QuaGroup

Version 1.3

Willem A. de Graaf

Willem A. de Graaf — Email: degraaf@science.unitn.it
— Homepage: http://www.science.unitn.it/~degraaf

Copyright

Contents

Chapter 1

Introduction

This is the manual for the GAP package QuaGroup, for doing computations with quantized enveloping algebras of semisimple Lie algebras.

Apart from the chapter you are currently reading, this document consists of two chapters. In Chapter 2 we give a short summary of parts of the theory of quantized enveloping algebras. This fixes the notations and definitions that we use. Then in Chapter 3 we describe the functions that constitute the package.

The package can be obtained from `http://www.math.uu.nl/people/graaf/quagroup.html` The directory `quagroup/doc` contains the manual of the package in `dvi`, `ps`, `pdf` and `html` format. The manual was built with the GAP share package GAPDoc, [LN01]. This means that, in order to be able to use the on-line help of QuaGroup, you have to install GAPDoc before calling `LoadPackage("quagroup");`.

The main algorithm of the package (on which virtually the whole functionality relies) is a method for computing with so-called PBW-type bases, analogous to Poincaré-Birkhoff-Witt bases in universal enveloping algebras. In both cases commutation relations between the generators are used. However, in the latter case all commutation relations are of the form $yx = xy + z$, where x, y are generators, and z is a linear combination of generators. In the case of quantized enveloping algebras the situation is generally much more complicated. For example, in the quantized enveloping algebra of type E_7 we have the following relation:

Example

```
F62*F26 = (q)*F26*F62+(1-q^2)*F28*F61+(-q+q^3)*F30*F60+(q^2-q^4)*F31*F59+
          (q^2-q^4)*F33*F58+(-q^3+q^5)*F34*F57+(q^4-q^6)*F35*F56+
          (q^-1-q-q^5+q^7)*F36*F55+(q^6)*F54
```

Due to the complexity of these commutation relations, some computations (even with rather small input) may take quite some time.

Remark: The package can deal with quantized enveloping algebras corresponding to root systems of rank at least up to eight, except E_8. In that case the computation of the necessary commutation relations took more than 2 GB. I wish to thank Steve Linton for trying this computation on the machines in St Andrews.

The following example illustrates some of the features of the package.

Example

```
# We define a root system by giving its type:
gap> R:= RootSystem( "B", 2 );
<root system of type B2>
# Corresponding to the root system we define a quantized enveloping algebra:
```

```
gap> U:= QuantizedUEA( R );
QuantumUEA( <root system of type B2>, Qpar = q )
# It is generated by the generators of a so-called PBW-type basis:
gap> GeneratorsOfAlgebra( U );
[ F1, F2, F3, F4, K1, K1+(q^-2-q^2)*[ K1 ; 1 ], K2, K2+(q^-1-q)*[ K2 ; 1 ],
  E1, E2, E3, E4 ]
# We can construct highest-weight modules:
gap> V:= HighestWeightModule( U, [1,1] );
<16-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = q )>
# For modules of small dimension we can compute the corresponding
# R-matrix:
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> V:= HighestWeightModule( U, [1,0] );;
gap> RMatrix( V );
[ [ q^2, 0, 0, 0, 0, 0, 0, 0, 0 ], [ 0, q^3, 0, q^2-q^4, 0, 0, 0, 0, 0 ],
  [ 0, 0, q^3, 0, 0, 0, q^2-q^4, 0, 0 ], [ 0, 0, 0, q^3, 0, 0, 0, 0, 0 ],
  [ 0, 0, 0, 0, q^2, 0, 0, 0, 0 ], [ 0, 0, 0, 0, 0, q^3, 0, q^2-q^4, 0 ],
  [ 0, 0, 0, 0, 0, 0, q^3, 0, 0 ], [ 0, 0, 0, 0, 0, 0, 0, q^3, 0 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0, q^2 ] ]
# We can compute elements of the canonical basis of the "negative" part
# of a quantized enveloping algebra:
gap> U:= QuantizedUEA( RootSystem("F",4) );;
gap> B:= CanonicalBasis( U );
<canonical basis of QuantumUEA( <root system of type F4>, Qpar = q ) >
gap> p:= PBWElements( B, [0,1,2,1] );
[ F3*F9^(2)*F24, F3*F9*F23+(q^2)*F3*F9^(2)*F24,
  (q+q^3)*F3*F9^(2)*F24+F7*F9*F24, (q^2)*F3*F9*F23+(q^2+q^4)*F3*F9^(2)*F
    24+(q)*F7*F9*F24+F7*F23, (q^4)*F3*F9^(2)*F24+(q)*F7*F9*F24+F8*F24,
  (q^4)*F3*F9*F23+(q^6)*F3*F9^(2)*F24+(q^3)*F7*F9*F24+(q^2)*F7*F23+(q^2)*F
    8*F24+F9*F21, (q+q^3)*F3*F9*F23+(q^3+q^5)*F3*F9^(2)*F24+(q^2)*F7*F9*F
    24+(q)*F7*F23+(q)*F9*F21+F16 ]
# We can construct (anti-) automorphisms of quantized enveloping
# algebras:
gap> t:= AntiAutomorphismTau( U );
<anti-automorphism of QuantumUEA( <root system of type F4>, Qpar = q )>
gap> Image( t, p[1] );
(q^4)*F3*F9*F23+(q^6)*F3*F9^(2)*F24+(q^3)*F7*F9*F24+(q^2)*F7*F23+(q^2)*F8*F
24+F9*F21
# (This is the sixth element of p.)
```

Chapter 2

Background

In this chapter we summarize some of the theoretical concepts with which QuaGroup operates. Due to the rather mathematical nature of this chapter everything has been written in LaTeX. Therefore, it will be almost unreadable in the html version.

2.1 Gaussian Binomials

Let v be an indeterminate over $\mathbb{Q}$. For a positive integer n we set

$$[n] = v^{n-1} + v^{n-3} + \cdots + v^{-n+3} + v^{-n+1}.$$

We say that $[n]$ is the *Gaussian integer* corresponding to n. The *Gaussian factorial* $[n]!$ is defined by

$$[0]! = 1,\ [n]! = [n][n-1]\cdots[1],\ \text{for } n > 0.$$

Finally, the *Gaussian binomial* is

$$\begin{bmatrix} n \\ k \end{bmatrix} = \frac{[n]!}{[k]![n-k]!}.$$

2.2 Quantized enveloping algebras

Let $\mathfrak{g}$ be a semisimple Lie algebra with root system Φ. By $\Delta = \{\alpha_1, \ldots, \alpha_l\}$ we denote a fixed simple system of Φ. Let $C = (C_{ij})$ be the Cartan matrix of Φ (with respect to Δ, i.e., $C_{ij} = \langle \alpha_i, \alpha_j^\vee \rangle$). Let $d_1, \ldots, d_l$ be the unique sequence of positive integers with greatest common divisor 1, such that $d_i C_{ji} = d_j C_{ij}$, and set $(\alpha_i, \alpha_j) = d_j C_{ij}$. (We note that this implies that (α_i, α_i) is divisible by 2.) By P we denote the weight lattice, and we extend the form $(\ ,\ )$ to P by bilinearity.

By $W(\Phi)$ we denote the Weyl group of Φ. It is generated by the simple reflections $s_i = s_{\alpha_i}$ for $1 \le i \le l$ (where s_α is defined by $s_\alpha(\beta) = \beta - \langle \beta, \alpha^\vee \rangle \alpha$).

We work over the field $\mathbb{Q}(q)$. For $\alpha \in \Phi$ we set

$$q_\alpha = q^{\frac{(\alpha,\alpha)}{2}},$$

and for a non-negative integer n, $[n]_\alpha = [n]_{v=q_\alpha}$; $[n]_\alpha!$ and $\begin{bmatrix} n \\ k \end{bmatrix}_\alpha$ are defined analogously.

The quantized enveloping algebra $U_q(\mathfrak{g})$ is the associative algebra (with one) over $\mathbb{Q}(q)$ generated by F_α, K_α, K_α^{-1}, E_α for $\alpha \in \Delta$, subject to the following relations

$$\begin{aligned} K_\alpha K_\alpha^{-1} = K_\alpha^{-1} K_\alpha = 1,\ K_\alpha K_\beta &= K_\beta K_\alpha \\ E_\beta K_\alpha &= q^{-(\alpha,\beta)} K_\alpha E_\beta \\ K_\alpha F_\beta &= q^{-(\alpha,\beta)} F_\beta K_\alpha \\ E_\alpha F_\beta &= F_\beta E_\alpha + \delta_{\alpha,\beta} \frac{K_\alpha - K_\alpha^{-1}}{q_\alpha - q_\alpha^{-1}} \end{aligned}$$

together with, for $\alpha \neq \beta \in \Delta$,

$$\sum_{k=0}^{1-\langle\beta,\alpha^\vee\rangle} (-1)^k \begin{bmatrix} 1-\langle\beta,\alpha^\vee\rangle \\ k \end{bmatrix}_\alpha E_\alpha^{1-\langle\beta,\alpha^\vee\rangle-k} E_\beta E_\alpha^k = 0$$

$$\sum_{k=0}^{1-\langle\beta,\alpha^\vee\rangle} (-1)^k \begin{bmatrix} 1-\langle\beta,\alpha^\vee\rangle \\ k \end{bmatrix}_\alpha F_\alpha^{1-\langle\beta,\alpha^\vee\rangle-k} F_\beta F_\alpha^k = 0.$$

The quantized enveloping algebra has an automorphism ω defined by $\omega(F_\alpha) = E_\alpha$, $\omega(E_\alpha) = F_\alpha$ and $\omega(K_\alpha) = K_\alpha^{-1}$. Also there is an anti-automorphism τ defined by $\tau(F_\alpha) = F_\alpha$, $\tau(E_\alpha) = E_\alpha$ and $\tau(K_\alpha) = K_\alpha^{-1}$. We have $\omega^2 = 1$ and $\tau^2 = 1$.

If the Dynkin diagram of Φ admits a diagram automorphism π, then π induces an automorphism of $U_q(\mathfrak{g})$ in the obvious way (π is a permutation of the simple roots; we permute the F_α, E_α, $K_\alpha^{\pm 1}$ accordingly).

Now we view $U_q(\mathfrak{g})$ as an algebra over $\mathbb{Q}$, and we let $\overline{\ }: U_q(\mathfrak{g}) \to U_q(\mathfrak{g})$ be the automorphism defined by $\overline{F_\alpha} = F_\alpha$, $\overline{K_\alpha} = K_\alpha^{-1}$, $\overline{E_\alpha} = E_\alpha$, $\overline{q} = q^{-1}$.

2.3 Representations of $U_q(\mathfrak{g})$

Let $\lambda \in P$ be a dominant weight. Then there is a unique irreducible highest-weight module over $U_q(\mathfrak{g})$ with highest weight λ. We denote it by $V(\lambda)$. It has the same character as the irreducible highest-weight module over $\mathfrak{g}$ with highest weight λ. Furthermore, every finite-dimensional $U_q(\mathfrak{g})$-module is a direct sum of irreducible highest-weight modules.

It is well-known that $U_q(\mathfrak{g})$ is a Hopf algebra. The comultiplication $\Delta : U_q(\mathfrak{g}) \to U_q(\mathfrak{g}) \otimes U_q(\mathfrak{g})$ is defined by

$$\begin{aligned} \Delta(E_\alpha) &= E_\alpha \otimes 1 + K_\alpha \otimes E_\alpha \\ \Delta(F_\alpha) &= F_\alpha \otimes K_\alpha^{-1} + 1 \otimes F_\alpha \\ \Delta(K_\alpha) &= K_\alpha \otimes K_\alpha. \end{aligned}$$

(Note that we use the same symbol to denote a simple system of Φ; of course this does not cause confusion.) The counit $\varepsilon : U_q(\mathfrak{g}) \to \mathbb{Q}(q)$ is a homomorphism defined by $\varepsilon(E_\alpha) = \varepsilon(F_\alpha) = 0$, $\varepsilon(K_\alpha) = 1$. Finally, the antipode $S : U_q(\mathfrak{g}) \to U_q(\mathfrak{g})$ is an anti-automorphism given by $S(E_\alpha) = -K_\alpha^{-1} E_\alpha$, $S(F_\alpha) = -F_\alpha K_\alpha$, $S(K_\alpha) = K_\alpha^{-1}$.

Using Δ we can make the tensor product $V \otimes W$ of two $U_q(\mathfrak{g})$-modules V, W into a $U_q(\mathfrak{g})$-module. The counit ε yields a trivial 1-dimensional $U_q(\mathfrak{g})$-module. And with S we can define a $U_q(\mathfrak{g})$-module structure on the dual V^* of a $U_q(\mathfrak{g})$-module V, by $(u \cdot f)(v) = f(S(u) \cdot v)$.

The Hopf algebra structure given above is not the only one possible. For example, we can twist Δ, ε, S by an automorphism, or an anti-automorphism f. The twisted comultiplication is given by

$$\Delta^f = f \otimes f \circ \Delta \circ f^{-1}.$$

The twisted antipode by

$$S^f = \begin{cases} f \circ S \circ f^{-1} & \text{if } f \text{ is an automorphism} \\ f \circ S^{-1} \circ f^{-1} & \text{if } f \text{ is an anti-automorphism.} \end{cases}$$

And the twisted counit by $\varepsilon^f = \varepsilon \circ f^{-1}$ (see [Jan96], 3.8).

2.4 PBW-type bases

The first problem one has to deal with when working with $U_q(\mathfrak{g})$ is finding a basis of it, along with an algorithm for expressing the product of two basis elements as a linear combination of basis elements. First of all we have that $U_q(\mathfrak{g}) \cong U^- \otimes U^0 \otimes U^+$ (as vector spaces), where U^- is the subalgebra generated by the F_α, U^0 is the subalgebra generated by the K_α, and U^+ is generated by the E_α. So a basis of $U_q(\mathfrak{g})$ is formed by all elements FKE, where F, K, E run through bases of U^-, U^0, U^+ respectively.

