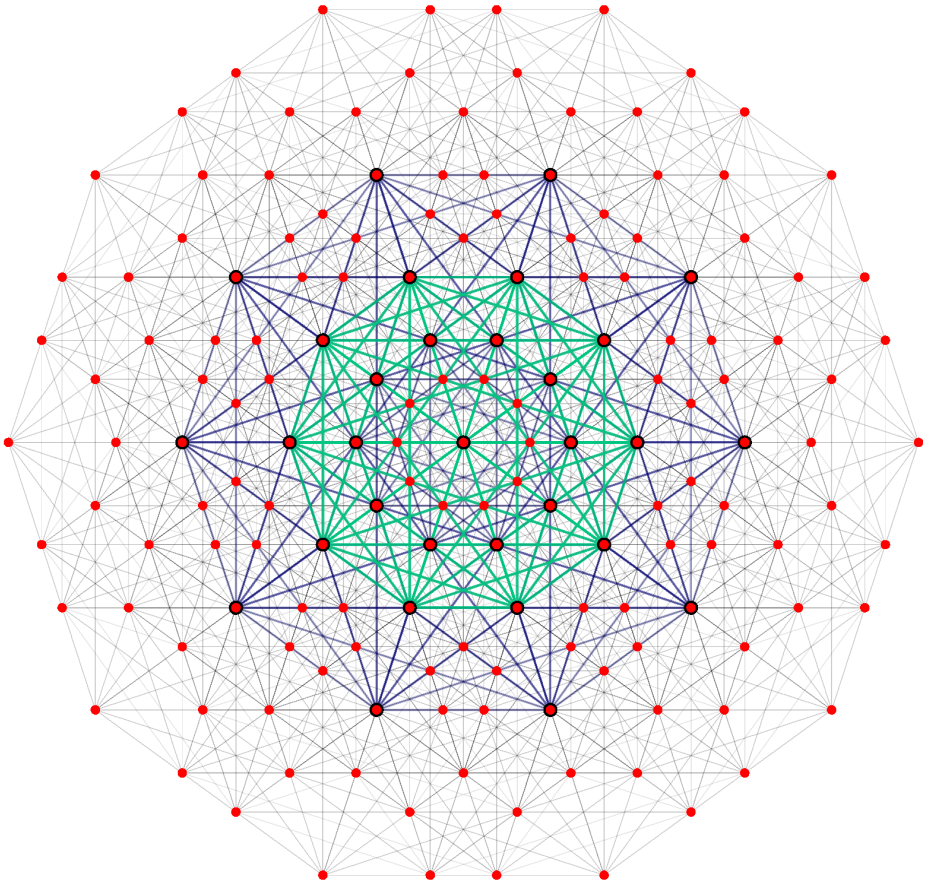
SimpleGroups: *Mathematica* package on (semi-)simple Lie groups

Alexey A. Vladimirov
Institute for Theoretical Physics II, Ruhr University Bochum, 44780 Bochum, Germany

We present the *Mathematica* package SimpleGroups, designed for the light computations in simple or semi-simple Lie groups. It contains various commands for dealing with roots and weights, weights diagrams, etc. In particular, it can decompose the given representation of groups over the representations of subgroups. It can be used for...



I. INTRODUCTION

We present the package for practical calculations in the (semi-)simple Lie groups. The package is programmed on the *Mathematica* language. And can be run on *Mathematica* 7.0 and higher. The package contains the commands to deal with roots systems, weight diagrams for any simple groups and their products. It contains the commands for calculation of Clebsch-Gordan decomposition, as well as, decomposition of representation onto representations of subgroups. It can calculates the Clebsch-Gordan coefficient for SU-group. Also it contains several commands for visualizing the results.

There are many packages and programmes for the group theory. Some of the are comprehensive, e.g. the programm LIE [1]. Some are devoted to particular aspects of the group theory calculations. What is the reason to produce one more programm for group calculation?

First of all, the package is based on the *Mathematica*, which is reference calculation system for the main part of physicists. One does not need to learn the interpretation language or use external compilations. The *Mathematica* is not the most efficient language, but it is the payment for the simpleness and universality of interpretator. The usage of *Mathematica* also allows to use the results for later calculations, without special transferring. Second, we try to use more common (for physicist) notations, and put several commands for transferring weights and indices from one notation to another. Third, we collect together algorithms, which can be found only separately.

