RMatrix package

A subpackage for QuantumGroups v2.
Version 2.0, June 22, 2006, Scott Morrison

Introduction

This package produces universal R-matrices, and their actions on representations.

Implementation

Start of package

Specify package dependencies:

```
BeginPackage["QuantumGroups`RMatrix`", {"QuantumGroups`",
   "QuantumGroups`Utilities`MatrixWrapper`", "QuantumGroups`Utilities`Debugging`",
   "QuantumGroups`RootSystems`", "QuantumGroups`Algebra`",
   "QuantumGroups`WeylGroups`", "QuantumGroups`Representations`",
   "QuantumGroups`QuantumRoots`", "QuantumGroups`MatrixPresentations`"}];
```

Usage messages

```
RMatrix::usage = "";
```

```
CheckRMatrixOppositeCommutes::usage = "";
```

Internals

```
Begin["`Private`"];
```

```
q = Global`q;
```

```
PartialRMatrix[Γ_][n_] :=
 PartialRMatrix[Γ][n] = Module[{p = Length[QuantumPositiveRoots[Γ]], iterators,
    r, d = CartanFactors[Γ], i = LongestWordDecomposition[Γ], l, t, rmatrix},
   DebugPrintHeld["Calculating ", PartialRMatrix[Γ][n]];
   l = QuantumRootHeight[Γ] /@ QuantumPositiveRoots[Γ];
   iterators = Table[{t[r], 0, (n - Sum[l[[k]] t[k], {k, r + 1, p}])/l[[r]]}, {r, p, 2, -1}] ~
     Join ~ {With[{t1 = (n - Sum[t[k] l[[k]], {k, 2, p}])/l[[1]]}, {t[1], t1, t1}]};
   rmatrix = Sum[If[p > 1, NonCommutativeMultiply, Times] @@
      Table[(q^d[[i[[r]]]])^(1/2 t[r] (t[r]+1)) (1 - q^(-2 d[[i[[r]]]]))^t[r] / qFactorial[t[r]][q^d[[i[[r]]]]]
        NonCommutativePower[SuperPlus[X_{Γ,r}], t[r]] ⊗ NonCommutativePower[
          SuperMinus[X_{Γ,r}], t[r]], {r, 1, p}], Evaluate[Sequence @@ iterators]];
   DebugPrintHeld["Finished calculating ", PartialRMatrix[Γ][n]];
   rmatrix
  ]
```

```
RMatrixAdjunct[Γ_, V1_, V2_, λ_] := Module[{partialWeightMultiplicities, exponents, d},
  partialWeightMultiplicities =
   QuantumGroups`MatrixPresentations`Private`WeightMultiplicityComponents[
    Γ, V1, V2, λ];
  exponents = KillingForm[Γ][λ - #, #] & /@ Weights[Γ, V2];
  d = Flatten[
    Table[#[[1]], {#[[2]]}] & /@ Transpose[{q^exponents, partialWeightMultiplicities}]];
  Matrix[DiagonalMatrix[d]]
 ]
```

```
PartialRMatrixPresentation[Γ_, n_, V_, W_, β_, λ_] :=
 PartialRMatrixPresentation[Γ, n, V, W, β, λ] =
  FastMatrixPresentation[Γ][PartialRMatrix[Γ][n]][V⊗W, β, λ]
```

```
CarefulFastMatrixPresentation[Γ_][X_][V_, β_, λ_] := Module[{told, tnew, rold, rnew},
  {tnew, rnew} = AbsoluteTiming[FastMatrixPresentation[Γ][X][V, β, λ]];
  {told, rold} = AbsoluteTiming[MatrixPresentation[Γ][X][V, β, λ]];
  If[rold =!= rnew, Print["Achtung, FastMatrixPresentation failed."]];
  DebugPrint["FastMatrixPresentation timing: ", {told, tnew}];
  rnew
 ]
```

