QuantumGroups package

Version 2, June 19 2005, Scott Morrison

Introduction

Implementation

```
(* We let the symbols X and K escape to the System` context,
to avoid shadowing problems with KnotTheory`. *)
System`K;
System`X;
```

```
DeclarePackage["QuantumGroups`Utilities`IntersectSubspaces`", {"IntersectSubspaces"}];
```

```
DeclarePackage["QuantumGroups`Utilities`MatrixWrapper`",
  {"OnesMatrix", "ZeroesMatrix", "ZeroMatrixQ", "NonZeroMatrixQ",
   "Matrix", "MatrixData", "identityMatrix", "KroneckerProduct",
   "BlockDiagonalMatrix", "AppendRows", "AppendColumns", "MatrixInverse",
   "PrepareInverse", "InterpolationInverseThreshold"}];
```

```
DeclarePackage["QuantumGroups`Utilities`Debugging`",
  {"DebugEcho", "DebugPrint", "DebugEvaluate", "DebugSet", "DebugPrintHeld"}];
```

```
DeclarePackage["QuantumGroups`Utilities`DataPackage`",
  {"DefiniteValues", "MatchingValues", "ValuesAsString", "PackageData",
   "PackageMatrixPresentations", "PackageDecompositionMaps", "PackageQuantumRoots",
   "PackageRMatrix", "PackagePartialRMatrixPresentation", "PackageHighWeightVectors",
   "PackageBraidingMatrices", "PackageBRPresentations", "PackageBraidingMaps"}];
```

In[ ]:=
```
DeclarePackage["QuantumGroups`RootSystems`",
  {"CartanMatrix", "CartanFactors", "LacingNumber", "Rank", "KillingForm",
   "ρ", "SimpleRoots", "SimpleReflection", "WeylOrbit", "RootWeightQ",
   "WeightsModRoots", "WeightInLatticeQ", "IntermediateLattices",
   "PositiveWeightQ", "InWeylPolytopeQ", "SortWeights", "SortWeightMultiplicities",
   "MinisculeWeightQ", "MinusculeRepresentationQ", "MinusculeRepresentations",
   "ReflectIntoPositiveWeylChamber", "ShortDominantRoots",
   "LongDominantRoots", "ShortSimpleRoots", "ShortRoots",
   "ShortDominantRootQ", "DualCoxeterNumber", "TwistFactor"}];
```

In[ ]:=
```
DeclarePackage["QuantumGroups`RootsOfUnity`",
  {"AlcoveDefiningRoot", "WeightInAlcoveQ", "AlcoveWeights",
   "AlcoveWeightsInLattice", "AlcoveRoots", "LevelFromRoot", "RootFromLevel"}];
```

```
DeclarePackage["QuantumGroups`Algebra`",
  {"1", "0", "PositiveGenerators", "NegativeGenerators", "CartanGenerators",
   "Generators", "NonCommutativePower", "Δ", "Δop", "OperatorWeight", "OperatorLength"}];
```

```
DeclarePackage["QuantumGroups`BraidAction`",
  {"T", "BraidAction", "BraidRelations", "CheckBraidRelations"}];
```

```
DeclarePackage["QuantumGroups`QuantumRoots`", {"ExpandQuantumRoot",
   "QuantumPositiveRoots", "QuantumNegativeRoots", "QuantumRootHeight"}];
```

In[•]:=
```
DeclarePackage["QuantumGroups`LittelmannPaths`",
  {"LittelmannPath", "LittelmannPathDecomposeRepresentation",
   "LittelmannPathWeightMultiplicities", "LittelmannPathOneStepLowerings",
   "LittelmannPathLowerings", "LittelmannPathEndpoint"}];
```

In[•]:=
```
DeclarePackage["QuantumGroups`Steinberg`", {"SteinbergDecomposeRepresentation"}];
```

In[•]:=
```
DeclarePackage["QuantumGroups`WeylGroups`",
  {"PositiveRoots", "LongestWordDecomposition", "LongestWord", "WeylGroup"}];
```

In[•]:=
```
DeclarePackage["QuantumGroups`Representations`", {"WeightMultiplicities",
   "WeightMultiplicity", "MultiplicityFreeQ", "DecomposeRepresentation", "Weights",
   "WeightDiameter", "PositiveWeights", "qDimension", "DualRepresentation"}];
```

