Algebraische Komplexitätstheorie
Vorlesung Sommersemester 1997

H.-G. Gräbe

22. April 1997

1 Einführung

Die algorithmische Lösung von Berechnungsproblemen hat seit jeher zu den Hauptaufgaben der Mathematik gehört. Lange Zeit basierten derartige Lösungen auf einem intuitiven Berechenbarkeitsbegriff. Erst in der ersten Hälfte dieses Jahrhunderts hat man, angeregt durch metamathematische Fragestellungen, intensiv nach einer allgemeinen Formalisierung des intuitiven Berechenbarkeitsbegriffs gesucht. Verschiedene formale Modelle wurden vorgeschlagen, wie z.B. Turingmaschinen, Registermaschinen, While-Programme, rekursive Funktionen, Markov-Algorithmen und Thue-Systeme. Diese Konzepte stellten sich aber alle als äquivalent heraus, was dann zu der Überzeugung führte, damit eine adäquate Explikation des intuitiven Berechenbarkeits- und Algorithmenbegriffs gefunden zu haben (These von Church). Dies hatte und hat enorme Auswirkungen. Zunächst einmal war es jetzt möglich, die algorithmische Unlösbarkeit von Problemen zu beweisen. Berühmte Beispiele sind das Halteproblem, das Postsche Korrespondenzproblem, das Wortproblem der Gruppentheorie oder Hilberts 10. Problem. Auf der anderen Seite hatten Konzepte wie Turingmaschinen und While-Programme einen großen Einfluß auf die Entwicklung der ersten Universalrechner und deren Programmierung.

Diese Fragen bilden heute für gewöhnlich den Kern eines (klassischen) Kurses zur Komplexitätstheorie. So schreibt R. Reischuk in der Einleitung zu [13]:

> Das Bestreben der Komplexitätstheorie ist es, grundlegende Aussagen zu machen mit welchem Aufwand algorithmische Probleme auf einer Maschine gelöst werden können. Dabei sind drei Komponenten zu berücksichtigen: die Problemstellung, die zur Verfügung stehenden Rechnersysteme und die darauf implementierten Algorithmen.
>
> Den Aufwand kann man auf verschiedene Weisen messen; für den Anwender sind in erster Linie die Rechenzeit und der notwendige Speicherplatz von Bedeutung. [...]
>
> Als Grundlage für diese Untersuchungen dienen formale Maschinenmodelle, mit denen die aufgeworfenen Fragen mathematisch exakt behandelt werden können. Ein Modell sollte einerseits so abstrakt sein, daß man auf relativ einfache Weise möglichst weitreichende Aussagen machen kann. Andererseits darf es sich nicht zu sehr von den in der Wirklichkeit vorhandenen Rechnerarchitekturen

> entfernen. Die Komplexitätstheorie bemüht sich, allgemeingültige Beziehungen aufzuzeigen. Um Konsequenzen für den praktischen Einsatz von Rechnern abzuleiten, bedarf es dann einer entsprechenden Interpretation.
>
> Der erste Schritt muß daher sein, geeignete Modelle zu spezifizieren und ihre Eigenschaften zu untersuchen. Stehen mehrere Modelle zur Auswahl, ist ein gegenseitiger Vergleich sinnvoll. [... Dabei stellte sich heraus], daß die verschiedenen Modelle, die bislang vorgeschlagen worden sind, keine allzu großen Unterschiede bezüglich ihrer Leistungsfähigkeit aufweisen, was man als ... Bestätigung für die formalisierte Vorgehensweise ansehen kann. Es bilden sich zwei Klassen heraus, die die sequentiellen Maschinen bzw. die parallelen Maschinen repräsentieren.

Bei diesem Herangehen an Fragen der Komplexitätstheorie spielen also Turingmaschinen, das RAM-Modell und Schaltkreisfamilien eine zentrale Rolle, an denen durch die genaue Untersuchung der Relationen zwischen Zeit- und Platzkomplexität in den verschiedenen Berechenbarkeitsmodellen die Gleichwertigkeit derselben demonstriert wird. Typische Techniken der Logik wie das Diagonalverfahren oder die gegenseitige Zurückführbarkeit von Problemen aufeinander bilden die wichtigsten Zugänge. Traditionell sind dabei schwierige kombinatorische Probleme zu lösen, was die enge Verbindung zu Fragen der diskreten Mathematik erklärt. Daneben interessiert die Klassifizierung von (oftmals nicht allzu komplexen) Algorithmen nach Schwierigkeitsklassen entsprechend ihrer Laufzeit- oder Speicherkomplexität und das gegenseitige Verhältnis dieser Komplexitätsklassen. Im Mittelpunkt der Untersuchungen stehen Algorithmen mit polynomialer Laufzeit und Fragen im Zusammenhang mit der Untersuchung der Klassen $\mathcal{P}$ und $\mathcal{NP}$.

Eine zweite Tradition der Untersuchung komplexitätstheoretischer Fragestellungen geht auf Probleme der numerischen Algebra und der Algorithmentheorie zurück, die nach der Effizienz unterschiedlicher Verfahren zur Lösung ein und derselben Problemstellung fragen. Neben der Analyse numerischer Verfahren ist hier insbesondere auf die Analyse verschiedener Such- und Sortierverfahren zu verweisen, wie sie etwa in D. Knuths bahnbrechender Monographie [11] expliziert wird.

Im Zeitalter der Digitalrechner tritt diese Frage nach effizienten Algorithmen immer mehr in den Vordergrund. Manche Probleme lassen sich sehr effizient lösen, während andere, darunter zahlreiche praxisrelevante, sich allen Anstrengungen zur effizienten Lösung widersetzten. Diese Tatsache führte auf die Frage nach einem tieferen Verständnis der inhärenten Schwierigkeiten bei der algorithmischen Lösung von Berechnungsproblemen und insbesondere zur Suche nach Möglichkeiten, diese auch mathematisch zu beschreiben. Neben der Effizienzanalyse konkreter Algorithmen, die zu *oberen Schranken* für die Komplexität eines Problems führen, wurden dabei auch mathematische Prinzipien entwickelt, die Aussagen über *untere Schranken* für ein gegebenes Problem zulassen.

Solche Untersuchungen finden immer auf der Basis eines bestimmten Berechnungsmodells und eines gewählten Kostenmaßes statt. Unter der *Komplexität* eines Problems versteht man jeweils die minimalen Kosten, die zu seiner Lösung im Rahmen des gewählten Berechnungsmodells erforderlich sind.

Ein wichtiger neuer Zweig dieser Untersuchungen eröffnet sich mit der fortschreitenden Vernetzung der Rechentechnik, die ein gleichzeitiges, paralleles oder verteiltes Berechnen ermöglicht. Im Gegensatz zum fast einheitlichen Design des von-Neumann-Rechners gibt es dafür die verschiedensten Speicher- und Rechnerarchitekturen sowie Netztopologien. Neben der Frage der möglichst optimalen Verteilung einer Rechnung auf mehrere Prozessoren bei einer beschränkten Prozessorenzahl spielt auch hier der theoretisch erreichbare Parallelisierungsgrad eine zentrale Rolle, der im Extremfall für jeden einzelnen Rechenschritt einen neuen Prozessor vorsieht, um die Abhängigkeit von konkreten Netztopologien zu eliminieren. Es stellt sich heraus, daß man diese Frage nach *der parallelen Komplexität* bei unbeschränkter Prozessorzahl im Rahmen des zu entwickelnden seriellen arithmetischen Berechnungsmodells ebenfalls formulieren und studieren kann.

Gegenstand dieser Vorlesung ist die *algebraische Komplexitätstheorie.* Darunter versteht man die Untersuchung der Komplexität algebraischer Berechnungsprobleme im Rahmen algebraischer Berechnungsmodelle, deren wichtigste die straight-line Programme und Berechnungsbäume sind. In der vorangegangenen Vorlesung "Basisalgorithmen der Computeralgebra" haben wir bereits einige der Algorithmen unter diesem Gesichtspunkt näher untersucht und entsprechende Komplexitätsaussagen für das Rechnen mit langen Zahlen, mit Polynomen, die ggT-Berechnung und teilweise auch für die Faktorisierung (in einem sequentiellen Berechnungsmodell) abgeleitet.

Betrachten wir aber zur Einstimmung zuerst drei Probleme, die die Anfänge der algebraischen Komplexitätstheorie entscheidend geprägt haben.

1.1 Schnelles Potenzieren

Aufgabe: Gegeben sind $x \in R, n \in \mathbf{N}_+$. Man berechne $x^n \in R$ mit möglichst geringem Aufwand.

Wir betrachten zwei verschiedene Lösungen:

1. Die klassische Iteration:

```
procedure power1(x:R,n:integer):R;
     p:=1;
     for i:=1 to n do p:=p*x;
     return p;
```

2. Schnelles Potenzieren (binäres Potenzieren):

```
procedure power2(x:R,n:integer):R;
     if n=0 then 1 else if n=1 then x
     else if odd(n) then x*power2(x*x,n div 2)
     else power2(x*x,n div 2);
```

Zählen wir zunächst jeweils die Anzahl der Multiplikationen:
$C_1(n) = O(n), C_2(n) \leq C_2(\frac{n}{2}) + 2$. Letzteres liefert $C_2(n) = O(log\, n)$

Das entspricht einem Berechnungsmodell, das die arithmetischen Operationen in R mit einheitlichen Kosten ansetzt. Dieses Herangehen bezeichnet man auch als *arithmetische Komplexität.* In einem solchen Modell ist also das binäre Potenzieren wesentlich effektiver als das übliche Verfahren. In praktischen Anwendungen tritt dieses Modell vor allem beim Rechnen in modularen Zahlbereichen auf.

Berücksichtigen wir dagegen die Länge der verarbeiteten Faktoren, so sieht die Rechnung anders aus. Die Berechnung des Produkts zweier ganzer Zahlen $a, b \in \mathbf{Z}$ benötigt mit dem klassischen Multiplikationsverfahren bekanntlich $O(l(a) \cdot l(b))$ Zeittakte, wobei $l(a)$ die Bitlänge der jeweiligen Zahl ist, d.h. die Anzahl der (in konstanter Zeit verarbeitbaren) Speichereinheiten, die zur Abspeicherung der Zahl a notwendig sind. Dabei legen wir die klassische Darstellung von a in einem bestimmten Positionssystem mit einer Basis β zugrunde, dessen Ziffern vom Prozessor des entsprechenden Computers in konstanten Taktraten verarbeitet werden können. Diese Bitlänge ist damit proportional zum Logarithmus der Zahl a, d.h. es gilt $l(a) \sim log(a)$. Dieses Komplexitätsmaß bezeichnet man deshalb auch als *logarithmische Komplexität.*

Sind a, b Zahlen der Länge l, so erhalten wir

$$C_1(l,n) = \sum_{k=1}^{n} k\, l^2 \sim \frac{n^2}{2} l^2$$

und für $n = 2^m$

$$\begin{aligned} C_2(l, n = 2^m) &= l^2 + C_2(2l, 2^{m-1}) = l^2 + (2l)^2 + C_2(2^2 l, 2^{m-2}) = \ldots = \\ & l^2 + (2l)^2 + \ldots + (2^{m-1} l)^2 = \tfrac{4^m - 1}{3} l^2 \sim \tfrac{n^2}{3} l^2 \end{aligned}$$

Bei letzteren Berechnungen sind wir von einer speziellen Form von n ausgegangen, wo die Berechnung der Potenz mit besonders wenigen Multiplikationen auskommt. Wir sehen also, daß in diesem Berechnungsmodell die Hauptkosten im zweiten Verfahren durch die letzte Multiplikation verursacht werden und insgesamt sich beide Varianten nur um einen konstanten Faktor unterscheiden, also aus komplexitätstheoretischer Sicht als etwa gleichwertig einzustufen sind.

Kehren wir noch einmal zur ersten Fragestellung zurück. Die zweite Lösung macht entschieden von der inneren Struktur des Problems Gebrauch, indem sie geschickt Zwischenergebnisse speichert und weiterverwendet, während das Herangehen im ersten Fall weitgehend "dumm" ist und in dieser Form ebenso für die Berechnung des Produkts beliebiger n Zahlen Verwendung finden könnte. Aus Sicht der algebraischen Komplexitätstheorie ergeben sich hier zwei Fragen, die die Entwicklung des Fach wesentlich mitgeprägt haben. Die erste Frage lautet, ob das binäre Potenzieren bereits der effektivste Trick ist, oder man vielleicht durch anderweitiges geschicktes Zusammenfassen der Faktoren noch effektiver sein kann. Obwohl der gesunde Menschenverstand auf diese Frage mit "nein" antwortet, ist dies jedoch kein mathematisch strenges Argument.

Diese Frage steht am Anfang der Entwicklung der algebraischen Komplexitätstheorie und wurde zuerst von A. Scholz [14] 1937 in der folgenden Formulierung aufgeworfen: Eine Folge $1 = a_0 < a_1 < \ldots < a_r = n$ heißt *Additionskette* für n, wenn sich jedes a_i mit $i \geq 1$ als $a_i = a_j + a_k$ mit $j, k < i$ darstellen läßt. Bestimme $l(n) = min\ r$, wobei das Minimum über alle solchen Additionsketten zu bilden ist.

Eine solche Kette entspricht der Berechnung von X^n über die Zwischenergebnisse $x_i := x_j \cdot x_k, i = 1, \ldots, r$ mit $x_0 = X$ und umgekehrt und funktioniert für jeden Ring R. $l(n)$

gibt die Anzahl der auszuführenden Multiplikationen an, wobei unser oben beschriebener Algorithmus

$$l(n) \leq \lfloor log(n) \rfloor w_2(n) - 1 \leq 2log(n)$$

ergibt, wenn $w_2(n)$ die Anzahl der Ziffern 1 in der Binärdarstellung von n, das *Hamming-gewicht* von n angibt und die Logarithmen zur Basis 2 genommen werden (was in Zukunft stillschweigend vorausgesetzt werden soll und bei O-Abschätzungen sowieso keine Rolle spielt).

Eine untere Schranke kann in diesem Fall durch ein Gradargument gefunden werden: Betrachten wir X als Unbestimmte, so kann der Grad eines Zwischenergebnisses in einem Berechnungsschritt nur um den Faktor 2 wachsen. Nach h Schritten ist also maximal ein Polynom vom Grad 2^h entstanden und somit

$$l(n) \geq \lceil log(n) \rceil.$$

Für $n = 2^m$ fallen untere und obere Schranke zusammen und `power2` ist somit, legt man die Anzahl der auszuführenden Multiplikationen zugrunde, optimal.

Aufgabe: Zeigen Sie, daß $l(mn) \leq l(m) + l(n)$ gilt.

Aufgabe: Entwerfen Sie einen Algorithmus, mit dem man $l(n)$ für $n \in \mathbf{N}_+$ berechnen kann. (Idee: Rekursive Erzeugung aller Listen von Additionsketten ?)

Insgesamt erhalten wir den folgenden Satz

Satz 1 *Die Länge $l(n)$ einer minimalen Additionskette für die natürliche Zahl n hat die Wachstumsordnung $l(n) = \Omega(log(n))$.*

Es bleibt allerdings eine Schere zwischen unterer und oberer Schranke, die einem Faktor 2 entspricht und die bis heute nicht geschlossen werden konnte. Für $w_2(n) = 1, 2$ sind die Schranken identisch. Für die verbleibenden Werte $n < 16$ gilt

n	*Intervall*	*Wert*
7	$3 \leq l(n) \leq 4$	4
11	$4 \leq l(n) \leq 5$	5
13	$4 \leq l(n) \leq 5$	5
14	$4 \leq l(n) \leq 5$	5
15	$4 \leq l(n) \leq 6$	5

Aufgabe: Prüfen Sie diese Rechnungen nach.

Die beste bekannte untere Schranke (Schönhage 1975) ist

$$l(n) \geq log(n) + log(w_2(n)) - 2.13,$$

die beste bekannte obere Schranke (Brauer 1939)

$$l(n) \leq log(n) + \frac{log(n)}{log(log(n))} + o(\frac{log(n)}{log(log(n))}).$$

Dieser Schranke liegt eine Blockversion des binären Potenzierens zugrunde: Man berechne zuerst für kleine Potenzen $2 \leq i < t := 2^\lambda$ durch maximal $t-1$ Multiplikationen alle Potenzen X^i und daraus X^n über die t-adische Darstellung

$$n = a_0 t^p + a_1 t^{p-1} + \ldots + a_p,$$

indem nacheinander $y_1 := X^{a_0 t}, x_1 := y_1 \cdot X^{a_1} = X^{a_0 t + a_1}, y_2 := x_1^t$ usw. bis $x_p := y_p \cdot X^{a_p} = X^n$ berechnet werden. Eine t-Potenz kann durch λ-faches Quadrieren berechnet werden, so daß wir insgesamt $t - 1 + p \cdot (\lambda + 1)$ Multiplikationen benötigen. Wegen $n \geq t^p = 2^{\lambda p}$ folgt insgesamt

$$l(n) < t + p + p \cdot \lambda \leq 2^\lambda + log(n) + \frac{log(n)}{\lambda}.$$

Für $\lambda = log(log(n)) - 2 \cdot log(log(log(n)))$ erhält man das behauptete Wachstumsverhalten

$$l(n) < \frac{log(n)}{log(log(n))^2} + log(n) + \frac{log(n)}{log(log(n))}.$$

Ein Vergleich der ursprünglichen und der Blockversion des binären Potenzierens weist auf eine weitere typische Abhängigkeit der Komplexitätstheorie hin: Eine verbesserte Laufzeiteffizienz läßt sich oft nur mit mehr Speicherplatzressourcen verwirklichen. `power2` muß sich nur die beiden letzten Zwischenergebnisse merken, `blockpower` dagegen alle Zwischenergebnisse $X^{a_0}, \ldots, X^{a_p}$. Solche Platz- vs. Zeitkomplexitätsaussagen wollen wir aber nicht weiter verfolgen.

Erlauben wir nun auch Divisionen zur Berechnung von X^n. Sei $d(n)$ die kleinste Anzahl von Operationen, die zur Berechnung von X^n in diesem neuen Modell notwendig sind. Sicher gilt $d(n) \leq l(n)$ und es kann tatsächlich $d(n) < l(n)$ sein.

Aufgabe: Zeigen Sie, daß $d(31) = 6 < l(31) = 7$ gilt und daß dies die kleinste Zahl n mit $d(n) < l(n)$ ist.

Wir sehen an diesem Beispiel, daß das Ergebnis der komplexitätstheoretischen Untersuchungen ganz wesentlich vom verwendeten Berechnungsmodell abhängen kann und eine Frage wie “Berechne X^n bei gegebenem X” erst durch die Angabe eines solchen eine korrekte Spezifikation erfährt.

Andererseits ist das erste Verfahren gut geeignet, auch auf einer parallelen Architektur umgesetzt zu werden. In der Tat, hat man viele ($> n$) Prozessoren zur Verfügung, so kann man im ersten Takt das Produkt je zweier benachbarter der n Faktoren berechnen, im zweiten Takt genauso das von je zwei benachbarten der verbliebenen $\frac{n}{2}$ Faktoren usw. Man kommt auf diese Weise mit $O(log(n))$ paralellen Rechenschritten aus, d.h. erreicht einen gewaltigen (und wie wir weiter sehen werden, optimal möglichen) Effektivitätszuwachs durch den Einsatz einer parallelen Architektur im Vergleich zum sequentiellen Vorgehen. Im Gegensatz dazu ist ein solcher Zuwachs beim binären Potenzieren, wenigstens auf so einfache Weise, nicht möglich, weil jeder sequentielle Schritt das Ergebnis des unmittelbar vorangehenden in essentieller Weise benötigt.

1.2 Evaluierung eines univariaten Polynoms

Problemstellung: Gegeben sei das Polynom $p = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$ mit Koeffizienten aus einer k-Algebra R. Man finde ein optimales Verfahren zur Evaluierung von allgemeinen Ausdrücken der gegebenen Bauart.

Bekanntlich kann ein solches Polynom mit dem Hornerschema

```
procedure horner(x:R,f=(a_0,...,a_n)):R;
     s:=a_0;
```

```
for i:=1 to n do s:=s*x+a_i;
return s;
```

unter Verwendung eines einzigen zusätzlichen Registers berechnet werden, wobei n Multiplikationen und n Additionen in R auszuführen sind. Als Geburtsjahr der algebraischen Komplexitätstheorie wird meist das Jahr 1954 angesehen, als Alexander Ostrowski [12] die Frage nach der Optimalität der Hornerschen Regel zur Diskussion stellte und unter anderem die Vermutung formulierte, daß es kein Verfahren zur Evaluierung allgemeiner Polynome gibt, das mit weniger als n Multiplikationen auskommt.

