Umfangreichere Programmieraufgabe:

Berechnung von Elementen der kristallinen Basis der Quantengruppe $U_q^+(sl_n)$

Dirk Dudek
BUGH Wuppertal
dudek@rz.uni-wuppertal.de

Jens Bender
BUGH Wuppertal
bender@rz.uni-wuppertal.de

März 1998 — April 1998

Zusammenfassung

Dieses Papier stellt eine Dokumentation der umfangreicheren Programmieraufgabe dar. Es handelt sich dabei um ein Programm, dessen Aufgabe es ist, Basiswechselkoeffizienten zwischen der kristallinen Basis und der PBW–Basis der Gruppe $U_q^+(sl_n)$ zu berechnen. Der zugrundeliegende Algorithmus wird von der Theorie der Quantengruppen geliefert.

Inhaltsverzeichnis

Die Darstellung des mathematischen Hintergrunds und des Algorithmus folgt weitgehend den Notizen von Markus Reineke.

1 Mathematischer Hintergrund

Ein zentrales Objekt in der Theorie der Quantengruppen ist die folgende Algebra, der sogenannte positive Teil der „DRINFELD–JIMBO–Quantisierung der universellen Einhüllenden der LIE–Algebra sl_n".

Definition 1.1:

> *$U_q^+(sl_n)$ bezeichne die $\mathbb{Q}(v)$–Algebra mit Erzeugenden E_i, $i = 1, \cdots, n$ und Relationen*
>
> $$E_i\,E_j = E_j\,E_i \qquad \textit{falls} \quad |i-j| \geq 2$$
>
> $$E_i^2\,E_j - (v+v^{-1})\,E_i\,E_j\,E_i + E_j\,E_i^2 = 0 \qquad \textit{falls} \quad |i-j| = 1.$$

Das heißt, $U_q^+(sl_n)$ ist die freie assoziative $\mathbb{Q}(v)$–Algebra in den Erzeugenden E_i modulo dem durch die Relationen erzeugten zwei-seitigen Ideal.

Eines der Ziele der Theorie der Quantengruppen ist es, die Struktur dieser Algebra möglichst genau zu verstehen.
Hierzu ist es sinnvoll, in einem gewissen Sinne „gute" Basen von $U_q^+(sl_n)$ als (unendlich-dimensionalen) $\mathbb{Q}(v)$–Vektorraum zu finden. Zwei solche Basen werden durch relativ tiefliegende Sätze von RINGEL beziehungsweise LUSZTIG geliefert.

1.1 Die PBW–Basis

RINGEL konstruiert wie folgt eine Algebra $H_q(Q)$ aus der Darstellungstheorie des Köchers

$$Q = A_n : \quad 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow \cdots \longrightarrow n\ .$$

Sei $H_q(Q)$ der $\mathbb{Q}(v)$–Vektorraum mit Basis

$$\{E_{[M]} \mid [M] \text{ Isomorphieklasse einer endlich–dimensionalen } Q\text{–Darstellung}\}$$

wobei der Grundkörper $\mathbb{k}$ beliebig sei. Definiere auf $H_q(Q)$ eine Multiplikation wie folgt:

$$E_{[M]} \cdot E_{[N]} := \sum_{[X]} v^{[M,M]+[N,N]+<\underline{\dim}\,M,\underline{\dim}\,N>-[X,X]} \cdot F_{MN}^X(v^2) \cdot E_{[X]}.$$

Dabei bezeichnet

$$[A,B] := \dim_{\mathbb{k}} \mathrm{Hom}_{\mathbb{k}Q}(A,B)$$

und

$$< d_1, d_2 >= \sum_{i=1}^{n} (d_1)_i \cdot (d_2)_i - \sum_{i=1}^{n-1} (d_1)_i \cdot (d_2)_{i+1}$$

für $d_1,\, d_2 \in \mathbb{N}^n$ die (nicht–symmetrische) Eulerform.
Ferner sei $F^X_{MN}(v^2) \in \mathbb{Z}[v]$ das sogenannte Hall–Polynom:

Da die Darstellungstheorie eines Dynkin–Köchers nicht vom Grundkörper $\Bbbk$ abhängt (die Unzerlegbaren sind immer durch positive Wurzeln klassifiziert), kann man Q–Darstellungen über jedem Körper realisieren. Also kann man

$$F^X_{MN}(q) = \#\{U \subset X \mid U \cong N,\ X/_U \cong M \text{ über } \mathbb{F}_q\}$$

für jede Primzahlpotenz q definieren.
Tatsächlich setzt sich diese Funktion zu einem Polynom in $\mathbb{Z}[q]$ fort.

