+2)} = x^2 + 2x - 2 \in \mathbb{Z}_5[x] \end{aligned}$$

auf

$$u^{(2)} = (x^2 - 1) + (x^2 + 2x - 2)5 \in \mathbb{Z}_{25}[x].$$

Ähnlich führt

$$u_2(x) = -\frac{(-2x^3 + 2x)}{(-2x^2 + 2)} = -x \in \mathbb{Z}_5[x]$$

auf

$$u^{(3)} = (x^2-1) + (x^2+2x-2)5 + (-x)5^2 \in \mathbb{Z}_{125}[x].$$

Fährt man fort den p-adische Polynomkoeffizient $u_3(x)$ zu berechnen, erhält man $f(u^{(3)}) = 0$ (im Ring $\mathbb{Z}[x]$), also ist man fertig. Die gesuchte Quadratwurzel von $a(x)$ ist daher

$$u(x) = u^{(3)} = 6x^2 - 15x - 11 \in \mathbb{Z}[x]. \quad \square$$

7.2.2 Quadratische p-adische Iteration

Beim Newton-Verfahren — wie in (7.17) und (7.20) beschrieben — steigt die Ordnung der Approximation pro Iteration um eins. Es ist aber auch möglich das Newton-Verfahren auf eine Weise zu entwickeln, daß sich die Ordnung der Approximation pro Iterationsschritt *verdoppelt*. Dies entspricht dem Konzept der *quadratischen* Konvergenz der bekannten analytischen Anwendungen des Newton-Verfahrens. Bei der quadratischen Version ergibt sich im Schritt k die Approximation der Ordnung $n_k = 2^{k-1}$

$$u^{(k)} = u_0(x) + u_1(x)p + \ldots + u_{n_k-1}(x)p^{n_k-1}$$

für eine Lösung $\tilde{u}$ von $f(u) = 0$ und man berechnet die Aktualisierung $\Delta u^{(k)}$, so daß

$$u^{(k+1)} = u^{(k)} + \Delta u^{(k)} \tag{7.21}$$

eine Approximation der Ordnung $2n_k = 2^k$ ist; d.h.

$$\begin{aligned} \Delta u^{(k)} &= u_{n_k}(x)p^{n_k} + \ldots + u_{2n_k-1}(x)p^{2n_k-1} \\ &= p^{n_k}\left(u_{n_k}(x) + \ldots + u_{2n_k-1}(x)p^{n_k-1}\right). \end{aligned}$$

Der Formel (7.18) entsprechend erhält man mit Satz 7.3

$$\begin{aligned} f(u^{(k)} + \Delta u^{(k)}) &= f(u^{(k)}) + f'(u^{(k)})\Delta u^{(k)} \\ &\quad + g(u^{(k)}, \Delta u^{(k)})[\Delta u^{(k)}]^2 \end{aligned}$$

für ein gewisses Polynom $g(u, w)$. Beachtet man, daß (von oben) $\Delta u^{(k)}$ durch p^{n_k} dividiert werden kann und benutzt man Argumente ähnlich dem linearen Fall erhält man folgende Formel, die von dem aktualisierten $\Delta u^{(k)}$ erfüllt werden muß:

$$0 = \phi_{p^{n_k}}\left(\frac{f(u^{(k)})}{p^{n_k}}\right) + \phi_{p^{n_k}}(f'(u^{(k)}))\frac{\Delta u^{(k)}}{p^{n_k}} \in \mathbb{Z}_{p^{n_k}}[x], \tag{7.22}$$

wobei $n_k = 2^{k-1}$. Wie zuvor ergibt sich das Ergebnis, daß für alle $k \geq 1$

$$f'(u^{(k)}) \equiv f'(u^{(1)}) \pmod{p}$$

gilt. Zu diesem Zeitpunkt führt dieses Ergebnis zu keiner Vereinfachung der Ableitung, da der Modulo-Homomorphismus, der auf die Ableitung angewandt werden muß, nun $\phi_{p^{n_k}}$ statt ϕ_p ist. Gleichwohl wünscht man wieder durch den Ableitungsterm in (7.22) zu dividieren. Die dazu nötige Bedingung, daß dieser ungleich Null ist, ist genau die gleiche wie für die lineare Aktualisierungsformel

$$f'(u^{(1)}) \not\equiv 0 \pmod p,$$

weil dies, aus oben folgend, garantiert, daß $f'(u^{(k)}) \not\equiv 0 \pmod p$ gilt und damit auch $f'(u^{(k)}) \not\equiv 0 \pmod{p^{n_k}}$. (Mit anderen Worten, ist der Ableitungsterm in (7.22) ungleich Null für $k = 1$, dann auch für alle $k \geq 1$). Schließlich liefert das Auflösen nach dem Aktualisierungsterm in (7.22) die *quadratische Aktualisierungsformel*

$$\frac{\Delta u^{(k)}}{p^{n_k}} = -\frac{\phi_{p^{n_k}}\left(\dfrac{f(u^{(k)})}{p^{n_k}}\right)}{\phi_{p^{n_k}}(f'(u^{(k)}))} \in \mathbb{Z}_{p^{n_k}}[x]. \tag{7.23}$$

Wie bei der linearen Aktualisierungsformel muß die Division in (7.23) im Polynomring $\mathbb{Z}_{p^{n_k}}[x]$ exakt sein, falls eine Polynomlösung zum Originalproblem existiert.

Der Satz 7.5 beweist formal die quadratische Konvergenzeigenschaft des p-adischen Newtonverfahrens (7.21) mit der quadratischen Aktualisierungsformel (7.23). Bei den meisten praktischen Problemen, die das p-adische Newtonverfahren benötigen, wird die lineare Methode der quadratischen Methode vorgezogen. Man beachte, daß die quadratische Iteration weniger Schritte benötigt, aber die Kosten für jeden Iterationsschritt (außer dem ersten) bedeutend höher sind als bei der linearen Iteration, weil der Ring $\mathbb{Z}_{p^{n_k}}[x]$, in dem die Aktualisierungsformel (7.23) berechnet werden muß, einen Zuwachs größer als k hat. Außerdem muß die Ableitung, die im Divisor in (7.23) auftritt, bei jedem Iterationsschritt neu berechnet werden, während der Divisor in der linearen Aktualisierungsformel (7.20) für alle Iterationsschritte fest ist. Aus diesen Gründen ist die lineare Iteration der quadratischen Iteration hinsichtlich der Gesamteffizienz oft vorzuziehen.

Satz 7.5
Sei $f(u) \in \mathbb{Z}[x]$ so, daß die Polynomgleichung $f(u) = 0$ eine Polynomlösung $\tilde{u} = u(x) \in \mathbb{Z}[x]$ besitzt. $u_0(x) \in \mathbb{Z}_p[x]$ sei eine p-adische Approximation erster Ordnung für die Lösung $\tilde{u}$, so daß

$$f(u_0(x)) \equiv 0 \pmod p$$

erfüllt ist. Des weiteren erfülle $u_0(x)$

$$f'(u_0(x)) \not\equiv 0 \pmod p.$$

Dann ist die Iterationsfolge

$$\begin{aligned} u^{(1)} &= u_0(x) \\ u^{(k+1)} &= u^{(k)} + \Delta u^{(k)}, \quad k = 1, 2, 3, \ldots, \end{aligned}$$

wobei $\Delta u^{(k)}$ durch die quadratische Aktualisierungsformel (7.23) definiert ist, so, daß $u^{(k+1)}$ eine p-adische Approximation der Ordnung 2^k zur Lösung $\tilde{u}$ ist.

Der Beweis wird übergangen. Er ist eine Induktion über k.

7.2.3 Ideal-adische Iteration

Nun gilt es, einen multivariaten Auswertungshomomorphismus

$$\phi_I : \mathbb{Z}_p[x_1, \ldots, x_\nu] \longrightarrow \mathbb{Z}_p[x_1]$$

mit Kern $I = \langle x_2 - \alpha_2, \ldots, x_\nu - \alpha_\nu \rangle$ für bestimmte Werte $\alpha_i \in \mathbb{Z}_p$ $(2 \le i \le \nu)$ zu invertieren. Der Invertierungsvorgang wird durch eine Ideal-adische Version des Newtonverfahrens ausgeführt. Man erhält die Ideal-adische Approximation erster Ordnung

$$u^{(1)} = \phi_I(\tilde{u}) \in \mathbb{Z}_p[x_1] = \mathbb{Z}_p[x]/I$$

(im folgenden steht $\mathbb{Z}_p[x]$ für $\mathbb{Z}_p[x_1, \ldots, x_\nu]$) zur Lösung $\tilde{u} \in \mathbb{Z}_p[x]$ und wie vorhin gelte — als zusätzliche Information — fortan die Annahme, daß die Lösung $\tilde{u}$ der Polynomgleichung

$$f(u) = 0$$

mit $f(u) \in \mathbb{Z}_p[x][u]$ genügt. Man möchte eine Iterationsformel definieren, so daß beim k-ten Schritt die Ideal-adische Approximation k-ter Ordnung $u^{(k)}$ aktualisiert wird auf die Ideal-adische Approximation $(k+1)$-ter Ordnung $u^{(k+1)}$ durch Addition des Korrekturterms $\Delta u^{(k)} \in I^k$. Durch Anwenden des Satzes 7.3 auf das Polynom $f(u) \in \mathbb{Z}_p[x][u]$ erhält man folgende 'Taylorentwicklung':

$$\begin{aligned} f(u^{(k)} + \Delta u^{(k)}) &= f(u^{(k)}) + f'(u^{(k)})\Delta u^{(k)} \\ &\quad + g(u^{(k)}, \Delta u^{(k)})[\Delta u^{(k)}]^2 \end{aligned} \tag{7.24}$$

für ein gewisses Polynom $g(u, w)$. Ist nun $u^{(k)} + \Delta u^{(k)}$ die Ideal-adische Approximation $(k+1)$-ter Ordnung $u^{(k+1)}$ und benutzt man die Eigenschaft (7.19), so ergibt sich

$$f(u^{(k)} + \Delta u^{(k)}) \in I^{k+1}.$$

Ebenso, da $\Delta u^{(k)} \in I^k$, folgt

$$[\Delta u^{(k)}]^2 \in I^{2k}.$$

Deshalb führt das Anwenden des Homomorphismus $\phi_{I^{k+1}}$ auf (7.24) zur Gleichung

$$0 = \phi_{I^{k+1}}(f(u^{(k)})) + \phi_{I^{k+1}}(f'(u^{(k)}))\Delta u^{(k)} \in \mathbb{Z}_p[x]/I^{k+1}, \tag{7.25}$$

die vom Korrekturterm $\Delta u^{(k)} \in I^k$ erfüllt werden muß.
Man betrachte den Iterationsschritt $k = 1$. In diesem Fall hat der Korrekturterm $\Delta u^{(1)} \in I$ die Form

$$\Delta u^{(1)} = \sum_{i=2}^{\nu} u_i(x_1)(x_i - \alpha_i), \tag{7.26}$$

wobei die Koeffizienten $u_i(x_1) \in \mathbb{Z}_p[x_1]$ zu bestimmen sind. Benutzt man Eigenschaft (7.19) und die Tatsache, daß $u^{(1)} \equiv \tilde{u} \pmod{I}$ ist, so schließt man, daß $f(u^{(1)}) \in I$ und kann daher

$$f(u^{(1)}) = \sum_{i=2}^{\nu} c_i(x_i - \alpha_i) \tag{7.27}$$

für gewisse Koeffizienten $c_i \in \mathbb{Z}_p[x]$, $2 \leq i \leq \nu$ schreiben. Nun wird der Homomorphismus ϕ_{I^2} für $k = 1$ in der Gleichung (7.25) angewandt und da ϕ_{I^2} alle Ideal-adischen Terme vom Totalgrad größer oder gleich 2 unterdrückt, folgt aus (7.27)

$$\phi_{I^2}(f(u^{(1)})) = \sum_{i=2}^{\nu} c_i(x_1)(x_i - \alpha_i),$$

wobei die Koeffizienten $c_i(x_1) \in \mathbb{Z}_p[x_1]$ definiert sind durch die Koeffizienten $c_i \in \mathbb{Z}_p[x]$ die in (7.27) durch

$$c_i(x_1) = \phi_I(c_i),\ 2 \leq i \leq \nu$$

auftreten. Gleichung (7.25) ergibt jetzt

$$0 = \sum_{i=2}^{\nu} c_i(x_1)(x_i - \alpha_i) + \phi_{I^2}(f'(u^{(1)})) \left(\sum_{i=2}^{\nu} u_i(x_1)(x_i - \alpha_i) \right) \in \mathbb{Z}_p[x]/I^2. \tag{7.28}$$

Nun kann die Ideal-adische Darstellung von $\phi_{I^2}(f'(u^{(1)}))$ in der Form

$$\phi_{I^2}(f'(u^{(1)})) = \phi_I(f'(u^{(1)})) + \sum_{i=2}^{\nu} d_i(x_1)(x_i - \alpha_i)$$

für gewisse Koeffizienten $d_i(x_1) \in \mathbb{Z}_p[x_1]$, $2 \leq i \leq \nu$ geschrieben werden. Setzt man diese Form in Gleichung (7.28) ein, führt dies auf

$$0 = \sum_{i=2}^{\nu} c_i(x_1)(x_i - \alpha_i) + \phi_I(f'(u^{(1)})) \left(\sum_{i=2}^{\nu} u_i(x_1)(x_i - \alpha_i) \right) \in \mathbb{Z}_p[x]/I^2, \tag{7.29}$$

wobei berücksichtigt wurde, daß

$$\left(\sum_{i=2}^{\nu} d_i(x_1)(x_i - \alpha_i) \right) \left(\sum_{i=2}^{\nu} u_i(x_1)(x_i - \alpha_i) \right) \in I^2.$$

Ein Koeffizientenvergleich auf der linken und rechten Seite der Gleichung (7.29) führt schließlich auf

$$u_i(x_1) = -\frac{c_i(x_1)}{\phi_I(f'(u^{(1)}))} \in \mathbb{Z}_p[x_1],\ 2 \leq i \leq \nu. \tag{7.30}$$

Gleichung (7.30) ist die gesuchte Aktualisierungsformel, die den Korrekturterm (7.26) definiert; die Division, die in (7.30) vorkommt muß in dem univariaten Polynomring $\mathbb{Z}_p[x_1]$ exakt sein, falls die gegebene Gleichung $f(u) = 0$ eine Polynomlösung hat. Man beachte, daß die Koeffizienten $c_i(x_1)$, die in (7.30) auftreten, einfach die Koeffizienten der linearen Terme in der Ideal-adischen Darstellung von $f(u^{(1)})$ sind.

Im allgemeinen Iterationsschritt ist der k-te Korrekturterm $\Delta u^{(k)} \in I^k$ der Term mit Totalgrad k in der Ideal-adischen Darstellung der Lösung $\tilde{u} = u(x_1, \ldots, x_\nu)$ und seine allgemeine Form besteht aus k geschachtelten Summationen wie folgt:

$$\Delta u^{(k)} = \sum_{i_1=2}^{\nu} \cdots \sum_{i_k=i_{k-1}}^{\nu} u_{\mathrm{i}}(x_1) \prod_{j=1}^{k} (x_{i_j} - \alpha_{i_j}), \tag{7.31}$$

wobei der Multiindex i den Vektor der Indizes i$= (i_1, \ldots, i_k)$ bezeichnet. Die Koeffizienten $u_{\mathrm{i}}(x_1) \in \mathbb{Z}_p[x_1]$ sind zu bestimmen. Man erhält die Ideal-adische Approximation k-ter Ordnung $u^{(k)}$ und der Korrekturterm $\Delta u^{(k)}$ muß die Gleichung (7.25) erfüllen. Wie vorhin schließt man, daß $f(u^{(k)}) \in I^k$ und folgert daraus, daß

$$\phi_{I^{k+1}}(f(u^{(k)})) = \sum_{i_1=2}^{\nu} \cdots \sum_{i_k=i_{k-1}}^{\nu} c_{\mathrm{i}}(x_1) \prod_{j=1}^{k} (x_{i_j} - \alpha_{i_j})$$

für gewisse Koeffizienten $c_{\mathrm{i}}(x_1) \in \mathbb{Z}_p[x_1]$ gilt. Ebenso kann der Term $\phi_{I^{k+1}}(f'(u^{(k)}))$ in der Gleichung (7.25) durch $\phi_I(f'(u^{(k)}))$ ersetzt werden, weil, wie im Fall $k = 1$, die Terme der Ordnung größer als 1 in der Ideal-adischen Darstellung von $\phi_{I^{k+1}}(f'(u^{(k)}))$ verschwinden, wenn mit $\Delta u^{(k)} \in I^k$ durchmultipliziert wird (da es eine Multiplikation im Ring $\mathbb{Z}_p[x]/I^{k+1}$ ist). Da für alle $k \geq 1$ $u^{(k)} \equiv u^{(1)} \pmod I$ ist, wird mit Eigenschaft (7.19) impliziert, daß $f'(u^{(k)}) \equiv f'(u^{(1)}) \pmod I$ gilt; d.h. es gilt

$$\phi_I(f'(u^{(k)})) = \phi_I(f'(u^{(1)})) \quad \text{für alle } k \geq 1.$$

Gleichung (7.25) geht damit über in

$$\begin{aligned} 0 \;=\; & \sum_{i_1=2}^{\nu} \cdots \sum_{i_k=i_{k-1}}^{\nu} c_{\mathrm{i}}(x_1) \prod_{j=1}^{k} (x_{i_j} - \alpha_{i_j}) \\ & + \phi_I(f'(u^{(1)})) \left(\sum_{i_1=2}^{\nu} \cdots \sum_{i_k=i_{k-1}}^{\nu} u_{\mathrm{i}}(x_1) \prod_{j=1}^{k} (x_{i_j} - \alpha_{i_j}) \right) \in \mathbb{Z}_p[x]/I^{k+1}. \end{aligned} \tag{7.32}$$

Erfüllt die gegebene Approximation erster Ordnung $u^{(1)}$ schließlich die Bedingung

$$f'(u^{(1)}) \not\equiv 0 \pmod{I} \tag{7.33}$$

erhält man durch Koeffizientenvergleich auf der linken und rechten Seite der Gleichung (7.32) das *lineare Ideal-adische Newtonverfahren*:

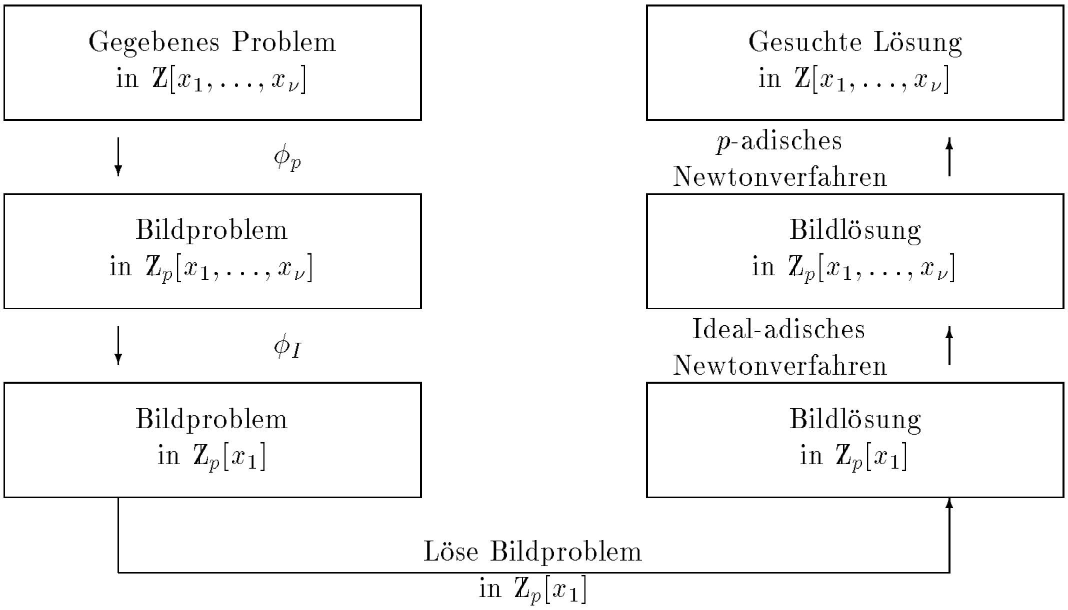
$$u^{(k+1)} = u^{(k)} + \Delta u^{(k)}, \tag{7.34}$$

wobei $\Delta u^{(k)}$ der Korrekturterm (7.31) ist, dessen Koeffizienten definiert sind durch

$$u_{\mathrm{i}}(x_1) = -\frac{c_{\mathrm{i}}(x_1)}{\phi_I(f'(u^{(1)}))} \in \mathbb{Z}_p[x_1]. \tag{7.35}$$

Die Division, die in (7.35) auftritt, muß im univariaten Polynomring $\mathbb{Z}_p[x_1]$ exakt sein, falls die gegebene Gleichung $f(u) = 0$ eine Polynomlösung besitzt. Man beachte, daß die Koeffizienten $c_{\mathrm{i}}(x_1)$, die in (7.35) vorkommen, einfach die Koeffizienten der Terme mit Totalgrad k in der Ideal-adischen Darstellung von $f(u^{(k)})$ sind. Des weiteren beachte man, daß $f(u^{(k)})$ keine Terme vom Totalgrad kleiner k (bezüglich I) hat.

Beim linearen Ideal-adischen Newtonverfahren (7.34)-(7.35) werden im Iterationsschritt k *alle* Ideal-adischen Terme der Lösung $\tilde{u}$, die einen Totalgrad k (bezüglich I) haben, berechnet. Es ist wie im p-adischen Fall möglich, ein quadratisch Ideal-adisches Newtonverfahren zu definieren. Solch eine Iteration würde im k-ten Iterationsschritt eine Ideal-adische Approximation 2^k-ter Ordnung $u^{(k+1)}$ ergeben. In anderen Worten, die quadratische Iteration würde im k-ten Iterationsschritt alle Ideal-adischen Terme der Lösung $\tilde{u}$ berechnen, die einen Totalgrad von $2^{k-1}, 2^{k-1}+1, \ldots, 2^k - 1$ haben. Wie schon im p-adischen Fall bemerkt wurde, hat die quadratische Iteration auch hier gegenüber der linearen Iteration sehr viel höhere Kosten pro Iterationsschritt zur Folge, so daß hinsichtlich der Gesamteffizienz die lineare Iteration als die allgemein Überlegene angesehen wird.



Homomorphie-Diagramm für das ***p*-adische und Ideal-adische Newtonverfahren.**

Beispiel:
Man bestimme das Polynom $u(x, y, z) \in \mathbb{Z}_5[x, y, z]$, das die Quadratwurzel des Polynoms

$$\begin{aligned} a(x,y,z) &= x^4 + x^3y^2 - x^2y^4 + x^2yz + 2x^2z - 2x^2 - 2xy^3z \\ &\quad + xy^2z - xy^2 - y^2z^2 + yz^2 - yz + z^2 - 2z + 1 \in \mathbb{Z}_5[x,y,z] \end{aligned}$$

ist. Dann kann $u(x)$ als Lösung der Polynomgleichung

$$f(u) = a(x, y, z) - u^2 = 0$$

ausgedrückt werden. Wählt man die Auswertungspunkte $y = 0$ und $z = 0$ (d.h. das Ideal $I = \langle y, z \rangle$), so muß die Ideal-adische Approximation erster Ordnung $u^{(1)} = u(x, 0, 0) \in \mathbb{Z}_5[x]$ eine Quadratwurzel von $a(x, 0, 0)$ in $\mathbb{Z}_5[x]$ sein. Es ist

$$a(x, 0, 0) = x^4 - 2x^2 + 1$$

das klar die Quadratwurzel

$$u^{(1)} = u(x, 0, 0) = x^2 - 1 \in \mathbb{Z}_5[x]$$

hat. Um das lineare Ideal-adische Newtonverfahren anzuwenden, beachte man, daß

$$\phi_I(f'(u^{(1)})) = \phi_I(-2u^{(1)}) = -2x^2 + 2$$

ist. Es ist bequemer $a(x, y, z)$ in seiner Ideal-adischen Darstellung bezüglich I auszudrücken:

$$\begin{aligned} a(x,y,z) &= [(x^4 - 2x^2 + 1)] + [(2x^2 - 2)z] \\ &\quad +[(x^3 - x)y^2 + (x^2 - 1)yz + (1)z^2] + [(x)y^2z + (1)yz^2] \\ &\quad +[(-x^2)y^4 + (-2x)y^3z + (-1)y^2z^2]. \end{aligned}$$

Nun ist

$$\phi_{I^2}(f(u^{(1)})) = \phi_{I^2}(a((x,y,z) - (x^2 - 1)^2 z) \in \mathbb{Z}_5[x,y,z]/I^2.$$

Der erste Korrekturterm ist

$$\Delta u^{(1)} = u_2(x)y + u_3(x)z,$$

wobei $u_2(x) = 0$ (da der entsprechende Term in $\phi_{I^2}(f(u^{(1)}))$ Null ist) und $u_3(x) = -\dfrac{c_3(x)}{(-2x^2+2)} = -\dfrac{(2x^2-2)}{(-2x^2+2)} = 1 \in \mathbb{Z}_5[x,y,z]/I^2$ ist. Deshalb gilt

$$u^{(2)} = u^{(1)} + \Delta u^{(1)} = (x^2 - 1) + z \in \mathbb{Z}_5[x,y,z]/I^2.$$

Für die nächste Iteration hat man

$$\begin{aligned} \phi_{I^3}(f(u^{(2)})) &= \phi_{I^3}(a(x,y,z) - [(x^2 - 1) + z]^2) \\ &= (x^3 - x)y^2 + (x^2 - 1)yz \in \mathbb{Z}_5[x,y,z]/I^3. \end{aligned}$$

Der neue Korrekturterm ist

$$\Delta u^{(2)} = u_{22}(x)y^2 + u_{23}(x)yz + u_{33}(x)z^2$$

mit

$$\begin{aligned} u_{33}(x) &= 0 \quad \text{(weil der entsprechende Term in } \phi_{I^3}(f(u^{(2)})) = 0 \text{ ist)} \\ u_{22}(x) &= -\frac{c_{22}(x)}{(-2x^2+2)} = -\frac{(x^3 - x)}{(-2x^2+2)} = -2x \in \mathbb{Z}_5[x] \\ u_{23}(x) &= -\frac{c_{23}(x)}{(-2x^2+2)} = -\frac{(x^2 - 1)}{(-2x^2+2)} = -2 \in \mathbb{Z}_5[x]. \end{aligned}$$

Damit ist

$$u^{(3)} = u^{(2)} + \Delta u^{(2)} = (x^2 - 1) + z + (-2x)y^2 + (-2)yz \in \mathbb{Z}_5[x,y,z]/I^3.$$

Man findet $f(u^{(3)}) = 0$ im Ring $\mathbb{Z}_5[x,y,z]$, also ist man fertig. Die gesuchte Quadratwurzel von $a(x,y,z)$ ist daher

$$u(x,y,z) = u^{(3)} = x^2 - 2xy^2 - 2yz + z - 1 \in \mathbb{Z}_5[x,y,z]. \quad \square$$

7.3 Das Lemma von Hensel

7.3.1 Bivariates Newtonverfahren

Wie bereits zu sehen war, hebt das Newtonverfahren das Bild $\phi_{\langle I,\, p\rangle}(u) \in \mathbb{Z}_p[x]$ eines Polynoms hoch zu dem gesuchten Polynom $u \in \mathbb{Z}[x_1, \ldots, x_\nu]$, d.h. es invertiert den zusammengesetzten Homomorphismus

$$\phi_{\langle I,\, p\rangle} : \mathbb{Z}[x_1, \ldots, x_\nu] \longrightarrow \mathbb{Z}_p[x], \tag{7.36}$$

vorausgesetzt u wird als Lösung einer Polynomgleichung

$$f(u) = 0 \tag{7.37}$$

ausgedrückt, für gewisse $f(u) \in \mathbb{Z}[x_1, \ldots, x_\nu][u]$.
Die meist gebräuchlichen Anwendungen des Newtonverfahrens für solch einen Lifting-Vorgang enthalten Probleme, die folgendermaßen ausgedrückt werden können:

$$F(u, w) = 0, \qquad F(u, w) \in \mathbb{Z}[x_1, \ldots, x_\nu][u, w]. \tag{7.38}$$

(7.38) hat das Lösungspaar (u, w), also muß der Lifting-Prozeß auf 2 Polynome angewandt werden.

Beispiel: Faktorisierungsproblem bei Polynomen
Sei $a(x_1, \ldots, x_\nu) \in \mathbb{Z}[x_1, \ldots, x_\nu]$. Gesucht sind seine Faktoren im gleichen Ring.
Durch Anwenden von (7.36), wird dieses Problem auf das Problem, ein univariates Polynom über $\mathbb{Z}_p$ zu faktorisieren, reduziert (ein vergleichbar einfaches Problem, wie später zu sehen sein wird).
Sei $a_0(x_1)$ ein Bild von $a(x_1, \ldots, x_\nu)$ in $\mathbb{Z}_p[x_1]$ und angenommen es sei $u_0(x_1)$ ein Faktor von $a_0(x_1)$ in $\mathbb{Z}_p[x_1]$.
In $\mathbb{Z}_p[x_1]$ gilt dann die Beziehung:

$$a_0(x_1) = u_0(x_1) w_0(x_1).$$

Daher besteht das Problem darin, multivariate Polynome $u(x_1, \ldots, x_\nu)$, $w(x_1, \ldots, x_\nu) \in \mathbb{Z}[x_1, \ldots, x_\nu]$ zu finden, die

$$F(u, w) = a(x_1, \ldots, x_\nu) - uw = 0 \tag{7.39}$$

erfüllen, so daß

$$\begin{aligned} u(x_1, \ldots, x_\nu) &\equiv u_0(x_1) \pmod{\langle I, p\rangle} \\ w(x_1, \ldots, x_\nu) &\equiv w_0(x_1) \pmod{\langle I, p\rangle} \end{aligned} \tag{7.40}$$

gilt. Mit anderen Worten: man versucht die Faktoren $u_0(x_1)$, $w_0(x_1) \in \mathbb{Z}_p[x_1]$ 'hochzuheben' auf die Faktoren u, $w \in \mathbb{Z}[x_1,\ldots,x_\nu]$ durch Anwenden einer Form des Newtonverfahrens auf die nichtlineare Gleichung (7.39). Dieser Vorgang kann so lange wiederholt werden bis alle Faktoren von $a(x_1,\ldots,x_\nu) \in \mathbb{Z}[x_1,\ldots,x_\nu]$ gefunden sind.

Beispiel: Der gcd von multivariaten Polynomen
Seien $a(x_1,\ldots,x_\nu), b(x_1,\ldots,x_\nu) \in \mathbb{Z}[x_1,\ldots,x_\nu]$. Wie vorhin genügt es, durch Anwenden von (7.36) $u_0(x) = gcd(a_0(x_1), b_0(x_1))$ im euklidischen Ring $\mathbb{Z}_p[x_1]$ zu berechnen (der euklidische Algorithmus dazu ist bereits bekannt).
Man definiert den Kofaktor $w_0(x_1) = \dfrac{a_0(x_1)}{u_0(x_1)}$ und stellt sich das Problem, die Bildpolynome $u_0(x_1)$, $w_0(x_1) \in \mathbb{Z}_p[x_1]$ auf die multivariaten Polynome u, $w \in \mathbb{Z}[x_1,\ldots,x_\nu]$ 'hochzuheben'.

In diesem Abschnitt wird besprochen, wie und unter welchen Bedingungen das Newtonverfahren zur Lösung des Problems (7.39)-(7.40) angewandt werden kann. Man beachte, (7.39) ist eine einzige Gleichung in zwei Unbekannten; man würde erwarten, daß diese ohne zusätzliche Annahmen keine eindeutige Lösung besitzt. Anstatt diese zusätzlichen Bedingungen anhand einer zweiten Gleichung der Form $G(u,w) = 0$ zu geben, kommen diese mehr indirekt in der folgenden Herleitung vor.

Die allgemeine Form des Newtonverfahrens für die bivariate Polynomgleichung

$$F(u,w) = 0$$

kann mit Hilfe des Satzes 7.4 bestimmt werden. Angenommen man hat das Approximationspaar $u^{(k)}, w^{(k)}$ zu dem Lösungspaar $\bar{u}, \bar{w}$ gegeben und man will nun dazu das Korrekturtermpaar $\Delta u^{(k)}, \Delta w^{(k)}$ berechnen. Dann führt Satz 7.4 auf die Gleichung

$$\begin{aligned} F(u^{(k)} + \Delta u^{(k)}, w^{(k)} + \Delta w^{(k)}) &= F(u^{(k)}, w^{(k)}) + F_u(u^{(k)}, w^{(k)})\Delta u^{(k)} \\ &\quad + F_w(u^{(k)}, w^{(k)})\Delta w^{(k)} + E, \end{aligned}$$

wobei der Term E alle Ausdrücke höherer Ordnung bezüglich $\Delta u^{(k)}, \Delta w^{(k)}$ enthält. Durch Argumente, die wie zuvor genau formalisiert werden können (oder etwas lockerer formuliert: man setze die linke Seite auf Null und ignoriere den Term höherer Ordnung E), erkennt man, daß die Korrekturterme so gewählt werden sollten, daß sie die folgende Gleichung (modulo einem gewissen Ideal)

$$F_u(u^{(k)}, w^{(k)})\Delta u^{(k)} + F_w(u^{(k)}, w^{(k)})\Delta w^{(k)} = -F(u^{(k)}, w^{(k)}) \tag{7.41}$$

erfüllen. Daraus sieht man, daß die eigentliche Berechnung, die bei einem Iterationsschritt im Newtonverfahren geleistet wird, das Lösen der diophantischen Polynomgleichung (7.41) ist, die folgende Form hat

$$A^{(k)}\Delta u^{(k)} + B^{(k)}\Delta w^{(k)} = C^{(k)}$$

wobei $A^{(k)}, B^{(k)}, C^{(k)}$ gegebene Polynome sind und $\Delta u^{(k)}, \Delta w^{(k)}$ unbekannte Polynome sind, die es zu bestimmen gilt. Die Gleichung (7.41) hat allgemein entweder keine Lösung oder einen ganzen Lösungsraum.

Für das spezielle Problem hier, ist die Eindeutigkeit der Lösung der diophantischen Gleichung sicherzustellen.

Diophantische Polynomgleichung:

$$\sigma(x)a(x) + \tau(x)b(x) = c(x) \tag{7.42}$$

$a, b, c \in F[x]$: gegebene Polynome über dem Körper F,
$\sigma, \tau \in F[x]$: noch zu bestimmen.
Satz 7.6 gibt genügend Bedingungen für die Existenz und Eindeutigkeit einer Lösung. (Der Beweis ist konstruktiv). Sind $a(x), b(x)$ teilerfremd, dann kann (7.42) für jede rechte Seite $c(x)$ gelöst werden.

Satz 7.6
Sei $F[x]$ ein univariater euklidischer Ring über dem Körper F, $a(x), b(x) \in F[x]$ gegebene, von Null verschiedene Polynome und $g(x) = \gcd(a(x), b(x)) \in F[x]$. Dann existieren zu jedem gegebenen Polynom $c(x) \in F[x]$ mit $g(x)|c(x)$ eindeutige Polynome $\sigma(x), \tau(x) \in F[x]$, so daß

$$\sigma(x)a(x) + \tau(x)b(x) = c(x)$$

und

$$\deg(\sigma(x)) < \deg(b(x)) - \deg(g(x))$$

gilt.
Ist überdies $\deg(c(x)) < \deg(a(x)) + \deg(b(x)) - \deg(g(x))$, dann erfüllt $\tau(x)$

$$\deg(\tau(x)) < \deg(a(x)) - \deg(g(x)).$$

7.3.2 Das Lemma von Hensel

Man betrachte die Problemlösung von (7.39)-(7.40) für den univariaten Fall. Zu einem gegebenen Polynom $a(x) \in \mathbb{Z}[x]$ und dem Faktorpaar $u_0(x), w_0(x) \in \mathbb{Z}_p[x]$ mit

$$a(x) \equiv u_0(x)w_0(x) \pmod{p}$$

will man die Faktoren $u_0(x), w_0(x)$ aus dem Bildbereich $\mathbb{Z}_p[x]$ 'hochheben' auf das Faktorpaar $u(x), w(x) \in \mathbb{Z}[x]$. Mit anderen Worten, man will den Modulo-Homomorphismus

$$\phi_p : \mathbb{Z}[x] \longrightarrow \mathbb{Z}_p[x]$$

invertieren, durch Verwenden des Newtonverfahrens zur Berechnung der Lösung $\bar{u} = u(x), \bar{w} = w(x)$ in $\mathbb{Z}[x]$ der nichtlinearen Gleichung

$$F(u,w) = a(x) - uw = 0, \tag{7.43}$$

so daß gilt

$$\begin{aligned} u(x) &\equiv u_0(x) \pmod p \\ w(x) &\equiv w_0(x) \pmod p. \end{aligned} \tag{7.44}$$

Durch Schreiben der Lösungspolynome $\bar{u}$ und $\bar{w}$ in ihrer p-adischen Polynomdarstellung

$$\begin{aligned} \bar{u} &= u_0(x) + u_1(x)p + \ldots + u_n(x)p^n \\ \bar{w} &= w_0(x) + w_1(x)p + \ldots + w_n(x)p^n \end{aligned} \tag{7.45}$$

(wobei n groß genug sein muß, daß durch $1/2p^{n+1}$ alle ganzzahligen Koeffizienten in $a(x)$ und seinen Faktoren $\bar{u}$ und $\bar{w}$ beschränkt werden) hofft man, die p-adischen Polynomkoeffizienten $u_i(x), w_i(x) \in \mathbb{Z}_p[x]$ für $i = 1, 2, \ldots, n$ zu bestimmen. Seien die p-adischen Approximationen k-ter Ordnung $u^{(k)}, w^{(k)}$ von $\bar{u}, \bar{w}$ gegeben durch die ersten k Terme in (7.45) und $\Delta u^{(k)} = u_k(x)p^k, \Delta w^{(k)} = w_k(x)p^k$. Man beachte, daß $u^{(1)} = u_0(x)$ und $w^{(1)} = w_0(x)$ gilt. Man erkennt, daß die Korrekturterme die diophantische Polynomgleichung (7.41) modulo p^{k+1} erfüllen müssen, welche für die nichtlineare Gleichung (7.43) die Form

$$-w^{(k)}\Delta u^{(k)} - u^{(k)}\Delta w^{(k)} \equiv -\left(a(x) - u^{(k)}w^{(k)}\right) \pmod{p^{k+1}}$$

hat. Da $u^{(k)}w^{(k)}$ eine p-adische Approximation k-ter Ordnung von $a(x)$ sein muß, kann man durch p^k teilen. Werden noch zusätzlich die negativen Vorzeichen entfernt, erhält man

$$w^{(k)}u_k(x) + u^{(k)}w_k(x) \equiv \frac{a(x) - u^{(k)}w^{(k)}}{p^k} \pmod p.$$

Nun kann der Modulo-Homomorphismus ϕ_p angewandt werden (weil es sich um eine Kongruenz modulo p handelt) und man erhält unter Beachtung, daß $\phi_p(w^{(k)}) = w_0(x)$ und $\phi_p(u^{(k)}) = u_0(x)$ ist, die diophantische Polynomgleichung

$$w_0(x)u_k(x) + u_0(x)w_k(x) \equiv \phi_p\left(\frac{a(x) - u^{(k)}w^{(k)}}{p^k}\right),$$

die im Ring $\mathbb{Z}_p[x]$ zu lösen ist. Da $\mathbb{Z}_p[x]$ ein euklidischer Ring ist (p sei eine Primzahl), kann man mit Satz 7.6, falls $u_0(x), w_0(x) \in \mathbb{Z}_p[x]$ teilerfremd sind, eindeutige Polynome $\sigma(x), \tau(x) \in \mathbb{Z}_p[x]$ finden, so daß

$$\sigma(x)u_0(x) + \tau(x)w_0(x) = \phi_p\left(\frac{a(x) - u^{(k)}w^{(k)}}{p^k}\right)$$

und

$$\deg(\sigma(x)) < \deg(w_0(x))$$

gilt. Man definiert dann $u^{(k+1)} = u^{(k)} + \tau(x)p^k, w^{(k+1)} = w^{(k)} + \sigma(x)p^k$ und behauptet, daß dies die p-adischen Approximationen der Ordnung $k+1$ für die Lösungen $\bar{u}, \bar{w}$ sind.