Wir wollen zuerst das Problem weiter spezifizieren. Zum einen können wir $a_0, \ldots, a_n$ als spezifische Elemente aus R betrachten und erhalten als Problem die Frage, ein *spezifisches* $p \in R[x]$ so effektiv wie möglich zu berechnen. Andererseits können wir $a_0, \ldots, a_n$ als unbestimmte Koeffizienten betrachten und nach einem effektiven Schema fragen, mit dem man ein *allgemeines* Polynom vom Grad n berechnen kann. Die entsprechende Berechnung ist im Polynomring $R[x, a_0, \ldots, a_n]$ auszuführen. Ist schließlich R ein Körper, so können wir auch nach $p \in R(x, a_0, \ldots, a_n)$ fragen, d.h. nach einer effektiven Berechnungsvorschrift für p als rationale Funktion. Daß dies zu unterschiedlichen Antworten führen kann, zeigt das Beispiel $p = \sum_{i=0}^n x^i$.

Das Hornerschema können wir in einem solchen Kontext folgendermaßen beschreiben: Es werden aus Eingabegrößen $g_{-n-1} := x, g_{-i} := a_i, i = 0, \ldots, n$ nacheinander die Zwischenergebnisse $g_{2i} := g_{2i-1} + g_{-i}$ und $g_{2i-1} := g_{2i-2} \cdot g_{-n-1}$ berechnet, unter denen sich das gesuchte Endergebnis befindet.

Zum Beweis der Vermutung von Ostrowski werden wir ein Berechnungsmodell verwenden, in dem Additionen, Subtraktionen und Multiplikationen mit einem Skalar, d.h. Elementen aus k, nicht gezählt werden, also nichts kosten. Ein solches Berechnungsmodell, das sogenannte *nichtskalare* oder *Ostrowski*-Berechnungsmodell ist durchaus zu rechtfertigen, wenn R etwa selbst ein Polynomring oder eine Matrixalgebra ist, da diese skalaren Operationen dann mit dem Aufwand $O(N)$ ausführbar sind, wenn N die Größe der Eingabe symbolisiert. Nichtskalare Multiplikationen in R sind dagegen teurer und dominieren für $N \gg 0$ die Laufzeit. Wichtiger ist aber, daß untere Schranken in einem solchen Berechnungsmodell, in dem nur nichtskalare Multiplikationen und Divisionen gezählt werden, zugleich eine untere Komplexitätsschranke für kompliziertere Berechnungsmodelle liefern, die in vielen Fällen bereits die erwartete Größenordnung hat.

Die Beweisidee der Ostrowski-Vermutung beruht auf der Beobachtung, daß die kostenfreien Operationen k-linear sind, also Vektorräume invariant lassen, während nichtskalare Operationen deren Dimension verändern. Ein Dimensionsargument liefert dann eine untere Schranke für die Anzahl der notwendigen nichtskalaren Operationen in einem *allgemeinen* Evaluationsschema.

Zählen wir statt der Multiplikationen nur die Additionen und Subtraktionen, so gilt ein analoges Resultat, das wir jetzt beweisen wollen.

Satz 2 *Jedes Programm, das nur $(+, -, *)$ verwendet und in der Lage ist, das allgemeine Polynom $p = \sum_{i=0}^n a_{n-i} x^i$ aus R und beliebig vorgegebenen Werten für $(x, a_0, \ldots, a_n)$ zu berechnen, benötigt mindestens n Additionen oder Subtraktionen.*

Beweis : Jedes solche Programm kann, aufgerufen mit $x = 1$, in ein Programm transformiert werden, das die Summe $\sum_{i=0}^{n} a_i$ aus der Eingabe $(a_0, \ldots, a_n)$ berechnet. Wir zeigen mit Induktion, daß zur Berechnung dieser Summe n Additionen oder Subtraktionen benötigt werden.

Gänzlich ohne Additionen oder Subtraktionen können nur Elemente der Art $c \prod a_i^{n_i}$ mit $c \in R$ berechnet werden. Damit benötigt die Berechnung von $a_0 + a_1$ wenigstens eine Addition. Für den Induktionsschritt sei P ein Programm, das obige Summe berechnet und $p_1(a) \pm p_2(a)$ die erste wesentliche Addition oder Subtraktion in P, wobei die beiden Summanden je reine Produkte seien. Wir können oBdA annehmen, daß in p_1 der Faktor a_n vorkommt. Setzen wir $a_n := 0$, so transformieren wir P in ein Programm, das $\sum_{i=}^{n-1} a_i$ berechnet, dafür aber eine Addition weniger benötigt als P. Da nach IV für diese Berechnung wenigstens $n-1$ Additionen oder Subtraktionen benötigt werden, enthält P somit wenigstens n Additionen oder Subtraktionen . □

Interessant ist, daß man *jedes spezielle* Polynom $p(x)$ vom Grad $n \geq 5$ über $R = \mathbf{C}$ stets mit weniger als n nichtskalaren Operationen berechnen kann, d.h. für jedes *spezielle* Polynom das Hornerschema nicht optimal ist. Wir hatten das bereits für $p = X^n$ gesehen. Die dem entsprechenden Vorgehen zugrunde liegende Idee heißt *Präkonditionierung*: Division mit Rest liefert für jedes $u \in \mathbf{C}$ eindeutig bestimmte Polynome $q(x), r(x)$ mit $deg\ r \leq 1$ und

$$p(x) = (x^2 - u) \cdot q(x) + r(x).$$

Wir wollen erreichen, daß $r(x)$ konstant ist. Für $v \neq 0$ mit $v^2 = u$ erhalten wir $p(v) = r(v)$ und $p(-v) = r(-v)$. r ist also genau dann konstant, wenn $p(v) = p(-v)$ gilt, d.h. wenn $v \neq 0$ eine Nullstelle von $c(x) := \sum_{i \equiv 1\ (2)} a_{n-i} x^i$ ist. Wendet man das Verfahren rekursiv auf $q(x)$ an und zählt die Kosten für das Präkonditionieren nicht mit, so kann man damit p mit $n + \lfloor \frac{n}{2} \rfloor + 2$ arithmetischen Operationen berechnen, davon $\lfloor \frac{n}{2} \rfloor$ nichtskalare Multiplikationen. Weiter unten werden wir sehen, daß das Verfahren auch im Fall, wenn $c(x)$ keine Nullstellen $v \neq 0$ besitzt, entsprechend modifiziert werden kann.

1.3 Multiplikation zweier Matrizen

> *Problemstellung: Finde ein optimales Verfahren zur Multiplikation zweier $n \times n$–Matrizen mit Einträgen aus R.*

Das klassische Verfahren benötigt n^3 Multiplikationen und weitere $n^3 - n^2$ Additionen und hat keine offensichtliche Verbesserung. Strassen (1969) bemerkte jedoch, daß man das Produkt zweier 2×2–Matrizen statt mit 8 bereits mit 7 Multiplikationen bestimmen kann: Seien

$$A := \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \text{und} \qquad B := \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

zwei solche Matrizen und

$$C := A \cdot B = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

ihr Produkt, so kann man aus den sieben Produkten

$$
\begin{aligned}
P_1 &= (a_{11}+a_{22})(b_{11}+b_{22}) \\
P_2 &= (a_{21}+a_{22})b_{11} \\
P_3 &= a_{11}(b_{12}-b_{22}) \\
P_4 &= a_{22}(b_{21}-b_{11}) \\
P_5 &= (a_{11}+a_{12})b_{22} \\
P_6 &= (a_{21}-a_{11})(b_{11}+b_{12}) \\
P_7 &= (a_{12}-a_{22})(b_{21}+b_{22})
\end{aligned}
$$

C wie folgt berechnen:

$$
\begin{aligned}
c_{11} &= P_1+P_4-P_5+P_7 \\
c_{12} &= P_3+P_5 \\
c_{21} &= P_2+P_4 \\
c_{22} &= P_1+P_3-P_2+P_6
\end{aligned}
$$

Auf den ersten Blick mag es erscheinen, als ob die eine gewonnene Multiplikation durch die 14 zusätzlichen Additionen mehr als aufgewogen wird. Beachtet man aber, das die obigen Formeln von der Kommutativität der Multiplikation keinen Gebrauch machen, so kann man dasselbe Schema auch für die Multiplikation von Matrizen der Größe $n = 2m$ anwenden, indem man dieselben jeweils in 4 $m \times m$–Blöcke zerlegt. Dann sind 18 Additionen von $m \times m$–Matrizen und 7 Multiplikationen derselben, also statt $8m^3$ Multiplikationen und $8m^3 - 4m^2$ Additionen nur $7m^3$ Multiplikationen und $7m^3 + 11m^2$ Additionen in R auszuführen. Einen noch größeren Effekt erreicht man, wenn man diesen Trick rekursiv auf die Multiplikation der Teilmatrizen anwendet. Bezeichnet $M(n)$ die Zahl der Multiplikationen in R, die dieser modifizierte Algorithmus benötigt und ist $n = 2^k$, so erhalten wir $M(1) = 1,\ M(2^{k+1}) = 7\,M(2^k)$ also insgesamt $M(2^k) = 7^k = n^{log(7)}$. Dieser Exponent $log(7) \approx 2.81$ ist kleiner als der klassische Exponent 3. Ist n keine reine Zweierpotenz, so kann man die Matrizen durch Nullelemente entsprechend auffüllen, was nur zu einem Zuwachs um maximal einen konstanten Faktor führt, für die Wachstumsordnung der Komplexität also nicht ausschlaggebend ist. Für die Anzahl $A(n)$ der Additionen bekommt man dasselbe Wachstum: $A(1) = 0,\ A(2^{k+1}) = 18 \cdot (2^k)^2 + 7\,A(2^k)$ liefert ebenfalls $A(n) = O(n^{log(7)})$.

AUFGABE: Finden Sie eine explizite Formel für $A(2^k)$.
$(= 6 \cdot (7^k - 4^k))$

Eine etwas aufwendigere Analyse (Strassen) liefert

Satz 3 *Zwei $n \times n$–Matrizen könne mit höchstens $4.7\,n^{log(7)}$ arithmetischen Operationen multipliziert werden.*

Dies gibt eine obere Schranke für den *Exponenten* $\omega = \omega(k)$ der Matrixmultiplikation, den kleinsten Wert $h \in \mathbf{R}$, so daß die Multiplikation in der Matrixalgebra $Mat_n(k)$ mit $O(n^h)$ arithmetischen Operationen möglich ist. Dieser spielt ob der zentralen Stellung der Matrizenmultiplikation in den verschiedensten Algorithmen eine wichtige Rolle in der arithmetischen Komplexitätstheorie. Heute ist bekannt (Coppersmith und Winograd 1990), daß $\omega < 2.38$ gilt. Die elementare untere Schranke lautet $\omega(k) \geq 2$. Es wird heute allgemein vermutet, daß hier sogar Gleichheit gilt, ohne daß abzusehen ist, wie man eine solche Behauptung beweisen könnte. Wir werden diesen Teil der algebraischen Komplexitätstheorie jedoch nicht betreten, sondern uns nur mit Aspekten der Einordnung von Matrixalgorithmen in parallele Komplexitätsklassen beschäftigen.

2 Polynomprodukt und Fouriertransformation

Ehe wir uns einer genaueren Spezifikation der verwendeten Berechnungsmodelle zuwenden, die vor allem für die Herleitung unterer Schranken notwendig ist, wollen wir eines der zentralen algorithmischen Probleme der algebraischen Komplexitätstheorie näher studieren; die Multiplikation von Polynomen, deren Koeffizienten in einer k-Algebra A liegen (von der wir für die meisten Anwendungen voraussetzen können, daß es sich um einen Integritätsbereich handelt). Dafür genügt es, auf einen intuitiven Algorithmenbegriff zurückzugreifen.

Beschreiben wir also zuerst das Berechnungsmodell für unsere Aufgabe. Diese besteht darin, aus den Koeffizienten zweier gegebener Polynome $a = \sum_{i<m} a_i x^i$ und $b = \sum_{j<n} b_j x^j$ mit $a_i, b_j \in A$ die Koeffizienten des Produkts $c = a\,b = \sum_l c_l x^l$, d.h. $c_l = \sum_{i+j=l} a_i b_j$ für $j = 0, \ldots, m+n-2$ zu berechnen. Unter einer Berechnung verstehen wir dabei eine Folge von Instruktionen (ohne Verzweigungen), die in einer Addition, Subtraktion, Multiplikation oder Division von zwei bereits berechneten Zwischenergebnissen oder Eingaben bestehen. Als weitere Instruktion wollen wir Multiplikationen mit einem Skalar $c \in k$ betrachten. Wir sagen, daß ein Ergebnis *berechnet* ist, wenn es sich unter den Zwischenergebnissen befindet. Wir wollen zwei verschiedene Kostenarten untersuchen: die *totalen arithmetischen Kosten*, die alle arithmetischen Operationen zählen, und die *nichtskalaren Kosten*, die nur die ausgeführten Multiplikationen und Divisionen zählen (ohne skalare Vervielfachung). Die minimalen totalen arithmetischen Kosten einer Berechnung, die ein gegebenes Problem löst, bezeichnen wir als die *totale arithmetische Komplexität* des Problems. Die *nichtskalare Komplexität* wird analog definiert.

Die Komplexität von Addition und Subtraktion zweier Polynome vom Grad $< n$ läßt sich einfach ermitteln. Das klassische Verfahren benötigt n Additionen oder Subtraktionen, womit die nichtskalare Komplexität gleich 0 ist und die totale Komplexität durch n beschränkt wird. Im *generischen Fall*, d.h. wenn die Koeffizienten alle algebraisch unabhängig über k sind, ist dies bereits die richtige Komplexität, denn wir haben dann n algebraisch unabhängige Elemente $a_i \pm b_i$ zu berechnen.

Satz 4 *Symbolische Addition und Subtraktion von zwei univariaten Polynomen vom Grad $< n$ hat die nichtskalare Komlexität 0 und die totale Komplexität $< n$, Letztere Wert ist exakt für generische Polynome.*

Wenden wir uns deshalb der Multiplikation von Polynomen zu. Auf den ersten Blick mag es erscheinen, als ob man in der Tat alle n^2 paarweisen Produkte der Koeffizienten berechnen muß, um daraus die entsprechenden Summen zu bilden. Die Erfahrung mit der Matrixmultiplikation läßt uns allerdings vorsichtiger werden, weil ja nicht diese Produkte, sondern nur einige Linearkombinationen von ihnen benötigt werden. Erinnern wir uns an die Vorlesung über die Basisalgorithmen der Computeralgebra, so wurde dort in der Tat ein Verfahren, die *Karatsuba-Multiplikation*, vorgestellt, die mit $O(n^{log(3)})$ Operationen auskommt. Wir werden im weiteren sehen, daß es noch effizientere Verfahren zur Polynommultiplikation gibt.

Betrachten wir jedoch zunächst die nichtskalare Komplexität. Wir erhalten das folgende auf den ersten Blick überraschende Ergebnis:

Satz 5 *Enthält k genügend Elemente ($|k| > n$, so kann man das Produkt von $a, b \in A[x]$ vom Grad $n := deg(a\,b)$ mit $n+1$ nichtskalaren Operationen berechnen. Dieser Wert ist optimal für generische a, b.*

Beweis : Die Idee des Beweises besteht darin, die Polynome an genügend vielen Werten $\lambda \in k$ zu evaluieren und aus diesen Werten das Produktpolynom durch Interpolation zurückzugewinnen. Für ein solches $\lambda \in k$ gilt

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

Jeder solche Wert kann also durch eine einzige nichtskalare Multiplikation berechnet werden, da in den Klammern k-lineare Kombinationen der Koeffizienten stehen, deren Berechnung als kostenfrei angenommen wurde. Für $n+1$ verschiedene Elemente $\lambda_0, \ldots, \lambda_n \in k$ lassen sich also $g_i = c(\lambda_i)$ mit $n+1$ nichtskalaren Operationen berechnen. Nun gilt aber

$$(g_0, \ldots, g_n)^T = ||\lambda_i^j||(c_0, \ldots, c_n)^T,$$

wobei die Übergangsmatrix eine van der Mondesche Matrix ist, also nichtsingulär. Sie besitzt damit eine Inverse (die ohne nichtskalare Kosten berechnet werden kann, da alle Operationen in k erfolgen) und die $(c_0, \ldots, c_n)$ sind somit k-lineare Kombinationen der $(g_0, \ldots, g_n)$.

Für den Beweis, daß im generischen Fall dies zugleich eine untere Schranke ist, verwenden wir ein Argument der linearen Algebra: Aus den Ausgangsgrößen können wir genau alle Elemente aus $H_0 = k < 1, a_i, b_j\,:\,i, j >$, der k-linearen Hülle der angegebenen Elemente, berechnen. Eine nichtskalare Operation erzeugt ein neues Zwischenergebnis, mit dessen Hilfe wir kostenlos wiederum nur k-lineare Kombinationen bilden können. Ist H_r die entsprechende lineare Hülle, die durch Hinzunahme von r nichtskalaren Zwischenergebnissen entsteht, so gilt also $dim_k(H_r) - dim_k(H_0) \leq r$. Da das Endergebnis alle Koeffizienten $c_l, l = 0, \ldots, n$ enthalten muß, $(1, a_i, b_j, c_l\,:\,i, j, l)$ aber k-linear unabhängig sind (eine lineare Abhängigkeit ergäbe eine algebraische Abhängigkeit zwischen den $(a_i, b_j\,:\,i, j)$), benötigt man also $r \geq n+1$ nichtskalare Schritte, um diese zu erzeugen. □

Berücksichtigen wir alle Operationen, so wird die Polynommultiplikation ein delikates Problem, das noch nicht bis in alle Details verstanden ist. Es stellt sich heraus, daß man in den meisten Fällen mit höchstens $O(n\,log(n))$ Operationen auskommen kann. Es wird vermutet, daß man diese Schranke noch bis auf die triviale untere Schranke $O(n)$ drücken kann, hat aber heute noch keine Idee, wie dies im Allgemeinen möglich sein könnte.

Betrachten wir zunächst den Fall $A = \mathbf{C}$ und versuchen, ein schnelles Multiplikationsverfahren für Polynome $mod\ x^N - 1$ mit $N > 2n$ zu entwerfen. $R := \mathbf{C}[x]/(x^N - 1)$ ist eine endlichdimensionale k-Algebra. Wählen wir $\omega \in \mathbf{C}$ als eine primitive N-te Einheitswurzel, so ist die Abbildung

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

ein Algebra-Isomorphismus in die Algebra der Diagonalmatrizen der Größe N, in der Multiplikation mit $O(N)$ Operationen ausführbar ist. Das Produkt von $a, b \in R$ kann man dann als

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

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

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

beschrieben. Es stellt sich heraus, daß man ein solches Produkt für diese spezielle Matrix besondes effizient berechnen kann: Für $N = 2M$ gilt

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

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

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

Dabei ist

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

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

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

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

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

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

Satz 6 *Das Produkt zweier Polynome vom Grad $< n$ mit komplexen Koeffizienten kann mit $O(n\, log(n))$ arithmetischen Operationen berechnet werden.*

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

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

Insbesondere ist dann ω eine *primitive* N-te Einheitswurzel, d.h. $\omega^k \neq 1$ für $0 < k < N$. Für einen Körper fallen diese beiden Begriffe zusammen. Im Allgemeinen ist wegen der Periodizität der Potenzen von ω der Ausdruck $1 - \omega^k$ ein Nichtnullteiler für $N \not| k$ und gleich 0 sonst.

Lemma 1 *Für eine N-te Hauptwurzel ω gilt $\sum_{i=0}^{N-1} \omega^i = 0$ und damit*

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

sowie für die Ideale in $A[X]$

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

und damit auch

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

AUFGABE: Beweisen Sie dieses Lemma.
Hinweis:
1. Multipliziere mit dem NNT $1 - \omega$.
2. Mit Induktion. Ist $a(x) \prod_{i<j}(x - \omega^i) = b(x)\,(x - \omega^j)$, so liefert Substitution $a(\omega^j) = 0$, weil der Kofaktor ein NNT ist.
3. $X^N - 1$ liegt im Idealdurchschnitt, ist also ein Vielfaches der LHS.