Finding a basis of U^0 is easy: it is spanned by all $K_{\alpha_1}^{r_1} \cdots K_{\alpha_l}^{r_l}$, where $r_i \in \mathbb{Z}$. For U^-, U^+ we use the so-called *PBW-type* bases. They are defined as follows. For $\alpha, \beta \in \Delta$ we set $r_{\beta,\alpha} = -\langle \beta, \alpha^\vee \rangle$. Then for $\alpha \in \Delta$ we have the automorphism $T_\alpha : U_q(\mathfrak{g}) \to U_q(\mathfrak{g})$ defined by

$$\begin{aligned}
T_\alpha(E_\alpha) &= -F_\alpha K_\alpha \\
T_\alpha(E_\beta) &= \sum_{i=0}^{r_{\beta,\alpha}} (-1)^i q_\alpha^{-i} E_\alpha^{(r_{\beta,\alpha}-i)} E_\beta E_\alpha^{(i)} \text{ (for } \alpha \neq \beta) \\
T_\alpha(K_\beta) &= K_\beta K_\alpha^{r_{\beta,\alpha}} \\
T_\alpha(F_\alpha) &= -K_\alpha^{-1} E_\alpha \\
T_\alpha(F_\beta) &= \sum_{i=0}^{r_{\beta,\alpha}} (-1)^i q_\alpha^i F_\alpha^{(i)} F_\beta F_\alpha^{(r_{\beta,\alpha}-i)} \text{ (for } \alpha \neq \beta),
\end{aligned}$$

(where $E_\alpha^{(k)} = E_\alpha^k / [k]_\alpha !$, and likewise for $F_\alpha^{(k)}$).

Let $w_0 = s_{i_1} \cdots s_{i_t}$ be a reduced expression for the longest element in the Weyl group $W(\Phi)$. For $1 \leq k \leq t$ set $F_k = T_{\alpha_{i_1}} \cdots T_{\alpha_{i_{k-1}}}(F_{\alpha_{i_k}})$, and $E_k = T_{\alpha_{i_1}} \cdots T_{\alpha_{i_{k-1}}}(E_{\alpha_{i_k}})$. Then $F_k \in U^-$, and $E_k \in U^+$. Furthermore, the elements $F_1^{m_1} \cdots F_t^{m_t}$, $E_1^{n_1} \cdots E_t^{n_t}$ (where the m_i, n_i are non-negative integers) form bases of U^- and U^+ respectively.

The elements F_α and E_α are said to have weight $-\alpha$ and α respectively, where α is a simple root. Furthermore, the weight of a product ab is the sum of the weights of a and b. Now elements of U^-, U^+ that are linear combinations of elements of the same weight are said to be homogeneous. It can be shown that the elements F_k, and E_k are homogeneous of weight $-\beta$ and β respectively, where $\beta = s_{i_1} \cdots s_{i_{k-1}}(\alpha_{i_k})$.

In the sequel we use the notation $F_k^{(m)} = F_k^m / [m]_{\alpha_{i_k}} !$, and $E_k^{(n)} = E_k^n / [n]_{\alpha_{i_k}} !$.

2.5 The $\mathbb{Z}$-form of $U_q(\mathfrak{g})$

For $\alpha \in \Delta$ set

$$\begin{bmatrix} K_\alpha \\ n \end{bmatrix} = \prod_{i=1}^{n} \frac{q_\alpha^{-i+1} K_\alpha - q_\alpha^{i-1} K_\alpha^{-1}}{q_\alpha^i - q_\alpha^{-i}}.$$

Then according to [Lus90], Theorem 6.7 the elements

$$F_1^{(k_1)} \cdots F_t^{(k_t)} K_{\alpha_1}^{\delta_1} \begin{bmatrix} K_{\alpha_1} \\ m_1 \end{bmatrix} \cdots K_{\alpha_l}^{\delta_l} \begin{bmatrix} K_{\alpha_l} \\ m_l \end{bmatrix} E_1^{(n_1)} \cdots E_t^{(n_t)},$$

(where $k_i, m_i, n_i \geq 0$, $\delta_i = 0, 1$) form a basis of $U_q(\mathfrak{g})$, such that the product of any two basis elements is a linear combination of basis elements with coefficients in $\mathbb{Z}[q, q^{-1}]$. The quantized enveloping algebra over $\mathbb{Z}[q, q^{-1}]$ with this basis is called the $\mathbb{Z}$-form of $U_q(\mathfrak{g})$, and denoted by $U_\mathbb{Z}$. Since $U_\mathbb{Z}$ is defined over $\mathbb{Z}[q, q^{-1}]$ we can specialize q to any nonzero element ε of a field F, and obtain an algebra U_ε over F.

We call $q \in \mathbb{Q}(q)$, and $\varepsilon \in F$ the quantum parameter of $U_q(\mathfrak{g})$ and U_ε respectively.

Let λ be a dominant weight, and $V(\lambda)$ the irreducible highest weight module of highest weight λ over $U_q(\mathfrak{g})$. Let $v_\lambda \in V(\lambda)$ be a fixed highest weight vector. Then $U_\mathbb{Z} \cdot v_\lambda$ is a $U_\mathbb{Z}$-module. So by specializing q to an element ε of a field F, we get a U_ε-module. We call it the Weyl module of highest weight λ over U_ε. We note that it is not necessarily irreducible.

2.6 The canonical basis

As in Section 2.4 we let U^- be the subalgebra of $U_q(\mathfrak{g})$ generated by the F_α for $\alpha \in \Delta$. In [Lus0a] Lusztig introduced a basis of U^- with very nice properties, called the *canonical basis*. (Later this basis was also constructed by Kashiwara, using a different method. For a brief overview on the history of canonical bases we refer to [Com06].)

Let $w_0 = s_{i_1} \cdots s_{i_t}$, and the elements F_k be as in Section 2.4. Then, in order to stress the dependency of the monomial

$$F_1^{(n_1)} \cdots F_t^{(n_t)} \tag{2.1}$$

on the choice of reduced expression for the longest element in $W(\Phi)$ we say that it is a w_0-monomial.

Now we let $^-$ be the automorphism of U^- defined in Section 2.2. Elements that are invariant under $^-$ are said to be bar-invariant.

By results of Lusztig ([Lus93] Theorem 42.1.10, [Lus96], Proposition 8.2), there is a unique basis $\mathbf{B}$ of U^- with the following properties. Firstly, all elements of $\mathbf{B}$ are bar-invariant. Secondly, for any choice of reduced expression w_0 for the longest element in the Weyl group, and any element $X \in \mathbf{B}$ we have that $X = x + \sum \zeta_i x_i$, where x, x_i are w_0-monomials, $x \neq x_i$ for all i, and $\zeta_i \in q\mathbb{Z}[q]$. The basis $\mathbf{B}$ is called the canonical basis. If we work with a fixed reduced expression for the longest element in $W(\Phi)$, and write $X \in \mathbf{B}$ as above, then we say that x is the *principal monomial* of X.

Let $\mathcal{L}$ be the $\mathbb{Z}[q]$-lattice in U^- spanned by $\mathbf{B}$. Then $\mathcal{L}$ is also spanned by all w_0-monomials (where w_0 is a fixed reduced expression for the longest element in $W(\Phi)$). Now let $\widetilde{w}_0$ be a second reduced expression for the longest element in $W(\Phi)$. Let x be a w_0-monomial, and let X be the element of $\mathbf{B}$ with principal monomial x. Write X as a linear combination of $\widetilde{w}_0$-monomials, and let $\widetilde{x}$ be the principal monomial of that expression. Then we write $\widetilde{x} = R_{w_0}^{\widetilde{w}_0}(x)$. Note that $x = \widetilde{x} \bmod q\mathcal{L}$.

Now let $\mathcal{B}$ be the set of all w_0-monomials $\bmod q\mathcal{L}$. Then $\mathcal{B}$ is a basis of the $\mathbb{Z}$-module $\mathcal{L}/q\mathcal{L}$. Moreover, $\mathcal{B}$ is independent of the choice of w_0. Let $\alpha \in \Delta$, and let $\widetilde{w}_0$ be a reduced expression for

the longest element in $W(\Phi)$, starting with s_α. The Kashiwara operators $\widetilde{F}_\alpha : \mathcal{B} \to \mathcal{B}$ and $\widetilde{E}_\alpha : \mathcal{B} \to \mathcal{B} \cup \{0\}$ are defined as follows. Let $b \in \mathcal{B}$ and let x be the w_0-monomial such that $b = x \bmod q\mathcal{L}$. Set $\widetilde{x} = R^{\tilde{w}_0}_{w_0}(x)$. Then $\widetilde{x}'$ is the $\widetilde{w}_0$-monomial constructed from $\widetilde{x}$ by increasing its first exponent by 1 (the first exponent is the n_1 in (2.1)). Then $\widetilde{F}_\alpha(b) = R^{w_0}_{\tilde{w}_0}(\widetilde{x}') \bmod q\mathcal{L}$. For $\widetilde{E}_\alpha$ we let $\widetilde{x}'$ be the $\widetilde{w}_0$-monomial constructed from $\widetilde{x}$ by decreasing its first exponent by 1, if this exponent is ≥ 1. Then $\widetilde{E}_\alpha(b) = R^{w_0}_{\tilde{w}_0}(\widetilde{x}') \bmod q\mathcal{L}$. Furthermore, $\widetilde{E}_\alpha(b) = 0$ if the first exponent of $\widetilde{x}$ is 0. It can be shown that this definition does not depend on the choice of w_0, $\widetilde{w}_0$. Furthermore we have $\widetilde{F}_\alpha\widetilde{E}_\alpha(b) = b$, if $\widetilde{E}_\alpha(b) \neq 0$, and $\widetilde{E}_\alpha\widetilde{F}_\alpha(b) = b$ for all $b \in \mathcal{B}$.

Let $w_0 = s_{i_1} \cdots s_{i_t}$ be a fixed reduced expression for the longest element in $W(\Phi)$. For $b \in \mathcal{B}$ we define a sequence of elements $b_k \in \mathcal{B}$ for $0 \leq k \leq t$, and a sequence of integers n_k for $1 \leq k \leq t$ as follows. We set $b_0 = b$, and if b_{k-1} is defined we let n_k be maximal such that $\widetilde{E}^{n_k}_{\alpha_{i_k}}(b_{k-1}) \neq 0$. Also we set $b_k = \widetilde{E}^{n_k}_{\alpha_{i_k}}(b_{k-1})$. Then the sequence $(n_1, \ldots, n_t)$ is called the *string* of $b \in \mathcal{B}$ (relative to w_0). We note that $b = \widetilde{F}^{n_1}_{\alpha_{i_1}} \cdots \widetilde{F}^{n_t}_{\alpha_{i_t}}(1)$. The set of all strings parametrizes the elements of $\mathcal{B}$, and hence of $\mathbf{B}$.

Now let $V(\lambda)$ be a highest-weight module over $U_q(\mathfrak{g})$, with highest weight λ. Let v_λ be a fixed highest weight vector. Then $\mathbf{B}_\lambda = \{X \cdot v_\lambda \mid X \in \mathbf{B}\} \setminus \{0\}$ is a basis of $V(\lambda)$, called the *canonical basis*, or *crystal basis* of $V(\lambda)$. Let $\mathcal{L}(\lambda)$ be the $\mathbb{Z}[q]$-lattice in $V(\lambda)$ spanned by $\mathbf{B}_\lambda$. We let $\mathcal{B}(\lambda)$ be the set of all $x \cdot v_\lambda \bmod q\mathcal{L}(\lambda)$, where x runs through all w_0-monomials, such that $X \cdot v_\lambda \neq 0$, where $X \in \mathbf{B}$ is the element with principal monomial x. Then the Kashiwara operators are also viewed as maps $\mathcal{B}(\lambda) \to \mathcal{B}(\lambda) \cup \{0\}$, in the following way. Let $b = x \cdot v_\lambda \bmod q\mathcal{L}(\lambda)$ be an element of $\mathcal{B}(\lambda)$, and let $b' = x \bmod q\mathcal{L}$ be the corresponding element of $\mathcal{B}$. Let y be the w_0-monomial such that $\widetilde{F}_\alpha(b') = y \bmod q\mathcal{L}$. Then $\widetilde{F}_\alpha(b) = y \cdot v_\lambda \bmod q\mathcal{L}(\lambda)$. The description of $\widetilde{E}_\alpha$ is analogous. (In [Jan96], Chapter 9 a different definition is given; however, by [Jan96], Proposition 10.9, Lemma 10.13, the two definitions agree).

The set $\mathcal{B}(\lambda)$ has $\dim V(\lambda)$ elements. We let Γ be the coloured directed graph defined as follows. The points of Γ are the elements of $\mathcal{B}(\lambda)$, and there is an arrow with colour $\alpha \in \Delta$ connecting $b, b' \in \mathcal{B}$, if $\widetilde{F}_\alpha(b) = b'$. The graph Γ is called the *crystal graph* of $V(\lambda)$.

2.7 The path model

In this section we recall some basic facts on Littelmann's path model.

From Section 2.2 we recall that P denotes the weight lattice. Let $P_\mathbb{R}$ be the vector space over $\mathbb{R}$ spanned by P. Let Π be the set of all piecewise linear paths $\xi : [0,1] \to P_\mathbb{R}$, such that $\xi(0) = 0$. For $\alpha \in \Delta$ Littelmann defined operators $f_\alpha, e_\alpha : \Pi \to \Pi \cup \{0\}$. Let λ be a dominant weight and let ξ_λ be the path joining λ and the origin by a straight line. Let Π_λ be the set of all nonzero $f_{\alpha_{i_1}} \cdots f_{\alpha_{im}}(\xi_\lambda)$ for $m \geq 0$. Then $\xi(1) \in P$ for all $\xi \in \Pi_\lambda$. Let $\mu \in P$ be a weight, and let $V(\lambda)$ be the highest-weight module over $U_q(\mathfrak{g})$ of highest weight λ. A theorem of Littelmann states that the number of paths $\xi \in \Pi_\lambda$ such that $\xi(1) = \mu$ is equal to the dimension of the weight space of weight μ in $V(\lambda)$ ([Lit95], Theorem 9.1).