Der folgende Satz beweist formal die Richtigkeit der obigen Methode. Dieser Satz, ein Standardergebnis der Algebra, ist um 1900 entstanden und als Lemma von Hensel bekannt geworden. Der Beweis dieses Lemmas ist konstruktiv, was aus der obigen Herleitung folgt. Dieser Prozeß heißt *Hensel-Konstruktion*.

Satz 7.7 (Lemma von Hensel)
Sei $p \in \mathbb{Z}$ eine Primzahl und $a(x) \in \mathbb{Z}[x]$ ein gegebenes Polynom über den ganzen Zahlen. Weiter seien $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ zwei teilerfremde Polynome über dem Körper $\mathbb{Z}_p$, so daß

$$a(x) \equiv u^{(1)}(x)w^{(1)}(x) \pmod{p}$$

gilt. Dann existieren für jede ganze Zahl $k \geq 1$ Polynome $u^{(k)}(x), w^{(k)}(x) \in Z_{p^k}[x]$, so daß

$$a(x) \equiv u^{(k)}(x)w^{(k)}(x) \pmod{p^k} \tag{7.46}$$

und

$$\begin{aligned} u^{(k)}(x) &\equiv u^{(1)}(x) \pmod{p} \\ w^{(k)}(x) &\equiv w^{(1)}(x) \pmod{p} \end{aligned} \tag{7.47}$$

gilt.

Beweis:
Induktion über k. Der Fall $k = 1$ ist gegeben. Für $k \geq 1$ nehme man an, man habe $u^{(k)}(x), w^{(k)}(x) \in \mathbb{Z}_{p^k}[x]$, die (7.46) und (7.47) erfüllen. Man definiert

$$c^{(k)}(x) = \phi_p\left(\frac{a(x) - u^{(k)}(x)w^{(k)}(x)}{p^k}\right), \tag{7.48}$$

wobei vor Anwenden von ϕ_p alle Operationen im Ring $\mathbb{Z}_{p^{k+1}}[x]$ durchgeführt werden. Da $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ teilerfremd sind, kann man durch Satz 7.6 eindeutige Polynome $\sigma^{(k)}(x), \tau^{(k)}(x) \in \mathbb{Z}_p[x]$ bestimmen mit

$$\sigma^{(k)}(x)u^{(1)}(x) + \tau^{(k)}(x)w^{(1)}(x) \equiv c^{(k)}(x) \pmod{p} \tag{7.49}$$

und

$$\deg(\sigma^{(k)}(x)) < \deg(w^{(1)}(x)). \tag{7.50}$$

Definiert man

$$\begin{aligned} u^{(k+1)}(x) &= u^{(k)}(x) + \tau^{(k)}(x)p^k \\ w^{(k+1)}(x) &= w^{(k)}(x) + \sigma^{(k)}(x)p^k, \end{aligned} \tag{7.51}$$

erhält man durch Multiplikation modulo p^{k+1}

$$\begin{aligned}
u^{(k+1)}(x)w^{(k+1)}(x) &\equiv u^{(k)}(x)w^{(k)}(x) + (\sigma^{(k)}(x)u^{(1)}(x) \\
&\quad +\tau^{(k)}(x)w^{(1)}(x))p^k \pmod{p^{k+1}} \\
&\equiv u^{(k)}(x)w^{(k)}(x) + c^{(k)}(x)p^k \pmod{p^{k+1}}, \\
&\quad \text{durch (7.49)} \\
&\equiv a(x) \pmod{p^{k+1}}, \text{ durch (7.48).}
\end{aligned}$$

Damit gilt (7.46) für $k+1$. Mit (7.51) ist auch klar, daß

$$\begin{aligned}
u^{(k+1)}(x) &\equiv u^{(k)}(x) \pmod{p} \\
w^{(k+1)}(x) &\equiv w^{(k)}(x) \pmod{p}
\end{aligned}$$

gilt und weil (7.47) für k gilt, gilt es ebenso für $k+1$. □

Korollar 7.8 (Eindeutigkeit der Hensel-Konstruktion)
Ist das gegebene Polynom $a(x) \in \mathbb{Z}[x]$ im Satz 7.7 normiert und sind die entsprechenden teilerfremden Faktoren $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ ebenfalls normiert gewählt, so bestimmen die Bedingungen (7.46)-(7.47) für jede ganze Zahl $k \geq 1$ eindeutig die normierten Polynomfaktoren $u^{(k)}(x), w^{(k)}(x) \in Z_{p^k}[x]$.

Der Beweis ist eine Induktion über k.

7.4 Der univariate Hensel-lifting Algorithmus

7.4.1 Beschreibung des Algorithmus

Die Hensel-Konstruktion von Satz 7.7 basiert auf dem linearen p-adischen Newtonverfahren. Zassenhaus war der erste, der vorschlug das Lemma von Hensel auf das Faktorisierungsproblem von Polynomen über den ganzen Zahlen anzuwenden und dabei, das quadratische p-adische Newtonverfahren zu benutzen. Diese quadratische Iteration heißt daher gewöhnlich auch *Zassenhaus-Konstruktion* und berechnet eine Folge von Faktoren modulo p^{2^k} für $k = 1, 2, 3, \ldots$. Jedoch ist eine quadratische Iteration nicht notwendig effizienter als eine lineare Iteration, weil die zusätzliche Komplexität für jeden Iterationsschritt der quadratischen Iteration den Vorteil von weniger Iterationsschritten übersteigt. Man hat z.B. in jedem Iterationsschritt der quadratischen Zassenhaus-Konstruktion, die diophantische Polynomgleichung der Form

$$\sigma^{(k)}(x)u^{(k)}(x) + \tau^{(k)}(x)w^{(k)}(x) \equiv c^{(k)}(x) \pmod{p^{2^{k-1}}} \tag{7.52}$$

für $\sigma^{(k)}(x), \tau^{(k)}(x) \in \mathbb{Z}_{p^{2^{k-1}}}[x]$ zu lösen. Die entsprechende Berechnung in der linearen Hensel-Konstruktion ist, das Lösen der gleichen diophantischen Polynomgleichung modulo p für $\sigma^{(k)}(x), \tau^{(k)}(x) \in \mathbb{Z}_p[x]$. Letztere Berechnung ist einfacher, weil sie in dem kleineren Ring $\mathbb{Z}_p[x]$ durchgeführt wird und es wird noch effizienter, da $u^{(k)}(x)$ und $w^{(k)}(x)$ in (7.52) durch die festen Polynome $u^{(1)}(x)$ und $w^{(1)}(x)$ im linearen Hensel Fall ersetzt werden können. Ein detailierter Vergleich dieser zwei p-adischen Konstruktionen wurde von Miola und Yun ausgearbeitet. Ihre Analyse zeigt, daß die Kosten zur Berechnung der gleichen p-adischen Approximationsordnung für die quadratische Zassenhaus-Konstruktion höher sind als die der linearen Hensel-Konstruktion. Daher werden hier die Details für die quadratische Zassenhaus-Konstruktion nicht näher aufgezeigt.

Algorithmus 7.1 Univariater Hensel-lifting Algorithmus.

begin

comment INPUT:

1. Primitives Polynom $a(x) \in \mathbb{Z}[x]$.
2. Primzahl p, die $\mathrm{Lc}[a(x)]$ nicht teilt.
3. Zwei teilerfremde Polynome $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$, so daß

$$a(x) \equiv u^{(1)}(x)w^{(1)}(x) \pmod p$$

   gilt.
4. Eine ganze Zahl B, die eine Schranke darstellt für alle ganzzahligen Koeffizienten in $a(x)$ und allen seinen möglichen Faktoren mit einem Grad von höchstens

$$\max(\deg[u^{(1)}(x)], \deg[w^{(1)}(x)]).$$

5. Optional: eine ganze Zahl $\gamma \in \mathbb{Z}$, die als ein Vielfaches von $\mathrm{Lc}[u(x)]$ bekannt ist, wobei $u(x)$ (siehe OUTPUT) einer der zu berechnenden Faktoren von $a(x)$ in $\mathbb{Z}[x]$ ist.

comment OUTPUT:

1. Falls Polynome $u(x), w(x) \in \mathbb{Z}[x]$ existieren, so daß

$$a(x) = u(x)w(x) \in \mathbb{Z}[x]$$

und

$$\begin{aligned} \boldsymbol{n}[u(x)] &\equiv \boldsymbol{n}[u^{(1)}(x)] \pmod{p}, \\ \boldsymbol{n}[w(x)] &\equiv \boldsymbol{n}[w^{(1)}(x)] \pmod{p} \end{aligned}$$

gilt, wobei $\boldsymbol{n}$ die Normalisierung 'normiere das Polynom als ein Element im Ring $\mathbb{Z}_p[x]$' bezeichnet, dann wird $u(x)$ und $w(x)$ berechnet.

2. Andernfalls wird die Nichtexistenz einer solchen Faktorisierung festgestellt.

comment 1. Definiere ein neues Polynom und seine modulo p Faktoren.

$\alpha \leftarrow$Lc$[a(x)]$;
if γ undefiniert **then** $\gamma \leftarrow \alpha$;
$a(x) \leftarrow \gamma \times a(x)$;
$u^{(1)}(x) \leftarrow \phi_p(\gamma \times \boldsymbol{n}[u^{(1)}(x)])$; $w^{(1)}(x) \leftarrow \phi_p(\alpha \times \boldsymbol{n}[w^{(1)}(x)])$;

comment 2. Wende den erweiterten euklidischen Algorithmus auf $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ an.

$s(x), t(x) \leftarrow$ Polynome aus $\mathbb{Z}_p[x]$ berechnet mit dem EEA, so daß gilt

$$s(x)u^{(1)}(x) + t(x)w^{(1)}(x) \equiv 1 \pmod{p};$$

comment 3. Initialisierung der Iteration.

$u(x) \leftarrow$ replaceLc$(u^{(1)}(x), \gamma)$; $w(x) \leftarrow$ replaceLc$(w^{(1)}(x), \alpha)$;
$e(x) \leftarrow a(x) - u(x) \times w(x)$;
$modulus \leftarrow p$;

comment 4. Iteriere bis entweder die Faktorisierung in $\mathbb{Z}[x]$ bekannt oder die Schranke $modulus$ erreicht ist.

while $e(x) \neq 0$ **and** $modulus < 2 \times B \times \gamma$ **do**
begin

comment 4.1. Löse im Ring $\mathbb{Z}_p[x]$ die diophantische Polynomgleichung

$$\sigma(x)u^{(1)}(x) + \tau(x)w^{(1)}(x) \equiv c(x) \pmod{p}$$

mit $c(x) = e(x)/modulus$.

$c(x) \leftarrow e(x)/modulus;$
$\tilde{\sigma}(x) \leftarrow \phi_p(s(x) \times c(x));\ \tilde{\tau}(x) \leftarrow \phi_p(t(x) \times c(x));$
$q(x), r(x) \leftarrow$ Polynome aus $\mathbb{Z}_p[x]$ mit der euklidischen
Division berechnet, so daß gilt

$$\tilde{\sigma}(x) = w^{(1)}(x)q(x) + r(x) \in \mathbb{Z}_p[x];$$

$\sigma(x) \leftarrow r(x);\ \tau(x) \leftarrow \phi_p(\tilde{\tau}(x) + q(x) \times u^{(1)}(x));$

comment 4.2 Aktualisiere die Faktoren und den Fehler in der neuen Faktorisierung.

$u(x) \leftarrow u(x) + \tau(x) \times modulus;$
$w(x) \leftarrow w(x) + \sigma(x) \times modulus;$
$e(x) \leftarrow a(x) - u(x) \times w(x);$
$modulus \leftarrow modulus \times p$
end;

comment 5. Prüfe Abbruchbedingung.

if $e(x) = 0$ **then**
begin comment Faktorisierung erhalten – entferne Inhalt.
$\delta \leftarrow cont[u(x)];$
$u(x) \leftarrow u(x)/\delta;\ w(x) \leftarrow w(x)/(\gamma/\delta);$
comment Beachte, daß $a(x) \leftarrow a(x)/\gamma$
$a(x)$ auf seinen Eingangswert zurücksetzt.
end
else comment es existiert keine solche Faktorisierung;
Abbruchstatus ist 'es existiert keine solche Faktorisierung'
end.

Der grundlegende Algorithmus zum 'Anheben' einer Faktorisierung in $\mathbb{Z}_p[x]$ auf eine Faktorisierung in $\mathbb{Z}[x]$ ist im Algorithmus 7.1 dargestellt. Im normierten Fall entpricht der Algorithmus 7.1 genau der Hensel-Konstruktion aus dem Beweis zum Lemma von Hensel, da nach dem Korollar zu Satz 7.7 im normierten Fall die Faktoren bei jedem Schritt im Lifting-Vorgang eindeutig bestimmt sind. Im nichtnormierten Fall führt die Nichteindeutigkeit der Faktoren modulo p^k zum 'Leitkoeffizienten-Problem', das kurz diskutiert wird und wie noch zu sehen sein wird, ist dies für die zusätzlichen Bedingungen und Operationen im Algorithmus 7.1 verantwortlich. Für jetzt, kann der Algorithmus 7.1 für den normierten Fall verstanden werden, indem einfach die gemachten Bedingungen (außer den im Lemma von Hensel erwähnten) ignoriert werden, ebenso Schritt 1 und die 'replaceLc' Operation in Schritt 3 (man benutze dafür die Initialisierung $u(x) \leftarrow u^{(1)}(x)$; $w(x) \leftarrow w^{(1)}(x)$) und man beachte, daß keine Korrektur für $u(x)$ und $w(x)$ in Schritt 5 für den normierten Fall notwendig ist.

Beispiel 1:
Man faktorisiere das normierte Polynom

$$a(x) = x^3 + 10x^2 - 432x + 5040 \in \mathbb{Z}[x]$$

über den ganzen Zahlen. Die Wahl von $p = 5$ und das Anwenden des Modulo-Homomorphismus ϕ_5 auf $a(x)$ führt auf

$$\phi_5(a(x)) = x^3 - 2x \in \mathbb{Z}_5[x].$$

Die eindeutige einheitsnormale (d.h. normierte) Faktorisierung in $\mathbb{Z}_5[x]$ dieses Polynoms ist

$$\phi_5(a(x)) = x(x^2 - 2) \in \mathbb{Z}_5[x].$$

Daher definiert man

$$u^{(1)}(x) = x;\ w^{(1)}(x) = x^2 - 2$$

und da $u^{(1)}(x)$ und $w^{(1)}(x)$ teilerfremd sind in $\mathbb{Z}_5[x]$ kann die Hensel-Konstruktion angewandt werden. Wird der Algorithmus 7.1 in der Form für den normierten Fall ausgeführt, führt zuerst in Schritt 2 der erweiterte euklidische Algorithmus auf

$$s(x) = -2x;\ t(x) = 2.$$

Die Initialisierung in Schritt 3 ergibt

$$\begin{aligned} u(x) &= x;\ w(x) = x^2 - 2; \\ e(x) &= 10x^2 - 430x + 5040; \\ modulus &= 5. \end{aligned}$$

Schritt 4 ist dann genau die Hensel-Konstruktion wie im Beweis zum Lemma von Hensel angegeben. (Hier wird die zweite Abbruchbedingung der **while**-Schleife ignoriert). Es ergibt sich die folgende Folge von Werten für $\sigma(x)$, $\tau(x)$, $u(x)$, $w(x)$ und $e(x)$:

nach Iteration	$\sigma(x)$	$\tau(x)$	$u(x)$	$w(x)$	$e(x)$
0	—	—	x	$x^2 - 2$	$10x^2 - 430x + 5040$
1	$x - 1$	1	$x + 5$	$x^2 + 5x - 7$	$-450x + 5075$
2	$-x + 2$	1	$x + 30$	$x^2 - 20x + 43$	$125x + 3750$
3	1	0	$x + 30$	$x^2 - 20x + 168$	0

Man beachte, daß nach Ende jedes Iterationsschrittes k $e(x)$ exakt durch $modulus = 5^{k+1}$ teilbar ist, wie zu Beginn des nächsten Schrittes gefordert ist. Die Iteration terminiert mit $u(x) = x + 30$ und $w(x) = x^2 - 20x + 168$ und man erhält daher die folgende Faktorisierung über den ganzen Zahlen:

$$a(x) = x^3 + 10x^2 - 432x + 5040 = (x + 30)(x^2 - 20x + 168). \quad \square$$

Beispiel 2:
In diesem Beispiel wird sichtbar, daß die Hensel-Konstruktion sogar dann angewandt werden kann, wenn das Polynom über den ganzen Zahlen gar nicht faktorisiert werden kann. Man betrachte das normierte Polynom

$$a(x) = x^4 + 1 \in \mathbb{Z}[x],$$

das irreduzibel über den ganzen Zahlen ist. Die Wahl von $p = 5$ und das Anwenden des Modulo-Homomorphismus ϕ_5 auf $a(x)$ führt auf $\phi_5(a(x)) = x^4 + 1$. Die eindeutige einheitsnormale Faktorisierung in $\mathbb{Z}_5[x]$ dieses Polynoms ist

$$x^4 + 1 = (x^2 + 2)(x^2 - 2) \in \mathbb{Z}_5[x].$$

Da die Polynome $u^{(1)}(x) = x^2 + 2$ und $w^{(1)}(x) = x^2 - 2$ teilerfremd in $\mathbb{Z}_5[x]$ sind, kann die Hensel-Konstruktion angewandt werden. In diesem Fall erhält man die unendliche Folge von Faktoren

$$a(x) \equiv u^{(k)}(x)w^{(k)}(x) \pmod{p^k}$$

für $k = 1, 2, 3, \ldots$.
Wird Algorithmus 7.1 auf diesen normierten Fall wie in Beispiel 1 angewandt, ist das Ergebnis von Schritt 2

$$s(x) = -1; \; t(x) = 1$$

und die Initialisierung von Schritt 3 ergibt

$$\begin{aligned} u(x) &= x^2 + 2; \; w(x) = x^2 - 2; \\ e(x) &= 5; \\ modulus &= 5. \end{aligned}$$

Läßt man zu, die **while**-Schleife in Schritt 4 viermal zu durchlaufen (wieder wird die zweite Abbruchbedingung der **while**-Schleife ignoriert), ergeben sich folgende Werte für $\sigma(x)$, $\tau(x)$, $u(x)$, $w(x)$ und $e(x)$:

nach Iteration	$\sigma(x)$	$\tau(x)$	$u(x)$	$w(x)$	$e(x)$
0	—	—	x^2+2	x^2-2	5
1	-1	1	x^2+7	x^2-7	50
2	-2	2	x^2+57	x^2-57	3250
3	-1	1	x^2+182	x^2-182	33125
4	2	-2	x^2-1068	x^2+1068	1140625

Diese Iteration könnte unendlich fortgesetzt werden und eine unendliche Folge von Faktoren liefern, die das Lemma von Hensel erfüllen. Man beachte, daß nach dem k-ten Iterationsschritt immer

$$u(x)w(x) \equiv x^4+1 \pmod{5^{k+1}}$$

gilt, wie es im Lemma von Hensel gefordert ist. Jedoch wird man nie quadratische Faktoren $u(x), w(x) \in \mathbb{Z}[x]$ erhalten, so daß gilt

$$u(x)w(x) = x^4+1 \in \mathbb{Z}[x]. \quad \Box$$

7.4.2 Das Leitkoeffizienten-Problem

Die Hensel-Konstruktion liefert eine Methode, eine Faktorisierung modulo p auf die Faktorisierung modulo p^l für jede ganze Zahl $l \geq 1$ 'hochzuheben'. Beispiel 2 zeigt, daß diese Konstruktion nicht notwendig auf eine Faktorisierung über den ganzen Zahlen führt. Hat jedoch das normierte Polynom die modulo p Faktorisierung

$$a(x) \equiv u^{(1)}(x)w^{(1)}(x) \pmod{p},$$

wobei $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ teilerfremde normierte Polynome sind und existiert eine Faktorisierung

$$a(x) = u(x)w(x) \in \mathbb{Z}[x] \tag{7.53}$$

über den ganzen Zahlen, so daß

$$u(x) \equiv u^{(1)}(x) \pmod{p}; \;\; w(x) \equiv w^{(1)}(x) \pmod{p} \tag{7.54}$$

gilt, dann muß die Hensel-Konstruktion diese Faktorisierung liefern. Im besonderen sei l groß genug, daß $\frac{1}{2}p^l > B$ ist, wobei B eine ganze Zahl ist, die die Größe aller ganzzahligen Koeffizienten von $a(x)$ und allen seinen möglichen Faktoren mit den speziellen Graden $\deg[u^{(1)}(x)]$ und $\deg[w^{(1)}(x)]$ beschränkt. (Techniken zur Berechnung einer solchen Schranke B sind in [Knu69] beschrieben). Dann erfüllt die Hensel-Konstruktion angewandt zur Berechnung der normierten Polynome $u^{(l)}(x), w^{(l)}(x) \in \mathbb{Z}_{p^l}[x]$

$$a(x) \equiv u^{(l)}(x)w^{(l)}(x) \pmod{p^l} \tag{7.55}$$

und

$$\begin{aligned} u^{(l)}(x) &\equiv u^{(1)}(x) \pmod p \\ w^{(l)}(x) &\equiv w^{(1)}(x) \pmod p. \end{aligned} \tag{7.56}$$

Mit dem Korollar zu Satz 7.7 sind die Faktoren $u^{(l)}(x), w^{(l)}(x) \in \mathbb{Z}_{p^l}[x]$ eindeutig bestimmt durch die Bedingungen (7.55)-(7.56). Existiert nun eine Faktorisierung (7.53), die (7.54) erfüllt, so wird eine andere solche normierte Faktorisierung in $\mathbb{Z}_{p^l}[x]$ durch $\phi_{p^l}(u(x))$ und $\phi_{p^l}(w(x))$ geliefert und aufgrund der Eindeutigkeit gilt

$$u^{(l)}(x) = \phi_{p^l}(u(x)); \;\; w^{(l)}(x) = \phi_{p^l}(w(x)).$$

Da aber $\frac{1}{2}p^l > B$ ist, gilt $\phi_{p^l}(u(x)) = u(x)$ und $\phi_{p^l}(w(x)) = w(x)$, was beweist, daß $u^{(l)}(x)$ und $w^{(l)}(x)$ die gesuchten Faktoren über den ganzen Zahlen sind.

Die obige Diskussion zeigt, daß im normierten Fall die Hensel-Konstruktion anhält, falls $\frac{1}{2}p^l > B$ ist. An diesem Punkt gilt entweder $u^{(l)}(x)w^{(l)}(x) = a(x)$ über den ganzen Zahlen oder es existiert keine Faktorisierung die (7.53)-(7.54) erfüllt. Die zweite Abbruchbedingung der **while**-Schleife im Schritt 4 des Algorithmus 7.1 ist im normierten Fall genau diese Bedingung. Man beachte, daß die Schranke B, die im Algorithmus 7.1 gegeben ist, unverändert sehr pessimistisch bleibt (d.h. sie ist größer als nötig), so daß die erste Abbruchbedingung der **while**-Schleife benötigt wird, um die Kosten für zuviele Iterationsschritte zu vermeiden, nachdem die Faktorisierung gefunden wurde.

Im nicht-normierten Fall ist die Situation nicht ganz so einfach. Die Hensel-Konstruktion verlangt (in Schritt 4.1 von Algorithmus 7.1) die Lösung $\sigma(x), \tau(x) \in \mathbb{Z}_p[x]$ der diophantischen Polynomgleichung

$$\sigma(x)u^{(1)}(x) + \tau(x)w^{(1)}(x) \equiv c(x) \pmod p. \tag{7.57}$$

Die Lösung dieser Gleichung ist nicht eindeutig bestimmt, aber die Eindeutigkeit wird (etwas künstlich) auferlegt durch die Forderung, daß die Lösung

$$\deg[\sigma(x)] < \deg[w^{(1)}(x)] \tag{7.58}$$

erfüllen muß (siehe Satz 7.6). Man beachte, daß die Aktualisierungsformeln (in Schritt 4.2 von Algorithmus 7.1)

$$u(x) \leftarrow u(x) + \tau(x) \times modulus; \; w(x) \leftarrow w(x) + \sigma(x) \times modulus$$

sind und es ist klar, daß der in (7.58) erzwungene Grad impliziert, daß der Leitkoeffizient von $w(x)$ nie aktualisiert wird. Im normierten Fall ist das gerade das, was gewünscht wird. Mehr noch, da

$$c(x) = \frac{a(x) - u(x)w(x)}{modulus} \tag{7.59}$$

ist, folgt im normierten Fall, daß

$$\deg[c(x)] < \deg[a(x)] = \deg[u^{(1)}(x)] + \deg[w^{(1)}(x)]$$

und deshalb erfüllt die Lösung von (7.57) mit dem Satz 7.6 auch

$$\deg[\tau(x)] < \deg[u^{(1)}(x)],$$

woraus folgt, daß der Leitkoeffizient von $u(x)$ im normierten Fall ebenfalls nie aktualisiert wird. Doch zurück zum nichtnormierten Fall. Als erstes muß angenommen werden, daß die gewählte Primzahl p den Leitkoeffizienten von $a(x)$ nicht teilt. Mit dieser Annahme stellt man sicher, daß $u^{(1)}(x)$ und $w^{(1)}(x)$ die 'korrekten' Grade im Sinne, daß $\deg[u^{(1)}(x)] + \deg[w^{(1)}(x)] = \deg[a(x)]$ gilt, haben. Damit erhält man mit (7.59), daß

$$\deg[c(x)] \leq \deg[a(x)] = \deg[u^{(1)}(x)] + \deg[w^{(1)}(x)]$$

woraus genau wie im Satz 7.6 folgt, daß

$$\deg[\tau(x)] \leq \deg[u^{(1)}(x)] \tag{7.60}$$

gilt. Der in (7.60) erzwungene Grad, erlaubt den Leitkoeffizienten von $u(x)$ zu aktualisieren, da, anders wie in (7.58), die Ungleichung hier nicht streng ist. Jetzt gilt am Ende jedes k-ten Iterationsschrittes die Beziehung

$$a(x) \equiv u(x)w(x) \pmod{p^{k+1}}$$

und weil durch (7.58) der Leitkoeffizient von $w(x)$ gezwungenermaßen unverändert bleibt, werden alle Aktualisierungen, die durch den Leitkoeffizienten von $a(x)$ gefordert werden, durch den Leitkoeffizienten von $u(x)$ erzwungen. (Man beachte, jede Einheit aus dem Ring $\mathbb{Z}_{p^k}[x]$ kann multipliziert werden in den einen Faktor und sein Inverses in den anderen Faktor, ohne die gegebene Beziehung zu verändern). Dies wird auch als *Leitkoeffizienten-Problem* bezeichnet und kann dafür verantwortlich sein, daß die Faktoren in der Hensel-Konstruktion nie auf eine Faktorisierung über den ganzen Zahlen führen, auch wenn eine solche Faktorisierung über den ganzen Zahlen existiert. Das folgende Beispiel soll Klarheit in das Leitkoeffizienten-Problem bringen.

Beispiel 3:
Man faktorisiere das folgende Polynom über den ganzen Zahlen:

$$a(x) = 12x^3 + 10x^2 - 36x + 35 \in \mathbb{Z}[x].$$

Um das Leitkoeffizientenproblem, das hier auftritt, richtig zu verstehen, ist hier zu Beginn die korrekte Antwort auf das Problem. Die vollständige einheitsnormale Faktorisierung von $a(x)$ über den ganzen Zahlen ist

$$a(x) = u(x)w(x) = (2x + 5)(6x^2 - 10x + 7) \in \mathbb{Z}[x].$$

Man versucht nun dieses Faktorisierungsproblem mit der Methode aus Beispiel 2 zu lösen. Die Wahl von $p = 5$ und das Anwenden des Modulo-Homomorphismus ϕ_5 auf $a(x)$ führt auf

$$\phi_5(a(x)) = 2x^3 - x \in \mathbb{Z}_5[x].$$

Die eindeutige einheitsnormale Faktorisierung in $\mathbb{Z}_5[x]$ dieses Polynoms ist

$$\phi_5(a(x)) = 2(x)(x^2 + 2) \in \mathbb{Z}_5[x],$$

wobei 2 eine Einheit in $\mathbb{Z}_5[x]$ ist. Um nun die initialen Faktoren $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_5[x]$ zum 'Hochheben' auszuwählen, muß die Einheit 2 entweder an den Faktor x oder an den Faktor $x^2 + 2$ angebunden werden. Dies ist genau das Problem der Nichteindeutigkeit, das bei jedem Schritt der Hensel-Konstruktion auftritt; bei diesem Initialschritt erhält man

$$\phi_5(a(x)) = (2x)(x^2 + 2) = (x)(2x^2 - 1) \in \mathbb{Z}_5[x].$$

Da zu diesem Problem die Antwort bereits gegeben ist, kann man sehen, daß die 'korrekten' Bilder mit ϕ_5 von $u(x), w(x) \in \mathbb{Z}[x]$

$$u^{(1)}(x) = 2x;\; w^{(1)}(x) = x^2 + 2$$

sind. Es ist jedoch wichtig zu beachten, daß diese 'korrekte' Anbindung von Einheiten an die Faktoren irrelevant ist; die andere Wahl für $u^{(1)}(x)$ und $w^{(1)}(x)$ in diesem Beispiel wäre genauso gültig und würde auf das gleiche 'Leitkoeffizienten-Problem' führen, das sich aus der obigen Wahl ergibt.
Die Polynome $u^{(1)}(x)$ und $w^{(1)}(x)$, die oben definiert worden sind, sind teilerfremd in $\mathbb{Z}_5[x]$, so daß die Hensel-Konstruktion angewandt werden kann. Der Algorithmus 7.1 wird nun in der Form wie für den normierten Fall von Beispiel 2 angewandt (d.h. die unveränderte Hensel-Konstruktion wie im Beweis zum Lemma von Hensel). In Schritt 2 des Algorithmus 7.1 führt der erweiterte euklidische Algorithmus, angewandt auf $u^{(1)}(x)$ und $w^{(1)}(x)$, auf

$$s(x) = x;\; t(x) = -2.$$

Die Initialisierung von Schritt 3 ergibt

$$\begin{aligned} u(x) &= 2x;\; w(x) = x^2 + 2; \\ e(x) &= 10x^3 + 10x^2 - 40x + 35; \\ modulus &= 5. \end{aligned}$$

Läßt man zu, die **while**-Schleife in Schritt 4 viermal zu durchlaufen (wieder wird die zweite Abbruchbedingung der **while**-Schleife ignoriert), ergeben sich folgende Werte für $\sigma(x)$, $\tau(x)$, $u(x)$, $w(x)$ und $e(x)$:

nach Iteration	$\sigma(x)$	$\tau(x)$	$u(x)$	$w(x)$	$e(x)$
0	—	—	$2x$	x^2+2	$10x^3+10x^2-40x+35$
1	$-2x-1$	$2x+1$	$12x+5$	$x^2-10x-3$	$125x^2+50x+50$
2	$2x+1$	1	$12x+30$	$x^2+40x+22$	$-500x^2-1500x-625$
3	$-2x-1$	0	$12x+30$	$x^2-210x-103$	$2500x^2+7500x+3125$
4	$2x+1$	0	$12x+30$	$x^2+1040x+522$	$-12500x^2-37500x-15625$

Diese Iteration könnte unendlich fortgesetzt werden und eine unendliche Folge von Faktoren liefern, die das Lemma von Hensel erfüllen — d.h. nach dem k-ten Iterationsschritt gilt immer

$$u(x)v(x) \equiv a(x) \pmod{5^{k+1}}.$$

Jedoch erfüllen diese Faktoren nie die gewünschte Beziehung

$$u(x)w(x) = a(x) \in \mathbb{Z}[x],$$

weil $w(x)$ immer normiert ist und es keinen normierten quadratischen Faktor von $a(x)$ über den ganzen Zahlen gibt. □

Im obigen Beispiel ist klar, daß der Leitkoeffizient von $a(x)$ vollständig vom Faktor $u(x)$ erzwungen wird, da $w^{(1)}(x)$ und damit alle aktualisierten $w(x)$ normiert sind. Beachtet man die korrekte Faktorisierung von $a(x)$ über den ganzen Zahlen, sieht man, daß der Leitkoeffizient von $a(x)$ aufgespaltet werden muß in der Form $12 = 2 \times 6$ mit dem Faktor 2 als Leitkoeffizient von $u(x)$ und dem Faktor 6 als Leitkoeffizient von $w(x)$. Der Algorithmus 7.1 enthält zusätzliche Anweisungen, die die Korrektheit des Leitkoeffizienten erzwingen. Der nächste Abschnitt ist der Erklärung dieser zusätzlichen Operationen gewidmet.

7.5 Spezielle Techniken für den nichtnormierten Fall

7.5.1 Beziehung zwischen berechneten und echten Faktoren

Der erste Schritt zur Lösung des Leitkoeffizienten-Problems ist das Erkennen, daß die mit der Hensel-Konstruktion berechneten Faktoren in folgendem Sinne 'meistens' die korrekten Faktoren über den ganzen Zahlen sind. Sei l groß genug, daß $\frac{1}{2}p^l > B$

ist, wobei B die Größe aller ganzzahligen Koeffizienten von $a(x)$ und seinen Faktoren beschränkt. Dann beweist der weiter unten stehende Satz 7.10, daß die mit der Hensel-Konstruktion berechneten Faktoren $u^{(l)}(x)$ und $w^{(l)}(x)$ mit

$$u^{(l)}(x)w^{(l)}(x) \equiv a(x) \pmod{p^l},$$

sich von den echten Faktoren über den ganzen Zahlen nur durch eine Einheit im Ring $\mathbb{Z}_{p^l}[x]$ unterscheiden (sofern eine entsprechende Faktorisierung über den ganzen Zahlen existiert). In Beispiel 3 im vorhergehenden Abschnitt war durch die Untersuchung von $a(x)$ und seinen bekannten Faktoren zu sehen, daß $B = 36$, also $l = 3$ ausreicht, daß die Faktoren

$$u^{(3)}(x) = 12x + 30; \;\; w^{(3)}(x) = x^2 + 40x + 22,$$

die im Iterationsschritt $k = 2$ berechnet wurden, bis auf Einheiten im Ring $\mathbb{Z}_{125}[x]$ die korrekten Faktoren sein müssen. Man beachte, daß

$$u^{(3)}(x)w^{(3)}(x) = 12x^3 + 510x^2 + 1464x + 660 \in \mathbb{Z}[x],$$

so daß $u^{(3)}(x)w^{(3)}(x) \neq a(x)$, aber

$$u^{(3)}(x)w^{(3)}(x) \equiv a(x) \pmod{5^3}.$$

In diesem Beispiel ist bekannt, daß der korrekte Leitkoeffizient von $w(x)$ 6 ist. Daher wird $w^{(3)}(x)$ im Ring $\mathbb{Z}_{125}[x]$ mit 6 multipliziert und entsprechend $u^{(3)}(x)$ mit $6^{-1} \in \mathbb{Z}_{125}[x]$ multipliziert, so daß die Beziehung

$$[6^{-1}u^{(3)}][6w^{(3)}(x)] \equiv a(x) \pmod{5^3}$$

erhalten bleibt. Da $6^{-1} = 21 \in \mathbb{Z}_{125}[x]$ ist, erhält man die Faktoren

$$\begin{aligned} u(x) &= 21u^{(3)}(x) = 2x + 5 \in \mathbb{Z}_{125}[x] \\ w(x) &= 6w^{(3)}(x) = 6x^2 - 10x + 7 \in \mathbb{Z}_{125}[x]. \end{aligned}$$

Damit hat man die gesuchten Faktoren $u(x)w(x) = a(x)$ im Ring $\mathbb{Z}[x]$.

Das obige Beispiel benutzt das Wissen, daß 6 der korrekte Leitkoeffizient von $w(x)$ ist, aber es scheint, daß dieses Wissen im allgemeinen Fall nicht verfügbar ist. Doch es wird hier kurz eine allgemeine Methode beschrieben, die durch eine leichte Abänderung des Originalproblems auf eine Situation führt, in der die korrekten Leitkoeffizienten von beiden Faktoren immer bekannt sind. Für jetzt ist das Ergebnis, daß $u^{(l)}$ und $w^{(l)}$ im Ring $\mathbb{Z}_{p^l}[x]$ assoziiert zu den echten Faktoren über den ganzen Zahlen sind, zu beweisen. Zu diesem Zweck erinnere man sich daran, daß die Einheiten eines Polynomringes genau die Einheiten seines Koeffizientenringes sind. Deshalb muß man wissen welche Elemente eines Ringes der Form $\mathbb{Z}_{p^k}$ Einheiten sind. Anders als im Körper $\mathbb{Z}_p$, in dem jedes von Null verschiedene Element eine Einheit ist, gibt es im Ring $\mathbb{Z}_{p^k}[x]$ $(k > 1)$ nichttriviale Elemente, die keine multiplikativen Inversen haben (z.B. ist das Element $p \in \mathbb{Z}_{p^k}$ keine Einheit). Der folgende Satz zeigt, daß die meisten Elemente des Ringes $\mathbb{Z}_{p^k}$ Einheiten sind und identifiziert solche Elemente, die keine Einheit sind.