Man zeigt nun, daß $H_q(Q)$ mit obigem Produkt tatsächlich eine assoziative Algebra bildet. Ringels Hauptsatz lautet:

Theorem 1.1: (Ringel)

Die Zuordnung

$$\eta : U_q^+(sl_n) \longrightarrow H_q(Q) \quad \textit{mit} \quad E_i \mapsto E_{[E_i]}$$

setzt sich zu einem Isomorphismus von $\mathbb{Q}(v)$*–Algebren fort.*

Dabei ist E_i die einfache Q–Darstellung zum Punkt i.

Also kann man durch

$$E_{[M]} := \eta^{-1}(E_{[M]}) \in U_q^+(sl_n)$$

eine Basis von $U_q^+(sl_n)$, die sogenannte *PBW–Basis*

$$B := \{E_{[M]} \in U_q^+(sl_n) \mid [M] \text{ Isomorphieklasse einer endlich–dim. } Q\text{–Darstellung}\}$$

definieren.
Prinzipiell ist die Struktur von $U_q^+(sl_n)$ damit bekannt, denn das Produkt von Basiselementen läßt sich komplett aus der Darstellungstheorie des Köchers Q bestimmen.

Aber: Die Berechnung von Hall–Polynomen ist im allgemeinen äußerst schwierig!

1.2 Die kanonische beziehungsweise kristalline Basis

Die Algebra $U_q^+(sl_n)$ besitzt einen Automorphismus $\overline{}$, der durch

$$\overline{E_i} := E_i \qquad \text{für alle} \quad i = 1, \ldots, n$$

und

$$\overline{v} := v^{-1}$$

gegeben ist. Die Zuordnung $\overline{}$ ist ein $\mathbb{Q}$–linearer (oder auch „$\mathbb{Q}(v)$–semilinearer“) Algebrenautomorphismus. Damit läßt sich der folgende Satz von Lusztig formulieren.

Theorem 1.2: (Lusztig)

Es gibt genau eine Basis $\mathcal{B}$ von $U_q^+(sl_n)$ mit

i) $\overline{b} = b$ *für alle* $b \in \mathcal{B}$.

ii) Schreibt man

$$b = \sum_{[M]} c_M E_{[M]},$$

so gibt es genau ein $[M_0]$ *mit*

$$c_{M_0} = 1 \qquad \textit{und} \qquad c_M \in v^{-1}\,\mathbb{Z}[v^{-1}] \quad \textit{für alle } M \ncong M_0.$$

Die Eindeutigkeit einer solchen Basis ist leicht einzusehen. Die Existenz dagegen ist ein sehr tiefliegendes Ergebnis, für das man die Struktur von $U_q^+(sl_n)$ und der PBW–Basis B sehr genau analysieren muß.

Die kristalline Basis $\mathcal{B}$ sieht sehr unscheinbar aus. Ihre Existenz hat allerdings enorme Konsequenzen! So lassen sich mit $\mathcal{B}$ zum Beispiel alle Fragen zur endlich-dimensionalen Darstellungstheorie der Lie–Algebra sl_n beziehungsweise der Gruppe $SL_n(\mathbb{C})$ auf kombinatorische Probleme reduzieren.

1.3 Basiswechsel zwischen kristalliner Basis und PBW–Basis

Eines der Hauptprobleme in der Theorie der kristallinen Basis $\mathcal{B}$ ist, daß ihre Kombinatorik zwar gut verstanden ist, ihre einzelnen Elemente aber unbekannt sind (außer für $n = 2$ und $n = 3$).

Man ist besonders an den *Basiswechselkoeffizienten* zwischen $\mathcal{B}$ und B interessiert, da man die PBW–Basis B als „verstanden“ annimmt.