Satz 7 *Sei $N > 1$ so daß $N \cdot 1_A \in A$ eine Einheit in der kommutativen k-Algebra A ist und sei $\omega \in A$ eine N-te Hauptwurzel. Dann ist*

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

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

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

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

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

Die inverse Transformation wird durch die Matrix

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

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

BEWEIS : ϕ ist offensichtlich ein Morphismus mit dem Kern

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

nach obigem Lemma. Daraus folgt ebenso die Formel für die inverse Transformation, was beweist, daß D_N ein Isomorphismus, also ϕ surjektiv ist. Der oben analysierte Algorithmus berechnet dann auch im allgemeinen Fall die genannten Produkte. $\square$

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

Satz 8 *Sei N die kleinste Zweierpotenz größer als n und A eine kommutative Algebra über einem Körper k mit $char(k) \neq 2$. Wenn A eine N-te Hauptwurzel enthält, so kann man das Produkt zweier Polynome $a, b \in A[X]$ mit $deg(ab) = n$ mit $O(n \, log(n))$ arithmetischen Operationen berechnen.*

AUFGABE: Im Fall $char(k) = 2$ kann man eine dreireihige Fouriertransformation mit dem Ansatz $N = 3M$ verwenden. Stellen Sie eine entsprechende DFT-Relation auf und zeigen Sie, daß man auf dieser Basis ebenfalls eine Polynom-Multiplikation mit $O(n \, log(n))$ Operationen entwerfen kann.

3 Berechnungsmodelle

Wir wollen nun das in unserer Analyse zu verwendende Berechnungsmodell genauer spezifizieren. Für die Analyse algebraischer Problemstellungen haben sich statt RAM-Modell und Turingmaschine zwei andere Modelle bewährt: SLPs und Berechnungsbäume.

Die allgemeine Idee eines Modells, das eine schrittweise Berechnung ohne Verzweigungen nachbildet, läßt sich wie folgt beschreiben: Ausgehend von einer Teilmenge I einer k-Algebra A, der *Eingabemenge*, wird über die vier Grundrechenarten sowie skalare Multiplikation mit Elementen aus k eine Ausgabemenge $F \subset A$ berechnet. Indem man den Grundrechenarten verschiedene Kosten zuordnet, bekommt man verschiedene Berechnungsmodelle.

Betrachten wir zunächst ein Beispiel. Wir wollen die rationale Funktion $f := \frac{x^5-1}{x-1}$ berechnen. Folgen wir der Idee des binären Potenzierens, so kann man das etwa mit folgender Zuweisungsfolge erreichen:

X_0:=x; X_1:=X_0*X_0; X_2:=X_1*X_1; X_3:=X_2*X_0;
X_4:=$X_3 - 1$; X_5:=$X_0 - 1$; X_6:=X_4/X_5;

Nennen wir dieses Programm Γ. In einer klassischen Programmiersprache implementiert, kann man mit diesem Programm $f(a)$ für alle $1 \neq a \in \mathbf{R}$ berechnen. Eine Programmiersprache, die etwas klüger ist und Operationen in einer k-Algebra A ausführen kann, wobei es mit einer Fehlermeldung anhält, wenn es eine Division ausführen soll, dessen Nenner in A nicht invertierbar ist. Der Abbruch mit einer Fehlermeldung hängt dann von x und A ab. Wir wollen deshalb das *Paar* $(A; x)$ als Eingabe betrachten und sagen, daß Γ auf dieser Eingabe *ausführbar* ist, wenn es nicht mit einer Fehlermeldung abbricht. In unserem Fall ist eine offensichtlich hinreichende Bedingung $x - 1 \not\in A^*$. So ist etwa Γ ausführbar auf $(k(X,Y); XY-1)$ und nicht ausführbar auf $(k[X,Y]; XY-1)$ (obwohl wegen $\frac{x^5-1}{x-1} = 1 + x + x^2 + x^3 + x^4$ auch im letzteren Fall die Division eindeutig ausführbar ist).

Wir wollen dieses Vorgehen nun etwas stärker formalisieren. Betrachten wir dazu eine Menge Ω, die unser System in der Lage ist auszuführen. Dazu sollten sicherlich die vier

Grundrechenarten $\{+,-,*,/\}$ gehören. Da wir die skalaren Operationen speziell berücksichtigen wollen, sollte die skalare Multiplikation mit Elementen $a \in k$, die wir mit den entsprechenden Elementen aus k identifizieren wollen, zur Verfügung stehen. Schließlich sollten die (0-aren) Operationen, die Konstanten aus k einführen, zur Verfügung stehen, so daß unser gesamtes Arsenal so aussieht: $\Omega = k^c \bigcup k \bigcup \{+,-,*,/\}$ mit

$$\begin{array}{l} + : (A,A) \to A, \quad - : (A,A) \to A, \quad * : (A,A) \to A, \quad / : (A,A^*) \to A, \\ \lambda : A \to A, \quad \lambda^c : () \to A \; mit \; \lambda \in k \end{array}$$

Mit diesen Operationen können wir unser Programm Γ präzisieren zu

X_0:=x; X_1:=X_0*X_0; X_2:=X_1*X_1; X_3:=X_2*X_0;
X_4:=1^c; X_5:=$X_3 - X_4$; X_6:=$X_0 - X_4$; X_7:=X_5/X_6;

Unser Programm besteht also aus einer Folge von Schritten $\Gamma = (\Gamma_1, \ldots, \Gamma_7)$, wobei die einzelnen Schritte formal als Instuktionen der Art

$$\begin{array}{l} \Gamma_1 = (*;0,0), \Gamma_2 = (*;1,1), \Gamma_3 = (*;2,0), \\ \Gamma_4 = (1^c), \Gamma_5 = (-;3,4), \Gamma_6 = (-;0,4), \Gamma_7 = (/;5,6) \end{array}$$

beschrieben werden können. Dies führt zu folgender formaler Definition:

Definition: Sei $\Omega = k^c \bigcup k \bigcup \{+,-,*,/\}$, $n \in \mathbf{N}$, A eine k-Algebra und $a \in A^n$. Das Paar $(A;a)$ nennen wir eine *Eingabe der Länge* n. Die Arität einer Operation $\omega_i \in \Omega$ bezeichnen wir mit ar_i.

(1) (Syntax eines SLPs) Ein *SLP* Γ (über k, der Eingabelänge n) ist eine Folge $(\Gamma_1, \ldots, \Gamma_r)$ von *Instruktionen*
$$\Gamma_i = (\omega_i; u_{i,1}, \ldots, u_{i,ar_i}),$$
mit $\omega_i \in \Omega$ und $u_{i,l} \in \mathbf{Z}, -n < u_{i,l} < i$ für $i = 1, \ldots, ar_i$.

(2) (Semantik eines SLPs) Sei $\Gamma = (\Gamma_1, \ldots, \Gamma_r)$ ein SLP der Eingabelänge n und Γ_i wie eben definiert. Γ heißt *ausführbar* auf der Eingabe $(A;a)$ mit *Ergebnisfolge* $b = (b_{-n+1}, \ldots, b_r) \in A^{n+r}$, wenn $b_i = a_{n+i}$ für $i = -n+1, \ldots, 0$ und
$$b_i = \omega_i(b_{u_{i,1}}, \ldots, b_{u_{i,ar_i}})$$
gilt. $(A;b)$ heißt die von $(A;a)$ produzierte *Ausgabe*.

(3) Wir sagen, daß Γ die Sequenz $F \subseteq A$ *berechnet*, wenn $F \subseteq \{b_{-n+1}, \ldots, b_r\}$ gilt.

Ein SLP kann man durch einen *gerichteten azyklischen Graphen* (DAG) Γ darstellen mit der Knotenmenge $V(\Gamma) = \{i \in \mathbf{Z} : -n < i \leq r\}$ und der Kantenmenge $E(\Gamma) = \{(u_{i,k}, i) : i = 1, \ldots, r, \; k = 1, \ldots, ar_i\}$. Sei $C(\Gamma)$ die Menge der Pfade in diesem Graphen, d.h. die Teilmengen $C \subset V$, die bzgl. der induzierten Ordnungsrelation linear geordnet sind. Da der Graph die kausalen Abhängigkeiten zwischen den einzelnen Programmschritten darstellt, ist seine *Größe* $L(\Gamma) := card(V) = r + n$ ein Maß für die serielle Ausführungszeit des dargestellten Algorithmus, seine *Tiefe* $D(\Gamma) := max_{C \in C(\Gamma)}(card(C))$, d.h. die Länge des längsten Pfades, dagegen ein Maß für die parallele Ausführungszeit, genügend verfügbare Prozessoren vorausgesetzt.

Oft nutzen wir Unterprogrammtechniken, wie etwa zur Berechnung von $\frac{x^{10}-1}{x^2-1}$. Dazu berechnen wir zur Eingabe $(A;X)$ zuerst X^2 und starten dann unser bisheriges Programm mit der Eingabe X^2 statt X. Wir verketten auf diese Weise zwei SLPs, was in der folgenden Definition verallgemeinert wird:

Definition: Sind $\Gamma^{(1)} := (\Gamma_1^{(1)}, \dots, \Gamma_r^{(1)})$ und $\Gamma^{(2)} := (\Gamma_1^{(2)}, \dots, \Gamma_s^{(2)})$ zwei SLPs, die Eingaben der Länge n bzw. $n+r$ erwarten, so bezeichnet man das SLP $\Gamma := (\Gamma_1, \dots, \Gamma_{r+s})$ mit

$$\Gamma_i := \begin{cases} (\omega_i^{(1)}; u_{i,1}^{(1)}, \dots, u_{i,ar_i}^{(1)}) & \text{für } i = 1, \dots, r \\ (\omega_{i-r}^{(2)}; u_{i-r,1}^{(2)} + r, \dots, u_{i-r,ar_{i-r}}^{(2)} + r) & \\ & \text{für } i = r+1, \dots, r+s \end{cases}$$

als die *Verkettung* von $\Gamma^{(1)}$ und $\Gamma^{(2)}$.

Untersuchen wir nun die Menge der Eingaben, auf denen ein SLP ausführbar ist. Ein Programm Γ der Eingabelänge n heißt *konsistent*, wenn es eine Eingabe $(A; a_1, \dots, a_n)$ mit einer (nichttrivialen) k-Algebra A und $a_i \in A$ gibt, auf der es ausführbar ist.

Bemerkung: Ist $\phi : A \longrightarrow A'$ ein k-Algebra-Morphismus und Γ ein SLP, das auf der Eingabe $(A; a_1, \dots, a_n)$ ausführbar ist die Ergebnisfolge $(A; b_1, \dots, b_r)$ liefert, so ist Γ auch auf der Eingabe $(A'; \phi(a)) = (A'; \phi(a_1), \dots, \phi(a_n))$ ausführbar und ergibt die Ergebnisfolge $(A'; \phi(b)) = (A'; \phi(b_1), \dots, \phi(b_r))$. Dies folgt einfach daraus, daß ϕ die Einheiten von A in Einheiten von A' abbildet.

Wir wollen nun in Erweiterung dieser Bemerkung zeigen, daß es zu jedem konsistenten Programm Γ eine *universelle* Eingabe $(U; u)$ gibt, wobei $u = (u_1, \dots, u_n)$ die Algebra U erzeugen, Γ auf $(U; u)$ ausführbar ist und aus der sich jede andere Eingabe $(A; a)$, auf der Γ ebenfalls ausführbar ist, auf die beschriebene Weise als homomorphes Bild einer Abbildung $\phi : U \longrightarrow A$ mit $\phi(u) = a$ herleiten läßt. Nennen wir zwei Eingaben $(A; a)$ und (A', a') *isomorph*, wenn es einen k-Algebra-Isomorphismus $\phi : A \longrightarrow A'$ mit $\phi(a) = a'$ gibt, so ist dieses universelle Objekt $(U; u)$, falls es existiert, eindeutig bis auf Isomorphie bestimmt.

Betrachten wir zunächst wieder die spezielle Berechnungsvorschrift für $\frac{x^5-1}{x-1}$, die wir oben studiert haben. Die verwendete Division führt dazu, daß sie genau auf solchen Eingaben $(A; a)$ ausführbar ist, für die $1 - a \in A^*$ eine Einheit ist. Es ist also nicht ausführbar auf $(k[X]; X)$, aber auf $(U; X)$, wenn U eine Teilalgebra der rationalen Funktionen $k(X)$ ist, die Elemente mit Nennern $(1 - X)^m$ enthält. Die kleinste solche Teilalgebra ist

$$U := \{\frac{g}{(1-X)^m} \ : \ g \in k[X], \ m \in \mathbf{N}\}.$$

Allgemeiner kann man für einen beliebigen Integritätsbereich R dessen Lokalisierung nach einer multiplikativen Menge definieren:

Definition: Eine Teilmenge $S \subset R$ heißt *multiplikativ*, wenn $1 \in S$ und

$$s_1, s_2 \in S \ \Rightarrow \ s_1 \cdot s_2 \in S$$

gilt.

Den Unterring

$$R_S := \{\frac{g}{s} \ : \ g \in R, \ s \in S\} \subset Q(R)$$

des Quotientenkörpers $Q(R)$ von R bezeichnet man als die *Lokalisierung* von R nach S.

AUFGABE: Beweisen Sie, daß R_S ein Ring ist.

Für den Fall, daß $S = \{d^m : m \in \mathbf{N}\}$ aus den Potenzen eines Polynoms $0 \neq d \in R$ gebildet wird, schreiben wir auch R_d statt R_S.

Die Lokalisierung hat die folgende *universelle Abbildungseigenschaft*: Ist $\phi : R \longrightarrow A$ ein Ringhomomorphismus mit $\phi(S) \in A^*$, so kann man ϕ vermöge

$$\phi : \frac{g}{s} \mapsto \phi(g) \cdot \phi(s)^{-1}$$

eindeutig zu einem Homomorphismus $\phi : R_S \longrightarrow A$ fortsetzen.

Satz 9 *Sei Γ ein konsistentes SLP der Eingabelänge n. Dann gibt es eine bis auf Isomorphie eindeutig bestimmte* universelle Eingabe $(U; u)$, *so daß gilt:*

1. *Γ ist auf $(U; u)$ ausführbar.*

2. *Wenn Γ auf $(A; a)$ ausführbar ist, so gibt es einen k-Algebra-Homomorphismus $\phi : U \longrightarrow A$ mit $\phi(u) = a$*

Genauer: U ist die Lokalisierung des Polynomrings $R := k[X_1, \ldots, X_n]$ nach einem geeigneten Polynom $d \in R$, $u = (X_1, \ldots, X_n)$ und Γ ist auf $(A; a)$ genau dann ausführbar, wenn $d(a) = d(a_1, \ldots, a_n) \in A^*$ gilt.

BEWEIS : Es bleibt nur die Existenz der universellen Eingabe $(U; u)$ zu zeigen. Wir führen diesen Beweis mit Induktion. Für $r = 0$ werden keine Instruktionen ausgeführt. In diesem Fall folgt aus der Universalitätseigenschaft von R, daß $d = 1$ genommen werden kann. Nehmen wir also an, daß wir für $\Gamma' := (\Gamma_1, \ldots, \Gamma_{r-1})$ ein $d'(X) \in R$ bereits gefunden haben, für welches $U' := R_{d'}$ die geforderte Universalitätseigenschaft besitzt. Ist $\omega_r \neq /$, so ist Γ_r ebenfalls auf U' ausführbar. Einzig eine partielle Operation $\Gamma_r = (/; i, j)$ bereitet Probleme. Diese ist nach Definition genau dann ausführbar, wenn $b_j(X) = \frac{f(X)}{d'(X)^k}$ invertierbar ist. Setzen wir $d(X) = d'(X) \cdot f(X)$. Dann ist Γ_r auf $U := U'_{b_j} = R_d$ ausführbar. Ist andererseits $(A; a)$ eine Eingabe, auf der $(\Gamma_1, \ldots, \Gamma_r)$ ausführbar ist, so ist insbesondere Γ' ausführbar. Damit existiert ein Homomorphismus $\phi : U' \longrightarrow A$ mit $\phi(x_i) = a_i$, so daß $b_j(a) \in A^*$. Damit kann man aber ϕ auf U fortsetzen. $\square$

In Verallgemeinerung der Begriffe "Größe" und "Tiefe" des DAG Γ definieren wir nun den Begriff der Kosten eines SLP:

DEFINITION: Sei A eine k-Algebra und $\Gamma = (\Gamma_1, \ldots, \Gamma_r)$ mit $\Gamma_i = (\omega_i; \ldots)$ ein SLP über der Operatorfamilie Ω. Sei weiter $c : \Omega \longrightarrow \mathbf{N} \cup \{\infty\}$ eine *Kostenfunktion.*

1. Die gewichtete Knotensumme $L_c(\Gamma) := \sum_{i=1}^{r} c(\omega_i)$ bezeichnen wir als die *seriellen Kosten* des Programms Γ bzgl. der Kostenfunktion c.

2. Die gewichtete Tiefe $D_c(\Gamma) := max_{C \in C(\Gamma)} \left(\sum_{i \in C} c(\omega_i)\right)$ bezeichnen wir als die *parallelen Kosten* des Programms Γ bzgl. der Kostenfunktion c.

3. Als *(serielle) Komplexität* $L_A^c(F \,|\, I)$ der endlichen Output-Menge $F \subset A$ bzgl. der Eingabe $I \subset A$ und der Kostenfunktion c bezeichnen wir die minimalen seriellen Kosten $min_\Gamma(L_c(\Gamma))$, wobei das Minimum über allen SLPs Γ zu nehmen ist, die F aus der Eingabe I berechnen.

4. Analog definieren wir die *parallele Komplexität* $D^c_A(F \mid I) := min_\Gamma(D_c(\Gamma))$.

Eine Reihe von Kostenfunktionen spielen im weiteren eine besondere Rolle. Dies sind die Funktionen

$$\begin{array}{ll} c_*(\omega) = 1 \; \textit{für} \; \omega \in \{*, /\}, & c_*(\omega) = 0 \; \textit{sonst} \\ c_+(\omega) = 1 \; \textit{für} \; \omega \in \{+, -\}, & c_+(\omega) = 0 \; \textit{sonst} \\ c_{tot}(\omega) = 1 \; \textit{für} \; \omega \in k \cup \{+, -, *, /\}, & c_{tot}(\omega) = 0 \; \textit{sonst} \end{array}$$

Entsprechend bezeichnen wir

$L^{tot}_A(F \mid I)$	als *totale Komplexität*,
$L^+_A(F \mid I) := L^{c_+}_A(F \mid I)$	als *additive Komplexität* und
$L^*_A(F \mid I) := L^{c_*}_A(F \mid I)$	als *multiplikative oder nichtskalare Komplexität.*

Es sei angemerkt, daß die Funktionen a^c der Auswahl von Konstanten aus k in allen Modellen nicht mit Kosten belegt sind. Die nichtskalare Komplexität spielt beim Beweis unterer Schranken eine zentrale Rolle. Wir schreiben deshalb auch kurz L_A statt L^*_A. Wenn die besondere Rolle des Grundkörpers k betont werden soll, verwenden wir den Index $L_{k \to A}$. Bezeichnen wir mit $CF(f)$ die Menge der Koeffizienten eines Polynoms $f \in A[X]$, so lautet die Aussage unseres Satzes über die Komplexität der Multiplikation zweier Polynome $a, b \in A[X]$ mit $deg(ab) = n$ in der neuen Notation

$$\begin{array}{ll} L_A(CF(ab) \mid CF(a) \cup CF(b)) \leq n + 1 & \\ L^{tot}_A(CF(ab) \mid CF(a) \cup CF(b)) \in O(n\, log(n)) & \text{wenn } k \text{ FFT erlaubt} \\ L^{tot}_A(CF(ab) \mid CF(a) \cup CF(b)) \in O(n\, log(n)\, log(log(n))) & \text{wenn } k \text{ FFT nicht erlaubt} \end{array}$$

Die folgenden Aussagen sind einfache Konsequenzen der Definition:

Satz 10 *Mit obigen Bezeichnungen gilt*

1. *Für* $F_1 \subseteq F$ *und* $I \subseteq I_1$ *gilt* $L^c_A(F_1 \mid I_1) \leq L^c_A(F \mid I)$.

