inv.m

Do not import this package directly. `inv.m` memoise many values automatically and is imported or cleared by `group.m`.

- Symmetry
- Clebsch-Gordan Coefficients
- Symmetries of CG Coefficients
- Conformal Blocks
- OPE Coefficients
- Bootstrap Equations
- Conversions of Bootstrap Equations

Symmetry

`symmetryGroup`

`symmetryGroup` represents the global symmetry of CFT.

`clearCG`

`clearCG[]` clears all values calculated by this package.

Clebsch-Gordan Coefficients

irrep-object

An irrep-object `r` is a symbol such that `symmetryGroup[isrep[r]]` is True. We use `r,s,t,r1,r2,...` to denote irrep-objects, and `id` is the irrep-object of the trivial irrep. `a,b,c,a1,a2,...` are indices of irreps. An index `a` of irrep `r` must sutisfy `1 <= a <= symmetryGroup[dim[r]]`. `n,m` represents multiplicity of Clebsch-Gordan coefficients. For more information, please refer to `arXiv:1903.10522`.

`inv`

`inv[r,s,t]` gives multiplicity of `id` in decomposition of `r \[TensorProduct] s \[TensorProduct] t`.

`inv[{r,s},{t}]` gives multiplicity of `t` in decomposition of `r \[TensorProduct] s`.

`inv[r1,r2,r3,r4]` gives an association whose key is a irrep `s` such that `inv[{r1,r2},{s}]>0` and `inv[r3,r4,s]>0`, whose value is a list `{inv[{r1,r2},{s}], inv[r3,r4,s]}`.

`invs`

`invs[r1,r2,r3,r4]` gives a list of `{s,n,m}` such that `cor[r1,r2,r3,r4][s,n,m]` is defined.

`ope`

`ope[r,s,t][n][a,b,c]` gives Clebsch-Gordan coefficient defined by `|r,a\[RightAngleBracket] \[TensorProduct] |s,b\[RightAngleBracket] = \[Sum] ope[r,s,t][n][a,b,c] |t,c,n\[RightAngleBracket]`.

`ope[r][a,b]` gives `Sqrt[dim[r]] ope[r,dual[r],id][1][a,b,1]`.

`cor`

`cor[r,s,t][n][a,b,c]` gives `\[Sum] ope[r,s,dual[t]][n][a,b,c2] ope[dual[t]][c2,c] / Sqrt[dim[t]]`.

`cor[r1,r2,r3,r4][s,n,m][a1,a2,a3,a4]` gives`\[Sum] cor[r1,r2,dual[s]][n][a1,a2,b] ope[r3,r4,dual[s]][a3,a4,b].`

Symmetries of CG Coefficients

`\[Sigma]`

`\[Sigma][r,s,t][n]` gives the sign change in swap between `r` and `s` in `cor`, i.e. `cor[r,s,t][n][a,b,c] = \[Sigma][r,s,t][n] cor[s,r,t][n][b,a,c]`.

`\[Sigma][r]` gives `\[Sigma][r,dual[r],id]`.

`\[Sigma][op[x,r,p,q]]` gives `p`.

`\[Tau]`

`\[Tau][r,s,t][n,m]` describes the behavior of `cor` under cyclic permutation of `r,s,t`, i.e. `cor[s,t,r][m][b,c,a] = \[Sum] cor[r,s,t][n][a,b,c] \[Tau][r,s,t][n,m]`.

`\[Omega]`

`\[Omega][r,s,t][n,m]` describes the behavior of `cor` under complex conjugate, i.e. `\[Sum] ope[dual[r]][a2,a] ope[dual[s]][b2,b] ope[dual[t]][c2,c] cor[dual[r],dual[s],dual[t]][m][a2,b2,c2]\[Conjugate] = \[Sum] cor[r,s,t][n][a,b,c] \[Omega][r,s,t][n,m]`.

`six`

`six[r1,r2,r3,r4][s,n,m,t,k,l]` describes the behavior of `cor` under swap between `r2` and `r4`, i.e. `cor[r1,r4,r3,r2][t,k,l][a1,a4,a3,a2] = \[Sum] cor[r1,r2,r3,r4][s,n,m][a1,a2,a3,a4] six[r1,r2,r3,r4][s,n,m,t,k,l]`.

`isReal`

`isReal[r]` gives whether `r` is a real representation or not.

`isComplex`

`isComplex[r]` gives whether `r` is a complex representation or not.

`isPseaudo`

`isPseaudo[r]` gives whether `r` is a pseaudo real representation or not.

Conformal Blocks

`op`

`op[x,r,p,q]` represents a primary operator object O whose name is `x`, which belongs to irrep `r`, which sign (=`\[Sigma][O]`) is `p` and whose parity of spin is `q` (=`±1`). If O appears in summation, we use 'op' as its name.

`op[x,r]` is a short-hand for `op[x,r,1,1]`.

`op[x]` is a short-hand for `op[x,r,1,1]` if you registered `op[x,r,1,1]` as a fundamental scalar.

`dualOp`

`dualOp[op[x,r,p,q]]` gives dual operator object of `op[x,r,p,q]`.

`format`

`format[eq]` gives readable format of `eq` with little loss of information. We need much redundancy to calculate properly, so formatted value cannot be used for any argument of our function. `format[eq]` is assumed to be used only for human-readability of last output.

`sum`

`sum[x,op[op,r,1,q]]` represents sum of `x` over all intermediate primary operator `O=op[op,r,1,q]` which belongs to irrep `r` and whose parity of spin is `q`. `sum[...]` is automatically expanded so as `x` to be `(F or H)*\[Beta]^2`.

