ABBILDUNGEN UND ALGORITHMEN

Reinhard Laue

Lehrstuhl II für Mathematik
Universität Bayreuth
Postfach 10 12 51
D-8580 Bayreuth

Zusammenfassung

Die Verwendung von Homomorphismen erlaubt in vielen Fällen, effektive Algorithmen zu entwerfen. Beispiele finden sich sowohl in der Algebra, insbesondere der Gruppentheorie, als auch in der Informatik. Es wird gezeigt, daß unter geeigneten Voraussetzungen Algorithmen mit logarithmischer Komplexität erhalten werden. Zur Illustration des Entwurfsprinzips werden mehrere Algorithmen skizziert.

§ 1 DAS HOMOMORPHIEPRINZIP

Die Theorie der Algorithmen hat die natürliche Aufgabenstellung, Prinzipien für den Entwurf effektiver Algorithmen herauszuarbeiten. Bekannte problemunabhängige Strategien sind etwa Divide-and-Conquer, Backtracking und Branch-and-Bound. Dabei ist die Komplexität der danach entworfenen Algorithmen häufig schwierig zu bestimmen. Hier soll das Homomorphieprinzip herausgestellt und eine allgemeine Abschätzung für die Komplexität hergeleitet werden.

1.1 Homomorphieprinzip:

1. Vereinfache eine gegebene Struktur durch eine Kette von Vergröberungsschritten.
2. Ein Problem in der gegebenen Struktur löse zuerst in der gröbsten Struktur und verfolge diese Lösung schrittweise zurück, bis das Ausgangsproblem gelöst ist.

Dies ist natürlich eine informelle Beschreibung, die für Komplexitätsabschätzungen präzisiert werden muß, siehe unten.

Zunächst sollen einige einfache Beispiele die Idee illustrieren und die Effektivität der Algorithmen zeigen.

1.2 Lösen von Kongruenzen.

Sei $X^2 \equiv -1 \bmod p^n$ zu lösen, wobei p eine Primzahl und $n \in \mathbb{N}$ ist. Ein naiver Algorithmus würde aus jeder Restklasse modulo p^n einen Repräsentanten daraufhin testen, ob er die Kongruenz löst. Der Aufwand würde im schlechtesten Fall etwa p^n Tests bedeuten.

Nach dem Homomorphieprinzip lösen wir die Aufgabe zuerst für $n = 1$, wobei p Tests erforderlich sein können. Ist -1 kein quadratischer Rest modulo p, so kann keine Lösung existieren.

Sei $x_1^2 \equiv -1 \bmod p$ gelöst. Dann durchlaufen wir die p verschiedenen Restklassen mod p^2, deren Repräsentanten modulo p zu x_1 kongruent sind, und prüfen, ob sie $x_2^2 \equiv -1 \bmod p^2$ lösen. Maximal sind wieder p Tests erforderlich. Allgemein wird x_{i+1} als Lösung von $x_{i+1}^2 \equiv -1 \bmod p^{i+1}$ dadurch erhalten, daß Repräsentanten der p Restklassen modulo p^{i+1} getestet werden, die modulo p^i zu x_i kongruent sind, für $i = 1,2,\ldots,n-1$.

Auf diese Weise wird eine Lösung in maximal $n \cdot p$ Tests gefunden. Es sei vermerkt, daß das Vorgehen nur das Prinzip erläutern soll und keinen Anspruch auf eine optimale Lösung dieses Problems erhebt.

1.3 Zugriffwege in Datenbanken

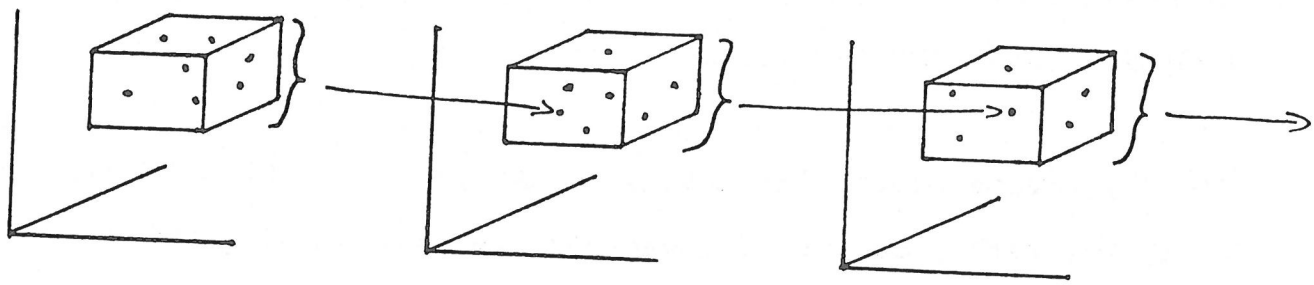
Es seien in einer Datei viele Datensätze mit Werte zu Attributen $A_1,\ldots,A_n$ zu speichern. Gesucht werden in der Datei solche Daten-