2. *Ist* $k \subset K$ *eine Körpererweiterung und* A *eine* K*-Algebra, so gilt*

$$L^c_{K \to A}(F \mid I) \leq L^c_{k \to A}(F \mid I).$$

3. *Ist* $\phi : A \longrightarrow A'$ *ein* k*-Algebra-Homomorphismus, so gilt*

$$L^c_{A'}(\phi(F) \mid \phi(I)) \leq L^c_A(F \mid I).$$

4. $L^c_A(F \mid I) \leq L^c_A(F \mid E) + L^c_A(E \mid I)$.

AUFGABE: Beweisen Sie diese Aussagen.

Mit diesen Definitionen können wir das Vektorraum-Dimensionsargument, das wir bereits verwendet haben, explizieren. Bezeichne dazu $H_k(F)$ die k-lineare Hülle, die von $\{1\} \cup F \subset A$ erzeugt wird. Jedes Element aus dieser Hülle kann aus F ohne nichtskalare Kosten erzeugt werden. Da in jedem nichtskalaren Programmschritt nur ein neues Element aus A erzeugt wird, liefert dies den folgenden

Satz 11 *(Dimensionssatz) Für die nichtskalare Komplexität gilt*

$$L_A(F \mid I) \geq dim_k(H_k(I \cup F)) - dim_k(H_k(I)).$$

Eine erste Anwendung des Dimensionssatzes stellt die folgende Präzisierung unseres Satzes über die nichtskalare Komplexität der Polynommultiplikation dar:

Satz 12 *Sei k ein unendlicher Körper.*

(1) Sei $A := \{a_0, \ldots, a_m, b_0, \ldots, b_n\}$, $k[a,b]$ ein Polynomring in $m+n+2$ algebraisch unabhängigen Ausdrücken und $c_l := \sum_{i+j=l} a_i b_j$. Sei weiter $C = \{c_0, \ldots, c_{m+n}\}, J \subset \{0, 1, \ldots, m+n\}$ gegeben und $I := A \cup \{c_j \mid j \in J\}$. Dann gilt

$$L_{k(a,b)}(C \mid I) = L_{k[a,b]}(C \mid I) = m + n + 1 - |J|.$$

(2) Sind $\{a_0, \ldots, a_n\}$ algebraisch unabhängig über k, $I = (a_0, \ldots, a_n)$ und $k[a] = k[a_0, \ldots, a_n]$ und $m \leq n$, so gilt

$$L_{k(a)}(a_0^2, \ldots, a_m^2 \mid I) = L_{k[a]}(a_0^2, \ldots, a_m^2 \mid I) = m + 1.$$

Beweis : Die oberen Schranken haben wir bereits hergeleitet, so daß nur noch die unteren Schranken zu beweisen sind. Diese folgen aber beide aus der algebraischen Unabhängigkeit der Parameter, die die lineare Unabhängigkeit der Ausdrücke in der Vereinigung von Eingabe- und Ausgabemenge nach sich ziehen. $\square$

4 Untere Schranken

In diesem Abschnitt wollen wir weitere Prinzipien kennenlernen, die es erlauben, untere Schranken für Berechnungsprobleme herzuleiten und damit Aussagen über die diesen Problemen inhärente Kompliziertheit abzuleiten.

Nehmen wir dazu zunächst die Betrachtung eines der Probleme aus der Einleitung von neuem auf.

4.1 Berechnung von Polynomen mit Präkonditionierung

Wir hatten bei der Betrachtung des Hornerschemas gesehen, daß dieses nicht optimal ist, wenn wir Präkonditionierung zulassen, d.h. die (kostenlose) Vorabberechnung geeigneter Elemente aus dem Grundkörper k, mit denen die Berechnung des zu evaluierenden Polynoms $f(x) \in k[x]$ effektiver möglich ist. Im Abschnitt 1 ergaben sich die neu einzuführenden Elemente aus k als Nullstellen von Polynomen, weshalb wir den Grundkörper als algebraisch abgeschlossen voraussetzen mußten. Formulieren wir zunächst das dort erzielte Ergebnis in unserer nunmehr eingeführten Notation: Gegeben ist ein Polynom $f(x) = \sum_{i=0}^n a_i x^i \in k[x]$ mit Koeffizienten aus einem Grundkörper k. Gesucht ist die Komplexität $L^c_{k \to K}(f \mid k \cup \{x\})$ der Evaluierung dieses Polynoms, wenn man das Argument x und alle Elemente eines geeigneten Erweiterungskörpers K/k (genauer: eine beliebige endliche Teilmenge derselben), etwa des algebraischen Abschlusses von k, frei zur Verfügung hat. Präkonditionierung bedeutet dabei, aus K geeignete Elemente auszuwählen, die bei der Evaluierung von $f(x)$ hilfreich sein können, jedoch die eigentlichen Rechnungen, die bei der Evaluierung anfallen, als Rechnungen in der k-Algebra K zu betrachten. Insbesondere ist die Multiplikation zweier Elemente aus K, die etwa aus den durch Präkonditionierung gewonnenen Elementen abgeleitet worden sind, eine nichtskalare Multiplikation.

Satz 13 *Sei K ein algebraisch abgeschlossener Erweiterungskörper von k. Dann gilt*

$$L^*_{k\to K}(f\,|\,K\cup\{x\}) \le \lfloor\frac{n}{2}\rfloor + 2$$

und

$$L^{tot}_{k\to K}(f\,|\,K\cup\{x\}) \le n + \lfloor\frac{n}{2}\rfloor + 2.$$

Beweis : Wir betrachten folgenden Ansatz

$$f(x) = g(x^2) + x\cdot h(x^2) \quad \text{mit} \quad g(x) = \sum_{2i\le n} a_{2i}x^i,\ h(x) = \sum_{2i+1\le n} a_{2i+1}x^i.$$

Ist h konstant, so folgt die Behauptung, da wir dann $g(x^2)$ nach dem Hornerschema berechnen können.

Ist h nicht konstant und ζ eine Nullstelle, so liefert der Ansatz

$$h(x) = (x-\zeta)h_1(x),\ g(x) = (x-\zeta)g_1(x) + \eta$$

die Formel

$$f(x) = (x^2-\zeta)f_1(x) + \eta \quad \text{mit} \quad f_1(x) = g_1(x^2) + x\cdot h_1(x^2).$$

Aus der Kenntnis von x^2 und f_1 können wir f durch eine weitere Multiplikation und zwei Additionen berechnen. Bezeichnet $M(n)$ die Anzahl der Multiplikationen zur Evaluierung eines Polynoms vom Grad n, erhalten wir als Rekursionsbeziehung $M(n) = M(n-2)+1$ und mit $M(0) = 1, M(1) = 2, M(2) = 3$ (berücksichtige, daß x^2 berechnet werden muß !) die erste Formel. Für die Anzahl $A(n)$ der Additionen gilt analog $A(n) = A(n-2)+2$ und mit den richtigen Anfangswerten $A(n) = n$, was die zweite Formel beweist. □

Das eben verwendete Verfahren benötigt zur Präkonditionierung die Berechnung von Nullstellen, was im Allgemeinen nur näherungsweise ausgeführt werden kann. Wir wollen deshalb fragen, ob es ein günstiges Präkonditionierungsverfahren gibt, das mit rationalen Operationen auskommt, also k nicht verläßt.

Satz 14 *(Paterson, Stockmeyer 1973) Ist k ein beliebiger Körper der Charakteristik* 0, *$f(x) \in k[x]$ mit $deg(f) < n$, so gilt*

$$L^*_k(f\,|\,k\cup\{x\}) \le \frac{n}{2} + 2\log(n).$$

Genauer: Es existiert ein entsprechendes SLP, das höchstens $\frac{n}{2} + 2\log(n)$ Multiplikationen und $\frac{3n}{2} - 1$ Additionen benötigt.

Beweis : Betrachten wir zunächst monische Polynome $g \in k[x]$ vom Grad $deg(g) = 2^l - 1$ und setzen

$$\mu_l := max(L^*(g\,|\,k\cup\{x^{2^i}, i=0,\dots,l-1\})\mid deg(g) = 2^l-1,\ lc(g) = 1).$$

Mit $m = 2^{l-1}$ können wir ein solches Polynom

$$g(x) = x^{2m-1} + a_{2m-2}x^{2m-2} + \ldots + a_1x + a_0$$

in der Form

$$g(x) = (x^m + \delta) g_1(x) + g_2(x)$$

mit monischen Polynomen

$$\begin{aligned} g_1(x) &= x^{m-1} + a_{2m-2}x^{m-2} + \ldots + a_{m+1}x + a_m \\ g_2(x) &= x^{m-1} + b_{m-2}x^{m-2} + \ldots + b_1 x + b_0 \end{aligned}$$

und

$$\delta = a_{m-1} - 1, b_i = a_i - a_{m+i}\delta$$

darstellen. Dies ergibt eine rekursive Vorschrift zur Berechnung monischer Polynome, die $\mu_{l+1} \leq 2\mu_l + 1$ und damit $\mu_l \leq 2^{l-1} - 1$ liefert, da x^m für Zweierpotenzen m als bekannt vorausgesetzt wird. Die Evaluierung nicht unbedingt monischer Polynome $f \in k[x]$ vom genauen Grad $deg(f) = 2^l - 1$ kann man auf diesen Fall durch eine Division durch den Leitkoeffizienten zurückführen (Präkonditionierung !); das korrekte Ergebnis bekommt man durch die umgekehrte Operation (im SLP !), was

$$L_k^*(f \mid k \cup \{x^{2^i}, i = 0, \ldots, l-1\}) \leq 2^{l-1}$$

impliziert.

Sei nun f ein beliebiges Polynom vom Grad $deg(f) = n - 1$, das wir oBdA als *dicht* voraussetzen wollen, d.h.

$$f(x) = \sum_{i<n} a_i x^i \quad \text{mit} \quad \forall\, a_i \neq 0.$$

Dies kann man erreichen, indem man $f_1(x) := f(x + c)$ evaluiert, wobei $c \in k$ so zu wählen ist, daß $f^{(i)}(c) \neq 0, i = 0, \ldots, n-1$ gilt (hier wird $char\ k = 0$ verwendet).

Aufgabe: Zeigen Sie, daß $f_1(x)$ dicht ist.

Sei weiter

$$n = 2^{l_0} + 2^{l_1} + \ldots + 2^{l_s} \quad \text{mit} \quad l_0 > l_1 > \ldots > l_s$$

$$f = f_0 + f_1\, x^{2^{l_0}} + f_2\, x^{2^{l_0} + 2^{l_1}} + \ldots + f_s\, x^{2^{l_0} + 2^{l_1} + \ldots + 2^{l_{s-1}}}$$

mit Polynomen f_i jeweils vom Grad $deg(f_i) = 2^{l_i} - 1$, da f dicht ist.

Wir berechnen nun f wie folgt: Zuerst bestimmen wir durch l_0-maliges fortgesetztes Quadrieren die Potenzen $x^2, x^4, \ldots, x^{2^{l_0}}$. Dann berechnen wir wie oben die Polynome $f_0, \ldots, f_s$ mit maximal

$$\sum_i 2^{l_i - 1} = \frac{n}{2}$$

Multiplikationen. Schließlich berechnen wir wie im Hornerschema

$$f = (\ldots (f_s\, x^{2^{l_{s-1}}} + f_{s-1}) x^{2^{l_{s-2}}} + \ldots) x^{2^{l_0}} + f_0$$

mit weiteren $s \leq l_0 \leq \log(n)$ Multiplikationen. Wir erhalten also insgesamt

$$L_k^*(f \mid k \cup \{x\}) \leq l_0 + \frac{n}{2} + s \leq \frac{n}{2} + 2\log(n).$$

Zählen wir auf demselben Weg die Additionen, so erhalten wir den zweiten Teil der Behauptung. $\square$

Aufgabe: (1) Finden Sie ein SLP, das unter Verwendung geeigneter komplexer Zahlen das Polynom

$$\sqrt{2}\, x^6 + i\, x^5 + x^4 + (1-\sqrt{2})\, x^2 - i\, x - 2$$

mit 5 Multiplikationen und 5 Additionen berechnet.

(2) Finden Sie ein SLP, das

$$3x^6 + 2x^5 + x^4 - 2x^2 - 2x - 2$$

unter Verwendung von 3 nichtskalaren Multiplikationen, 2 Multiplikationen mit ganzen Zahlen und 5 Additionen berechnet.

4.2 Untere Schranken und Transzendenzgrad

In diesem Abschnitt wollen wir eine weitere Methode zur Herleitung unterer Schranken kennenlernen, die den Transzendenzgrad einer Körpererweiterung ausnutzt. Sei dazu im folgenden K eine Erweiterung des Grundkörpers k. Wir betrachten den Polynomring $K[x_1,\ldots,x_n]$ mit einer fixierten Termordnung und den zugehörigen Quotientenkörper $K(x_1,\ldots,x_n)$ der rationalen Funktionen mit Koeffizienten aus K. Den Transzendenzgrad der Körpererweiterung K/k bezeichnen wir mit $trdeg_k(K)$.

Definition: Sei $F=\{f_1,\ldots,f_r\}\subset K(x_1,\ldots,x_n)$ eine endliche Menge von rationalen Funktionen mit Koeffizienten aus K und $f_i=\frac{p_i}{q_i}$ die eindeutig bestimmte Darstellung der rationalen Funktion f_i als Quotient zweier teilerfremder Polynome mit $lc(g)=1$. Als *Koeffizientenkörper* $Coeff(F)$ der Menge F über dem Grundkörper k bezeichnen wir den Körper, der über k von den Koeffizienten der Polynome $p_1,\ldots,p_r,q_1,\ldots,q_r$ erzeugt wird.

Beispiel:

$$\frac{x^2-\sqrt{2}-1}{x-\sqrt{2}+1}=\frac{(\sqrt{2}-1)x^2-1}{(\sqrt{2}-1)x-3+2\sqrt{2}}=\frac{x^3+(\sqrt{2}+1)x^2+(\sqrt{2}+1)x-(3+2\sqrt{2})}{x^2+2x-1}$$

Lemma 2 *Ist für $i=1,\ldots,r$ $f_i=\frac{g_i}{h_i}$ eine beliebige (möglicherweise ungekürzte) Darstellung der rationalen Funktion f_i, so gilt*

$$Coeff(F)\subseteq Coeff(g_1,\ldots,g_r,h_1,\ldots,h_r).$$

Beweis : Bezeichnen wir den Körper auf der rechten Seite mit K_0 und setzen p_i,q_i mit unbestimmten Koeffizienten an, so erhalten wir durch Koeffizientenvergleich aus den Relationen $g_iq_i=h_ip_i$ ein lineares Gleichungssystem in diesen unbestimmten Koeffizienten. Da wir andererseits wissen, daß die Darstellung $f_i=\frac{p_i}{q_i}$ eindeutig ist, hat dieses lineare Gleichungssystem also eine eindeutig bestimmte Lösung, die sich nach dem Gaußalgorithmus rational durch die Koeffizienten des Gleichungssystems, also die Koeffizienten der Polynome g_i,h_i ausdrücken läßt. $\square$

In den folgenden Überlegungen untersuchen wir die Komplexität

$$L^c(F):=L^c_{k\to K(X)}(F\,|\,K\cup k(X))\quad\text{mit}\quad c\in\{*,+,tot\}.$$

Im Gegensatz zum vorigen Kapitel bedeutet das, daß wir als Präkonditionierung nicht nur beliebige Zahlen aus dem Erweiterungskörper K und die Eingabesequenz $X=(x_1,\ldots,x_n)$ zulassen, sondern auch rationale Funktionen mit Koeffizienten aus k.

Die zentrale Aussage dieses Abschnitts ist der folgende

Satz 15 *Sei K/k eine Körpererweiterung und $F \subset K(x_1, \ldots, x_n) =: K(X)$ eine endliche Menge rationaler Funktionen in $x_1, \ldots, x_n$ über K. Dann gilt*

$$L^*(F) \geq \frac{1}{2} \left(trdeg_k(Coeff(F)) - |F| \right)$$

und

$$L^+(F) \geq trdeg_k(Coeff(F)) - |F|.$$

BEWEIS :

Wir haben zu zeigen, daß für jedes SLP $\Gamma = (\Gamma_1, \ldots, \Gamma_t)$ und jede zulässige Eingabe $\mathbf{b} := (b_{-N+1}, \ldots, b_0) \in (K \cup k(X))^N$, für die Γ auf $(K(X); \mathbf{b})$ die Menge F berechnet, d.h. für deren Ausgabemenge $F \subset \{b_{-N+1}, \ldots, b_t\}$ gilt, die Komplexität $L^c(\Gamma)$ durch die angegebenen unteren Schranken beschränkt ist.

Für die nichtskalare Schranke zeigen wir dazu mit Induktion nach t, daß $c_i \in K$, $u_i, v_i \in K[X]$, $i = 1, \ldots, t$ existieren, so daß $b_i = c_i + \frac{u_i}{v_i}$ und

$$trdeg_k(Coeff(u_{-N+1}, \ldots, u_t, v_{-N+1}, \ldots, v_t)) \leq 2 \cdot L^*(\Gamma)$$

gilt.

Daraus folgt die Behauptung: Ist $F = \{b_i \,|\, i \in I\}$ die Menge der zu erzeugenden rationalen Funktionen, so gilt nach Lemma für alle zulässigen Γ

$$trdeg_k(Coeff(F)) \leq trdeg_k(Coeff(\bigcup_{i \in I} \{u_i, v_i\})) + |F|$$

und damit

$$trdeg_k(Coeff(F)) \leq 2 \cdot L^*(F) + |F|.$$

Für $i \leq 0$ setzen wir

$$\begin{array}{ll} c_i = b_i, u_i = 0, v_i = 1 & \text{wenn } b_i \in K \\ c_i = 0, u_i, v_i \in k[X] & \text{wenn } b_i = \frac{u_i}{v_i} \in k[X]. \end{array}$$

Bezeichnen wir $K_i := Coeff(u_{-N+1}, \ldots, u_i, v_{-N+1}, \ldots, v_i)$, so gilt damit $K_0 \subseteq k$.

Ist nun $\Gamma_t = (\pm; \mu, \nu)$, d.h. $b_t = b_\mu \pm b_\nu$, setzen wir wegen

$$b_\mu \pm b_\nu = c_\mu \pm c_\nu + \frac{u_\mu v_\nu \pm v_\mu u_\nu}{v_\mu v_\nu}$$

$c_t = c_\mu \pm c_\nu,\ u_t = u_\mu v_\nu \pm v_\mu u_\nu,\ v_t = v_\mu v_\nu$, womit $K_{t-1} = K_t$ gilt.

Ist dagegen $\Gamma_t = (*; \mu, \nu)$, d.h. $b_t = b_\mu * b_\nu$, setzen wir

$$c_t = 0, \quad \frac{u_t}{v_t} = \frac{c_\mu c_\nu v_\mu v_\nu + c_\mu u_\nu v_\mu + c_\nu u_\mu v_\nu + u_\mu u_\nu}{v_\mu v_\nu}.$$

Dann gilt $K_t \subseteq K_{t-1}(c_\mu, c_\nu)$, d.h. der Transzendenzgrad steigt um höchstens 2. Analog gehe man für eine Division vor.

Skalare Vervielfachung mit Elementen $\lambda \in k$ und Hereinnahme von Konstanten ändert den Koeffizientenkörper ebenfalls nicht.

Für die additive Komplexitätsschranke gehen wir analog vor und zeigen, daß $c_i \in K$, $u_i, v_i \in K[X],\ i = 1, \ldots, t$ existieren, so daß $b_i = c_i \cdot \frac{u_i}{v_i}$ und

$$trdeg_k(Coeff(u_{-N+1}, \ldots, u_t, v_{-N+1}, \ldots, v_t)) \leq L^+(\Gamma)$$

gilt.

Für $i \leq 0$ setzen wir die Variablen ähnlich wie oben

$$\begin{array}{ll} c_i = b_i, u_i = v_i = 1 & \text{wenn } b_i \in K \\ c_i = 1, u_i, v_i \in k[X] & \text{wenn } b_i = \frac{u_i}{v_i} \in k[X]. \end{array}$$

Ist nun $\Gamma_t = (\pm; \mu, \nu)$, d.h. $b_t = b_\mu \pm b_\nu$, und z.B. $c_\mu \neq 0$, so setzen wir wegen

$$b_\mu \pm b_\nu = c_\mu \cdot \frac{u_\mu v_\nu \pm d v_\mu u_\nu}{v_\mu v_\nu}$$

$c_t = c_\mu,\ u_t = u_\mu v_\nu d \pm v_\mu u_\nu,\ v_t = v_\mu v_\nu$ mit $d = \frac{c_\mu}{c_\nu}$, womit $K_t \subseteq K_{t-1}(d)$ gilt.