Satz 7.9
Sei p eine Primzahl und k eine positive ganze Zahl. Ein Element $a \in \mathbb{Z}_{p^k}$ ist genau dann eine Einheit, wenn a nicht durch p (im Integritätsbereich $\mathbb{Z}$) teilbar ist.

Beweis:
Zunächst wird behauptet, daß die ganze Zahl p keine Einheit in $\mathbb{Z}_{p^k}$ ist. Für $k = 1$ gilt die Behauptung offensichtlich, denn dann ist p das Nullelement von $\mathbb{Z}_p$. Wäre p für $k > 1$ eine Einheit in $\mathbb{Z}_{p^k}$, dann würden ganze Zahlen p^{-1} und c existieren, so daß im Ring $\mathbb{Z}$

$$pp^{-1} = cp^k + 1,$$

womit

$$p(p^{-1} - cp^{k-1}) = 1$$

gilt, was $p|1$ impliziert. Letzteres ist nicht möglich, woraus die Behauptung folgt.
'$\Longrightarrow$': Man nehme an $p|a$, so daß $a = px$ für ein ganze Zahl x gilt. Ist a eine Einheit in $\mathbb{Z}_{p^k}$, dann existiert eine ganze Zahl a^{-1}, so daß

$$aa^{-1} \equiv 1 \pmod{p^k}$$

ist. Da aber $a = px$ gilt, folgt, daß

$$pxa^{-1} \equiv 1 \pmod{p^k}$$

ist, was impliziert, daß p ein Inverses modulo p^k besitzt. Dies steht im Widerspruch zur obigen Behauptung.
'$\Longleftarrow$': Angenommen a sei nicht durch p teilbar. Dann ist der $\gcd(a, p^k) = 1$, da die einzigen nichttrivialen Teiler von p^k die p^i $(1 \leq i \leq k)$ sind. Daher kann der erweiterte euklidische Algorithmus zur Berechnung von $a^{-1} \pmod{p^k}$ angewandt werden. □

Satz 7.10
Sei $a(x) \in \mathbb{Z}[x]$ ein gegebenes Polynom über den ganzen Zahlen, p eine Primzahl, die den $Lc[a(x)]$ (Leitkoeffizienten von $a(x)$) nicht teilt und seien $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ zwei teilerfremde Polynome über dem Körper $\mathbb{Z}_p$, so daß

$$a(x) \equiv u^{(1)}(x)w^{(1)}(x) \pmod{p} \tag{7.61}$$

gilt. Sei l eine ganze Zahl, so daß $\frac{1}{2}p^l > B$ ist, wobei B die Größe aller ganzzahligen Koeffizienten von $a(x)$ und allen seinen möglichen Faktoren mit einem Grad von *höchstens* $\max(\deg[u^{(1)}(x)], \deg[w^{(1)}(x)])$ *beschränkt. Seien* $u^{(k)}(x)$ *und* $w^{(k)}(x)$ *mit der Hensel-Konstruktion berechnete Faktoren, so daß*

$$a(x) \equiv u^{(k)}(x)w^{(k)}(x) \pmod{p^k} \tag{7.62}$$

und

$$\begin{aligned} u^{(k)}(x) &\equiv u^{(1)}(x) \pmod{p} \\ w^{(k)}(x) &\equiv w^{(1)}(x) \pmod{p} \end{aligned} \tag{7.63}$$

für $k = 1, 2, \ldots, l$ *gilt. Existieren Polynome* $u(x), w(x) \in \mathbb{Z}[x]$*,so daß*

$$a(x) = u(x)w(x) \in \mathbb{Z}[x] \tag{7.64}$$

und

$$\begin{aligned} \boldsymbol{n}[u(x)] &\equiv \boldsymbol{n}[u^{(1)}(x)] \pmod{p} \\ \boldsymbol{n}[w(x)] &\equiv \boldsymbol{n}[w^{(1)}(x)] \pmod{p}, \end{aligned} \tag{7.65}$$

wobei $\boldsymbol{n}$ *die Normalisierung 'normiere das Polynom als ein Element im Ring* $\mathbb{Z}_p[x]$*' bezeichnet, dann sind die Polynome* $u(x)$ *und* $u^{(l)}(x)$*, ebenso wie* $w(x)$ *und* $w^{(l)}(x)$*, assoziiert im Ring* $\mathbb{Z}_{p^l}[x]$*. Allgemeiner gilt: Für jedes* $k \geq 1$ *sind die Polynome* $\phi_{p^k}(u(x))$ *und* $u^{(k)}(x)$*, ebenso wie* $\phi_{p^k}(w(x))$ *und* $w^{(k)}(x)$*, im Ring* $\mathbb{Z}_{p^k}[x]$ *assoziiert.*

Beweis:
Sei $k \geq 1$ eine feste ganze Zahl. Aus der Annahme, daß p den $\mathrm{Lc}[a(x)]$ nicht teilt, folgt mit Satz 7.9, daß $\mathrm{Lc}[a(x)]$ eine Einheit in $\mathbb{Z}_{p^k}[x]$ ist. Daher kann man das normierte Polynom

$$\tilde{a}(x) = (\mathrm{Lc}[a(x)])^{-1}\, a(x) \in \mathbb{Z}_{p^k}[x]$$

definieren. Es folgt mit (7.62), daß

$$\mathrm{Lc}[a(x)] \equiv \mathrm{Lc}[u^{(k)}(x)]\mathrm{Lc}[w^{(k)}(x)] \pmod{p^k}$$

ist, also klar, daß p den $\mathrm{Lc}[u^{(k)}(x)]$ und $\mathrm{Lc}[w^{(k)}(x)]$ nicht teilt (sonst würde $p|\mathrm{Lc}[a(x)]$ gelten). Man kann daher ebenso die normierten Polynome

$$\begin{aligned} \tilde{u}^{(k)}(x) &= \mathrm{Lc}[u^{(k)}(x)]^{-1}u^{(k)}(x) \in \mathbb{Z}_{p^k}[x] \\ \tilde{w}^{(k)}(x) &= \mathrm{Lc}[w^{(k)}(x)]^{-1}w^{(k)}(x) \in \mathbb{Z}_{p^k}[x] \end{aligned}$$

definieren. Offensichtlich kann man die Polynome $u^{(1)}(x), w^{(1)}(x) \in \mathbb{Z}_p[x]$ normalisieren, was auf die normierten Polynome

$$\tilde{u}^{(1)}(x) = \boldsymbol{n}[u^{(1)}(x)];\;\; \tilde{w}^{(1)}(x) = \boldsymbol{n}[w^{(1)}(x)]$$

führt.

Es ist leicht nachzuprüfen, daß die Bedingungen (7.61), (7.62) und (7.63) erfüllt bleiben, falls $a(x), u^{(1)}(x), w^{(1)}(x), u^{(k)}(x), w^{(k)}(x)$ durch $\tilde{a}(x), \tilde{u}^{(1)}(x), \tilde{w}^{(1)}(x), \tilde{u}^{(k)}(x), \tilde{w}^{(k)}(x)$ ersetzt werden. Damit werden nach dem Korollar zu Satz 7.7 durch die Bedingungen (7.62)-(7.63) im normierten Fall eindeutig die normierten Polynomfaktoren $\tilde{u}^{(k)}(x), \tilde{w}^{(k)}(x) \in \mathbb{Z}_{p^k}[x]$ bestimmt. Man nehme nun an, es existieren Polynomfaktoren $u(x), w(x) \in \mathbb{Z}[x]$, die (7.64)-(7.65) erfüllen und betrachte die Polynome $\phi_{p^k}(u(x)), \phi_{p^k}(w(x)) \in \mathbb{Z}_{p^k}[x]$. Mit der gleichen Begründung wie oben, kann man diese beiden Polynome im Ring $\mathbb{Z}_{p^k}[x]$ normalisieren und erhält die normierten Polynome $\tilde{u}(x), \tilde{w}(x) \in \mathbb{Z}_{p^k}[x]$. Diese normierten Polynome liefern eine weitere Faktorisierung in $\mathbb{Z}_{p^k}[x]$, die die normierten Darstellungen von (7.62)-(7.63) erfüllen. Aufgrund der Eindeutigkeit gilt

$$\tilde{u}^{(k)}(x) = \tilde{u}(x); \;\; \tilde{w}^{(k)}(x) = \tilde{w}(x).$$

Es folgt, daß $u^{(k)}(x)$ und $\phi_{p^k}(u(x))$ und ebenso $w^{(k)}(x)$ und $\phi_{p^k}(w(x))$ im Ring $\mathbb{Z}_{p^k}[x]$ assoziiert sind.

Der obige Beweis gilt für jedes feste $k \geq 1$. Insbesondere für $k = l$; man beachte, da $\frac{1}{2}p^l > B$ ist, man damit $\phi_{p^l}(u(x)) = u(x)$ und $\phi_{p^l}(w(x)) = w(x)$ erhält. □

7.5.2 Modifizierte Hensel-Konstruktion

Das Ergebnis von Satz 7.10 kann dazu benutzt werden die Hensel-Konstruktion so zu 'festigen', daß sie auch im nichtnormierten Fall die korrekten Faktoren über den ganzen Zahlen erzeugt. Dazu versucht man eine Situation herzustellen, in der die korrekten Leitkoeffizienten der Faktoren a priori bekannt sind, was wie folgt erreicht werden kann. Man nehme an, das zu faktorisierende Polynom $a(x) \in \mathbb{Z}[x]$ sei primitiv. (Diese Annahme bedeutet nur, daß zur Faktorisierung eines beliebigen Polynoms über den ganzen Zahlen erst noch der Einheits-Teil und der Inhalt herausgenommen wird und sich so das Problem auf das Faktorisieren des primitiven Teils reduziert). $a(x)$ habe die modulo p Faktorisierung (7.61) und man nehme an, es existieren Faktoren $u(x), w(x) \in \mathbb{Z}[x]$, die (7.64)-(7.65) erfüllen. Die Leitkoeffizienten

$$\alpha = \mathrm{Lc}[a(x)]; \; \mu = \mathrm{Lc}[u(x)]; \; \nu = \mathrm{Lc}[w(x)]$$

erfüllen klar

$$\alpha = \mu\nu,$$

aber an dieser Stelle kennt man die korrekte Aufteilung von α in μ und ν nicht. Definiert man jedoch das neue Polynom

$$\tilde{a}(x) = \alpha a(x)$$

und sucht eine Faktorisierung von $\tilde{a}(x)$, erhält man die Beziehung

$$\tilde{a}(x) = \mu\nu u(x)w(x) = [\nu u(x)][\mu w(x)].$$

Mit anderen Worten, durch Definieren von $\tilde{u}(x) = \nu u(x)$ und $\tilde{w}(x) = \mu w(x)$ sieht man, daß eine Faktorisierung

$$\tilde{a}(x) = \tilde{u}(x)\tilde{w}(x) \in \mathbb{Z}[x]$$

existiert, in der der Leitkoeffizient von jedem Faktor α ist.

Die Hensel-Konstruktion kann nun für das Polynom $\tilde{a}(x)$ so modifiziert werden, daß sie für jedes $k \geq 1$ Faktoren $\tilde{u}^{(k)}(x), \tilde{w}^{(k)}(x) \in \mathbb{Z}_{p^k}[x]$ berechnet, die nicht nur die Bedingungen des Lemmas von Hensel erfüllen, sondern zusätzlich noch die Beziehungen

$$\tilde{u}^{(k)}(x) = \phi_{p^k}(\tilde{u}(x)); \;\; \tilde{w}^{(k)}(x) = \phi_{p^k}(\tilde{w}(x)) \tag{7.66}$$

erfüllen, wobei $\tilde{u}(x), \tilde{w}(x) \in \mathbb{Z}[x]$ die (unbekannten) Faktoren von $\tilde{a}(x)$ über den ganzen Zahlen sind. (Man beachte, daß die Beziehung (7.66) in Beispiel 3 nicht erfüllt ist). Die Modifikation, die in der Hensel-Konstruktion vorgenommen werden kann, ist eine einfache Berichtigung der Einheiten in jedem Iterationsschritt. Bezeichnen $u^{(k)}(x)$ und $w^{(k)}(x)$ die modulo p^k Faktoren von $\tilde{a}(x)$, die die Bedingungen des Lemma von Hensel erfüllen, dann können die modulo p^k Faktoren $\tilde{u}^{(k)}(x)$ und $\tilde{w}^{(k)}(x)$, die die Bedingungen des Lemma von Hensel und zusätzlich (7.66) erfüllen, durch

$$\begin{aligned} \tilde{u}^{(k)}(x) &= \phi_{p^k}(\alpha \times \mathrm{Lc}[u^{(k)}(x)]^{-1} \times u^{(k)}(x)) \\ \tilde{w}^{(k)}(x) &= \phi_{p^k}(\alpha \times \mathrm{Lc}[w^{(k)}(x)]^{-1} \times w^{(k)}(x)) \end{aligned} \tag{7.67}$$

definiert werden (wobei die hier vorkommenden modulo p^k Inversen garantiert existieren, vorausgesetzt es gilt die Bedingung, daß p den $\mathrm{Lc}[a(x)]$ nicht teilt). Die Assoziiertheitsbeziehung vom Satz 7.10 ist zu den Gleichheitsbeziehungen (7.66) verstärkt worden, durch das Wissens, daß der Leitkoeffizient von jedem Faktor α ist. Schließlich wenn $k = l$, wobei l groß genug ist, daß $\frac{1}{2}p^l$ die Größe aller ganzzahligen Koeffizienten von $\tilde{a}(x)$ und seinen Faktoren beschränkt wird, wird (7.66) zu

$$\tilde{u}^{(l)}(x) = \tilde{u}(x); \;\; \tilde{w}^{(l)}(x) = \tilde{w}(x),$$

so daß man die Faktoren von $\tilde{a}(x)$ (die als existent angenommen wurden) erhält. Man beachte, falls die Schranke B für das Originalpolynom $a(x)$ wie in Satz 7.10 definiert wurde, jetzt, da die modifizierte Hensel-Konstruktion auf das größere Polynom $\tilde{a}(x)$ angewandt wird, gefordert werden muß, daß l groß genug ist, so daß

$$\frac{1}{2}p^l > B \ \mathrm{Lc}[a(x)]$$

ist. Der letzte Schritt dieser modifizierten Hensel-Konstruktion ist aus der berechneten Faktorisierung

$$\tilde{a}(x) = \tilde{u}(x)\tilde{w}(x) \in \mathbb{Z}[x],$$

die Faktorisierung von $a(x)$ herzuleiten. Da angenommen wurde, $a(x)$ sei primitiv, gilt $a(x) = \mathrm{pp}[\tilde{a}(x)]$, woraus folgt, daß die gesuchten Faktoren von $a(x)$ definiert sind durch

$$u(x) = \mathrm{pp}[\tilde{u}(x)]; \;\; w(x) = \mathrm{pp}[\tilde{w}(x)].$$

Die Modifikation der Hensel-Konstruktion, die tatsächlich im Algorithmus 7.1 benutzt wird, ist eine effizientere Anwendung der obigen Ideen. Doch vor der Diskussion der verbesserten Version noch ein Beispiel.

Beispiel 4:
Man betrachte noch einmal das Problem von Beispiel 3, bei dem die Hensel-Konstruktion die Faktoren über den ganzen Zahlen nicht erzeugen konnte. Dies war

$$a(x) = 12x^3 + 10x^2 - 36x + 35 \in \mathbb{Z}[x]$$

und

$$a(x) \equiv u^{(1)}(x)w^{(1)}(x) \pmod 5,$$

wobei $u^{(1)}(x) = 2x$ und $w^{(1)}(x) = x^2 + 2$ sind. Man beachte, daß das Polynom $a(x)$ primitiv ist und die Primzahl 5 den Leitkoeffizienten 12 nicht teilt. In dem neuen Entwurf definiert man das neue Polynom

$$\tilde{a}(x) = 12a(x) = 144x^3 + 120x^2 - 432x + 420.$$

Es ist bekannt, daß, falls eine Faktorisierung von $a(x)$ existiert, die die Bedingungen (7.64)-(7.65) erfüllen, ebenso eine entsprechende Faktorisierung existiert, in der 12 der Leitkoeffizient von jedem Faktor ist. Die initiale Faktorisierung

$$\tilde{a}(x) \equiv u^{(1)}(x)w^{(1)}(x) \pmod 5$$

für den Fall $k = 1$, so daß (7.66) erfüllt ist, kann man durch Anwenden von (7.67) auf die gegebenen Polynome $u^{(1)}(x)$ und $w^{(1)}(x)$ erhalten:

$$\begin{aligned} \tilde{u}^{(1)}(x) &= \phi_5(12 \times 2^{-1} \times (2x)) = 2x \\ \tilde{w}^{(1)}(x) &= \phi_5(12 \times 1^{-1} \times (x^2 + 2)) = 2x^2 - 1. \end{aligned}$$

Wendet man den Iterationsschritt $k = 1$ der gewöhnlichen Hensel-Konstruktion auf das Polynom $\tilde{a}(x)$ an, so erhält man

$$\begin{aligned} u^{(2)}(x) &= \tilde{u}^{(1)}(x) + (-x + 1)5 = -3x + 5 \\ w^{(2)}(x) &= \tilde{w}^{(1)}(x) + (x - 1)5 = 2x^2 + 5x - 6. \end{aligned}$$

Die Berichtigung (7.67) führt auf

$$\begin{aligned} \tilde{u}^{(2)}(x) &= \phi_{25}(12 \times (-3)^{-1} \times (-3x + 5)) = 12x + 5 \\ \tilde{w}^{(2)}(x) &= \phi_{25}(12 \times 2^{-1} \times (2x^2 + 5x - 6)) = 12x^2 + 5x - 11. \end{aligned}$$

Im Iterationsschritt $k = 2$ erhält man

$$\begin{aligned} u^{(3)}(x) &= \tilde{u}^{(2)}(x) + (1)5^2 = 12x + 30 \\ w^{(3)}(x) &= \tilde{w}^{(2)}(x) + (-x+1)5^2 = 12x^2 - 20x + 14. \end{aligned}$$

Die Berichtigung (7.67) läßt die Faktoren unverändert — d.h. $\tilde{u}^{(3)}(x) = 12x + 30$; $\tilde{w}^{(3)}(x) = 12x^2 - 20x + 14$. An dieser Stelle ist

$$\tilde{a}(x) - \tilde{u}^{(3)}(x)\tilde{w}^{(3)}(x) = 0,$$

also hält die Iteration an und man hat die Faktorisierung von $\tilde{a}(x)$ erhalten. Daher sind die gesuchten Faktoren des Originalpolynoms $a(x)$:

$$\begin{aligned} u(x) &= \mathrm{pp}[\tilde{u}^{(3)}(x)] = 2x + 5 \\ w(x) &= \mathrm{pp}[\tilde{w}^{(3)}(x)] = 6x^2 - 10x + 7. \quad \Box \end{aligned}$$

7.5.3 Verwenden eines kleineren Multiplikators

In dem oben beschriebenen und im Beispiel 4 angewandten Entwurf beachte man, daß das tatsächlich faktorisierte Polynom $\tilde{a}(x)$ ganzahlige Koeffizienten enthalten kann, die bedeutend größer sind als die ganzzahligen Koeffizienten des Originalpolynoms $a(x)$. Dies kann zu einem Effizienzverlust, verglichen mit dem Entwurf, der direkt mit dem Originalpolynom $a(x)$ arbeitet, führen. Man betrachte den Entwurf, in dem die Hensel-Konstruktion auf das Originalpolynom $a(x)$ (genau wie in Beispiel 3) solange angewandt wird bis l groß genug ist, daß

$$\tfrac{1}{2}p^l > B \ \mathrm{Lc}[a(x)], \tag{7.68}$$

und dann am Ende eine 'stelle den Leitkoeffizienten wieder her'-Operation leistet. Ein Nachteil eines solchen Entwurfes ist es, daß die Iteration ihr 'automatisches' Abbruchkriterium verliert — d.h. es ist in einem solchen Schema im allgemeinen nicht möglich zu erkennen, ob genug Iterationen ausgeführt wurden, um die Schranke (7.68) zu erfüllen. Dieser Nachteil wird noch durch zwei zusätzliche Tatsachen verschärft: (i) In der Praxis ist die Schranke B meistens immer eine sehr pessimistische Schranke und (ii) jeder nachfolgende Iterationsschritt kostet gewöhnlich mehr als der Vorausgehende (z.B. man beachte in Beispiel 3 das Anwachsen der Größe der Koeffizienten von $e(x)$ und die Faktoren in den nachfolgenden Iterationsschritten). Deshalb kann ein iterativer Entwurf, der teure Iterationen durch eine von der Schranke (7.68) unabhängige Terminierung möglicherweise einspart, signifikant werden. Ein sehr ernst zunehmender Nachteil eines Entwurfs, der zum Schluß eine 'stelle den Leitkoeffizienten wieder her'-Operation benutzt, tritt im multivariaten Fall auf.

Das Problem der Größe der Koeffizienten, in dem Entwurf, der $\tilde{a}(x)$ faktorisiert, anstatt direkt $a(x)$ zu faktorisieren, kann unter bestimmten Umständen wie folgt teilweise gemindert werden. Angenommen man wählt einen Multiplikator γ, der kleiner ist als der $\mathrm{Lc}[a(x)]$ zur Definition des neuen Polynoms

$$\tilde{a}(x) = \gamma a(x). \tag{7.69}$$

Angenommen es ist bekannt, daß γ ein Vielfaches des Leitkoeffizienten eines der zu berechnenden Faktoren ist, z.B. von $u(x)$ — d.h. man nimmt an es ist bekannt, daß

$$\mathrm{Lc}[u(x)] | \gamma \tag{7.70}$$

gilt. Dann hat das durch (7.69) definierte Polynom $\tilde{a}(x)$ eine Faktorisierung, in der die Leitkoeffizienten der Faktoren bekannt sind, wobei wie gewöhnlich angenommen wird, daß eine entsprechende Faktorisierung des Originalpolynoms $a(x)$ existiert. (Man beachte, daß die Wahl $\gamma = \mathrm{Lc}[a(x)]$, die vorher benutzt wurde, ein spezieller Fall von (7.70) ist). Um diese Tatsache zu sehen, sei die Faktorisierung von $a(x)$

$$a(x) = u(x)w(x) \in \mathbb{Z}[x];$$

und wie zuvor definiert man die Leitkoeffizienten

$$\alpha = \mathrm{Lc}[a(x)];\ \mu = \mathrm{Lc}[u(x)];\ \nu = \mathrm{Lc}[w(x)].$$

Durch (7.70) definiert man zusätzlich die ganze Zahl

$$\beta = \gamma/\mu.$$

Dann erfüllt das durch (7.69) definierte Polynom $\tilde{a}(x)$ die Gleichung

$$\tilde{a}(x) = \beta\mu u(x)w(x) = [\beta u(x)][\mu w(x)].$$

Hieraus sieht man durch die Definition von $\tilde{u}(x) = \beta u(x)$ und $\tilde{w}(x) = \mu w(x)$, daß eine Faktorisierung

$$\tilde{a}(x) = \tilde{u}(x)\tilde{w}(x) \in \mathbb{Z}[x]$$

existiert, mit

$$\mathrm{Lc}[\tilde{u}(x)] = \beta\mu = \gamma;\ \mathrm{Lc}[\tilde{w}(x)] = \mu\nu = \alpha,$$

wobei α die bekannte ganze Zahl $\mathrm{Lc}[a(x)]$ ist und γ spezifiziert ist. Dies ist die Verallgemeinerung des vorhin diskutierten Entwurfes, der im Algorithmus 7.1 mit γ als optionale Eingabe implementiert ist. Wird γ bei der Eingabe nicht definiert, dann wird im Schritt 1 des Algorithmus $\gamma = \mathrm{Lc}[a(x)]$ gesetzt. Es scheint, daß die Spezifikation für ein γ kleiner als $\mathrm{Lc}[a(x)]$, das (7.70) erfüllt, in den meisten praktischen Fällen unmöglich ist. Umgekehrt jedoch ist, in der Anwendung des Hensel-lifting Algorithmus auf das wichtige Problem der gcd-Berechnung von Polynomen, die Spezifikation von γ immer möglich. Schließlich beachte man, daß durch (7.69) die Terminierungsbedingung (7.68) (für den Fall, wenn die Faktorisierung nicht existiert) in die Bedingung

$$\tfrac{1}{2}p^l > B\gamma$$

abgeändert werden kann.

7.5.4 Die replaceLc Operation

Der Entwurf des Algorithmus 7.1 ist nun bis auf eine signifikante Modifikation vollständig erklärt. Der beschriebene (und in Beispiel 4 angewandte) Entwurf verlangt, daß die Formeln (7.67) angewandt werden, um die Einheiten in jedem Iterationsschritt zu berichtigen. Man kann jedoch sehen, daß im Schritt 4 von Algorithmus 7.1 keine solche Berichtigung von Einheiten in jeder Iteration in der **while**-Schleife enthalten ist. Der Schritt 4 von Algorithmus 7.1 ist einfach eine Implementierung der ursprünglichen unveränderten Hensel-Konstruktion. Der Grund, warum es im Algorithmus 7.1 möglich ist, die extra Kosten für die Korrektur der Einheiten in jedem Iterationsschritt zu vermeiden, stammt von der noch zu erklärenden 'replaceLc' Operation aus Schritt 3. Diese neue Operation ist eine raffinierte Modifikation, die von Yun beschrieben und auf einen Vorschlag von Moses benutzt wird. Man betrachte das durch (7.69) definierte Polynom $\tilde{a}(x)$ und seine angepaßten modulo p Faktoren $\tilde{u}^{(1)}(x), \tilde{w}^{(1)}(x) \in \mathbb{Z}_p[x]$ (wie in Schritt 1 des Algorithmus 7.1), so daß

$$\tilde{u}^{(1)}(x) = \phi_p(\tilde{u}(x));\;\; \tilde{w}^{(1)}(x) = \phi_p(\tilde{w}(x)) \tag{7.71}$$

gilt, wobei $\tilde{u}(x)$ und $\tilde{w}(x)$ die oben diskutierten Faktoren von $\tilde{a}(x)$ über den ganzen Zahlen sind, so daß gilt

$$\mathrm{Lc}[\tilde{u}(x)] = \gamma;\; \mathrm{Lc}[\tilde{w}(x)] = \mathrm{Lc}[a(x)]. \tag{7.72}$$

Schreibt man die modulo p Faktoren in der Form

$$\begin{aligned} \tilde{u}^{(1)}(x) &= \mu_m x^m + \mu_{m-1}x^{m-1} + \ldots + \mu_0 \\ \tilde{w}^{(1)}(x) &= \nu_n x^n + \nu_{n-1}x^{n-1} + \ldots + \nu_0, \end{aligned}$$

wobei $\mu_m \neq 0$ und $\nu_n \neq 0$ sind, folgt aus (7.71)-(7.72), daß $\mu_m = \phi_p(\gamma)$ und $\nu_n = \phi_p(\mathrm{Lc}[a(x)])$ ist. Man nimmt nun an, daß die Faktoren $\tilde{u}^{(1)}(x)$ und $\tilde{w}^{(1)}(x)$ durch eine einfache Ersetzung der Leitkoeffizienten μ_m und ν_n mit γ und $\alpha = \mathrm{Lc}[a(x)]$ verändert werden. Zu diesem Zweck definiert man die algorithmische Operation *replaceLc* wie folgt:

> Zu einem gegebenen Polynom $a(x) \in R[x]$ über einem Koeffizientenring R und einem Element $r \in R$ ist das Ergebnis der Operation replaceLc$(a(x), r)$ das Polynom, das man aus $a(x)$ durch Ersetzen des Leitkoeffizienten von $a(x)$ mit r erhält.

In dieser algorithmischen Notation werden die Polynome $\tilde{u}^{(1)}(x)$ und $\tilde{w}^{(1)}(x)$ durch die Polynome replaceLc$(\tilde{u}^{(1)}(x), \gamma)$ und replaceLc$(\tilde{w}^{(1)}(x), \alpha)$ ersetzt. Seien nun $\tilde{u}^{(1)}(x)$ und $\tilde{w}^{(1)}(x)$ die modifizierten Faktoren — d.h.

$$\begin{array}{rcl} \tilde{u}^{(1)}(x) & = & \gamma x^m + \mu_{m-1}x^{m-1} + \ldots + \mu_0 \\ \tilde{w}^{(1)}(x) & = & \alpha x^n + \nu_{n-1}x^{n-1} + \ldots + \nu_0. \end{array} \tag{7.73}$$

Dann sind die Leitkoeffizienten von $\tilde{u}^{(1)}(x)$ und $\tilde{w}^{(1)}(x)$ nicht mehr in der gebräuchlichen Darstellung als Elemente des Körpers $\mathbb{Z}_p$ dargestellt, aber nichtsdestoweniger gilt die Eigenschaft

$$\tilde{a}(x) \equiv \tilde{u}^{(1)}(x)\tilde{w}^{(1)}(x) \pmod{p}.$$

Es kann deshalb (7.73) als Initialfaktoren für die Hensel-Konstruktion benutzt werden. Man betrachte die Form der auf (7.73) basierenden, durch die Hensel-Konstruktion berechneten sukzessiven Faktoren. Benutzt man die Notation von Schritt 4 des Algorithmus 7.1, so berechnet man zuerst

$$c(x) = \frac{e(x)}{p} = \frac{\tilde{a}(x) - \tilde{u}^{(1)}(x)\tilde{w}^{(1)}(x)}{p}$$

(wobei der Ring dieser Berechnung $\mathbb{Z}[x]$ ist). Da $\mathrm{Lc}[\tilde{a}(x)] = \gamma\alpha$ ist, gilt klar mit (7.73)

$$\deg[c(x)] < \deg[\tilde{a}(x)] = \deg[\tilde{u}^{(1)}(x)] + \deg[\tilde{w}^{(1)}(x)].$$

Diese strenge Ungleichung impliziert, daß die Hensel-Konstruktion in folgendem Sinne genau das Gleiche leistet wie im normierten Fall. Die Lösung $\sigma(x), \tau(x) \in \mathbb{Z}_p[x]$ der diophantischen Polynomgleichung im Schritt 4 des Algorithmus erfüllt (wie gewöhnlich) die Bedingung

$$\deg[\sigma(x)] < \deg[\tilde{w}^{(1)}(x)]$$

und zusätzlich gilt noch die Bedingung

$$\deg[\tau(x)] < \deg[\tilde{u}^{(1)}(x)]$$

(siehe Satz 7.6). Daher bleiben beim Aktualisieren der Faktoren im Schritt 4.2 des Algorithmus die Leitkoeffizienten beider Faktoren unverändert. Das ist eine wünschenswerte Eigenschaft, da diese Leitkoeffizienten bereits als die korrekten ganzzahligen Koeffizienten bekannt sind. Aus dem gleichen Grund bleiben in jeder sukzessiven Iteration der Hensel-Konstruktion die Leitkoeffizienten unverändert. Da die durch diesen Entwurf berechneten sukzessiven Faktoren das Lemma von Hensel erfüllen müssen, garantiert schließlich Satz 7.10, daß nach einer genügenden Zahl von Iterationen die berechneten Faktoren zu den echten Faktoren über den ganzen Zahlen (falls solche Faktoren existieren) assoziiert sind; aber da die berechneten Faktoren die gleichen Leitkoeffizienten wie die echten Faktoren besitzen, können sie nur assoziiert sein, wenn sie identisch sind. Deshalb werden die gesuchten Faktoren von $\tilde{a}(x)$ durch die Iteration in Schritt 4 des Algorithmus 7.1 ohne eine zusätzliche Berichtigung der Einheiten berechnet.

Beispiel 5:
Man betrachte noch einmal das Problem, das nichtnormierte Polynom aus Beispiel 3 zu faktorisieren:

$$a(x) = 12x^3 + 10x^2 - 36x + 35 \in \mathbb{Z}[x].$$

Diesmal wird der Algorithmus 7.1 in seiner ganzen Allgemeinheit angewandt. Die Eingabe für den Algorithmus ist das primitive Polynom $a(x)$, die Primzahl $p = 5$ (man beachte, daß p den $\mathrm{Lc}[a(x)]$ nicht teilt) und die zwei teilerfremden modulo p Faktoren $u^{(1)}(x) = 2x$ und $w^{(1)}(x) = x^2 + 2$ von $a(x)$. Der Wert für die Schranke B, die vom Algorithmus gefordert ist, wird in diesem Beispiel nicht benötigt, weil die Iteration durch das Finden der Faktorisierung beendet wird. Die ganze Zahl γ ist bei der Eingabe undefiniert.
In Schritt 1 des Algorithmus 7.1 werden folgende Werte bestimmt:

$$\begin{aligned} \alpha &= 12; \\ \gamma &= 12; \\ a(x) &= 144x^3 + 120x^2 - 432x + 420; \\ u^{(1)}(x) &= 2x;\ w^{(1)}(x) = 2x^2 - 1. \end{aligned}$$

In Schritt 2 wird der erweiterte euklidische Algorithmus im Ring $\mathbb{Z}_5[x]$ auf $u^{(1)}(x)$ und $w^{(1)}(x)$ angewandt und führt auf

$$s(x) = x;\ t(x) = -1.$$

In Schritt 3 werden die Leitkoeffizienten von $u^{(1)}(x)$ und $w^{(1)}(x)$ durch ihre korrekten ganzzahligen Koeffizienten ersetzt und man erhält:

$$\begin{aligned} u(x) &= 12x;\ w(x) = 12x^2 - 1; \\ e(x) &= 120x^2 - 420x + 420; \\ modulus &= 5. \end{aligned}$$

In Schritt 4 wird eine Folge von Werten für $\sigma(x)$, $\tau(x)$, $u(x)$, $w(x)$ und $e(x)$ wie folgt berechnet:

nach Iteration	$\sigma(x)$	$\tau(x)$	$u(x)$	$w(x)$	$e(x)$
0	—	—	$12x$	$12x^2-1$	$120x^2-420x+420$
1	$x-2$	1	$12x+5$	$12x^2+5x-11$	$-325x+475$
2	$-x+1$	1	$12x+30$	$12x^2-20x+14$	0

Schließlich erhält man in Schritt 5 die folgenden Werte:

$$\begin{aligned} \delta &= cont[12x+30]=6; \\ u(x) &= \frac{12x+30}{6}=2x+5; \\ w(x) &= \frac{12x^2-20x+14}{\frac{12}{6}}=6x^2-10x+7. \end{aligned}$$

Man beachte, daß diese Berechnung im Grunde genommen äquivalent zur Berechnung in Beispiel 4 ist, mit dem Unterschied, daß hier die Kosten zur Berichtigung der Einheiten in jedem Iterationsschritt vermieden wurden. □

7.6 Erweiterung des Lemma von Hensel auf den multivariaten Fall

Sei $F(x,y_1,\ldots,y_n)$ ein normiertes Polynom aus $\mathbb{Z}[y_1,\ldots,y_n][x]$. Für eine gegebene Menge ganzer Zahlen $b=\{b_1,\ldots,b_n\}$ sei

$$F(x,y_1,\ldots,y_n)\equiv G_1(x)H_1(x) \pmod{\langle p^l, y_1-b_1,\ldots,y_n-b_n\rangle},$$

wobei $G_1(x),H_1(x)\in\mathbb{Z}_{p^l}[x]$ und $\gcd(G_1,H_1)\equiv 1 \pmod{p^l}$. Sei $A(x),B(x)\in\mathbb{Z}_{p^l}[x]$ so gefunden, daß

$$AG_1+BH_1\equiv 1 \pmod{p^l}$$

und $\deg(A)<\deg(H_1)$, $\deg(B)<\deg(G_1)$ gilt.
Für $b=\{b_1,\ldots,b_n\}$ kann $F(x,y_1,\ldots,y_n)$ in folgender Variante der Taylor-Reihenrepräsentation dargestellt werden:

$$\begin{aligned} F(x,y_1,\ldots,y_n) &= F(x,b_1,\ldots,b_n)+\sum_{i=1}^{n}c_i(x)(y_i-b_i) \qquad (7.74)\\ &\quad +\sum_{i=1}^{n}\sum_{j=i}^{n}c_{ij}(x)(y_i-b_i)(y_j-b_j) \end{aligned}$$

$$+\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} c_{ijk}(x)(y_i-b_i)(y_j-b_j)(y_k-b_k)$$
$$+\dots\ .$$

Sei $S=\{y_1-b_1,\dots,y_n-b_n\}$. Bezeichnet man

$$F(x,b_1,\dots,b_n)\equiv F(x,y_1,\dots,y_n) \pmod S$$

(d.h. man unterdrückt alle Terme in (7.74), die Elemente von S enthalten) und

$$\begin{aligned} F &\pmod{S^2} = F(x,b_1,\dots,b_n)+\sum_{i=1}^{n} c_i(x)(y_i-b_i)\\ &\vdots\\ F &\pmod{S^m}, \end{aligned}$$

dann ist der Induktionsschritt für den allgemeinen Algorithmus:
Sei R der Ring $\mathbb{Z}_{p^l}[y_1,\dots,y_n]$,
$(R/S^m)[x]$ stehe für $R[x] \pmod{S^m}$, $\{y_i\}=y_1,\dots,y_n$. Für $m\geq 1$ seien $G_m(x,\{y_i\})$ und $H_m(x,\{y_i\})\in(R/S^m)[x]$, so daß gilt:

$$F(x,\{y_i\})\equiv G_m(x,\{y_i\})H_m(x,\{y_i\}) \pmod{\langle p^l,S^m\rangle}.$$

Sei $D_m=G_mH_m-F$ mit

$$D_m=\sum_{i_1=1}^{n}\cdots\sum_{i_m=1}^{n} c_{i_1,i_2,\dots,i_m}(x)\prod_{j=1}^{m}(y_{i_j}-b_{i_j}).$$

Sei $C_I(x)$ ein typisches Koeffizientenpolynom von D_m. Wie für den univariaten Fall finde man $Q_I(x),P_I(x)$ mithilfe der Polynomdivision, so daß gilt:

$$C_I(x)A(x)=Q_I(x)H_1(x)+P_I(x).$$

Sei $A_I(x)=P_I(x)$, $B_I(x)=Q_I(x)G_1(x)+G_I(x)B(x)$ (A und B sind oben definiert). Damit erhält man:

$$A_IG_1+B_IH_1\equiv C_I \pmod{p^l}$$

mit $\deg(A_I)<\deg(H_1)$ und $\deg(B_I)<\deg(G_1)$. Setzt man

$$G_{m+1}=G_m-\sum_{\substack{i_j=1\\ j=1,\dots,m}}^{n} B_{i_1,i_2,\dots,i_m}(x)\prod_{j=1}^{m}(y_{i_j}-b_{i_j})$$

und

$$H_{m+1}=H_m-\sum_{\substack{i_j=1\\ j=1,\dots,m}}^{n} A_{i_1,i_2,\dots,i_m}(x)\prod_{j=1}^{m}(y_{i_j}-b_{i_j}),$$

dann gilt

$$\begin{aligned}
G_{m+1}H_{m+1} &\equiv G_mH_m - \sum_{\substack{i_j=1\\ j=1,\ldots,m}}^{n} (A_{i_1,i_2,\ldots,i_m}G_1 + B_{i_1,i_2,\ldots,i_m}H_1)\prod_{j=1}^{m}(y_{i_j}-b_{i_j})\\
&\equiv G_mH_m - \sum_{\substack{i_j=1\\ j=1,\ldots,m}}^{n} c_{i_1,i_2,\ldots,i_m}\prod_{j=1}^{m}(y_{i_j}-b_{i_j})\\
&\equiv F \pmod{\langle p^l, S^{m+1}\rangle}.
\end{aligned}$$

Der Algorithmus beginnt unter der Annahme l sei groß genug, so daß $\frac{1}{2}p^l$ alle ganzzahligen Koeffizienten beschränkt und wird solange fortgesetzt bis $m=n$ gilt, wobei n die Anzahl der Elemente von S begrenzt, die in jedem einzelnen Term $C_I\prod_{j=1}^{m}(y_{i_j}-b_{i_j})$ in der Reihendarstellung (7.74) von F auftreten.
Dieser Algorithmus stellt sicher, daß

$$\begin{aligned}
\forall m\geq 1: \quad G_mH_m &\equiv F \pmod{\langle p^l, S^m\rangle}\\
\text{und } G_m &\equiv G_1 \pmod{\langle p^l, S\rangle},\ H_m\equiv H_1 \pmod{\langle p^l, S\rangle}
\end{aligned}$$

gilt, d.h. ist man bei n angekommen, sind G_n und H_n Polynome in $\mathbb{Z}$.
Ist b eine Menge „günstiger Werte“ erhält man $F=G_nH_n$ in $\mathbb{Z}$, andernfalls wird eine neue Menge von Werten benutzt.