sätze, der Attributwerte gewissen anzugebenden Schranken genügen. Um solche Daten schnell zu finden, kann man wie folgt vorgehen [7].

Die Daten werden als Punkte im n-dimensionalen Raum $\mathbb{N}^n \subseteq \mathbb{R}^n$ aufgefaßt. Da nicht alle Daten zugleich im Arbeitsspeicher Platz finden, werden sie zu Blöcken zusammengefaßt, die unter jeweils einer Adresse auf externen Speichern abgelegt werden.

Um Datenpunkte zu solchen Blöcken zusammenzufassen, zerlegt man den Raum in Würfel, so daß die in einem Würfel enthaltenen Datenpunkte in einen Block passen. Damit wird der Würfel durch einen Punkt ersetzt, der als Inhalt die Adresse des Blocks im Speicher besitzt.

$n = 3$



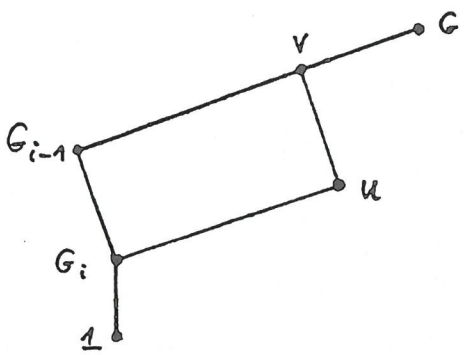
Damit hat man den ersten Vergröberungsschritt vorgenommen. Interpretiert man die erhaltene Punktmenge im $\mathbb{R}^n$ wieder als Datenmenge, so läßt sich das gleiche Vorgehen erneut praktizieren. Wir iterie-

ren, bis die verbleibende Punktmenge im Arbeitsspeicher gehalten werden kann. Beim Zugriff wird schrittweise entschieden, in welchen Würfel jeweils Daten bzw. Adressen von Daten liegen, die der Anfrage entsprechen.

1.4 Alle Untergruppen einer p-Gruppe.

Eine Gruppe von einer Primzahlpotenzordnung p^n, kurz p-Gruppe, besitzt eine Kette $G = G_0 > G_1 > G_2 > \ldots > G_n = 1$ von Normalteilern, so daß $|G_{i-1}/G_i| = p$ ist, für $i = 1,2,\ldots,n$. Um alle Untergruppen von G zu berechnen, bestimmen wir aus den Untergruppen, die G_{i-1} enthalten, alle Untergruppen, die G_i enthalten, für $i = 1,2,\ldots,n$. Dieses läßt sich besonders einfach durchführen, wenn wir minimale Erzeugendensysteme verwenden.

Sei $V = < G_{i-1}, v_1, \ldots, v_d >$, wobei die $v_i \in G$ sind und d die minimale Erzeugendenzahl von V/G_{i-1} ist. Ist $G_{i-1} = < G_i, x >$ mit $x \in G$, so ist jedenfalls $V = < G_i, x, v_1, \ldots, v_d >$.



Ist sogar $V = < G_i, v_1, \ldots, v_d >$, so liegt xG_i in $\phi(V/G_i)$, siehe [8] Seite 273. Dann enthält jede maximale Untergruppe von V mit G_i auch G_{i-1}.

Modulo G_i gibt es dann keine weiteren Untergruppen, deren homomorphes Bild modulo G_{i-1} gerade V/G_{i-1} ist.