Folgendes ist darüber bekannt:

Zunächst einmal kann man nach der zweiten Bedingung in Lusztigs Theorem jedem Element aus $\mathcal{B}$ eine Isomorphieklasse einer endlich–dimensionalen Q–Darstellung zuordnen. Also hat man eine Parametrisierung von $\mathcal{B}$ gefunden und setzt daher

$$\mathcal{B} = \{\mathcal{E}_{[M]} \mid [M] \text{ Isomorphieklasse einer endlich–dimensionalen } Q\text{–Darstellung}\}.$$

Dann gilt:

Satz 1.3: (Lusztig)

> *Schreibt man*
> $$\mathcal{E}_{[M]} = \sum_{[N]} \zeta_N^M E_{[N]},$$
> *so gilt*
>
> *a)* $\zeta_N^M \neq 0 \;\Rightarrow\; M \leq_{\deg} N$ *(Entartungen von* $\Bbbk Q$*–Moduln).*
>
> *b)* $\zeta_M^M = 1.$
>
> *c)* $\zeta_N^M \in v^{-1}\,\mathbb{Z}[v^{-1}]$ *für alle* $M \leq_{\deg} N$ *mit* $M \ncong N$.
>
> *d)* $v^{[N,N]-[M,M]} \cdot \zeta_N^M \in \mathbb{N}[v^2]$ *ist ein Polynom in* v^2*, dessen Koeffizienten die Dimensionen der „Schnitt-Kohomologiegruppen“ des Orbitabschlusses* $\overline{O_M}$ *am Punkt* N *sind.*

Der letzte Punkt im Satz benötigt tiefliegende Methoden der algebraischen Geometrie („Weil–Vermutungen“, „perverse Garben“), zeigt aber einen Zusammenhang zwischen der Struktur von $U_q^+(sl_n)$ und der Geometrie von Köcherdarstellungen auf.

2 Der Algorithmus

Durch explizite Berechnung einiger spezieller Hall–Polynome ist es kürzlich Markus Reineke[1] gelungen einen Algorithmus zur Berechnung der Polynome ζ_N^M (also der Basiswechselkoeffizienten) anzugeben.

2.1 Reduktion von ζ nach ω

Im ersten Schritt definiere die Laurent–Polynome $\omega_N^M \in \mathbb{Z}[v, v^{-1}]$ durch

$$\overline{E_{[M]}} = \sum_{[N]} \omega_N^M E_{[N]}.$$

Wieder gelten:

a) $\omega_N^M \neq 0 \;\Rightarrow\; M \leq_{\deg} N.$

b) $\omega_M^M = 1.$

Die Polynome ζ_N^M lassen sich aus den ω_N^M durch Auflösung einer Rekursion über die Entartungsordnung $\leq_{\deg}$ berechnen:

[1] Manuskript

Lemma 2.4:

Angenommen es sind bekannt

a) Alle Koeffizienten ω_N^M.

b) Alle Polynome ζ_L^M *mit* $M \leq_{\deg} L <_{\deg} N$.

Dann findet man die Polynome ζ_N^M *wie folgt: Schreibe*

$$\sum_{L:\, M \leq_{\deg} L <_{\deg} N} \overline{\zeta_L^M} \cdot \omega_N^L = \sum_{k \in \mathbb{Z}} a_k\, v^k.$$

Dann gilt:

$$\zeta_N^M = \sum_{k<0} a_k\, v^k.$$

Damit läuft die Rekursion durch, denn der Anfang $\zeta_M^M = 1$ ist bekannt.

Bleibt noch das Problem, die Entartungsordnung explizit zu machen. Hierzu benötigt man einen Satz von Bongartz und ein Lemma von Riedtmann.

Theorem 2.5: (Bongartz)

$$M \leq_{\deg} N \quad \Leftrightarrow \quad [M, U] \leq [N, U]$$

für alle unzerlegbaren $\Bbbk Q$*–Moduln* U.

Die Unzerlegbaren und die Dimensionen von Homomorphismenräumen zwischen Unzerlegbaren sind für $Q = A_n$ bekannt. Es gilt nämlich:

$$E_{ij}: \; 0 \xrightarrow{0} \cdots \xrightarrow{0} 0 \xrightarrow{0} \underset{\substack{\uparrow \\ i}}{\Bbbk} \xrightarrow{1} \cdots \xrightarrow{1} \underset{\substack{\uparrow \\ j}}{\Bbbk} \xrightarrow{0} 0 \xrightarrow{0} \cdots \xrightarrow{0} 0$$

sind für $1 \leq i \leq j \leq n$ die unzerlegbaren Moduln und

$$\dim_{\Bbbk} \operatorname{Hom}_{\Bbbk Q}(E_{ij}, E_{kl}) = [E_{ij}, E_{kl}] = \begin{cases} 1 & \text{falls } 1 \leq k \leq i \leq l \leq j \leq n \\ 0 & \text{sonst} \end{cases}$$

die Dimensionen der Homomorphismenräume.

