BraidAction package

A subpackage for QuantumGroups v2.
Version 2.0, June 8, 2006, Scott Morrison

Introduction

This package defines the action of braid groups on the generators of a quantum group.

Implementation

Start of package

Specify package dependencies:

```
BeginPackage["QuantumGroups`BraidAction`",
  {"QuantumGroups`", "QuantumGroups`Utilities`Debugging`", "QuantumGroups`RootSystems`",
   "QuantumGroups`Algebra`", "QuantumGroups`Representations`"}];
```

Usage messages

```
T;
```

```
BraidAction::usage =
  "BraidAction[Γ][{T_i,T_j,...}, Z] computes the action of T_iT_j... on Z.";
```

```
BraidRelations::usage = "BraidRelations[Γ] returns
    the braid relations for the braid group associated to Γ.";
CheckBraidRelations::usage = "CheckBraidRelations[Γ] checks that the
    action specified by BraidAction[Γ] satisfies the relations
    returned by BraidRelations[Γ] on the generators of Γ.";
```

Internals

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

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

```
ExpandReducedPowers[Γ_][F_] := F /.
  {ReducedPower[X_{i_}^+, n_] :→
    With[{d = CartanFactors[Γ][[i]]}, NonCommutativePower[X_i^+, n] / qFactorial[n][q^d]],
   ReducedPower[X_{i_}^-, n_] :→ With[{d = CartanFactors[Γ][[i]]},
     NonCommutativePower[X_i^-, n] / qFactorial[n][q^d]]} /. OrderingRules[Γ]
```

```
BraidAction[Γ_][{word___}, 0] := 0
```

```
BraidAction[Γ_][{T_{i_}}, X_{j_}^+] := BraidAction[Γ][{T_i}, X_j^+] =
  If[i == j, -X_i^- ** K_i,
   With[{a = CartanMatrix[Γ][[i, j]], d = CartanFactors[Γ][[i]]}, ExpandReducedPowers[Γ][
     Sum_{r=0}^{-a} (-1)^(r-a) q^(-d r) ReducedPower[X_i^+, -a - r] ** X_j^+ ** ReducedPower[X_i^+, r]]]]
```

```
BraidAction[Γ_][{T_{i_}}, X_{j_}^-] := BraidAction[Γ][{T_i}, X_j^-] =
  If[i == j, -K_i^-1 ** X_i^+,
   With[{a = CartanMatrix[Γ][[i, j]], d = CartanFactors[Γ][[i]]}, ExpandReducedPowers[Γ][
     Sum_{r=0}^{-a} (-1)^(r-a) q^(d r) ReducedPower[X_i^-, r] ** X_j^- ** ReducedPower[X_i^-, -a - r]]]]
```

```
BraidAction[Γ_][{T_{i_}}, K_{j_}] := K_j ** NonCommutativePower[K_i, -CartanMatrix[Γ][[i, j]]]
```

```
BraidAction[Γ_][{T_{i_}}, K_{j_}^-1] := NonCommutativePower[K_i, CartanMatrix[Γ][[i, j]]] ** K_j^-1
```

```
OrderingRules[Γ_] :=
 OrderingRules[Γ] = With[{d = CartanFactors[Γ], a = CartanMatrix[Γ]}, {
    K_{i_} ** K_{i_}^-1 :→ 1,
    K_{i_}^-1 ** K_{i_} :→ 1,
    Y___ ** K_{i_}^{n_.} ** K_{j_}^{m_.} ** Z___ /; i > j :→ Y ** K_j^m ** K_i^n ** Z,
    X_{j_}^+ ** K_{i_}^{n_.} :→ q^(-n d[[i]] a[[i,j]]) K_i^n ** X_j^+,
    K_{i_}^{n_.} ** X_{j_}^- :→ q^(-n d[[i]] a[[i,j]]) X_j^- ** K_i^n,
    X_{i_}^+ ** X_{j_}^- :→ X_j^- ** X_i^+ + DiscreteDelta[i - j] (K_i - K_i^-1) / (q^d[[i]] - q^(-d[[i]]))
   }]
```

```
MemoryConserve::start : Running Share[] to conserve memory.

MemoryConserve::end : Finished running Share[]; 11776 bytes of memory freed.
```

```
ExtraOrderingRules[Γ_] :=
 ExtraOrderingRules[Γ] = With[{d = CartanFactors[Γ], a = CartanMatrix[Γ]},
   {Y___ ** X_{i_}^+ ** X_{j_}^+ ** Z___ /; (i < j ∧ a⟦i, j⟧ == 0) :> Y ** X_j^+ ** X_i^+ ** Z,
    Y___ ** X_{i_}^- ** X_{j_}^- ** Z___ /; (i < j ∧ a⟦i, j⟧ == 0) :> Y ** X_j^- ** X_i^- ** Z}]