Bemerkung: Dies ist die Herleitung für normierte multivariate Polynome. Im nichtnormierten Fall muß ähnlich dem univariaten Fall vorgegangen werden.

7.7 Der EZGCD-Algorithmus

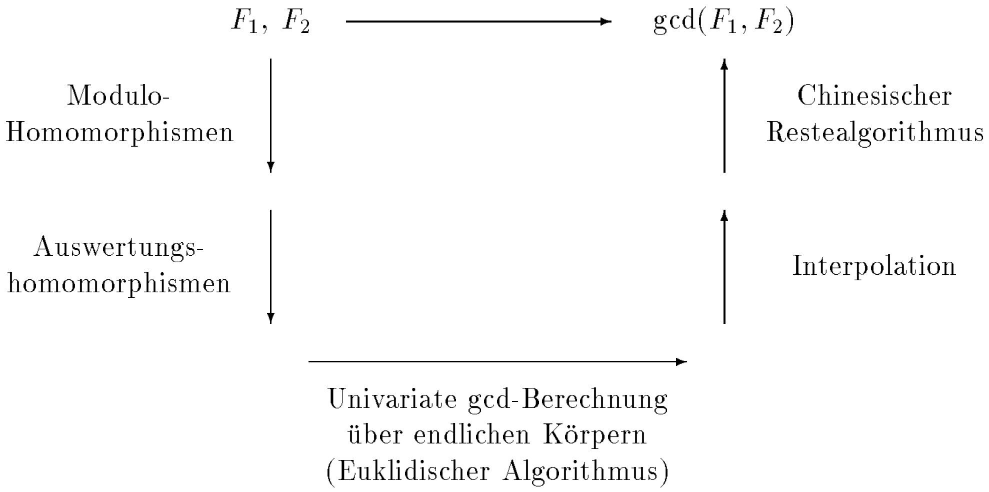
Erweiterter Zassenhaus-Algorithmus zur Berechnung des größten gemeinsamen Teiler von multivariaten Polynomen.

Einführung: gcd-Algorithmen

- Klassisch: erweiterter euklidischer Algorithmus
- Reduzierter und Subresultanter Algorithmus
- Modularer Algorithmus
- p-adisch: der EZGCD (Moses & Yun)

Die ersten beiden wurden bereits vorgestellt. Die Schwierigkeit dabei ist das Anwachsen der Koeffizienten in den Zwischenergebnissen (speziell, wenn der ggT klein ist). Am besten sind diese, wenn der Grad jeder Variable klein ist (≤ 2).

Modularer Algorithmus: (Brown)
F_1, F_2: multivariate Polynome



Dieser Algorithmus ist dafür gestrickt in den „worst cases“ besser zu arbeiten und ist unempfindlich gegen dünn besetzte Polynome. Am besten ist er bei univariaten Polynomen und kleinem gcd (wie der Erste).
Daher braucht man einen Algorithmus, der die Vorzüge des modularen Algorithmus immer ausnutzt. Dies ist der EZGCD-Algorithmus, der auf dem Hensel-Lifting basiert. (In einigen Fällen ist der reduzierte Algorithmus besser).

7.7.1 Ein Überblick über den EZGCD-Algorithmus

$\bar{F}(x_1, \ldots, x_\nu), \bar{G}(x_1, \ldots, x_\nu) \in \mathbb{Z}[x_1, \ldots, x_\nu]$
Dieser Algorithmus berechnet den $\gcd(\bar{F}, \bar{G})$ und die Kofaktoren (erweiterter Algorithmus).

Schritt 1: Inhalt und primitiver Teil

> *Man betrachte $\bar{F}, \bar{G}$ als Polynome in $\mathbb{Z}[x_2, \ldots, x_\nu][x_1]$. Seien $\bar{f} = \operatorname{cont}(\bar{F})$, $\bar{g} = \operatorname{cont}(\bar{G})$, $\bar{d} = \gcd(\bar{f}, \bar{g})$, $F = \operatorname{pp}(\bar{F})$ und $G = \operatorname{pp}(\bar{G})$.*

Den Inhalt erhält man durch eine Variante dieses Algorithmus, $\bar{d}$ erhält man rekursiv. Es bleibt

$$D = \gcd(F, G),\ \hat{F} = \frac{F}{D},\ \hat{G} = \frac{G}{D}$$

zu berechnen. Dann ist der $\gcd(\bar{F}, \bar{G}) = \bar{d}D$ und die Kofaktoren sind $(\frac{\bar{f}}{\bar{d}})\hat{F}$ und $(\frac{\bar{g}}{\bar{d}})\hat{G}$. [$\mathrm{cont}(P) = \gcd($ alle Koeffizienten von $P)$; $\mathrm{pp}(P) = \frac{P}{\mathrm{cont}(P)}$]

Schritt 2: Auswertung und univariater gcd

Man wähle eine Menge von $(\nu - 1)$ ganzen Zahlen $b = \{b_2, \ldots, b_\nu\}$, so daß die Grade von F und G in x_1 durch Einsetzen von $x_i = b_i$, $i = 2, \ldots, \nu$ nicht kleiner werden.

Solch eine Auswertung heißt eine *gültige* Auswertung. Es wird die Schreibweise $P_b(x_1) = P(x_1, b_2, \ldots, b_\nu)$ für eine gegebene Menge b von ganzen Zahlen benutzt.

Man berechne $F_b(x_1)$, $G_b(x_1)$, $D_0(x_1) = \gcd(F_b, G_b)$.

Dieser gcd kann mit dem univariaten modularen GCD-Algorithmus berechnet werden, der in diesem Fall sehr effizient ist. Mit der quadratischen Erweiterung des Lemmas von Hensel kann der chinesische Restealgorithmus ersetzt werden, falls die Schranke für die Koeffizienten außerordentlich groß ist.

Schritt 3: Vorbereitung des EZ Algorithmus

Falls $\deg(D_0(x_1)) = 0$ *dann* $D_0(x_1) = 1$ *und* $D = \bar{d}$.
Falls $\deg(D_0(x_1)) = \deg(F_b(x_1))$ *oder* $\deg(G_b(x_1))$ *dann gilt F teilt G oder G teilt F oder eine neue gültige b-Auswertung muß gemacht werden.*
Andernfalls, bestimme man ob
3.1 $\gcd(D_0(x_1), F_b(x_1)/D_0(x_1)) = 1$ *oder*
$\gcd(D_0(x_1), G_b(x_1)/D_0(x_1)) = 1$.

Wie bereits zu sehen war, ist dies eine notwendige Bedingung, um den EZ Algorithmus anwenden zu können. Trifft die Bedingung 3.1 nicht für zwei verschiedene Auswertungen zu, dann benutzt der Algorithmus in diesem Fall eine spezielle Methode (quadratfreie Zerlegung von D), die auch im EZ Algorithmus angewandt wird. Angenommen die Bedingung 3.1 ist erfüllt, dann kann man unter der Voraussetzung

3.2 $\deg(D_0(x_1)) = \deg_{x_1}(D(x_1, \ldots, x_\nu))$

fortfahren. Diese wird in verschiedenen Schritten des Algorithmus getestet und tritt gezwungenermaßen ein.

Schritt 4: Anwenden des EZ Algorithmus

> *Angenommen, F erfüllt die Bedingung 3.1. Dann benutze man den EZ Algorithmus mit F, $D_0(x_1)$ und $F_b(x_1)/D_0(x_1)$, um $D^*(x_1,\dots,x_\nu)$ zu erhalten.*

Aus den Eigenschaften des EZ Algorithmus folgt, daß $D_b^*(x_1) = D_0(x_1)$ und D^* ein echter primitiver Teiler von F ist.

Bemerkung: Der EZ Algorithmus, der hier benutzt wird benötigt nur $F(x_1,\dots,x_\nu)$ und $D_0(x_1)$, um $D^*(x_1,\dots,x_\nu)$ zu erhalten. $G(x_1,\dots,x_\nu)$ wird an dieser Stelle ignoriert. Dies führt auf eine

Erweiterung: Man kann die Schritte 1, 2, 3 wiederholt benutzen, um den gemeinsamen univariaten gcd $D_0(x_1)$ für mehrere Polynome $\bar{F}_i(x_1,\dots,x_\nu)$, $i = 1,\dots,k$ zu erhalten. Dann braucht man zur Konstruktion des multivariaten gcd der primitiven Teile von $\bar{F}_1,\dots,\bar{F}_k$ nur eines der Polynome $\bar{F}_i$.

Die obige Bemerkung ist eine direkte Konsequenz aus dem folgenden Satz und seinem Korollar:
$D(x_1,\dots,x_\nu) = \gcd(F(x_1,\dots,x_\nu), G(x_1,\dots,x_\nu))$
$b = \{b_2,\dots,b_\nu\}$, so daß $\deg(F_b(x_1)) = \deg_{x_1}(F)$ und $\deg_{x_1}(G) = \deg(G_b(x_1))$ gilt.
$D_0(x_1) = \gcd(F_b(x_1), G_b(x_1))$

Satz 7.11
Erfüllt $D_0(x_1)$ die Eigenschaften

(i) $\deg\ D_0(x_1) = \deg_{x_1}(D)$

(ii) $\gcd(D_0(x_1), F_b(x_1)/D_0(x_1)) = 1$

dann gilt für jeden primitiven Teiler $D^*(x_1,\dots,x_\nu)$ von F mit $D_b^*(x_1) = D_0(x_1)$

$$D(x_1,\dots,x_\nu) = D^*(x_1,\dots,x_\nu)Q(x_2,\dots,x_\nu),$$

wobei Q nicht von x_1 abhängt.

Korollar 7.12 (Korrektheit des EZGCD-Algorithmus)
Seien $\bar{F}(x_1,\ldots,x_\nu)$ und $\bar{G}(x_1,\ldots,x_\nu)$ multivariate Polynome über den ganzen Zahlen und $\bar{D}(x_1,\ldots,x_\nu)$ ihr gcd, $F = \mathrm{pp}_{x_1}(\bar{F})$, $\bar{f} = \mathrm{cont}_{x_1}(\bar{F})$, $G = \mathrm{pp}_{x_1}(\bar{G})$, $\bar{g} = \mathrm{cont}_{x_1}(\bar{G})$. *Sei $b = \{b_2,\ldots,b_\nu\}$ eine Menge von ganzzahligen Werten, die die Bedingungen $\deg_{x_1}(F) = \deg(F_b(x_1))$, $\deg_{x_1}(G) = \deg(G_b(x_1))$ erfüllen und ein $D_0(x_1)$ erzeugen, das die Bedingungen (i) und (ii) des Satzes 7.11 erfüllen. Dann erfüllt $D^*(x_1,\ldots,x_\nu)$, das man von $D_0(x_1)$ mit dem EZ Algorithmus erhält,*

$$\bar{D}(x_1,\ldots,x_\nu) = D^*(x_1,\ldots,x_\nu)\ \underbrace{\mathrm{gcd}(\bar{f},\bar{g})}_{\bar{d}}.$$

7.7.2 Der EZGCD-Algorithmus

Im folgenden detailierten Algorithmus entsprechen die Schritte (A1) bis (A9) dem vorher gegebenen Überblick und die Schritte (S1) bis (S7) dem erwähnten Spezialfall. Diese Schritte enthalten das Wesentliche der Ideen für einen Algorithmus zur quadratfreien Zerlegung eines multivariaten Polynoms unter Benutzung des EZ Algorithmus.

(A1) $\bar{f} = \mathrm{cont}(\bar{F})$, $\bar{g} = \mathrm{cont}(\bar{G})$ und $\bar{d} = \mathrm{gcd}(\bar{f},\bar{g})$.
$F = \bar{F}/\bar{f}$ und $G = \bar{G}/\bar{g}$

(A2) Falls $b = \{b_2 = 0, b_3 = 0,\ldots,b_\nu = 0\}$ eine gültige Auswertung ist, dann berechne $F_b(x_1)$, $G_b(x_1)$ und $D_b(x_1) = \mathrm{gcd}(F_b(x_1), G_b(x_1))$.
Andernfalls suche ein neues gültiges b, das so viele Nullen wie möglich enthält und berechne $D_b(x_1) = \mathrm{gcd}(F_b(x_1), G_b(x_1))$.
Setze $\delta = \deg(D_b)$.

(A3) Falls $\delta = 0$ dann $D_b(x_1) = 1$, return $\bar{D} = \bar{d}$, $(\bar{f}/\bar{d})F$ und $(\bar{g}/\bar{d})G$. Springe nach (A5), falls für den gegenwärtigen Wert von δ (A4) bereits durchlaufen wurde.

(A4) Wähle eine neue Auswertung b' und berechne $D_{b'}(x_1) = \mathrm{gcd}(F_{b'}(x_1), G_{b'}(x_1))$.
Falls $\deg(D_{b'}) < \delta$ dann setze $b = b'$, $D_b = D_{b'}$, $F_b = F_{b'}$, $G_b = G_{b'}$, $\delta = \deg(D_{b'})$ und wiederhole (A4).
Falls $\deg(D_{b'}) = \delta$ dann gehe zurück zu (A3).
Falls $\deg(D_{b'}) > \delta$ dann wiederhole (A4).

(A5) Falls $\deg_{x_1}(F) \le \deg_{x_1}(G)$ und $\delta = \deg_{x_1}(F)$ und falls F teilt G dann return $\bar{D} = \bar{d}F$, $\bar{f}/\bar{d}$ und $(\bar{g}/\bar{d})(G/F)$.
Ähnlich, falls $\deg_{x_1}(G) < \deg_{x_1}(F)$ und $\delta = \deg_{x_1}(G)$ und falls G teilt F dann return $\bar{D} = \bar{d}G$, $(\bar{f}/\bar{d})(F/G)$ und $(\bar{g}/\bar{d})$.

Schlägt die Testdivision in beiden Fällen fehl, nimmt man zusätzlich Auswertungen vor bis ein neuer univariater GCD gefunden wird, dessen Grad kleiner δ ist. Dann setzt man b, D_b, F_b, G_b und δ auf die entsprechenden neuen Werte und geht zurück zu (A3).
Falls $\delta \neq \deg_{x_1}(F)$ und $\delta \neq \deg_{x_1}(G)$ dann fahre mit (A6) fort.

(A6) Falls $\gcd(D_b, F_b/D_b) = 1$ dann setze $B = F$, $D_0 = D_b$ und $\hat{B}_0 = F_b/D_b$ sonst falls $\gcd(D_b, G_b/D_b) = 1$ dann setze $B = G$, $D_0 = D_b$ und $\hat{B}_0 = G_b/D_b$.
Andernfalls gehe zu (S1) zur Spezialfallbetrachtung.

(A7) Wende den EZ Algorithmus auf B, D_0 und $\hat{B}_0$ an, um den primitiven Teiler von B, D^* zu bekommen.

(A8) Für $B = F$ teste ob D^* teilt G
Für $B = G$ teste ob D^* teilt F
Falls der Test erfolgreich ist dann fahre fort.
Andernfalls mache zusätzliche Auswertungen von F und G bis ein neuer univariater GCD gefunden wird, dessen Grad kleiner δ ist. Setze B, D_b, F_b, G_b und δ auf die entsprechenden neuen Werte und gehe zurück nach (A3).

(A9) Return $\bar{D} = \bar{d}D^*$, $(\bar{f}/\bar{d})(F/D^*)$ und $(\bar{g}/\bar{d})(G/D^*)$.

(S1) Berechne $\tilde{F} = \gcd(F, \partial F/\partial x_1)$ und $L = F/\tilde{F}$ mit den Schritten (A2)-(A9).

(S2) Berechne $D = \gcd(L, G)$ und $\tilde{G} = G/D$ mit den Schritten (A2)-(A9), wobei b, die schon gefundene gültige, glückliche Auswertung ist.

(S3) Setze $\tilde{D}_b(x_1) = \gcd(L_b(x_1), \tilde{F}_b(x_1), \tilde{G}_b(x_1))$.

(S4) Falls $\deg(\tilde{D}_b(x_1)) = 0$ dann setze $\tilde{D}_b(x_1) = 1$ und verlasse diesen Spezialfall-Algorithmus (S1)-(S7) durch (A9).

(S5) Falls $L_b/\tilde{D}_b = 1$ dann überspringe (S6).

(S6) Wende den EZ Algorithmus auf L, $\tilde{D}_b$, $L_b/\tilde{D}_b$ an und erhalte dabei $\tilde{D} = \gcd(L, \tilde{F}, \tilde{G})$.

(S7) Setze $D^* = D^*\tilde{D}$, $\tilde{F}_b(x_1) = \tilde{F}_b(x_1)/\tilde{D}_b(x_1)$, $\tilde{G}_b(x_1) = \tilde{G}_b(x_1)/\tilde{D}_b(x_1)$ und gehe zurück nach (S3).

Bemerkungen:

zu (A2): Hier wird mit der Reihendarstellung eines Polynoms gearbeitet, wobei jeder Term von der Form

$$c_{i_1,i_2,\ldots,i_m}(x_1)(x_{i_1}-b_{i_1})\cdots(x_{i_m}-b_{i_m})$$

ist, so daß es wünschenswert ist, möglichst viele $b_i = 0$ zu bekommen, um nicht unnötigerweise die Zahl der Terme ansteigen zu lassen.

zu (A4): Es kann nachgewiesen werden, daß die Zahl „ungünstiger" b-Werte endlich und klein ist. Es wird daher immer eine glückliche Menge gefunden und der Algorithmus terminiert.

zu (A6): Falls $\gcd(D, F/D) \neq 1$ und $\gcd(D, G/D) \neq 1$, dann schlagen beide Tests fehl und die Methode für den Spezialfall muß angewandt werden. Der einfachste solche Fall ist: $F = UV^2$, $G = U^2V \to D = UV$.

zu (A7): Die EZ Methode konstruiert aus D_0 und B_0 D^* und B^*, so daß $B = D^*B^*$ ist, also ist D^* ein Teiler von B.

Kapitel 8

Faktorisierung

Eines – wenn nicht das – grundlegende Problem in der Computer Algebra ist die Faktorisierung von Polynomen.

8.1 Einführung

8.1.1 Der Algorithmus von Kronecker

Es seien $g, f \in \mathbb{Z}[x]$ univariate Polynome über den ganzen Zahlen. Der Faktorisierungsalgorithmus von Kronecker basiert auf der folgenden einfachen Idee:

> Wird $f(x)$ von $g(x)$ geteilt, dann gilt $g(a)|f(a)$ für jede Konstante a.

Desweiteren wird jedes Polynom vom Grad n eindeutig durch $(n+1)$ Interpolationswerte bestimmt.
Ein Algorithmus zur Bestimmung der Faktoren eines Polynoms $f(x)$ ist damit:

(1) $n = \deg(f)$. Falls $n = 1$ return f.

(2) Setze $k = 1$.

(3) Suche von $f(x)$ Faktoren $g(x)$ vom Grad k, durch Betrachten aller $(k+1)$-Faltungskombinationen der ganzzahligen Faktoren der Werte $f(0), f(1), \ldots, f(k)$.

(4) Generiere für jede solche Kombination ein eindeutiges Polynom $g(x)$ vom Grad k, das durch diese Punkte geht.

(5) Falls gilt $g(x)$ teilt $f(x)$, dann wurde ein Faktor gefunden. Fahre fort $f(x)/g(x)$ zu faktorisieren.

(6) Wenn alle Kombinationen ausgeschöpft sind, setze $k = k + 1$.

(7) Falls $k \leq \frac{n}{2}$, dann gehe zu Schritt (3).

Beispiel:
Man faktorisiere $f(x) = x^3 + 6x^2 + 11x + 6$.

$k = 1$. (Linearfaktoren)

	Teiler
$f(0) = 6$	$\pm 1, \pm 2, \pm 3, \pm 6$
$f(1) = 24$	$\pm 1, \pm 2, \pm 3, \pm 4, \pm 6, \pm 8, \pm 12 \pm 24$

Man probiert alle Paare $(g(0), g(1))$ durch, wobei $g(0)$ aus der ersten und $g(1)$ aus der zweiten Liste genommen wird.

Interpolation

$$\left.\begin{array}{rcl} g(0) & = & 1 \\ g(1) & = & 1 \end{array}\right\} \Longrightarrow g(x) = 1 \quad \text{ein trivialer Faktor von } f$$

$$\left.\begin{array}{rcl} g(0) & = & 1 \\ g(1) & = & -1 \end{array}\right\} \Longrightarrow g(x) = -2x + 1 \quad \text{kein Teiler von } f$$

$$\left.\begin{array}{rcl} g(0) & = & 1 \\ g(1) & = & 2 \end{array}\right\} \Longrightarrow g(x) = x + 1 \quad \text{teilt } f$$

$$f(x) = (x+1)(x^2 + 5x + 6)$$

Nun muß $x^2 + 5x + 6$ faktorisiert werden. Entweder man fährt mit dem Algorithmus von Kronecker fort oder man benutzt die Formel zur Lösung von quadratischen Gleichungen. Man erhält die Lösungen -3 und -2. Damit ergibt sich

$$f(x) = (x+1)(x+2)(x+3).$$

Analyse dieses Beispiels: In diesem Beispiel sind $8 \times 16 = 128$ mögliche Linearfaktoren von f zu prüfen. Dies muß immer gemacht werden, wenn f irreduzibel ist. Daher hatte man hier Glück, gleich beim dritten Versuch einen Linearfaktor zu finden.

Entwurf einer Komplexitätsanalyse für den Algorithmus von Kronecker

- Falls $f(i)$ eine Primzahl $f(i) = p$ ist, so hat diese wenigstens 4 Teiler: $\pm 1, \pm p$.

- Falls $f(i)$ keine Primzahl ist, so hat sie mindestens 6 Teiler: $\pm 1, \pm p, \pm f(i)$, wobei p eine Primzahl ist.

Daher ist die Zahl der Polynominterpolationen, die der Algorithmus zu leisten hat, um die Irreduzibilität eines Polynoms vom Grad n zu zeigen, ungefähr

$$\begin{aligned}\sum_{k=1}^{\frac{n}{2}} 6^{k+1} &= 6\sum_{k=1}^{\frac{n}{2}} 6^k \\ &= 6\left(\frac{6^{\frac{n}{2}+1}-6}{6-1}\right) \approx 6^{\frac{n}{2}+1}\end{aligned}$$

Also zeigt der Algorithmus ein exponentielles Wachstum.
Tatsächlich ist die Situation gerade so schlecht wie hier, da jede Interpolation vom Grad k ungefähr k^2 Multiplikationen benötigt und die dabei erzeugten Koeffizienten länger als ein Wort werden können ($\log^2$(max. Koeffizientengröße)).
Daher ist dieser Algorithmus praktisch unbrauchbar.
Die Lösung dieses Problems in $\mathbb{Z}$ ist nicht in $\mathbb{Z}$ durchführbar. Man muß dieses Problem mit Homomorphismen auf ein anderes Problem transformieren.

8.1.2 Probabilistische Algorithmen

In diesem Kapitel wird das Konzept der probabilistischen Algorithmen verwendet, das hier eingeführt wird.

Dazu zunächst eine Definition für *deterministische Algorithmen* nach Knuth:

1. Eine Berechnungsmethode ist ein Quadrupel (Q, I, O, f) mit $I \subset Q$, $O \subset Q$, $f : Q \longrightarrow Q$ und $f(p) = p' \, \forall p \in I$. Q ist der Zustand der Berechnung, I die Eingabe, O die Ausgabe und f die Berechnungsregeln.
   - Jedes $x \in I$ ergibt eine Folge von Berechnungen $x_0, x_1, \ldots$, die durch $x_{k+1} = f(x_k)$, $k \geq 0$, $x_0 = x$ definiert ist.
   - Die Folge terminiert nach K Schritten (K: kleiner als die k's, so daß $x_k \in O$ gilt)
2. Ein Algorithmus ist eine Berechnungsmethode, die nach endlich vielen Schritten für alle $x \in I$ terminiert.
3. Ein deterministischer Algorithmus ist eine formale Beschreibung für eine *endliche*, *definite* Prozedur, deren Ausgabe bezüglich der Eingaben *eindeutig* ist.

Die mittlere Berechnungszeit hängt mit der Verteilung der Eingaben zusammen.

Probabilistische Algorithmen beziehen sich nun auf die drei hervorgehobenen Wörter in der obigen Definition. Nach Rabin gibt es 3 Arten von probabilistische Algorithmen.

1. **Algorithmen erster Art**
   - Bei mindestens einem Schritt der Prozedur werden einige Zahlen zufällig gewählt (nicht definit).
   - Sonst: deterministisch.

   Als eine Konsequenz hängt die mittlere Berechnungszeit mit gleichlangen Folgen von Zufallszahlen zusammen.

2. **Algorithmen zweiter Art**
   - Gleich wie Algorithmen erster Art (nicht definit).
   - Zusätzlich: Die Ausgabe ist mit einer Wahrscheinlichkeit von $(1-\epsilon)$ korrekt, wobei ϵ sehr klein ist (nicht eindeutig).

3. **Algorithmen dritter Art**
   - Gleich wie Algorithmen erster Art (nicht definit).
   - Eine Folge von Zufallswahlen kann (mit einer Wahrscheinlichkeit von $\epsilon \to 0$) unendlich sein, d.h. der Algorithmus muß nicht terminieren (nicht endlich). Trotzdem ist die mittlere Berechnungszeit endlich.

Bei probabilistischen Algorithmen wird die asymptotische Berechnungsdauer (Komplexität) durch die mittlere Berechnungszeit ersetzt.

Beispiel 1: Näheste Paar Algorithmus
(Algorithmus erster Art)
Seien $x_1, \ldots, x_n$ n Punkte im k-dimensionalen Raum $\mathbb{R}^k$. Die Aufgabe ist nun das näheste Paar x_i, x_j (oder eines davon) zu finden, so daß

$$d(x_i, x_j) = \min d(x_p, x_q) \qquad 1 \le p < q \le n$$

gilt, wobei d die gewöhnliche Abstandsfunktion in $\mathbb{R}^k$ ist.
Die klassische Idee dazu stammt von Yuval: Man betrachte Hüllen der Punkte $S = \{x_1, \ldots, x_n\}$ und sucht das näheste Paar in diesen Hüllen.
Rabin wählt zufällig eine Teilmenge von Punkten.

Beispiel 2: Primzahltest
(Algorithmus zweiter Art)
(Hier wird der Algorithmus benutzt, der weiter unten bei der Faktorisierung von Polynomen unter Benutzung probabilistischer Methoden vorgestellt wird). Primzahltests für große Zahlen (z.B. Lenstra's Methode oder Adleman's Methode) sind sehr teuer und können für dieses Problem nicht benutzt werden.
Die ursprüngliche Idee stammt von Miller (Bedingung von Miller).
Sei n eine ganze Zahl.$W_n(b)$ bezeichne die folgende Bedingung für eine ganze Zahl $1 < b < n$:

(a) $b^{n-1} \not\equiv 1 \bmod n$, oder
(b) $\exists i$, so daß $2^i | (n-1)$ und $1 < \gcd(b^m - 1, n) < n, \quad m = \frac{n-1}{2^i}$ gilt.

Wird die Bedingung $W_n(b)$ für ein b erfüllt, so ist n eine zusammengesetzte Zahl. Man sagt, b sei ein Zeuge für die Teilbarkeit von n.
Der Beweis dafür ist [Bet88]. Die Idee zu dem Beweis ist, daß wenn (a) gilt, dann ist die Fermatsche Bedingung verletzt und wenn (b) gilt, dann hat n einen echten Teiler. Die Irrtumswahrscheinlichkeit ist $\frac{1}{2^\mu}$. Wendet man diese Art von Tests an, erhält man die sogenannten „Pseudo-Primzahlen“.
Die Idee von Rabin liegt in der Tatsache, daß eine zusammengesetzte Zahl n viele Zeugen für die Teilbarkeit besitzt. Aufgrund dieser Tatsache wird durch die probabilistische Methode der Test beschleunigt.
In dem Algorithmus wird zu einem gegebenen n und $\epsilon > 0$ eine Zufallswahl von μ Zahlen

$$1 < b_1, \ldots, b_\mu < n$$

durchgeführt. Ist $\log_2(\frac{1}{\epsilon}) \leq \mu$, dann gibt der Algorithmus mit einer Wahrscheinlichkeit größer als $(1-\epsilon)$ die korrekte Antwort. Der folgende Test-Algorithmus wird eine gegebene Anzahl mal wiederholt, um die Wahrscheinlichkeit für eine korrekte Antwort zu verbessern.

Rabin Test für ganze Zahlen

Eingabe:	n,ungerade ganze Zahl > 1
Ausgabe:	entweder „n ist wahrscheinlich prim“
	oder „n ist zusammengesetzt“
Schritt 1:	wähle a zufällig aus $1 < a < n$
Schritt 2:	zerlege n in $2^l m$
	$n - 1 = 2^l m$, m ungerade

Schritt 3: [Teste]
$b = a^m \bmod n$, $i = 0$
while $b \neq -1$ **and** $b \neq 1$ **and** $i < l$
do { $b = b^2 \bmod a$, $i = i + 1$ }
Schritt 4: [Entscheide]
if $i > 0$ **and** $b = 1$
then „n ist zusammengesetzt "
else „n ist wahrscheinlich prim"

Beispiel 3: Irreduzible Polynome
(Algorithmus dritter Art)

repeat
1. generiere zufälliges Polynom $g \in GF(p)[x]$ vom Grad n
2. teste, ob g irreduzibel

until Test ist wahr

Dieser Algorithmus muß nicht notwendigerweise terminieren (d.h. er ist nicht endlich), da unendlich viele Folgen von zufällig gewählten Polynomen über $GF(p)$ exisitieren, die alle reduzibel sind. Jedoch geht die Wahrscheinlichkeit dafür mit wachsender Zahl von Versuchen gegen 0. Die maximale Zeit für diesen Algorithmus ist $t_{max} = \infty$. In Wirklichkeit aber ist die mittlere zu erwartende Zeit $\bar{t} = O(n^3 \log(p))$.
Der Irreduzibilitätstest basiert auf dem folgenden Satz:

Satz 1
Seien $l_1, \ldots, l_k$ alle Primteiler von n. Mit der Bezeichnung $\frac{n}{l_i} = m_i$ gilt:
Ein Polynom $g(x) \in GF(p)[x]$ vom Grad n ist über $GF(p)$ irreduzibel

$$\iff \begin{array}{ll} (a) & g(x) | (x^{p^n} - x) \\ (b) & \gcd(g(x), x^{p^{m_i}} - x) = 1, \quad 1 \leq i \leq k. \end{array}$$

Durch diesen Satz wird der zweite Schritt im Algorithmus in 2 Teile zerlegt:

(2.1) falls $g(x)|(x^{p^n} - x)$, dann ist Test 1 erfolgreich

(2.2) falls $\gcd(g(x), x^{p^{m_i}} - x) = 1$ für alle m_i, dann ist Test 2 erfolgreich.

Bemerkungen:

1. Diese Methode erzeugt irreduzible Polynome über GF(p) für einen gegebenen Grad n. Die Eingabeparameter zu diesem Algorithmus sind n und die Primzahl p.

2. Effiziente irreduzible Polynome sind wichtig zur Darstellung von endlichen Körpern, denn es gilt für ein irreduzibles Polynom g vom Grad n über $GF(p)$

$$GF(p^n) \cong \mathbb{Z}_p[x]/\langle g(x)\rangle,$$

wobei $\langle g\rangle$, das von g generierte Ideal ist. (Dies ist ähnlich zu der Darstellung von algebraischen Erweiterungen $Q(\alpha) \cong Q[\alpha]/\langle g(\alpha)\rangle$).
Daher braucht man effiziente Irreduzibilitätstests, weil in diesen Körpern eine Arithmetik (modulo p) und (modulo g) benutzt wird, wobei p eine Primzahl und g ein irreduzibles Polynom ist.

8.2 Algorithmen zur quadratfreien Zerlegung von Polynomen

8.2.1 Der univariate Fall

Definition 2
Sei R ein ZPE-Ring. Ein Polynom $P \in R[x]$ heißt **quadratfrei** *(SQFR), wenn P keinen Teiler bzw. Faktor mit einer Vielfachheit ≥ 2 besitzt.*

Das Problem, eine quadratfreie Zerlegung für das Polynom P zu finden, läuft darauf hinaus, Polynome $P_1, P_2, \ldots, P_k$ zu bestimmen, so daß

$$P = P_1 P_2^2 \cdots P_k^k, \quad P_k \neq 1$$

gilt, wobei jedes P_i quadratfrei und der $\gcd(P_i, P_j) = 1$ für $i \neq j$, $1 \leq i, j \leq k$ ist.
Die quadratfreie Zerlegung von Polynomen ist sehr hilfreich bei der Faktorisierung von Polynomen, der Partialbruchzerlegung und der Integration rationaler Funktionen.

