RepresentationTensors package

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

Introduction

Implementation

Start of package

Specify package dependencies:

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

Usage messages

```
RepresentationTensor::usage = "";
```

```
Domain::usage = "";
```

```
Codomain::usage = "";
```

```
DomainBasis::usage = "";
```

```
CodomainBasis::usage = "";
```

```
Algebra::usage = "";
```

```
IdentityMap::usage = ""; ZeroMap::usage = "";
```

```
ZeroTensorQ::usage = "";
```

```
CheckRepresentationTensor::usage = "";
```

```
RepresentationTensorErrors;
```

```
QuantumTrace::usage = "";
```

```
Distributor; Associator; InverseAssociator;
```

```
BraidingMap;
InverseBraidingMap;
NormalisedBraidingMap;
InverseNormalisedBraidingMap;
```

```
DecompositionMap; InverseDecompositionMap;
```

```
PermuteDirectSummands; DirectSumProjectionMap; DirectSumInclusionMap;
```

Internals

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

Basic stuff

```
RepresentationTensor /: Codomain[RepresentationTensor[_, V_, _, _, _, _]] := V
RepresentationTensor /: Domain[RepresentationTensor[_, _, _, W_, _, _]] := W
```

```
RepresentationTensor /: CodomainBasis[RepresentationTensor[_, _, β_, _, _, _]] := β
RepresentationTensor /: DomainBasis[RepresentationTensor[_, _, _, _, β_, _]] := β
```

```
RepresentationTensor /: Algebra[RepresentationTensor[Γ_, _, _, _, _, _]] := Γ
```

```
IdentityMap[Γ_, CircleTimes[], β_] :=
 RepresentationTensor[Γ, ℂ, β, ℂ, β, {{ZeroVector[Rank[Γ]], identityMatrix[1]}}]
```

```
IdentityMap[Γ_, V_, β_] := RepresentationTensor[Γ, V, β, V, β,
  {#, identityMatrix[WeightMultiplicity[Γ, V, #]]} & /@ Weights[Γ, V]]
```

```
ZeroMap[Γ_, V_, βV_, W_, βW_] := RepresentationTensor[Γ, V, βV, W, βW,
  {#, zeroMatrix[WeightMultiplicity[Γ, V, #], WeightMultiplicity[Γ, W, #]]} & /@
   Weights[Γ, V]]
```

```
ZeroTensorQ[RepresentationTensor[_, _, _, _, _, matrices_]] :=
 And @@ (ZeroMatrixQ[#[[2]]] & /@ matrices)
```

```
RepresentationTensor /: Together[RepresentationTensor[Γ_, Vc_, βc_, Vd_, βd_, data_]] :=
 RepresentationTensor[Γ, Vc, βc, Vd, βd, Together[data]]
```

Checking representation tensors

```
CheckRepresentationTensor[F : RepresentationTensor[Γ_, Vc_, βc_, Vd_, βd_, _], λ_] :=
 And @@ (DebugEcho[ZeroMatrixQ[
       Simplify[MatrixPresentation[Γ][#][Vc, βc, λ].F[λ] - F[λ + OperatorWeight[Γ][#]].
          MatrixPresentation[Γ][#][Vd, βd, λ]]]] & /@ Generators[Γ])
```

```
RepresentationTensorErrors[F : RepresentationTensor[Γ_, Vc_, βc_, Vd_, βd_, _], λ_] :=
 DeleteCases[{#, MatrixPresentation[Γ][#][Vc, βc, λ].F[λ],
     F[λ + OperatorWeight[Γ][#]].MatrixPresentation[Γ][#][Vd, βd, λ]} & /@
   Generators[Γ], {_, a_, b_} /; ZeroMatrixQ[Simplify[a - b]]]
```

```
CheckRepresentationTensor[F : RepresentationTensor[Γ_, _, _, V_, _, _]] :=
 And @@ (CheckRepresentationTensor[F, #] & /@ Weights[Γ, V])
```