All paths appearing in Π_λ are so-called Lakshmibai-Seshadri paths (LS-paths for short). They are defined as follows. Let $\leq$ denote the Bruhat order on $W(\Phi)$. For $\mu, \nu \in W(\Phi) \cdot \lambda$ (the orbit of λ under the action of $W(\Phi)$), write $\mu \leq \nu$ if $\tau \leq \sigma$, where $\tau, \sigma \in W(\Phi)$ are the unique elements of minimal length such that $\tau(\lambda) = \mu$, $\sigma(\lambda) = \nu$. Now a rational path of shape λ is a pair $\pi = (\nu, a)$, where $\nu = (\nu_1, \ldots, \nu_s)$ is a sequence of elements of $W(\Phi) \cdot \lambda$, such that $\nu_i > \nu_{i+1}$ and $a = (a_0 = 0, a_1, \cdots, a_s = 1)$ is a sequence of rationals such that $a_i < a_{i+1}$. The path π corresponding to these sequences is given

by

$$\pi(t) = \sum_{j=1}^{r-1} (a_j - a_{j-1})\nu_j + \nu_r(t - a_{r-1})$$

for $a_{r-1} \leq t \leq a_r$. Now an LS-path of shape λ is a rational path satisfying a certain integrality condition (see [Lit94], [Lit95]). We note that the path $\xi_\lambda = ((\lambda),(0,1))$ joining the origin and λ by a straight line is an LS-path.

Now from [Lit94], [Lit95] we transcribe the following:

- Let π be an LS-path. Then $f_\alpha\pi$ is an LS-path or 0; and the same holds for $e_\alpha\pi$.
- The action of f_α, e_α can easily be described combinatorially (see [Lit94]).
- The endpoint of an LS-path is an integral weight.
- Let $\pi = (\nu, a)$ be an LS-path. Then by $\phi(\pi)$ we denote the unique element σ of $W(\Phi)$ of shortest length such that $\sigma(\lambda) = \nu_1$.

Let λ be a dominant weight. Then we define a labeled directed graph Γ as follows. The points of Γ are the paths in Π_λ. There is an edge with label $\alpha \in \Delta$ from π_1 to π_2 if $f_\alpha\pi_1 = \pi_2$. Now by [Kas96] this graph Γ is isomorphic to the crystal graph of the highest-weight module with highest weight λ. So the path model provides an efficient way of computing the crystal graph of a highest-weight module, without constructing the module first. Also we see that $f_{\alpha_{i_1}} \cdots f_{\alpha_{i_r}} \xi_\lambda = 0$ is equivalent to $\widetilde{F}_{\alpha_{i_1}} \cdots \widetilde{F}_{\alpha_{i_r}} v_\lambda = 0$, where $v_\lambda \in V(\lambda)$ is a highest weight vector (or rather the image of it in $\mathcal{L}(\lambda)/q\mathcal{L}(\lambda)$), and the $\widetilde{F}_{\alpha_k}$ are the Kashiwara operators on $\mathcal{B}(\lambda)$ (see Section 2.6).

2.8 Notes

I refer to [Hum90] for more information on Weyl groups, and to [Ste01] for an overview of algorithms for computing with weights, Weyl groups and their elements.

For general introductions into the theory of quantized enveloping algebras I refer to [Car98], [Jan96] (from where most of the material of this chapter is taken), [Lus92], [Lus93], [Ros91]. I refer to the papers by Littelmann ([Lit94], [Lit95], [Lit98]) for more information on the path model. The paper by Kashiwara ([Kas96]) contains a proof of the connection between path operators and Kashiwara operators.

Finally, I refer to [Gra01] (on computing with PBW-type bases), [Gra02] (computation of elements of the canonical basis) for an account of some of the algorithms used in QuaGroup.

Chapter 3

QuaGroup

In this chapter we describe the functionality provided by QuaGroup.

3.1 Global constants

3.1.1 QuantumField

◊ `QuantumField` (global variable)

This is the field $Q(q)$ of rational functions in q, over Q.

Example

```
gap> QuantumField;
QuantumField
```

3.1.2 _q

◊ `_q` (global variable)

This is an indeterminate; `QuantumField` is the field of rational functions in this indeterminate. The identifier `_q` is fixed once the package QuaGroup is loaded. The symbol `_q` is chosen (instead of `q`) in order to avoid potential name clashes. We note that `_q` is printed as `q`.

Example

```
gap> _q;
q
gap> _q in QuantumField;
true
```

3.2 Gaussian integers

3.2.1 GaussNumber

◊ `GaussNumber( n, par )` (operation)

This function computes for the integer `n` the Gaussian integer $[n]_{v=\texttt{par}}$ (cf. Section 2.1).

Example
```
gap> GaussNumber( 4, _q );
q^-3+q^-1+q+q^3
```

3.2.2 GaussianFactorial

◊ GaussianFactorial(n, par) (operation)

This function computes for the integer n the Gaussian factorial $[n]!_{v=\texttt{par}}$.

Example
```
gap> GaussianFactorial( 3, _q );
q^-3+2*q^-1+2*q+q^3
gap> GaussianFactorial( 3, _q^2 );
q^-6+2*q^-2+2*q^2+q^6
```

3.2.3 GaussianBinomial

◊ GaussianBinomial(n, k, par) (operation)

This function computes for two integers n and k the Gaussian binomial n choose k, where the parameter v is replaced by par.

Example
```
gap> GaussianBinomial( 5, 2, _q^2 );
q^-12+q^-8+2*q^-4+2+2*q^4+q^8+q^12
```

3.3 Roots and root systems

In this section we describe some functions for dealing with root systems. These functions supplement the ones already present in the GAP library.

3.3.1 RootSystem

◊ RootSystem(type, rank) (operation)
◊ RootSystem(list) (operation)

Here type is a capital letter between "A" and "G", and rank is a positive integer (≥ 1 if type="A", ≥ 2 if type="B", "C", ≥ 4 if type="D", $6,7,8$ if type="E", 4 if type="F", and 2 if type="G"). This function returns the root system of type type and rank rank. In the second form list is a list of types and ranks, e.g., ["B", 2, "F", 4, "D", 7].

The root system constructed by this function comes with he attributes PositiveRoots, NegativeRoots, SimpleSystem, CartanMatrix, BilinearFormMat. Here the attribute SimpleSystem contains a set of simple roots, written as unit vectors. PositiveRoots is a list of the positive roots, written as linear combinations of the simple roots, and likewise for NegativeRoots. CartanMatrix(R) is the Cartan matrix of the root system R, where the entry on position (i,j) is given by $\langle \alpha_i, \alpha_j^\vee \rangle$ where α_i is the i-th simple root. BilinearFormMat(R) is the matrix of the bilinear form, where the entry on position (i,j) is given by (α_i, α_j) (see Section 2.2).

WeylGroup(R) returns the Weyl group of the root system R. We refer to the GAP reference manual for an overview of the functions for Weyl groups in the GAP library. We mention the functions ConjugateDominantWeight(W, wt) (returns the dominant weight in the W-orbit of the weight wt), and WeylOrbitIterator(W, wt) (returns an iterator for the W-orbit containing the weight wt). We write weights as integral linear combinations of fundamental weights, so in GAP weights are represented by lists of integers (of length equal to the rank of the root system).

Also we mention the function PositiveRootsAsWeights(R) that returns the positive roots of R written as weights, i.e., as linear combinations of the fundamental weights.

Example

```
gap> R:=RootSystem( [ "B", 2, "F", 4, "E", 6 ] );
<root system of type B2 F4 E6>
gap> R:= RootSystem( "A", 2 );
<root system of type A2>
gap> PositiveRoots( R );
[ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ] ]
gap> BilinearFormMat( R );
[ [ 2, -1 ], [ -1, 2 ] ]
gap> W:= WeylGroup( R );
Group([ [ [ -1, 1 ], [ 0, 1 ] ], [ [ 1, 0 ], [ 1, -1 ] ] ])
gap> ConjugateDominantWeight( W, [-3,2] );
[ 2, 1 ]
gap> o:= WeylOrbitIterator( W, [-3,2] );
<iterator>
# Using the iterator we can loop over the orbit:
gap> NextIterator( o );
[ 2, 1 ]
gap> NextIterator( o );
[ -1, -2 ]
gap> PositiveRootsAsWeights( R );
[ [ 2, -1 ], [ -1, 2 ], [ 1, 1 ] ]
```

3.3.2 BilinearFormMatNF

◊ BilinearFormMatNF(R) (attribute)

This is the matrix of the "normalized" bilinear form. This means that all diagonal entries are even, and 2 is the minimum value occurring on the diagonal. If R is a root system constructed by RootSystem (3.3.1), then this is equal to BilinearFormMat(R).

3.3.3 PositiveRootsNF

◊ PositiveRootsNF(R) (attribute)

This is the list of positive roots of the root system R, written as linear combinations of the simple roots. This means that the simple roots are unit vectors. If R is a root system constructed by RootSystem (3.3.1), then this is equal to PositiveRoots(R).

One of the reasons for writing the positive roots like this is the following. Let a, b be two elements of PositiveRootsNF(R), and let B be the matrix of the bilinear form. Then a*(B*b) is the result of applying the bilinear form to a, b.

Example

```
gap> R:= RootSystem( SimpleLieAlgebra( "B", 2, Rationals ) );;
gap> PositiveRootsNF( R );
[ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ], [ 1, 2 ] ]
# We note that in this case PositiveRoots( R ) will give the positive roots in
# a different format.
```

3.3.4 SimpleSystemNF

◊ SimpleSystemNF(R) (attribute)

This is the list of simple roots of R, written as unit vectors (this means that they are elements of PositiveRootsNF(R)). If R is a root system constructed by RootSystem (3.3.1), then this is equal to SimpleSystem(R).

3.3.5 PositiveRootsInConvexOrder

◊ PositiveRootsInConvexOrder(R) (attribute)

This function returns the positive roots of the root system R, in the "convex" order. Let $w_0 = s_1 \cdots s_t$ be a reduced expression of the longest element in the Weyl group. Then the k-th element of the list returned by this function is $s_1 \cdots s_{k-1}(\alpha_k)$. (Where the reduced expression used is the one returned by LongestWeylWord(R).) If α, β and $\alpha+\beta$ are positive roots, then $\alpha+\beta$ occurs between α and β (whence the name convex order).

In the output all roots are written in "normal form", i.e., as elements of PositiveRootsNF(R).

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> PositiveRootsInConvexOrder( R );
[ [ 1, 0 ], [ 3, 1 ], [ 2, 1 ], [ 3, 2 ], [ 1, 1 ], [ 0, 1 ] ]
```

3.3.6 SimpleRootsAsWeights

◊ SimpleRootsAsWeights(R) (attribute)

Returns the simple roots of the root system R, written as linear combinations of the fundamental weights.

Example

```
gap> R:= RootSystem( "A", 2 );;
gap> SimpleRootsAsWeights( R );
[ [ 2, -1 ], [ -1, 2 ] ]
```

3.4 Weyl groups and their elements

Now we describe a few functions that deal with reduced words in the Weyl group of the root system R. These words are represented as lists of positive integers i, denoting the i-th simple reflection (which corresponds to the i-th element of SimpleSystem(R)). For example [3, 2, 1, 3, 1] represents the expression $s_3 s_2 s_1 s_3 s_1$.

3.4.1 ApplyWeylElement

◊ ApplyWeylElement(W, wt, wd) (operation)

Here wd is a (not necessarily reduced) word in the Weyl group W, and wt is a weight (written as integral linear combination of the simple weights). This function returns the result of applying wd to wt. For example, if wt=μ, and wd = [1, 2] then this function returns $s_1 s_2(\mu)$ (where s_i is the simple reflection corresponding to the i-th simple root).

Example

```
gap> W:= WeylGroup( RootSystem( "G", 2 ) ) ;;
gap> ApplyWeylElement( W, [ -3, 7 ], [ 1, 1, 2, 1, 2 ] );
[ 15, -11 ]
```

3.4.2 LengthOfWeylWord

◊ LengthOfWeylWord(W, wd) (operation)

Here wd is a word in the Weyl group W. This function returns the length of that word.

Example

```
gap> W:= WeylGroup( RootSystem( "F", 4 ) ) ;
<matrix group with 4 generators>
gap> LengthOfWeylWord( W, [ 1, 3, 2, 4, 2 ] );
3
```

3.4.3 LongestWeylWord

◊ LongestWeylWord(R) (attribute)

Here R is a root system. LongestWeylWord(R) returns the longest word in the Weyl group of R.

If this function is called for a root system R, a reduced expression for the longest element in the Weyl group is calculated (the one which is the smallest in the lexicographical ordering). However, if you would like to work with a different reduced expression, then it is possible to set it by SetLongestWeylWord(R, wd), where wd is a reduced expression of the longest element in the Weyl group. Note that you will have to do this before calling LongestWeylWord, or any function that may call LongestWeylWord (once the attribute is set, it will not be possible to change it). Note also that you must be sure that the word you give is in fact a reduced expression for the longest element in the Weyl group, as this is not checked (you can check this with LengthOfWeylWord (3.4.2)).

We note that virtually all algorithms for quantized enveloping algebras depend on the choice of reduced expression for the longest element in the Weyl group (as the PBW-type basis depends on this).

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> LongestWeylWord( R );
[ 1, 2, 1, 2, 1, 2 ]
```

3.4.4 ReducedWordIterator

◊ ReducedWordIterator(W, wd) (operation)

Here W is a Weyl group, and wd a reduced word. This function returns an iterator for the set of reduced words that represent the same element as wd. The elements are output in ascending lexicographical order.

Example

```
gap> R:= RootSystem( "F", 4 );;
gap> it:= ReducedWordIterator( WeylGroup(R), LongestWeylWord(R) );
<iterator>
gap> NextIterator( it );
[ 1, 2, 1, 3, 2, 1, 3, 2, 3, 4, 3, 2, 1, 3, 2, 3, 4, 3, 2, 1, 3, 2, 3, 4 ]
gap> k:= 1;;
gap> while not IsDoneIterator( it ) do
> k:= k+1; w:= NextIterator( it );
> od;
gap> k;
2144892
```

So there are 2144892 reduced expressions for the longest element in the Weyl group of type F_4.