Für die restlichen möglichen Operationen kann man offensichtlich $K_{t-1} = K_t$ erreichen. □

Korollar 1 *Ist $F = \{f_1, \ldots, f_t\} \subset K[x_1, \ldots, x_n]$ eine Menge von Polynomen ohne konstanten Term, so gilt*

$$L^*(F) \geq \frac{1}{2} trdeg_k(Coeff(F)).$$

Beweis : Wende den Satz auf die Polynome $F' = \{f_1 + Y_1, \ldots, f_t + Y_t\}$ mit neuen Unbestimmten $Y_1, \ldots, Y_t$ an. Ein optimales SLP zur Berechnung von F kann zu einem SLP zur Berechnung von F' durch Hinzufügen von t Additionen erweitert werden. Da außerdem $trdeg_k(Coeff(F')) = trdeg_k(Coeff(F)) + t$ gilt, folgt die Behauptung. □

Wir erhalten als weitere Konsequenz eine untere Schranke für die Komplexität der Evaluierung eines univariaten Polynoms mit Präkonditionierung:

Korollar 2 *(Motzkin 1955, Belaga 1961) Ist $f = \sum_{i=0}^n a_i x^i \in K[x]$ ein Polynom mit Koeffizienten, die algebraisch unabhängig über einem Unterkörper k sind, so gilt $L^*(f) \geq \frac{n}{2}$ und $L^+(f) \geq n$.*

Wie wir oben gesehen hatten, ist diese Schranke, wenigestens für Körper der Charakteristik Null, asympotisch optimal.

Betrachten wir als eine weitere Anwendung nun die Evaluierung einer rationalen Funktion.

Korollar 3 *(Motzkin 1955) Ist f die rationale Funktion*

$$f = \frac{a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0}{x^n + b_{n-1} x^{n-1} + \ldots + b_0},$$

wobei $a_0, \ldots, a_n, b_0, \ldots, b_{n-1} \in K$ algebraisch unabhängig über k sind, so gilt $L^*(f) = n,\ L^+(f) = 2n,\ L^{tot}(f) = 3n$.

Beweis : Daß die angegebenen Werte untere Schranken sind, folgt aus dem Satz. Es bleibt also nur noch ein Verfahren anzugeben, mit dem man diese Werte auch erreichen kann. Dieses leitet sich aus der engen Verbindung zwischen Euklidschem Algorithmus und Kettenbrüchen her: Ist $f = \frac{p}{q}$ wie oben gegeben, so liefert Division mit Rest $p = a_n q + d_n r$ mit $d_n \in K$, einem Polynom r vom Grad $n-1$ und $lc(r) = 1$ und $q = x \cdot r + s$, wobei s ebenfalls ein Polynom vom Grad $n-1$ ist. Diese Beziehungen kann man in der Formel

$$f = a_n + \frac{d}{x + \frac{s}{r}}$$

zusammenfassen, in der der Bruch $\frac{s}{r}$ dieselbe Struktur wie $\frac{p}{q}$ hat, nur Zähler und Nenner haben den Grad $n-1$. Unter Kenntnis von $\frac{s}{r}$ benötigen wir also eine weitere Division und zwei weitere Additionen zur Berechnung von f. $\square$

4.3 Die Substitutionsmethode

In diesem Kapitel wollen wir ausschließlich die nichtskalare Komplexität untersuchen. Da hierfür die Bildung von Linearkombinationen bereits bekannter Größen keine Kosten verursacht, wollen wir den Begriff der *Berechnungssequenz* einführen.

Definition: Ist $I \subset A$ eine Eingabemenge, so heißt eine Sequenz $(g_1, \ldots, g_r) \in A^r$ *Berechnungssequenz*, wenn für jedes $i = 1, \ldots, r$ $u_i, v_i \in H_k(I, g_1, \ldots, g_{i-1})$ existieren, so daß $g_i = u_i \cdot v_i$ oder $v_i \in A^*$ und $g_i = \frac{u_i}{v_i}$ gilt.

Offensichtlich fällt die Länge einer kürzesten Berechnungssequenz mit der nichtskalaren Komplexität eines Problems zusammen.

Wir wollen in diesem Abschnitt die Idee aus der Einleitung verfeinern, die den Beweis lieferte, daß das Hornerschema bzgl. Additionen optimal ist. Unetrsuchen wir dazu zuerst zwei Teilfragen:

(1) Was ist für Unbestimmte $A := (a_1, \ldots, a_n)$ im Polynomring $k[A] = k[a_1, \ldots, a_n]$ die Komplexität der Berechnung von $f = \sum_{i=1}^n a_i^2$? Ist k der Körper der komplexen Zahlen, so kommt man für gerades n mit $\frac{n}{2}$ Multiplikationen aus, wenn man die gesuchte Summe als

$$\sum_{i=1}^{n} a_i^2 = \sum_{j=1}^{n/2} (a_{2j-1} + i a_{2j})(a_{2j-1} - i a_{2j})$$

berechnet. Über $\mathbf{R}[A]$ sind solche Vereinfachungen auf den ersten Blick nicht möglich. Daß es generell nicht geht, wollen wir jetzt beweisen. Sei dazu $(g_1, \ldots, g_r)$ eine optimale Berechnungssequenz und $g_1 = uv$ mit $u, v \in H_{\mathbf{R}}(a_1, \ldots, a_n)$ affinen $(\mathbf{R})$-linearen Kombinationen der $a_1, \ldots, a_n$. Sei oBdA $u = a_n - \sum_{i<n} c_i a_i - c_0$ mit $c_i \in \mathbf{R}$, was durch Umbenennung und Umskalierung ohne Probleme erreicht werden kann. Betrachten wir nun den $\mathbf{R}$-Algebra-Homomorphismus

$$\sigma : \mathbf{R}[a_1, \ldots, a_n] \longrightarrow \mathbf{R}[a_1, \ldots, a_{n-1}] \quad \text{induziert durch} \quad a_n \mapsto \sum_{i<n} c_i a_i + c_0.$$

Dann ist $(\sigma(g_1), \ldots, \sigma(g_r))$ eine Berechnungssequenz für

$$\sigma(f) = a_1^2 + \ldots + a_{n-1}^2 + (\sum_{i<n} c_i a_i + c_0)^2,$$

in der der erste Berechnungsschritt $\sigma(g_1) = 0$ weggelassen werden kann. Die Komplexität von $\sigma(f)$ in $\mathbf{R}[a_1, \ldots, a_{n-1}]$ ist also höchstens $r-1$. Auf dieselbe Weise können wir weitere Substitutionen vornehmen, bis wir zu einer Funktion

$$h := a_1^2 + \ldots + a_m^2 + A_{m+1}^2 + \ldots + A_n^2$$

der Komplexität 0 kommen, wobei $m > 1$ gilt, wenn $r < n$ war. Hier sind A_i, $i > m$, durch affin lineare Substitutionen entstanden, haben also die Gestalt $A_i = \sum_{j \le m} c_{ij} a_j + d_i$. Ein Polynom der multiplikativen Komplexität 0 ist aber affin linear in den Eingangsgrößen, unser Polynom für $m > 1$ dagegen nicht, da der Koeffizient vor a_1^2 gerade $1 + \sum_{i>m} c_{i1}^2 > 0$ ist. Damit haben wir also

$$L^*_{\mathbf{R}[A]}(a_1^2 + \ldots + a_n^2 \mid A) = n$$

bewiesen.

(2) Untersuchen wir als nächstes, wie sich das Hornerschema in die nichtskalare Komplexität der Evaluierung univariater Polynome mit unbestimmten Koeffizienten einordnet. Wir wissen bereits, daß es nicht optimal ist, wenn Präkonditionierung zugelassen wird. Ostrowski vermutete, daß ohne solche Präkonditionierung das Hornerschema optimal ist. Wir wollen nun zeigen, daß für $f = \sum_{i=0}^n a_i x^i$ und $A = k(x)[a_0, \ldots, a_n]$

$$L^*_A(f \mid k(x) \cup \{a_0, \ldots, a_n\}) = n$$

gilt. Sei dazu wie oben $(g_1, \ldots, g_r)$ eine optimale Berechnungssequenz und $g_1 = uv$ oder $g_1 = u/v$ mit $u, v \in H_k(a_0, \ldots, a_n) + k(x)$. u und v können nicht beide aus $k(x)$ stammen, da sonst g_1 in der Eingabemenge zur Verfügung steht, also nicht extra berechnet werden müßte. Da im Fall $g_1 = u/v$ für den Nenner $v \in A^* = k(x)$ gilt, können wir $u \notin k(x)$ und damit wie oben oBdA $u = a_n - \sum_{i<n} c_i a_i - d$ mit $c_i \in k, d \in k(x)$ annehmen und das Bild $\sigma(f)$ unter dem durch die entsprechende Substitution induzierten Homomorphismus

$$\sigma : k(x)[a_1, \ldots, a_n] \longrightarrow k(x)[a_1, \ldots, a_{n-1}]$$

betrachten. Wie oben ist die Komplexität der Berechnung von $\sigma(f)$ höchstens $r-1$. Wiederholen wir das Argument, so landen wir schließlich bei einem Polynom

$$h = \sum_{i=0}^{n} A_i x^i \in k(x)[a_i : i \in I] \quad \text{mit} \quad A_i = \begin{cases} a_i & \text{für } i \in I \\ \sum_{j \in I} c_{ij} a_j + d_i & \text{für } i \notin I \end{cases}$$

mit $c_{ij} \in k, d_i \in k(x)$, wobei I für die Menge der nicht ersetzten Indizes steht. h liegt einerseits in der k-linearen Hülle $H_k(a_0, \ldots, a_n) + k(x)$, andererseits gilt

$$h = \sum_{i \in I} \left(x^i + \sum_{j \notin I} c_{ji} x^j \right) a_i + \sum_{j \notin I} d_j x^j .$$

Damit dieser Ausdruck aber k-linear in $a_0, \ldots, a_n$ wird, muß $I \subseteq \{0\}$ sein. Daraus folgt unmittelbar die zu beweisende Schranke.

Untersuchen wir nun, inwieweit man die vorgestellte Methode auf *rationale Funktionenkörper* übertragen kann. Das Hauptproblem, das einer direkten Übertragung unserer bisherigen Argumentation entgegensteht, ist die Tatsache, daß man einen durch die Substitution $x_i \mapsto a_i \in A$ induzierten k-Algebra-Homomorphismus

$$\sigma : k[x_1, \ldots, x_n] \longrightarrow A$$

nicht auf alle rationalen Funktionen $f(X) = \frac{p(X)}{q(X)} \in k(X)$ übertragen kann, sondern nur auf solche, für die $q(a) \in A^*$ gilt.

$$D(\sigma) := \{\frac{g}{h} \in k(X) \mid \sigma(h) \in A^*\}$$

ist dabei die größte Unter-k-Algebra von $k(X)$, auf die man σ fortsetzen kann.

(3) Betrachten wir die Fragestellung (1) in dem neuen Modell. Sei zunächst $G = (g_1, \ldots, g_r)$ eine optimale Berechnungssequenz für $f = a_1^2 + \ldots + a_n^2$ über $\mathbf{R}(A)$ modulo $\{a_1, \ldots, a_n\}$ und $g_1 = uv$ oder $g_1 = v/u$ mit $u = a_n - \sum_{i<n} c_i a_i - c_0$ wie unter (1). Wie gesagt, kann man den dort betrachteten Homomorphismus σ nicht zu einem solchen des Funktionenkörpers $\mathbf{R}(a_1, \ldots, a_n)$ fortsetzen. Das ist aber für unsere Argumentation auch nicht notwendig: Fassen wir alle in der Berechnungssequenz G auftretenden Nenner zusammen, sehen wir, daß diese f nicht nur über $\mathbf{R}(A)$, sondern bereits über einer Lokalisierung $\mathbf{R}[A]_d$ berechnet. Wir modifizieren deshalb unser bisheriges Vorgehen folgendermaßen: Statt eines Homomorphismus σ betrachten wir eine ganze Familie von Homomorphismen

$$\sigma_\lambda : \mathbf{R}[a_1, \ldots, a_n] \longrightarrow \mathbf{R}(a_1, \ldots, a_{n-1}),$$

die durch $a_n \mapsto \sum_{i<n} c_i a_i + c_0 + \lambda$, d.h. $\sigma_\lambda(u) = \lambda$ gekennzeichnet sind. Einen solchen Homomorphismus kann man offensichtlich dann auf $\mathbf{R}[A]_d$ fortsetzen, wenn

$$\sigma_\lambda(d) = d(a_1, \ldots, a_{n-1}, \sum_{i<n} c_i a_i + c_0 + \lambda) \neq 0$$

gilt. Da $d \in \mathbf{R}[a_1, \ldots, a_n] \sim \sigma_\lambda(d) \in \mathbf{R}[a_1, \ldots, a_{n-1}][\lambda]$ ein nichttriviales Polynom ist, verschwindet der Ausdruck aber nur für endlich viele Werte von $\lambda \in \mathbf{R}$. Für jedes andere λ berechnet $(\sigma_\lambda(g_1), \ldots, \sigma_\lambda(g_r))$ den Wert $\sigma_\lambda(f)$, wobei der erste Berechnungsschritt $\sigma_\lambda(g_1) = \sigma_\lambda(u)\sigma_\lambda(v) = \lambda\sigma_\lambda(v)$ auch ohne nichtskalare Multiplikation möglich ist. Wir haben also aus G eine Berechnungssequenz von $\sigma_\lambda(f)$ in $\mathbf{R}(a_1, \ldots, a_{n-1})$ mit $r-1$ Schritten hergeleitet und können damit genau wie in (1) argumentieren.

(4) Wir wollen nun zeigen, daß das Hornerschema auch bei Zulassung von Divisionen noch optimal ist. Wir könnten ähnlich wie im Punkt (3) vorgehen, wollen hier aber einen weiteren Zugang verwenden. Ist $(g_1, \ldots, g_r)$ wieder eine optimale Berechnungssequenz und $g_1 = uv$ oder $g_1 = u/v$, so wollen wir ein abgeleitetes Berechnungsproblem in $k(a_0, \ldots, a_n, x)$ als $k(v)$-Algebra studieren. Über dieser wird der erste Schritt skalar und kann aus der Berechnungssequenz weggelassen werden. Statt durch eine Substitution wird hier die Reduktion also durch eine Erweiterung des Grundkörpers erreicht.

Satz 16 *(Pan 1966) Seien $a_0, \ldots, a_n, x$ Unbestimmte über einem Körper k. Dann gilt*

$$L^*_{k \to k(a,x)} \left(\sum_{i=0}^{n} a_i\, x^i \mid k(x) \cup \{a_0, \ldots, a_n\} \right) = n.$$

Beweis : Der Vektorraum $H_k(a_0,\ldots,a_n)+k(x)$ läßt sich als die direkte Summe der beiden Unterräume $k(x)$ und des durch $a_0,\ldots,a_n$ aufgespannten Raums L darstellen.

Zum Beweis des Satzes zeigen wir die Gültigkeit des folgenden allgemeineren Sachverhalts:

> Sind $A_0,\ldots,A_n \in L,\ b_0,\ldots,b_n \in k(X)$ und
> $$L^*_{k\to k(a,x)}\left(\sum_{i=0}^{n}(A_i+b_i)\,x^i \mid k(x)\cup\{a_0,\ldots,a_n\}\right)\le r,$$
> so hat der durch $A_0,\ldots,A_n$ aufgespannte k-Unterraum eine Dimension $\le r+1$.

Mit $A_i=a_i, b_i=0$ folgt dann die Aussage unseres Satzes.

Für $r=0$ ist diese Behauptung einfach zu sehen: $\sum_{i=0}^{n}(A_i+b_i)\,x^i \in L\oplus k(x)$ impliziert $A_1=\ldots=A_n=0$.

Sei also $r>0$, $(g_1,\ldots,g_r)$ eine optimale Berechnungssequenz für $\sum_{i=0}^{n}(A_i+b_i)\,x^i$ im Modell $k\to k(a,x)$ modulo $k(x)\cup\{a_0,\ldots,a_n\}$ und $g_1=uv$ oder $g_1=u/v$ mit $u,v\in L+k(x)$, wobei wir wegen der Optimalität $u\notin k(x)$ oder $v\notin k(x)$ voraussetzen können.

Sei zunächst $v\notin k(x)$, wobei wir wieder oBdA $v=a_n-\sum_{i<n}c_ia_i-d$ mit $c_i\in k, d\in k(x)$ annehmen können. Nach Konstruktion ist die Komplexität von $\sum_{i=0}^{n}(A_i+b_i)\,x^i$ in $k(v)\to k(a,x)$ höchstens $r-1$. Mit $A_i=\sum_j c_{ij}a_j,\ c_{ij}\in k$ folgt weiter

$$\begin{aligned}A_i+b_i&=\textstyle\sum_{j<n}c_{ij}a_j+c_{in}(\sum_{j<n}c_ja_j+d+v)+b_i\\&=\textstyle\sum_{j<n}(c_{ij}+c_jc_{in})a_j+(c_{in}(d+v)+b_i)\\&=A'_i+b'_i\end{aligned}$$

wobei A'_i in der k- und damit auch $k(v)$-linearen Hülle von $a_0,\ldots,a_{n-1}$ liegen und $b'_i\in k(v)(x)$ gilt. Wegen

$$A'_i=A_i+c_{in}(\sum_{j<n}c_ja_j-a_n)$$

kann andererseits die Dimension der neuen linearen Hülle $Lin(A'_1,\ldots,A'_n)$, die im übrigen von einer Grundkörpererweiterung unabhängig ist, um höchstens 1 von der alten abweichen. Da diese andererseits nach Induktionsvoraussetzung höchstens r ist, folgt die Induktionsbehauptung.

Den verbleibenden Fall $g_1=u/v,\ v\in k(x)^*$ behandelt man durch eine analoge Zurückführung auf ein Berechnungsproblem über $k(u/v)\to k(a,x)$.

Aufgabe: Führen Sie diesen Teil des Beweises aus.

□

4.4 Ableitungen

Als letztes Prinzip zur Herleitung unterer Schranken wollen wir noch eine Beziehung herleiten, die zeigt, daß man aus einem optimalen Programm für die Berechnung einer rationalen Funktion f mit wenig zusätzlichem Aufwand ein solches herleiten kann, das auch die partiellen Ableitungen von f zu berechnen vermag.

Sei dazu wieder $R:=k[X]:=k[x_1,\ldots,x_n]$ der Polynomring in n Variablen und $k(X):=k(x_1,\ldots,x_n)$ der zugehörige Ring rationaler Funktionen. Weiterhin wollen wir kurz ∂_i für die partielle Ableitung $\frac{\partial}{\partial x_i}$ schreiben. Eine Subalgebra $A \subset k(X)$ heißt *differentiell abgeschlossen*, wenn

$$f \in A \Longrightarrow \forall i\ \partial_i f \in A$$

gilt. Offensichtlich sind alle Lokalisierungen $k[X]_S$ differentiell abgeschlossen.

Satz 17 *(Baur, Strassen 1983) Sei $A = R_S$ eine Lokalisierung von $R := k[X]$ und $f \in A$. Dann gilt*

$$L^*_A(f, \partial_1 f, \ldots, \partial_n f) \leq 3L^*_A(f)$$

und

$$L^{tot}_A(f, \partial_1 f, \ldots, \partial_n f) \leq 5L^{tot}_A(f)$$

BEWEIS : Sei $\Gamma = (\Gamma_1, \ldots, \Gamma_t)$ ein optimales Programm zur Berechnung von f, das aus der Eingabe $(A; x_1, \ldots, x_n)$ die Ausgabesequenz $(x_1, \ldots, x_n, r_1, \ldots, r_t)$ mit $r_i(X) \in A$ berechnet. Wir wollen daraus mit Induktion ein Programm $\partial\Gamma$ konstruieren, das auch die Ableitungen von $f(X)$ berechnen kann.

Für $t = 0$ ist dies trivial, da nur $f = x_i$ sein kann, also $\partial_i f \in k$ als Element des Grundkörpers kostenfrei zur Verfügung steht.