Operations

```
RepresentationTensor /: RepresentationTensor[Γ_, V_, _, W_, _, matrices_][λ_] :=
  (Cases[matrices, {λ, m_Matrix} :> m] ~Join~
     {ZeroesMatrix[WeightMultiplicity[Γ, V, λ], WeightMultiplicity[Γ, W, λ]]})[[1]]
```

```
RepresentationTensor /: Plus[t__RepresentationTensor] /;
   (SameQ[Codomain /@ {t}] ∧ SameQ[Domain /@ {t}] ∧ SameQ[CodomainBasis /@ {t}] ∧
     SameQ[DomainBasis /@ {t}] ∧ SameQ[Algebra /@ {t}]) := Module[{t1, Γ, domain, λ},
  t1 = First[{t}];
  Γ = Algebra[t1];
  domain = Domain[t1];
  RepresentationTensor[Γ, Codomain[t1],
   CodomainBasis[t1], Domain[t1], DomainBasis[t1],
   Table[(λ = Weights[Γ, domain][[i]];
     {λ, Plus @@ (#[λ] & /@ {t})}), {i, 1, Length[Weights[Γ, domain]]}]]
 ]
```

```
RepresentationTensor /: α_ RepresentationTensor[Γ_, V1_, β1_, V2_, β2_, matrices_List] :=
 RepresentationTensor[Γ, V1, β1, V2, β2, {#[[1]], α #[[2]]} & /@ matrices]
```

```
RepresentationTensor /: (F : RepresentationTensor[Γ_, V1_, β1_, V2_, β2_, _List]).
  (G : RepresentationTensor[Γ_, V2_, β2_, V3_, β3_, _List]) := RepresentationTensor[Γ,
  V1, β1, V3, β3, ((*Print["Composing weight ",#];*){#, Together[F[#].G[#]]}) & /@
   SortWeights[Γ][Union[Weights[Γ, V1] ~Join~ Weights[Γ, V3]]]]
```

```
RepresentationTensor /: (F : RepresentationTensor[Γ_, V3_, β3_, V2_, β2_, _]).
  (Gs : {RepresentationTensor[Γ_, V2_, β2_, V1_, β1_, _] ...}) := F.# & /@ Gs
```

```
RepresentationTensor /: (Fs : {RepresentationTensor[Γ_, V3_, β3_, V2_, β2_, _] ...}).
  (G : RepresentationTensor[Γ_, V2_, β2_, V1_, β1_, _]) := #.G & /@ Fs
```

```
RepresentationTensor /: CircleTimes[F : RepresentationTensor[Γ_, V1_, βc_, V2_, βd_, _],
  G : RepresentationTensor[Γ_, V3_, βc_, V4_, βd_, _]] :=
 Module[{λ, μ, codomain = V1 ⊗ V3, domain = V2 ⊗ V4, productWeights, rightWeights},
  (*Print[codomain];
  Print[domain];*)
  productWeights =
   SortWeights[Γ][Union[Weights[Γ, domain] ~Join~ Weights[Γ, codomain]]];
  (*Print[productWeights];*)
  rightWeights = SortWeights[Γ][Union[Weights[Γ, V4] ~Join~ Weights[Γ, V3]]];
  (*Print[rightWeights];*)
  RepresentationTensor[Γ, codomain, βc, domain, βd,
   Table[λ = productWeights[[i]];
    {λ, BlockDiagonalMatrix @@ Table[μ = rightWeights[[j]];
       Expand[MatrixKroneckerProduct[F[λ - μ], G[μ]]], {j, 1, Length[rightWeights]}]},
    {i, 1, Length[productWeights]}]
  ]
 ]
```

```
RepresentationTensor /: (F : RepresentationTensor[Γ_, V1_, βc_, V2_, βd_, _]) ⊕
  (G : RepresentationTensor[Γ_, V3_, βc_, V4_, βd_, _]) :=
 (DebugPrint["taking direct sums of tensors..."];
  RepresentationTensor[Γ, V1 ⊕ V3, βc,
   V2 ⊕ V4, βd, With[{λ = #}, DebugPrint["... weight ", λ];
     {λ, BlockDiagonalMatrix[F[λ], G[λ]]}] & /@ Weights[Γ, V2 ⊕ V4]])
```

```
RepresentationTensor /:
 Inverse[RepresentationTensor[Γ_, V1_, β1_, V2_, β2_, matrices_]] :=
 RepresentationTensor[Γ, V2, β2, V1, β1, (PrepareInverse[#[[2]]] & /@ matrices;
   {#[[1]], Inverse[#[[2]]]} & /@ matrices)]
```

```
QuantumTrace[RepresentationTensor[Γ_, V_, β_, V_, β_, matrices_]] :=
 Together[Sum[(Global`q^(2 KillingForm[Γ][ρ[Γ], matrices[[i,1]]]
     Tr[(*MatrixPresentation[Γ][K_{Γ,ρ}][V,β,matrices[[i,1]].*)matrices[[i, 2]]]), {i, 1, Length[matrices]}]]
```