3.4.5 ExchangeElement

◊ ExchangeElement(W, wd, ind) (operation)

Here W is a Weyl group, and wd is a *reduced* word in W, and ind is an index between 1 and the rank of the root system. Let v denote the word obtained from wd by adding ind at the end. This function *assumes* that the length of v is one less than the length of wd, and returns a reduced expression for v that is obtained from wd by deleting one entry. Nothing is guaranteed of the output if the length of v is bigger than the length of wd.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> wd:= LongestWeylWord( R );;
gap> ExchangeElement( WeylGroup(R), wd, 1 );
[ 2, 1, 2, 1, 2 ]
```

3.4.6 GetBraidRelations

◊ GetBraidRelations(W, wd1, wd2) (operation)

Here W is a Weyl group, and wd1, wd2 are two reduced words representing the same element in W. This function returns a list of braid relations that can be applied to wd1 to obtain wd2. Here a braid relation is represented as a list, with at the odd positions integers that represent positions in a word, and at the even positions the indices that are on those positions after applying the relation. For example, let wd be the word [1, 2, 1, 3, 2, 1] and let r = [3, 3, 4, 1] be a relation. Then the result of applying r to wd is [1, 2, 3, 1, 2, 1] (i.e., on the third position we put a 3, and on the fourth position a 1).

We note that the function does not check first whether wd1 and wd2 represent the same element in W. If this is not the case, then an error will occur during the execution of the function, or it will produce wrong output.

Example

```
gap> R:= RootSystem( "A", 3 );;
gap> wd1:= LongestWeylWord( R );
[ 1, 2, 1, 3, 2, 1 ]
gap> wd2:= [ 1, 3, 2, 1, 3, 2 ];;
gap> GetBraidRelations( WeylGroup(R), wd1, wd2 );
[ [ 3, 3, 4, 1 ], [ 4, 2, 5, 1, 6, 2 ], [ 2, 3, 3, 2, 4, 3 ],
  [ 4, 1, 5, 3 ] ]
```

3.4.7 LongWords

◊ LongWords(R) (attribute)

For a root system R this returns a list of triples (of length equal to the rank of R). Let t be the k-th triple occurring in this list. The first element of t is an expression for the longest element of the Weyl group, starting with k. The second element is a list of braid relations, moving this expression to the value of LongestWeylWord(R). The third element is a list of braid relations performing the reverse transformation.

Example

```
gap> R:= RootSystem( "A", 3 );;
gap> LongWords( R )[3];
[ [ 3, 1, 2, 1, 3, 2 ],
  [ [ 3, 3, 4, 1 ], [ 4, 2, 5, 1, 6, 2 ], [ 2, 3, 3, 2, 4, 3 ],
      [ 4, 1, 5, 3 ], [ 1, 3, 2, 1 ] ],
  [ [ 4, 3, 5, 1 ], [ 1, 1, 2, 3 ], [ 2, 2, 3, 3, 4, 2 ],
      [ 4, 1, 5, 2, 6, 1 ], [ 3, 1, 4, 3 ] ] ]
```

3.5 Quantized enveloping algebras

In QuaGroup we deal with two types of quantized enveloping algebra. First there are the quantized enveloping algebras defined over the field QuantumField (3.1.1). We say that these algebras are "generic" quantized enveloping algebras, in QuaGroup they have the category IsGenericQUEA. Secondly, we deal with the quantized enveloping algebras that are defined over a different field.

3.5.1 QuantizedUEA

◊ QuantizedUEA(R) (attribute)
◊ QuantizedUEA(R, F, v) (operation)
◊ QuantizedUEA(L) (attribute)
◊ QuantizedUEA(L, F, v) (operation)

In the first two forms R is a root system. With only R as input, the corresponding generic quantized enveloping algebra is constructed. It is stored as an attribute of R (so that constructing it twice for the same root system yields the same object). Also the root system is stored in the quantized enveloping algebra as the attribute RootSystem.

The attribute GeneratorsOfAlgebra contains the generators of a PBW-type basis (see Section 2.4), that are constructed relative to the reduced expression for the longest element in the Weyl group that is contained in LongestWeylWord(R). We refer to ObjByExtRep (3.5.2) for a description of the construction of elements of a quantized enveloping algebra.

The call QuantizedUEA(R, F, v) returns the quantized universal enveloping algebra with quantum parameter v, which must lie in the field F. In this case the elements of GeneratorsOfAlgebra are the images of the generators of the corresponding generic quantized enveloping algebra. This means that if v is a root of unity, then the generators will not generate the whole algebra, but rather a finite dimensional subalgebra (as for instance $E_i^k = 0$ for k large enough). It is possible to construct elements that do not lie in this finite dimensional subalgebra using ObjByExtRep (3.5.2).

In the last two cases L must be a semisimple Lie algebra. The two calls are short for QuantizedUEA(RootSystem(L)) and QuantizedUEA(RootSystem(L), F, v) respectively.

Example

```
# We construct the generic quantized enveloping algebra corresponding
# to the root system of type A2+G2:
gap> R:= RootSystem( [ "A", 2, "G", 2 ] );;
gap> U:= QuantizedUEA( R );
QuantumUEA( <root system of type A2 G2>, Qpar = q )
gap> RootSystem( U );
<root system of type A2 G2>
gap> g:= GeneratorsOfAlgebra( U );
[ F1, F2, F3, F4, F5, F6, F7, F8, F9, K1, K1+(q^-1-q)*[ K1 ; 1 ], K2,
  K2+(q^-1-q)*[ K2 ; 1 ], K3, K3+(q^-1-q)*[ K3 ; 1 ], K4,
  K4+(q^-3-q^3)*[ K4 ; 1 ], E1, E2, E3, E4, E5, E6, E7, E8, E9 ]
# These elements generate a PBW-type basis of U; the nine elements Fi,
# and the nine elements Ei correspond to the roots listed in convex order:
gap> PositiveRootsInConvexOrder( R );
[ [ 1, 0, 0, 0 ], [ 1, 1, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ],
  [ 0, 0, 3, 1 ], [ 0, 0, 2, 1 ], [ 0, 0, 3, 2 ], [ 0, 0, 1, 1 ],
  [ 0, 0, 0, 1 ] ]
# So, for example, F5 is an element of weight -[ 0, 0, 3, 1 ].
# We can also multiply elements; the result is written on the PBW-basis:
gap> g[17]*g[4];
(q^-6-1)*F4*[ K4 ; 1 ]+(q^-3)*F4*K4
# Now we construct a non-generic quantized enveloping algebra:
gap> R:= RootSystem( "A", 2 );;
gap> U:= QuantizedUEA( R, CF(3), E(3) );;
gap> g:= GeneratorsOfAlgebra( U );
[ F1, F2, F3, K1, K1+(-E(3)+E(3)^2)*[ K1 ; 1 ], K2,
  K2+(-E(3)+E(3)^2)*[ K2 ; 1 ], E1, E2, E3 ]
```

As can be seen in the example, every element of U is written as a linear combination of monomials in the PBW-generators; the generators of U^- come first, then the generators of U^0, and finally the generators of U^+.

3.5.2 ObjByExtRep

◊ ObjByExtRep(fam, list) (operation)

Here fam is the elements family of a quantized enveloping algebra U. Secondly, list is a list describing an element of U. We explain how this description works. First we describe an indexing system for the generators of U. Let R be the root system of U. Let t be the number of positive roots, and rank the rank of the root system. Then the generators of U are Fk, Ki (and its inverse), Ek, for k=1...t, i=1..rank. (See Section 2.4; for the construction of the Fk, Ek, the value of LongestWeylWord(R) is used.) Now the index of Fk is k, and the index of Ek is t+rank+k. Furthermore, elements of the algebra generated by the Ki, and its inverse, are written as linear combinations of products of "binomials", as in Section 2.5. The element

$$K_i^d \begin{bmatrix} K_i \\ s \end{bmatrix}$$

(where $d = 0, 1$), is indexed as [t+i, d] (what happens to the s is described later). So an index is either an integer, or a list of two integers.

A monomial is a list of indices, each followed by an exponent. First come the indices of the Fk, (1..t), then come the lists of the form [t+i, d], and finally the indices of the Ek. Each index is followed by an exponent. An index of the form [t+i, d] is followed by the s in the above formula.

The second argument of ObjByExtRep is a list of monomials followed by coefficients. This function returns the element of U described by this list.

Finally we remark that the element

$$K_i^d \begin{bmatrix} K_i \\ s \end{bmatrix}$$

is printed as Ki[Ki ; s] if d=1, and as [Ki ; s] if d=0.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> fam:= ElementsFamily( FamilyObj( U ) );;
gap> list:= [ [ 2, 3, [ 4, 0 ], 8, 6, 11 ], _q^2,    # monomial and coefficient
> [ 1, 7, 3, 5, [ 5, 1 ], 3, 8, 9 ], _q^-1 + _q^2 ]; # monomial and coefficient
[ [ 2, 3, [ 4, 0 ], 8, 6, 11 ], q^2, [ 1, 7, 3, 5, [ 5, 1 ], 3, 8, 9 ],
  q^-1+q^2 ]
gap> ObjByExtRep( fam, list );
(q^2)*F2^(3)*[ K1 ; 8 ]*E1^(11)+(q^-1+q^2)*F1^(7)*F3^(5)*K2[ K2 ; 3 ]*E3^(9)
```

3.5.3 ExtRepOfObj

◇ ExtRepOfObj(elm) (operation)

For the element elm of a quantized enveloping algebra, this function returns the list that defines elm (see ObjByExtRep (3.5.2)).

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> g:= GeneratorsOfAlgebra(U);
[ F1, F2, F3, K1, K1+(q^-1-q)*[ K1 ; 1 ], K2, K2+(q^-1-q)*[ K2 ; 1 ], E1,
  E2, E3 ]
gap> ExtRepOfObj( g[5] );
[ [ [ 4, 1 ], 0 ], 1, [ [ 4, 0 ], 1 ], q^-1-q ]
```

3.5.4 QuantumParameter

◊ QuantumParameter(U) (attribute)

Returns the quantum parameter used in the definition of U.

Example

```
gap> R:= RootSystem("A",2);;
gap> U0:= QuantizedUEA( R, CF(3), E(3) );;
gap> QuantumParameter( U0 );
E(3)
```

3.5.5 CanonicalMapping

◊ CanonicalMapping(U) (attribute)

Here U is a quantized enveloping algebra. Let U0 denote the corresponding "generic" quantized enveloping algebra. This function returns the mapping U0 --> U obtained by mapping q (which is the quantum parameter of U0) to the quantum parameter of U.

Example

```
gap> R:= RootSystem("A", 3 );;
gap> U:= QuantizedUEA( R, CF(5), E(5) );;
gap> f:= CanonicalMapping( U );
MappingByFunction( QuantumUEA( <root system of type A
3>, Qpar = q ), QuantumUEA( <root system of type A3>, Qpar =
E(5) ), function( u ) ... end )
gap> U0:= Source( f );
QuantumUEA( <root system of type A3>, Qpar = q )
gap> g:= GeneratorsOfAlgebra( U0 );;
gap> u:= g[18]*g[9]*g[6];
(q^2)*F6*K2*E6+(q)*K2*[ K3 ; 1 ]
gap> Image( f, u );
(E(5)^2)*F6*K2*E6+(E(5))*K2*[ K3 ; 1 ]
```

3.5.6 WriteQEAToFile

◊ WriteQEAToFile(U, file) (operation)

Here U is a quantized enveloping algebra, and file is a string containing the name of a file. This function writes some data to file, that allows ReadQEAFromFile (3.5.7) to recover it.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",3) );;
gap> WriteQEAToFile( U, "/home/wdg/A3" );
```

3.5.7 ReadQEAFromFile

◊ ReadQEAFromFile(file) (operation)

Here file is a string containing the name of a file, to which a quantized enveloping algebra has been written by WriteQEAToFile (3.5.6). This function recovers the quantized enveloping algebra.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",3) );;
gap> WriteQEAToFile( U, "/home/wdg/A3" );
gap> U0:= ReadQEAFromFile( "/home/wdg/A3" );
QuantumUEA( <root system of type A3>, Qpar = q )
```

3.6 Homomorphisms and automorphisms

Here we describe functions for creating homomorphisms and (anti)-automorphisms of a quantized enveloping algebra.

3.6.1 QEAHomomorphism

◊ `QEAHomomorphism( U, A, list )` (operation)

Here `U` is a generic quantized enveloping algebra (i.e., with quantum parameter `_q`), `A` is an algebra with one over `QuantumField`, and `list` is a list of `4*rank` elements of `A` (where `rank` is the rank of the root system of `U`). On the first rank positions there are the images of the F_α (where the α are simple roots, listed in the order in which they occur in `SimpleSystem( R )`). On the positions `rank+1...2*rank` are the images of the K_α. On the positions `2*rank+1...3*rank` are the images of the K_α^{-1}, and finally on the positions `3*rank+1...4*rank` occur the images of the E_α.

This function returns the homomorphism `U -> A`, defined by this data. In the example below we construct a homomorphism from one quantized enveloping algebra into another. Both are constructed relative to the same root system, but with different reduced expressions for the longest element of the Weyl group.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> SetLongestWeylWord( R, [1,2,1,2,1,2] );
gap> UR:= QuantizedUEA( R );;
gap> S:= RootSystem( "G", 2 );;
gap> SetLongestWeylWord( S, [2,1,2,1,2,1] );
gap> US:= QuantizedUEA( S );;
gap> gS:= GeneratorsOfAlgebra( US );
[ F1, F2, F3, F4, F5, F6, K1, K1+(q^-1-q)*[ K1 ; 1 ], K2,
  K2+(q^-3-q^3)*[ K2 ; 1 ], E1, E2, E3, E4, E5, E6 ]
gap> SimpleSystem( R );
[ [ 1, 0 ], [ 0, 1 ] ]
gap> PositiveRootsInConvexOrder( S );
[ [ 0, 1 ], [ 1, 1 ], [ 3, 2 ], [ 2, 1 ], [ 3, 1 ], [ 1, 0 ] ]
# We see that the simple roots of R occur on positions 6 and 1
# in the list PositiveRootsInConvexOrder( S ); This means that we
# get the following list of images of the homomorphism:
gap> imgs:= [ gS[6], gS[1],      # the images of the F_{\alpha}
> gS[7], gS[9],                  # the images of the K_{\alpha}
> gS[8], gS[10],                 # the images of the K_{\alpha}^{-1}
> gS[16], gS[11] ];              # the images of the E_{\alpha}
[ F6, F1, K1, K2, K1+(q^-1-q)*[ K1 ; 1 ], K2+(q^-3-q^3)*[ K2 ; 1 ], E6, E1
 ]
gap> h:= QEAHomomorphism( UR, US, imgs );
<homomorphism: QuantumUEA( <root system of type G
```

```
2>, Qpar = q ) -> QuantumUEA( <root system of type G2>, Qpar = q )>
gap> Image( h, GeneratorsOfAlgebra( UR )[3] );
(1-q^4-q^6+q^10)*F1*F6^(2)+(-q^2+q^6)*F2*F6+(q^4)*F4
```

3.6.2 QEAAutomorphism

◊ QEAAutomorphism(U, list) (operation)
◊ QEAAutomorphism(U, f) (operation)

In the first form U is a generic quantized enveloping algebra (i.e., with quantum parameter _q), and list is a list of 4*rank elements of U (where rank is the rank of the corresponding root system). On the first rank positions there are the images of the F_α (where the α are simple roots, listed in the order in which they occur in SimpleSystem(R)). On the positions rank+1...2*rank are the images of the K_α. On the positions 2*rank+1...3*rank are the images of the K_α^{-1}, and finally on the positions 3*rank+1...4*rank occur the images of the E_α.