In[•]:=
```
DeclarePackage["QuantumGroups`MatrixPresentations`",
  {"ChangeOfBasisMatrix", "ShortRootBasis", "FundamentalBasis", "MatrixPresentation",
   "HighWeightVectors", "HighWeightVectorQ", "AllHighWeightVectors"}];
```

```
DeclarePackage["QuantumGroups`AlgebraRelations`", {"Relations", "CheckRelations"}];
```

```
DeclarePackage["QuantumGroups`RMatrix`", {"RMatrix", "CheckRMatrixOppositeCommutes"}];
```

```
DeclarePackage["QuantumGroups`RepresentationTensors`",
  {"RepresentationTensor", "Domain", "Codomain", "DomainBasis", "CodomainBasis",
   "Algebra", "IdentityMap", "ZeroTensorQ", "CheckRepresentationTensor",
   "RepresentationTensorErrors", "QuantumTrace", "Distributor", "Associator",
   "InverseAssociator", "BraidingMap", "InverseBraidingMap", "NormalisedBraidingMap",
   "InverseNormalisedBraidingMap", "DecompositionMap", "InverseDecompositionMap",
   "PermuteDirectSummands", "DirectSumProjectionMap", "DirectSumInclusionMap"}];
```

```
DeclarePackage["QuantumGroups`Braiding`", {"BraidingData", "CheckBraidingData", "BR",
   "LoadAllBraidingDataFromWiki", "PackageBraidingData", "WriteBraidingDataToWiki"}];
```

```
Print[
 "Loading QuantumGroups` version 2.0\n",
 "Read more at http://katlas.math.toronto.edu/wiki/QuantumGroups"
]
```

In[ ]:=
```
BeginPackage["QuantumGroups`"];
```

```
QuantumGroupsDirectory::usage =
  "QuantumGroupsDirectory[] should hopefully return the location
    the QuantumGroups` package was loaded from.";
QuantumGroupsDataDirectory::usage = "QuantumGroupsDataDirectory[] specifies were
    the QuantumGroups` package should look for, and save, precomputed data.";
```

In[ ]:=
```
{A, B, C, D, E, F, G};
```

```
A1 = A[1];
A2 = A[2];
A3 = A[3];
A4 = A[4];
A5 = A[5];
A6 = A[6];
A7 = A[7];
A8 = A[8];
A9 = A[9];
A10 = A[10];
A11 = A[11];
A12 = A[12];
B2 = B[2];
B3 = B[3];
B4 = B[4];
B5 = B[5];
B6 = B[6];
B7 = B[7];
B8 = B[8];
C3 = C[3];
C4 = C[4];
C5 = C[5];
C6 = C[6];
C7 = C[7];
C8 = C[8];
D4 = D[4];
D5 = D[5];
D6 = D[6];
D7 = D[7];
D8 = D[8];
E6 = E[6];
E7 = E[7];
E8 = E[8];
F4 = F[4];
G2 = G[2];
```

In[ ]:=
```
{Irrep, ℂ, β};
```

In[ ]:=
```
SetAttributes[DirectSum, {Flat, OneIdentity}]
```

In[ ]:=
```
CircleTimes /: Power[V_, CircleTimes[n_]] := TensorPower[V, n]
```

In[ ]:=
```
CirclePlus = DirectSum;
```

```
CircleTimes[x : Except[_Integer]] := x
```

```
CircleTimes[a__, b_, c_] := CircleTimes[CircleTimes[a, b], c]
```

In[ ]:=
```
TensorPower[_, 0] := CircleTimes[]
TensorPower[x_, n_?NaturalQ] := Fold[CircleTimes, x, Table[x, {n - 1}]]
```

In[ ]:=
```
{TensorPower, DirectSum};
```

```
QuantumGroups::loading = "Loading precomputed data in `1`."
```

In[ ]:=
```
qInteger::usage = "qInteger[n][q] computes the quantum integer n with the variable q.";
{qFactorial};
qNumberQ::usage =
  "qNumberQ[x] tests if x is a rational function in q. It (fakes) does so simply by
    replacing q with 3.14159, and testing if the resulting expression is a number.";
```

In[ ]:=
```
UnsortedUnion::usage =
  "UnsortedUnion[list] a list of all the unique elements in list, in
    the order that they first appear.";
```

In[ ]:=
```
NaturalQ::usage = "NaturalNumberQ[n] tests if n is a non-negative integer.";
```