Dadurch macht es der Satz von Bongartz leicht zu entscheiden, ob zwei (durch ihre Zerlegung in Unzerlegbare $M = \bigoplus_{i \leq j} E_{ij}^{m_{ij}}$) gegebene Moduln ineinander entarten.

Ferner existiert eine einfache Beziehung für die Dimensionen der Ext^1–Gruppen zwischen Unzerlegbaren, nämlich:

$$\dim_{\Bbbk} \operatorname{Ext}^1_{\Bbbk Q}(E_{ij}, E_{kl}) = \begin{cases} 1 & \text{falls } 1 \leq i < k \leq j+1 \leq l \leq n \\ 0 & \text{sonst} \end{cases}$$

Ein weiteres Lemma zur Degenerationsordnung ist das folgende:

Lemma 2.6: (RIEDTMANN)

> *Genau dann gilt $M \leq_{\deg} N$, wenn es Zahlen $x_U \in \mathbb{N}$ gibt, wobei U die nicht-projektiv Unzerlegbaren durchläuft, so daß gilt:*
>
> $$N \oplus \bigoplus_U (B_U)^{x_U} = M \oplus \bigoplus_U (U \oplus \tau U)^{x_U},$$
>
> *wobei die direkten Summen über alle U unzerlegbar, nicht-projektiv laufen und*
>
> $$0 \longrightarrow \tau U \longrightarrow B_U \longrightarrow U \longrightarrow 0$$
>
> *die* AUSLANDER–REITEN*–Sequenz mit Ende U bezeichnet.*

Wendet man dieses Lemma auf den Köcher $Q = A_n$ an, so erhält man aus einem Modul $M = \bigoplus_{i \leq j} E_{ij}^{m_{ij}}$ alle Entartungen $N = \bigoplus_{i \leq j} E_{ij}^{n_{ij}}$ indem man Zahlen $(x_{ij})_{1 \leq i \leq j < n}$ wählt und

$$n_{ij} = m_{ij} - x_{i-1,j} - x_{i,j-1} + x_{i-1,j-1} + x_{ij}$$

setzt (unter der Nebenbedingung, daß alle diese n_{ij} natürlich wieder ≥ 0 sein müssen!).

2.2 Reduktion von ω nach γ

Im zweiten Schritt definiere die LAURENT–Polynome $\gamma_N^M \in \mathbb{Z}[v, v^{-1}]$ durch

$$\mathcal{M}_{[M]} = \sum_{[N]} \gamma_N^M E_{[N]},$$

wobei die $\mathcal{M}_{[M]}$ eine dritte Basis von $U_q^+(sl_n)$, die sogenannte *Monomialbasis*[2], bilden.

Es ist

$$\begin{aligned}
\mathcal{M}_{[M]} \quad = \quad & E_n^{(a_{nn})} \\
& \cdot E_{n-1}^{(a_{n-1,n-1})} \cdot E_n^{(a_{n-1,n})} \\
& \cdot E_{n-2}^{(a_{n-2,n-2})} \cdot E_{n-1}^{(a_{n-2,n-1})} \cdot E_n^{(a_{n-2,n})} \\
& \vdots \\
& \cdot E_2^{(a_{22})} \cdot E_3^{(a_{23})} \cdot \ldots \cdot E_n^{(a_{2n})} \\
& \cdot E_1^{(a_{11})} \cdot E_2^{(a_{12})} \cdot \ldots \cdot E_n^{(a_{1n})} \\
\in \quad & U_q^+(sl_n)
\end{aligned}$$

[2]Diese Basis ist für die Struktur von $U_q^+(sl_n)$ eher uninteressant und wird nur in diesem Algorithmus benötigt.