Ist $U = \langle G_i, v_1, \ldots, v_d \rangle \neq V$, so ist erstens $\{x, v_1, \ldots, v_d\}$ ein minimales Erzeugendensystem von V modulo G_i und zweitens U eine der Untergruppen, die G_i enthalten und deren homomorphes Bild modulo G_{i-1} gerade V/G_{i-1} ist. Alle solchen Untergruppen sind genau die Untergruppen $\langle G_i, v_1 x^{a_1}, v_2 x^{a_2}, \ldots, v_d x^{a_d} \rangle$ mit $0 \leq a_i < p$.

Mit leichten Zusatzüberlegungen erhält man alle Normalteiler von G und unter Einsatz weiterer Theorie alle maximalen abelschen Untergruppen. Für eine ausführliche Beschreibung siehe [10].

§ 2 KOMPLEXITÄTEN

Um die Komplexitäten von Algorithmen abschätzen zu können, die auf dem Homomorphieprinzip beruhen, benötigen wir einige Präzisierungen unserer Beschreibung dieses Prinzips.

2.1 Definition.

Seien S_1 und S_2 Strukturen, d.h. Mengen mit darauf vereinbarten Familien von Relationen.

a) Eine Abbildung $f: S_1 \to S_2$ <u>vererbt</u> ein Problem P_1 in S_1 auf ein Problem P_2 in S_2, wenn das Bild jeder Lösung von P_1 unter f in einer Lösung von P_2 enthalten ist.

b) Eine Folge $(f_1,\ldots,f_n)$ von Abbildungen $f_i:S_i \to S_{i+1}$ <u>löst</u> ein Problem P_1 in S_1 <u>auf</u>, wenn gilt:

$$\exists_{P_2,\ldots,P_{n+1}} \forall_i \; f_i \text{ vererbt } P_i \text{ auf } P_{i+1}.$$

Damit können wir folgendes Ergebnis formulieren.

2.2 Satz.

$(f_1,\ldots,f_n)$ löse P_1 in S_1 auf, und zu $T \subseteq S_i$ finde ein Algorithmus A_i alle in T enthaltenen Lösungen von P_i in $a_i(|T|)$ Schritten, wobei a_i monoton ist. Dann ist P_1 in

$$\sum_{i=1}^{n} a_i(l_{i+1}\cdot|f_i^{-1}|)\cdot m_{i+1}$$

Schritten lösbar, wobei l_i die maximale Mächtigkeit der Lösungsmengen in S_i und $|f_i^{-1}|$ die maximale Mächtigkeit einer Urbildklasse von f_i ist.

Zum Beweis ist nur zu bemerken, daß in einem einzelnen Schritt zu jeder der m_{i+1} Lösungen von P_{i+1} das vollständige Urbild bei f_i als Eingabe für A_i dient. Da a_i monoton ist, ist der Aufwand jeweils höchstens $a_i(|f_i^{-1}|\cdot l_{i+1})$. ∎

Seine Effizienz bezieht dieses Vorgehen daraus, daß mit jeder als Lösung ausgeschlossenen Teilmenge von S_{i+1} große Teilmengen von S_i nicht mehr betrachtet zu werden brauchen. Dies wird etwa bei dem Beispiel 1.1 sofort deutlich. Allerdings haben wir dort und

und im allgemeinen bei rekursiven Algorithmen stets schärfere Voraussetzungen. Für diese läßt sich dann eine handlichere Aussage herleiten.

2.3 Satz.

$(f_1,\ldots,f_n)$ löse P_1 in S_1 auf, wobei jedes f_i surjektiv sei, sei $|S_{n+1}| = 1$. Jede Urbildklasse $f_i(y)^{-1}$ für jedes y und jedes i sei gleichmächtig, etwa von der Mächtigkeit k. Ein Algorithmus A benötigt $a(|T|)$ Schritte, a monoton, um die Lösungen von P_i in $T \subseteq S_i$ zu finden. Zudem habe jede Lösung höchstens Mächtigkeit l, und es sei m eine obere Schranke für die Zahl der Lösungen für jedes i. Dann ist P_1 in

$$a(k\cdot l)\cdot \log_k(|S_1|)\cdot m$$

Schritten lösbar.

Beweis.

Wir zeigen zuerst, daß $n = \log_k(|S_1|)$ ist. Für $n = 1$ ist $S_1 = f_1^{-1}(S_2)$ und $|S_2| = 1$, also $|S_1| = k$ und $1 = \log_k(k)$. Sei nun $n > 1$. Aus $|S_1| = k\cdot|S_2|$ folgt mit $n-1 = \log_k(|S_2|)$ dann $\log_k(|S_1|) = 1+\log_k(|S_2|) = 1+n-1 = n$.