One of the main feature of `SimpleGroups` is the usage of the Gelfand-Tsetlin (Gelfand-Tsetlin patterns, GT-patterns) state-notation for the calculation. The GT-patterns are the patterns for unique enumeration of the states within multiplet. They was introduced in ref. [2] for the SU and SO -groups, and recently revised in [**?**], for all classical series. The algorithm based on GT-patterns are much more efficient then classical ones. Since the GT-patterns are not very well-known in physics society we collect main rules in appendix.

The structure of this maintenance article is following. In the sect.II we shortly summarize all definitions which we used. The sect. III contains description of the general package structure, descriptions of some commands with many examples. The main information and list of all variables, commands and options can be found in tables 1,2,3, in text. In the appendix we put the main expressions for calculation with GT-patterns.

II. SIMPLE LIE GROUPS BASICS

The Lie group is a finite group which generators T_i form the algebra under commutation,

$$[T_i, T_j] = {c_{ij}}^k T_k, \tag{2.1}$$

where ${c_{ij}}^k$ are complex numbers called structure constants. The number of group generators called *dimension* of the algebra. The Lie group is *simple* if it is non-Abelian and ideal of its algebra is trivial or algebra it-self. The Lie group is *semisimple* if its algebra has non-zero ideal. Thus, the Lie algebra of direct product if two simple groups is semisimple.

The convenient way to deal with the algebra is to present it in the standard form. In the standard form the generators of the algebra are split onto two parts: *Cartan* generators H_i, *shifting* E_α generators. The Cartan generators are the maximum set of generators which commute with each other:

$$[H_i, H_j] = 0. \tag{2.2}$$

The number of the Cartan generators called *rank* of the group and denoted by l. The shifting generators can be choused in the such way that:

$$[H_i, E_\alpha] = \alpha_i E_\alpha, \tag{2.3}$$

where α_i is an element of the *root vector*, or just *root*, $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_l\}$. The standard form of the algebra is:

$$[H_i, H_j] = 0, \tag{2.4}$$
$$[\boldsymbol{H}, E_\alpha] = \boldsymbol{\alpha} E_\alpha, \tag{2.5}$$
$$[E_\alpha, E_{-\alpha}] = \boldsymbol{\alpha} \cdot \boldsymbol{H}, \tag{2.6}$$
$$[E_\alpha, E_\beta] = N_{\alpha\beta} E_{\alpha+\beta} \quad \text{if } \boldsymbol{\alpha} + \boldsymbol{\beta} \text{ is a root}, \tag{2.7}$$
$$[E_\alpha, E_\beta] = 0 \quad \text{otherwise.}$$

The set of all roots called *root system*, Φ. It satisfies the following properties:

- Φ is finite, and span the euclidian space and does not contain 0.
- If $\boldsymbol{\alpha} \in \Phi$, then the only aliquot root to it is $-\boldsymbol{\alpha} \in \Phi$.
- If $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \Phi$ then $\langle \beta\alpha \rangle = \frac{2\boldsymbol{\beta}\cdot\boldsymbol{\alpha}}{\boldsymbol{\alpha}\boldsymbol{\alpha}}$ is integer.
- The reflection operation $\sigma_\alpha(\boldsymbol{\beta}) = \boldsymbol{\beta} - \boldsymbol{\alpha}\langle\beta\alpha\rangle$ leaves Φ invariant for all $\boldsymbol{\alpha} \in \Phi$.

The given root system completely defines the algebra. Approximately all practical tasks related to Lie groups can be reformulated in terms of roots.

One can chouse l linear-independent roots $\boldsymbol{\alpha}_i (i = 1..l)$ to be the basis Δ in the root space. These roots are called *simple* root. Every root from Φ can be written as $\boldsymbol{\beta} = \sum_{i=1}^{l} k_i \boldsymbol{\alpha}_i$, where coefficients k_i are simultaneously non-negative or non-positive integers. If all k_i are non-negative (non-positive) the root $\boldsymbol{\beta}$ is called *positive*, $\boldsymbol{\beta} \succ 0$ (*negative*, $\boldsymbol{\beta} \prec 0$). Thus the root system is partially ordered by comparison $\succ$, one says that $\boldsymbol{\beta} \succ \boldsymbol{\alpha}$ if $\boldsymbol{\beta} - \boldsymbol{\alpha} \succ 0$.