Dabei ist

$$E_i^{(r)} = E_{[E_i{}^r]}$$

und die a_{ij}, $1 \leq i \leq j \leq n$ ergeben sich aus $M = \bigoplus_{i \leq j} E_{ij}^{m_{ij}}$ durch:

$$a_{ij} = \sum_{k=j}^{n} m_{ik} \qquad \text{für alle} \quad 1 \leq i \leq j \leq n.$$

Die Elemente $\mathcal{M}_{[M]}$ erfüllen die Bedingung

$$\overline{\mathcal{M}_{[M]}} = \mathcal{M}_{[M]}$$

und es gilt wieder

a) $\gamma_N^M \neq 0 \quad \Rightarrow \quad M \leq_{\deg} N$.

b) $\gamma_M^M = 1$.

Daraus läßt sich wieder ein rekursives Berechnungsverfahren ableiten.

Lemma 2.7:

Angenommen es sind bekannt

a) Alle Koeffizienten γ_N^M.

b) Alle Polynome ω_L^M *mit* $M \leq_{\deg} L <_{\deg} N$.

Dann erhält man die Koeffizienten ω_N^M *durch:*

$$\omega_N^M = \gamma_N^M - \sum_{L\colon M <_{\deg} L \leq_{\deg} N} \overline{\gamma_L^M} \cdot \omega_N^L.$$

2.3 Reduktion von γ nach c

Im dritten Schritt definiere die Koeffizienten $c_i^r(M, M_a)$ durch

$$E_i^{(r)} \cdot E_{[M]} = \sum_{a=(a_i,\ldots,a_n)} c_i^r(M, M_a) \cdot E_{[M_a]} \tag{1}$$

für gewisse Moduln M_a, die im folgenden Kapitel 2.4 definiert werden. Für einen gegeben Modul $M = \bigoplus_{k \leq l} E_{kl}^{m_{kl}}$ läuft die Summe über alle Tupel $(a_i, \ldots a_n)$ mit

i) $a_k \geq 0$ für alle $k = i, \ldots, n$.

ii) $\sum_{k=i}^{n} a_k = r$.

Damit läßt sich das Element $\mathcal{M}_{[M]}$, d.h. alle Koeffizienten γ_N^M, berechnen, indem man mit dem Eins–Element $E_{[0]}$ der Algebra $U_q^+(sl_n)$ startet und wiederholt mit entsprechenden E_i's von links multipliziert, was bei Kenntnis der $c_i^r(M, M_a)$ explizit möglich ist.

2.4 Berechnung der Koeffizienten $c_i^r(M, M_a)$

Im vierten und letzten Schritt geht das einzig mathematisch Neue vom Algorithmus ein, denn die Koeffizienten $c_i^r(M, M_a)$ waren bislang nicht bekannt.

Satz 2.8:

Schreibe $M = \bigoplus_{k \leq l} E_{kl}^{m_{kl}}$. *Dann gilt:*

$$c_i^r(M, M_a) = v^{\sum_{k=i}^n a_k \left(\sum_{l=k+1}^n (m_{il} - m_{i+1,l}) \right) + \sum_{i \leq k < l \leq n} a_k a_l} \cdot \prod_{k=i}^n \begin{bmatrix} m_{ik} + a_k \\ a_k \end{bmatrix}. \tag{2}$$

Hierbei bedeuten:

$$[s] := \frac{v^s - v^{-s}}{v - v^{-1}} = v^{s-1} + v^{s-3} + \cdots + v^{3-k} + v^{1-k}$$

$$[s]! := [1] \cdot [2] \cdot \cdots \cdot [s]$$

$$\begin{bmatrix} s+t \\ t \end{bmatrix} := \frac{[s+t]!}{[s]! \cdot [t]!}$$

$$M_a := \bigoplus_{1 \leq k \leq l \leq n} E_{kl}^{m'_{kl}} \qquad \text{mit} \quad m'_{kl} = \begin{cases} m_{kl} + a_l & \text{falls } k = i \\ m_{kl} - a_l & \text{falls } k = i+1 \\ m_{kl} & \text{sonst} \end{cases}$$

Wegen der Beziehung

$$\sum_{i \leq k < l \leq n} a_k a_l = \sum_{k=i}^n \sum_{l=k+1}^n a_k a_l$$

läßt sich die Gleichung (2) umschreiben in

$$c_i^r(M, M_a) = v^{\sum_{k=i}^n a_k \left(\sum_{l=k+1}^n (m_{il} - m_{i+1,l} + a_l) \right)} \cdot \prod_{k=i}^n \begin{bmatrix} m_{ik} + a_k \\ a_k \end{bmatrix}. \tag{3}$$