```

```
CollectTerms[Z_] := Collect[Z, _NonCommutativeMultiply, Together]
```

```
differ[Z1_, Z2_] := CollectTerms[Z1 - Z2] =!= 0
```

```
fixedPoint[function_, expr_, test_] := NestWhile[function, expr, test, 2]
```

```
General::spell1 : Possible spelling error: new symbol name "fixedPoint" is similar to existing symbol
   "FixedPoint". More…
```

```
ReorderQuantumMonomial[Γ_][Z_] :=
 fixedPoint[CollectTerms[# /. OrderingRules[Γ]] &, Z, differ]
```

```
ReorderQuantumMonomial[Γ_][Z_Plus] /; Length[Z] ≤ termThreshold :=
 CollectTerms[ReorderQuantumMonomial[Γ] /@ Z]
ReorderQuantumMonomial[Γ_][Z_Plus] /; Length[Z] > termThreshold :=
 CollectTerms[Plus @@ (ReorderQuantumMonomial[Γ] /@ partialPartition[Z, termThreshold])]
```

```
BraidAction[Γ_][{T_}, Z_NonCommutativeMultiply] :=
 BraidAction[Γ][{T}, Z] = Module[{result},
   DebugPrintHeld["Calculating ", BraidAction[Γ][{T}, Z]];
   result = ReorderQuantumMonomial[Γ][BraidAction[Γ][{T}, #] & /@ Z];
   DebugPrintHeld["Finished calculating ", BraidAction[Γ][{T}, Z]];
   result
  ]
```

```
termThreshold = 20;
```

```
partialPartition[Z_, n_Integer] :=
 With[{h = Head[Z]}, h @@ # & /@ Partition[List @@ Z, n, n, {1, 1}, {}]]
```

```
BraidAction[Γ_][{word__}, Z_Plus] /; Length[Z] ≤ termThreshold :=
 CollectTerms[BraidAction[Γ][{word}, #] & /@ Z]
BraidAction[Γ_][{word__}, Z_Plus] /; Length[Z] > termThreshold := Module[{sum},
  DebugPrint["Distributing BraidAction[",
   Γ, "][", {word}, ", ...] over ", Length[Z], " terms."];
  sum = Plus @@ (
     (DebugPrint[" ... computing ", termThreshold, " terms"];
       BraidAction[Γ][{word}, #]) & /@ partialPartition[Z, termThreshold]
    );
  DebugPrint[" ... and assembling all the terms"];
  CollectTerms[sum]
 ]
BraidAction[Γ_][{word__}, α_?qNumberQ Z_] := α BraidAction[Γ][{word}, Z]
```

```
BraidAction[Γ_][{T_, S__}, Z_] := BraidAction[Γ][{T, S}, Z] = Module[{result},
   DebugPrintHeld["Calculating ", BraidAction[Γ][{T, S}, Z]];
   result = CollectTerms[BraidAction[Γ][{T}, BraidAction[Γ][{S}, Z]]];
   DebugPrintHeld["Finished calculating ", BraidAction[Γ][{T, S}, Z]];
   result
  ]
```

```
BraidAction[Γ_][{}, Z_] := Z
```

```
BraidRelations[Γ_] := Module[{m = CartanMatrix[Γ] × Transpose[CartanMatrix[Γ]] /.
     {n_?# ≥ 4 & :→ ∞, 3 → 6, 2 → 4, 1 → 3, 0 → 2}, w},
  w[i_, j_, n_] := Take[{T_i, T_j, T_i, T_j, T_i, T_j}, n];
  DeleteCases[Flatten[Table[If[m⟦i, j⟧ < ∞, w[i, j, m⟦i, j⟧] == w[j, i, m⟦i, j⟧], True],
     {i, 1, Rank[Γ]}, {j, i, Rank[Γ]}]], True]
 ]
```

```
CheckBraidRelation[Γ_][word1_ == word2_] := And @@
  Simplify[BraidAction[Γ][word1, #] == BraidAction[Γ][word2, #] & /@ Generators[Γ] //.
    OrderingRules[Γ] ~ Join ~ ExtraOrderingRules[Γ]]
```

```
CheckBraidRelations[Γ_] := And @@ (CheckBraidRelation[Γ] /@ BraidRelations[Γ])
```

```
End[];
```

End of package

```
EndPackage[];
```

Testing

```
CheckBraidRelations[B_2]
```

```
True
```

```
(*CheckBraidRelations[B_3]*)(*This doesn't work,
because it needs Serre relations to simplify the results...*)
```

```
MemoryConserve::start : Running Share[] to conserve memory.

MemoryConserve::end : Finished running Share[]; 68528 bytes of memory freed.
```