Associators

```
Associator[Γ_, V1_, V2_, V3_, β_] := RepresentationTensor[Γ, (V1 ⊗ V2) ⊗ V3, β,
  V1 ⊗ (V2 ⊗ V3), β, {#, Associator[Γ, V1, V2, V3, β, #]} & /@ Weights[Γ, V1 ⊗ (V2 ⊗ V3)]]
```

```
InverseAssociator[Γ_, V1_, V2_, V3_, β_] :=
 RepresentationTensor[Γ, V1 ⊗ (V2 ⊗ V3), β, (V1 ⊗ V2) ⊗ V3, β,
  {#, InverseAssociator[Γ, V1, V2, V3, β, #]} & /@ Weights[Γ, V1 ⊗ (V2 ⊗ V3)]]
```

```
Associator[Γ_, U_, V_, W_, β_, λ_] :=
 Matrix[LeftAssociator[Γ, U, V, W, λ].Inverse[RightAssociator[Γ, U, V, W, λ]]]
```

```
InverseAssociator[Γ_, U_, V_, W_, β_, λ_] :=
 Matrix[RightAssociator[Γ, U, V, W, λ].Inverse[LeftAssociator[Γ, U, V, W, λ]]]
```

Okay. Let's write down the two maps $U_a \otimes V_b \otimes W_c \to U_a \otimes (V \otimes W)_{b+c} \to (U \otimes (V \otimes W))_{a+b+c}$ and $U_a \otimes V_b \otimes W_c \to (U \otimes V)_{a+b} \otimes W_c \to ((U \otimes V) \otimes W)_{a+b+c}$.

```
ConstituentWeights[Γ_, V1_, V2_, V3_, λ_] :=
 Select[Flatten[Outer[{λ - #1 - #2, #2, #1} &, Weights[Γ, V3], Weights[Γ, V2], 1], 1],
  MemberQ[Weights[Γ, V1], #[[1]]] &]
```

```
(*RightAssociatedWeightSpaceInclusion[Γ_,{U_,V_,W_},{a_,b_,c_}]:=
 TensorProductWeightSpaceInclusion[Γ,{U,V⊗W},{a,b+c}].
    MatrixKroneckerProduct[identityMatrix[WeightMultiplicity[Γ,U,a]],
     TensorProductWeightSpaceInclusion[Γ,{V,W},{b,c}]]
   LeftAssociatedWeightSpaceInclusion[Γ_,{U_,V_,W_},{a_,b_,c_}]:=
  TensorProductWeightSpaceInclusion[Γ,{U⊗V,W},{a+b,c}].
   MatrixKroneckerProduct[TensorProductWeightSpaceInclusion[Γ,{U,V},{a,b}],
    identityMatrix[WeightMultiplicity[Γ,W,c]]]*)
```

```
(*RightAssociator[Γ_,U_,V_,W_,λ_]:=
 AppendRows@@(RightAssociatedWeightSpaceInclusion[Γ,{U,V,W},#]&/@
      ConstituentWeights[Γ,U,V,W,λ])
   LeftAssociator[Γ_,U_,V_,W_,λ_]:=AppendRows@@
   (LeftAssociatedWeightSpaceInclusion[Γ,{U,V,W},#]&/@
     ConstituentWeights[Γ,U,V,W,λ])*)
```

```
CoordinateInclusion /: DirectSum[i : CoordinateInclusion[n_Integer, _List] ..] :=
 CoordinateInclusion[n, Join @@ {i}[[All, 2]]]
```

```
CoordinateInclusion /: CoordinateInclusion[n_Integer, p_List].
  CoordinateInclusion[m_Integer, q_List] := CoordinateInclusion[n, p[[#]] & /@ q]
```

```
CoordinateInclusion /: Inverse[CoordinateInclusion[n_, p_List] /; Length[p] == n] :=
 CoordinateInclusion[n, Ordering[p]]
```

```
CoordinateInclusion /:
 CoordinateInclusion[n_Integer, p_List] ⊗ CoordinateInclusion[m_Integer, q_List] :=
 CoordinateInclusion[n m, Flatten[Outer[#2 + m (#1 - 1) &, p, q]]]
```

```
CoordinateInclusion /: Matrix[CoordinateInclusion[n_, p_List]] :=
 Matrix[n, Length[p], IdentityMatrix[n][[All, p]]]
```

```
CoordinateInclusion[n_Integer] := CoordinateInclusion[n, Range[n]]
```

```
FindFirst[a_, b_] := Position[a, b, {1}, 1, Heads → False][[1, 1]]
```

```
CoordinateInclusion[Γ_, {V_, W_}, {a_, b_}] := CoordinateInclusion[
  WeightMultiplicity[Γ, V ⊗ W, a + b],
  QuantumGroups`MatrixPresentations`Private`WeightMultiplicityPartialSums[
     Γ, V, W, a + b][[FindFirst[Weights[Γ, W], b]]]
   + Range[WeightMultiplicity[Γ, V, a] × WeightMultiplicity[Γ, W, b]]
 ]
```

```
RightAssociatedWeightSpaceInclusion[Γ_, {U_, V_, W_}, {a_, b_, c_}] :=
 CoordinateInclusion[Γ, {U, V ⊗ W}, {a, b + c}].
  (CoordinateInclusion[WeightMultiplicity[Γ, U, a]] ⊗
    CoordinateInclusion[Γ, {V, W}, {b, c}])
LeftAssociatedWeightSpaceInclusion[Γ_, {U_, V_, W_}, {a_, b_, c_}] :=
 CoordinateInclusion[Γ, {U ⊗ V, W}, {a + b, c}].(CoordinateInclusion[Γ, {U, V}, {a, b}] ⊗
    CoordinateInclusion[WeightMultiplicity[Γ, W, c]])
```

```
RightAssociator[Γ_, U_, V_, W_, λ_] :=
 DirectSum @@ (RightAssociatedWeightSpaceInclusion[Γ, {U, V, W}, #] & /@
    ConstituentWeights[Γ, U, V, W, λ])
LeftAssociator[Γ_, U_, V_, W_, λ_] := DirectSum @@
  (LeftAssociatedWeightSpaceInclusion[Γ, {U, V, W}, #] & /@
    ConstituentWeights[Γ, U, V, W, λ])
```