Satz 3 (Fundamentalsatz)
Sei R ein ZPE-Ring. Ist das Polynom $P = P_1 P_2^2 \cdots P_k^k \in R[x]$ primitiv in x, wobei alle Polynome P_i $(1 \leq i \leq k)$ quadratfrei und paarweise teilerfremd sind, dann gilt

$$\gcd(P, \frac{dP}{dx}) = P_2 P_3^2 \cdots P_k^{k-1}.$$

Beweis:
Sei $D = \gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x})$. Für jedes P_i kann man auch $P = P_i^i Q$ schreiben, d.h. es gilt

$$\frac{\mathrm{d}P}{\mathrm{d}x} = i P_i^{i-1} \left(\frac{\mathrm{d}P_i}{\mathrm{d}x}\right) Q + P_i^i \left(\frac{\mathrm{d}Q}{\mathrm{d}x}\right).$$

Aus dieser Gleichung folgt, daß P_i^{i-1} ein Teiler von $\frac{\mathrm{d}P}{\mathrm{d}x}$ ist und da $\gcd(P_i, P_j) = 1$ ($i \neq j \leq k$) gilt, daß das Produkt $\prod_{i=2}^{k} P_i^{i-1}$ ein Teiler von D ist. Damit ist nun

$$\frac{\mathrm{d}P}{\mathrm{d}x} = \left(\prod_{i=2}^{k} P_i^{i-1}\right) \left(\sum_{i=1}^{k} i \left(\frac{\mathrm{d}P_i}{\mathrm{d}x}\right) \prod_{j \neq i} P_j\right).$$

Sei S ein Teiler von einem P_j. Dann teilt S kein anderes P_i ($i \neq j$), da $\gcd(P_i, P_j) = 1$ gilt. P_j ist quadratfrei und primitiv bezüglich x und es ist der $\gcd(P_j, \frac{\mathrm{d}P_j}{\mathrm{d}x}) = 1$, d.h. S ist kein Teiler von $\frac{\mathrm{d}P_j}{\mathrm{d}x}$. Dies bedeutet, daß S jeden Term der Summe, bis auf $j\frac{\mathrm{d}P_j}{\mathrm{d}x}$, teilt. Damit ist S kein Teiler der Summe $\sum_{i=1}^{k} i \left(\frac{\mathrm{d}P_i}{\mathrm{d}x}\right) \prod_{j \neq i} P_j$ und es ist der $\gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x}) = \prod_{i=2}^{k} P_i^{i-1}$. □

Ein einfacher Algorithmus zur quadratfreien Zerlegung stammt von Horowitz:

Algorithmus A

$$\begin{array}{ll}
C_1 \leftarrow \gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x}) & \{C_1 = P_2 P_3^2 \cdots P_k^{k-1}\} \\
D_1 \leftarrow \frac{P}{C_1} & \{D_1 = P_1 P_2 \cdots P_k\} \\
\textbf{for } i = 1 \textbf{ step } 1 \textbf{ until } C_i = 1 \textbf{ do} & \{C_i = P_{i+1} P_{i+2}^2 \cdots P_k^{k-i}\} \\
 & \{D_i = P_i P_{i+1} \cdots P_k\} \\
\quad C_{i+1} \leftarrow \gcd(C_i, \frac{\mathrm{d}C_i}{\mathrm{d}x}) & \{C_{i+1} = P_{i+2} P_{i+3}^2 \cdots P_k^{k-i-1}\} \\
\quad D_{i+1} \leftarrow \frac{C_i}{C_{i+1}} & \{D_{i+1} = P_{i+1} \cdots P_k\} \\
\quad P_i \leftarrow \frac{D_i}{D_{i+1}} &
\end{array}$$

Eine einfache Verbesserung dieses Algorithmus erreicht man durch die beiden folgenden Korollare zum Satz 3:

Korollar 4
Es sei $P' = \frac{dP}{dx}$ und $G = \gcd(P, P')$. Dann gilt

$$\begin{aligned}
\frac{P'}{G} - \left(\frac{P}{G}\right)' = \ & P_1 \quad (P_2' P_3 \cdots P_k + 2 P_2 P_3' \cdots P_k + \ldots \\
+ \ & (k-1) P_2 P_3 \cdots P_{k-1} P_k').
\end{aligned}$$

Beweis:
Aus Satz 3 folgt

$\frac{P'}{G} = P_1'P_2\cdots P_k + 2P_1P_2'P_3\cdots P_k + \ldots kP_1\cdots P_{k-1}P_k'$,
$\frac{P}{G} = P_1P_2\cdots P_k$,
$\left(\frac{P}{G}\right)' = P_1'P_2\cdots P_k + \ldots + P_1\cdots P_{k-1}P_k'$. □

Korollar 5
Es sei $P' = \frac{dP}{dx}$ *und* $G = \gcd(P, P')$*. Dann gilt*

$$\gcd(\frac{P}{G}, \left(\frac{P'}{G}\right) - \left(\frac{P}{G}\right)') = P_1.$$

Beweis:
Mit $\frac{P}{G} = P_1P_2\cdots P_k$ und $\frac{P'}{G} - \left(\frac{P}{G}\right)' = P_1Q$, wobei $Q = P_2'P_3\cdots P_k + \ldots + (k-1)P_2P_3\cdots P_{k-1}P_k'$ ist, folgt, daß P_1 den gcd teilt.
Um zu sehen, daß P_1 der gcd ist, genügt es zu zeigen, daß $\gcd(P_2\cdots P_k, Q) = 1$. Sei $P^* = P_2P_3^2\cdots P_k^{k-1}$. Dann folgt mit Satz 3

$$G^* = \gcd(P^*, \frac{\mathrm{d}P^*}{\mathrm{d}x}) = P_3P_4^2\cdots P_k^{k-2},$$

wobei die entsprechenden Kofaktoren teilerfremd sind, d.h. der $\gcd(P^*/G^*, \frac{\mathrm{d}P^*}{\mathrm{d}x}/G^*) = 1$. Es gilt nun aber gerade $P^*/G^* = P_2\cdots P_k$ und $\frac{\mathrm{d}P^*}{\mathrm{d}x}/G^* = Q$. □

Algorithmus B zur quadratfreien Zerlegung

$G \leftarrow \gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x})$	$\{G = P_2P_3^2\cdots P_k^{k-1}\}$
$C_1 \leftarrow \frac{P}{G}$	$\{C_1 = P_1P_2\cdots P_k\}$
$D_1 \leftarrow \frac{\mathrm{d}P}{\mathrm{d}x}/G - \frac{\mathrm{d}C_1}{\mathrm{d}x}$	$\{D_1 = P_1\frac{\mathrm{d}P_2}{\mathrm{d}x}P_3\cdots P_k + \cdots\}$
	$\{\cdots + (j-1)P_1\cdots P_{j-1}\frac{\mathrm{d}P_j}{\mathrm{d}x}P_{j+1}\cdots P_k + \cdots\}$
	$\{\cdots + (k-1)P_1\cdots P_{k-1}\frac{\mathrm{d}P_k}{\mathrm{d}x}\}$
for $i = 1$ **step** 1 **until** $C_i = 1$ **do**	$\{C_i = P_iP_{i+1}\cdots P_k\}$
	$\{D_i = P_i(\frac{\mathrm{d}P_{i+1}}{\mathrm{d}x}P_{i+2}\cdots P_k + \cdots\}$
	$\{\cdots + (j-i)P_{i+1}\cdots P_{j-1}\frac{\mathrm{d}P_j}{\mathrm{d}x}P_{j+1}\cdots P_k + \cdots\}$
	$\{\cdots + (k-i)P_{i+1}\cdots P_{k-1}\frac{\mathrm{d}P_k}{\mathrm{d}x})\}$
$\quad P_i \leftarrow \gcd(C_i, D_i)$	
$\quad C_{i+1} \leftarrow \frac{C_i}{P_i}$	
$\quad D_{i+1} \leftarrow \frac{D_i}{P_i} - \frac{\mathrm{d}C_{i+1}}{\mathrm{d}x}$	

Benutzt man die Schreibweise $(G, A^*, B^*) \leftarrow \gcd(A, B)$ mit $G = \gcd(A, B)$, $A^* = A/G$ und $B^* = B/G$ (Kofaktoren), so vereinfacht sich der Algorithmus zu:

$$(W, C_1, D_1') \leftarrow \gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x})$$

for $i = 1$ **step** 1 **until** $C_i = 1$ **do**

$$(P_i, C_{i+1}, D_{i+1}') \leftarrow \gcd(C_i, D_i' - \frac{\mathrm{d}C_i}{\mathrm{d}x})$$

Dabei wurde $D_i' \equiv D_i + \frac{dC_i}{dx}$ statt D_i verwendet. In diesem Algorithmus werden alle gemachten gcd-Berechnungen benutzt. Es bleibt außer W im Initialisierungsschritt keine Berechnung ungenutzt. Dies ist der Unterschied zum vorhergehenden Algorithmus.

Komplexitätsanalyse:

Algorithmus A: $O(kn^3 \log^2 n)$

Algorithmus B: $O(n \log^2 n)$
$n = \log($ Koeffizientenschranke = größte Koeffizient in P_i für alle i)

8.2.2 Multivariater SQFR-Algorithmus (EZSQFR Algorithmus)

Diese Methode basiert auf dem multivariaten EZ-Algorithmus, der im vorhergehenden Kapitel beschrieben wurde.
Vor der Beschreibung dieses Algorithmus, wird noch ein einfacher Trick vorgestellt, mit dem man eventuell erheblich Zeit einsparen kann:
Ein gegebenes multivariates Polynom kann bereits bezüglich einer speziellen Variable quadratfrei sein. Kann dies nun mit relativ geringem Aufwand festgestellt werden, bevor man die gesamte Maschinerie zur quadratfreien Zerlegung anwirft, so ist es möglich erhebliche Berechnungszeit einzusparen.
Ein solcher Test zur Zeiteinsparung besteht aus:

(a) Auswertung des Polynoms durch zufällig gewählte „gültige" Punkte (siehe folgende Lemmata).

(b) Berechnung des gcd und der Ableitung aus dem erhaltenen univariaten Polynom bezüglich der Hauptvariablen.

(c) Überprüfung, ob der gcd eine ganze Zahl ist. Ist dies erfüllt, so ist das ursprüngliche Polynom quadratfrei.

Die Korrektheit dieses Testes wird durch das folgende Lemma veranschaulicht:

Lemma 6
Sei $b = (b_1, \ldots, b_\nu)$ eine beliebige gültige Auswertung für $P(x, y_1, \ldots, y_\nu) \in \mathbb{Z}[x, y_1, \ldots, y_\nu]$. Ist $P_b(x, b_1, \ldots, b_\nu)$ quadratfrei in $\mathbb{Z}[x]$, dann ist auch P quadratfrei.

Beweis:
Man nehme an P sei nicht quadratfrei, d.h. $P = QR^2$. Dann erhält man mit dem Auswertungshomomorphismus $P_b = Q_b R_b^2$, das ebenfalls nicht quadratfrei sein kann. □

Die Nützlichkeit dieses Testes beruht auf dem nächsten Lemma.

Lemma 7
Ist P ein quadratfreies multivariates Polynom in $\mathbb{Z}[x, y_1, \ldots, y_\nu]$, dann kann eine Menge von ganzen Zahlen $b = (b_1, \ldots, b_\nu)$ gewählt werden, so daß b eine gültige Auswertung für P ist (d.h. $deg_x(P) = deg(P_b(x))$). $P_b(x)$ ist ebenfalls quadratfrei.

Beweis:
Sei $P = P_1 P_2 \cdots P_k$ mit $\gcd(P_i, P_j) = 1$ für $i \neq j$.
$P_b(x)$ ist quadratfrei gdw. $P_{i_b}(x)$ für alle i quadratfrei ist und $\gcd(P_{i_b}(x), P_{j_b}(x)) = constant$ für $i \neq j$, was äquivalent ist zu Res_x [1] $(P_{i_b}(x), P_{j_b}(x)) \neq 0$.
Sei $R(x, y_1, \ldots, y_\nu) = \prod_i \mathrm{Res}_x(P_i, \frac{\mathrm{d}P_i}{\mathrm{d}x}) \prod_{i<j} \mathrm{Res}_x(P_i, P_j)$. Dann ist $R \neq 0$, weil P quadratfrei ist. $P_b(x)$ ist quadratfrei gdw. $R_b(x) \neq 0$. Es reicht nun zu bemerken, daß es nur endlich viele ganzzahlige Werte für $b_1, \ldots, b_\nu$ gibt, so daß $R_b(x) = 0$ oder $(ldc(P))_b = 0$. □

Durch die Endlichkeit der ganzzahligen Wurzeln von $R(x, y_1, \ldots, y_\nu)$ ist die Wahrscheinlichkeit für eine gültige, aber „ungünstige" Wahl der Menge b nicht allzu groß.

Nun wird ein Lemma bewiesen, mit dessen Hilfe auf die Methode zur Lösung des Spezialfalles im EZGCD-Algorithmus, wenn dieser zur Berechnung von $N = \gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x})$ im zweiten Schritt aufgerufen wird, verzichtet werden kann.

Lemma 8
Sei P primitiv in $\mathbb{Z}[y_1, \ldots, y_\nu][x]$ und $R = \gcd(P, \frac{dP}{dx})$. Dann gilt

$$\gcd(R, \left(\frac{dP}{dx}\right)/R) = 1.$$

[1] $Res_x(A, B)$ ist die Resultante von A, B bezüglich der Variablen x. Siehe dazu [GCL92] oder auch [Bet88].

Beweis:
Sei $P = P_1 P_2^2 \cdots P_k^k$ und $Q = \frac{\mathrm{d}P}{\mathrm{d}x}$. Aus Satz 3 folgt $R = P_2 P_3^2 \cdots P_k^{k-1}$. Angenommen es gilt $D = \gcd(R, Q/R) \neq 1$, dann ist $Q = RDQ''$ für ein bestimmtes Q''. Da D ein Teiler von R ist, muß es einen irreduziblen Faktor $C \neq 1$ für ein bestimmtes P_i $(i \geq 2)$ geben. Aber dieses C teilt offensichtlich auch P/R, so daß $P = RCP''$ für ein bestimmtes P''. Damit ist RC ein Teiler von P und Q im Widerspruch dazu, daß R der gcd ist. Also muß der $\gcd(R, Q/R) = 1$ sein. □

Algorithmus C: EZSQFR

Eingabe: Primitives, multivariates Polynom $P(x, y_1, \ldots, y_\nu)$ aus $\mathbb{Z}[x, y_1, \ldots, y_\nu]$ mit der Hauptvariable x.

Ausgabe: Eine Liste von Polynomen $P_1, P_2, \ldots, P_k \in \mathbb{Z}[x, y_1, \ldots, y_\nu]$, so daß $P = P_1 P_2^2 \cdots P_k^k$ für ein $k \geq 1$ gilt, mit $\gcd(P_i, \frac{\mathrm{d}P_i}{\mathrm{d}x}) = 1$, $\gcd(P_i, P_j) = 1$ für $i \neq j$ und $P_k \neq 1$.

(1) Führe den Test zur Prüfung auf Quadratfreiheit zweimal mit zufälligen gültigen Auswertungen durch. Ist der Test erfolgreich, dann gebe P zurück, andernfalls fahre fort.

(2) Wende den EZGCD-Algorithmus auf P und $\frac{\mathrm{d}P}{\mathrm{d}x}$ an. Sei b die gültige und günstige Auswertung für P und p eine günstige Primzahl für $P_b(x)$, die für den EZGCD-Algorithmus gebraucht wird. Man erhält $N = \gcd(P, \frac{\mathrm{d}P}{\mathrm{d}x})$ und $L = P/N$.

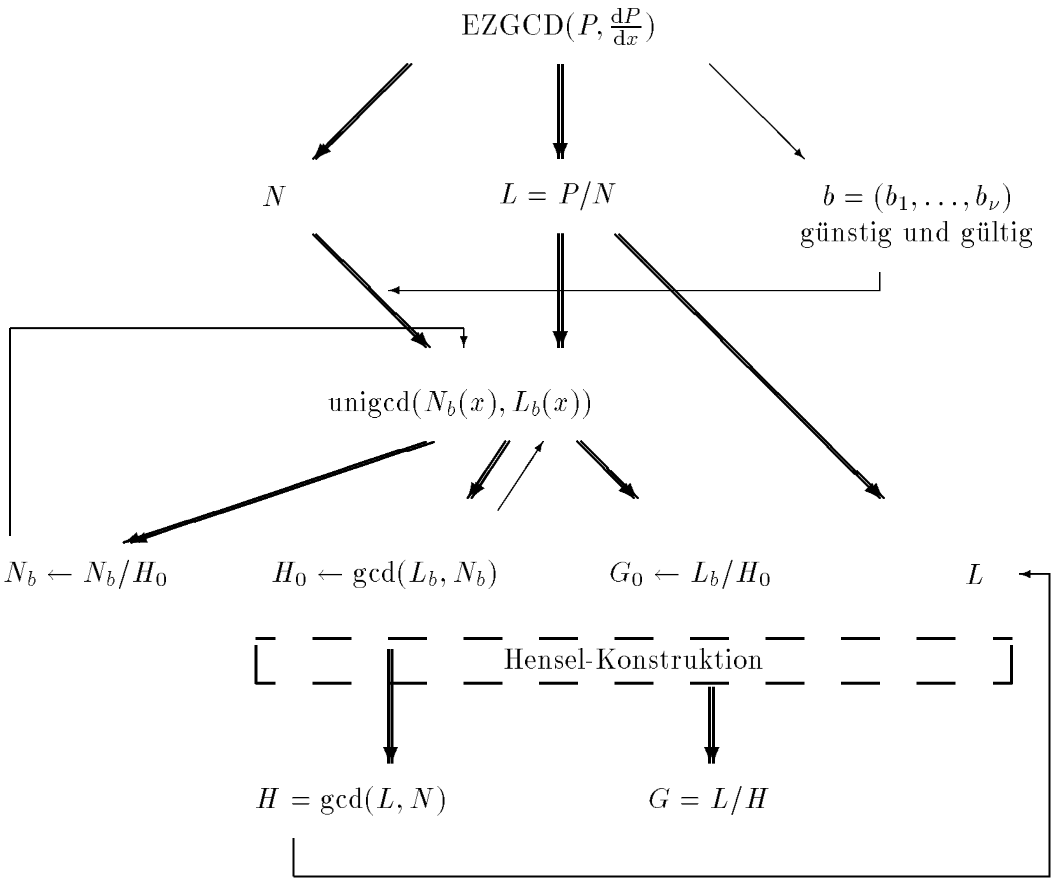
(3) Setze
$$\begin{aligned} L_b &\leftarrow L(x, b_1, \ldots, b_\nu), \\ N_b &\leftarrow N(x, b_1, \ldots, b_\nu), \\ i &\leftarrow 1. \end{aligned}$$

(4) Wende einen univariaten gcd-Algorithmus auf L_b, N_b an.
$$\begin{aligned} H_0 &\leftarrow \gcd(L_b, N_b), \\ G_0 &\leftarrow L_b/H_0, \\ N_b &\leftarrow N_b/H_0. \end{aligned}$$
Setze $L_b \leftarrow H_0$.

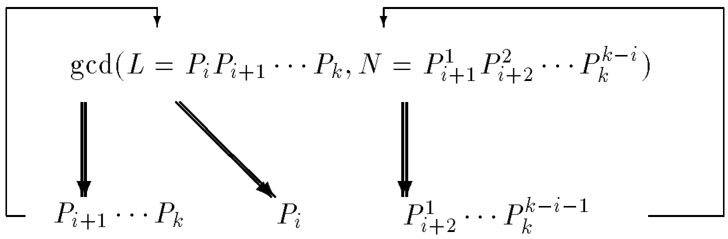
(5) Falls $G_0 = 1$, dann setze $P_i \leftarrow 1$. Andernfalls wende den multivariaten EZ-Algorithmus (Hensel) auf L, G_0 und H_0 an, um multivariate Polynome G und H mit $L = GH$ über $\mathbb{Z}$ zu bekommen.
Setze $P_i \leftarrow G$, $L \leftarrow H$ und $i \leftarrow i + 1$.

(6) Falls $N_b \neq 1$, dann gehe zu Schritt (4). Andernfalls setze $k \leftarrow 1$, $P_k \leftarrow L$ und gebe $P_1, P_2, \ldots, P_k$ zurück.

Schema des EZSQFR-Algorithmus



Induktionsschritt:



$\Longrightarrow$: Berechnungsergebnis $\longrightarrow$: benutzt als Argument in der nächsten Iteration

Der Algorithmus C (EZSQFR) entspricht einer verbesserten Version des Algorithmus A für den univariaten Fall. Es ist auch möglich, einen Algorithmus entsprechend dem Algorithmus B für den univariaten Fall zu erstellen.
Hier noch ein knapper Entwurf für solch einen Algorithmus.

Algorithmus D

(a) $(N, L, D) \leftarrow \mathrm{EZGCD}(P, \frac{\mathrm{d}P}{\mathrm{d}x})$

(b) Benutze *gültige* und *günstige* Auswertungspunkte $b = \{b_1, \ldots, b_\nu\}$ für P aus der obigen EZGCD-Berechnung, um $L_b(x) = L(x, y_1, \ldots, y_\nu)$ und $D_b(x) = D(x, y_1, \ldots, y_\nu)$ zu erhalten.
Setze $i \leftarrow 1$.

(c) Schleife:

$(P_0, L_b, D_b) \leftarrow \mathrm{unigcd}(L_b(x), D_b(x) - \frac{\mathrm{d}L_b(x)}{\mathrm{d}x})$
Falls $P_0 = 1$, dann setze $P_i \leftarrow 1$. Andernfalls wende die *Hensel-Konstruktion* auf L,P_0 und L_b an, um $L = P_i L_{neu}$ zu erhalten. Setze $L \leftarrow L_{neu}$ und $i \leftarrow i + 1$.

Falls $L \neq 1$, dann gehe zurück in die Schleife. Andernfalls setze $P_i \leftarrow L$ und gebe $P_1, P_2, \ldots, P_k$ zurück, wobei $k \leftarrow i$.

Wie im Algorithmus B, bleibt auch hier kein Teilergebnis ungenutzt.

Dieser – wie auch die vorhergehenden – Algorithmen zeigen, daß die Kosten für eine quadratfreie Zerlegung im wesentlichen gleich den Kosten zur Berechnung der gcd sind.

8.3 Faktorisierung modulo p — Berlekamp-Algorithmus

Für diesen Algorithmus sind einige Ergebnisse aus der Algebra grundlegend. Diese notwendigen Ergebnisse mit ihren Beweisen werden nun zuerst aufgelistet.

Proposition 9
Seien $u(x), v(x)$ zwei Polynome aus $\mathbb{Z}_p[x]$ (p eine Primzahl). Dann gilt

$$(u(x) + v(x))^p = u(x)^p + v(x)^p.$$

Beweis:
Es ist

$$\begin{aligned}(u(x)+v(x))^p &= u(x)^p + \binom{p}{1} u(x)^{p-1} v(x) + \dots \\ &\quad + \binom{p}{p-1} u(x) v(x)^{p-1} + v(x)^p \\ &= u(x)^p + v(x)^p,\end{aligned}$$

weil alle Binomialkoeffizienten $\binom{p}{1} \cdots \binom{p}{p-1}$ Vielfache von p sind und die Arithmetik in diesem Ring (modulo p) ist. $\square$

Proposition 10 (Kleiner Fermatscher Satz)
Für eine Primzahl p und eine beliebige ganze Zahl a gilt

$$a^p \equiv a \bmod p.$$

Beweis:
Induktion über a (OBdA sei a positiv). Die Behauptung ist wahr für $a = 0$ und $a = 1$. Angenommen es gilt $(a-1)^p \equiv (a-1) \bmod p$. Dann gilt

$$\begin{aligned} a = a - 1 + 1 &\stackrel{\text{Prop.9}}{\Longrightarrow} a^p = (a-1+1)^p = (a-1)^p + 1^p \\ &\stackrel{\text{Ind.An.}}{\Longrightarrow} a^p = a - 1 + 1 = a. \quad \square \end{aligned}$$

Korollar 11
Jede ganze Zahl modulo p ist eine Wurzel von $x^p - x$. Daher gilt

$$x^p - x = (x-0)(x-1)\cdots(x-(p-1)).$$

Proposition 12
Für ein Polynom $v(x) \in \mathbb{Z}_p[x]$ (p eine Primzahl) gilt

$$(v(x))^p \equiv v(x^p) \bmod p.$$

Beweis:
Es sei $v(x) = v_n x^n + v_{n-1} x^{n-1} + \cdots + v_0 = v_n x^n + \tilde{v}_{n-1}(x)$. Dann ist

$$v(x)^p \equiv (v_n x^n + \tilde{v}_{n-1}(x))^p \stackrel{\text{Prop.9}}{\equiv} (v_n x^n)^p + (\tilde{v}_{n-1}(x))^p.$$

Durch Anwenden dieses Ergebnis auf $\tilde{v}_{n-1}(x)^p \ldots$ erhält man schließlich

$$\begin{aligned} v(x)^p &\equiv (v_n x^n)^p + (v_{n-1}x^{n-1})^p + \ldots + (v_0)^p \\ &\stackrel{\text{Prop.10}}{\equiv} v_n x^{np} + v_{n-1}x^{(n-1)p} + \ldots + v_0 \equiv v(x^p). \quad \square \end{aligned}$$

Benutzt man die Ergebnisse aus dem vorhergehenden Abschnitt, dann genügt es, für das Problem der Faktorisierung von Polynomen nur quadratfreie Polynome zu betrachten. Daher nimmt man für ein $u(x) \in \mathbb{Z}_p[x]$ an, es sei durch

$$u(x) = p_1(x)p_2(x)\cdots p_r(x)$$

mit paarweise teilerfremden p_i darstellbar.
Nun stellt sich das Problem:

Wie findet man die p_i zu einem gegebenen $u(x)$?

Berlekamp hat dieses Problem durch Anwenden des chinesischen Restesatzes gelöst. Ist $(s_1, s_2, \ldots, s_r)$ ein beliebiges r-Tupel ganzer Zahlen modulo p, so folgt aus dem CRS, daß es ein eindeutiges Polynom $v(x)$ gibt mit

$$\begin{array}{l} v(x) \equiv s_1 \pmod{p_1(x), p} \\ \quad\cdots \\ v(x) \equiv s_r \pmod{p_r(x), p} \\ \deg(v) < \deg(p_1) + \deg(p_2) + \ldots + \deg(p_r) = \deg(u). \end{array} \tag{8.1}$$

Das Polynom $v(x)$ ermöglicht es, die Faktoren von $u(x)$ zu erhalten. Für $r \geq 2$ und $s_1 \neq s_2$ ist der $\gcd(u(x), v(x) - s_1)$ durch p_1 teilbar, aber nicht durch p_2. Man kann also aus den Lösungen $v(x)$ von (8.1) Information über die Faktoren von $u(x)$ gewinnen. Daher muß man diese Lösungen näher untersuchen.
Erstens erfüllt $v(x)$ die Bedingung $v(x)^p \equiv s_j^p \equiv s_j \equiv v(x) \pmod{p_j(x), p}$ für $1 \leq j \leq r$ und deshalb

$$v(x)^p \equiv v(x) \pmod{u(x), p}, \quad \deg(v) < \deg(u). \tag{8.2}$$

Zweitens gilt Proposition 10 und sein Korollar

$$x^p - x \equiv (x-0)(x-1)\cdots(x-(p-1)) \pmod{p};$$

und ersetzt man x durch $v(x)$, so gilt auch

$$v(x)^p - v(x) \equiv (v(x)-0)(v(x)-1)\cdots(v(x)-(p-1)) \pmod{p}. \tag{8.3}$$

Erfüllt $v(x)$ die Gleichung (8.2), dann folgt daraus, daß $u(x)$ die linke Seite von (8.3) teilt und damit muß jeder irreduzible Faktor von $u(x)$ einen der p teilerfremden Faktoren der rechten Seite von (8.3) teilen. Mit anderen Worten, alle Lösungen von (8.2)

müssen von der Form (8.1) für bestimmte $s_1, s_2, \ldots, s_r$ sein. Also *gibt es genau* p^r *Lösungen von* (8.2).
Die Lösungen $v(x)$ der Kongruenz (8.2) liefern einen Schlüssel zur Faktorisierung von $u(x)$.
Jetzt stellt sich die Frage:

Ist es einfacher, die Gleichung (8.2) *zu lösen, als* $u(x)$ *zu faktorisieren?*

Die Antwort ist ja. Der eigentliche Grund dafür ist, daß die Menge der Lösungen von (8.2) unter der Addition abgeschlossen ist.
Sei $n = \deg(u)$ und

$$x^{pk} \equiv q_{k,n-1}x^{n-1} + \ldots + q_{k,1}x + q_{k,0} \pmod{u(x), p}.$$

Damit kann man eine $(n \times n)$-Matrix, die sogenannte Q-Matrix (nach der Schreibweise von Berlekamp) konstruieren

$$Q = \begin{pmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,n-1} \\ \vdots & \vdots & & \vdots \\ q_{n-1,0} & q_{n-1,1} & \cdots & q_{n-1,n-1} \end{pmatrix}.$$

Dann ist $v(x) = v_{n-1}x^{n-1} + \ldots + v_1 x + v_0$ eine Lösung von (8.2) genau dann, wenn

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

letztere Gleichung gilt genau dann, wenn

$$\begin{aligned} v(x) &= \sum_j v_j x^j = \sum_j \sum_k v_k q_{k,j} x^j = \sum_k v_k \sum_j q_{k,j} x^j \\ &\equiv \sum_k v_k x^{pk} = v(x^p) \equiv v(x)^p \pmod{u(x), p}. \end{aligned}$$

Die Eigenvektoren von $Q \pmod p$ zum Eigenwert 1 sind die Lösungen von (8.2).
Aus diesen Bemerkungen und Ergebnissen folgt nun der Algorithmus von Berlekamp.

Algorithmus von Berlekamp: $\boldsymbol{u(x)} \in \mathbb{Z}_p[\boldsymbol{x}]$

B1. Prüfe, ob $u(x)$ quadratfrei ist. Ist dies nicht erfüllt, dann führe eine quadratfreie Zerlegung von $u(x)$ durch und wende diesen Algorithmus auf jeden erhaltenen Faktor an. Andernfalls fahre fort.

B2. Bilde die Matrix Q.

B3. Bestimme eine Basis der Eigenvektoren von Q zum Eigenwert 1. Mit anderen Worten, diagonalisiere die Matrix $(Q-I)$, wobei $I=(\delta_{ij})$ die $(n\times n)$-Einheitsmatrix ist. Bestimme mit $n=\deg(u)$ den Rang $n-r$ der Matrix und linear unabhängige Vektoren $v^{[1]},\ldots,v^{[r]}$, so daß $v^{[j]}(Q-I)=(0,0,\ldots,0)$ für $1\le j\le r$ gilt. (Der erste Vektor $v^{[1]}$ kann immer als (1,0,...,0) gewählt werden und stellt die triviale Lösung $v^{[1]}(x)=1$ von (8.2) dar. Dies entspricht der Tatsache, daß ganze Zahlen immer Lösungen von (8.2) sind). r ist die Anzahl irreduzibler Faktoren von $u(x)$. Daher ist für $r=1$ das Polynom irreduzibel und die Prozedur terminiert.

B4. Berechne die $\gcd(u(x),v^{[2]}(x)-s)$ für $0\le s<p$. Das Ergebnis ist eine nichttriviale Faktorisierung von $u(x)$, da $(v^{[2]}(x)-s)\neq 0$ ist und einen Grad kleiner als $\deg(u)$ hat und man beweisen kann, daß wenn $v(x)$ die Gleichung (8.2) erfüllt, dann

$$u(x)=\prod_{0\le s<p}\gcd(v(x)-s,u(x))$$

gilt. Hat man noch nicht alle r Faktoren erhalten, so kann man weitere Faktoren durch Benutzung der anderen Eigenvektoren und Berechnen von $\gcd(v^{[k]}(x)-s,w(x))$ für $0\le s<p$, wobei $w(x)$ alle Faktoren, die bisher gefunden wurden umfaßt, erhalten. Dies wird für $k=3,4,\ldots$ solange fortgeführt, bis man r Faktoren gefunden hat.

Die Prozedur von Berlekamp faktorisiert modulo p ein beliebiges Polynom vom Grad n in $O(n^3+prn^2)$, wobei r die Anzahl der Faktoren ist. (Man kann beweisen, daß der mittlere Wert für r näherungsweise $\ln n$ ist). Diese Prozedur ist sehr schnell für kleine p ($p=2$ oder 3), für $p>25$ kann sie dagegen sehr aufwendig sein.

Diagonalisierung von $(Q-I)$

Diese Aufgabe wird vom Nullraum-Algorithmus übernommen. Schritt (B.3) des Berlekamp-Algorithmus enthält 'finde den Nullraum von $(Q-I)$'.
Angenommen, A sei eine $n\times n$ Matrix über einem *Körper*, dessen Rang $(n-r)$ es zusammen mit den Eigenvektoren $v^{[k]}$, so daß

$$v^{[k]}A=(0,0,\ldots,0)\qquad k=1,\ldots,r$$

ist, zu bestimmen gilt. Ein Algorithmus für diese Berechnung kann von der Tatsache Gebrauch machen, daß man eine Spalte von A mit einem von Null verschiedenen Faktor multiplizieren und zu einer beliebigen Spalte das Vielfache einer anderen Spalte addieren kann, ohne dabei den Rang und die Vektoren $v^{[k]}$ zu verändern. Diese Transformationen laufen darauf hinaus A durch AB zu ersetzen, wobei B eine nichtsinguläre Matrix ist.

Nullraum-Algorithmus

Eingabe: A, eine $n \times n$ Matrix mit Elementen a_{ij} aus einem Körper, $0 \le i, j \le n$.

Ausgabe: r linear unabhängige Vektoren $v^{[k]}$ $(k = 1, \ldots, r)$, die $v^{[k]}A = (0, 0, \ldots, 0)$ erfüllen, wobei $n - r$ der Rang der Matrix A ist.

N1. [Initialisierung]
Setze $c_0 \leftarrow c_1 \leftarrow \cdots \leftarrow c_{n-1} \leftarrow -1$, $r \leftarrow 0$. (Während der Berechnung sind die c_j nur ≥ 0, falls $a_{c_j j} = -1$ und alle anderen Einträge der Zeile c_j gleich Null sind).

N2. [Schleife über k]
Durchlaufe Schritt (N.3) für $k = 0, 1, \ldots, n-1$ und beende dann den Algorithmus.

N3. [Zeilen auf Abhängigkeit untersuchen]
Falls es ein j $(0 \le j < n)$ gibt mit $a_{kj} \ne 0$ und $c_j < 0$, dann: multipliziere Spalte j von A mit $-1/a_{kj}$ (damit wird a_{kj} äquivalent zu -1); addiere $a_{ki} \times$ Spalte j zur Spalte i für alle $i \ne j$; setze $c_j \leftarrow k$. (Da die $a_{sj} = 0$ für alle $s < k$ sind, haben diese Operationen keine Auswirkungen auf die Zeilen $0, 1, \ldots, k-1$ von A.)
Falls es kein j $(0 \le j < n)$ gibt mit $a_{kj} \ne 0$ und $c_j < 0$ dann setze $r \leftarrow r + 1$ und gebe den Vektor

$$v^{[r]} = (v_0, v_1, \ldots, v_{n-1})$$

aus, der durch

$$v_j = \begin{cases} a_{ks}, & \text{falls } c_s = j \ge 0 \\ 1, & \text{falls } j = k \\ 0, & \text{sonst} \end{cases}$$

definiert ist.

Beispiel:
(aus Knuth, Bd. II). Man faktorisiere das Polynom $u(x) = x^8 + x^6 + 10x^4 + 10x^3 + 8x^2 + 2x + 8$ modulo 13.

Berlekamp-Schritt 1: $\gcd(u, u') = 1$. Also ist $u(x)$ quadratfrei.

Berlekamp-Schritt 2: Konstruktion der Q-Matrix. Hier ist es eine 8×8 Matrix. Die erste Zeile ist immer $(1, 0, 0, \ldots, 0)$ für $x^0 \bmod u = 1$. Die zweite Zeile steht für $x^{13} \bmod u(x)$. Allgemein kann $x^k \bmod u(x)$ leicht (für kleine k) wie folgt bestimmt werden: Ist

$$u(x) = x^n + u_{n-1}x^{n-1} + \ldots + u_1 x + u_0$$

und

$$x^k \equiv a_{k,n-1}x^{n-1} + \ldots + a_{k,0} \pmod{u(x), p}$$

dann ist

$$\begin{aligned} x^{k+1} &\equiv a_{k,n-1}x^n + \ldots + a_{k,0}x \\ &\equiv a_{k,n-1}(-u_{n-1}x^{n-1} - \ldots - u_1 x - u_0) \\ &\quad +a_{k,n-2}x^{n-1} + \ldots + a_{k,0}x \\ &= a_{k+1,n-1}x^{n-1} + \ldots + a_{k+1,0} \end{aligned}$$

mit

$$a_{k+1,j} = a_{k,j-1} - a_{k,n-1}u_j.$$

Dabei wird $a_{k,-1}$ auf Null gesetzt, so daß $a_{k+1,0} = -a_{k,n-1}u_0$ ist. Diese Schieberegister-Rekurrenz macht es leicht $x^1, x^2, \ldots \pmod{u(x), p}$ zu berechnen. In einem Programm erhält man ein ein-dimensionales Array $(a_{n-1}, \ldots, a_1, a_0)$ und setzt wiederholt $t \leftarrow a_{n-1}$, $a_{n-1} \leftarrow (a_{n-2} - tu_{n-1}) \bmod p \ldots a_1 \leftarrow (a_0 - tu_1) \bmod p$ und $a_0 \leftarrow (-tu_0) \bmod p$. Man erhält damit die Folge der Koeffizienten von $x^k \pmod{u(x), 13}$:

k	$a_{k,7}$	$a_{k,6}$	$a_{k,5}$	$a_{k,4}$	$a_{k,3}$	$a_{k,2}$	$a_{k,1}$	$a_{k,0}$
0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	0
2	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	1	0	0	0	0
5	0	0	1	0	0	0	0	0
6	0	1	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0
8	0	12	0	3	3	5	11	5
9	12	0	3	3	5	11	5	0
10	0	4	3	2	8	0	2	8
11	4	3	2	8	0	2	8	0
12	3	11	8	12	1	2	5	7
13	11	5	12	10	11	7	1	2

Daher ist $(2, 1, 7, 11, 10, 12, 5, 11)$ die zweite Zeile von Q. Ähnlich bestimmt man $x^{26} \pmod{u(x), 13}, \ldots, x^{91} \pmod{u(x), 13}$ und erhält

$$
\begin{aligned}
Q &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 7 & 11 & 10 & 12 & 5 & 11 \\ 3 & 6 & 4 & 3 & 0 & 4 & 7 & 2 \\ 4 & 3 & 6 & 5 & 1 & 6 & 2 & 3 \\ 2 & 11 & 8 & 8 & 3 & 1 & 3 & 11 \\ 6 & 11 & 8 & 6 & 2 & 7 & 10 & 9 \\ 5 & 11 & 7 & 10 & 0 & 11 & 7 & 12 \\ 3 & 3 & 12 & 5 & 0 & 11 & 9 & 12 \end{pmatrix}, \\
Q - I &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 7 & 11 & 10 & 12 & 5 & 11 \\ 3 & 6 & 3 & 3 & 0 & 4 & 7 & 2 \\ 4 & 3 & 6 & 4 & 1 & 6 & 2 & 3 \\ 2 & 11 & 8 & 8 & 2 & 1 & 3 & 11 \\ 6 & 11 & 8 & 6 & 2 & 6 & 10 & 9 \\ 5 & 11 & 7 & 10 & 0 & 11 & 6 & 12 \\ 3 & 3 & 12 & 5 & 0 & 11 & 9 & 11 \end{pmatrix}.
\end{aligned}
$$

Bemerkung: *Q enthält in seinen Zeilen die Werte für $x^{pk} \pmod{u(x), p}$. Diese können zur Berechnung von bestimmten potenzierten Werten von x benutzt werden.*

Berlekamp-Schritt 3: Nun wird der Nullraum-Algorithmus mit $A = Q - I$ ausgeführt.
$k = 0$: Ausgabe des Vektors $v^{[1]} = (1, 0, 0, 0, 0, 0, 0, 0)$.
$k = 1$: Im Schritt N3 kann j entweder $0, 2, 3, 4, 5, 6$ oder 7 sein. Die Wahl ist willkürlich. Da $a_{15} = 12 = -1$ ist, wird $j = 5$ zur Berechnung von Hand gewählt. Die Spaltenoperationen in Schritt (3) ändern dann A zu der Matrix

$$
\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 \\ 11 & 6 & 5 & 8 & 1 & 4 & 1 & 7 \\ 3 & 3 & 9 & 5 & 9 & 6 & 6 & 4 \\ 4 & 11 & 2 & 6 & 12 & 1 & 8 & 9 \\ 5 & 11 & 11 & 7 & 10 & 6 & 1 & 10 \\ 1 & 11 & 6 & 1 & 6 & 11 & 9 & 3 \\ 12 & 3 & 11 & 9 & 6 & 11 & 12 & 2 \end{pmatrix}
$$

(Das eingerahmte Element $\boxed{12}$ deutet hier an, daß $c_5 = 1$ ist.) Für $k = 2$ kann man $j = 4$ wählen. So fortfahrend, erhält man folgende Matrizen, die alle den gleichen

Nullraum wie $Q - I$ haben:

$$\overset{k=2}{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 & 0 \\ 8 & 1 & 3 & 11 & 4 & 9 & 10 & 6 \\ 2 & 4 & 7 & 1 & 1 & 5 & 9 & 3 \\ 12 & 3 & 0 & 5 & 3 & 5 & 4 & 5 \\ 0 & 1 & 2 & 5 & 7 & 0 & 3 & 0 \\ 11 & 6 & 7 & 0 & 7 & 0 & 6 & 12 \end{pmatrix}} \quad \overset{k=3}{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 & 0 \\ 0 & \boxed{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 9 & 8 & 9 & 11 & 8 & 8 & 5 \\ 1 & 10 & 4 & 11 & 4 & 4 & 0 & 0 \\ 5 & 12 & 12 & 7 & 3 & 4 & 6 & 7 \\ 2 & 7 & 2 & 12 & 9 & 11 & 11 & 2 \end{pmatrix}}$$

$$\overset{k=4}{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 & 0 \\ 0 & \boxed{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{12} \\ 1 & 10 & 4 & 11 & 4 & 4 & 0 & 0 \\ 8 & 2 & 6 & 10 & 11 & 11 & 0 & 9 \\ 1 & 6 & 4 & 11 & 2 & 0 & 0 & 10 \end{pmatrix}} \quad \overset{k=5}{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{12} & 0 & 0 & 0 \\ 0 & \boxed{12} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \boxed{12} \\ \boxed{12} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 5 & 5 & 0 & 9 \\ 12 & 9 & 0 & 0 & 11 & 9 & 0 & 10 \end{pmatrix}}$$

Nun ist jede Spalte, die keinen gerahmten Eintrag besitzt vollständig mit Nullen aufgefüllt. Daher liefert der Algorithmus für $k = 6$ und $k = 7$ zwei weitere Vektoren:

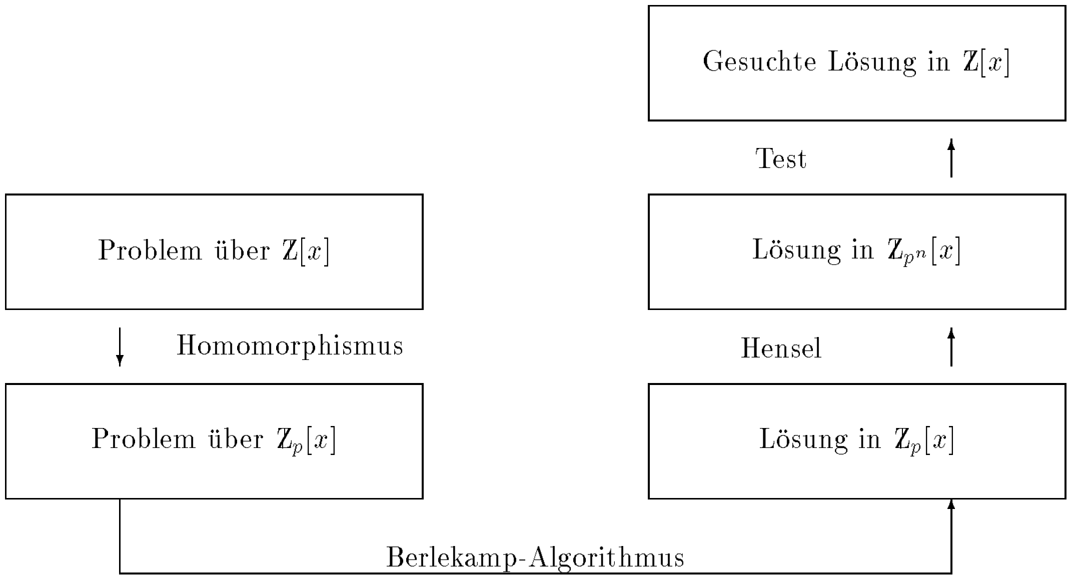
$$v^{[2]} = (0, 5, 5, 0, 9, 5, 1, 0), \quad v^{[3]} = (0, 9, 11, 9, 10, 12, 0, 1).$$

Da die Berechnung 3 linear unabhängige Vektoren erzeugt hat, muß $u(x)$ genau 3 irreduzible Faktoren haben. *Berlekamp-Schritt 4:* Die Berechnung der $\gcd(u(x), v^{[2]}(x) - s)$ für $0 \leq s < 13$ mit $v^{[2]}(x) = x^6 + 5x^5 + 9x^4 + 5x^2 + 5x$ ergibt $x^5 + 5x^4 + 9x^3 + 5x + 5$ bei $s = 0$ und $x^3 + 8x^2 + 4x + 12$ bei $s = 2$; für alle anderen Werte von s ergibt der gcd 1. Daher liefert $v^{[2]}$ nur 2 der 3 Faktoren und man muß weiter die $\gcd(v^{[3]}(x) - s, x^5 + 5x^4 + 9x^3 + 5x + 5)$ berechnen mit $v^{[3]}(x) = x^7 + 12x^5 + 10x^4 + 9x^3 + 11x^2 + 9x$.Man erhält $x^4 + 2x^3 + 3x^2 + 4x + 6$ für $s = 6$ und $x + 3$ für $s = 8$; für alle anderen Werte erhält man 1. Damit ist die vollständige Faktorisierung

$$u(x) = (x^4 + 2x^3 + 3x^2 + 4x + 6)(x^3 + 8x^2 + 4x + 12)(x + 3).$$

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

Idee:



Es sind bereits bis auf „Hensel" und „Test" alle Schritte bekannt.