In the second form U is a non-generic quantized enveloping algebra, and f is an automorphism of the corresponding generic quantized enveloping algebra. The corresponding automorphism of U is constructed. In this case f must not be the bar-automorphism of the corresponding generic quantized enveloping algebra (cf. BarAutomorphism (3.6.6)), as this automorphism doesn't work in the non-generic case.

The image of an element x under an automorphism f is computed by Image(f, x). Note that there is no function for calculating pre-images (in general this seems to be a very hard task). If you want the inverse of an automorphism, you have to construct it explicitly (e.g., by QEAAutomorphism(U, list), where list is a list of pre-images).

Below we construct the automorphism ω (cf. Section 2.2) of the quantized enveloping of type A_3, when the quantum parameter is _q, and when the quantum parameter is a fifth root of unity.

Example

```
# First we construct the quantized enveloping algebra:
gap> R:= RootSystem( "A", 3 );;
gap> U0:= QuantizedUEA( R );
QuantumUEA( <root system of type A3>, Qpar = q )
gap> g:= GeneratorsOfAlgebra( U0 );
[ F1, F2, F3, F4, F5, F6, K1, K1+(q^-1-q)*[ K1 ; 1 ], K2,
  K2+(q^-1-q)*[ K2 ; 1 ], K3, K3+(q^-1-q)*[ K3 ; 1 ], E1, E2, E3, E4, E5, E6 ]
# Now, for instance, we map F_{\alpha} to E_{\alpha}, where \alpha
# is a simple root. In order to find where those F_{\alpha}, E_{\alpha}
# are in the list of generators, we look at the list of positive roots
# in convex order:
gap> PositiveRootsInConvexOrder( R );
[ [ 1, 0, 0 ], [ 1, 1, 0 ], [ 0, 1, 0 ], [ 1, 1, 1 ], [ 0, 1, 1 ],
  [ 0, 0, 1 ] ]
# So the simple roots occur on positions 1, 3, 6. This means that we
# have the following list of images:
gap> imgs:= [ g[13], g[15], g[18], g[8], g[10], g[12], g[7], g[9], g[11],
> g[1], g[3], g[6] ];
[ E1, E3, E6, K1+(q^-1-q)*[ K1 ; 1 ], K2+(q^-1-q)*[ K2 ; 1 ],
  K3+(q^-1-q)*[ K3 ; 1 ], K1, K2, K3, F1, F3, F6 ]
gap> f:= QEAAutomorphism( U0, imgs );
<automorphism of QuantumUEA( <root system of type A3>, Qpar = q )>
gap> Image( f, g[2] );
```

```
(-q)*E2
# f induces an automorphism of any non-generic quantized enveloping
# algebra with the same root system R:
gap> U1:= QuantizedUEA( R, CF(5), E(5) );
QuantumUEA( <root system of type A3>, Qpar = E(5) )
gap> h:= QEAAutomorphism( U1, f );
<automorphism of QuantumUEA( <root system of type A3>, Qpar = E(5) )>
gap> Image( h, GeneratorsOfAlgebra(U1)[7] );
(-E(5)+E(5)^4)*[ K1 ; 1 ]+K1
```

3.6.3 QEAAntiAutomorphism

◊ QEAAntiAutomorphism(U, list) (operation)
◊ QEAAntiAutomorphism(U, f) (operation)

These are functions for constructing anti-automorphisms of quantized enveloping algebras. The same comments apply as for QEAAutomorphism (3.6.2).

3.6.4 AutomorphismOmega

◊ AutomorphismOmega(U) (attribute)

This is the automorphism ω (cf. Section 2.2).

Example

```
gap> R:= RootSystem( "A", 3 );;
gap> U:= QuantizedUEA( R, CF(5), E(5) );
QuantumUEA( <root system of type A3>, Qpar = E(5) )
gap> f:= AutomorphismOmega( U );
<automorphism of QuantumUEA( <root system of type A3>, Qpar = E(5) )>
```

3.6.5 AntiAutomorphismTau

◊ AntiAutomorphismTau( ) (attribute)

This is the anti-automorphism τ (cf. Section 2.2).

Example

```
gap> R:= RootSystem( "A", 3 );;
gap> U:= QuantizedUEA( R, CF(5), E(5) );
QuantumUEA( <root system of type A3>, Qpar = E(5) )
gap> t:= AntiAutomorphismTau( U );
<anti-automorphism of QuantumUEA( <root system of type A3>, Qpar = E(5) )>
```

3.6.6 BarAutomorphism

◊ BarAutomorphism(U) (attribute)

This is the automorphism $\bar{}$ defined in Section 2.2 Here U must be a generic quantized enveloping algebra.

Example
```
gap> U:= QuantizedUEA( RootSystem(["A",2,"B",2]) );;
gap> bar:= BarAutomorphism( U );
<automorphism of QuantumUEA( <root system of type A2 B2>, Qpar = q )>
gap> Image( bar, GeneratorsOfAlgebra( U )[5] );
(-q^-2+q^2)*F4*F7+F5
```

3.6.7 AutomorphismTalpha

◊ AutomorphismTalpha(U, ind) (operation)

This is the automorphism T_α (cf. Section 2.4), where α is the ind-th simple root.

Example
```
gap> U:= QuantizedUEA( RootSystem( "B", 3 ) );;
gap> f:=AutomorphismTalpha( U, 1 );
<automorphism of QuantumUEA( <root system of type B3>, Qpar = q )>
gap> a:= GeneratorsOfAlgebra( U )[3];
F3
gap>  Image( f, a );
F2
```

3.6.8 DiagramAutomorphism

◊ DiagramAutomorphism(U, perm) (operation)

This is the automorphism of U induced by a diagram automorphism of the underlying root system. The diagram automorphism is represented by the permutation perm, which is the permutation of the simple roots performed by the diagram automorphism.

In the example below we construct the diagram automorphism of the root system of type A_3, which is represented by the permutation (1,3).

Example
```
gap> R:= RootSystem( "A", 3 );;
gap> U:= QuantizedUEA( R );;
gap> f:= DiagramAutomorphism( U, (1,3) );
<automorphism of QuantumUEA( <root system of type A3>, Qpar = q )>
gap> g:= GeneratorsOfAlgebra( U );
[ F1, F2, F3, F4, F5, F6, K1, K1+(q^-1-q)*[ K1 ; 1 ], K2,
  K2+(q^-1-q)*[ K2 ; 1 ], K3, K3+(q^-1-q)*[ K3 ; 1 ], E1, E2, E3, E4, E5, E6
 ]
gap> Image( f, g[1] );
F6
```

3.6.9 *

◊ *(f, h) (operation)

We can compose automorphisms and anti-automorphisms using the infix * operator. The result of composing two automorphisms is an automorphism. The result of composing an automorphism and an anti-automorphism is an anti-automorphism. The result of composing two anti-automorphisms is an automorphism.

Example

```
gap> U:= QuantizedUEA( RootSystem( "B", 3 ) );;
gap> f:=AutomorphismTalpha( U, 1 );
<automorphism of QuantumUEA( <root system of type B3>, Qpar = q )>
gap> h:= AutomorphismOmega( U );
<automorphism of QuantumUEA( <root system of type B3>, Qpar = q )>
gap> f*h;
<automorphism of QuantumUEA( <root system of type B3>, Qpar = q )>
gap> t:= AntiAutomorphismTau( U );;
gap> T:= AutomorphismTalpha( U, 2 );;
gap> Tinv:= t*T*t;
<automorphism of QuantumUEA( <root system of type B3>, Qpar = q )>
# (The last call may take a little while.)
gap> x:= Image( T, GeneratorsOfAlgebra( U )[1] );
(1-q^4)*F1*F3+(-q^2)*F2
gap> Image( Tinv, x );
F1
```

According to [Jan96], 8.14(10), $\tau \circ T_\alpha \circ \tau$ is the inverse of T_α.

3.7 Hopf algebra structure

Here we describe functions for dealing with the Hopf algebra structure of a quantized enveloping algebra. This structure enables us to construct tensor products, and dual modules of modules over a quantized enveloping algebra. We refer to the next section (Section 3.8) for some functions for creating modules.

3.7.1 TensorPower

◊ `TensorPower( U, d )` (operation)

Here `U` is a quantized universal enveloping algebra, and `d` a non-negative integer. This function returns the associative algebra with underlying vector space the `d`-fold tensor product of `U` with itself. The product is defined component wise.

Example

```
gap> U:= QuantizedUEA( RootSystem( [ "B", 2 ] ) );;
gap> T:= TensorPower( U, 3 );
<algebra over QuantumField, with 36 generators>
gap> g:= GeneratorsOfAlgebra( T );;
gap> x:= g[1];
1*(1<x>1<x>F1)
gap> y:= g[30];
1*(E2<x>1<x>1)
gap> x*y;
1*(E2<x>1<x>F1)
```

3.7.2 UseTwistedHopfStructure

◊ `UseTwistedHopfStructure( U, f, finv )` (operation)

Here U is a quantized enveloping algebra, and f, finv two (anti-) automorphisms of U, where finv is the inverse of f. After calling this function the Hopf structure on U is used that is obtained from the "normal" Hopf structure (see Section 2.3) by twisting it with f.

A call to this function sets the attribute HopfStructureTwist, which is the list [f, finv].

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2), CF(5), E(5) );;
gap> t:= AntiAutomorphismTau( U );;
gap> UseTwistedHopfStructure( U, t, t );
```

3.7.3 ComultiplicationMap

◊ ComultiplicationMap(U, d) (operation)

This is a homomorphism from the quantized enveloping algebra U to the d-fold tensor power of U with itself. It is obtained by a repeated application of the comultiplication of U. So for d=2 we get the comultiplication of U.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2), CF(5), E(5) );;
gap> D:= ComultiplicationMap( U, 3 );
<Comultiplication of QuantumUEA( <root system of type A2>, Qpar =
E(5) ), degree 3>
gap> Image( D, GeneratorsOfAlgebra(U)[4] );
1*(K1<x>K1<x>K1)
```

3.7.4 AntipodeMap

◊ AntipodeMap(U) (attribute)

This is the antipode map of the quantized enveloping algebra U, which is constructed as an anti-automorphism of U.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2), CF(5), E(5) );;
gap> a:= AntipodeMap( U );
<anti-automorphism of QuantumUEA( <root system of type A2>, Qpar = E(5) )>
```

3.7.5 CounitMap

◊ CounitMap(U) (attribute)

This is the counit map of the quantized enveloping algebra U, which is constructed as a function from U to the ground field.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2), CF(5), E(5) );;
gap> co:= CounitMap( U );
function( u ) ... end
gap> x:= GeneratorsOfAlgebra( U )[4];
K1
gap> co( x );
1
```

3.8 Modules

Here we describe some functions for constructing left modules over quantized enveloping algebras. We refer to the GAP reference manual for an overview of basic functions for algebra modules, which are also applicable to the modules constructed by the functions described in this section. We mention `MatrixOfAction`, `DirectSumOfAlgebraModules`. The action of an element of the algebra on an element of the module is calculated by the infix operator `^`.

3.8.1 HighestWeightModule (for a quantized env. alg.)

◊ `HighestWeightModule( U, wt )` (operation)

Here `U` is a quantized universal enveloping algebra, and `wt` a dominant weight (i.e., a list of length equal to the rank of the root system, consisting of non-negative integers). This function returns a finite-dimensional highest-weight module of highest weight `wt` over `U`. If `U` is generic then this is the unique irreducible highest-weight module over `U`. Otherwise it is the Weyl module, cf. Section 2.5. In this last case the module is not necessarily irreducible.

Let `V` denote the module returned by this function. The first basis element of the attribute `Basis( V )` is a highest-weight vector; it is written as `1*v0`. Other basis elements are written as, for example, `F2*F9*v0`, which means that this vector is the result of letting the PBW-monomial `F2*F9` act on the highest-weight vector.

Example

```
gap> U:= QuantizedUEA( RootSystem( [ "A", 2, "G", 2 ] ) );;
gap> V:= HighestWeightModule( U, [ 0, 1, 0, 2 ] );
<231-dimensional left-module over QuantumUEA( <root system of type A2 G
2>, Qpar = q )>
gap> Basis( V )[1];
1*v0
gap> Basis(V)[23]+(_q^2+_q^-2)*Basis(V)[137];
F3*F5*v0+(q^-2+q^2)*F8^(6)*v0
# We compute the action of an element on a vector:
gap> gg:= GeneratorsOfAlgebra( U );;
gap> x:= gg[21]*gg[5];
F5*E4+(-q^-1)*F6*K3
gap> x^Basis(V)[1];
(-q^-1)*F6*v0
```

3.8.2 IrreducibleQuotient

◊ `IrreducibleQuotient( V )` (attribute)

Here `V` is a highest-weight module over a non-generic quantized enveloping algebra. This function returns the quotient of `V` by the maximal submodule not containing the highest weight vector. This is not necessarily equal to `V` if the quantum parameter is a root of 1.