Nun sehen wir, daß im i-ten Schritt aus den m_{i+1} Lösungen von P_{i+1} mittels A_i jeweils mit Aufwand $a(k\cdot l)$ die Lösungen von P_i berechnet werden. Der Aufwand ist also mit $n = \log_k(|S_1|)$

$$\sum_{i=1}^{n} m_{i+1} \cdot a(k \cdot l)$$

$$\leq a(k \cdot l) \cdot m \cdot \sum_{i=1}^{n} 1$$

$$= a(kl) \cdot m \cdot \log_k(|S_1|)$$

Da $a(k \cdot l)$ eine Konstante ist, ist unter den Voraussetzungen von 2.3 die Effizienz von A für das asyptotische Verhalten nicht wesentlich. Allerdings sind in der Praxis eher kleine Strukturen interessant, so daß $a(k \cdot l)$ groß ist gegenüber $\log_k(|S_1|)$. Von daher ist natürlich ein besonders effizientes A anzustreben.

In vielen Fällen bedeutet eine Auflösung, die sich nicht mehr verfeinern läßt, daß die zu betrachtenden Unterstrukturen sehr spezielle Gestalt haben. Dann kann Aweit effektiver gewählt werden, als in der allgemeinen Situation. Als Beispiel sei auf 1.4 verwiesen sowie auf den folgenden skizzierten Algorithmus.

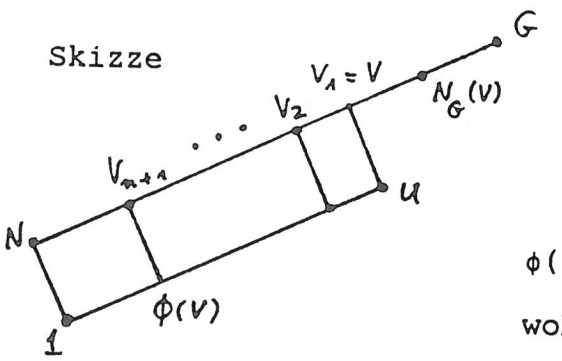
2.4 Konjugiertenklassen in p-Gruppen

Wir betrachten sowohl die Konjugiertenklassen von Elementen als auch die von Untergruppen. Jeweils können wir Repräsentantensysteme dadurch erhalten, daß wir eine p-Gruppe auf einer Menge von p Elementen operieren lassen. Da jede Bahn als Länge eine p-Potenz haben muß, sind dann entweder alle Elemente in einer Bahn oder es liegen keine zwei in einer Bahn.

Im Falle der Elemente steigt der Algorithmus von Felsch und Neubüser [5] wie in 1.4 eine Kette von Normalteilern G_i ab. Zu jedem Repräsentanten xG_{i+1} wird stets auch sein Satbilisator $N_G(xG_{i+1})$ berechnet. Die Bahnen von $N_G(xG_{i+1})$ auf xG_{i+1} mod G_i entsprechen dann den Bahnen von G auf G/G_i, die modulo G_{i+1} das Element x enthalten.

Im Falle der Untergruppen wird das Homomorphieprinzip auch zum Entwurf des Algorithmus A für den Einzelschritt erneut angewendet. Wir berechnen zu jedem Repräsentanten V wiederum seinen Stabilisator $N_G(V)$ mit. Um die Notation zu vereinfachen, schildern wir das Vorgehen, wenn Repräsentanten $U \neq V$ gesucht werden, die modulo einem minimalen Normalteiler N auf einen gegebenen Repräsentanten V abgebildet werden.

Skizze



Dazu verwenden wir eine Kette von Normalteilern von $N_G(V)$, der Form

$$\phi(V)=V_{n+2} < N\phi(V)=V_{n+1} < \ldots < V_1=V,$$

wobei $|V_i/V_{i+1}| = p$ ist für jedes i.

Die Auflösung durch Homomorphismen erhalten wir, wenn wir

$$U = U_1 \to U_2 = U_1 \cap V_2 \to \ldots \to V_{n+1} \cap U = \phi(V)$$

verwenden.