There are two often used elements belongs to the root system. The root $\boldsymbol{\beta} = \sum_{i=1}^{l} k_i \boldsymbol{\alpha}_i$ with coefficients $k_i \geqslant p_i$, $\boldsymbol{\gamma} = \sum_{i=1}^{l} p_i \boldsymbol{\alpha}_i$ for any $\boldsymbol{\gamma}$, is called *maximal* root. The vector $\boldsymbol{\delta} = \frac{1}{2}\sum_{\alpha\succ 0} \alpha$ is called *Weyl* vector.

The set of simple roots completely defines the root system. Let us chouse the set of simple roots $\{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, .., \boldsymbol{\alpha}_l\}$. The matrix C with matrix element $C_{ij} = \langle \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j \rangle$ is called *Cartan matrix*. Cartan matrix depends on the order of simple roots, is independent on the choice of the simple roots. Cartan matrix gives comprehensive information about algebra. From the definition of the root system comes out that the only elements of Cartan matrix are $0, \pm 1, \pm 2, \pm 3$.

It is very convenient to present the Cartan matrix in form of the *Dynkin diagram*. The vertex of Dynkin graph denotes the simple root. Two vertices ($\boldsymbol{\alpha}_i$, $\boldsymbol{\alpha}_j$) are connected by $C_{ij}C_{ji}$(no summation) links. If there are more then two links the arrow point to shorter root.

There is an important theorem on classification which states that there are finite number of Dynkin graph topologies, and, therefore, there are only finite number of simple-algebra types. There are four series of algebras A_l, B_l, C_l and D_l, and five exceptional algebras E_6, E_7, E_8, F_4 and G_2. The relation of these algebras with special algebras are the following: $A_l = \mathfrak{su}(l+1)$, $B_l = \mathfrak{so}(2l+1)$, $C_l = \mathfrak{sp}(2l)$ and $D_l = \mathfrak{so}(2l)$.

The realization of these systems can be found in many books, particulary we use the "canonical" realization by Bourbaki [4]. Although it is not intuitive and sometimes uses bigger dimension target space, but it has rather simple form and easier to program.

A *representation* of a group is a homomorphism of group onto a group of linear operators acting on a linear vector space V. The dimension of V is called the *dimension of the representation*.

Every state in V can be unique (up to possible degeneration coming from other symmetries) labeled by eigenvalues of Cartan generators:

$$\boldsymbol{H}|V_\mu\rangle = \boldsymbol{\mu}|V_\mu\rangle. \tag{2.8}$$

The vector of eigenvalues $\boldsymbol{\mu}$ is called the *weight* of the state. All weights form the weight lattice in the root space such that $\langle \boldsymbol{\mu}, \boldsymbol{\alpha} \rangle$ =integer, for any root α. The basis on the lattice is usually choused to be dual to simple roots: $\langle \boldsymbol{\lambda}_i, \boldsymbol{\alpha}_j \rangle = \delta_{ij}$. These weights are called *fundamental weights* and expressed in through the Cartan matrix as follows:

$$\boldsymbol{\lambda}_i = (C^{-1})_i{}^k \boldsymbol{\alpha}_k. \tag{2.9}$$

The coefficients of decomposition of a weight over fundamental weights are called *Dynkin indices*, $\boldsymbol{\mu} = \sum_{i=1}^{l} k_i \boldsymbol{\lambda}_i$. The set of weights is partially ordered by comparison $\succ$ similar to the root space.

The shifting operator E_α transforms the state V_μ to $V_{\mu+\alpha}$: $E_\alpha V_\mu = V_{\mu+\alpha}$. Thus the shifting generators corresponded to the positive (negative) roots are called *rising*(*lowering*) generators. There is the state in V which is annihilated by all rising generators. The weight corresponded to this state is called the *eldest* weight of the representation. It completely defines the representation, and the common practice is to denote the representation by writing the Dynkin indices of its eldest weight. The eldest weights of the *fundamental* representations are the fundamental weights.

The dimension of the representation can be calculated using the Weyl formula:

$$\mathrm{dim}V = \prod_{\alpha\succ 0} \frac{(\boldsymbol{\delta} + \boldsymbol{\lambda}) \cdot \boldsymbol{\alpha}}{\boldsymbol{\delta} \cdot \boldsymbol{\alpha}}, \tag{2.10}$$

where $\boldsymbol{\delta}$ is Weyl vector, $\boldsymbol{\lambda}$ is the eldest weight of the representation and $\boldsymbol{\alpha}$ are positive root.