Example

```
gap> R:= RootSystem( "A", 2 );;
gap> U:= QuantizedUEA( R, CF(3), E(3) );;
gap> V:= HighestWeightModule( U, [1,1] );
<8-dimensional left-module over QuantumUEA( <root system of type A2>, Qpar =
```

```
E(3) )>
gap> IrreducibleQuotient( V );
<7-dimensional left-module over QuantumUEA( <root system of type A2>, Qpar =
E(3) )>
```

3.8.3 HWModuleByTensorProduct

◊ HWModuleByTensorProduct(U, wt) (operation)

Here U must be a *generic* quantized enveloping algebra, and wt a dominant weight. This function returns the irreducible highest-weight module with highest weight wt. The algorithm uses tensor products (whence the name). On some inputs this algorithm is faster than the one use for HighestWeightModule:for a quantized env. alg. (3.8.1), on some inputs it is slower. I do not know any good heuristics.

The basis supplied with the module returned is the canonical basis.

Example

```
gap> U:= QuantizedUEA( RootSystem("G",2) );;
gap> V:= HWModuleByTensorProduct( U, [2,1] );
<189-dimensional left-module over QuantumUEA( <root system of type G
2>, Qpar = q )>
# (This is a case where this algorithm is a lot faster.)
```

3.8.4 DIYModule

◊ DIYModule(U, V, acts) (operation)

Here U is a generic quantized enveloping algebra, and V is a vector space over the field QuantumField. U acts on V and the action is described by the data in the list acts. acts is a list of lists, of length 4*l, where l is the rank of the root system. acts describes the actions of the generators $[F_1,\ldots,F_l,K_1,\ldots,K_l,K_1^-1,\ldots,K_l^-1,E_1,\ldots,E_l]$. (Here F_k is the generator F_{α_k}, where α_k is the k-th simple root, and likewise for E_k.) The action of each generator is described by a list of length dim V, giving the images of the basis elements of V. If an image is zero then it may be omitted: in that case there is a "hole" in the list. This function returns the U-module defined by the input.

Let R be a root system of type A_1, and U the corresponding quantized enveloping algebra (generated by F,K,K^-1,E). In the example below we construct the 2-dimensional U-module with basis vectors v_1,v_2, and U-action given by $Fv_1=v_2$, $Fv_2=0$, $Kv_1=qv_1$, $Kv_2=q^{-1}v_2$, $Ev_1=0$, $Ev_2=v_1$.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",1) );
QuantumUEA( <root system of type A1>, Qpar = q )
gap> V:= QuantumField^2;
( QuantumField^2 )
gap> v:= BasisVectors( Basis(V) );
[ [ 1, 0 ], [ 0, 1 ] ]
gap> acts:= [ [ v[2], 0*v[1] ], [ _q*v[1], _q^-1*v[2] ],
> [ _q^-1*v[1], _q*v[2] ], [ 0*v[1], v[1] ] ];;
gap> M:= DIYModule( U, V, acts );
<2-dimensional left-module over QuantumUEA( <root system of type A
1>, Qpar = q )>
```

3.8.5 TensorProductOfAlgebraModules

◊ TensorProductOfAlgebraModules(V, W) (operation)
◊ TensorProductOfAlgebraModules(V, W) (operation)

Here V and W are two modules over the same quantized enveloping algebra U. This function constructs the tensor product of V and W (as a U-module). For this the comultiplication map of U is used (see ComultiplicationMap (3.7.3)).

In the second form list is a list of U-modules. In that case the iterated tensor product is constructed.

Example

```
gap> U:= QuantizedUEA( RootSystem( [ "A", 2 ] ) );;
gap> V1:= HighestWeightModule( U, [ 1, 0 ] );;
gap> V2:= HighestWeightModule( U, [ 0, 1 ] );;
gap> TensorProductOfAlgebraModules( V1, V2 );
<9-dimensional left-module over QuantumUEA( <root system of type A2>, Qpar = q )>
```

3.8.6 HWModuleByGenerator

◊ HWModuleByGenerator(V, v, hw) (operation)

Here V is a module over a generic quantized enveloping algebra U, v is a highest-weight vector (i.e., all E_αv=0), of weight hw, which must be dominant. This function returns a highest-weight module over U isomorphic to the submodule of V generated by v.

Example

```
gap> U:= QuantizedUEA( RootSystem("B",2) );;
gap> W1:= HighestWeightModule( U, [1,0] );;
gap> W2:= HighestWeightModule( U, [0,1] );;
gap> T:= TensorProductOfAlgebraModules( W1, W2 );
<20-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = q )>
gap> HWModuleByGenerator( T, Basis(T)[1], [1,1] );
<16-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = q )>
```

3.8.7 InducedQEAModule

◊ InducedQEAModule(U, V) (operation)

Here U is a non-generic quantized enveloping algebra, and V a module over the corresponding generic quantized enveloping algebra. This function returns the U-module obtained from V by setting _q equal to the quantum parameter of U.

Example

```
gap> R:= RootSystem("B",2);;
gap> U:= QuantizedUEA( R );;
gap> U0:= QuantizedUEA( R, CF(3), E(3) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> W:= InducedQEAModule( U0, V );
<16-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = E(3) )>
```

```
# This module is isomorphic to the one obtained by
# HighestWeightModule( U0, [1,1] );
```

3.8.8 GenericModule

◊ GenericModule(W) (attribute)

For an induced module (see InducedQEAModule (3.8.7)) this function returns the corresponding module over the generic quantized enveloping algebra.

3.8.9 CanonicalMapping

◊ CanonicalMapping(W) (attribute)

Here W is an induced module. Let V be the corresponding generic module (GenericModule (3.8.8)). This function returns the map V --> W, that sets _q equal to the quantum parameter of the acting algebra of W.

Example

```
gap> R:= RootSystem("B",2);;
gap> U:= QuantizedUEA( R );;
gap> U0:= QuantizedUEA( R, CF(3), E(3) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> W:= InducedQEAModule( U0, V );;
gap> f:= CanonicalMapping( W );
MappingByFunction( <
16-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = q )>, <
16-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = E(3) )>, function( v ) ... end )
gap> Image( f, _q^2*Basis(V)[3] );
(E(3)^2)*e.3
```

3.8.10 U2Module

◊ U2Module(U, hw) (operation)

Here U must be a quantized enveloping algebra of type A_2. This function returns the highest-weight module over U of highest-weight hw (which must be dominant). This function is generally a lot faster than HighestWeightModule:for a quantized env. alg. (3.8.1).

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> A2Module( U, [4,7] );
<260-dimensional left-module over QuantumUEA( <root system of type A
2>, Qpar = q )>
```

3.8.11 MinusculeModule

◊ MinusculeModule(U, hw) (operation)

Here U must be a generic quantized enveloping algebra, and hw a minuscule dominant weight. This function returns the highest-weight module over U of highest-weight hw. This function is generally somewhat faster than HighestWeightModule:for a quantized env. alg. (3.8.1).

Example

```
gap> U:= QuantizedUEA( RootSystem("A",5) );;
gap> MinusculeModule( U, [0,0,1,0,0] );
<20-dimensional left-module over QuantumUEA( <root system of type A
5>, Qpar = q )>
```

3.8.12 DualAlgebraModule

◊ DualAlgebraModule(V) (attribute)

Here V is a finite-dimensional left module over a quantized enveloping algebra U. This function returns the dual space of V as an algebra module. For this the antipode map of U is used (see AntipodeMap (3.7.4)).

Let M denote the module returned by this function. Then M has as basis the dual basis with respect to Basis(V). An element of this basis is printed as F@v, where v is an element of Basis(V). This is the function which takes the value 1 on v and 0 on all other basis elements. A general element of M is a linear combination of these basis elements.

The elements of M can be viewed as functions which take arguments. However, internally the elements of M are represented as wrapped up functions. The function corresponding to an element m of M is obtained by ExtRepOfObj(m) (the result of which is printed in the same way as m, but is not equal to it).

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> M:= DualAlgebraModule( V );
<8-dimensional left-module over QuantumUEA( <root system of type A
2>, Qpar = q )>
gap> u:= GeneratorsOfAlgebra( U )[2];
F2
gap> vv:= BasisVectors( Basis( M ) );
[ (1)*F@1*v0, (1)*F@F1*v0, (1)*F@F3*v0, (1)*F@F1*F3*v0, (1)*F@F2*v0,
  (1)*F@F1*F2*v0, (1)*F@F2*F3*v0, (1)*F@F2^(2)*v0 ]
gap> u^vv[3];
<zero function>
# (The zero of the dual space is printed as <zero function>).
gap> u^vv[4];
(q^3-q^5)*F@1*v0
# We get the function corresponding to a vector in M by using ExtRepOfObj:
gap> f:= ExtRepOfObj( vv[1] );
(1)*F@1*v0
# We can calculate images of this function:
gap> List( Basis(V), v -> Image( f, v ) );
[ 1, 0, 0, 0, 0, 0, 0, 0 ]
```

3.8.13 TrivialAlgebraModule

◊ TrivialAlgebraModule(U) (attribute)

Returns the trivial module over the quantized enveloping algebra U. For this the counit map of U is used.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> V:= TrivialAlgebraModule( U );
<left-module over QuantumUEA( <root system of type A2>, Qpar = q )>
```

3.8.14 WeightsAndVectors

◊ WeightsAndVectors(V) (operation)

Here V is a left module over a quantized enveloping algebra. WeightsAndVectors(V) is a list of two lists; the first of these is a list of the weights of V, the second a list of corresponding weight vectors. These are again grouped in lists: if the multiplicity of a weight is m, then there are m weight vectors, forming a basis of the corresponding weight space.

Modules constructed by HighestWeightModule:for a quantized env. alg. (3.8.1) come with this attribute set. There is a method installed for computing WeightsAndVectors(V), for modules V over a generic quantized enveloping algebra, such that all basis vectors (i.e., all elements of Basis(V)) are weight vectors.

Example

```
gap> U:= QuantizedUEA( RootSystem( "A", 2 ) );;
gap> V:= HighestWeightModule( U, [ 1, 1 ] );;
gap> WeightsAndVectors( V );
[ [ [ 1, 1 ], [ -1, 2 ], [ 2, -1 ], [ 0, 0 ], [ -2, 1 ], [ 1, -2 ],
      [ -1, -1 ] ],
  [ [ 1*v0 ], [ F1*v0 ], [ F3*v0 ], [ F1*F3*v0, F2*v0 ], [ F1*F2*v0 ],
      [ F2*F3*v0 ], [ F2^(2)*v0 ] ] ]
```

3.8.15 HighestWeightsAndVectors

◊ HighestWeightsAndVectors(V) (attribute)

Is analogous to WeightsAndVectors (3.8.14); now only the highest weights are listed along with the corresponding highest-weight vectors.

There is a method installed for this using WeightsAndVectors (3.8.14); which means that it works if and only if WeightsAndVectors(V) works.

Example

```
gap> U:= QuantizedUEA( RootSystem( [ "A", 2 ] ) );;
gap> V:= HighestWeightModule( U, [ 1, 1 ] );;
gap> HighestWeightsAndVectors( V );
[ [ [ 1, 1 ] ], [ [ 1*v0 ] ] ]
```

3.8.16 RMatrix

◊ RMatrix(V) (attribute)

Here V is a module over the a quantized enveloping algebra U. This function returns the matrix of a linear map $\theta : V \otimes V \to V \otimes V$ that is a solution to the quantum Yang-Baxter equation. We have that $\theta \circ P$ is an isomorphism of U-modules, where $P : V \otimes V \to V \otimes V$ is the linear map such that $P(v \otimes w) = w \otimes v$. For more details we refer to [Jan96], Chapter 7.

This function works for modules for which WeightsAndVectors (3.8.14) works.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",1) );;
gap> V:= HighestWeightModule( U, [1] );;
gap> RMatrix( V );
[ [ 1, 0, 0, 0 ], [ 0, q, 1-q^2, 0 ], [ 0, 0, q, 0 ], [ 0, 0, 0, 1 ] ]
```

3.8.17 IsomorphismOfTensorModules

◊ IsomorphismOfTensorModules(V, W) (operation)

Here V, W are two modules over the same quantized enveloping algebra U. This function returns a linear map $\theta : V \otimes W \to W \otimes V$ that is an isomorphism of U-modules.

This function is only guaranteed to work correctly if the Hopf algebra structure is non-twisted (see UseTwistedHopfStructure (3.7.2)).

This function works for modules for which WeightsAndVectors (3.8.14) works.

Example

```
gap> U:= QuantizedUEA( RootSystem("B",2) );;
gap> V:= HighestWeightModule( U, [1,0] );;
gap> W:= HighestWeightModule( U, [0,1] );;
gap> h:= IsomorphismOfTensorModules( V, W );;
gap> VW:= Source( h );
<20-dimensional left-module over QuantumUEA( <root system of type B
2>, Qpar = q )>
gap> Image( h, Basis(VW)[13] );
q*(1*v0<x>F3*v0)+1-q^2*(F4*v0<x>F2*v0)+q^-1-q^3*(F3*v0<x>1*v0)
```

3.8.18 WriteModuleToFile

◊ WriteModuleToFile(V, file) (operation)

Here V is a module over a quantized enveloping algebra, and file is a string containing the name of a file. This function writes some data to file, that allows ReadModuleFromFile (3.8.19) to recover it.

We remark that this function currently is only implemented for generic quantized enveloping algebras.

3.8.19 ReadModuleFromFile

◊ ReadModuleFromFile(file) (operation)

Here `file` is a string containing the name of a file, to which a module over a quantized enveloping algebra has been written by `WriteModuleToFile` (3.8.18). This function recovers the module. More precisely: a new module is constructed that is isomorphic to the old one. In the process the algebra acting on the module is constructed anew (from data written to the file). This algebra can be accessed by `LeftActingAlgebra( V )`.

We remark that this function currently is only implemented for generic quantized enveloping algebras.

Example

```
gap> U:= QuantizedUEA( RootSystem("A",3) );;
gap> V:= HighestWeightModule( U, [1,1,1] );;
gap> WriteModuleToFile( V, "/home/wdg/A3mod" );
gap> W:= ReadModuleFromFile( "/home/wdg/A3mod" );
<64-dimensional left-module over QuantumUEA( <root system of type A
3>, Qpar = q )>
```

3.9 The path model

In this section we describe functions for dealing with the path model. We work only with LS-paths, which are represented by two lists, one of weights, and one of rationals (see Section 2.7).

3.9.1 DominantLSPath

◊ `DominantLSPath( R, wt )` (operation)

Here `R` is a root system, and `wt` a dominant weight in the weight lattice of `R`. This function returns the LS-path that is the line from the origin to `wt`.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> DominantLSPath( R, [1,3] );
<LS path of shape [ 1, 3 ] ending in [ 1, 3 ] >
```

3.9.2 Falpha (for an LS-path)

◊ `Falpha( path, ind )` (operation)

Is the result of applying the path operator $f_{\alpha_{ind}}$ to the LS-path `path` (where α_{ind} is the `ind`-th simple root).