Für die GAUSS'schen Binomialkoeffizienten ist leider keine explizite Formel als Polynome in $\mathbb{N}[v, v^{-1}]$ bekannt. Sie lassen sich jedoch rekursiv durch

$$\begin{bmatrix} s \\ t \end{bmatrix} = v^{s-t} \cdot \begin{bmatrix} s-1 \\ t-1 \end{bmatrix} + v^{-t} \begin{bmatrix} s-1 \\ t \end{bmatrix}$$

berechnen. Der Abbruch der Rekursion wird geliefert durch

$$\begin{bmatrix} s \\ s \end{bmatrix} = \begin{bmatrix} s \\ 0 \end{bmatrix} = 1.$$

Liest man die vier Schritte des Algorithmus umgekehrt, so lassen sich die Basiswechsel–Polynome ζ_N^M durch iteriertes Anwenden der Formel (1) aus dem dritten Schritt auf den Null–Modul und zweimaliges Auflösen einer Rekursion über die Entartungsordnung explizit berechnen.

3 Das Programm

3.1 Anforderungen an das Programm

Als Eingabe sollte das Programm die folgenden Parameter akzeptieren:

a) Die Länge n des Köchers A_n.

b) Koeffizienten m_{ij}, $1 \leq i,j \leq n$, durch welche ein beliebiger $\Bbbk Q$–Modul als Zerlegung in Unzerlegbare $M = \bigoplus_{i \leq j} E_{ij}^{m_{ij}}$ eindeutig festliegt.

Für die so spezifizierte Köcherlänge n und den Modul M soll dann eine Liste von Paaren der Form

$$N \text{ mit } M \leq_{\deg} N, \ \zeta_N^M$$

ausgegeben werden, d.h. jeder Modul N mit $M \leq_{\deg} N$ und der zugehörige Basiswechselkoeffizient ζ_N^M.

Realistische Schranken für die Eingabewerte liegen bei

$$n \leq 8 \qquad \text{und} \qquad \dim_{\Bbbk} M \leq 20.$$

Ferner kann die Rechenzeit für Eingaben nahe diesen Schranken im mehrstündigen Bereich liegen, denn bereits für $n \geq 4$ sind so gut wie keine nicht-trivialen Beispiele bekannt.

Sowohl die Eingabe der Parameter als auch die Ausgabe der Liste von Paaren dürfen schlicht gehalten werden.

3.2 Wahl der Programmiersprache

Für die Umsetzung des Algorithmus in ein ausführbares Programm kamen die Compiler–Programmiersprache C und die interpretierte Sprache *Mathematica* in die engere Auswahl.
Die Entscheidung fiel aus folgenden Gründen schließlich auf C:

a) Die Programmiersprache C ist stark verbreitet und gehört zu den Sprachen, in denen heutzutage programmiert wird. Daher und durch die universelle Einsetzbarkeit von C wird eine große Portabilität unseres Programms gewährleistet.

   Im Gegensatz dazu ist *Mathematica* zwar an vielen Universitäten vorhanden, aber dort nicht immer allgemein zugänglich oder durch alternative Sprachen wie etwa *Maple* in ihrer Nutzung beeinträchtigt. Ferner erfordert *Mathematica* einige Benutzungskenntnisse, was den Einsatzbereich eines in dieser Sprache geschriebenen Programms zusätzlich eingeschränkt.

b) Aus der mathematischen Beschreibung des Algorithmus ist sofort ersichtlich, daß dieser eine *Vielzahl* an Rechenoperationen mit Polynomen und insbesondere LAURENT–Polynomen umfaßt.

   Zur Handhabung der Polynome stellt *Mathematica* eine ganze Reihe von bequemen Funktionen und Operationen (Polynomdivision, LAURENT–Reihen, Binomialkoeffizienten, ...) zur Verfügung. Diese würden es z.B. erlauben die rekursive Berechnung der GAUSS'schen Binomialkoeffizienten zu vermeiden.

   Die Arithmetik mit Polynomen muß in *C* zwar erst programmiert werden, aber dieser zusätzliche Aufwand ist gering. Auch die Verwaltung der Polynome erfordern keine größeren Umstände.

c) Nahe der Parameter–Eingabeschranken wird der Speicherbedarf und der Rechenaufwand verhältnismäßig groß, so daß der Zeitaufwand bei einem auf *Mathematica* basierenden Programm erheblich höher ist als bei einem vergleichbaren *C*–Programm (Interpreter ↔ Compiler). Insbesondere werden wohl die zum Teil tiefgehenden Rekursionen über die Degenerationsordnung mit einem *C*–orientierten Programm schneller abgearbeitet als durch *Mathematica*–Programme.