„Hensel": Man bestimmt zu einer gegebenen Faktorisierung des Polynoms f modulo p (Berlekamp) die Faktorisierung von f modulo p^k mit genügend großem p^k.

„Test": Die Faktorisierung modulo p^k enthält „fast immer" die richtigen Informationen, die benötigt werden. Durch Kombination dieser Faktoren zu Produkten erhält man die Faktorisierung in $\mathbb{Z}[x]$.

Schritt „Hensel":
Sei f normiert und habe die Faktorisierung $f = gh \pmod{p}$ mit g und h teilerfremd. Dies kann man auch als $f_1 = g_1 h_1 \pmod{p^1}$ schreiben. Gesucht ist nun die entsprechende Faktorisierung $f_k = g_k h_k \pmod{p^k}$ mit genügend großem p^k.
Offensichtlich ist $g_2 \equiv g_1 \pmod{p}$. Daher kann man g_2 auch als $g_2 = g_1 + p\hat{g}_2$ schreiben. Das gleiche gilt für f und h, so daß man für $f_2 = g_2 h_2 \pmod{p^2}$ nun

$$f_1 + p\hat{f}_2 = (g_1 + p\hat{g}_2)(h_1 + p\hat{h}_2) \pmod{p^2}$$

erhält. Da $f_1 = g_1 h_1 \pmod{p}$ gilt, kann diese Gleichung in der Form

$$\frac{f_1 - g_1 h_1}{p} + \hat{f}_2 = \hat{g}_2 h_1 + \hat{h}_2 g_1 \pmod{p}$$

geschrieben werden. Die linke Seite dieser Gleichung ist bekannt und die rechte Seite erhält man durch Anwenden des erweiterten euklidischen Algorithmus auf g_1 und h_1, die zueinander teilerfremd sind. Man findet dadurch die Polynome $\hat{g}_2$ und $\hat{h}_2$ mit $\deg(\hat{g}_2) < \deg(g_1)$ und $\deg(\hat{h}_2) < \deg(h_1)$. Daraus kann man g_2 und h_2 bestimmen.
So fortfahrend erhält man $g_3 \equiv g_2 \pmod{p^2}$ und damit $g_3 = g_2 + p^2\hat{g}_3$. Das gleiche gilt für f und h, so daß man für $f_3 = g_3h_3 \pmod{p^3}$

$$f_2 + p^2\hat{f}_3 = (g_2 + p^2\hat{g}_3)(h_3 + p^2\hat{h}_3) \pmod{p^3}$$

erhält. Da $f_2 = g_2h_2 \pmod{p^2}$ gilt, kann diese Gleichung in der Form

$$\frac{f_2 - g_2h_2}{p^2} + \hat{f}_3 = \hat{g}_3h_2 + \hat{h}_3g_2 \pmod{p}$$

geschrieben werden. Mehr noch, da $g_2 \equiv g_1 \pmod{p}$ und $h_2 \equiv h_1 \pmod{p}$ gilt, kann diese Gleichung auf

$$\frac{f_2 - g_2h_2}{p^2} + \hat{f}_3 = \hat{g}_3h_1 + \hat{h}_3g_1 \pmod{p}$$

vereinfacht werden. Man wendet wieder den erweiterten euklidischen Algorithmus auf g_1 und h_1 an und erhält wie zuvor die Gradbedingung $\deg(\hat{g}_3) < \deg(g_1)$ und $\deg(\hat{h}_3) < \deg(h_1)$. Daraus kann man g_3 und h_3, von g_2 und h_2 ausgehend, bestimmen. Dies führt man für jede Potenz p^k von p solange fort bis p^k groß genug ist.
(Der euklidische Algorithmus wird immer auf dieselben Polynome angewandt; es reicht deshalb eine Anwendung vollkommen aus.)

Algorithmus Hensel (linear)

Eingabe: f, g_1, h_1, p, k (f normiert);

Ausgabe: g_k, h_k;

$g \leftarrow g_1$;
$h \leftarrow h_1$;
$\{greciprocal, hreciprocal\} \leftarrow \text{ExtendedEuclid}(g_1, h_1, p)$;
for $i = 2$ **to** k **do** $\quad discrepancy \leftarrow \dfrac{f - gh \pmod{p^i}}{p^{i-1}}$;
  $gcorrection \leftarrow discrepancy * hreciprocal \pmod{p, g_1}$;
  $hcorrection \leftarrow discrepancy * greciprocal \pmod{p, h_1}$;
  $g \leftarrow g + p^{i-1}gcorrection$;
  $h \leftarrow h + p^{i-1}hcorrection$;
return g, h;

Dies ist nichts anderes als eine einfache Version des allgemeinen Hensel-lifting, das im vorhergehenden Kapitel ausführlich beschrieben ist. Nun folgt der vollständige Faktorisierungsalgorithmus (der Einfachheit halber) unter der Annahme $f(x)$ sei normiert und quadratfrei.

Algorithmus zur Faktorisierung in einer Variable

Eingabe: $f(x) \in \mathbb{Z}[x]$, univariat, normiert und quadratfrei.

Ausgabe: $f(x) = g_1(x) \cdots g_l(x)$, wobei die g_i's irreduzibel sind.

```
do    p ← prime();                              [ gibt jeweils eine verschiedene Prim-
                                                  zahl aus einer Liste (2,3,5,...) zurück]
      f_p ← f   (mod p);
      while deg(f) ≠ deg(f_p) or gcd(f_p, f_p') ≠ 1;
{g_1, ..., g_n} ← Berlekamp(f_p, p);
if n = 1 then return {f};
k ← log_p(2 * Landau-Mignotte-Schranke(f));    [ siehe Bemerkung 1 ]
factors ← Hensel(f, {g_1, ..., g_n}, p, k);
result ← {};
for all g_i, i = 1, ..., n do                   [ siehe Bemerkung 2 ]
      g ← g_i;
      if g teilt f then
            result ← result ∪ {g};
            factors ← factors \ {g};
            n ← n − 1;
combine ← 2;
while 2 * combine ≤ n do
      for each combine-Teilmenge E of factors do
            g ← ∏_{h∈E} h   (mod p^k);
      if g teilt f then
            result ← result ∪ {g};
            factors ← factors \ {g};
            n ← n − combine;
            if 2 * combine > n then
                  exit aus beiden Schleifen;
      combine ← combine + 1;
if factors ≠ ∅ then
      g ← ∏_{h∈factors} h   (mod p^k);
      result ← result ∪ {g};
return result.
```

Bemerkung 1: Die Landau-Mignotte-Schranke liefert eine ganze Zahl, so daß alle Koeffizienten von allen Faktoren von f kleiner als diese Zahl sind. Diese wird mit 2 multipliziert, um die Wahl des Vorzeichens berücksichtigen zu können.

Bemerkung 2: Mit diesem Schritt beginnt der „Test". Ist ein Faktor ein Teiler von f, dann ist er wirklich ein Teiler, andernfalls testet man verschiedene Produktkombinationen der g_i: 2 Faktoren, 3 Faktoren, ...

Leitkoeffizient:
Es wurde f als normiert angenommen. Man kann ein Polynom $f(x) = \sum_{i=0}^{n} a_i x^i$ immer so transformieren, daß es normiert ist:

(1) multipliziere f mit a_n^{n-1}

(2) ersetze x durch $y = \frac{x}{a_n}$,

wodurch das Polynom übergeht in

$$\sum_{i=0}^{n} a_i a_n^{n-1-i} x^i = x^n + a_{n-1}x^{n-1} + a_n a_{n-2} x^{n-2} + \ldots + a_n^{n-1} a_0.$$

Der Nachteil dieser Methode ist, daß die Landau-Mignotte-Schranke wächst und damit wächst auch die Größe aller ganzen Zahlen.
Zur Minimierung dieses Problems gibt es technische Spitzfindigkeiten.

Im folgenden wird eine alternative Beschreibung der Prozedur zur Faktorisierung univariater Polynome über den ganzen Zahlen gegeben, die weniger technisch ist und an die aktuelle Implementierung anschließt.

Berlekamp-Hensel-Algorithmus

A: quadratfreies, normiertes, positives ganzzahliges und univariates Polynom.

D1. Berechne die getrennte Grad Faktorisierung $((d_1, A_1)\ldots(d_k, A_k))$ von A für eine geeignete Primzahl p und die Menge C der Grade der Faktoren. Jedes $A_i = \prod_{j=1}^{k_i} f_{ij}$ ist das Produkt irreduzibler Faktoren f_{ij} vom gleichen Grad d_i für alle j.

D2. Für jedes i mit $d_i = \deg(A_i)$, nimm A_i in die Ausgabeliste L auf. Wende den Berlekamp-Algorithmus (mit p) auf die verbleibenden Faktoren an. Man erhält $G = (f_{i_1}, \ldots, f_{i_{k_i}})$. Die f_{ij} sind irreduzible Faktoren von A_i über $GF(p)$.

D3. Wende die Hensel-Konstruktion auf die Liste G an. Man erhält die Liste G^* der Faktoren modulo p^N, wobei p^N die Schranke $a_n d 2^n$ für die Größe aller Koeffizienten übertrifft.

D4. Kombiniere die so „hochgehobenen“ Faktoren modulo p^N von G^* zu ganzzahligen Faktoren der Liste L unter Benutzung der Menge C aus Schritt D1.

8.5 Faktorisierung in mehreren Variablen

Das Prinzip ist dem Fall univariater Polynome sehr ähnlich, aber es ist technisch sehr viel komplizierter.
Die Idee hierzu ist wieder eine Auswertung für alle Variablen bis man ein univariates Polynom (d.h. einen Auswertungshomomorphismus) findet. Der Algorithmus ist wie folgt:

proc $factorise(f, r, s)$;

> (Die Variablen des Polynoms f seien $x_1, \ldots, x_r$ und s der Index der Variable die ersetzt werden soll, wobei angenommen wird, daß $x_{s+1}, \ldots, x_{r-1}$ bereits durch ganzzahlige Werte ersetzt wurden. f wird als quadratfrei angenommen und beim ersten Aufruf als normiert bezüglich x_r.)

> **if** $s = 0$ **then return** $factorise\text{-}univariate(f)$;
> **do** $v := random()$;
> $fv := f_{x_s := v}$;
>   **while** $\deg(f) \neq \deg(fv)$ **or** $\gcd(fv, fv') \neq 1$;
> $\{g_1, \ldots, g_n\} := factorise(fv, r, s-1)$;
> **if** $n = 1$ **then return** $\{f\}$;
> $k := \deg(f, x_s)/2$;
> $factors := Hensel(f, \{g_1, \ldots, g_n\}, x_s - v, k)$;
> $result := \{\}$;
> **for each element** g **of** $factors$ **do**
>   **if** g teilt f **then**
> $result := result \cup \{g\}$;
> $factors := factors \setminus \{g\}$;
> $n := n - 1$;
> $combine := 2$;
> **while** $combine < n$ **do**
>   **for each** $combine-$Teilmenge E **of** $factors$ **do**
> $g := \prod_{h \in E} h \pmod{(x_s - v)^k}$;

if g teilt f **then**
$result := result \cup \{g\}$;
$factors := factors \setminus E$;
$n := n - combine$;
$combine := combine + 1$;
if $factors \neq \emptyset$ **then**
$g := \prod_{h \in factors} h \pmod{(x_s - v)^k}$;
$result := result \cup \{g\}$;
return $result$.

Der Algorithmus ist rekursiv in den Variablen.
Bemerkungen:

- $random()$ erzeugt einen zufälligen Wert für die Auswertung. Kleine Werte werden bevorzugt.
- Der Hensel-Teilalgorithmus hebt eine Faktorisierung modulo $(x_s - v)$ an auf eine Faktorisierung modulo $(x_s - v)^k$. Die Hauptmodifikation zu dem weiter vorne gegebenen Algorithmus liegt in der Ersetzung von p durch $y - r$.
- Die Landau-Mignotte-Schranke wird nicht benötigt, da der Grad eines Faktors immer kleiner als der Grad des Initialpolynoms ist.

Probleme:

- *Schlechte Reduktion*, aber die Situation ist hier viel besser, wie die für univariate Polynome. Dies erklärt sich durch den folgenden Satz:

  Satz 13 (Hilbertscher Irreduzibilitätssatz)
  Sei $f(y_1, \ldots, y_s, x_1, \ldots, x_r)$ ein irreduzibles Polynom aus $\mathbb{Z}[y_1, \ldots, y_s, x_1, \ldots, x_r]$ und $M(N)$ die Zahl der Auswertungen $y_1 = b_1, \ldots, y_s = b_s$ mit $|b_i| < N$, so daß das Polynom $f(b_1, \ldots, b_s, x_1, \ldots, x_r)$ reduzibel in $\mathbb{Z}[x_1, \ldots, x_r]$ ist (d.h. die Auswertung ist schlecht). Dann gibt es Konstanten α und C mit $0 < \alpha < 1$ und $M(N) \leq C(2N+1)^{s-\alpha}$.

  Die Anzahl der möglichen Auswertungen ist $P(N) = (2N+1)^s$. Der Satz besagt nun, daß $\lim_{N \to \infty} M(N)/P(N) = 0$ gilt, was nichts anderes bedeutet als, daß die Wahrscheinlichkeit einer ungünstigen Auswertung für genügend große ganze Zahlen gegen 0 strebt. In der Praxis ist eine schlechte Reduktion sehr selten.

- *Schlechte Nullen*, d.h. es ist nicht möglich 0 für den Wert von v zu benutzen. Ein multivariates Polynom muß aus praktischen Gründen als dünn besetztes (sparse) Polynom dargestellt werden. Ein dünn besetztes Polynom bleibt auch modulo x_s^n dünn besetzt, wird aber in x_s dicht besetzt (dense) bei modulo $(x_s-v)^n$ für ein von Null verschiedenes v. Dies ist für die gegebene rekursive Methode nicht weiter schlimm, wirkt sich jedoch sehr nachteilig auf die parallele Version dieses Algorithmus aus.

- *Leitkoeffizient.* Das Problem ist hier sehr viel schlimmer, wie für den univariaten Fall. Die Lösungen hierfür, aber sind viel zu technisch, um hier relevant zu sein.

8.6 Getrennte Grad Faktorisierung

Es ist möglich die Faktorisierungsprozedur zu vereinfachen, um Faktoren modulo p zu erhalten. Dazu benutzt man zwei Ergebnisse der Theorie von Polynomen über endlichen Körpern.

(i) Ein irreduzibles Polynom $q(x)$ vom Grad d ist ein Teiler von $(x^{p^d} - x)$.

(ii) Das gleiche Polynom $q(x)$ ist kein Teiler von $(x^{p^c} - x)$ für $c < d$.

Man kann damit die irreduziblen Faktoren von jedem Grad , durch Anwenden der im folgenden beschriebenen Strategie (distinct degree factorization), separat finden.

(D1) Schließe alle quadratischen Faktoren aus, d.h. mache $u(x)$ quadratfrei. Setze $v(x) \leftarrow u(x)$, $w(x) \leftarrow x$, $d \leftarrow 0$.

(D2) Es gilt $w(x) = x^{p^d} \bmod v(x)$. Alle irreduziblen Faktoren von $v(x)$ sind verschieden (da $u(x)$ quadratfrei ist) und sind vom Grad $> d$.
Falls $d + 1 > \frac{1}{2}\deg(v)$ ist, dann terminiert das Verfahren, da entweder $v(x) = 1$ oder $v(x)$ irreduzibel ist.
Andernfalls setze $d \leftarrow d + 1$, $w(x) \leftarrow w(x)^p \bmod v(x)$.

(D3) Berechne $g_d(x) = \gcd(w(x) - x, v(x))$. $g_d(x)$ ist das Produkt aller irreduziblen Faktoren von $u(x)$ vom Grad d. Falls $g_d(x) \neq 1$, dann ersetze $v(x)$ durch $v(x)/g_d(x)$ und $w(x)$ durch $w(x) \bmod v(x)$.
Berechne die Faktoren von $g_d(x)$, falls $\deg(g_d(x)) > d$ ist und gehe zurück zu Schritt (D2).

Um $g_d(x)$ in seine irreduziblen Faktoren zu zerlegen, kann man das Verfahren von Rabin (im Körper mit p^d Elementen) benutzen oder das einfachere von Cantor-Zassenhaus.

Beispiel:
In dem Beispiel zum Algorithmus von Berlekamp haben die 3 Faktoren verschiedene Grade. Daher würde die getrennte Grad Faktorisierungsprozedur diese direkt, ohne $g_d(x)$ faktorisieren zu müssen, finden.
Aber für $u(x) = (x^3 + \ldots)(x+1)(x+2)$, findet man für $d = 1 \quad g_d(x) = x^2 + 3x + 2$.
Damit ist $\deg(g_d(x)) > 1$ und man muß $g_d(x)$ faktorisieren.

8.7 Probabilistische Verfahren

Man vergleiche dazu zusätzlich [Bet88].

8.7.1 Das Verfahren von Rabin

Dieses Verfahren ist eine Verallgemeinerung des Verfahrens von Berlekamp für endliche Körper $\mathbb{Z}_p$ (p prim).

8.7.1.1 Nullstellenbestimmung in $GF(q)$

Es gilt auch hier die Bezeichnung, wie allgemein üblich, $q = p^n$ für p prim. Man betrachte das Polynom $f(x) \in GF(q)$ mit $\deg(f) = m$.

Bemerkung: Es ist nicht möglich ein Polynom direkt als ein Körperelement darzustellen. Man hat sich daher bei der Bezeichnung $f(x) \in GF(q)$ immer $f(x)$ als $f(x) \in \mathbb{Z}_p[x]/(g(x)) \cong GF(q)$ mit irgendeinem irreduziblen Polynom $g(x) \in \mathbb{Z}_p[x]$ vom Grad n vorzustellen.

(a) Annahme: q ist *ungerade*
Sei
$$f_1(x) = \gcd(f(x), x^{q-1} - 1).$$
Ist $f_1(x) = 1$, dann hat $f(x)$ keine Nullstelle in $GF(q)$, da allgemein für $f_1(x)$ gilt
$$f_1(x) = (x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_k),\ k \le m$$
und es sind alle α_i paarweise verschiedene Lösungen von $f(x) = 0$ in $GF(q)$.
Im nächsten Schritt betrachtet man die Faktorisierung
$$x^{q-1} - 1 = (x^d - 1)(x^d + 1) \quad \text{mit } d = (q-1)/2$$

und versucht, ob $\gcd(f_1(x), x^d - 1)$ ein echter Teiler von $f_1(x)$ ist. Es ist jedoch nicht garantiert, daß dieser gcd von 1 oder von $f_1(x)$ verschieden ist. (Man erhält nur diejenigen Lösungen α_i, die die Gleichung $\alpha_i^d - 1 = 0$ erfüllen, während andere α_i die Gleichung $\alpha_i^d + 1 = 0$ erfüllen). Dieses Problem wird durch Zufallswahl, d.h. durch Substituieren von $(x^d - 1)$ durch $[(x+\delta)^d - 1]$ mit einer zufälligen Wahl von δ gelöst.

(b) q ist *gerade* (d.h. $q = 2^n$)
Anstatt des Polynomes $x^{q-1} - 1$ verwendet man in diesem Fall das Polynom

$$T(x) = x + x^2 + \ldots + x^{2^{n-1}}.$$

(In der Literatur heißt dieses Polynom auch Spur bzw. trace). Da für jedes $\alpha \in GF(2^n)$ aufgrund von $\alpha^{2^n} = \alpha \quad T(\alpha^2) = T(\alpha)$ und des weiteren $T^2(\alpha) = T(\alpha^2)$ (dies liegt am Aufbau dieses speziellen Polynomes) gilt, ist $T^2(\alpha) - T(\alpha) = 0$, also $T(\alpha) \in GF(2)$, d.h. jedes α ist Lösung von $T(x) = 0$ oder $T(x) = 1$.
Ebenso gilt $T(\alpha + \beta) = T(\alpha) + T(\beta)$.
Mit dem Polynom $T(x)$ kann das gleiche Verfahren, wie im Fall p ungerade, genutzt werden.

Bemerkung: Das Verfahren für q gerade basiert auf der Faktorisierung

$$x^{p^n} - x = \prod_{0 \le s < p} (T(x) - s),$$

das für alle p gilt, während das Verfahren für q ungerade auf der Faktorisierung

$$x^{p^n} - x = x(x^{(p^n-1)/2} - 1)(x^{(p^n-1)/2} + 1)$$

beruht, die nur für ungerade Primzahlen gilt.

Algorithmus zur Nullstellenbestimmung in $GF(q)$

[q ungerade; für q gerade verwende $T(x)$ an Stelle von $x^{q-1} - 1$]

(1) $f_1(x) = \gcd(f(x), x^{q-1} - 1)$
Falls $\deg(f_1(x)) = 0$, dann gebe 'keine Nullstelle' zurück.

(2) Wiederhole die Zufallswahl δ in $GF(q)$ solange für

$$f_\delta(x) = \gcd(f_1(x), (x+\delta)^d - 1) \quad \text{mit } d = (q-1)/2,$$

bis $f_\delta \neq 1$ und $f_\delta \neq f_1$.

(3) Setze

$$f_2(x) = \begin{cases} f_\delta(x) & \text{falls } \deg(f_\delta) \le \frac{1}{2}\deg(f_1) \\ f_1(x)/f_\delta(x) & \text{sonst.} \end{cases}$$

Falls $f_2(x)$ linear ist, dann gebe f_2 zurück.

(4) Setze $f_1(x) \leftarrow f_2(x)$. Gehe zurück nach (2).

8.7.1.2 Faktorisierung über $GF(q)$

Es sei $f(x) \in GF(p)$ ein Polynom mit $\deg(f(x)) = n$ und

$$g_m(x) = \gcd(f(x), x^{p^m} - x) \quad \text{für } 1 \le m < n.$$

Es ist $GF(p^m) = \{$ alle Elemente vom Grad 1 und alle Elemente vom Grad i mit $i|m$ über $\mathbb{Z}_p$ $\}$.
Damit ist $g_m(x)$ das Produkt aller irreduziblen Faktoren $h(x)$, welche Teiler von $f(x)$ mit $\deg(h(x))|m$ sind. Daraus ergibt sich folgendes Verfahren:

- Wähle $g_m(x) \neq 1$ mit kleinstem m. Sei $\deg(g_m(x)) = l$. Dann ist

  $$g_m(x) = h_1(x) \cdots h_k(x) \quad \text{mit } k \cdot m = l$$

  und es sind die $h_i(x)$ irreduzible Faktoren vom Grad m.

- Bestimme eine Lösung α von $g_m(x) = 0$ in $GF(p^m)$. Diese Lösung α ist eine Lösung eines dadurch eindeutig festgelegten Faktors h_i.

- Bestimme h_i. Dafür gibt es zwei Verfahren.

Eines davon ist das folgende: $h_i(x)$ ist irreduzibel vom Grad m. Da $\Phi(\xi) = \xi^p$ ein Automorphismus von $GF(p^m)$ (der sogenannte Frobenius-Automorphismus) über dem Körper $\mathbb{Z}_p$ ist, sind die Konjugierten von α

$$\alpha_0 = \alpha, \quad \alpha_1 = \alpha^p, \quad \ldots, \quad \alpha_{m-1} = \alpha^{p^{m-1}}.$$

Das Polynom $h_i(x)$ erhält man dann durch die Berechnung in $GF(p^m)$ von

$$h_i(x) = (x - \alpha_0)(x - \alpha_1) \cdots (x - \alpha_{m-1}).$$

Algorithmus Rabin

$A(x) \in GF(p)$

(1) [Getrennte Grad Faktorisierung] Setze $m = \lfloor \deg(A)/2 \rfloor$ und wähle das erste $l \in \{1, 2, \ldots, m\}$, so daß $g_l(x) = \gcd(A(x), x^{p^l} - x) \neq 1$, sonst return.

(2) [Nun gilt $g_l(x) = \prod_{i=1}^{k} h_i(x)$, $\deg(h_1) = \cdots = \deg(h_k) = l$] Bestimme eine Lösung von $g_l(x)$ in $GF(p^l)$. Der entsprechende Faktor $h_l(x)$ ist gegeben durch $h_i(x) = (x - \alpha) \cdots (x - \alpha^{p^{l-1}})$.

(3) Sammle die Faktoren h_i, setze $A \leftarrow A/h_i$, gehe nach (1).

8.7.2 Das Verfahren von Cantor-Zassenhaus

Dieses Verfahren basiert auf der folgenden Identität. Ist p eine ungerade Primzahl so gilt (g_d ist gleich wie im Abschnitt 8.6)

$$\begin{aligned} g_d(x) = \gcd(g_d(x), t(x)) \quad &\cdot \quad \gcd(g_d(x), t(x)^{(p^d-1)/2} - 1) \\ &\cdot \quad \gcd(g_d(x), t(x)^{(p^d-1)/2} + 1) \end{aligned}$$

für alle Polynome $t(x)$, da $\left(t(x)^{p^d} - t(x)\right)$ ein Vielfaches aller irreduziblen Polynome vom Grad d ist. Man beweist, wählt man $t(x)$ als ein zufälliges Polynom vom Grad $\leq (2d-1)$ aus $GF(p^d)$, so ist

$$\gcd(g_d(x), t(x)^{(p^d-1)/2} - 1)$$

ein nichttrivialer Faktor von $g_d(x)$ in nur rund 50 Prozent der Zeit. Also sollte es nicht lange dauern, alle Faktoren zu bestimmen.
Man kann $t(x)$ als normiert annehmen. Dann kann man im Fall $d = 1 \quad t(x) = x + s$ mit zufälliger Wahl von s verwenden. Dies ergibt einen sehr effizienten Algorithmus [Bet88]. Der Hauptgrund dafür, daß dieses Verfahren effizienter ist als das von Rabin, liegt darin, daß das Verfahren von Rabin auf der Nullstellenbestimmung in einem Erweiterungskörper basiert; das von Cantor-Zassenhaus dagegen die Arithmetik eines Erweiterungskörper vermeidet, d.h. es wird mit $x^{p \cdot i}$ anstatt mit x^{p^i} gearbeitet.

Kapitel 9

Vereinfachung

9.1 Einführung

Beispiel:
Bildet man die Ableitung $\frac{\partial}{\partial x}(ax + xe^{x^2})$ und benutzt dabei die folgenden Regeln für alle x, u, v und jede Konstante c

$\frac{\partial c}{\partial x} \to 0$, $\frac{\partial x}{\partial x} \to 1$, $\frac{\partial(u+v)}{\partial x} \to \frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}$, $\frac{\partial(uv)}{\partial x} \to u \cdot \frac{\partial v}{\partial x} + \frac{\partial u}{\partial x} \cdot v$, $\frac{\partial(u^v)}{\partial x} \to vu^{v-1}\frac{\partial u}{\partial x} + (\log u) \cdot u^v \frac{\partial v}{\partial x}$,

so erhält man

$a \cdot 1 + 0 \cdot x + 1 \cdot e^{x^2} + x(x^2 e^{x^2-1} \cdot 0 + (\log e) \cdot e^{x^2} \cdot (2 \cdot x^1 + (\log x) \cdot x^2 \cdot 0))$.

Führt man dagegen die Ableitung „von Hand“ durch, ergibt sich

$$a + e^{x^2} + 2x^2 e^{x^2} = a + (2x^2 + 1)e^{x^2}.$$

Daran erkennt man die Notwendigkeit von Algorithmen und Techniken, die Ausdrücke automatisch vereinfachen können.
Etwas allgemeiner als in diesem Beispiel, hier eine Liste von Vereinfachungen, die teilweise zwingend oder aber sehr hilfreich in Computer Algebra Systemen sind.

1. Unterdrücken von Klammern: $(((u)v) + w) = uv + w$.

2. Identitäts-Vereinfachung: $0 \cdot u \to 0$, $1 \cdot u \to u$, $\frac{u}{1} \to u$, $v^0 \to 1$ $(v \neq 0)$, $0^w \to 0$ $(w > 0)$, oder bei Matrizen (I Einheitsmatrix) $A \cdot I \to A$.

3. Vorzeichen: $(-u)(-v^3) = uv^3$. Man könnte aber genauso wollen, daß $-(a+b) \to -a - b \ldots$.

4. Numerische Vereinfachungen: $\frac{5}{8} - \frac{1}{8} \to \frac{1}{2}$, $9! \to 362880$. Viele derartige Vereinfachungen sind nicht trivial, wie z.B. $(33282)^{\frac{1}{2}} \sin(\frac{13\pi}{6}) \to \frac{122}{\sqrt{2}}$. Ebenso treten Probleme bei transzendenten Zahlen wie e^e, e, π auf und es stellt sich die Frage nach der Entscheidbarkeit.

5. Assoziativgesetze: $(uv)w + (p+q) \to uvw + p + q$.

6. Ordnung: z.B. bei der Polynomdarstellung.

7. Zusammenfassung gemeinsamer Faktoren: $u + (\frac{2}{3})u \to \frac{5}{3}u$, $2^{x+2} = 4 \cdot 2^x$, $e^{5+\log u} \to e^5 e^{\log u}, \ldots$

8. Operationen auf den Exponenten: $(u^w)^v \to u^{vw}$, $(uv)^w \to u^w \cdot v^w$.

9. Distributivgesetze: $(u+v)w \to uw + vw$, $(p-q)(u+v) \to pu - qu + pv - pv$.

10. Potenzerweiterung: $(a+b)^2 = a^2 + 2ab + b^2$, aber soll man auch $(1+x)^{100}$ auf diese Weise „vereinfachen“?

11. ggT-Vereinfachung: $\frac{4u^4+12u^3+12u^2+4u}{2u^4-2u^3-2u^2+2u} \to \frac{2u+2}{u-1}$, aber soll man dies auch für $\frac{u^{100}-v^{100}}{u-v} \to u^{99} - uv^{98} + \ldots + v^{99}$ durchführen?

12. Rationale Ausdrücke: $\frac{1}{x} - \frac{1}{y} - \frac{y-x}{xy} \to 0$, $x + \frac{x^3}{9} + \frac{x^5}{25} + \frac{x^7}{49} \to \frac{225x^7+441x^5+1225x^3+11025x}{11025}$.

13. Spezielle Argumente: $\sin 0 \to 0$, $\cos 0 \to 1$, $\log 1 \to 0$ oder $2k\pi i$, $\log e \to 1$ oder $2m\pi i + 1$, $\arctan 1 \to \frac{\pi}{4}$ oder $\frac{\pi}{4} + n$.

14. Radikalvereinfachung:
$\sqrt{(x^2-x+1)(x+\sqrt{x-1})} - (x+\sqrt{x-1}\sqrt{x-\sqrt{x-1}}) \to 0$, $(2^{\frac{1}{3}} + 4^{\frac{1}{3}})^3 - 6(2^{\frac{1}{3}} + 4^{\frac{1}{3}}) - 6 \to 0$.

15. Transzendente Identitäten: Vereinfachung oder Komplexitätsvergrößerung von Ausdrücken?
$\cos(-x) = \cos x$, $\log(uv) = \log u + \log v$, $\log(e^{2x} + 2e^x + 1) - 2\log(e^x+1) \to 0$, $\log\tan(\frac{x}{2} - \frac{\pi}{4}) - \sinh^{-1}\tan x \to 0$.

16. Faktorisierung von Polynomen.

...und viele viele andere mögliche Verfahren mehr zur Vereinfachung, wie formales ableiten, differenzieren, integrieren, Matrix-Operationen, Gleichungslösen, Potenzreihenerweiterung, ...

Vereinfachungsprobleme reichen von trivialen Fragestellungen ($a+x$ oder $x+a$) bis zu sehr schwierigen Fragen (wie z.B. die Frage: ist $\pi + e$ rational?).

9.2 Klassifikation von CA-Systemen aufgrund der gewählten Vereinfachung

Die Vereinfachung spielt die Hauptrolle im Systementwurf. Hat man bei den folgenden Punkten

- Datenrepräsentation und Datenstrukturen
- welcher Wechsel der Repräsentation soll automatisch durchgeführt werden
- welche dieser Wechsel verbleiben unter der Kontrolle des Benutzers
- was für zusätzliche Vereinfachungsprozeduren soll das System enthalten

eine Entscheidung getroffen, so ist bereits alles wesentliche entschieden. Die klassischen Beispiele hierzu sind

$$\begin{array}{ccl} & & \text{beibehalten?} \\ & \nearrow & \\ (x+y)^{1000} & & \\ & \searrow & \\ & & \text{ausmultiplizieren?} \\ & & \text{beibehalten?} \\ & \nearrow & \\ x^{1000}-y^{1000} & & \\ & \searrow & \\ & & \text{faktorisieren?} \end{array}$$

Eine Klassifizierung von CA-Systemen, die auf den Möglichkeiten und den Verfahren der Vereinfachung beruht, wurde von J. Moses durchgeführt. Das Kriterium dabei ist: Wie wird ein vom Benutzer gegebener Ausdruck vom CA-System abgeändert und zu was wird dieser erweitert?