The result is `fail` if $f_{\alpha_{ind}}$`(path)=0`.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> p:=DominantLSPath( R, [1,3] );;
gap> p1:=Falpha( p, 1 );
<LS path of shape [ 1, 3 ] ending in [ -1, 4 ] >
gap> Falpha( p1, 1 );
fail
```

3.9.3 Ealpha (for an LS-path)

◊ `Ealpha( path, ind )` (operation)

Is the result of applying the path operator $e_{\alpha_{\texttt{ind}}}$ to the LS-path `path` (where $\alpha_{\texttt{ind}}$ is the `ind`-th simple root).

The result is `fail` if $e_{\alpha_{\texttt{ind}}}$(`path`)=0.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> p:=DominantLSPath( R, [1,3] );;
gap> Ealpha( p, 2 );
fail
gap> p1:=Falpha( p, 1 );;
gap> Ealpha( p1, 1 );
<LS path of shape [ 1, 3 ] ending in [ 1, 3 ] >
```

3.9.4 LSSequence

◊ `LSSequence( path )` (attribute)

returns the two sequences (of weights and rational numbers) that define the LS-path path.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> p:=DominantLSPath( R, [1,3] );;
gap> p1:= Falpha( Falpha( p, 1 ), 2 );;
gap> LSSequence( p1 );
[ [ [ 11, -4 ], [ -1, 4 ] ], [ 0, 1/4, 1 ] ]
```

3.9.5 WeylWord

◊ `WeylWord( path )` (attribute)

Here `path` is an LS-path in the orbit (under the root operators) of a dominant LS-path ending in the dominant weight λ. This means that the first direction of path is of the form $w(\lambda)$ for some w in the Weyl group. This function returns a list $[i_1,\ldots,i_m]$ such that $w = s_{i_1}\cdots s_{i_m}$.

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> p:=DominantLSPath( R, [1,3] );;
gap> p1:= Falpha( Falpha( Falpha( p, 1 ), 2 ), 1 );;
gap> WeylWord( p1 );
[ 1, 2, 1 ]
```

3.9.6 EndWeight

◊ `EndWeight( path )` (attribute)

Here `path` is an LS-path; this function returns the weight that is the endpoint of path

Example

```
gap> R:= RootSystem( "G", 2 );;
gap> p:=DominantLSPath( R, [1,3] );;
gap> p1:= Falpha( Falpha( Falpha( p, 1 ), 2 ), 1 );;
gap> EndWeight( p1 );
[ 0, 3 ]
```

3.9.7 CrystalGraph (for root system and weight)

◊ `CrystalGraph( R, wt )` (function)

This function returns a record describing the crystal graph of the highest-weight module with highest weight `wt`, over the quantized enveloping algebra corresponding to `R`. It is computed using the path model. Therefore the points in the graph are LS-paths.

Denote the output by `r`; then `r.points` is the list of points of the graph. Furthermore, `r.edges` is a list of edges of the graph; this is a list of elements of the form `[ [ i, j ], u ]`. This means that there is an arrow from point `i` (i.e., the point on position `i` in `r.points`) to point `j`, with label `u`.

Example

```
gap> R:= RootSystem( "A", 2 );;
gap> CrystalGraph( R, [1,1] );
rec(
  points := [ <LS path of shape [ 1, 1 ] ending in [ 1, 1 ] >, <LS path of sha\
pe [ 1, 1 ] ending in [ -1, 2 ] >, <LS path of shape [ 1, 1 ] ending in
        [ 2, -1 ] >, <LS path of shape [ 1, 1 ] ending in [ 0, 0 ] >,
      <LS path of shape [ 1, 1 ] ending in [ 0, 0 ] >,
      <LS path of shape [ 1, 1 ] ending in [ 1, -2 ] >,
      <LS path of shape [ 1, 1 ] ending in [ -2, 1 ] >,
      <LS path of shape [ 1, 1 ] ending in [ -1, -1 ] > ],
  edges := [ [ [ 1, 2 ], 1 ], [ [ 1, 3 ], 2 ], [ [ 2, 4 ], 2 ],
      [ [ 3, 5 ], 1 ], [ [ 4, 6 ], 2 ], [ [ 5, 7 ], 1 ], [ [ 6, 8 ], 1 ],
      [ [ 7, 8 ], 2 ] ] )
```

3.10 Canonical bases

Here we describe functions for computing the canonical basis of the negative part of a quantized enveloping algebra, and of a module.

3.10.1 Falpha (for a PBW-monomial)

◊ `Falpha( x, ind )` (operation)

Here `x` is a PBW-monomial in U^- (i.e., a monomial in the F_α, where α runs over the positive roots). This function returns the result of applying the `ind`-th Kashiwara operator $\widetilde{F}_{\alpha_{\texttt{ind}}}$ to x (cf. Section 2.6).

Example

```
gap> U:= QuantizedUEA( RootSystem( "F", 4 ) );;
gap> x:= One( U );
1
```

```
gap> Falpha( Falpha( x, 3 ), 2 );
F3*F9
```

3.10.2 Ealpha (for a PBW-monomial)

◊ Ealpha(x, ind) (operation)

Here x is a PBW-monomial in U^- (i.e., a monomial in the F_α, where α runs over the positive roots). This function returns the result of applying the ind-th Kashiwara operator $\widetilde{E}_{\alpha_{\mathtt{ind}}}$ to x (cf. Section 2.6). The result is fail if $\widetilde{E}_{\alpha_{\mathtt{ind}}}$(x)=0.

Example

```
gap> U:= QuantizedUEA( RootSystem( "F", 4 ) );;
gap> Ealpha( One( U ), 2 );
fail
gap> g:= GeneratorsOfAlgebra( U );;
gap> x:= g[1]*g[4]*g[7]*g[17];
F1*F4*F7*F17
gap> Ealpha( x, 3 );
F1*F2*F7*F17
```

3.10.3 CanonicalBasis

◊ CanonicalBasis(U) (attribute)

Is the canonical basis of the quantized universal enveloping algebra U. When this is constructed nothing is computed. By using PBWElements (3.10.4), MonomialElements (3.10.5), Strings (3.10.6) information about elements of the canonical basis can be obtained.

Example

```
gap> U:= QuantizedUEA( RootSystem( "F", 4 ) );;
gap> B:= CanonicalBasis( U );
<canonical basis of QuantumUEA( <root system of type F4>, Qpar = q ) >
```

3.10.4 PBWElements

◊ PBWElements(B, rt) (operation)

Here B is the canonical basis of a quantized uea, and rt a list of non-negative integers representing an element of the root lattice (e.g., if the simple roots are α, β and rt = [3, 2], then rt represents $3\alpha+2\beta$).

It is possible to add the option lowrank, as follows PBWElements(B, rt :lowrank). In that case a somewhat different method will be used, that is significantly faster if the underlying root system has rank 2,3. It is about equally fast for ranks 4,5; and slower for ranks greater than 5.

Example

```
gap> U:= QuantizedUEA( RootSystem( "F", 4 ) );;
gap> B:= CanonicalBasis( U );;
gap> PBWElements( B, [1,2,1,0] );
[ F1*F3^(2)*F9, F1*F3*F7+(q^4)*F1*F3^(2)*F9, (q^4)*F1*F3^(2)*F9+F2*F3*F9,
```

```
    (q^2)*F1*F3*F7+(q^2+q^6)*F1*F3^(2)*F9+(q^2)*F2*F3*F9+F2*F7,
    (q^4)*F1*F3*F7+(q^8)*F1*F3^(2)*F9+(q^4)*F2*F3*F9+(q^2)*F2*F7+F3*F4 ]
gap> U:= QuantizedUEA( RootSystem("G",2) );;
gap> B:= CanonicalBasis( U );;
gap> PBWElements( B, [2,3] : lowrank );
[ F1^(2)*F6^(3), F1*F5*F6^(2)+(q^8+q^10)*F1^(2)*F6^(3),
  (q^2)*F1*F5*F6^(2)+(q^6+q^12)*F1^(2)*F6^(3)+F3*F6^(2),
  (q^8)*F1*F5*F6^(2)+(q^18)*F1^(2)*F6^(3)+(q^6)*F3*F6^(2)+F5^(2)*F6 ]
```

3.10.5 MonomialElements

◊ `MonomialElements( B, rt )` (operation)

This does the same as `PBWElements` (3.10.4), except that the elements are written as linear combinations of monomials in the generators F_α, where α runs through the simple roots.

We remark that this information is also computed "behind the scenes" when calling `PBWElements( B, rt )`. However, it is not computed if the option `lowrank` is present in the call to `PBWElements`.

Example

```
gap> U:= QuantizedUEA( RootSystem( "F", 4 ) );;
gap> B:= CanonicalBasis( U );;
gap> MonomialElements( B, [1,2,1,0] );
[ F1*F3^(2)*F9, F1*F3*F9*F3+(-1)*F1*F3^(2)*F9, F3^(2)*F1*F9, F3*F1*F9*F3,
  F3*F9*F3*F1+(-1)*F3^(2)*F1*F9 ]
```

3.10.6 Strings

◊ `Strings( B, rt )` (operation)

Here `B`, `rt` are the same as in `PBWElements` (3.10.4). This returns the list of strings corresponding to the elements of `B` of weight `rt` (cf. Section 2.6). For example, if on the k-th position of the list returned by this function we have `[ 1, 2, 2, 3 ]`, then the principal monomial of the k-th element of `PBWElements( B, rt )` is $\widetilde{F}_1^2\widetilde{F}_2^3(1)$ (where $\widetilde{F}_i$ is the i-th Kashiwara operator).

We remark that this information is also computed "behind the scenes" when calling `PBWElements( B, rt )`. However, it is not computed if the option `lowrank` is present in the call to `PBWElements`.

Example

```
gap> U:= QuantizedUEA( RootSystem( "F", 4 ) );;
gap> B:= CanonicalBasis( U );;
gap> Strings( B, [1,2,1,0] );
[ [ 1, 1, 2, 2, 3, 1 ], [ 1, 1, 2, 1, 3, 1, 2, 1 ], [ 2, 2, 1, 1, 3, 1 ],
  [ 2, 1, 1, 1, 3, 1, 2, 1 ], [ 2, 1, 3, 1, 2, 1, 1, 1 ] ]
gap> Falpha( Falpha( Falpha( Falpha( One(U), 3 ), 1 ), 2 ), 2 );
F2*F3*F9
gap> PBWElements( B, [1,2,1,0] )[3];
(q^4)*F1*F3^(2)*F9+F2*F3*F9
```

3.10.7 PrincipalMonomial

◊ PrincipalMonomial(u) (operation)

Here u is an element of the output of PBWElements (3.10.4). This function returns the unique monomial of u that has coefficient 1.

Example

```
gap> U:= QuantizedUEA( RootSystem("G",2) );;
gap> B:= CanonicalBasis( U );;
gap> p:= PBWElements( B, [4,4] : lowrank )[4];
(q^9)*F1^(2)*F3*F6^(3)+F1^(2)*F5^(2)*F6^(2)+(q^9+q^11+q^13)*F1^(3)*F5*F6^(
3)+(q^20+q^22+2*q^24+q^26+q^28)*F1^(4)*F6^(4)
gap> PrincipalMonomial( p );
F1^(2)*F5^(2)*F6^(2)
```

3.10.8 StringMonomial

◊ StringMonomial(u) (operation)

Here u is a monomial in the negative part of a quantized enveloping algebra, e.g., as output by PrincipalMonomial (3.10.7). This function computes the corresponding "string" (see Section 2.6). The strings are output in the same way as in 3.10.6.

Example

```
gap> U:= QuantizedUEA( RootSystem("G",2) );;
gap> B:= CanonicalBasis( U );;
gap> p:= PBWElements( B, [1,2] : lowrank )[2];;
gap> m:=PrincipalMonomial( p );
F5*F6
gap> StringMonomial( m );
[ 2, 2, 1, 1 ]
gap> Falpha( Falpha( Falpha( One(U), 1 ), 2 ), 2 );
F5*F6
```

3.10.9 Falpha (for a module element)

◊ Falpha(V, v, ind) (operation)

Here V is a module over a quantized enveloping algebra, v an element of it, and ind an index between 1 and the rank of the root system. The function returns the result of applying the ind-th Kashiwara operator $\widetilde{F}_{\mathtt{ind}}$ to v. Here the Kashiwara operators are different from the ones described in Section 2.6. We refer to [Jan96], 9.2 for the definition of the operators used here.

Example

```
gap> U:= QuantizedUEA( RootSystem("B",2) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> Falpha( V, Basis(V)[1], 1 );
F1*v0
```

3.10.10 Ealpha (for a module element)

◊ `Ealpha( V, v, ind )` (operation)

Here `V` is a module over a quantized enveloping algebra, `v` an element of it, and `ind` an index between 1 and the rank of the root system. The function returns the result of applying the `ind`-th Kashiwara operator $\widetilde{E}_{\texttt{ind}}$ to `v`. Here the Kashiwara operators are different from the ones described in Section 2.6. We refer to [Jan96], 9.2 for the definition of the operators used here.