There are many ways to construct the weight diagram of the representation. One of the most efficient is to use the Freidantel formula:

$$\left((\boldsymbol{\lambda} + \boldsymbol{\delta}) \cdot (\boldsymbol{\lambda} + \boldsymbol{\delta}) - (\boldsymbol{\mu} + \boldsymbol{\delta}) \cdot (\boldsymbol{\mu} + \boldsymbol{\delta})\right) m(\boldsymbol{\mu}) = 2 \sum_{\alpha\succ 0} \sum_{i>0} m(\boldsymbol{\mu} + i\boldsymbol{\alpha})(\boldsymbol{\mu} + i\boldsymbol{\alpha}) \cdot \boldsymbol{\alpha}, \tag{2.11}$$

TABLE I: The global constants setting by SetGroup.

Name of constant	Description
`SimpleRoots`	The set of simple roots
`PositiveRoots`	The set of positive roots
`MaximalRoots`	The list of maximal roots
`FundamentalWeights`	The set of fundamental weights in the same order as simple roots
`WeylVector`	Weyl vector
`Rank`	The rank of group
`CoxeterNumber`	The Coxeter number of group

where $m(\boldsymbol{\mu})$ is multiplicity of the state with weight $\boldsymbol{\mu}$, $\boldsymbol{\lambda}$ is the eldest weight of the representation and $\boldsymbol{\delta}$ is the Weyl vector of the algebra. This relation with addition relations $m(\boldsymbol{\lambda}) = 1$ and $m(\boldsymbol{\lambda} + i\boldsymbol{\alpha}) = 0$ iteratively gives the whole weight diagram.

There are important problem in the Lie group theory: how to uniquely enumerate the states in the multiplet. Without the universal enumeration of the states inside the group many applications are senseless, e.g. the construction of Clebsch-Gordan coefficients. One of the possible solution is to use the semi-classical Young tableaux. However this is not very convenient because such notation relates more to the tensor construction of the group representation then to Cartan-Weyl construction. Also there are no commonly accepted version of Young tableaux for SO and SP group, see e.g. [6]. The alternative variant of the states enumerations is the usage of, so called, Gelfand-Tsetlin patterns (GT-patterns). They were introduces for the SU and SO-groups in 50-s in refs. [2]. And recently reconsidered for all classical groups by Mosev in [3]. The main idea of the GT-patterns consists (schematically) the following: one takes the set of groups with lowering rank (e.g. $SU(4) \to SU(3) \to SU(2) \to U(1)$) and enumerate the state by it belongness to the multiplet of every group in the chain. The main technical details on the GT-patterns are collected in the appendix.

Also GT-patterns provide very efficient method of the computation of weight diagrams. One can just write down all unforbidden (see appendix) GT-patterns with the given upper line, which is fixed by the eldest weight, and the result is all states from the given representation. In the package we use this algorithm everywhere where it is possible.

III. PACKAGE COMMANDS

A. Installation

The package is written on *Mathematica* 7, and we not guarantee the proper work of commands on earlier versions.

For installation of the package chouse **File→Install...** . In opened window set **Type of Item to Install:** Package, **Source:** the path to the file `SimpleGroups.m`. Into the line **Install Name** write the name of package (here we use the name SimpleGroups). Press the **Finish**-button.

Now the package will be automatically loaded after the calling command: `<<SimpleGroups‘` . It also shows the version of the package installed on your *Mathematica*. The latest version can be found on site: www.tp2.de.

B. General description and constants

The package SimpleGroups can be viewed as a library of roots and weights for the simple groups equipped with constructor and deconstructor of weight diagrams. Also it contains the commands for plotting the multiples.

Although one can apply the package commands to his/her-own realization of the root space, the primary intend is to use the builded-in "canonical" realization of [4]. Many commands are constructed in such a way that they exploit the fitches of that realization, such that absence of square roots, normalization, etc. The universal realization also allows to avoid repeated procedures and, therefore, speeds up calculations. To set the group under consideration use the command `SetGroup[]`.

The command `SetGroup[]` sets the global constants such as rank of the group, root system, fundamental weight, etc. The full list of the constants can be in table I.