Die Wahl der Programmiersprache *C* begründet sich also hauptsächlich durch die große Portabilität und der besseren Rechengeschwindigkeit von *C* gegenüber *Mathematica*, die durch wenige Zusatzprozeduren (Arithmetik mit Polynomen) nicht gemindert wird.

Als Compiler wurde der *GNU–C–Compiler* gewählt, der auf praktisch allen Plattformen zur Verfügung steht. Das Programm nutzt einige Besonderheiten dieses Compilers ([0]–Arrays, inline–Funktionen), was aber die Portabilität nicht einschränkt.

3.3 Aufruf und Benutzung

Die Aufrufschablone des Programms lautet
`kristallin [--test] [--all] [--verbose] [--debug] [--logmem]`

Dabei bedeuten die einzelnen Optionen:

`--test` Die in Abschnitt 3.4 aufgeführten Tests werden durchgeführt. Schlägt einer der Tests fehl, wird der Text „Programm buggy“ auf `stderr` ausgegeben und das Programm wechselt in den `--debug`-Modus. Natürlich verlangsamt diese Option den Programmlauf etwas.

`--all` Nach Start des Programms wird als erstes die Länge des zu behandelnden Köchers erfragt, danach der zu betrachtende Dimensionsvektor. Wird diese Option nicht angegeben, wird anschließend der zu betrachtende Modul gegeben durch seine Zerlegung in Unzerlegbare eingegeben. Wird diese Option aber angegeben, werden nacheinander alle Moduln, die zum gegebenen Dimensionsvektor passen, behandelt. Diese Option eignet sich besonders, um einen gewissen Überblick zu bekommen und um die Testläufe ökonomischer durchführen zu können.

`--verbose` Zusätzlich zu den ζ_N^M werden auch alle benötigten ω_N^M und γ_N^M ausgegeben. Weiterhin werden vor der Eingabe des Moduls alle Unzerlegbaren und alle Moduln zum gegebenen Dimensionsvektor ausgegeben. Die Moduln werden dabei durch eine Kurzschreibweise dargestellt, und zwar werden die Multiplizitäten der Unzerlegbaren in der Reihenfolge $E_{1,n}$, $E_{1,n-1}$, $E_{2,n}$, $E_{1,n-2}$, …, E_{11}, …, E_{nn} ausgegeben (n ist die Länge des Köchers).

`--debug` Zusätzlich zu `--verbose` wird die Berechnung der $\mathcal{M}_{[M]}$ ausführlich angezeigt, einschließlich aller $c_i^r(M, M_a)$.

`--logmem` Der maximal benutzte Speicher für diesen Programmlauf wird angezeigt. Ebenso wie `--test` wird das Programm etwas verlangsamt, außerdem wird der Speicherverbrauch etwas erhöht. Dieser zusätzliche Speicherverbrauch wird aber nicht mitgerechnet, ebenso bleibt der durch `malloc` eingebrachte Overhead unberücksichtigt.

Bei der Eingabe der Moduln als direkte Summe der Unzerlegbaren ist zu beachten, daß nur die Multiplitzitäten der Unzerlegbaren abgefragt werden, die noch nicht eindeutig durch vorhergehende Eingaben und den Dimensionsvektor festgelegt sind. Die Eingaben müssen im bei der Eingabeaufforderung angegebenen Bereich liegen, damit sie akzeptiert werden.

3.4 Tests

Die Theorie der Quantengruppen und der Köcheralgebren liefert mehrere, einfach zu überprüfende Tests für das Programm. Zu diesen Tests zählen:

a) Für die Koeffizienten $c_i^r(M, M_a)$ gilt:

$$E_i^{(r)} \cdot E_{[M]} = c_i^r(M, M_a) \cdot E_{[M \oplus E_i{}^r]} \qquad \text{falls} \quad \mathrm{Ext}^1_{\Bbbk Q}(E_i{}^r, M) = 0.$$