Für $t > 0$ betrachten wir das Programm $\Gamma' = (\Gamma_2, \ldots, \Gamma_t)$, das aus Γ entstanden ist, indem die erste Instruktion entfernt und jede nachfolgende Referenz auf deren Ergebnis durch Referenz auf eine zusätzliche Eingabevariable x_0 ersetzt worden ist. Die Substitution $x_0 \mapsto r_1(X)$ induziert einen Algebra-Homomorphismus

$$\sigma :\ R' := k[x_0, \ldots, x_n] \longrightarrow A,$$

so daß Γ' auf der Lokalisierung $A' := R'_{S'}$ mit $S' = \sigma^{-1}(S)$ und der Eingabe $(x_0, \ldots, x_n)$ ausführbar ist. Dieses SLP berechnet eine Funktion $F(x_0, \ldots, x_n) \in A'$, so daß $f = F(r_1, x_1, \ldots, x_n)$ und

$$c_*(\Gamma') = c_*(\Gamma) - c_*(\Gamma_1)$$

gilt.

Nach Voraussetzung gibt es ein SLP $\partial\Gamma'$ mit

$$c_*(\partial\Gamma') \leq 3c_*(\Gamma'),$$

das $(F, \partial_0 F, \ldots, \partial_n F)$ berechnet. Stellen wir diesem SLP die Instruktion Γ_1 voran, so daß überall statt x_0 der Wert $r_1 = \sigma(x_0)$ eingesetzt wird, so berechnet das so modifizierte Programm $(\sigma(F), \sigma(\partial_0 F), \ldots, \sigma(\partial_n F))$ mit

$$f(x_1, \ldots, x_n) = \sigma(F) = F(r_1(X), x_1, \ldots, x_n).$$

Nach der Kettenregel gilt aber

$$\begin{aligned} \partial_i f &= \tfrac{\partial}{\partial x_0} F(r_1(X), x_1, \ldots, x_n) \cdot \tfrac{\partial}{\partial x_i} r_1(X) + \tfrac{\partial}{\partial x_i} F(r_1(X), x_1, \ldots, x_n) \\ &= \sigma(\partial_i F) + \sigma(\partial_0 F) \cdot \partial_i r_1. \end{aligned}$$

Da r_1 als Ergebnis von Γ_1 nur zwei der Inputvariablen x_a, x_b verbindet, ist $\partial_i r_1 = 0$ für $i \neq a, b$ und somit $\partial_i f$ bereits berechnet.

Ist $r_1 = x_a \cdot x_b$, so gilt $\partial_a r_1 = x_b$, also

$$\partial_a f = \sigma(\partial_a F) + \sigma(\partial_0 F) \cdot x_b.$$

Zu dessen Berechnung ist eine weitere Multiplikation notwendig. Analoges gilt für $\partial_b f$. Im SLP $\partial\Gamma$ sind also neben Γ_1 zwei weitere zusätzliche Multiplikationen auszuführen, so daß wir

$$c_*(\partial\Gamma) = c_*(\partial\Gamma') + 3$$

erhalten, woraus die Behauptung folgt.

Ist $r_1 = x_a / x_b$, so gilt

$$\partial_a r_1 = \frac{1}{x_b} \quad \text{und} \quad \partial_b r_1 = -\frac{x_a}{x_b^2}$$

Damit ergibt sich $\partial_a f = \sigma(\partial_a F) + u$ und $\partial_b f = \sigma(\partial_b F) - u \cdot r_1$ mit $u = \sigma(\partial_0 F)/x_b$. Wir benötigen also auch in diesem Fall drei zusätzliche Multiplikationen/Divisionen.

Analog behandelt man die anderen Möglichkeiten für Γ_1 sowie die Aussage über die totale Komplexität. $\square$

Betrachten wir eine Anwendung dieses Prinzips: Sei $f = x_1^m + \ldots + x_n^m$. Ein trivialer Algorithmus auf der Basis des binären Potenzierens berechnet f mit $n \log_2(m)$ Multiplikationen. Wegen

$$(\partial_1 f, \ldots, \partial_n f) = (x_1^{m-1}, \ldots, x_n^{m-1})$$

erhalten wir $L^*(f) \geq \frac{1}{3} n \log(m-1)$, so daß jeder Algorithmus zur Berechnung von f wenigstens $O(n \log(m))$ Multiplikationen benötigt.

5 Serielle und parallele Komplexität

5.1 Parallele Algorithmen

Die Miniaturisierung mikroelektronischer Bauteile stoßt langsam an physikalische Grenzen, zu deren Überwindung grundlegend neue Prinzipien zur Wirkung gebracht werden müßten. Andererseits beherrscht man den technologischen Prozeß ihrer Herstellung immer perfekter, was dazu führt, daß klassische Bauelemente immer billiger und in immer größeren Stückzahlen gefertigt werden können. Damit tritt die Frage des parallelen Einsatzes mehrerer Prozessoren/Prozesse und damit die Ablösung sequentieller von-Neumann-Rechner durch parallele Architekturen stärker ins Blickfeld praktischer Anwendungen.

Dieser Paradigmenwechsel wird in den letzten Jahren von der stärkeren Zuwendung zur Untersuchung der Parallelisierbarkeit algebraischer Algorithmen begleitet. Die dabei zu berücksichtigenden Probleme sind jedoch vielschichtiger als im sequentiellen Fall. Es gibt keine einheitliche Rechnerarchitektur mehr, sondern verschiedene Speichermodelle und Kommunikationstopologien spielen eine Rolle.

Das Modell der SLP ist besonders für die Untersuchung synchron getakteter Prozesse mit gemeinsamem Speicher geeignet. Bei unseren weiteren Betrachtungen wollen wir deshalb von einem solchen Parallelitätsmodell ausgehen. Wir wollen weiterhin davon ausgehen, daß die Kommunikations- und Synchronisationskosten gegenüber den Kosten für die arithmetischen Berechnungen gering sind. Des weiteren sollen Probleme, die aus Lese-Schreib-Kollisionen herrühren, unberücksichtigt bleiben.

Betrachten wir zunächst wieder als Beispiel die Berechnung von $\sum_{i=1}^{n} a_i$. Der offensichtliche sequentielle Algorithmus benötigt zur Berechnung dieser Summe $n-1$ Additionen.

Haben wir genügend Prozessoren zur Verfügung, so können wir in $t(n) = log_2(n)$ parallelen Schritten jeweils die Summe zweier benachbarter Summanden bilden, wofür wir im ersten Schritt $\frac{n}{2}$, im zweiten Schritt $\frac{n}{4}$ usw. Prozessoren benötigen. Obwohl nur am Anfang ausgelastet, brauchen wir für dieses *Fan-In*-Verfahren also $p(n) = \frac{n}{2}$ Prozessoren. Das Produkt $w = t \cdot p$ aus diesen beiden Größen nennen wir das *Potential* des entsprechenden Algorithmus. Es ist stets größer gleich der Zahl T der Knoten des Berechnungsgraphen, was ja ein Maß für die serielle Komplexität ist. In unserem Fall weichen beide Größen nur um einen logarithmischen Faktor voneinander ab. Einen solchen Algorithmus bezeichnet man als *prozessor-effizient.*

Ist die Anzahl der Prozessoren p beschränkt und $p \ll n$, so kann jeder der Prozessoren zwei Summanden aus dem Pool herausnehmen und durch deren Summe ersetzen, solange noch $\geq 2p$ Summanden vorhanden sind. Danach kann mit obigem Verfahren fortgesetzt werden, was insgesamt $t_p = \frac{n}{p} - 2 + log_2(p)$ parallele Schritte erfordert. Vergrößern wir die Zahl der Prozessoren um einen Faktor s, so verkürzt sich die parallele Ausführungszeit um einen Faktor $\frac{t_p}{t_{sp}} \in O(s)$. Dies ist eine Version von *Brents Scheduling-Prinzip*: Ein einzelner Prozessor kann s Prozessoren in der Zeit $O(s)$ simulieren.

Im allgemeinen Fall kann man bei beschränkter Prozessorzahl und einem prozessor-effizient parallelisierbaren seriellen Algorithmus ebenfalls nach Brents Prinzip vorgehen und die bei unbeschränkter Parallelität in einem Takt parallel abarbeitbaren (und damit voneinander unabhängigen) Schritte in mehreren Zeitintervallen auf die verfügbaren Prozessoren verteilen. Wir sehen deshalb, daß es selbst für realistische Anwendungen interessant ist, die unbeschränkte Parallelisierbarkeit von Algorithmen zu untersuchen.

Stehen genügend Prozessoren zur Verfügung, so ist die Frage nach dem Grad der Parallelisierbarkeit eine nach dem Verhältnis von Tiefe $D(\Gamma)$ und Größe $L(\Gamma)$ des Berechnungsgraphen des entsprechenden SLP Γ. Ist insbesondere der Berechnungsgraph ein (binärer) *Baum* mit Wurzel, d.h. werden alle Eingabegrößen zur Berechnung der einzigen Ausgabegröße wirklich verwendet, so gilt trivialerweise $D(\Gamma) \geq log_2(L(\Gamma))$. Ein SLP, für das $D(\Gamma) \in O(log_2(L(\Gamma)))$ gilt, nennen wir deshalb *effizient parallelisierbar.* Obiges Fan-In-Verfahren zur Summenberechnung ist ein solches.

5.2 Komplexität ausgewählter Algorithmen

Sehen wir uns als nächstes die Berechnung von x^n in $k[x]$ an. Wir hatten gesehen, daß mit Hilfe des binären Potenzierens diese Aufgabe seriell in $log_2(n)$ Schritten ausgeführt werden kann.

Dies ist aus Gradgründen auch parallel nicht effektiver möglich: Bezeichnen wir für eine rationale Funktion $f(x) = \frac{p(x)}{q(x)} \in k(x)$ mit $gcd(p,q) = 1$ als deren *Grad* $deg(f) := max(deg(p), deg(q))$, so gilt offensichtlich für $\circ \in \{+,-,*,/\}$

$$deg(f_1 \circ f_2) \leq deg(f_1) + deg(f_2)$$

und damit der folgende Satz:

Satz 18 *(Kung 1974) Die Berechnung von $f(x)$ benötigt mindestens $log(deg(f))$ parallele Schritte, d.h. $D^{tot}(f \mid k \cup \{x\}) \geq log(deg(f))$.*

Dieser Algorithmus ist also nicht nur nicht effizient, sondern in einem intuitiven Sinne überhaupt nicht parallelisierbar.

Interessanterweise gilt diese Schranke nicht für die nichtskalare Komplexität, denn wegen

$$\frac{n}{x^n - 1} = \sum_{j=0}^{n-1} \frac{\omega^j}{x - \omega^j}$$

für eine primitive n-te Einheitswurzel $\omega \in \mathbf{C}$ gibt es einen parallelen Algorithmus, der zur Berechnung von x^n nur in zwei Schritten Multiplikationen/Divisionen verwendet.

Im Gegensatz zum seriellen Fall kann man allerdings in $O(log_2(n))$ parallelen Schritten nicht nur x^n, sondern alle x^k, $k = 1, \ldots, n$ berechnen: Jede solche Potenz x^k läßt sich als Produkt geeigneter Potenzen x^{2^i} berechnen. Wir verwenden $n + 1$ Prozessoren, wobei einer durch fortgesetztes Quadrieren die Potenzen x^{2^i} berechnet und jeder der anderen Prozessoren die entsprechenden Produkte akkumuliert.

Dasselbe gilt sogar für iterierte Produkte $p_k = x_1 \cdot \ldots \cdot x_k$, $k = 1, \ldots, n$ mit verschiedenen Faktoren:

`Iterierte-Produkte(x)`

Input: $x = (x_1, \ldots, x_n)$
Output: $p_k = \prod_{i=1}^{k} x_i$, $k = 1, \ldots, n$

`Wenn` $n = 1$ `return` $(p_1 := x_1)$
`Berechne` $y_i = x_{2i-1} \cdot x_{2i}$, $i = 1, \ldots, [n/2]$ `parallel`
`Berechne rekursiv` p'`:=Iterierte-Produkte(y)`
`Berechne` $p_1(x) = x_1$, $p_{2k}(x) = p'_k$, $p_{2k+1}(x) = p'_k \cdot x_{2k+1}$ `parallel.`

Ist die Anzahl der parallelen Schritte dieses Algorithmus $t(n)$, so gilt die Rekursionsbeziehung $t(n) = t(n/2) + 2$, also $t(n) = 2\,log(n)$, wobei insgesamt $p(n) = n/2$ Prozessoren benötigt werden. Da das zugehörige SLP Γ allein n Eingabeknoten hat, gilt $L(\Gamma) \geq n$, d.h. Γ ist effizient parallelisierbar.

Dieses Laufzeitverhalten zeigen viele der klassischen Verfahren des symbolischen Rechnens, ohne daß man auf die trickreichen Modifikationen zur Senkung des seriellen Aufwands zurückgreifen muß. Betrachten wir etwa das klassische Verfahren zur Multiplikation zweier n-reihiger Matrizen. Es hat eine serielle Laufzeit $O(n^3)$ und kann parallel wie folgt organisiert werden: Berechne in einem ersten parallelen Takt die n^3 verschiedenen Produkte der Koeffizienten und stelle daraus in weiteren $log(n)$ parallelen Additionsschritten die Koeffizienten der Produktmatrix her. Das Verfahren ist also effizient parallelisierbar.

Dasselbe gilt für die Multiplikation zweier univariater Polynome vom Grad $< n$. Der Berechnungsbaum des klassischen Verfahrens hat $O(n^2)$ Knoten und eine Tiefe von $t(n) = log(n)$. Das Verfahren ist also ebenfalls effizient parallelisierbar.

Betrachten wir die Multiplikation auf der Basis der diskreten Fourier-Transformation: Der Kern der Berechnung war eine Transformation der Multiplikation in der Polynomalgebra in eine in der Algebra der Diagonalmatrizen der Größe N, wobei N die kleinste Zweierpotenz $\geq 2n$ ist. Die Multiplikation in der Algebra der Diagonalmatrizen kann in einem parallelen bzw. N seriellen Schritten ausgeführt werden. Die Transformation erfolgt über eine rekursiv implementierbare Matrix-Vektor-Multiplikation unter Verwendung der Beziehung ($M = N/2$)

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

$$= \begin{pmatrix} DFT_M(\omega^2)\, g_a + \Delta_M\, DFT_M(\omega^2)\, u_a \\ DFT_M(\omega^2)\, g_a - \Delta_M\, DFT_M(\omega^2)\, u_a \end{pmatrix}$$

Hier können wir rekursiv die beiden Multiplikationen $DFT_M\, v$ parallel ausführen und haben im Anschluß in zwei weiteren parallelen Schritten die entsprechenden Vektoroperationen (Multiplikation mit der Skalarmatrix Δ_M und Addition) komponentenweise auszuführen. Für die parallele Laufzeit erhalten wir $t(N) = t(M) + 2$, also $t(n) \in O(log(n))$. Die Anzahl der notwendigen Prozessoren ergibt sich ebenfalls rekursiv zu $p(N) = 2p(M)$, also $p(n) \in O(n)$.

Wir benötigen also dieselbe parallele Zeit wie beim klassischen Verfahren, kommen aber mit einer geringeren Zahl von Prozessoren aus.

Betrachten wir abschließend die Polynomevaluation. Wir hatten gesehen, daß im seriellen Fall hierfür das Hornerschema, das mit $O(n)$ Operationen auskommt, besonders optimal ist. Die in ihm bestehende Datenabhängigkeit erlaubt keine unmittelbare Parallelisierung. Andererseits hatten wir oben gesehen, daß sich in einem parallelen Modell mit n Prozessoren alle Potenzen x^k, $k = 0, \ldots, n$ in $log(n)$ parallelen Schritten berechnen lassen. Ein abschließendes Fan-In-Verfahren benötigt weitere $log(n)$ parallele Schritte, so daß das naive Verfahren mit $p(n) = n$ Prozessoren und $t(n) \in O(log(n))$ parallelen Schritten auskommt.

Aufgabe: Das Hornerschema kann man als Folge von Berechnungen $s_k = x \cdot s_{k-1} + a_k$, $k = 1, \ldots, n$ formulieren. Zeigen Sie, daß allgemein eine rekursiv gegebene Folge $\{x_k\}$ mit $x_k = a_k \cdot x_{k-1} + b_k$, $k = 1, \ldots, n$ mit vorgegebenen Folgen $\{a_k\}$ und $\{b_k\}$ in $O(log(n))$ parallelen Schritten berechnet werden kann.

5.3 Schnelle parallele Matrix-Algorithmen

Sei $A \in Mat_n(K)$ eine quadratische Matrix mit Einträgen aus einem Körper K. Der Gauß-Algorithmus spielt eine zentrale Rolle für eine Reihe von Matrixalgorithmen wie das Lösen eines linearen Gleichungssystems mit Koeffizientenmatrix A, die Berechnung von $det(A)$ oder die Berechnung der Inversen A^{-1}. Der Gauß-Algorithmus modifiziert dabei im Schritt i, $i = 1, \ldots, n$ die Einträge a_{kl}, $i + 1 \le k, l \le n$ einer $(n - i)$-reihigen Matrix jeweils zu $a_{kl} - a_{il}a_{kj}/a_{ii}$. Dafür sind insgesamt $\sum_{i=1}^{n} 3(n-i)^2 \in O(n^3)$ Operationen notwendig. Die Berechnung der verschiedenen Elemente eines Schritts kann man parallel ausführen, jedoch werden alle Ergebnisse des Schritts i benötigt, um Schritt $i + 1$ auszuführen. Eine offensichtliche Parallelisierung ist also nur bis $t(n) = n$, $p(n) = n^2$ möglich.

Im folgenden werden wir sehen, daß auch diese Matrixberechnungen in besserer als paralleler linearer Zeit möglich sind. Dies wurde zuerst 1976 von Csanky [8] für Körper der Charakteristik 0 beschrieben und 1984 von Berkowitz [2] auf beliebige kommutative Ringe mit 1 verallgemeinert.

Im folgenden sei deshalb K ein Körper der Charakteristik 0 und $A \in Mat_n(K)$ eine quadratische n-reihige Matrix mit Einträgen aus K. Deren *charakteristisches Polynom*

$$c_A(x) := det(x \cdot I - A) = x^n + c_1(A)x^{n-1} + \ldots + c_n(A)$$

zu berechnen bedeutet, dessen Koeffizienten $\{c_0(A) := 1, c_1(A), \ldots, c_n(A)\}$ zu finden. Über dem algebraischen Abschluß von K läßt sich A durch Basiswechsel in die Jordansche Normalform $B = C^{-1}\,A\,C$ transformieren. Dies ist eine Dreiecksmatrix, auf deren Diagonale die Eigenwerte $\lambda_1, \ldots, \lambda_n$ der Matrix A stehen. Wegen

$$c_A(x) = c_B(x) = \prod_{i=1}^{n} (x - \lambda_i)$$

gilt deshalb $c_i(A) = (-1)^i\, e_i(\lambda_1, \ldots, \lambda_n)$, wobei $e_i(x_1, \ldots, x_n)$ für die i-te elementarsymmetrische Summe von $x_1, \ldots, x_n$ steht. Diese hängen eng mit den Potenzsummen

$$s_k(x_1, \ldots, x_n) := \textstyle\sum_i x_i^k$$

zusammen.

Lemma 3 *Für $k = 1, \ldots, n$ gilt*

$$k \cdot c_k(x) + \sum_{r=1}^{k} s_r(x)\, c_{k-r}(x) = 0$$

oder in Matrixschreibweise

$$\begin{pmatrix} 1 & & & & & \\ s_1 & 2 & & & & \\ s_2 & s_1 & 3 & & & \\ s_3 & s_2 & s_1 & 4 & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ s_{n-1} & s_{n-2} & s_{n-3} & s_{n-4} & \cdots & n \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \vdots \\ c_n \end{pmatrix} = - \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \vdots \\ s_n \end{pmatrix}$$

In kompakter Form können wir dies als $S\mathbf{c} = -\mathbf{s}$ schreiben, wobei S obige untere Dreiecksmatrix ist.