The argument of the `SetGroup[]` can be `A[]`, `B[]`, `C[]`, `D[]`, `E[]`, `F[4]`, `G[2]`, `SU[]`,`SO[]`,`SP[]`, where in square brackets the rank stays, or their product e.g. `SU[2]SU[2]`. The letter is understood as the direct product of groups, therefore, the roots and weights constants would have the block-diagonal form. Note, that `MaximalRoot` in this case will return the list of maximal roots for every subgroup in product. We used the following correspondence in group notation:

$$\texttt{A[1]=B[1]=C[1]=D[1]}, \quad \texttt{D[2]=A[1]A[1]}, \quad \texttt{D[3]=A[3]},$$
$$\texttt{SU[}l\texttt{]=A[}l-1\texttt{]}, \quad \texttt{SP[}l\texttt{]=C[}l\texttt{]}, \quad \texttt{SO[}2l+1\texttt{]=B[}l\texttt{]}, \quad \texttt{SO[}2l\texttt{]=D[}l\texttt{]}.$$

The commands can be separated onto two categories, which use the global definition set by `SetGroup[]` and which not. Full list of commands in the package version 1.0 is given in table II, where we have used the usual *Mathematica* notation: `__` means that one can more then one argument of the type, `___` means that one can put zero or any number of arguments of the type. Some commands use an options which are used in usual way: `option->optionName`. The full list of used options is presented in Table III.

Most part of commands presented in table III are clear with out addition explanation. For the rest we give more detailed description in the next sections.

C. BuildRep, DynkinLabels & Weight

Since the package uses the explicit realizations of the weight spaces one of the main command is `BuildRep`. The command returns the weight diagram for the representation specified by the eldest weight.

There are thee options which can be used within `BuildRep`: `RepNotation`, `OutputMethod` and `NormalizationMethod`. The first is responsible for the reading of the argument and can be `"DynkinLabels"` (default) or `"EldestWeight"`. Note, for the `"EldestWeight"` the canonical realization should be used. The option `OutputMethod` is responsible for the format of the result. By definition the resulted weights are presented by their Dynkin labels (`"DynkinLabels"`), but they also can be presented as `"Weights"` or as `"GTPatterns"`. The last version available only for simple classical groups (A, B, C, D-series) and returns the Gelfand-Tsetlin patterns (see the details of pattern definitions in Appendix). With the option `OutputMethod->"Weights"` one can use the addition option `NormalizationMethod` which specifies the realization of the weight space the details can be found in the section D.

Often during the consideration representation properties one needs to switch between different notations. There are two commands which are responsible for such transitions. The first, `DynkinLabels`, makes transformation of weight vectors (or GT-patterns) to Dynkin indices. The second, `Weight`, makes transformation of the Dynkin indices (or GT-patterns) to weight vectors. The form of argument is specified by the option `InputMethod` (see table II for more details). Note, that the command `Weight` also has the option `NormalizationMethod` (see section D). Both command works on single weight or over list of weights or over weight diagram.

The main regimes of the commands is illustrated by the following program, which calculate and present the quark-triplet in different forms:

in `SetGroup[SU[3]]`

in `BuildRep[{1,0}]` (Dynkin indices of weights)

out `{{{0, -1}, 1}, {{-1, 1}, 1}, {{1, 0}, 1}}`

in `BuildRep[{1,0},OutputMethod->"Weights"]` (Weights in the canonical realization)

out `{{{-1/3, -1/3, 2/3}, 1}, {{-1/3, 2/3, -1/3}, 1}, {{2/3, -1/3, -1/3}, 1}}`

in `BuildRep[{1,0},OutputMethod->"Weights",NormalizationMethod->"Coxeter"]` (Weights in the `Coxeter` realization)

out $\{\{\{0, \frac{-1}{\sqrt{3}}\}, 1\}, \{\{\frac{1}{2}, \frac{1}{2\sqrt{3}}\}, 1\}, \{\{\frac{1}{2}, \frac{1}{2\sqrt{3}}\}, 1\}\}$

in `BuildRep[{1,0},OutputMethod->"GTPatterns"]` (Gelfand-Tsetlin patterns)

out
$$\begin{pmatrix} \{1,0,0\} & \{1,0,0\} & \{1,0,0\} \\ \{0,0\} & \{1,0\} & \{1,0\} \\ \{0\} & \{0\} & \{1\} \end{pmatrix}$$