Example

```
gap> U:= QuantizedUEA( RootSystem("B",2) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> v:= Falpha( V, Basis(V)[2], 2 );
(q^2)*F1*F4*v0+F2*v0
gap> Ealpha( V, v, 2 );
F1*v0
```

3.10.11 CrystalBasis

◊ `CrystalBasis( V )` (attribute)

Here `V` is a finite-dimensional left module over a quantized enveloping algebra. This function returns the canonical, or crystal basis of V (see Section 2.6).

This function only works for modules for which `WeightsAndVectors` (3.8.14) works.

Example

```
gap> U:= QuantizedUEA( RootSystem( "B", 2 ) );;
 gap> V:= HighestWeightModule( U, [1,1] );
 <16-dimensional left-module over QuantumUEA( <root system of type B2>, Qpar
= q )>
 gap>  CrystalBasis( V );
 Basis( <16-dimensional left-module over QuantumUEA( <root system of type B
 2>, Qpar = q )>, [ 1*v0, F1*v0, F4*v0, F1*F4*v0, (q^2)*F1*F4*v0+F2*v0, F2*F4*v0,
(q)*F2*F4*v0+F3*v0, (-q^-4)*F1*F2*v0, (-q^-1)*F1*F3*v0+(-q^-3)*F2^(2)*v0,
(-q^-2)*F2^(2)*v0, F3*F4*v0, (-q^-4)*F2*F3*v0+(-q^-2)*F2^(2)*F4*v0,
(-q^-2)*F2*F3*v0, (q^-4)*F2^(3)*v0, (-q^-1)*F3^(2)*v0, (q^-5)*F2^(2)*F3*v0 ] )
```

3.10.12 CrystalVectors

◊ `CrystalVectors( V )` (attribute)

Here `V` is a finite-dimensional left module over a quantized enveloping algebra. Let `C` be the crystal basis of `V` (i.e., output by `CrystalBasis` (3.10.11)). This function returns a list of cosets of the basis elements of `C` modulo qL, where L is the $\mathbb{Z}[q]$-lattice spanned by `C`.

The coset of a vector `v` is printed as `<v>`.

The crystal vectors are used to construct the point set of the crystal graph of `V` (`CrystalGraph:for a module` (3.10.15)).

This function only works for modules for which `WeightsAndVectors` (3.8.14) works.

Example

```
gap> U:= QuantizedUEA( RootSystem( "B", 2 ) );;
gap> V:= HighestWeightModule( U, [1,1] );
<16-dimensional left-module over QuantumUEA( <root system of type B
```

```
2>, Qpar = q )>
gap> CrystalVectors( V );
[ <1*v0>, <F1*v0>, <F4*v0>, <F2*v0>, <F1*F4*v0>, <F3*v0>,
  <(-q^-4)*F1*F2*v0>, <F2*F4*v0>, <F1*F3*v0>, <F3*F4*v0>,
  <(-q^-1)*F1*F3*v0+(-q^-3)*F2^(2)*v0>, <(-q^-4)*F2*F3*v0+(-q^-2)*F2^(2)*F
    4*v0>, <F2^(2)*F4*v0>, <(q^-4)*F2^(3)*v0>, <(-q^-1)*F3^(2)*v0>,
  <(q^-5)*F2^(2)*F3*v0> ]
```

3.10.13 Falpha (for a crystal vector)

◊ `Falpha( v, ind )` (operation)

Here `v` is a crystal vector, i.e., an element of `CrystalVectors( V )`, where `V` is a left module over a quantized enveloping algebra. This function returns the result of applying the `ind`-th Kashiwara operator $\widetilde{F}_{\alpha_{\mathtt{ind}}}$ to v. The result is `fail` if $\widetilde{F}_{\alpha_{\mathtt{ind}}}$`(v)=0`.

Example

```
gap> U:= QuantizedUEA( RootSystem( "B", 2 ) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> c:=CrystalVectors( V );;
gap> Falpha( c[2], 2 );
<F2*v0>
gap> Falpha( c[3], 2 );
fail
gap> Falpha( Falpha( Falpha( c[1], 1 ), 2 ), 1 );
fail
gap> p:= DominantLSPath( RootSystem( "B", 2 ), [1,1] );
<LS path of shape [ 1, 1 ] ending in [ 1, 1 ] >
gap> Falpha( Falpha( Falpha( p, 1 ), 2 ), 1 );
fail
```

The last part of this example is an illustration of the fact that the crystal graph of a highest-weight module can be obtained by the path method (see Section 2.7).

3.10.14 Ealpha (for a crystal vector)

◊ `Ealpha( v, ind )` (operation)

Here `v` is a crystal vector, i.e., an element of `CrystalVectors( V )`, where `V` is a left module over a quantized enveloping algebra. This function returns the result of applying the `ind`-th Kashiwara operator $\widetilde{E}_{\alpha_{\mathtt{ind}}}$ to v. The result is `fail` if $\widetilde{E}_{\alpha_{\mathtt{ind}}}$`(v)=0`.

Example

```
gap> U:= QuantizedUEA( RootSystem( "B", 2 ) );;
gap> V:= HighestWeightModule( U, [1,1] );;
gap> c:=CrystalVectors( V );;
gap> Ealpha( c[3], 1 );
fail
gap> Ealpha( c[3], 2 );
<1*v0>
```

3.10.15 CrystalGraph (for a module)

◊ CrystalGraph(V) (function)

Returns the crystal graph of the module V. The points of this graph are the cosets output by CrystalVectors (3.10.12). The edges work in the same way as in CrystalGraph:for root system and weight (3.9.7).

Example

```
gap> U:= QuantizedUEA( RootSystem("A",2) );;
gap> V1:= HighestWeightModule( U, [1,0] );;
gap> V2:= HighestWeightModule( U, [0,1] );;
gap> W:= TensorProductOfAlgebraModules( V1, V2 );;
gap> CrystalGraph( W );
rec(
  points := [ <1*(1*v0<x>1*v0)>, <1*(F1*v0<x>1*v0)>, <1*(1*v0<x>F3*v0)>,
      <1*(1*v0<x>F2*v0)+q^-1*(F2*v0<x>1*v0)>,
      <-q^-1*(1*v0<x>F2*v0)+q^-1*(F1*v0<x>F3*v0)>, <1*(F2*v0<x>F3*v0)>,
      <-q^-1*(F1*v0<x>F2*v0)>, <-q^-1*(F2*v0<x>F2*v0)>,
      <-q^-3*(1*v0<x>F2*v0)+-q^-1*(F1*v0<x>F3*v0)+1*(F2*v0<x>1*v0)> ],
  edges := [ [ [ 1, 2 ], 1 ], [ [ 1, 3 ], 2 ], [ [ 2, 4 ], 2 ],
      [ [ 3, 5 ], 1 ], [ [ 4, 6 ], 2 ], [ [ 5, 7 ], 1 ], [ [ 6, 8 ], 1 ],
      [ [ 7, 8 ], 2 ] ] )
```

3.11 Universal enveloping algebras

Here we describe functions for connecting a quantized enveloping algebra to the corresponding universal enveloping algebra.

3.11.1 UEA

◊ UEA(L) (attribute)

This function returns the universal enveloping algebra u of the semisimple Lie algebra L. The generators of u are the generators of a Kostant lattice in the universal enveloping algebra (these generators are obtained from L by LatticeGeneratorsInUEA(L), see the GAP reference manual).

Example

```
gap> L:= SimpleLieAlgebra( "B", 2, Rationals );
<Lie algebra of dimension 10 over Rationals>
gap> u:= UEA( L );
<algebra over Rationals, with 10 generators>
gap> g:= GeneratorsOfAlgebra( u );
[ y1, y2, y3, y4, x1, x2, x3, x4, ( h9/1 ), ( h10/1 ) ]
```

3.11.2 UnderlyingLieAlgebra

◊ UnderlyingLieAlgebra(u) (attribute)

For a universal enveloping algebra u constructed by UEA (3.11.1), this returns the corresponding semisimple Lie algebra

Example

```
gap> L:= SimpleLieAlgebra( "B", 2, Rationals );;
gap> u:= UEA( L );;
gap> UnderlyingLieAlgebra( u );
<Lie algebra of dimension 10 over Rationals>
```

3.11.3 HighestWeightModule (for a universal env. alg)

◊ HighestWeightModule(u, hw) (operation)

For a universal enveloping algebra u constructed by UEA (3.11.1), this returns the irreducible highest weight module over u with highest weight hw, which must be dominant. This module is the same as the corresponding highest weight module over the semisimple Lie algebra, but in this case the enveloping algebra u acts.

Example

```
gap> L:= SimpleLieAlgebra( "B", 2, Rationals );;
gap> u:= UEA( L );;
gap> HighestWeightModule( u, [2,3] );
<140-dimensional left-module over <algebra over Rationals, with
10 generators>>
```

3.11.4 QUEAToUEAMap

◊ QUEAToUEAMap(L) (attribute)

Here L is a semisimple Lie algebra. Set u := UEA(L), and U := QuantizedUEA(L) (so u, U are the universal enveloping algebra, and "generic" quantized enveloping algebra of L respectively). Then QUEAToUEAMap(L) returns the algebra homomorphism from U to u obtained by mapping q to 1, a generator Fi, corresponding to a simple root to the generator yi (corresponding to the same simple root), and likewise for Ei and xi. This means that Ki is mapped to one, and [Ki : s] to hi choose s.

The canonical basis of U is mapped to the canonical basis of u.

Example

```
gap> L:= SimpleLieAlgebra( "B", 2, Rationals );;
gap> f:= QUEAToUEAMap( L );
<mapping: QuantumUEA( <root system of rank
2>, Qpar = q ) -> Algebra( Rationals, [ y1, y2, y3, y4, x1, x2, x3, x4,
  ( h9/1 ), ( h10/1 ) ] ) >
gap> U:= Source( f );
QuantumUEA( <root system of rank 2>, Qpar = q )
gap> u:= Range( f );
<algebra over Rationals, with 10 generators>
gap> B:= CanonicalBasis( U );;
gap> p:= PBWElements( B, [1,2] );
[ F1*F4^(2), (q+q^3)*F1*F4^(2)+F2*F4, (q^4)*F1*F4^(2)+(q)*F2*F4+F3 ]
gap> pu:= List( p, x -> Image( f, x ) );
[ y1*y2^(2), 2*y1*y2^(2)+y2*y3-2*y4, y1*y2^(2)+y2*y3-1*y4 ]
gap> V:= HighestWeightModule( u, [2,1] );
<40-dimensional left-module over <algebra over Rationals, with
10 generators>>
```

```
gap> List( pu, x -> xˆBasis(V)[1] );
[ 0*v0, y2*y3*v0+-2*y4*v0, y2*y3*v0+-1*y4*v0 ]
# Which gives us a piece of the canonical basis of V.
```

References

[Car98] R. W. Carter. Representations of simple Lie algebras: modern variations on a classical theme. In *Algebraic groups and their representations (Cambridge, 1997)*, pages 151–173. Kluwer Acad. Publ., Dordrecht, 1998. 13

[Com06] Editorial Committee. A note on the paper: "A survey of the work of George Lusztig" by R. W. Carter [Nagoya Math. J. **182** (2006), 1–45; 2235338]. *Nagoya Math. J.*, 183:i–ii, 2006. 11

[Gra01] W. A. de Graaf. Computing with quantized enveloping algebras: PBW-type bases, highest-weight modules, *R*-matrices. *J. Symbolic Comput.*, 32(5):475–490, 2001. 13

[Gra02] W. A. de Graaf. Constructing canonical bases of quantized enveloping algebras. *Experimental Mathematics*, 11(2):161–170, 2002. 13

[Hum90] J. E. Humphreys. *Reflection groups and Coxeter groups*. Cambridge University Press, Cambridge, 1990. 13

[Jan96] J. C. Jantzen. *Lectures on Quantum Groups*, volume 6 of *Graduate Studies in Mathematics*. American Mathematical Society, 1996. 10, 12, 13, 28, 36, 42, 43

[Kas96] M. Kashiwara. Similarity of crystal bases. In *Lie algebras and their representations (Seoul, 1995)*, pages 177–186. Amer. Math. Soc., Providence, RI, 1996. 13

[Lit94] P. Littelmann. A Littlewood-Richardson rule for symmetrizable Kac-Moody algebras. *Invent. Math.*, 116(1-3):329–346, 1994. 13

[Lit95] P. Littelmann. Paths and root operators in representation theory. *Ann. of Math. (2)*, 142(3):499–525, 1995. 12, 13

[Lit98] P. Littelmann. Cones, crystals, and patterns. *Transform. Groups*, 3(2):145–179, 1998. 13

[LN01] F. Lübeck and M. Neunhöffer. *GAPDoc, a GAP documentation meta-package*, 2001. 6

[Lus90] G. Lusztig. Quantum groups at roots of 1. *Geom. Dedicata*, 35(1-3):89–113, 1990. 11

[Lus92] G. Lusztig. Introduction to quantized enveloping algebras. In *New developments in Lie theory and their applications (Córdoba, 1989)*, pages 49–65. Birkhäuser Boston, Boston, MA, 1992. 13

[Lus93] G. Lusztig. *Introduction to quantum groups*. Birkhäuser Boston Inc., Boston, MA, 1993. 11, 13

[Lus96] G. Lusztig. Braid group action and canonical bases. *Adv. Math.*, 122(2):237–261, 1996. 11

[Lus0a] G. Lusztig. Canonical bases arising from quantized enveloping algebras. *J. Amer. Math. Soc.*, 3(2):447–498, 1990a. 11

[Ros91] M. Rosso. Représentations des groupes quantiques. *Astérisque*, (201-203):Exp. No. 744, 443–483 (1992), 1991. Séminaire Bourbaki, Vol. 1990/91. 13

[Ste01] J. R. Stembridge. Computational aspects of root systems, Coxeter groups, and Weyl characters. In *Interaction of combinatorics and representation theory*, volume 11 of *MSJ Mem.*, pages 1–38. Math. Soc. Japan, Tokyo, 2001. 13

Index