Die Bedingung $\mathrm{Ext}^1_{\Bbbk Q}(E_i{}^r, M) = 0$ läßt sich leicht für einen gegeben Modul $M = \bigoplus_{k \le l} E_{kl}^{m_{kl}}$ durch

$$m_{i+i,l} = 0 \quad \text{für alle } l = i+1, \dots, n$$

überprüfen.

b) Ein weiteres Kriterium für die $c_i^r(M, M_a)$–Koeffizienten ergibt sich aus:

$$E_i^{(r)} \cdot E_{[M]} = \sum_{a=(a_i,\dots,a_n)} v^\alpha \cdot E_{[M_a]} \qquad \text{falls} \quad \mathrm{Hom}_{\Bbbk Q}(M, E_i) = 0,$$

wobei $\alpha \in \mathbb{Z}$. Für einen Modul $M = \bigoplus_{k \le l} E_{kl}^{m_{kl}}$ gilt dabei:

$$\mathrm{Hom}_{\Bbbk Q}(M, E_i) = 0 \quad \Leftrightarrow \quad m_{il} = 0 \ \text{ für alle } l = i, \dots, n.$$

c) Bei der Berechnung der ζ–Koeffizienten muß die folgende Bedingung erfüllt sein:
$$\mathcal{E}_{[M]} = E_1^{(d_1)} \cdots E_n^{(d_n)} = \sum_{N:\,\underline{\dim}\,N=d} \underbrace{v^{-\dim \mathrm{Ext}^1(N,N)}}_{=\zeta_N^M} \cdot E_{[N]}.$$
Dabei ist vorausgesetzt:
$$\mathrm{Ext}^1_{\Bbbk Q}(M,M) = 0 \qquad \text{und} \qquad \underline{\dim}\, M = d.$$

d) Der wirksamste Test überhaupt ist die Überprüfung der ζ–Koeffizienten gemäß Satz 1.3 nach LUSZTIG:
$$v^{[N,N]-[M,M]} \cdot \zeta_N^M \in \mathbb{N}[v^2],$$
da die Positivität der Polynom–Koeffizienten mit tiefliegenden Methoden der algebraischen Geometrie bewiesen worden ist.

Alle obigen Tests sind mit ins Programm eingebaut und werden bei aktivierter Option `--test` automatisch durchlaufen.

Insbesondere für den Köcher A_2 sind explizite Formeln bekannt, die im Vorfeld nachgerechnet wurden und mit der Programmausgabe übereinstimmen.

3.5 Umsetzung des Algorithmus in Programmtext

Der Algorithmus wurde wie im Theorieteil (Kapitel 2) geschildert umgesetzt. Dabei sind dem Programm einige erklärende Kommentare als Programmtext hinzugefügt worden, um wichtige Programmdetails zu beleuchten.

Das Programm besitzt *keine* Schranken für die Eingabeparameter, wie sie in Kapitel 3.1 angegeben sind. Praktisch sind beliebig hohe Werte für Köcherlänge und Modul–Dimension möglich. Doch die protokollierten Testläufe machen deutlich, daß große Eingabewerte hohen Speicherplatzbedarf fordern und die Geduld des Programmnutzers auf die Probe stellen.

3.6 Probeläufe

Die Probeläufe wurden auf der SGI–Workstation O_2 mit einem MIPS R5000 Prozessor (180 MHz Taktfrequenz) und 160 MB RAM durchgeführt. Dabei waren stets die Optionen `--all` `--test` `--logmem` aktiviert und die Ausgabe direkt auf dem Bildschirm präsentiert (also ohne Ausgabeumlenkung).

In der folgenden Tabelle sind einige der Probeläufe protokolliert:

Dimensionsvektor	Anzahl der Moduln	Speicherverbrauch in [kB]	Rechenzeit in [s]
2–2	3	1	< 1
1–2–2	7	3	< 1
2–2–2	10	6	< 1
2–2–2–2	35	60	1
1–2–3–4	40	105	< 1
2–2–3–3	56	182	1
3–3–3–3	112	811	5
4–4–4–4	294	7306	104
2–2–2–2–2	125	647	5
2–2–2–2–2–2	450	7058	90
2–2–2–2–2–2–2	2625	77752	1740

Sowohl die Rechenzeit als auch der Speicheraufwand werden erheblich geringer, wenn die Option `--all` nicht benutzt wird, sondern einzelne Moduln berechnet werden.

Es sei schließlich noch erwähnt, daß während aller Probeläufe die Option `--test` keine Fehlermeldung produzierte!