D. PhysicalNormalization & PhysicalNormalizationMatrix

"Canonical" realization of the root spaces is very convenient for calculations and programming, economical for memory. But in most its part "canonical" realization has no "experimental" sense. The good example is familiar to everyone $SU(3)$ algebra. The rank of algebra is 2, and in the "canonical" realization simple roots have the form:

$$\alpha_1 = \{1, -1, 0\}, \quad \alpha_2 = \{0, 1, -1\}. \tag{3.1}$$

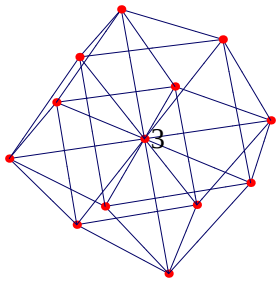


FIG. 1: Weight diagram of $\{1,0,1\}$ representation of $SU(4)$. Edges connects weight differs by positive roots.

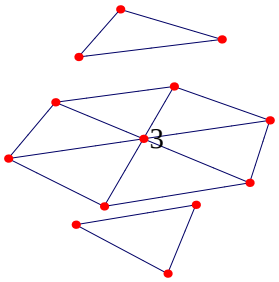


FIG. 2: Weight diagram of $\{1,0,1\}$ representation of $SU(4)$. Edges connects weight differs by positive roots of $SU(3)$ sub-group, based on α_1 and α_2 roots.

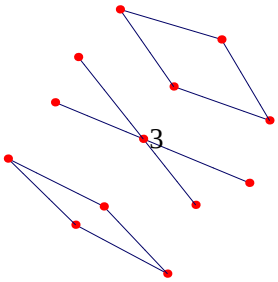


FIG. 3: Weight diagram of $\{1,0,1\}$ representation of $SU(4)$. Edges connects weight differs by positive roots of $SU(2)\times SU(2)$ sub-group, based on α_1 and α_3 roots.

The first fundamental weight is

$$\lambda_1 = \{\frac{2}{3}, -\frac{1}{3}, -\frac{1}{3}\},$$

and not everyone can recognize here the usual $(I_3, M) = \{\frac{1}{2}, \frac{1}{2\sqrt{3}}\}$ – the weight of u-quark.

To make the live easier we introduce the command `PhysicalNormalization`. This command rotates and re-scales (and also remove the extra-components of weights) the root space into more practical form. In new realization the first root is pointing into the 1-direction, the second root lies in the $(1,2)$-plane, the third in $(2,3)$-plane, etc. The length of roots is determined by the normalization condition $\sum_{\alpha\succ 0} \boldsymbol{\alpha}\cdot\boldsymbol{\alpha} = K$, where $K = \frac{h}{2}$ (h is the Coxeter number) by definition.

The normalization factor K can be changed by option `NormalizationMethod`. It takes three possible values `Coxeter`, `Unity` and `Half`, which satisfies to $K = \frac{h}{2}$, 1 and $\frac{1}{2}$.

The command can be applied to an individual weight and as well, to list of weights or to a weight diagram.

For the purpose of process clearness the rotation-scaling matrixes can be separately obtained by command `PhysicalNormalizationMatrix`, which arguments are: the name of algebra (`"A"`, `"B"`,..,`"G"`) and the rank. For example, the above transformation matrix was:

in `PhysicalNormalizationMatrix["A",2]`

out
$$\begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2\sqrt{3}} & \frac{1}{2\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{pmatrix}$$

E. DecomposeRep & CGDecomposition

During the investigation of multiplets intrinsic structure one often needs to extract the multiplets of sub-algebra. Also such task are often appears in GUT-theories, where one wants to destroy the multiplet of a "big" group onto multiplet of "small" ones. Although this question is very difficult in general form it is very easy (but not efficient) in explicit realization. Let us consider the illustrative example.

The algebra $SU(3)$ can be decomposed in several ways. For example, $SU(3)$ sub-group can be based on any doublet of roots connected by single line in Dynkin diagram (including maximal root). Let us decompose the adjoint ($\{1,0,1\}$) $SU(4)$-multiplet over $SU(3)$-multiplets. The $SU(3)$ subgroup we build on α_1, α_2 roots. The weights which are related to that subgroup should have the same third component. We plot the "stratified" weight diagram on fig.2. It is seen by naked eye that there are four sub-multiples, i.e. triplet(the upper plane), anti-triplet(the lower plane) and reducible composition of octet and singlet (in the middle plane). If we chouse as sub-group $SU(2)\times SU(2)$ built, say, on α_1 and α_3 roots, we will found the following picture (fig.3): There are two $(\frac{1}{2},\frac{1}{2})$-representations (the upper and lower planes), and reducible composition of $(1,0)$, $(0,1)$ and $(0,0)$ (in the middle plane).