(a) **Radikale:** Der Ausdruck wird radikal umgeformt und in eine wohldefinierte interne Darstellung gebracht. Als ein Ergebnis dieser Entscheidung kann man mit einem solchen CA-System nur wohldefinierte Klassen von Ausdrücken, wie Polynome, rationale Funktionen, abgeschnittene Potenzreihen, Poissonreihen, ... handhaben. Für jede dieser Klassen gibt es eine kanonische Form. Daher benötigt man keine algorithmische Vereinfachung. Es wird lediglich eine Vereinfachung gebraucht, um Ausdrücke in die kanonische Form zu bringen. Das bedeutet beispielsweise für multivariate Polynome, daß eine vordefinierte Ordnung auf den Variablen ($x > y > z \ldots$) vorausgesetzt ist und so ein Polynom als ein Polynom bezüglich der führenden Variable betrachtet wird, d.h. z.B. wird

$$3x^2y^2 - 2x^2yz^3 + 5x^2z^2 + 4x - 6y^3z + y^3 + 3y^2 + z^4 + 1$$

dargestellt als

$$(3y^2 - (2z^3)y + 5z^2)x^2 + (4)x + ((-6z + 1)y^3 + (3)y^2 + z^4 + 1).$$

Schwierigkeiten treten z.B. bei $(1 + x)^{1000}$ auf, das erweitert wird, obwohl dies ziemlich unvernünftig ist.
Rationale Funktionen werden als $\frac{P}{P'} \equiv \frac{\text{kanonische Form}}{\text{kanonische Form}}$ dargestellt und eine Summe von Brüchen auf einen Bruch transformiert, und gleichzeitig werden die ggT's der Zähler und Nenner zum Kürzen gesucht. Beispielsweise wird

$$\frac{2x+3}{x^2+2x+5} - \frac{1}{3x+2} + \frac{3x}{x^2+2x+6}$$

auf einen Bruch gebracht, was im Normalfall nicht unbedingt die Lösung ist, die man sich vorgestellt hat.
Diese Systeme sind erweitert worden zur Benutzung allgemeinerer Ausdrücke, wie Exponentialfunktionen, Logarithmen und trigonometrische Funktionen. Es wurde dabei z.B. ein „kanonischer" Ausdruck für $\sin x$ eingeführt:

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i}.$$

Dies führt auf Ausdrücke wie $\int \sin x \cos^{10} x \, \mathrm{d}x = \int \frac{1}{2i}(y - \frac{1}{y})\frac{1}{2^{10}}(y + \frac{1}{y})^{10}\frac{\mathrm{d}y}{iy}$, welche zur Benutzung unbrauchbar sind.
Beispiele solcher Systeme findet man in der „alten Generation", wie SAC-1 und ALPAK.

(b) **Neue Linke** (New Left): Diese CA-Systeme transformieren nicht notwendigerweise einen Ausdruck in seine kanonische Repräsentation. Dem Benutzer bleibt damit teilweise die Kontrolle erhalten. Dies erreicht man dadurch, daß eine kanonische Darstellung in systematischer Weise nicht mehr gefordert wird. Beispiele dafür sind REDUCE und ALTRAN.
Behandelt werden in diesen Systemen mehrere Datentypen und Ausdrücke. Sie sind leicht zu erweitern. Eine Idee, die hinter diesen Fähigkeiten steckt, ist einige Datentypen zu „etikettieren" („labels").
Beispiele:

(1) $xe^x + x^2 \sin x$. Es werden die „labels" $y = e^x$, $z = \sin x$ eingeführt, woraus sich $xy + x^2z$ ergibt.

(2) $\frac{e^{2x}+e^x}{e^x}$ führt mit $y = e^{2x}$ und $z = e^x$ auf $\frac{y+z}{z}$.

In diesen Systemen kommt Vereinfachung zur kanonischen Form und in speziellen Darstellungen vor.

(c) **Liberale** (Liberals): Sie basieren auf einer sehr allgemeinen Repräsentation von Ausdrücken. Vereinfachungen werden durch Umformungen erzielt, die dicht an die Berechnungen von Hand anschließen. Diese Systeme

- fassen Terme in Summen und Exponenten in Produkten zusammen,
- haben Regeln für '0' und '1',
- unterdrücken bestimmte Operatoren (z.B. $a+(b+c) \to a+b+c, \quad a+(-b) \to a-b$),
- vereinfachen oft (aber nicht immer) spezielle Argumente ($\sin 2\pi \to 0, \quad e^{2\log(y)+x} \to y^2 e^x$).

Sie unterscheiden sich von Radikalen und Neue Linke Systemen, weil

- wie für Neue Linke Systeme die Erweiterung nur auf Wunsch durchgeführt wird.
- die Regel $\frac{a}{b}+\frac{c}{d} \to \frac{ad+bc}{bd}$ nicht zwingend ist. Sollte diese trotzdem erforderlich sein, wird der ggT nicht automatisch berechnet.
- nicht vereinfachte Formen möglich sind $(1 \cdot \sin x + 0 \cdot \cos x)$, aber nicht während einer Berechnung.
- Ausdrücke, wie e^x, $x!$, $\sum_{i=0}^{n} c_i x^i$ nicht durch „labels" ersetzt und durch diese dargestellt werden.
- eine Information eine lokale Bedeutung hat, d.h. daß $\sin(x)$ in einem Ausdruck eine andere Bedeutung haben kann wie $\sin(x)$ in einem anderen Teil des gleichen Ausdrucks.

Nachteil: Nicht effizient in bezug auf Ausführungszeit und Speicher.
Vorteil: „natürliche" Darstellung von Ausdrücken.

(d) **Konservative** (Conservatives): Die Hauptidee ist die Einsicht, daß es nicht möglich ist, sich ein Verfahren zur Vereinfachung vorzustellen, das optimal für alle Fälle ist. Es hat daher wenig Sinn, die Vereinfachung zu automatisieren. Dafür werden dem Benutzer Hilfen bereitgestellt sich sein eigenes Vereinfachungssystem zu entwerfen.
Alle CA-Systeme, die auf dieser Philosophie gründen sind verschwunden, weil eine Vereinfachung immer sehr schwer zu implementieren und immer sehr kostspielig ist.

(e) **Katholisch** (Catholics): Hier werden mehrere Darstellungen für einen Ausdruck angeboten. Das Prinzip hier ist: greift eine Technik nicht mehr, muß eine andere herangezogen werden. Dem Benutzer muß es möglich sein, leicht von einer Repräsentation in eine andere zu gelangen. Beispiele dafür sind MACSYMA und Scratchpad-I.
Nachteil: Die Systeme wachsen schnell und werden sehr groß.
Ein Beispiel für eine spezielle Darstellung, ausgewählt für eine Berechnung im Körper der Himmelsmechanik, ist die Poissonreihe

$$S(x,y) = \sum (P_j \cos(j \cdot y) + Q_j \sin(j \cdot y))$$

mit $x = (x_1, \ldots x_m)$ Variablen in Polynomen, $j = j_1, \ldots, j_n$ ganze Zahlen, $y = (y_1, \ldots y_n)$ trigonometrische Variablen und Q_j, P_j Polynome in x_i mit rationalen oder Gleitkommazahlen Koeffizienten.

Eine letzte Bemerkung zu Vereinfachung, die auf der Darstellung basiert, ist, daß ein konservatives System nicht erkennen kann, daß $\sqrt{6} = \sqrt{2}\sqrt{3}$ ist und es nicht immer ausgeschlossen ist mit einem „radikalen" System zu „beweisen", daß

$$1 = \sqrt{1}\sqrt{1} = \sqrt{1} = \sqrt{(-1)(-1)} = \sqrt{-1}\sqrt{-1} = i \cdot i = -1$$

gilt. Ein Vereinfacher muß daher (im Rahmen von algebraischen Zahlen) mit sehr großer Sorgfalt entworfen werden.

9.3 Vereinfachung in Systemen

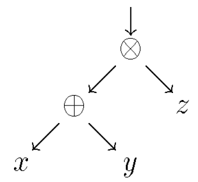
Vereinfachung wird meist auf zwei Abstraktionsebenen ausgeführt.

(1) Durch algorithmische Verfahren, die wiederum auf bekannten mathematischen Methoden beruhen. Diese werden in den folgenden Abschnitten vorgestellt.

(2) „Einfache" Vereinfachung durch Programmieren der Handhabung von Ausdrücken, was hier kurz in diesem Abschnitt beschrieben wird.

Viele der „einfachen" Vereinfachungen, die in Abschnitt 9.1 aufgelistet wurden, können auf einer sehr einfachen Ebene durchgeführt werden, besonders leicht bei der Umformung von Ausdrücken. Man muß sich daran erinnern, daß Ausdrücke hauptsächlich (wie in Lisp) in einer Baumstruktur dargestellt werden, z.B.

$(x+y)*z \quad \rightarrow$



.

Jeder dieser Grundoperatoren $(*, +, \ldots)$ hat wohldefinierte Eigenschaften. Es ist daher natürlich für die Funktionen $*$ und $-$, Funktionen aufzustellen, die die Vereinfachung jeweils relevant für diese durchführen. Praktisch heißt dies, TIMES wird durch STIMES (Vereinfachungs-Produktbefehl) ersetzt, der die Operation $*$ leistet und alle möglichen Vereinfachungsregeln anwendet. Diese Idee kann auch auf Spezialfälle ausgedehnt werden. Es ist beispielsweise möglich, Ausdrücke zu kennzeichnen, wenn sie Entsprechungen zu faktorisierten Polynomen, quadratfeier Faktorisierung, ggT-freie Gebrochenrationale … haben. Damit ist ein kontinuierlicher Informationsfluß durch die Berechnung hindurch (billig) verfügbar.
Auf ähnliche Weise führt man SPLUS, SEXPT, SQUOTIENT ein und ebenso Vereinfacher für den Betrag, trigonometrische Funktionen, Logarithmen … .

Es gibt noch eine andere Vereinfachung, die ziemlich leicht zu implementieren ist, die Vereinfachung für gemeinsame Unter-Ausdrücke. Die Idee besteht darin, nur einen Ausdruck zu behalten, wenn der Vereinfacher den Originalausdruck zurück gibt.
Die „Philosophie“, die hinter so einer Methode steht, ist, daß ein Vereinfacher als ein befehlsgesteuerter Transformierer von kanonischen Ausdrücken oder Formen entworfen werden kann. Verbunden mit dem Vereinfacher bekommt man eine lexikographische Ordnung. Dies wird gefordert, um beispielsweise $a + b - a + b \to 2b$ zu erhalten.

9.3.1 Benutzergesteuerte Vereinfachung

Wie kann man die Benutzersteuerung bei einem Vereinfacher der obigen Form zulassen? Dies wird in den beiden Möglichkeiten

- unterdrücken der eingebauten Regeln zur Vereinfachung durch Neudefinition S(OPERATOR) mit neuen Vereinfachungsregeln,
- benutzen eines gewöhnlich vorhandenen Befehls (das Akronym dafür ist): DECLARE,

realisiert. Dieser Befehl erlaubt die Einführung von Eigenschaften, wie Kommutativität (z.B. folgt mit DECLARE(F, COMMUTATIVE) die Vereinfachung F(A,B) $-$ F(B,A) $\to$ 0), Additivität, Antisymmetrie, Linearität, Assoziativität (links-, rechts- oder beide), Gleichheit, … .

Einen solchen Vereinfacher findet man in MACSYMA. Zusätzlich sind in diesem System noch spezielle Vereinfachungs-Pakete vorhanden für kanonische rationale Ausdrücke, Vereinfachung von Summen, Taylorreihen, trigonometrische Ausdrücke, Poissonreihen, … . Auch findet man dort noch die folgende weitere Technik zur Vereinfachung.

9.3.2 Identifikation durch Mustererkennung

Bisher wurde nur die *syntaktische* (oder lexikographische) *Identifikation* betrachtet. Nun wird die *semantische Identifikation* in den Mittelpunkt gestellt. Dies geschieht durch das Prädikatenkonzept.

Beispiel:
Man betrachte die Vorzeichen-Funktion

$$\mathrm{SIGINT}(x) = \begin{cases} -1 & : \quad x < 0 \\ 0 & : \quad x = 0 \\ +1 & : \quad x > 0 \end{cases} .$$

Nun definiert man das Prädikat NEGATPRED(x) mit true, falls SIGINT(x) $= -1$, sonst mit false. Der Befehl DECLARE erlaubt es, eine Form mit einem Prädikat zu verbinden, z.B.

$$\mathrm{DECLARE}(\mathrm{name},\ \mathrm{Prädikat}(\mathrm{arg}_1, \ldots, \mathrm{arg}_n)),$$

also speziell hier DECLARE(A, NEGATPRED(x)).

Der Vorteil der semantischen Identifikation ist die Möglichkeit, sehr leicht Eigenschaften von Operatoren (wie z.B., daß die Addition kommutativ ist mit Nullelement) benutzen zu können, um die unterschiedlichen Teile einer Form und das Auftreten dieser Form in einem Ausdruck zu erkennen. So kann man beispielsweise für Polynome eine eindeutige Form P mit unmittelbarer Identifikation der Koeffizienten finden.

Die *Identifikations-Definition* erreicht man durch einen Befehl vom Typ DEFMATCH, der ein neues Programm (Prädikat) definiert, das nur dann ausgeführt wird, wenn eine bestimmte semantische Form erkannt wurde.

$$\mathrm{DEFMATCH}(\mathrm{Programmname},\ \mathrm{Form},\ varform_1, \ldots, varform_n).$$

Beispiel: (eines fiktiven Programms)

- DEFMATCH(linear, $ax + b,\ x$)
- linear($3y + 4,\ y$) $\rightarrow b = 4,\ a = 3,\ x = y$
- linear($zy + 4 + x,\ y$) $\rightarrow b = x + 4,\ a = z,\ x = y$

9.3.3 Der Identifikationsalgorithmus

(Dieser Algorithmus ist aus MACSYMA entnommen).

Definition 9.1
Eine **nicht identifizierte** *(oder ungepaarte) Variable einer Form ist eine deklarierte Variable ohne zugewiesenem Wert.*
Eine Form p heißt **verglichen mit** *einem Ausdruck e, wenn p mit e identifiziert wird. Ist der Vergleich erfolgreich, so werden allen ungepaarten Variablen von p Werte zugewiesen, schlägt der Vergleich dagegen fehl, wird 'false' ausgewiesen.*

Definition 9.2
Eine Form, in der alle Variablen gepaart sind, heißt **feste Form** *(Konstanten, Zahlen, atomare Namen, ...).*

Definition 9.3
Eine Form heißt **verankert**, *wenn alle festen Teile aus dem gegenwärtigen Ausdruck der Form derart entfernt wurden, daß*
entweder

(i) der verbleibende Teil der Form aus einer isolierten ungepaarten Variablen besteht, die weder in einer Summe noch in einem Produkt steht,

oder

(ii) es mindestens einen festen Unterteil der Form gibt, so daß jeder Ausdruck, der diesen darstellt in mindestens zwei Teile aufgespaltet werden kann und jeder dieser Teile einem verankerten Unterteil der originalen Form entspricht.

Das Hauptziel des Algorithmus ist es, die verankerten Formen zu identifizieren. Bei der Identifizierung eines Ausdruckes e mit einer Form p werden im Algorithmus folgende Schritte ausgeführt:

(1) Falls p eine feste Form ist, dann identifiziert p den Ausdruck e genau dann, wenn $p - e$ nach durchgeführter (kanonischer) Vereinfachung zu '0' wird.

(2) Falls p eine Summe $\sum a_i$ ist, werden alle festen a_i aus e entfernt und die verbleibenden a_i werden wie folgt geprüft:

- **(2A)** Ist a_i ein Produkt mit mehr als einer ungepaarten Variablen, dann liegt eine Mehrdeutigkeit vor (siehe (2E)).

(2B) Ist a_i ein Produkt einer deklarierten Variablen v und einer festen Form f, dann wird das Prädikat von v auf die Koeffizienten von f in e angewandt. Wird dabei 'false' zurückgeliefert, schlägt die Identifizierung fehl, sonst wird fortgefahren.

(2C) Ist a_i eine ungepaarte Variable, so muß es die einzige ungepaarte Variable a_i sein (eine Auswahl kann möglich sein).

(2D) Ist a_i eine Exponentiation, so gibt es mehrere Möglichkeiten: feste Basis, fester Exponent, beide nicht fest

(d1) feste Basis: Man sucht in e einen Exponentiationsoperator mit der gleichen Basis. Hat man damit Erfolg, werden die Exponenten verglichen. Ist die Suche dagegen erfolglos, testet man, ob der Exponent 1 möglich ist (man setzt die Basis = Summe und subtrahiert diese von e).

(d2) fester Exponent: Man sucht in e einen Exponentiationsoperator mit gleichem Exponent. Bei Erfolg werden die Basis Formen verglichen, sonst wird, falls der Exponent < 0 (ganzzahlig) ist, '1' von e subtrahiert und (sofern kein Nenner vorhanden) die Basis Form mit '1' verglichen. Falls der Exponent > 0 ist wird zu '0' verglichen, d.h.

$a + \frac{1}{b}$	wird identifiziert zu	$x + 1$	für $a = x,\ b = 1$
	und zu	x	für $a = x - 1,\ b = 1$
$a + b^2$	wird identifiziert zu	x	für $b = 0,\ a = x$

(d3) beide nicht fest: man sucht eine Identifikation wie in (2E).

(d4) Spezialfall: man sucht eine Exponentiation, die das einzige Element der ungepaarten Summe ist ($y^x + A$, wobei A identifiziert ist und subtrahiert wird).
Falls die Basis B fest ist, sucht man auch nach $B**E$ und vergleicht zu '1'.
Falls der Exponent E fest ist, sucht man auch nach $B**E$ und vergleicht zu '0'.

(2E) Ist a_i eine bestimmte Funktion (sin, ...) wird nach dem ersten Auftreten dieser Funktion gesucht und dann auf den Argumenten getestet.

(2F) Ist der Name für a_i nicht gepaart, versucht man a_i mit allen Funktionen (even, $+$, $*$, $**$) wie in (2E) zu identifizieren.

(3) Falls p ein Produkt $\prod a_i$ ist, ersetze man 'subtrahiere' durch 'dividiere' und 'Summe' durch 'Produkt' in (2) (mit Ausnahme von (2A) und (2B), weil Produkte von Produkten nicht betrachtet werden).

(4) Falls p eine Exponentiation ist, geht man ähnlich vor wie in (2D). Dabei sind die Fälle (d1), (d2) und (d4) gleich. Ist weder Basis noch Exponent fest (d3), so fährt man wie folgt fort

- e ist '1' $\rightarrow p$ wird zu $1 ** 0$ verglichen,
- e ist '0' $\rightarrow p$ wird zu $0 ** 1$ verglichen,
- e ist keine Potenz $\rightarrow p$ wird zu $e ** 1$ verglichen,
- e ist eine Potenz $\rightarrow$ Basis und Exponent werden jeweils verglichen.

(5) Falls p eine bestimmte Funktion ist, vergleicht man nach der Suche in einer Tabelle, sucht die Anzahl der Argumente und identifiziert diese Argumente.

(6) Falls p eine nicht spezifizierte Funktion ist, geht man wie in (5) vor und testet zusätzlich auf Funktionsnamen.

(7) Falls p eine nicht identifizierte atomare Variable ist („Atom“ in Lisp), wird p verglichen mit e.

Es ist möglich diese Operationen in jeder Stufe beliebiger Tiefe zu mischen und einzufügen. Dieser Algorithmus ist ausreichend, da für jeden Ausdruck, der syntaktisch akzeptiert wird ein Verfahren zur Mustererkennung und Identifikation definiert ist. Ein solcher Algorithmus terminiert immer.

9.4 Das Problem des Null-Vergleichs

Wie bereits zu sehen war (und dies wird im nächsten Abschnitt bestätigt werden), ist das Problem der Vereinfachung kein wohldefiniertes Problem. Aber ein Teilproblem davon – das Erkennen, ob ein Ausdruck äquivalent zu Null ist – ist wohldefiniert. Dies ist das sogenannte Problem des Null-Vergleichs (zero equivalence problem). Jedoch ist auch dieses Teilproblem noch schwierig genug. Die Hauptfrage, die sich in diesem Zusammenhang stellt ist: Für welche Funktionenklassen ist das Null-Vergleichsproblem entscheidbar?
Die Antwort darauf lautet, daß dieses Problem nicht immer entscheidbar ist.

Satz 9.4 (Richardson)
Sei $\mathcal{R}$ die Klasse aller Ausdrücke bestehend aus

(1) rationalen Zahlen und π,

(2) einer Variablen x,

(3) den Operatoren Addition $+$, Multiplikation $*$, Komposition $\cdot$,

(4) Sinus und Betrag.

Dann gilt, für alle $E \in \mathcal{R}$ ist das Prädikat 'E ist identisch Null' unentscheidbar (nicht rekursiv).

Zum Beweis dieses Satzes benötigt man den Satz von Matiyasevič, der zeigt, daß das 10. Hilbertsche Problem[1] nicht lösbar ist.

Satz 9.5 (Matiyasevič, 1970)
Es gibt eine Menge $\mathcal{P} = \{P(x_1, \ldots, x_n)\}$ von Polynomen über den ganzen Zahlen, so daß das Prädikat 'es existieren nicht negative ganze Zahlen $a_1, \ldots, a_n$, so daß $P(a_1, \ldots, a_n) = 0$' auf $\mathcal{P}$ unentscheidbar (nicht rekursiv) ist.

Nun gilt es zu untersuchen, wie sich dieser Satz auf den Fall univariater Polynome anwenden läßt. Dazu zunächst eine Erklärung zur Schreibweise. Ist $f : \mathbb{R} \longrightarrow \mathbb{R}$ eine Funktion, dann bedeutet $f^{(0)}(x) = x$ und $f^{(n+1)}(x) = f(f^{(n)}(x))$ für $n \geq 0$.

Lemma 9.6
Es seien $h(x) = x \sin x$ und $g(x) = x \sin(x^3)$. Dann gibt es für alle $a_1, a_2, \ldots, a_n \in \mathbb{R}$ und für alle $0 < \epsilon < 1$ ein b, so daß

$$|h(g^{(k-1)}(b)) - a_k| < \epsilon \qquad \forall\, 1 \leq k \leq n$$

gilt.

Beweisskizze:
Der Beweis ist eine Induktion über n.
(i) Für n=1 trivial.
(ii) Richardson zeigt zuerst, daß zu zwei gegebenen reellen Zahlen x und y eine Zahl $b > 0$ derart existiert, daß $|h(b) - x| < \epsilon$ und $g(b) = y$ ist.
Angenommen, das Lemma sei richtig für n. Durch Anwenden der Induktionsannahme auf $a_2, \ldots, a_{n+1}$ gibt es ein b' mit

$$|h(g^{(k-2)}(b') - a_k| < \epsilon \text{ für } k \in \{2, \ldots, n+1\}$$

Es existiert aber auch (oben $x = a_1, y = b'$) ein b mit

$$|h(b) - a_1| < \epsilon \text{ und } g(b) = b'$$

Aus diesen beiden Gleichungen folgt die Behauptung für $n + 1$. $\square$

Das Lemma besagt, daß jedes endliche Tupel reeller Zahlen durch eine relle Zahl repräsentiert werden kann, wobei deren Position im Tupel keine Rolle spielt. Diese Methode ist sicherlich nicht eindeutig. Daher nimmt man an, daß jedes n-Tupel reeller Zahlen $(x_1, \ldots, x_n)$ durch eine reelle Zahl x (näherungsweise) dargestellt werden kann. Dies führt auf die

[1] **10. Hilbertsches Problem** (Diophantisches Problem)**:** Gibt es einen Algorithmus, der entscheidet, ob ein multivariates Polynom mit ganzzahligen Koeffizienten eine ganzzahlige Lösung besitzt?

Definition 9.7
Eine Funktion $F(x_1, \ldots, x_n) \in \mathcal{R}$ wird von $G(x_1, \ldots, x_n) \in \mathcal{R}$ dominiert, wenn für alle rellen Zahlen $x_1, \ldots, x_n$ gilt:

1. $G(x_1, \ldots, x_n) > 1$
2. $\forall\, \Delta_1, \Delta_2, \ldots, \Delta_n \in \mathbb{R}$ *mit* $|\Delta_i| < 1 \quad G(x_1, \ldots, x_n) > F(x_1 + \Delta_1, \ldots, x_n + \Delta_n)$

Lemma 9.8
Zu jeder Funktion $F \in \mathcal{R}$ existiert eine dominante Funktion G.

Der Beweis ist eine Induktion über die Anzahl der Operatoren in F. Man wendet beispielsweise folgende Transformationen auf F an, wobei angenommen wird, daß f, f_1, f_2 jeweils von g, g_1, g_2 dominiert werden. Dann ergibt sich, falls

- $F = f_1 + f_2 \rightarrow G = g_1^2 + g_2^2 + 2$
- $F = f_1 * f_2 \rightarrow G = (g_1^2 + 2) * (g_2^2 + 2)$
- $F = x \rightarrow G = x^2 + 2$
- $F = \sin(f) \rightarrow G = 2.$ $\square$

Richardson bewies dann den folgenden wichtigen Satz.

Satz 9.9
Für alle $P \in \mathcal{P}$ existiert ein $F \in \mathcal{R}$, so daß gilt

(i) *es existiert ein n-Tupel nicht negativer ganzer Zahlen $\bar{a} = (a_1, \ldots, a_n)$, so daß $P(\bar{a}) = 0$*

$\Longleftrightarrow$

(ii) *es existiert ein $\bar{b} = (b_1, \ldots, b_n)$ mit $b_i \geq 0$ und reell, so daß $F(\bar{b}) < 0$.*

Der Beweis ist sehr technisch und wird hier deshalb übergangen. Im Beweis des Satzes von Richardson geht es weiter unter Benutzung von Techniken und Ergebnissen aus der Logik. Man zeigt nun, daß, falls es für $F(x) \in \mathcal{R}$ möglich ist, zu entscheiden, ob $F(x) \equiv 0$, man dann auch entscheiden kann, daß ein $b \in \mathbb{R}$ existiert, so daß $G(b) < 0$ mit $F(x) = |G(x)| - G(x)$ ist. Diese beiden Aussagen stehen aber, wie man zeigt, im Widerspruch zueinander.

Richardson fragte sich, ob es möglich sei, zu entscheiden, ob ein Ausdruck in einem reellen Intervall äquivalent zu Null ist. Der oben skizzierte Beweis baut auf einer Reduktionsmethode auf, die darin besteht, das Problem des Null-Vergleichs für weniger komplexe Ausdrücke zu betrachten. Dabei wählte er einen lexikographischen und keinen algebraischen Komplexitätgrad (d.h. e^{e^x} ist komplexer wie e^x). Die Definition des Komplexitätsgrades ist unerheblich, sofern die gewählte Definition durchgängig beibehalten wird. Der Satz von Richardson zeigt ein Unentscheidbarkeitsergebnis. Der folgende Satz demonstriert ein entscheidbares Resultat.

Entscheidungsalgorithmus 9.10 (Richardson)
In der Funktionenklasse zusammengesetzt aus

(1) rationalen Zahlen und π,

(2) der Variablen x,

(3) $+$, $*$ und Division $/$,

(4) $\log |u|$ *und exp(u) (u beliebige Funktion aus dieser Klasse)*

kann man beweisen, daß das Prädikat '$F(x) \equiv 0$ ist true / false' entscheidbar ist, was darauf hinaus läuft zu zeigen, daß ein konstanter Ausdruck (abhängig von F) '0' ist. Damit existiert eine Entscheidungsverfahren (nicht wirklich ein Algorithmus) für diese Klasse.

Hier eine kurze Skizze dieses Verfahrens.
Sei F eine Funktion der obigen Klasse. Man zerlegt F in Unterausdrücke (auf eine bestimmte, wählbare Weise) und ordnet diese nach ihrer „Komplexität". Dabei ist y_1 ein Unterausdruck von y_2, falls y_1 von geringerer Komplexität ist als y_2. Angenommen, es ist y der Unterausdruck von F mit größter Komplexität und es sei $y = \log(u)$. Dann wird F zerlegt in

$$F = a_n y^n + a_{n-1} y^{n-1} + \ldots + a_1 y + a_0.$$

Das Entscheidungsverfahren wird auf a_n angewandt. Ist $a_n \equiv 0$, dann wird das Verfahren auf

$$F_1 = a_{n-1} y^{n-1} + \ldots + a_0$$

angewandt, das von geringerer Komplexität als F ist und es gilt

$$a_n \equiv 0 \implies [\, F \equiv 0 \iff F_1 \equiv 0\,].$$

Ist $a_n \not\equiv 0$, dann bildet man

$$\mathbb{D}F_2 = \tfrac{F}{a_n} = y^n + \tfrac{a_{n-1}}{a_n} y^{n-1} + \ldots + \tfrac{a_0}{a_n}$$

und dann

$$\mathbb{D}F_3 = F_2' = ny^{n-1}y' + \ldots + \tfrac{a_n a_0' - a_0 a_n'}{a_n^2}.$$

Nun gilt:

$$F_2 \equiv 0 \implies F_3 \equiv 0 \qquad \text{und} \qquad F_3 \equiv 0 \implies F_2 \equiv \text{konstant}.$$

Dieses Schema zieht sich durch das ganze Verfahren. Es basiert darauf, daß die Ableitung von y weniger komplex ist als die Funktion y. Ist beispielsweise $y = \log(u)$, dann ist $y' = \frac{u'}{u}$ und (u^{-1} und u) weniger komplex als y, so daß das Verfahren fortgesetzt werden kann. Ist dagegen $y = e^u$, gilt dies nicht mehr.
Sei $y = e^u$ und $F = a_n y^n + \ldots + a_0$. Das Entscheidungsverfahren wird auf a_0 angewandt. Ist

$$\begin{aligned}
a_0 \equiv 0 &\implies F_1 = a_n y^n + \ldots + a_1 y = Qy \\
a_0 \not\equiv 0 &\implies F_2 = \frac{F}{a_0} \\
& F_2' = \left(\frac{a_n' a_0 - a_n a_0' + n a_n a_0 u'}{a_0^2}\right) y^n + \ldots \\
& + \left(\frac{a_1' a_0 - a_1 a_0' + a_1 a_0 u'}{a_0^2}\right) y.
\end{aligned}$$

Nun gilt

$$\frac{F_2'}{y} \equiv 0 \implies F \equiv \text{konstant}$$

(und $\frac{F_2'}{y}$ ist von geringerer Komplexität).
Das Verfahren reduziert die Frage des Null-Vergleichs auf das Problem zu entscheiden, ob der Wert eines Ausdrucks für eine Konstante gleich '0' ist. Da dieses Problem allgemein unentscheidbar ist, ist dieses Verfahren auch nicht wirklich ein Algorithmus, aber es ist ein gültiges Verfahren für die hier betrachtete Funktionenklasse.

9.5 Versuch zur Formalisierung des Problems der Vereinfachung

Das Problem der Vereinfachung hat zwei Anliegen:

(1) einfachere äquivalente Objekte zu erzeugen,

(2) kanonische Darstellungen für äquivalente Objekte zu berechnen (siehe Kapitel 2).

Sei $\mathcal{T}$ eine Klasse von Objekten (Ausdrücken), z.B. Terme (erster Ordnung), (eine reduzierte Klasse) logischer Formeln, eine Klasse von Programmen, ...
Sei $\sim$ eine Äquivalenzrelation auf $\mathcal{T}$ (funktionale Äquivalenz, Gleichheit aufgrund der Axiome, Kongruenz modulo einem Ideal ...).

Das Problem des Findens einfacherer äquivalenter Objekte besteht darin, algorithmische Verfahren S ($S : \mathcal{T} \longrightarrow \mathcal{T}$) zu finden, die für alle $t \in \mathcal{T}$ die Bedingungen

(i) $S(t) \sim t$

(ii) $S(t) \leq t$

erfüllen, wobei sich '$\leq$' auf das Komplexitätsmaß bezieht, das man ausgewählt hat (z.B. „kürzer", „weniger Speicher", ...). Es wird (aus offensichtlichen Gründen) $S(t) \leq t$ für fast alle $t \in \mathcal{T}$ gefordert.

„Kanonische Vereinfachung" besteht darin, ein Verfahren S (kanonischer Vereinfacher) ($S : \mathcal{T} \longrightarrow \mathcal{T}$) für $\sim$ zu finden, so daß für alle $s, t \in \mathcal{T}$ gilt:

(i) $S(t) \sim t$

(ii) $s \sim t \implies S(s) = S(t)$

($S(t)$ ist die kanonische Form von t).

Beispiel:
Das Transformieren elementarer arithmetischer Operationen in Polynome, wie

$$\frac{1}{2}(2x+2)(x-1) \to x^2 - 1,$$

ist ein trivialer kanonischer Vereinfacher.

Die beiden Probleme (1) und (2) sind nicht völlig unabhängig voneinander. Ein kanonischer Vereinfacher definiert ein Maß für die „Einfachheit". Aber eine Definition für Einfachheit kann auch einen kanonischen Vereinfacher induzieren. Auf der anderen Seite kann eine kanonische Form Ausdrücke erzeugen, die nicht notwendig am einfachsten sind. So ist beispielsweise $x^2 + 2x + 1$ die kanonische Transformation für $(x+1)^2$.

In CA-Systemen ist der Entwurf eines kanonischen Vereinfachers ein grundlegendes Problem, da dieser eine direkte Verbindung zu

- algorithmischen Berechnungen in verschiedenen algebraischen Bereichen
- dem Problem algorithmischen Entscheidens auf Äquivalenzrelationen

herstellt. Ein kanonischer Vereinfacher erzeugt sofort einen Algorithmus zur Entscheidung auf Äquivalenzen und zur Berechnung in algebraischen Bereichen, die auf diesen Äquivalenzen definiert sind.

Satz 9.11 (Kanonische Vereinfachung und Entscheidbarkeit)
Sei $\mathcal{T}$ eine entscheidbare Menge von Objekten und $\sim$ eine Äquivalenzrelation auf $\mathcal{T}$. Dann gilt:
$\sim$ *ist entscheidbar* $\iff$ *es existiert ein kanonischer Vereinfacher S für $\sim$.*

Beispiel:
Es sei eine abelsche Halbgruppe definiert durch die Generatoren a, b, c, f, s und den Relationen $as = c^2s$, $bs = cs$, $s = f$. Die (Termersetzungs-)Regeln $s \to f$, $cf \to bf$ und $b^2f \to af$ bilden dann (was gezeigt werden kann) einen kanonischen Vereinfacher. $a^5bc^3f^2s^3$ und $a^5b^2c^2s^5$ repräsentieren das gleiche Element der Halbgruppe, da ihre kanonischen Formen identisch sind: a^7f^5. ($a^5bc^3f^2s^3 \overset{cf\to bf}{\longrightarrow} a^5b^3cf^2s^3 \overset{b^2f\to af}{\longrightarrow} a^6bcf^2s^3 \overset{cf\to bf}{\longrightarrow} a^6b^2cf^2s^3 \overset{b^2f\to af}{\longrightarrow} a^7f^2s^3 \overset{s\to f}{\longrightarrow} a^7f^5$ und ähnlich für $a^5b^2c^2s^5$).
Dagegen sind cs^2 und c^2s zwei verschiedene Elemente, weil sie unterschiedliche kanonische Formen haben:

$$cs^2 \overset{s\to f}{\longrightarrow} cf^2 \to bf^2,$$
$$c^2s \to c^2f \to cbf \to b^2f \to af.$$

Satz 9.12 (Kanonische Vereinfachung und Berechenbarkeit)
Sei $\mathcal{T}$ eine entscheidbare Menge, R eine auf $\mathcal{T}$ zweiwertige berechenbare Operation und $\sim$ eine Äquivalenzrelation auf $\mathcal{T}$, die bezüglich R eine Kongruenz ist. Weiter wird angenommen, S sei ein kanonischer Vereinfacher von $\sim$. Sei

$$\mathrm{Rep}(\mathcal{T}) := \{\, t \in \mathcal{T} \mid S(t) = t \,\}$$

die Menge der kanonischen Repräsentanten und

$$R'(s,t) := S(R(s,t)) \quad \forall\, s,t \in \mathrm{Rep}(\mathcal{T}).$$

Dann ist $(\mathrm{Rep}(\mathcal{T}), R')$ *isomorph zu* $(\mathcal{T}/\!\sim, R/\!\sim)$*, die Menge* $\mathrm{Rep}(\mathcal{T})$ *entscheidbar und* R' *berechenbar.*

(Es ist auch folgende Schreibweise üblich: $R/\!\sim (C_s, C_t) := C_{R(s,t)}$, wobei C_t die Kongruenzklasse von t bezüglich $\sim$ ist.)

Dieser Satz besagt, ist man im Besitz eines kanonischen Vereinfachers für eine Äquivalenzrelation, die eine Kongruenz bezüglich einer berechenbaren Operation ist, dann kann man die Struktur der Faktoren in Verbindung mit R algorithmisch „beherrschen“.

Beispiel:
Die Termersetzungsregeln $x^3 \to x^2$, $x^2y \to x^2$ definieren auf $\mathcal{T} := \mathbb{Q}[x,y]$ für die Kongruenzrelation modulo dem Ideal I, das durch die beiden Polynome $x^3y - x^3$ und $x^3y - x^2$ erzeugt wird, einen kanonischen Vereinfacher S (man veranschauliche sich das).
Rep($\mathcal{T}$) ist die Menge von Polynomen $f \in \mathbb{Q}[x,y]$, in denen kein Monom von f ein Vielfaches von x^3 oder x^2y ist.

Aufgrund der Unentscheidbarkeit einiger funktionaler Äquivalenzen führt man zur Vereinfachung Konzepte ein, die schwächer als das Kanonische sind:

- normale Vereinfachung (oder Null-Vergleich)
- reguläre Vereinfachung

Normale Vereinfachung
Eine berechenbare Abbildung von $\mathcal{T}$ nach $\mathcal{T}$ heißt genau dann ein normaler Vereinfacher (oder Nullvergleich) für die Äquivalenzrelation $\sim$ in $\mathcal{T}$, wenn für alle $t \in \mathcal{T}$ gilt:

(i) $\mathrm{S}(t) \sim t$

(ii) $t \sim 0 \Longrightarrow \mathrm{S}(t) = \mathrm{S}(0)$.

Es gelten – wie man zeigen kann – folgende Eigenschaften:

(1) jeder kanonische Vereinfacher ist ein normaler Vereinfacher

(2) existiert eine Operation M auf $\mathcal{T}$ mit $[\, s \sim t \Leftrightarrow M(s,t) \sim 0\,]$, dann gilt
$[\,\exists$ normaler Vereinfacher $]\ \Longrightarrow\ [\,\exists$ kanonischer Vereinfacher für $\sim\,]$

Reguläre Vereinfachung
Sie wird in einem Kontext verwendet, in dem transzendente Funktionen (exp, log, ...) benutzt werden. Für die Zwecke hier in diesem Zusammenhang, reicht es völlig aus zu wissen, daß ein regulärer Vereinfacher garantiert, daß alle nicht rationalen Terme in einem vereinfachten Ausdruck algebraisch unabhängig sind.