No need to do these two fast, they're easy anyway:

```
FastMatrixPresentation[Γ_][1 ⊗ 1][V_ ⊗ W_, β_, λ_] :=
 MatrixPresentation[Γ][1 ⊗ 1][V ⊗ W, β, λ]
FastMatrixPresentation[Γ_][SuperPlus[X_{Γ_,r_}] ⊗ SuperMinus[X_{Γ_,r_}]][V_ ⊗ W_, β_, λ_] :=
 MatrixPresentation[Γ][SuperPlus[X_{Γ,r}] ⊗ SuperMinus[X_{Γ,r}]][V ⊗ W, β, λ]
```

```
FastMatrixPresentation[Γ_][(X : (NonCommutativeMultiply[(SuperPlus[X_Γ_,_]) ..])) ⊗
    (Y : (NonCommutativeMultiply[(SuperMinus[X_Γ_,_]) ..]))][V_ ⊗ W_, β_, λ_] :=
 Module[{result},
  If[WeightMultiplicity[Γ, V ⊗ W, λ + OperatorWeight[Γ][X]] == 0,
   Return[ZeroesMatrix[WeightMultiplicity[Γ, V ⊗ W, λ + OperatorWeight[Γ][X ⊗ Y]],
     WeightMultiplicity[Γ, V ⊗ W, λ]]]];
  If[WeightMultiplicity[Γ, V ⊗ W, λ + OperatorWeight[Γ][Y]] == 0,
   Return[ZeroesMatrix[WeightMultiplicity[Γ, V ⊗ W, λ + OperatorWeight[Γ][X ⊗ Y]],
     WeightMultiplicity[Γ, V ⊗ W, λ]]]];
  result = Simplify[MatrixPresentation[Γ][X ⊗ Y][V ⊗ W, β, λ]];
  Return[result]
 ]
```

```
FastMatrixPresentation[Γ_][A_Plus][V_, β_, λ_] :=
 FastMatrixPresentation[Γ][#][V, β, λ] & /@ A
```

```
FastMatrixPresentation[Γ_][α_?qNumberQ A_][V_, β_, λ_] :=
 α FastMatrixPresentation[Γ][A][V, β, λ]
```

```
FastMatrixPresentation[Γ_][X_][V_, β_, λ_] :=
 (DebugPrint["FastMatrixPresentation degrading to MatrixPresentation."];
  MatrixPresentation[Γ][X][V, β, λ])
```

```
RMatrix[Γ_, V1_, V2_, β_, λ_] /; MemberQ[Weights[Γ, V1 ⊗ V2], λ] :=
 Module[{n = -1, w, m, data},
  w = Weights[Γ, V1 ⊗ V2];
  m = Length[w];
  data = Simplify[Inner[Dot,
     Table[RMatrixAdjunct[Γ, V1, V2, w[[i]]], {i, 1, m}], FixedPoint[(n++;
        # + Table[PartialRMatrixPresentation[Γ, n, V1, V2, β, w[[i]]], {i, 1, m}]) &, 0],
     List]
    ];
  Table[RMatrix[Γ, V1, V2, β, w[[i]]] = data[[i]], {i, 1, m}];
  RMatrix[Γ, V1, V2, β, λ]
 ]
RMatrix[Γ_, V1_, V2_, β_, λ_] /; ! MemberQ[Weights[Γ, V1 ⊗ V2], λ] := Matrix[0, 0]
```

```
CheckRMatrixOppositeCommutes[Γ_, Z_][V1_, V2_, β_, λ_] :=
 With[{R1 = RMatrix[Γ, V1, V2, β, λ],
   R2 = RMatrix[Γ, V1, V2, β, λ + OperatorWeight[Γ][Δ[Z]]]},
  ZeroMatrixQ[Simplify[MatrixPresentation[Γ][Δop[Z]][V1 ⊗ V2, β, λ] -
     R2.MatrixPresentation[Γ][Δ[Z]][V1 ⊗ V2, β, λ].Inverse[R1]]
  ]
 ]
```

```
CheckRMatrixOppositeCommutes[Γ_][V1_, V2_, β_, λ_] :=
 And @@ (CheckRMatrixOppositeCommutes[Γ, #][V1, V2, β, λ] & /@ PositiveGenerators[Γ])
```

```
CheckRMatrixOppositeCommutes[Γ_][V1_, V2_, β_] :=
 And @@ (CheckRMatrixOppositeCommutes[Γ][V1, V2, β, #] & /@ Weights[Γ, V1 ⊗ V2])
```

```
End[];
```

End of package

```
EndPackage[];
```