`single`

`single[x]` represents `x`. We need to wrap `x` for redundancy. `single[...]` is automatically expanded so as `x` to be `(Fp or Hp)*\[Beta]^2`, or `(Fp or Hp)`.

`F`

`F[o1,o2,o3,o4,s,p]` represents generalized conformal block `F_{p,s}^{1,2,3,4}`, where `o1,...,o4` are primary scalars.

`F[a,b,c,d]` represents normal conformal block of type-F.

`H`

`H[a,b,c,d]` represents normal conformal block of type-H.

`Fp`

`Fp[o1,o2,o3,o4,o]` represents generalized conformal block `F_{o}^{1,2,3,4}`, where `o1,...,o4` and intermediate `o` are primary scalars.

`Fp[a,b,c,d,o]` represents normal conformal block of type-F with intermediate `o`.

`Hp`

`Hp[a,b,c,d,o]` represents normal conformal block of type-H with intermediate `o`.

OPE Coefficients

`\[Lambda]`

`\[Lambda][o1,o2,o3][n]` gives `n`-th OPE coefficient of `o1\[Times]o2 \[Rule] OverBar[o3]`.

`\[Alpha]`

`\[Alpha][o1,o2,o3][n]` gives `n`-th coefficient of three-point function `\[LeftAngleBracket]o1 o2 o3\[RightAngleBracket]`.

`\[Mu]`

`\[Mu][o1,o2,o3][n]` gives n-th OPE coefficient of `o1\[Times]o2 \[Rule] OverBar[o3]`, where `o3` is registered as a fundamental scalar.

`\[Nu]`

`\[Nu][o1,o2,o3][n]` gives `n`-th coefficient of three-point function `\[LeftAngleBracket]o1 o2 o3\[RightAngleBracket]`, where `o3` is registered as a fundamental scalar."

`\[Beta]`

`\[Beta][o1,o2,o3][n]` gives minimal basis to describe `\[Lambda]`, `\[Alpha]`, etc.

`\[Lambda]`, `\[Alpha]`, `\[Mu]` and `\[Nu]` are linear combination of `\[Beta]`.

Bootstrap Equations

`eqn`

`eqn[{a,b,...}]` represents bootstrap equation that claims `a,b,...` must equals to `0`, where `a,b,...` are real-linear combination of sum and single.

`eqn[sec,{a,b,...}]` represents extracted part of `eqn[{...}]` which contains only `sum` or `single` related to `sec`.

`bootAll`

`bootAll[ops]` generates bootstrap equation from all four-point functions of `ops`.

`bootAll[]` generates bootstrap equation from all four-point functions of fundamental scalars.

`setOps`

`setOps[ops]` registers `ops` and duals of `ops` as fundamental scalars.

Fundamental scalars are used to seperate sum of conformal blocks over scalars to `single[...]` and `sum[...]`.

`one`

`one` represents the unit operator. This is implicitly registered as a fundamental scalar.

`extract`

`extract[x,op[op,r,1,p]]` extracts terms of the form `sum[...,op[op,r,1,p]]` from `x`.

`extract[x,scalar]` extracts terms of the form `single[(Fp or Hp)*\[Beta]^2]` from `x`.

`extract[x,unit]` extracts terms of the form `single[Fp or Hp]` from `x` (contribution of the unit operator).

`extract` is a projection: `x == extract[x,unit] + extract[x,scalar] + \[Sum]_{r,p} extract[x,op[op,r,1,p]]` and `extract[x,sec] == extract[extract[x,sec]]`.

`unit`

`unit` is a option for `extract` and means contribution of unit operator.

`scalar`

`scalar` is a option for `extract` and means contribution of fundamental scalars but unit operator.

`sector`

`sector[eq]` gives a list of all nontrivial option `sec` for extract applied to `eq`, i.e. `{sec | extract[eq,sec] != 0}`.

Conversions of Bootstrap Equations

`makeG`

`makeG[eqn[sec,{a,b,...}]]` gives an undirected graph whose vertices are OPE coefficients `\[Beta]` in extracted bootstrap equation `eqn[sec,{a,b,...}]`.

`makeMat`

`makeMat[eqn[sec,{a,b,...}]]` gives a matrix-representation of extracted bootstrap equation `eqn[sec,{a,b,...}]`.

`makeSDP`

`makeSDP[eqn[{a,b,...}]]` converts whole bootstrap equation `eqn[{a,b,...}]` into sdp-object.

`sdpobj`

`sdpobj[secs,scalarnum,vals,mats]` is a sdp-object. `secs` is section data of bootstrap equation. `scalarnum` is the number of connected components in scalar sections. `vals` are real constants in bootstrap equation. `mats` are matrix-representation of bootstrap equation.

`toQboot`

`toQboot[sdp]` converts sdp-object into `c++` code for `qboot`.

`toCboot`

`toCboot[sdp]` converts sdp-object into `python` code for `cboot`.

`toTeX`

`toTeX[eq]` gives `LaTeX` string of `eq` (you need call `Print[toTeX[eq]]` to paste to your `tex` file).

`toTeX[eqn[{a,b,...}]]` gives `LaTeX` string of `eq` with `align` environment (you need call `Print[toTeX[eq]]` to paste to your `tex` file).

`repToTeX`

`repToTeX[r]` is needed to transrate irrep-object `r` as `LaTeX` string. Please set appropriate value.

`opToTeX`

`opToTeX[o]` is needed to transrate operator name `o` as `LaTeX` string. Please set appropriate value.