Beweis : Für den Beweis dieser Beziehung führen wir die folgenden erzeugenden Funktionen ein:

$$S(t) := \sum_{r=1}^{\infty} s_r(x) t^{r-1} = \sum_i \sum_r x_i^r t^{r-1} = \sum_{i=1}^{n} \frac{x_i}{1 - x_i\, t}$$

$$C(t) := \sum_{i=0}^{n} c_i(x) t^i = \prod_{i=1}^{n} (1 - x_i\, t)$$

Wegen

$$\log\, C(t) = \sum_{i=1}^{n} \log(1 - x_i\, t)$$

und

$$\frac{C'(t)}{C(t)} = \frac{d}{dt} \log C(t) = \sum_{i=1}^{n} \frac{-x_i}{1 - x_i t}$$

gilt

$$S(t) \cdot C(t) + C'(t) = 0,$$

woraus sich die angegebenen Beziehungen durch Koeffizientenvergleich unmittelbar ergeben. □

Zur effektiven Berechnung des charakteristischen Polynoms beachte man nun, daß sich $s_i(A) := s_i(\lambda_1, \ldots, \lambda_n) = tr(B^i) = tr(A^i)$ als Spur einer Potenz von A ergibt. Wir haben also zunächst die Teilaufgabe

MatPowers(A): Gegeben ist $A \in Mat_n(K)$. Berechne A^i, $i = 1, \ldots, n$.

zu lösen. Dies kann ähnlich dem binären Potenzieren durch sukzessives Berechnen der Potenzen $A^{2^i}, 2^i \leq n$ und paralleles Aufmultiplizieren der gesuchten Ergebnisse in $O(log(n))$ parallelen Schritten von jeweils $\leq n$ Matrixmultiplikationen geschehen. Da jede einzelne Matrixmultiplikation ihrerseits $O(log(n))$ parallele Schritte erfordert, erhalten wir einen Gesamtalgorithmus mit $t(n) \in O(log(n)^2)$ und $p(n) \in O(n^4)$.

Satz 19 *Die iterierten Matrixprodukte A^i, $i = 1, \ldots, n$ von $A \in Mat_n(K)$ lassen sich in $O(log(n)^2)$ parallelen Schritten mit $O(n^4)$ Prozessoren berechnen.*

Daraus ergeben sich die $s_i(A), i = 1, \ldots n$ in weiteren $O(log(n))$ parallelen Schritten mit n^2 Prozessoren. Die gesuchten Koeffizienten $c_i(A), i = 1, \ldots, n$ erhalten wir schließlich aus der Beziehung $\mathbf{c} = -S^{-1}\mathbf{s}$ in $O(log(n))$ parallelen Schritten mit n^2 Prozessoren, wenn wir vorher S^{-1} effektiv berechnet haben. S ist aber eine untere Dreiecksmatrix. Diese kann rekursiv über den Ansatz

$$S = \begin{pmatrix} A & 0 \\ B & C \end{pmatrix} \quad \Longrightarrow \quad S^{-1} = \begin{pmatrix} A^{-1} & 0 \\ C^{-1} B A^{-1} & C^{-1} \end{pmatrix}$$

in paralleler Zeit $O(log(n)^2)$ mit $O(n^3)$ Prozessoren invertiert werden. Hierbei sind A und C wieder untere Dreiecksmatrizen mit $n/2$ Reihen und für die parallele Laufzeit gilt die Rekursionsformel $t(n) = t(n/2) + 2\,m(n/2)$, wobei $m(n) \in O(log(n))$ die parallele Laufzeit für die Multiplikation n-reihiger Matrizen ist.

Insgesamt gilt also

Satz 20 *Ist $A \in Mat_n(K)$ eine quadratische Matrix mit Einträgen aus einem Körper der Charakteristik 0, so lassen sich die Koeffizienten des charakteristischen Polynoms $c_A(x)$ von A in $O(log(n)^2)$ parallelen Schritten mit $O(n^4)$ Prozessoren berechnen.*

Aus $det(A) = (-1)^n\, c_n(A) = (-1)^n\, c_A(0)$ ergibt sich daraus mit demselben Aufwand die Determinante von A und wegen $c_A(A) = 0$ und damit

$$A^{-1} = -\frac{A^{n-1} + c_1(A)A^{n-2} + \ldots + c_{n-1}(A)}{c_n(A)}$$

auch deren Inverse, wenn $det(A) \neq 0$.

Wir haben damit den folgenden Satz bewiesen

Satz 21 *(Csanky 1976) Für eine quadratische Matrix $A \in Mat_n(K)$ mit Einträgen aus einem Körper der Charakteristik 0 lassen sich $MatPowers(A)$, $det(A)$, A^{-1} und die Koeffizienten des charakteristischen Polynoms $c_A(x)$ in $O(log(n)^2)$ parallelen Schritten mit $O(n^4)$ Prozessoren berechnen.*

5.4 Die Komplexitätsklassen NC

Im klassichen Fall spielt die asymptotische Untersuchung des Laufzeitverhaltens von Algorithmen eine wichtige Rolle. Dazu ist es zuerst einmal notwendig, daß der Algorithmus "skalierbar" ist, d.h. eine von einem Parameter $n \in \mathbf{N}$ abhängende Klassen von "gleichartigen" Berechnungsaufgaben löst.

Definition: Als *Berechnungsaufgabe* bezeichnen wir dabei eine Serie $\{P_n^{g_n,h_n}, n \in \mathbf{N}\}$, wobei zu P_n und jedem zulässigen Grundbereich A eine Abbildung

$$P_n(A) : A^{g_n} \longrightarrow A^{h_n}$$

gehört, deren Funktionswerte sich auf arithmetischem Wege aus den jeweiligen Argumenten ergeben. Wir sagen, daß eine Serie Γ_n von SLPs die Berechnungsaufgabe *löst*, wenn Γ_n auf $(A; a_1, \ldots, a_{g_n})$ angewendet die Folge $P_n(A)(a_1, \ldots, a_{g_n})$ berechnet. Wir nennen die Serie Γ_n *uniform*, wenn es einen Algorithmus gibt, der in der Lage ist, diese SLP's zu erzeugen.

Für die Untersuchung paralleler Laufzeiten können wir uns auf den Fall $h_n = 1$ beschränken, da wir für $h_n > 1$ die Einträge des Ergebnisvektors unabhängig voneinander berechnen können. Statt auf gemeinsame Zwischenergebnisse zurückzugreifen, berechnen wir dazu diese Zwischenergebnisse mehrfach, was die parallele Laufzeit nicht verändert und die notwendige Prozessorzahl um einen Faktor $\leq h_n$ erhöht.

Im seriellen Fall spielt die Klasse der polynomialen Algorithmen eine wichtige Rolle. Das sind solche, für die $L(\Gamma_n) \in n^{O(1)}$ gilt, d.h. die ihr Ergebnis in polynomialer serieller Zeit berechnen. Ein solches Verfahren ist effektiv parallelisierbar, wenn $D(\Gamma_n) \in O(log(L(\Gamma_n))) = O(log(n))$ ist. Umgekehrt hat eine Serie von SLP mit logarithmisch wachsender Tiefe und zusammenhängendem Berechnungsbaum eine polynomial beschränkte Prozessorzahl, da $L(\Gamma) \leq 2^{D(\Gamma)}$ gilt.

Wir hatten allerdings im letzten Abschnitt gesehen, daß sich für viele in polynomialer Zeit seriell lösbare Probleme nur SLPs mit polylogarithmischer Tiefe angeben lassen, d.h. solche, wo $D(\Gamma_n) \in O(log(n)^k)$ für geeignetes $k > 1$ gilt. Für solche SLPs folgt aus obiger Schranke nicht einmal die polynomiale Beschränktheit der Prozessorzahl. Ein seriell in polynomialer Laufzeit berechenbares Problem, das parallel in polylogarithmischer Zeit, aber mit einer nichtpolynomialen Zahl von Prozessoren berechenbar ist, wäre dann zwar aus Laufzeitsicht gut, aber nicht prozessor-effizient parallelisiert.

Auf Nicholas Pippenger (1979) geht deshalb die folgende Definition der Komplexitätsklassen NC (Nick's Class) zurück:

Definition: NC^k ist die Menge aller Berechnungsprobleme, für die es eine uniforme Familie von SLP's Γ_n gibt mit $L(\Gamma_n) \in n^{O(1)}$ und $D(\Gamma_n) \in O(log(n)^k)$, d.h. die ihr Ergebnis mit einer polynomialen Anzahl von Prozessoren in paralleler Zeit $O(log(n)^k)$ berechnen.

Die Vereinigung dieser Mengen $NC := \bigcup_k NC^k$ sind dann genau die Berechnungsprobleme, die in polylogarithmischer Zeit mit polynomialer Prozessorzahl parallel berechenbar sind.

Bei der Untersuchung der sequentiellen Komplexität spielt das Konzept der Reduktion eines Problems auf ein anderes in polynomialer Zeit eine wichtige Rolle. Unter diesen Reduktionen sind die Klasse P der polynomialen Algorithmen und auch die Klasse NP abgeschlossen und es lassen sich NP-vollständige Probleme definieren, auf die alle anderen Probleme aus NP polynomial reduzierbar sind.

Ein ähnliches Konzept erhält man, wenn man innerhalb der Klasse NC die Reduktion durch NC^1-Algorithmen betrachtet.

DEFINITION: Ein Berechnungsproblem $\{R_n\}$ heißt NC^1-reduzierbar auf das Problem $\{S_n\}$, wenn es eine uniforme Serie von SLPs Γ_n in NC^1 gibt, die R_n berechnen und die neben gewöhnlichen Knoten auch *Orakelknoten* besitzen, in denen aus einer Eingabesequenz $(A; y_1, \ldots, y_r)$ eine Ausgabesequenz $(z_1, \ldots, z_s) := S_n(A)(y_1, \ldots, y_r)$ berechnet wird. Ein solcher Knoten ist mit dem Gewicht $log(r+s)$ bei der Berechnung der Tiefe zu berücksichtigen.

Wir schreiben in diesem Fall $R \leq S$.

Bemerkung: Wie für die polynomiale Reduzierbarkeit im seriellen Fall folgt aus $R \leq S$ und $S \in NC^k$, $k \geq 1$ auch $R \in NC^k$. Wir müssen dazu nur in dem Algorithmus, der die NC^1-Reduzierbarkeit beweist, alle Orakelknoten durch entsprechende Aufrufe des effizienten Verfahrens für S ersetzen.

Betrachten wir die Matrixberechnungsprobleme aus dem vorigen Abschnitt, also (für eine k-Algebra A)

ItMatProd(A) - die Berechnung des Produkts von n Matrizen $M_1, \ldots, M_n \in Mat_n(A)$,

MatPowers(A) - die Berechnung der Potenzen M^i, $i = 1, \ldots, n$ einer Matrix $M \in Mat_n(A)$,

CharPoly(A) - die Berechnung der Koeffizienten des charakteristischen Polynoms einer Matrix $M \in Mat_n(A)$,

Det(A) - die Berechnung der Determinante einer Matrix $M \in Gl_n(A)$ und

MatInv(A) - die Berechnung der Inversen einer Matrix $M \in Gl_n(A)$.

Wir hatten im vorigen Abschnitt gesehen, daß alle diese Probleme in NC^2 liegen, wenn $A = K$ ein Körper der Charakteristik Null ist. Es ist ein offenes Problem, ob es effizientere parallele Algorithmen für eines dieser Probleme gibt. Dies würde nach sich ziehen, daß alle anderen Probleme auch zu der besseren Komplexitätsklasse gehören würden, weil folgender Satz gilt:

Satz 22 *Über einem Körper K sind die Probleme* **ItMatProd***(K),* **MatPowers***(K),* **CharPoly***(K),* **Det***(K) und* **MatInv***(K) NC^1-äquivalent.*

Manche dieser Aussagen über die NC^1-Reduzierbarkeit lassen sich einfach aus allgemeinen Überlegungen für Matrizen über beliebigen Algebren A herleiten. So sind etwa **ItMatProd**(A) und **MatPowers**(A) NC^1-äquivalent: Für $M_1, \ldots, M_n \in Mat_n(A)$ bilden wir die Blockmatrix

$$N = \begin{pmatrix} 0 & M_1 & 0 & \dots & 0 \\ 0 & 0 & M_2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & M_n \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \in Mat_{n(n+1)}(A)$$

Dann hat N^n nur einen nichtverschwindenden Block in der rechten oberen Ecke mit dem Eintrag $M_1 \cdot \ldots \cdot M_n$. Das Transformationsprogramm $MatPowers \Rightarrow ItMatProd$ besteht also aus einem einzigen Orakelknoten zur Berechnung von N^n, der mit $log((n^2+n)^2) \in O(log(n))$ zur Tiefe beiträgt.

Die schwierigste Implikation ist $MatPowers \Rightarrow CharPoly$, welche auf Arbeiten von Berkowitz (1984) und Chistov (1985) zurückgehen.

AUFGABE: Zeigen Sie $ItMatProd(A) \leq MatInv(A) \leq Det(A) \leq Charpoly(A)$.

Hinweise:

$Charpoly \Rightarrow Det$: Verwende $det(M) = c_M(0)$.

$Det \Rightarrow MatInv$: Verwende die Adjunktenformel.

$MatInv \Rightarrow ItMatProd$: Zeige, daß die Matrix $I - N \in Mat_{n(n+1)}(A)$ invertierbar ist und untersuche den Zusammenhang zum iterierten Matrixprodukt.

5.5 Strassens Satz von der Vermeidung von Divisionen

Im einleitenden Beispiel zur Berechnung von x^n hatten wir gesehen, daß die Berechnung von x^{31} allein durch Multiplikationen mindestens 7 Operationen benötigt, unter Verwendung von Divisionen jedoch mit 6 Operationen auskommt. In diesem Abschnitt wollen wir untersuchen, wie weit Berechnungen in $k(X)$ und $k[X]$ voneinander entfernt sind. Genauer gesagt werden wir zeigen, daß man mit überschaubarem Aufwandszuwachs jeden Divisionsschritt durch Multiplikationen ersetzen kann.

Betrachten wir dazu rationale Funktionen $r(X) \in k(X)$, die im Ursprung einen wohldefinierten Wert haben. Solche rationalen Funktionen können in der Form $r = \frac{f}{g}$ mit $g(0) \neq 0$ dargestellt werden, wobei $f, g \in k[X]$ geeignete Polynome sind und g einen nichtverschwindenden Absolutterm hat. Durch weitergehende Normierung können wir r eindeutig in der Form $r = \frac{p(x)}{1-q(x)}$ darstellen, wobei Zähler und Nenner teilerfremd sind und $q(0) = 0$ gilt.

Betrachten wir die Menge

$$\mathcal{O} := \{\frac{p(x)}{q(x)} \,|\, p, q \in k[X], q(0) \neq 0\}.$$

Sie bildet einen Unterring von $k(X)$, den *Ring der rationalen Potenzreihen*. Der Name rührt daher, daß man jedes Element $f \in \mathcal{O}$ eindeutig in eine Potenzreihe in einer Umgebung des Ursprungs zerlegen kann, die die Gestalt

$$f = \sum_{i=0}^{\infty} f^{(i)}$$

mit homogenen Polynomen $f^{(i)} \in k[X]$ vom Grad i hat. So ist etwa

$$\frac{1}{1-(x+y^2)} = 1 + (x+y^2) + (x+y^2)^2 + (x+y^2)^3 + \ldots = \\ 1 + x + (x^2+y^2) + (x^3+2xy^2) + (y^4+3x^2y^2) + \ldots$$

Die entsprechende Zerlegung läßt sich etwa über die multivariate Taylorreihenentwicklung

$$f = \sum_{\alpha} D^\alpha f(0) X^\alpha \quad \text{mit} \quad D^\alpha := \frac{1}{\alpha_1! \ldots \alpha_n!} \frac{\partial^{\alpha_1 + \ldots + \alpha_n}}{\partial x_1^{\alpha_1} \ldots \partial x_n^{\alpha_n}}$$

gewinnen und heißt deshalb die *Taylorreihenentwicklung* von $f \in \mathcal{O}$. Die Menge der Einheiten $\mathcal{O}^*$ besteht genau aus den rationalen Funktionen $f \in \mathcal{O}$ mit $f(0) \neq 0$.

Für $d \in \mathbf{N}$ und $f \in \mathcal{O}$ sei $C_d(f) := \{f^{(0)}, \ldots, f^{(d)}\}$ und für eine endliche Teilmenge $F \subset \mathcal{O}$ sei $C_d(F) = \cup_{f \in F} C_d(f)$. Wir erhalten auf diese Weise eine Menge von homogenen Polynomen in $k[X]$, die in enger Beziehung zu den rationalen Funktionen $f \in F$ stehen und wollen untersuchen, wie $L_{\mathcal{O}}(F)$ und $L_{k[X]}(C_d(F))$ zusammenhängen.

Die Grundidee ist dabei, daß man nicht nur Multiplikationen, sondern auch Divisionen in $\mathcal{O}$ allein durch *Multiplikationen* der homogenen Bestandteile der entsprechenden Potenzreihen ersetzen kann. Ist etwa $f = \sum_{i=0}^\infty f^{(i)}$, $g = \sum_{i=0}^\infty g^{(i)}$, so gilt

$$fg = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} f^{(j)} g^{(i-j)} \right).$$

Für den Quotienten $\frac{f}{g}$ mit $g^{(0)} = 1$ bekommt man eine etwas kompliziertere Darstellung, da

$$\frac{1}{1-h} = 1 + \sum_{i=1}^{\infty} h^{(i)}$$

gilt und $h^{(i)}$ nur homogene Bestandteile vom Grad $\geq i$ besitzt, wenn $h^{(0)} = 0$ ist.

Satz 23 *Sei $F \subset \mathcal{O}$ eine endliche Menge rationaler Potenzreihen und $d \in \mathbf{N}$. Dann gilt*

$$L^*_{k[X]}(C_d(F)) \leq \binom{d}{2} L^*_{\mathcal{O}}(F)$$

und

$$L^{tot}_{k[X]}(C_d(F)) \leq (d+1)^2 L^{tot}_{\mathcal{O}}(F),$$

wobei sich die Tiefe des optimalen Berechnungsgraphen Γ_1 über $k[X]$ von der des optimalen Berechnungsgraphen Γ_2 über $\mathcal{O}$ nur um einen Faktor $\leq \log(d)$ unterscheidet.

Beweis : Wir beweisen nur die erste der beiden Aussagen. Die zweite und dritte ergeben sich analog.

Sei $l = L^*_{\mathcal{O}}(F)$ und $(u_1, \ldots, u_l)$ eine optimale Berechnungssequenz für F in $(\mathcal{O}; X)$, wobei jeder Schritt die Gestalt $u_i = a_i \cdot b_i$ oder $u_i = a_i / b_i$ hat mit geeigneten Linearkombinationen a_i, b_i der vorausgehenden Ergebnisse $u_j, j < i$. Da sich die homogenen Bestandteile einer Linearkombination rationaler Funktionen allein durch skalare Operationen auf den homogenen Bestandteilen der eingehenden Summanden berechnen lassen, bleibt zu zeigen, daß man jede der beiden Operationen durch höchstens $\binom{d}{2}$ nichtskalare Operationen ersetzen kann, um die zugehörigen homogenen Bestandteile $C_d(u_i)$ aus $C_d(a_i)$ und $C_d(b_i)$ zu berechnen:

1. Für $f, g \in \mathcal{O}$ gilt

$$L^*_{k[X]}(C_d(fg) \mid C_d(f,g)) \leq \binom{d}{2}.$$

2. Ist außerdem $g^{(0)} \neq 0$, so gilt auch

$$L^*_{k[X]}(C_d(f/g) \mid C_d(f,g)) \leq \binom{d}{2}.$$

Für $u = fg$ gilt

$$\begin{aligned} u^{(0)} &= f^{(0)} g^{(0)}, \\ u^{(1)} &= f^{(0)} g^{(1)} + f^{(1)} g^{(0)}, \\ u^{(2)} &= f^{(0)} g^{(2)} + f^{(1)} g^{(1)} + f^{(2)} g^{(0)}, \ldots \end{aligned}$$

Beachten wir noch, daß wegen $f^{(0)}, g^{(0)} \in k$ die entsprechenden Multiplikationen skalar sind, erhalten wir insgesamt $1 + 2 + 3 + \ldots + (d-1) = \binom{d}{2}$ nichtskalare Multiplikationen für die homogenen Bestandteile.