Distributors

Braidings

```
BraidingMap[Γ_, V_ ⊗ W_, β_] := BraidingMap[Γ, V ⊗ W, β] =
  RepresentationTensor[Γ, W ⊗ V, β, V ⊗ W, β,
   {#, SwitchTensorProductWeightSpace[Γ, V ⊗ W, #].RMatrix[Γ, V, W, β, #]} & /@
    Weights[Γ, V ⊗ W]]
```

```
NormalisedBraidingMap[Γ_, V_ ⊗ V_, β_] :=
 NormalisedBraidingMap[Γ, V⊗V, β] = With[{b = BraidingMap[Γ, V⊗V, β]},
   Simplify[qDimension[Γ][V] / QuantumTrace[b]] b
  ]
```

```
NormalisedBraidingMap[Γ_, V_ ⊗ W_, β_] := BraidingMap[Γ, V⊗W, β]
```

```
InverseBraidingMap[Γ_, V_ ⊗ W_, β_] :=
 InverseBraidingMap[Γ, V⊗W, β] = Inverse[BraidingMap[Γ, V⊗W, β]]
```

```
InverseNormalisedBraidingMap[Γ_, V_ ⊗ W_, β_] :=
 InverseNormalisedBraidingMap[Γ, V⊗W, β] = Inverse[NormalisedBraidingMap[Γ, V⊗W, β]]
```

```
SwitchTensorProductWeightSpace[Γ_, V_ ⊗ W_, λ:{__Integer}] :=
 Sum[TensorProductWeightSpaceInclusion[
     Γ, {W, V}, {Weights[Γ, W][[i]], λ - Weights[Γ, W][[i]]}].
   SwitchTensorProduct[WeightMultiplicity[Γ, V, λ - Weights[Γ, W][[i]]],
    WeightMultiplicity[Γ, W, Weights[Γ, W][[i]]]].
   Transpose[TensorProductWeightSpaceInclusion[Γ, {V, W},
     {λ - Weights[Γ, W][[i]], Weights[Γ, W][[i]]}]],
  {i, 1, Length[Weights[Γ,W]]}]
```

```
SwitchTensorProduct[d1_Integer, d2_Integer] := Matrix[d1 d2, d1 d2,
  UnitVector[d1 d2, #] & /@ Ordering[Flatten[Table[{j, i}, {i, 1, d1}, {j, 1, d2}], 1]]]
```