Zuerst werden die Bahnen von $N_G(V)$ auf der Menge der Komplemente von $V_{n+1}/\phi(V)$ in $V_n/\phi(V)$ bestimmt. Da es genau p solcher Komplemente gibt, haben wir einen der beiden obigen Fälle. Wir bestimmen Repräsentanten U_n und deren Stabilisatoren $N_G(U_n)$.

Im nächsten Schritt werden die Bahnen von $N_G(U_n)$ auf der Menge der Komplemente von V_n/U_n in V_{n-1}/U_n in der gleichen Weise behandelt.

Dies wird iteriert, bis schließlich alle gesuchten Bahnen von Komplementen von N in V_1 repräsentiert sind. Eine etwas andere Version dieses Algorithmus ist in [10] ausführlich beschrieben.

§ 3 ALGORITHMEN FÜR ABBILDUNGEN

Wurde das Homomorphieprinzip bisher angewandt, um mittels Abbildungen Algorithmen für unterschiedliche Probleme zu erläutern, so spezialisieren wir uns nun auf Algorithmen für Abbildungen. Wir betrachten dabei eine Situation, die von der Pólya und de Bruijn-schen Abzähltheorie her wohlbekannt ist.

Eine Gruppe A operiere auf einer Struktur S_1, und eine Gruppe B auf einer Struktur S_2. Gesucht wird ein Repräsentantensystem für die Bahnen von A×B auf der Menge $\mathrm{Hom}(S_1,S_2)$, wobei $f^{(a,b)}(s) = f(s^{a^{-1}})^b$ für $s \in S_1$ die Operation beschreibt. Homomorphismen

kann man bei Strukturen unterschiedlich definieren. Wir verlangen, daß eine Abbildung $f:S_1 \to S_2$ die Grundmengen und die Relationen verträglich abbilden muß. Ist $S_1 = (\Omega_1, \mathfrak{R}_1)$, $S_2 = (\Omega_2, \mathfrak{R}_2)$ mit Mengen Ω_1, Ω_2 und Familien von Relationen $\mathfrak{R}_1, \mathfrak{R}_2$ so ist $f = (\mu, \nu)$ und $\mu:\Omega_1 \to \Omega_2$, $\nu:\mathfrak{R}_1 \to \mathfrak{R}_2$ so daß

(i) $\forall\, R \in \mathfrak{R}_1 \qquad R^{\nu}|_{\mu(\Omega_1)} = \{(\omega_1^{\mu},\ldots,\omega_n^{\mu}) \mid (\omega_1,\ldots,\omega_n) \in R\}$

(ii) $\{R^{\nu}|_{\mu(\Omega_1)} R \in \mathfrak{R}_1\} = \{R|_{\mu(\Omega_1)} \mid R \in \mathfrak{R}_2\}$ gilt.

Während bei der Abzähltheorie üblicherweise nur Mengen S_1 und S_2 betrachtet werden, wollen wir komplexere Strukturen zulassen. Dann lassen sich Isomorphieprobleme bei Objekten lösen, die mittels Homomorphismen aus höheren Strukturen konstruiert werden. Als Beispiel seien semidirekte Produkte von Gruppen erwähnt [9].

Um das Problem einer algorithmischen Behandlung zugänglich zu machen, transformieren wir es in ein gruppentheoretisches Problem. Das hat den zusätzlichen Vorteil, daß dieselben Algorithmen für verschiedene Typen von Strukturen verwendet werden können.

Eine solche Transformation wollen wir kurz demonstrieren:

Die Menge $\mathrm{Iso}(S_1,S_2)$ aller Isomorphismen einer Struktur S_1 auf eine Struktur S_2 erhalten wir, indem wir nach einem festen Isomorphismus φ alle Automorphismen γ von S_2 ausführen:

$$\mathrm{Iso}(S_1,S_2) = \{\varphi\gamma \mid \gamma \in \mathrm{Aut}\ S_2\}$$

Ist $(\alpha,\beta) \in A \times B \leq \text{Aut } S_1 \times \text{Aut } S_2$, so ist $\alpha^{-1} \gamma_1 \beta = \varphi \gamma_2$ für $\gamma_1, \gamma_2 \in \text{Aut } S_2$ genau dann, wenn $\varphi^{-1}\alpha^{-1}\varphi \gamma_1 \beta = \gamma_2$ ist, d.h. $\gamma_2 \in A^{\varphi}\gamma_1 B$, wobei $A^{\varphi}\gamma_1 B$ eine Doppelnebenklasse von A^{φ} und B in Aut S_2 ist.