9.5.1 Kanonische Vereinfachung für Polynome und rationale Ausdrücke

9.5.1.1 Polynome

(a) Die Menge aller Terme der Form $a_n x^n + \ldots + a_1 x + a_0 \quad (n \geq 0)$, $a_i \in R$, $a_n \neq 0$ bildet zusammen mit dem Term 0 ein System kanonischer Formen für $R[x]$.

(b) Im multivariaten Fall ($x^j := x_1^{j_1} \cdots x_n^{j_n}$) bildet $\{a_{i_1} x^{i_1} + \ldots + a_{i_m} x^{i_m}\} \cup \{0\}$ ein System kanonischer Formen für $R[x_1, \ldots, x_n]$.

(c) Man kann für Polynome auch eine kanonische rekursive Form benutzen.
Bsp.: $(3x^2 + 5)y^2 + (x - 3)y + (x^3 - 2x + 1)$.

Polynome über Gruppen
Sei G eine Gruppe und $\mathcal{R}$ die Menge der Formen $a_0 x^{n_1} a_1 x^{n_2} \cdots a_{r-1} x^{n_r} a_r$, wobei $r \in \mathbb{Z}$, $r \geq 0$, $n_i \in \mathbb{Z}$, $n_i \geq 0$, ($a_t \in G$, $t = 0, 1, \ldots, r$) und ($a_t \neq 1$ für $t = 1, 2, \ldots, r-1$). Dann ist $\mathcal{R}$ ein System kanonischer Formen für $G[x]$.
Es ist auch möglich kanonische Formen für Verbände, Boolesche Algebren, ... zu definieren. (Mehr dazu im Buch von Lausch, Nöbauer: „Algebra of polynomials“; dort werden die kanonischen Formen mit normalen Formen bezeichnet.)

9.5.1.2 Rationale Ausdrücke

Rationale Ausdrücke werden als Elemente des Quotientenkörpers über $R[x_1, \ldots, x_n]$ (R integer) dargestellt. Es werden kanonische Vereinfacher für die Äquivalenzrelation $\sim$ gesucht, die auf dem Bereich

$$R[x_1, \ldots, x_n] \times (R[x_1, \ldots, x_n] \setminus \{0\})$$

definiert ist. Man unterscheidet die beiden Fälle, in denen rationale Ausdrücke

- keine Quotienten von Polynomen (z.B. $(x - \frac{x}{x^2})/(x + 1)$) sind. In diesem Fall benutzt man die rationale Arithmetik der jeweiligen algebraischen Bereichen und den ggT.

- Quotienten von Polynomen (z.B. $(x^2 - 1)/(x - 1)$) sind. Hier wird auf den ggT zurückgegriffen.

9.5.2 Kanonische Vereinfachung von Ausdrücken mit Radikalen

Für Ausdrücke, die aus Variablen $x_1, \ldots, x_n$, Konstanten, den arithmetischen Funktionen $(+, -, *, /)$ und dem Wurzelzeichen $\sqrt[q]{\ }$ (oder gebrochenem Exponent S^r) zusammengesetzt sind

- gibt es einen kanonischen Vereinfachungsalgorithmus für nicht geschachtelte Radikale (d.h. $\sqrt{\cdots\sqrt{\cdots}}$ sind ausgeschlossen),
- für Ausdrücke, die geschachtelte Radikale enthalten, sind jedoch nur Teillösungen bekannt.

9.5.3 Transzendente Funktionen

Einige Klassen von Ausdrücken sind definiert durch: Variablen (x_i), Konstanten π, e, arithmetischen Funktionen $(+, -, *, /$, Betrag), exp, log, sin, error-function (erf), … .

Hierfür

- gibt es keine allgemeine Lösung
- können nicht alle Ausdrücke auf eine kanonische Form gebracht werden (Richardson)
- sind verschiedene Techniken möglich und verfügbar
  - (i) spezielle Verfahren für bestimmte Klassen von Ausdrücken
    - (a) Null-Vergleich für rationale Exponentialausdrücke (i, π, rationale Zahlen, $x_1, \ldots, x_n$, $+, -, *, /$, exp)
    - (b) kanonische Vereinfacher für Exponentialpolynome (rationale Zahlen, Variable x, $+, -, \cdot$, exp); sie basieren auf bekannten Identitäten, wie

    $$\exp(x + y) = \exp(x) \cdot \exp(y).$$

    Die Beweise hierzu bauen meist auf „Vermutungen“ aus der Zahlentheorie auf.
  - (ii) Punktauswertung.
    Wird zum Testen auf funktionale Äquivalenz von transzendenten Ausdrücken benutzt. Die Idee hat einen sehr einfachen Hintergrund. Sind zwei Ausdrücke s und t funktional nicht äquivalent, dann hat $(s - t)$ (aus

funktionaler Sicht) höchstens abzählbar viele Nullstellen. Man erhält also $(s = t)$ aus $(s - t)(z) = 0$ mit einer Wahrscheinlichkeit von 1.
Dieses Verfahren ist aber in der Praxis sehr kompliziert und nicht praktikabel.

(iii) Reduktionsverfahren.
Reduktion auf Untermengen, für die Äquivalenzrelationen bekannt sind. Ein derartiges Verfahren kann beispielsweise dazu benutzt werden, einen kanonischen Vereinfacher für formales ableiten transzendenter Ausdrücke einer gegebenen Menge zu finden.

(iv) Struktursätze.
(Dies ist die modernste und systematischste Methode). Die Grundidee dazu ist: Um einen transzendenten Ausdruck t in $\mathbb{Q}$ zu vereinfachen, berechnet man eine transzendente Erweiterung.

(a) Man konstruiert eine (transzendente) Erweiterung K von $\mathbb{Q}$, so daß $t \in$ K.

(b) t wird durch Benutzen rationaler Arithmetik vereinfacht.

Bemerkung:
Es gibt noch viele andere Aspekte, die oft vollständig zur Theorie der Programmiersprachen gehören. Davon seien hier nur

- vollständige Reduktionssysteme
- Church-Rosser Eigenschaft
- Knuth-Bendix Vervollständigungsalgorithmus
- Äquivalenz von Gleichungstheorien

erwähnt.

Andere Aspekte, die später aufgezeigt werden, (können oder) spielen ebenfalls eine Rolle beim Entwurf eines allgemeinen Vereinfachungsschemas.

- Standard oder Gröbner Basis
- Elimination von Quantoren.

Kapitel 10

Formale Potenzreihen

Dieses Kapitel orientiert sich stark an [Zip93].

Definition 10.1 (Formale Potenzreihen(FPS)) *Formale Potenzreihen sind unendliche Potenzreihen (Konvergenz spielt keine Rolle).*

Beispiele:

$$\begin{aligned} f(t) &= 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \frac{t^4}{4!} + \dots \\ g(t) &= 1 + t + 2!t^2 + 3!t^3 + 4!t^4 + \dots \end{aligned}$$

$g(t)$ konvergiert für keinen Wert $t \neq 0$.

10.1 Mathematischer Hintergrund

FPS verhalten sich zu Polynomen wie p-adische Zahlen zu den ganzen Zahlen. Einige Operationen können für FPS effizient ausgeführt werden, die es für Polynome nicht gibt.

Sei $\{A_n\}$ eine Folge von Ringen, mit Homomorphismen $\{\Theta_n\}$:

$$\dots \xleftarrow{\Theta_{n-2}} A_{n-1} \xleftarrow{\Theta_{n-1}} A_n \xleftarrow{\Theta_n} A_{n+1} \xleftarrow{\Theta_{n+1}} \dots$$

Kohärente Folge: $\{a_n\}$ mit $n \in A_n$ so daß $\Theta_n(a_{n+1}) = a_n$. Die Menge der kohärenten Folgen ist auch ein Ring (leicht zu beweisen). Er heißt auch der *inverse Grenzwert* von A_n und wird

$$\hat{A} = \varprojlim A_n$$

geschrieben. Sei R ein kommutativer Ring und m ein Ideal von R. Weil $m^{n+1} \subseteq m^n$, gibt es eine kanonische Abbildung

$$\Theta_n : R/m^{n+1} \mapsto R/m^n$$

Die m-adische *Vervollständigung* von R wird durch den inversen Grenzwert von $\{R/m^{n+1}\}$ definiert.

$$R_m := \varprojlim R/m^n$$

R_m ist kein Körper, da kein Element von R_m ein Inverses in R_m hat.

Beispiele:

$$\begin{aligned} \mathbb{Z}_p &= \varprojlim \mathbb{Z}/(p)^n \\ \mathbb{Z}[[X]] &= \varprojlim \mathbb{Z}[X]/(X)^n \end{aligned}$$

Bemerkung: Algorithmen für arithmetische Operationen auf FPS sind für konvergente Potenzreihen gültig. Daraus ergeben sich mögliche Anwendungen in der Analysis.

10.2 Definitionen und Einführung

Definition 10.2 *$k[[t]]$ ist der Ring der FPS mit Koeffizienten aus k.*

Der Grenzwert einer Cauchyfolge ist eine FPS und vice versa.

Beispiel: $1, 1+t, 1+t+\frac{t^2}{2}, 1+t+\frac{t^2}{2}+\frac{t^3}{6}, \ldots$

10.2.1 Inverses

Viele Elemente von $k[[t]]$ haben multiplikative Inverse, auch wenn ihre Bilder in k[t]das nicht haben. Zum Beispiel ist

$$(1+t)(1-t+t^2-t^3+\ldots+(-1)^n t^n) = 1-t^{n+1}.$$

Also gilt:

$$\frac{1}{1+t} = 1-t+t^2-t^3+\ldots+(-1)^n t^n+\ldots,$$

Es gibt allerdings Elemente von $k[[t]]$, die kein Inverses haben, z.B. $t+t^2$. Dennoch ist es möglich, formal ein Inverses für $t+t^2$ zu konstruieren:

$$\frac{1}{t+t^2} = \frac{1}{t}\frac{1}{t+1} = \frac{1}{t} - 1 + t - t^2 + \ldots + (-1)^{n-1} t^n + \ldots$$

Da der Exponent des ersten Terms negativ ist, ist diese Potenzreihe nicht in $k[[t]]$. Sie ist eine formale Laurentreihe.

Definition 10.3 (Formale Laurentreihe) *Der Quotientenkörper von $k[[t]]$ heißt Ring formaler Laurentreihen in t und wird $k((t))$ geschrieben. Er besteht aus Potenzreihen mit endlich vielen Gliedern mit negativem Koeffizienten.*

10.2.2 Algebraische Elemente

Habe k eine von 2 verschiedene Charakteristik. Dann ist

$$\sqrt{1+t} = 1 + \frac{1}{2}t - \frac{1}{8}t^2 + \frac{1}{16}t^3 - \frac{5}{128}t^4 + \ldots \tag{10.1}$$

und $\sqrt{1+t} \notin k[t]$, aber die rechte Seite von (10.1) ist eine FPS (und eine Quadratwurzel von $(1+t) \pmod{t^5}$). Aber nicht alle algebraischen Funktionen von Elementen aus $k[t]$ sind in $k[[t]]$ ($\sqrt{t+t^2} \notin k[[t]]$). Aber wie bei den Laurentreihen existiert auch hier eine formale Lösung:

$$\sqrt{t+t^2} = \sqrt{t}\sqrt{t+1} = t^{1/2} + \frac{1}{2}t^{3/2} - \frac{1}{8}t^{5/2} + \frac{1}{16}t^{7/2} - \frac{5}{128}t^{9/2} + \ldots$$

Wir sind im Ring $k[[t^{1/2}]]$. Im allgemeinen ist eine algebraischen Funktion eines Elements von $k[[t]]$ in $K[[t^{1/n}]]$, wobei K eine algebraische Erweiterung von k ist und $n \in \mathbb{Z}$.

10.2.3 Transzendente Elemente

$k[[t]]$ kann viele Elemente enthalten, die transzendent über $k[t]$ sind. Wenn z.B. die Charakteristik $\neq 0$ ist verhält sich

$$1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \frac{t^4}{4!} + \dots$$

wie e^t, (z.B. ist es seine eigene Ableitung). Das gleiche gilt für $\sin t, \cos t, \dots$

Es ist offensichtlich, daß es transzendente Operationen gibt, die auf Polynomen nicht ausgeführt werden können. Zum Beispiel kann man die ersten Terme von $\log(1+t+t^2)$ mit

$$\log(1+t) = t - \frac{t^2}{2} + \frac{t^3}{3} - \frac{t^4}{4} + \dots$$

errechnen.

Bemerkung: Es gibt hier sehr subtile Komplikationen: Zum Beispiel

$$\log(2+t) = \log 2 + \log\left(1 + \frac{t}{2}\right) = \log 2 + \frac{t}{2} - \frac{t^2}{8} + \frac{t^3}{24} - \frac{t^4}{64} + \dots$$

Diese Potenzreihe ist ein Element von $k(\log 2)[[t]]$. Man muß k transzendent erweitern.

Schluß: Wir müssen zwei verschiedene Erweiterungen von FPS betrachten.

1. PS mit negativem Exponenten
2. PS mit gebrochenem Exponenten

10.3 Potenzreihenarithmetik

Da FPS unendliche Objekte sind, entstehen Probleme. Es gibt drei prinzipielle Möglichkeiten damit umzugehen:

1. Abgeschnittene Potenzreihen.
2. Generatoren, die auf Anfrage weitere Terme liefern.
3. Allgemeine Formeln für den n-ten Koeffizienten einer Potenzreihe.

3 ist schwierig und nicht immer möglich, es gibt Koeffizienten ohne geschlossene Form. 2 ist implementiert und eigentlich ganz elegant, oft sind aber die Kosten zu hoch. 1 ist der übliche Ansatz und der mit der breitesten Anwendbarkeit.

Elemente von $k((t))$:

$$f = c_1 t^{e_1} + c_2 t^{e_2} + \ldots + c_n t^{e_n} + o(t^{e_n}) \tag{10.2}$$

$c_i \in k, e_i$ Brüche ganzer Zahlen, $e_1 < e_2 < \ldots < e_n$

$o(t^{e_n})$ bedeutet, daß wir nur Terme, deren Ordnung $\leq e^n$ explizit darstellen. Andere Comperalgebrasysteme z.B. Maple halten alle Terme mit Ordnung $< e^n$. Diese schreiben dann $O(t^{e_n})$.

10.3.1 Uniforme Parameter

Betrachten wir

$$f = t^{1/3} + t^{1/2} + t + t^{12/5} + o(t^{12/5}) \tag{10.3}$$

Es ist besser, einen uniformen Parameter einzuführen:

$$f = (t^{1/30})^{10} + (t^{1/30})^{15} + (t^{1/30})^{30} + (t^{1/30})^{72} + o((t^{1/30})^{72})$$

f ist also eine FPS in dem uniformen Parameter $t^{1/30}$.

Bemerkung: Das korrespondiert zu einem Verzweigungspunkt der Ordnung 30. Es erfaßt auch die Struktur der zugrundeliegenden Riemannschen Fläche. Jedoch geht das über das, was für uns wichtig ist, hinaus.

Wir gehen davon aus, daß f in (10.2) einen uniformen Parameter $u = t^{1/r}, r$ eine positive ganze Zahl, hat und schreiben es als ein Element von $k((u))$:

$$f = c_1 u^{re_1} + c_2 u^{re_2} + \ldots + c_n u^{re_n} + \ldots$$

re_i sind ganze Zahlen.

r heißt *Verzweigungsordnung* von f bei 0. $\operatorname{val} f := re_i$ heißt *Valenz* von f, $\operatorname{ord} f := re_n$ heißt *Ordnung*.

Beispiel: In (10.3) $\operatorname{val} f = 10, \operatorname{ord} f = 72$.

10.3.2 Darstellung

In einer Implementierung ist es besser uniforme Parameter zu verwenden, da die Exponenten dann ganzzahlig sind. Da Potenzreihen in der Regel dicht sind, ist es bequem die Koeffizienten in einem Array abzuspeichern. Eine typische Datenstruktur wäre:

```
TPS = struct {
      VerzweigungsOrdnung : Integer;
      Valenz : Integer;
      Ordnung : Integer;
      Koeffizienten : K[Ordnung — Valenz];
}
```

Das i-te Element des Koeffizienten gehört zum Exponenten $\dfrac{i + \operatorname{val} f}{r}$.

10.3.3 Basisoperationen

Wenn die Potenzreihen verschiedene Verzweigungsordnungen haben, sagen wir r und s, dann müssen sie in neue Potenzreihen mit Verzweigungsordnung $\operatorname{lcm}(r, s)$ umgewandelt werden. Betrachten wir

$$\begin{aligned} P(t) &= p_0 + p_1 t + p_2 t^2 + \ldots = \sum_{i \geq 0} p_i t^i \\ Q(t) &= q_0 + q_1 t + q_2 t^2 + \ldots = \sum_{i \geq 0} p_i t^i \end{aligned} \tag{10.4}$$

Addition

$$P(t) + Q(t) = (p_0 + q_0) + (p_1 + q_1)t + \ldots = \sum_{i \geq 0} (p_i + q_i) t^i \tag{10.5}$$

Multiplikation

$$\begin{aligned} P(t)Q(t) &= \sum_{i \geq 0} \sum_{j \geq 0} p_i q_j t^{i+j} \\ &= \sum_{k \geq 0} \left(\sum_{0 \leq l \leq k} p_l q_{k-l} \right) t^k \\ &= p_0 q_0 + (p_0 q_1 + p_1 q_0) t + (p_0 q_2 + p_1 q_1 + p_2 q_0) t^2 + \ldots \end{aligned} \tag{10.6}$$

In (10.6) ist die Cauchy'sche Multiplikationsformel benutzt worden. Die Formeln (10.5) und (10.6) können leicht für Valenz $\neq 0$ angepaßt werden. Seien z.B. $\operatorname{val} P = a, \operatorname{val} Q = b$, dann ist

$$P(t) + Q(t) = \sum_{i \geq \min(a,b)} (p_i + q_i) t^i \tag{10.7}$$

wobei $p_i = 0$ für $i < a$ und $q_i = 0$ für $i < b$. Für die Multiplikation haben wir:

$$P(t)Q(t) = \sum_{k\geq 0} \left(\sum_{0\leq l\leq k} p_{a+l}q_{b+k-l} \right) t^{a+b+k} \tag{10.8}$$

Beachte:

$$\operatorname{val}(P+Q) = \min(a,b), \qquad \operatorname{val}(PQ) = a+b$$

Es gibt eine weitere Schwierigkeit: Die Ordnung der resultierenden Potenzreihe kann von der Ordnung der ursprünglichen Potenzreihe verschieden sein. z.B.:

$$\left(1 + 2t + 3t^2 + 4t^3 + o(t^3)\right) + \left(1 + \frac{t}{2} + \frac{t^2}{3} + o(t^2)\right) = 2 + \frac{5}{2}t + \frac{10}{3}t^2 + o(t^2).$$

Eine direkte Anwendung von (10.5) könnte zu Problemen führen, da die resultierende Potenzreihe nur von der Ordnung 2 ist. Das gleiche gilt für das Multiplizieren von abgeschnittenen Potenzreihen. Die folgende Tabelle faßt diese Fakten zusammen. Das „$\geq$“ in der Valenzspalte heißt, daß die Valenz größer werden kann, wenn der führende Koeffizient von Summe, Produkt oder Potenz verschwindet. Für Produkte kann das nur passieren, wenn der Koeffizientenring Nullteiler hat.

	Valenz	*Ordnung*
$P(t)$	a	m
$Q(t)$	b	n
$P(t)+Q(t)$	$\geq \min(a,b)$	$\min(m,n)$
$P(t)Q(t)$	$\geq a+b$	$a+b+\min(m-a,n-b)$
$P(t)^s$	$\geq sa$	$(s-1)a+m$
$(P\circ Q)(t)$	$\geq ab$	$\min(b(\operatorname{val}_2 P)+n-b, bm)$

Tabelle 10.1: Valenz und Ordnung bei Anwenden der Basisoperationen

10.4 Potenzen von Potenzreihen

Das Berechnen von Potenzen ist einfacher als bei Polynomen und basiert auf der folgenden Proposition.

Proposition 10.4.1 (J.C.P.Miller) *Sei $P(t) = p_0 + p_1t + p_2t^2 + \ldots$ eine Potenzreihe und sei*

$$P(t)^s = p_0^{(s)} + p_1^{(s)}t + p_2^{(s)}t^2 + \ldots \tag{10.9}$$

deren s-te Potenz, dann lassen sich die Koeffizienten $p_i^{(s)}$ so ausdrücken:

$$p_0^{(s)} = p_0^s, \qquad p_k^{(s)} = \frac{1}{p_0 k} \sum_{1 \le l \le k} (ls - k + l) p_l p_{k-l}^{(s)} \tag{10.10}$$

Beweisskizze: Wir konstruieren eine Differentialgleichung mit $P(t)^s$, $P(t)$ und deren Ableitungen. Wenn man die Reihen für $P(t)^s$ und $P(t)$ einsetzt erhält man ein System linearer Gleichungen, das man lösen kann. Diese Technik ist ziemlich allgemein.

Zuerst gilt bei $t = 0$: $p_0^{(s)} = p_0^s$.

Sei $y(t) = P(t)^s$. Dann erfüllt $y(t)$ die Differentialgleichung:

$$sP'(t)y(t) - P(t)y'(t) = 0$$

weil

$$y(t) = P(t)^s \Rightarrow y'(t) = sP'(t)P(t)^{s-1} \Rightarrow P(t)y'(t) = sP'(t)P(t)^s.$$

Einsetzen der Entwicklungen ergibt

$$s\left(\sum_{n\ge 0} np_n t^{n-1}\right)\left(\sum_{m\ge 0} p_m^{(s)} t^m\right) - \left(\sum_{n\ge 0} p_n t^n\right)\left(\sum_{m\ge 0} mp_m^{(s)} t^{m-1}\right) = 0.$$

Multiplikation mit t und Verwendung der Cauchy'schen Multiplikationsformel bringt:

$$\sum_{k\ge 0}\left(\sum_{0\le l\le k} s(k-l)p_{k-l}p_l^{(s)} - (k-l)p_l p_{k-l}^{(s)}\right) t^k = 0.$$

Wir können jeden Koeffizienten von t^k Null setzen. Es entsteht ein System linearer Gleichungen für $p_l^{(s)}$.

$$\begin{aligned}
sp_1p_0^{(s)} - p_0p_1^{(s)} &= 0\\
2sp_2p_0^{(s)} + (s-1)p_1p_1^{(s)} - 2p_0p_2^{(s)} &= 0\\
&\vdots\\
kspkp_0^{(s)} + (ks - s - k)p_{k-1}p_1^{(s)} + \ldots + (s-k+1)p_1p_{k-1}^{(s)} - kp_0p_k^{(s)} &= 0
\end{aligned}$$

und somit

$$\sum_{1\le l\le k} (ls - k - l)p_l p_{k-l}^{(s)} = kp_0p_k^{(s)}$$

□

Die Formel (10.10) erlaubt einem, eine Potenzreihe mit Valenz 0 in jede Potenz zu erheben (nicht nur ganzzahlige). Mögliche Anwendung: Prüfen ob ein gegebenes Polynom eine perfekte Potenz ist.

Wenn die Valenz von 0 verschieden ist, ist dieses Problem ist delikater. Sei $P(t)$ eine Potenzreihe der Valenz a und der Ordnung m, so

$$\begin{aligned} P(t) &= p_a t^a + p_{a+1} t^{a+1} + \ldots + p_m t^m + o(t^m) \\ &= t^a \left[p_a + p_{a+1} t + \ldots + p_m t^{m-a} + o(t^{m-a}) \right] \end{aligned}$$

Unter Benutzung von (10.10) erhält man

$$P^s(t) = t^{sa} \left[p_a^{(s)} + p_{a+1}^{(s)} t + \ldots + p_m^{(s)} t^{m-a} + o(t^{m-a}) \right]$$

Wenn sa ganzzahlig ist, gibt es keine Probleme (insbesondere wenn $a = 0$). Ausmultiplizieren dieses Ausdrucks ergibt die Koeffizienten der Tabelle auf Seite 252. Wenn sa rational ist, muß man einen neuen uniformen Parameter einführen. Wenn sa auch keine rationale Zahl ist, kann das Ergebnis nicht als formale Potenzreihe dargestellt werden.

10.5 Komposition formaler Potenzreihen

Seien $P(t)$ und $Q(t)$ formale Potenzreihen. Nehmen wir an, $\mathrm{val}(P) = 0$. Betrachten wir was passiert, wenn auch $\mathrm{val}(Q) = 0$ und wir einfach t in $P(t)$ durch $Q(t)$ ersetzen.

$$(P \circ Q)(t) = \sum_{i \geq 0} p_i (q_0 + q_1 t + q_2 t^2 + \ldots)^i$$

Der konstante Term von $P \circ Q$ ist selbst eine unendliche Reihe. Daher können wir abgeschnittene Potenzreihen nicht für die Berechnung von $P \circ Q$ benutzen. Jetzt nehmen wir an $\mathrm{val}(Q) = 1$ und schreiben:

$$Q^k(t) = q_1^{(k)} t + q_2^{(k)} t^2 + q_3^{(k)} t^3 + \ldots$$

Klar ist $q_i^{(j)} = 0$ für $i < j$ und $q_k^{(k)} = q_1^k$. Mit (10.10) hängen $q_1^{(k)}, \ldots q_n^{(k)}$ nur von $q_1, \ldots, q_n$ ab. Ersetzen von t durch Q in der Potenzreihenentwicklung von $P(t)$ und Neuordnen der Terme ergibt:

$$\begin{aligned} R(t) &= (P \circ Q)(t) = p_0 + p_1 Q(t) + p_2 Q^2(t) + p_3 Q^3(t) + \ldots \\ &= p_0 + p_1 q_1^{(1)} t + (p_1 q_2^{(1)} + p_2 q_2^{(2)}) t^2 (p_1 q_3^{(1)} + p_2 q_3^{(2)} + p_3 q_3^{(3)}) t^3 + \ldots \end{aligned}$$

also

$$r_k = p_1 q_k^{(1)} + p_2 q_k^{(2)} + p_3 q_k^{(3)} + \ldots + p_k q_k^{(k)} \tag{10.11}$$

10.5.1 Ordnung und Valenz

In diesem Fall ist $\mathrm{ord}(P \circ Q)$ das Minimum der Ordnungen von P und Q, $\mathrm{val}(P \circ Q) = 0$.

Valenz: Die Lösung im allgemeinen Fall ist ganz einfach,

$$\mathrm{val}\ P \circ Q = (\mathrm{val}\ P)(\mathrm{val}\ Q)$$

(es sei denn, der führende Koeffizient von Q ist nilpotent).

Ordnung: Die Lösung ist subtiler. Nehmen wir vorerst an, $\mathrm{val}\ P = 0$ und

$$P(t) = p_0 + p_{a_2} t^{a_2} + \ldots + p_m t^m + o(t^m)$$

a_2 ist dabei der zweitkleinste Exponent von $P(t)$ mit einem von 0 verschiedenen Koeffizienten.

$$a_2 = \mathrm{val}(P(t) - P(0))$$

Einsetzen von $Q(t)$ für t in $P(t)$ ergibt

$$(P \circ Q)(t) = p_0 + p_{a_2}(q_b t^b + \ldots + q_n t^n + (t^n))^{a_2} + \ldots + p_m Q^m(t) + o(Q^m(t))$$

Die Ordnung von $(P \circ Q)$ ist das Minimum der Ordnungen aller Terme. Da

$$\mathrm{ord}(Q^i(t)) \leq \mathrm{ord}(Q^j(t)) \text{ für } i \leq j$$

haben wir

$$\mathrm{ord}(P \circ Q) = \min(a_2 b + n - b, mb).$$

Wenn $\mathrm{val}(P) = a, a \neq 0$ können wir

$$P(t) = t^a[\hat{P}(t) + o(t^m - a)]$$

schreiben, wobei $\mathrm{val}(\hat{P}) = 0$. Der zweitkleinste Exponent von $\hat{P}$ ist $a_2 - a$. Wir können jetzt die Ordnung aller Faktoren, wenn $Q(t)$ für t eingesetzt wurde berechnen und dann die Multiplikationsformel von oben benutzen um die Ordnung von $(P \circ Q)$ bestimmen.
Erster Faktor:

$$\mathrm{ord}\ Q^a(t) = ab + n - b$$

$$\mathrm{val}\ Q^a(t) = ab$$

(wenn $a \neq 0$. wenn $a = 0$ dann gilt $\mathrm{ord}\ Q^a(t) = \infty$).

Zweiter Faktor:

$$\mathrm{ord}(\hat{P} \circ Q) = \min((a_2 - a)b + n - b, (m - a)b) = \min(a_2 b + n - b, mb) - ab$$

$$\mathrm{val}(\hat{P} \circ Q) = 0.$$

Also, falls $a \neq 0$

$$\begin{aligned}
\mathrm{ord}(P \circ Q) &= ab + \min(ab + n - b - ab, \min(a_2 b + n - b, mb) - ab) \\
&= \min(ab + n - b, a_2 b + n - b, mb) \\
&= \min(ab + n - b, mb).
\end{aligned}$$

Diese Ergebnisse können wie folgt zusammengefaßt werden:

$$\mathrm{ord}(P \circ Q) = \begin{cases} 0 & \text{für val } Q = 0 \\ \min(a_2 b + n - b, mb) & \text{für val } P = 0 \\ \min(ab + n - b, mb) & \text{sonst} \end{cases}$$

Mit

$$\mathrm{val}_2(P) := \begin{cases} \mathrm{val}\ P & \text{für val } P \neq 0 \\ \mathrm{val}\ (P(t) - P(0)) & \text{für val } P = 0 \end{cases}$$

bekommt man, wie in der Tabelle auf Seite 252:

$$\mathrm{ord}(P \circ Q) = \min((\mathrm{val}_2 P)(\mathrm{val}\ Q) + \mathrm{ord}\ Q - \mathrm{val}\ Q, (\mathrm{ord}\ P)(\mathrm{val}\ Q)).$$

10.5.2 Horner–Regel

Wie berechnet man die Koeffizienten von $(P \circ Q)$?

Um die Sache einfach zu halten, nehmen wir $\mathrm{val}\ P = 1, \mathrm{val}\ Q = b, \mathrm{ord}\ P = \mathrm{ord}\ Q = n$ an. Der direkteste Weg ist die Horner–Regel für Polynome. Die Horner–Regel erlaubt es ein Polynom in ein anderes einzusetzen.

$$F(X) = f_0 X^d + f_1 X^{d-1} + \ldots + f_d$$

dann ist

$$F(G)) = (\ldots((f_0 G + f_1)G + f_2)G + \ldots)$$

Der Algorithmus ist einfach (*Fterms* ist eine Liste von (Koeffizient,Exponent)):

TermsHorner$(Fterms, G) := \{$
  $H := lc(Fterms);$
  $f := le(Fterms);$
  foreach$(e, c) \in Fterms\{$
    $H := G^{f-e} * H + c;$
    $f := e;$
  $\}$

return$(H * G^{e_{old}})$;
}

In unserem Fall führen wir $Q^{[r]}$, die Potenzreihe von Q abgeschnitten bei der Ordnung r, ein.

$$Q^{[r]} = q_1 t^b + q_2 t^{b+1} + \ldots + q_{r-b+1} t^r + o(t^r)$$

Wir können $P \circ Q$ als

$$P \circ Q = Q^{[n]}(p_1 + Q^{[n-b]}(p_2 + Q^{[n-2b]}(p_3 + \ldots)))$$

schreiben. Dieser Ausdruck berechnet alle Koeffizienten von $P \circ Q$ korrekt. Jedoch werden immer noch einige Terme zuviel berechnet. Um dies zu vermeiden müßte man

$$p_1 Q^{[n]} + p_2 (Q^{[n-b]})^2 + p_3 (Q^{[n-2b]})^3 + \ldots$$

berechnen, aber das läßt sich nicht zu einer Horner–Regel umordnen.

10.5.3 Matrixform der Berechnung

Ein Weg die Berechnung überflüssiger Terme zu vermeiden ist es (10.11) für die Berechnung von $P \circ Q$ zu benutzen. Der Platz, der von dieser Methode benötigt wird, ist $O(n^2)$ im Gegensatz zu $O(n)$ für die Horner–Regel. Daher wird dies nicht benutzt. Aber die Struktur dieser Darstellung ist interessant und nützlich. Wir definieren die Matrixrepräsentation einer formalen Potenzreihe folgendermaßen. Nehmen wir an val $P = 1$.

$$A_P = \begin{pmatrix} p_1^{(1)} & p_2^{(2)} & p_3^{(3)} & \cdots \\ 0 & p_2^{(2)} & p_3^{(3)} & \cdots \\ 0 & 0 & p_3^{(3)} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Es ist leicht zu beweisen, daß $A_P A_Q = A_{P \circ Q}$, wobei die Multiplikation eine Standardmatrizenmultiplikation ist. Dies ist also eine Methode $P \circ Q$ zu berechnen, aber die Horner–Regel ist effizienter.

10.5.4 Differentialgleichungstechniken

Wenn $P(t)$ eine Funktion ist, die eine Differentialgleichung erfüllt und $P \circ Q$ häufig benötigt wird, kann es den Aufwand Wert sein, eine Rekursionsrelation für die Koeffizienten von $P \circ Q$ in Abhängigkeit von den Koeffizienten von Q aufzustellen. Sei zum

Beispiel

$$R(t) = e^{Q(t)} = \sum_{m=0}^{\infty} r_m t^m.$$

Dann ist

$$R'(t) = Q'(t)e^{Q(t)} = Q'(t)R(t).$$

Es folgt die Relation

$$b_k = \sum_{1 \le l \le k} \frac{l}{k} a_l b_{k-l}$$

(Für den 0–Term $R(0) = r_0 = e^{q_0}$).

10.6 Reversion von Potenzreihen

Dies ist eine Operation, die mit Polynomen nicht möglich ist. Seien zwei formale Potenzreihen gegeben:

$$\begin{aligned} P(t) &= p_1 t + p_2 t^2 + p_3 t^3 + \ldots, \\ Q(t) &= q_1 t + q_2 t^2 + q_3 t^3 + \ldots, \end{aligned}$$

so daß $(P \circ Q)(t) = t$, dann sagt man P sei die *Reversion* von Q. Die einfachste Methode P aus Q zu berechnen ist, die Methode der unbestimmten Koeffizienten zu benutzen, um ein System von Gleichungen für die p_i in Abhängigkeit von den q_i aufzustellen. Unter Benutzung von (10.11) haben wir:

$$\begin{aligned} p_1 q_1 &= 0 \\ p_1 q_2 + p_2 q_2^{(2)} &= 0 \\ p_1 q_3 + p_2 q_3^{(2)} + p_3 q_3^{(3)} &= 0 \\ p_1 q_k + p_2 q_k^{(2)} + p_3 q_k^{(3)} + \ldots + p_k q_k^{(k)} &= 0 \end{aligned}$$

wobei die $q_i^{(s)}$ die Koeffizienten von t^i in $Q^A(t)$ sind. Diese Gleichungen sind linear in den p_i und haben eine eindeutige Lösung (solange $q_k^{(k)} = q_1^k$ nicht verschwindet).

Ein besonders schöner Weg die Berechnungen zu organisieren ist es, die Matrixrepräsentation der Potenzreihe zu benutzen. Wenn $(P \circ Q)(t) = t$ dann ist $A_P = A_Q^{-1}$.

A_Q wird Zeile für Zeile aufgebaut, indem man eine Reihe mit $Q(t)$ multipliziert um die nächste Reihe zu erzeugen. Da die Matrix ein obere Dreiecksmatrix ist, kann man die Inverse berechnen in dem man nur das „Rückwärtseinsetzen“ der Gauß Elimination verwendet.

Literaturhinweise

Die klassischen ‘Moderne Algebra’ Bücher sind [Wae70] (auf deutsch [Wae]), [Jac85]. Eine gute Einführung in die Algebra sind [Mey] und [Lip81]. Weitere, mehr anwendungsorientierte Bücher sind [BCL83], [DST89]. Eine Fülle an Material findet man in [Knu81] und [AHU74].

Jedoch das Standardwerk in Computer-Algebra ist im Moment sicher [GCL92].

Literaturverzeichnis

[AHU74] A. Aho, Hopcroft, J.D. Ullman, „The Design and Analysis of Computer Algorithms“, Addison-Wesley (1974).

[AMS93] Notices of the AMS, July/August 1993, Volume 40, Number 6.

[Art71] E. Artin: „Galois Theory“, Notre Dame Mathematical Lectures No.2, 6th printing (1971).

[Akr89] A.G. Akrtitas, „Elements of Computer Algebra“, Wiley & Sons (1989).

[BCL83] „Computer Algebra“, Eds. B. Buchberger, G.E. Collins & R. Loos, 2nd edition, Springer-Verlag (1983).

[Ber68] E.R. Berlekamp: „Algebraic Coding Theory“, McGraw Hill (1968).

[Bet88] T. Beth: Skript WS87/88 „Computer Algebra I“, Uni Karlsruhe (1988).

[Coh93] H. Cohen, „A Course in Computational Algebraic Number Theory“, Graduate Texts in Mathematics, Springer-Verlag (1993).

[DST89] J.H. Davenport, Y. Siret, E. Tournier, „Computer Algebra, Systems and Algorithms for Algebraic Computation“, 2nd printing, Academic Press (1989).

[GCL92] K.O. Geddes, S.R. Czapor, G. Labahn: „Algorithms for Computer Algebra“, Kluwer Academic Publication (1992).

[Jac85] N. Jacobson, „Basic Algebra I, Basic Algebra II“, Freeman (1985, 1980).

[Knu81] D.E. Knuth: „The Art of Computer Programming“ Vol. 2, „Seminumerical Algorithms“, 2nd edition, Addison-Wesley (1981).

[Lip81] J.D. Lipson: „Elements of Algebra and Algebraic Computing“, Benjamin/Cummings (1981).

[Mey] K. Meyberg, „Algebra“ Teil 1 und 2, Hanser Verlag.

[Mut80] C. Mutafian: „Equationes Algébriques et théorie de Galois“, Vuibert (1980).

[Rot83] Encyclopedia of Mathematics & its Applications, „Finite Fields“, Ed. Rota, Addison-Wesley (1983).

[Wae70] B.L. Van der Waerden: „Algebra“ Vol. 1&2 (früher: „Modern Algebra“), F. Ungar Pub.Co. (1970).

[Wae] B.L. Van der Waerden: „Algebra“ (Band I und II), Heidelberger Taschenbücher.

[Win88] F. Winkler: Skript Universität Linz (1988).

[Zip93] R. Zippel: „Effective Polynomial Computation“, Kluwer Academic Publication (1993).