Decomposition tensors

```
HighWeightVectors[Γ_][(U_ ⊗ V_) ⊗ W_, β_, λ_] := HighWeightVectors[Γ][(U⊗V) ⊗ W, β, λ] =
  Module[{irreps, decomposition, inclusions, result},
   DebugPrintHeld["Calculating ", HighWeightVectors[Γ][(U⊗V) ⊗ W, β, λ],
    " (iterated tensor product)"];
   irreps = DecomposeRepresentation[Γ][U⊗V];
   decomposition = DecompositionMap[Γ, U⊗V, β] ⊗ IdentityMap[Γ, W, β];
   DebugPrintHeld["Calculated decomposition map..."];
   inclusions = Table[DirectSumInclusionMap[Γ, irreps, i, β] ⊗ IdentityMap[Γ, W, β],
     {i, 1, Length[irreps]}];
   result = Flatten[Table[
      decomposition[λ].inclusions[[i]][λ].# & /@ HighWeightVectors[Γ][irreps[[i]] ⊗ W, β, λ],
      {i, 1, Length[irreps]}
     ], 1];
   DebugPrintHeld["Finished calculating ",
    HighWeightVectors[Γ][(U⊗V) ⊗ W, β, λ], " (iterated tensor product)"];
   result
  ]
```

```
LoadDecompositionMaps[Γ_n_] := Module[{},
  Off[Get::noopen, Needs::nocont];
  Needs[
   "QuantumGroups`Data`" <> SymbolName[Γ] <> ToString[n] <> "`DecompositionMaps`"];
  On[Get::noopen, Needs::nocont];
  LoadDecompositionMaps[Γn] = False;
  True
 ]
```

```
weightToString[λ : {__Integer}] :=
 StringDrop[StringJoin @@ ((ToString[#] <> "$") & /@ λ), -1]
```

```
LoadDecompositionMaps[Γ_n_, Z_] := Module[{tensorPowerQ, tensorPowerPattern, data},
  If[LoadDecompositionMaps[Γn], True,
   tensorPowerQ[V_][W_] := MatchQ[W, V] || MatchQ[W, U_ ⊗ V /; tensorPowerQ[V][U]];
   tensorPowerPattern = U_ ⊗ (V : (Irrep[Γn][λ_])) /; tensorPowerQ[V][U];
   If[MatchQ[Z, tensorPowerPattern],
    data = Z /. X : (_ ⊗ Y : Irrep[Γn][λ_]) :> {λ, Count[X, Irrep[Γn][λ], ∞]};
    Off[Get::noopen, Needs::nocont];
    Needs[
     "QuantumGroups`Data`" <> SymbolName[Γ] <> ToString[n] <> "`DecompositionMaps`" <>
      "w" <> weightToString[data[[1]]] <> "`k" <> ToString[data[[2]]] <> "`"];
    On[Get::noopen, Needs::nocont];
   ];
   LoadDecompositionMaps[Γn, Z] = False;
   True
  ]
 ]
```

```
DecompositionMap[Γ_, Irrep[Γ_][λ_], β_] := IdentityMap[Γ, Irrep[Γ][λ], β]
```

```
DecompositionMap[Γ_, V : DirectSum[Irrep[Γ_][_] ..], β_] := Inverse[
  PermuteDirectSummands[Γ][V, β, Ordering[V][[Ordering[Ordering[SortWeights[Γ][V]]]]]]]
```

```
DecompositionMap[Γ_, V_ ⊗ W_, β_] :=
 If[LoadDecompositionMaps[Γ, V⊗W], DecompositionMap[Γ, V⊗W, β],
  Module[{result},
   DecompositionMap[Γ, V⊗W, β] = result =
     (DebugPrint[DecompositionMap, Γ, V, W];
      RepresentationTensor[Γ, V⊗W, β,
       DecomposeRepresentation[Γ][V⊗W], β, {#, (DebugPrint[" ... weight ", #];
           QuantumGroups`MatrixPresentations`Private`DirectSumDecomposition[Γ][
            V⊗W, β, #])} & /@ Reverse[Weights[Γ, V⊗W]]]);
   PackageDecompositionMaps[Γ];
   result
  ]
 ]
```

```
DecompositionMap[Γ_, (U_ ⊗ V_) ⊗ W_, β_] :=
 If[LoadDecompositionMaps[Γ, (U⊗V)⊗W], DecompositionMap[Γ, (U⊗V)⊗W, β],