Die Menge $A^{\varphi}\backslash \text{Aut } S_2 / B$ aller Doppelnebenklassen von A und B in Aut S_2 entspricht daher den Bahnen von $A \times B$ auf $\text{Iso}(S_1, S_2)$.

Diese Transformation ist ein Schritt bei einer groben Auflösung des Problems, Repräsentanten der Bahnen auf $\text{Hom}(S_1, S_2)$ zu finden.

Wir betrachten dabei die Schritte

f → (Kern f, Bild f) → (char. Klasse (Kern f), char. Klasse (Bild f))

wobei die charakteristische Klasse jeweils die Bahn unter der vollen Automorphismengruppe bezeichnet.

Bei der schrittweisen Lösung bestimmen wir zuerst die Bahnen von Aut S_2 auf der Menge der möglichen Bildstrukturen durch Angabe jeweils eines Repräsentanten U und seines Stabilisators $N_{\text{Aut } S_2}(U)$, sowie die Bahnen von Aut S_1 auf der Menge der möglichen Kernrelationen durch Angabe jeweils eines Repräsentanten K und seines Stabilisators $N_{\text{Aut } S_1}(K)$.

Die Bahnen von B auf der Menge der möglichen Bildstrukturen in einer charakteristischen Klasse mit Repräsentanten U entsprechen den Doppelnebenklassen

$$N_{\text{Aut } S_2}(U)\backslash \text{Aut } S_2/B,$$

da B durch Rechtsmultiplikation auf $N_{\text{Aut } S_2}(U)\backslash \text{Aut } S_2$ ähnlich operiert wie auf der charakteristischen Klasse von U. Entsprechend beschreiben die Doppelnebenklassen

$$N_{\text{Aut } S_1}(K)\backslash \text{Aut } S_1/A$$

die Bahnen von A auf der charakteristischen Klasse eines Repräsentanten K.

Es sind jeweils Repräsentanten und ihre Stabilisatoren $N_B(U)$ bzw. $N_A(K)$ zu bestimmen.

Zu Paaren von Repräsentanten (U,K) mit $U \cong S_1/K$ ist jeweils ein Isomorphismus $\varphi: S_1/K \to U$ zu bestimmen und die von $N_A(K)$ auf S_1/K induzierte Automorphismengruppe $\overline{N_A(K)}$ sowie die von $N_B(U)$ auf B induzierte Automorphismengruppe $\overline{N_B(U)}$. Dann entsprechen die Bahnen von $\overline{N_A(K)} \times \overline{N_B(U)}$ auf $\text{Iso}(S_1/K, U)$ nach der Vorüberlegung den Doppelnebenklassen

$$\overline{N_A(K)}^{\varphi}\backslash \text{Aut } U/\overline{N_B(U)}$$

in Aut U.

Durch Verkettung des natürlichen Homomorphismus $\nu: S_1 \to S_1/K$ mit den Repräsentanten $\varphi\gamma$ der obigen Doppelnebenklassen erhalten wir

dann alle Repräsentanten von Bahnen von A×B auf $Hom(S_1,S_2)$ bei denen die Kerne und Bilder der Homomorphismen in der gleichen Bahn wie K bzw. U liegen.

In jedem Schritt ist dabei das Problem zu lösen, Doppelnebenklassenrepräsentanten zu finden. Die Allgemeinheit der von uns gemachten Voraussetzungen läßt vermuten, daß dieses gruppentheoretische Problem nicht immer ganz leicht zu lösen ist. In der Tat sind bisher keine besonders effektiven Algorithmen bekannt, für verschiedene Ansätze siehe [4], [6], [2]. In [9] haben wir das Homomorphieprinzip benutzt, um dieses Problem zu lösen. Damit wird die obige grobe Auflösung des Problems, Repräsentanten in $Hom(S_1,S_2)$ zu finden, in jedem der 3 Schritte durch eine Auflösung des jeweiligen Problems, Doppelnebenklassenrepräsentanten zu finden, verfeinert.