In[ ]:=
```
If[$VersionNumber < 6,
 UnitVector::usage =
   "UnitVector[n,i] returns the i-th n-dimensional unit vector, if i is an
     integer between 1 and n, and the n-dimensional zero vector otherwise.";
]
```

```
If[$VersionNumber ≥ 6.,
 Unprotect[IdentityMatrix];
 IdentityMatrix[0] = {};
 Protect[IdentityMatrix];
]
```

In[ ]:=
```
ZeroVector::usage = "ZeroVector[n] returns the n-dimensional 0 vector.";
```

In[ ]:=
```
UnitVectorQ::usage = "UnitVectorQ[v] tests if v is a unit coordinate vector.";
```

In[ ]:=
```
ZeroVectorQ::usage = "ZeroVectorQ[v] tests if v is the zero vector.";
```

```
AdjointRepresentation[A_n_] := Irrep[A_n][UnitVector[n, 1] + UnitVector[n, n]]
AdjointRepresentation[G2] = Irrep[G2][UnitVector[2, 2]];
AdjointRepresentation[F4] = Irrep[F4][UnitVector[4, 1]];
AdjointRepresentation[E6] = Irrep[E6][UnitVector[6, 2]];
AdjointRepresentation[E7] = Irrep[E7][UnitVector[7, 1]];
AdjointRepresentation[E8] = Irrep[E8][UnitVector[8, 8]];
```

In[ ]:=
```
Begin["`Private`"];
```

In[ ]:=
```
QuantumGroupsDirectory[] := QuantumGroupsDirectory[] =
  StringDrop[(File /. Flatten[FileInformation[ToFileName[#, "QuantumGroups"]] & /@
       ($Path /. "." → Directory[])]), -14]
```

```
(* might be dangerous if QuantumGroupsDirectory[] is somehow incorrect! *)
If[! MemberQ[$Path, QuantumGroupsDirectory[]],
 AppendTo[$Path, QuantumGroupsDirectory[]]]
```

```
If[StringTake[QuantumGroupsDirectory[], -7] == "package",
  QuantumGroupsDataDirectory[] := StringDrop[QuantumGroupsDirectory[], -7] <> "data";
  Print["Found precomputed data in ", QuantumGroupsDataDirectory[]];
  If[! MemberQ[$Path, QuantumGroupsDataDirectory[]],
   AppendTo[$Path, QuantumGroupsDataDirectory[]]],
  Print["Remember to set QuantumGroupsDataDirectory[] to the
     appropriate path, if you've downloaded precomputed data."]
 ];
```

In[ ]:=
```
qInteger[n_Integer][q_] := Sum[q^k, {k, -n + 1, n - 1, 2}]
```

In[ ]:=
```
qFactorial[n_Integer][q_] := Expand[Times @@ Table[qInteger[i][q], {i, 1, n}]]
```

In[ ]:=
```
qNumberQ[x_] := NumberQ[x /. Global`q → 3.14159]
```

In[ ]:=
```
UnsortedUnion[x_] := Module[{f}, f[y_] := (f[y] = Sequence[]; y); f /@ x]
```

In[ ]:=
```
NaturalQ[n_] := NonNegative[n] && IntegerQ[n]
```

In[ ]:=
```
ZeroVector[n_] := Table[0, {n}]
```

```
If[$VersionNumber < 6,
 (UnitVector[n_, i_Integer] /; (1 ≤ i ≤ n) := Module[{z = Table[0, {n}]}, z[[i]] = 1;
    z]);
 (UnitVector[n_, i_Integer] := Table[0, {n}]),
 Unprotect[UnitVector];
 (UnitVector[n_, i_Integer] /; (i < 1 ∨ i > n) := (Message[UnitVector::nokun, n, i];
    ZeroVector[n]));
 Protect[UnitVector];
]
```

In[ ]:=
```
UnitVectorQ[v_?VectorQ] := Complement[v, {0, 1}] == {} ∧ Count[v, 1] == 1
```

In[ ]:=
```
ZeroVectorQ[v_?VectorQ] := Union[v] === {0} ∨ v == {}
```

In[ ]:=
```
End[];
```

```
<< QuantumGroups`Utilities`Debugging`
<< QuantumGroups`Utilities`DataPackage`
```

In[ ]:=
```
EndPackage[];
```