Für $u = f/g$ gilt $f = ug$ und durch Inversion obiger Formeln erhalten wir

$$\begin{aligned} u^{(0)} &= f^{(0)}/g^{(0)}, \\ u^{(1)} &= \left(f^{(1)} - u^{(0)} g^{(1)}\right)/g^{(0)}, \\ u^{(2)} &= \left(f^{(2)} - u^{(1)} g^{(1)} - u^{(0)} g^{(2)}\right)/g^{(0)}, \ldots, \end{aligned}$$

was wiederum eine Berechnungssequenz für $C_d(u)$ mit $\binom{d}{2}$ nichtskalaren Multiplikationen ergibt. □

In diesem Satz haben wir das klassische Verfahren für die Berechnung der homogenen Komponenten verwendet. Durch ausgefeiltere Methoden kann man

$$L^*_{k[X]}(C_d(F)) \leq 5.75\, d\, L^*_{\mathcal{O}}(F)$$

zeigen.

Auf der Basis obigen Satzes können wir die folgende Aussage über die beschränkte Nützlichkeit von Divisionen bei der Berechnung polynomialer Ausdrücke formulieren:

Satz 24 *(Strassen 1973) Sei $F \subset k[X]$ eine endliche Menge von Polynomen vom Grad $\leq d$ über einem unendlichen Körper. Dann gilt*

$$L^*_{k[X]}(F) \leq \binom{d}{2} L^*_{k(X)}(F).$$

Beweis : Sei $(u_1, \ldots, u_l)$ eine optimale Berechnungssequenz von F über $k(X)$. Da k unendlich ist, finden wir $\lambda = (\lambda_1, \ldots, \lambda_n) \in k^n$, so daß $g(\lambda) \neq 0$ für alle auftretenden Nenner $g(X) \in k[X]$. Führen wir die Substitution $y_i = x_i - \lambda_i$ aus, so erhalten wir eine Berechnungssequenz über $\mathcal{O}(Y)$, die wir mit unserem Satz mit einer solchen über $k[Y] = k[X]$ vergleichen können. □

Wir erhalten insbesondere die Aussage, daß Divisionen bei der Evaluierung quadratischer Polynome nicht weiterhelfen:

Korollar 4 *(Strassen 1973) Sei $F \subset k[X]$ eine endliche Menge von Polynomen vom Grad ≤ 2 über einem unendlichen Körper. Dann gilt*

$$L^*_{k[X]}(F) = L^*_{k(X)}(F).$$

5.6 Der Parallelisierungssatz für Parsebäume

Wir hatten oben in verschiedenen Beispielen gesehen, daß sich für die meisten Polynome prozessor-effizient parallelisierbare Evaluierungsschemata angeben lassen. Wir wollen dies nun systematischer untersuchen. Wir wollen uns dabei auf die Signatur $\{+, -, *\}$ beschränken.

Betrachten wir zunächst einen allgemeinen arithmetischen Ausdruck, der n Terme und damit $n - 1$ arithmetische Operationen in beliebiger Klammerung enthält. Einem solchen Ausdruck entspricht ein Parsebaum, mit dem wir ihn identifizieren wollen. Auf den "Kopf" gestellt entspricht dieser Parsebaum seinerseits dem Berechnungsgraphen eines SLP, in dem (wegen der Baumeigenschaft) jedes Zwischenergebnis im weiteren nur einmal verwendet wird. Jeden Berechnungsgraphen können wir in einen solchen Baum transformieren, wenn wir einfach die Ergebnisse, die im folgenden mehrfach verwendet werden, mehrfach berechnen. Die parallele Laufzeit eines solchen modifizierten Programms bleibt unverändert, seine Größe ändert sich allerdings.

Wir wollen nun ein Verfahren angeben, das es erlaubt, diesen Parsebaum so umzuformen, daß er denselben Ausdruck in paralleler Zeit $4\,log(n)$ berechnet. OBdA wollen wir annehmen, daß jede Variable in den Ausdruck nur einmal eingeht. Dies ist keine wesentliche Einschränkung, da etwa ein parallel-effektives Verfahren zur Berechnung von $E = (a_1 \cdot a_2 + b_1)/(b_2 \cdot b_3)$ leicht auf die Berechnung von $(a^2 + b)/b^2$ angewendet werden kann.

Betrachten wir zunächst ein Beispiel, an dem wir das Vorgehen verdeutlichen wollen. Sei $E = (((ax_1 + b)x_2 + c)x_3 + d)x_4 + e$. Der zugehörige Berechnungsbaum hat die Tiefe 8. Setzen wir $u := (ax_1 + b)x_2 + c$ und reorganisieren rekursiv den Teilbaum $E' = (ux_3 + d)x_4 + e$ der Tiefe 4 als $E'_{neu} = u(x_3x_4) + (dx_4 + e)$ der Tiefe 3, in dem die Variable u erst spät in die Rechnungen einbezogen wird. Analog können wir u als $u_{neu} = a(x_1x_2) + (bx_2 + c)$ reorganisieren. Fügen wir die reorganisierten Berechnungsbäume zusammen, erhalten wir einen solchen für den Ausdruck E, der nur noch die Tiefe 5 hat.

Satz 25 *(Brent 1974) Ein Parsebaum, der $q > 2$ Operationen $\{+, -, *\}$ enthält, kann in einen äquivalenten Parsebaum der Tiefe $\leq 4\,log_2(q)$ transformiert werden.*

Beweis : (nach [4, Theorem 6.3.2.]) Für einen Parsebaum E mit paarweise verschiedenen Variablen als Blätter bezeichne $|E|$ die Anzahl seiner Blätter (damit ist $|E| - 1$ die Anzahl der arithmetischen Operationen). Wir setzen $t(E) := max(3, 4\,log_2(|E| - 1))$ und zeigen mit Induktion nach $|E|$ die Gültigkeit der folgenden Aussagen:

> (1) Jeder solche Parsebaum E läßt sich darstellen als $E = A * B + C$ mit Parsebäumen A, B, C der Tiefe $\leq t(E) - 2$. Insbesondere folgt daraus
>
> (1a) Jeder Parsebaum E läßt sich zu einem Parsebaum der Tiefe $\leq t(E)$ reorganisieren.

(2) Für jede in E vorkommende Variable y kann man E umformen zu $E = A * y + B$ mit Parsebäumen A, B der Tiefe $\leq t(E) + 2$, in denen y nicht mehr vorkommt.

Man überzeugt sich leicht, daß die beiden Aussagen für $|E| \leq 4$ richtig sind. (Beispiel $n = 4$: Parsebaum kann 2-2-Baum oder 3-1-Baum sein; man spiele alle Fälle durch)

Sei nun $|E| = n$ und nehmen wir an, daß die beiden Aussagen für Parsebäume mit weniger als n Variablen gelten. Wir suchen zuerst in E einen Knoten x, so daß für den Teilbaum $E(x) = L \circ R$ mit Wurzel x die Beziehungen $|E(x)| \geq \frac{n+1}{2}$ und $|L|, |R| < \frac{n}{2}$ gelten.

Sei E' der Parsebaum, der aus E entsteht, indem man den Teilbaum $E(x)$ durch eine neue Variable u ersetzt. Aus $|E'| = |E| - |E(x)| + 1 \leq \frac{n+1}{2}$ folgt $t(E') \leq t(E) - 4$ und nach (2) finden wir Parsebäume A_1, B_1 der Tiefe $\leq t(E) - 2$ mit $E' = A_1 * u + B_1$. Wegen $|L|, |R| < \frac{n}{2}$ findet man nach (1a) Parsebäume für diese Ausdrücke der Tiefe $\leq t(E) - 4$, also einen für $E(x)$ der Tiefe $\leq t(E) - 3$. Dann ist aber $E = A_1 * E(x) + B_1$ eine Darstellung (1) von E.

Für die zweite Aussage zerlegen wir den Parsebaum E mit Rücksicht auf die Lage von y: Wir wählen den Knoten z so, daß $y \in E(z) = L \circ R$, $|E(z)| \geq \frac{n+1}{2}$ und für den Teilbaum von $E(z)$, der y enthält (dies sei oBdA L), $|L| < \frac{n}{2}$ gilt. Über den rechten Teilbaum können wir dann nur $|R| \leq n - 1$ aussagen. Den entsprechenden Teilausdruck E' können wir mit (2) wieder als $E' = A_2 * u + B_2$ mit Parsebäumen A_2, B_2 der Tiefe $\leq t(E) - 2$ darstellen. Dasselbe gilt für den linken Teilbaum $L = A_3 * y + B_3$. Schließlich kann nach (1) der rechte Teilbaum R in $\leq t(E)$ parallelen Schritten berechnet werden. Für $\circ =' +'$ erhalten wir daraus

$$E = A_2 * (L + R) + B_2 = (A_2 * A_3)y + (A_2 * B_3 + B_2 + A_2 * R),$$

dessen beiden y-Koeffizienten jeweils in $\leq t(E) + 2$ parallelen Schritten berechnet werden können ($A_2 * B_3 + B_2$ kann in den Schritten $t(E) - 1$ und $t(E)$ berechnet werden!). Für $\circ =' *'$ erhalten wir analog

$$E = A_2 * (L * R) + B_2 = (A_2 * A_3 * R)y + ((A_2 * B_3) * R + B_2).$$

□

AUFGABE: ([4, Theorem 6.3.2.]) Zeigen Sie, daß dasselbe Ergebnis auch für die Operationenmenge $\{+, -, *, /\}$ gilt.

Hinweis: Zeigen Sie

(1) Jeden Parsebaum E kann man umformen zu $E = F/G$, wobei die Parsebäume F und G unabhängig voneinander in paralleleler Zeit $t(F), t(G) \leq t(E) - 2$ evaluiert werden können.

(2) Für jede in E vorkommende Variable y kann man E umformen zu $E = (Ay + B)/(Cy + D)$ mit Parsebäumen A, B, C, D, in denen y nicht mehr vorkommt und die unabhängig voneinander in paralleleler Zeit $\leq t(E)$ evaluiert werden können.

Der Satz von Brent zeigt, daß jede Berechnung, die auf einen Parse*baum* führt, der seriell in polynomialer Laufzeit abgearbeitet werden kann, in der Klasse NC^1 liegt. Wir hatten gesehen, daß man andererseits jeden Berechnungsgraphen eines SLP durch Mehrfachberechnung von mehrfach verwendeten Teilergebnissen in einen Baum derselben *Tiefe* überführen kann. Da die *Größe* des Berechnungsgraphen dabei jedoch zunimmt, kann die vordem polynomiale Prozessorzahl nach diesen Dopplungen exponentiell werden. Das Ergebnis taugt also nicht für eine generelle Einordnung von arithmetischen Berechnungsproblemen in eine der Klassen NC^k.

5.7 Der Parallelisierungssatz für Polynome

Um so überraschender ist das folgende Resultat, das sich auf allgemeine Berechnungsgraphen für multivariate Polynome bezieht.

Satz 26 *(Valiant, Skyum, Berkowitz, Rackoff 1981)*
Jedes sequentielle Programm, das ein multivariates Polynom $f \in k[X]$ vom Grad d in C Schritten berechnet, kann in ein paralleles Programm der Tiefe $O(log(d)\,(log(C) + log(d)))$ und der Größe $O(C^3 d^6)$ transformiert werden.

Beweis : Wir können das serielle Programm f als Folge von Berechnungen $v_i := v_i' \circ v_i''$ mit $v_i', v_i'' \in k \cup \{x_1, \ldots, x_n\} \cup \{v_1, \ldots, v_{i-1}\}$ und $\circ \in \{+, -, *\}$ vorliegend annehmen. $f(v_i)$ bezeichne das im i-ten Knoten berechnete Polynom, $d(v_i)$ dessen Grad, wobei wir $d(v_i') \geq d(v_i'')$ annehmen wollen.

Wir bezeichnen ein Programm f als *homogen*, wenn $d(v_i') = d(v_i'')$ für alle additiven Programmschritte gilt. Die folgende Tatsache zeigt, daß wir f als homogen annehmen können:

> Wenn ein nicht homogenes Programm ein Polynom p vom Grad d in C Schritten berechnet, so gibt es ein homogenes Programm der Größe $O(Cd^2)$, das die $d+1$ homogenen Komponenten von p berechnet.

Dazu ersetze man jede Zeile des Ausgangsprogramms durch mehrere Zeilen, die aus den jeweiligen homogenen Bestandteilen der $f(v_i')$ und $f(v_i'')$ die homogenen Bestandteile von $f(v_i)$ berechnen. Für $\circ \in \{+, -\}$ geschieht das komponentenweise, für $\circ = *$ durch Berechnen der $(d+1)^2/2$ relevanten Produkte der Koeffizienten und nachfolgendes Aufsummieren zu $d + 1$ homogenen Komponenten von p. Im ersten Fall wird eine Zeile also durch $d + 1$ Zeilen, im zweiten Fall durch $d(d + 1)$ Zeilen ersetzt.

Für ein solches homogenes Programm f der Größe C können wir oBdA $f(v_i) \neq 0$ und $d(v_i) \leq d(v_C)$ annehmen. Ihm entspricht ein Berechnungsbaum mit der Knotenmenge $V = k \cup X \cup \{v_1, \ldots, v_C\}$. Für $v \in V$ wollen wir die Länge des längsten in v endenden Pfads als dessen *Tiefe* bezeichnen.

Definieren wir nun für $v, w \in V$ ein Polynom $f(v, w) \in k[X]$ mit Induktion nach der Tiefe von w, das den "Anteil" symbolisiert, der im Zweig von v nach w berechnet wird:

function f(v,w)
  if $v = w$ **then** $f(v, w) = 1$
  else if $w \in k \cup X$ **then** $f(v, w) = 0$
  else if $w = w' \pm w''$ **then** $f(v, w) = f(v, w') \pm f(v, w'')$
  else if $w = w' * w''$ **then** $f(v, w) = f(w'') * f(v, w')$

Ist f ein homogenes Programm, so sind alle $f(v)$ und $f(v,w)$ homogen; für letztere beweist man mit Induktion, daß sie den Grad $d(v,w) := d(w) - d(v)$ haben.

Definieren wir nun für $a > 0$ den *Schnitt*

$$V_a := \{t \in V \mid d(t) > a, t = t' * t'', d(t'), d(t'') \leq a\}.$$

Für jeden solchen Knoten t gilt nach Konstruktion $d(t) = d(t') + d(t'')$.

Sei $d(v) \leq a < d(w)$. Dann gilt

$$f(v,w) = \sum_{t \in V_a} f(v,t) \cdot f(t,w) \quad \text{und} \quad f(w) = \sum_{t \in V_a} f(t) \cdot f(t,w).$$

AUFGABE: Beweisen Sie die Aussage durch Induktion nach der Tiefe von w bei fixiertem v und a, indem Sie die Fälle

(1) $w = w' + w''$,
(2) $w = w' \cdot w'', a \geq d(w')$, d.h. $w \in V_a$ und
(3) $w = w' \cdot w'', a < d(w') \leq d(w)$

unterscheiden.
Wir wollen nun aus einem homogenen Programm f der Größe C, das ein Polynom p vom Grad d berechnet, ein Programm f' der Größe $O(C^3)$ herstellen, das ebenfalls p berechnet, in dem aber jeder Knoten eine Tiefe von höchstens $O(\log(C)\log(d)$ hat. Wir wollen f' in $\log(d)$ Schritten konstruieren, wobei jeder Schritt die Tiefe des Programms um höchstens $2\log(C)$ vergrößert.

Im Schritt 0 berechnen wir alle $f(w)$, $f(v,w)$ vom Grad $d(w), d(v,w) \leq 2^0 = 1$. Da dies Linearformen in $X = (x_1, \ldots, x_n)$ sind und $C \geq n$ gilt, benötigt man für deren Berechnung mit einem Fan-In-Verfahren höchstens $2 + \log(C)$ parallele Schritte.

Im Schritt $i+1$ berechnen wir alle $f(w)$, $f(v,w)$ vom Grad $2^i < d(w), d(v,w) \leq 2^{i+1}$. Dies geschieht in zwei Schritten.

Zunächst berechnen wir alle geforderten $f(w)$. Dazu betrachten wir den Schnitt zu $a = 2^i$

$$f(w) = \sum_{t \in V_a} f(t) \cdot f(t,w) = \sum_{t \in V_a} f(t') \cdot f(t'') \cdot f(t,w).$$

Nach Voraussetzung ist $d(t'), d(t''), d(t,w) = d(w) - d(t) \leq a = 2^i$, die entsprechenden Knoten also schon berechnet. $f(w)$ kann also in $\leq \log(C)$ parallelen Schritten berechnet werden.

Nun berechnen wir alle geforderten $f(v,w)$. Dazu betrachten wir den Schnitt zu $a = d(v) + 2^i$

$$f(v,w) = \sum_{t \in V_a} f(v,t) \cdot f(t,w) = \sum_{t \in V_a} f(v,t') \cdot f(t'') \cdot f(t,w).$$

Jedes der $f(v,t'), f(t,w)$ ist wiederum bereits berechnet. Ist $d(t'') \leq 2^{i+1}$, so steht es aus der ersten Phase dieses Schritts bereits zur Verfügung. Ist $f(v,t') = 0$, so brauchen wir es nicht. Ist $f(v,t') \neq 0$, also $d(t') \geq d(v)$ und $d(t'') > 2^{i+1}$, so folgt

$$d(t) = d(t') + d(t'') > d(v) + 2^{i+1} \geq d(w)$$

wegen $d(v,w) = d(w) - d(v) \leq 2^{i+1}$. Folglich ist dann $f(t,w) = 0$. Wir kommen also wieder mit $\log(C)$ parallelen Schritten aus.

Die Größe des neuen Programms wird durch die Berechnung der $\leq C^2$ Funktionen $f(v,w)$ dominiert. Da jedesmal eine Summe von maximal C Summanden zu bilden ist, erhalten wir als Gesamtgröße einen Wert der Ordnung $O(C^3)$.

Damit haben wir den Satz bewiesen: Ein inhomogenes Programm der Größe C wird in ein homogenes der Ordnung Cd^2 und dieses schließlich in eines der Größe $O((Cd^2)^3)$ und der Tiefe $O(\log(Cd^2)\log(d)) = O((\log(C) + \log(d))\log(d))$ transformiert. □

Korollar 5 *Jedes Berechnungsproblem, das auf die Berechnung von Polynomen $p_n(X)$ vom Grad n zurückgeführt werden kann, und wofür ein serieller polynomialer Algorithmus existiert, liegt in der Klasse NC^2.*

In der Tat, in diesem Fall ist $d = n$, $C = n^{O(1)}$.

Literatur

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman: *The Design and Analysis of Computer Algorithms.* Addison Wesley, Reading, Mass. 1976.

[2] S.J. Berkowitz: On computing the determinant in small parallel time using a small number of processors. *Inf. Proc. Letters* **18** (1984), 147 - 150.

[3] T. Beth: Das Verfahren der schnellen Fourier-Transformation. BI Mannheim, 1984

[4] A. Borodin, I. Munro: *The computational complexity of algebraic and numeric problems.* American Elsevier, New York 1975.

[5] A. Borodin, J. von zur Gathen, J. Hopcroft: Fast parallel matrix and gcd computations. *Information and Control* **52** (1982), 241 - 256.

[6] D. Bini, V. Pan: *Polynomial and Matrix Computations.* Birkhäuser, Boston 1994.

[7] P. Bürgisser, M. Clausen, M.A. Shokrollahi: *Algebraic Complexity Theory.* Springer Verlag, 1996.

[8] L. Csanky: Fast parallel matrix inversion algorithms. *SIAM J. Comput.* **5** (1976), 618 - 623.

[9] J. von zur Gathen: Parallel algorithms for algebraic problems. *SIAM J. Comp.* **13** (1984), 802 - 824.

[10] J. von zur Gathen: Parallel algebraic computations. A survey. In: LNCS 233 (1986), 93 - 112

[11] D. E. Knuth: *The art of computer programming.* Addison Wesley, Reading, Mass. 1991.

[12] A.M. Ostrowski: On two problems in abstract algebra connected with Horner's rule. In *Studies in Mathematics and Mechanics presented to Richard von Mises*, Academic Press, 1954, 40–48.

[13] R. Reischuk: *Einführung in die Komplexitätstheorie.* B.G. Teubner, Stuttgart 1990.

[14] A. Scholz: *Jber. DMV* **47** (1937), 41 - 43.

[15] V. Strassen: Algebraic complexity theory. In : *Handbook of Theoretical Computer Science* (ed. J. van Leeuwen), ch. 11, 635 - 672, Elsevier, Amsterdam 1990.

[16] L.G. Valiant, S. Skyum, S. Berkowitz, C. Rackoff: Fast parallel computation of polynomials using few processors. *J. Comp.* **12** (1983), 641 - 644.