Wir gehen bei dem Problem, ein Repräsentantensystem Rep(A\G/B) der Doppelnebenklassen zweier Untergruppen A und B einer Gruppe G zu finden, von einer Kette von Untergruppen $G = A_0 > A_1 > \ldots > A_n = A$ aus. (Diese ist in vielen praktischen Anwendungen leicht zu finden, jedoch könnten im allgemeinen sowohl A als auch B maximale Untergruppen von G sein!)

Die Auflösung wird dann durch die Abbildungen

$f_i : A_i\, gB \rightarrow A_{i-1}\, gB$ für $i = n, n-1, \ldots, 1$ gegeben.

Wir verwenden also, daß Doppelnebenklassen von A_{i-1} und B in G in Doppelnebenklassen von A_i und B in G zerfallen. Um im Einzelschritt aus $Rep(A_{i-1}\backslash G/B)$ durch einen Algorithmus $Rep(A_i\backslash G/B)$ zu erhalten, benutzen wir, daß wir die Doppelnebenklassen $A_{i-1}\backslash G/B$ als Bahnen von B auf der Menge der Rechtsrestklassen $A_{i-1}\backslash G$ von A_{i-1} in G bei Rechtsmultiplikation auffassen können. (An dieser Stelle könnte auch ein anderer Algorithmus verwendet werden!) Wir berechnen daher jeweils zu jedem Repräsentanten g den Stabilisator $S = Stab_B(A_{i-1}g)$ in B. Ist $A_{i-1}=A_ia_1U...UA_ia_n$, so entsprechen die Bahnen von S auf $\{A_ia_1g,...,A_ia_ng\}$ den Doppelnebenklassen $A_{i-1}\times B$, die bei f_i auf $A_{i-1}g$ B abgebildet werden. Wir berechnen also Repräsentanten und deren Stabilisatoren durch diesen Einzelschrittalgorithmus.

In [9] haben wir dazu ein Programm in der Sprache Cayley [3] angegeben und ein Beispiel demonstriert. Im ersten Schritt wurden dabei aus 8 Nebenklassen 3 Doppelnebenklassen gefunden. Im nächsten Schritt mußten Bahnen auf jeweils 6 Nebenklassen für jeden der 3 Repräsentanten bestimmt werden. Hier ergaben sich insgesamt 11 Repräsentanten. Im letzten Schritt wurden dann Bahnen auf jeweils 7 Nebenklassen zu jedem der 11 Repräsentanten berechnet. Damit wurden als Ergebnis Repräsentanten der 59 Doppelnebenklassen gefunden.

§ 4 Schlußbemerkung.

In den vorstehenden Paragraphen wurde das Homomorphieprinzip formuliert und zum Entwurf effektiver Algorithmen verwendet. Das Anliegen dieses Artikels ist es vorzuschlagen, dieses Prinzip gezielt zu nutzen, wenn Probleme algorithmisch gelöst werden sollen. Die skizzierten gruppentheoretischen Algorithmen sind bis auf eine Ausnahme [5], die als Auslöser anzusehen ist, auf diese Weise entstanden. Als Übung möge der Leser ihm bekannte effektive Algorithmen (z.B. die Gauß-Elimination) daraufhin untersuchen, in wieweit implizit das Homomorphieprinzip verwendet wurde.

Literatur

[1] M.D. Atkinson (Editor). Computational group theory, proceedings of the LMS Symposium on computational group theory. Academic Press, 1984.

[2] G. Butler. On computing double coset representatives in permutation groups. In [1], pp. 283-290.

[3] J. Cannon. The group language Cayley: programming manual (Draft). University of Sidney, 1980.

[4] J.H. Conway. An algorithm for double coset enumeration? In [1], pp. 33-37.

[5] V. Felsch, J. Neubüser. An algorithm for the computation of conjugacy classes and centralizers in p-groups. Springer Lecture Notes in Computer Science Vol. 72, pp. 452-465.

[6] R.H. Gilman. Enumerating infinitely many cosets. In [1], pp. 51-55.

[7] H. Hinterberger, J. Nievergelt, K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. ACM TODS 9 (1984), 38-71.

[8] B. Huppert. Endliche Gruppen I. Springer Berlin Heidelberg New York, S. 273.

[9] R. Laue. Computing double coset representatives for the generation of solvable groups. Springer Lecture Notes in Computer Science vol. 144, 65-70.

[10] R. Laue, J. Neubüser, V. Schoenwaelder. Algorithms for finite soluble groups and the SOGOS system. In [1], pp. 105-135.