  Module[{distributor, firstDecomposition,
    summandDecompositions, disordered, domain, disordering, result},
   DecompositionMap[Γ, (U⊗V)⊗W, β] = With[{Z = DecomposeRepresentation[Γ][U⊗V]},
     DebugPrintHeld["Beginning ", DecompositionMap[Γ, (U⊗V)⊗W, β]];
     distributor = Distributor[Γ][Z⊗W, β];
     DebugPrint["...prepared distributor"];
     firstDecomposition = DecompositionMap[Γ, U⊗V, β];
     DebugPrint["...prepared first decomposition map"];
     summandDecompositions = (DecompositionMap[Γ, #⊗W, β] & /@ Z);
     DebugPrint["...prepared all decomposition maps"];
     disordered =
      (firstDecomposition ⊗ IdentityMap[Γ, W, β]).distributor.summandDecompositions;
     DebugPrint["... calculated the composition"];
     domain = Domain[disordered];
     disordering = Ordering[domain][[Ordering[Ordering[SortWeights[Γ][domain]]]]];
     DebugPrint["... permuting bases"];
     result = disordered.
       PermuteDirectSummands[Γ][domain[[disordering]], β, Ordering[disordering]];
     DebugPrintHeld["Finished ", DecompositionMap[Γ, (U⊗V)⊗W, β]];
     result
    ];
   PackageDecompositionMaps[Γ];
   result
  ]
 ]
```

```
InverseDecompositionMap[Γ_, V : Irrep[_][_], β_] := Inverse[DecompositionMap[Γ, V, β]]
InverseDecompositionMap[Γ_, V : (_ ⊕ _), β_] := Inverse[DecompositionMap[Γ, V, β]]
```

```
InverseDecompositionMap[Γ_, V : (_ ⊗ _), β_] :=
 If[LoadDecompositionMaps[Γ, V], InverseDecompositionMap[Γ, V, β],
  Module[{},
   DebugPrintHeld["Beginning ", InverseDecompositionMap[Γ, V, β]];
   InverseDecompositionMap[Γ, V, β] = Inverse[DecompositionMap[Γ, V, β]];
   DebugPrintHeld["Finished ", InverseDecompositionMap[Γ, V, β]];
   PackageDecompositionMaps[Γ];
   InverseDecompositionMap[Γ, V, β]
  ]
 ]
```

```
If[$VersionNumber ≥ 6.,
 BlockMatrix[b : {{___Matrix} ...}] := AppendColumns @@ (AppendRows @@ # & /@ b)
]
```

```
BlockPermutationMatrix[permutation : {__Integer}, blocksizes : {__Integer}] :=
 BlockMatrix[Table[Table[If[permutation[[j]] == i, identityMatrix[blocksizes[[i]]],
     ZeroesMatrix[blocksizes[[permutation[[j]]]], blocksizes[[i]]]],
    {i, 1, Length[blocksizes]}], {j, 1, Length[permutation]}]]
```

```
PermuteDirectSummands[Γ_][V_DirectSum, β_, p_] :=
 RepresentationTensor[Γ, V[[p]], β, V, β, {#, BlockPermutationMatrix[p,
      Function[{W}, WeightMultiplicity[Γ, W, #]] /@ (List @@ V)]} & /@ Weights[Γ, V]]
```

```
DirectSumProjectionMap[Γ_, V_DirectSum, k_, β_] := RepresentationTensor[Γ, V[[k]], β, V, β,
  {#, QuantumGroups`MatrixPresentations`Private`DirectSumProjection[Γ][V, k, #]} & /@
   Weights[Γ, V]]
DirectSumInclusionMap[Γ_, V_DirectSum, k_, β_] := RepresentationTensor[Γ, V, β, V[[k]],
  β, {#, QuantumGroups`MatrixPresentations`Private`DirectSumInclusion[Γ][V, k, #]} & /@
   Weights[Γ, V]]
```

```
End[];
```

End of package