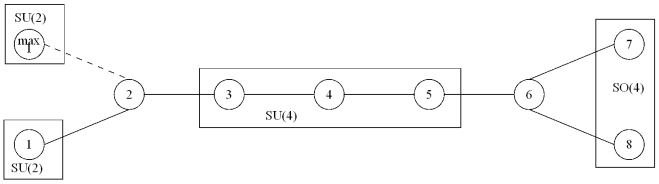


FIG. 4: Dynkin diagram for $SO(16)$ and the choice of roots for sub-group $SU(2) \times SU(2) \times SU(4) \times SO(4)$.

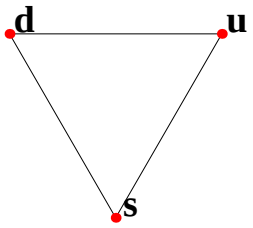


FIG. 5: One can plot any comments to weight with PlotRep command

The process of decomposition can be algorithmized in the following way. One builds the root system on the set of simple roots, and sorts all weights in the given weight diagram over elderness with respect to that system. Then one takes the eldest weight and subtracts the related multiplet, after that takes next eldest weight and subtracts its multiplet, and so on until the complete decomposition of the diagram. Although this algorithm is not very efficient it is only known unified way to make an arbitrary decomposition.

The described procedure is encoded in `RepDecompose` command, which takes two arguments. The first argument is precalculated weight diagram of representation to decompose. The second is the set simple roots which forms the sub-algebra. **Important note:** The input precalculated weight diagram must be in the weight vectors (with the same realization as simple root for decomposition).**Another important note:** There is no checks on the self-consistence of the set of simple roots, since there are too many possibilities of decompositions and forms. The evaluation of the command may take many time for large multiplet and large rank sub-groups. The result of the command is the list of the Dynkin indices of representations.

Let us present the example. The group $SO(16)$ contains $SU(2) \times SU(2) \times SU(4) \times SO(4)$ based on the following roots: $\{\{\alpha_1\}, \{M\}, \{\alpha_3, \alpha_4, \alpha_5\}, \{\alpha_7, \alpha_8\}\}$ (M is maximal root), see fig.4. The following commands would give the decomposition of the fundamental $\overline{\mathbf{128}}$ representation:

in `SetGroup[SO[16]]`

in `rep128=BuildRep[{0,0,0,0,0,0,0,1},OutputMethod->"Weights"];` (see important note)

in
```
SubSimpleRoots={SimpleRoots[[1]],MaximalRoots[[1]],SimpleRoots[[3]],SimpleRoots[[4]],
SimpleRoots[[5]],SimpleRoots[[7]],SimpleRoots[[8]]};
```

in `DecomposeRep[rep128,SubSimpleRoots]`

out
```
{{1,0,0,0,0,1,0},{1,0,1,0,0,0,1}, {0,1,0,0,0,0,1},{1,0,0,1,0,1,0},{0,1,1,0,0,1,0},
{1,0,0,0,1,0,1},{1,0,0,0,0,1,0},{0,1,0,1,0,0,1},{0,1,0,0,1,1,0},{0,1,0,0,0,0,1}}
```

Thus, we have:

$$\overline{\mathbf{128}}\Big|_{SO(16)} = 2\left[\frac{1}{2}\times 0\times \mathbf{1}\times\left(\frac{1}{2},0\right)\right]\oplus 2\left[0\times\frac{1}{2}\times\mathbf{1}\times\left(0,\frac{1}{2}\right)\right]\oplus\left[\frac{1}{2}\times 0\times\mathbf{4}\times\left(0,\frac{1}{2}\right)\right]\oplus \tag{3.2}$$

$$\left[0\times\frac{1}{2}\times\mathbf{4}\times\left(\frac{1}{2},0\right)\right]\oplus\left[\frac{1}{2}\times 0\times\overline{\mathbf{4}}\times\left(0,\frac{1}{2}\right)\right]\oplus\left[0\times\frac{1}{2}\times\overline{\mathbf{4}}\times\left(\frac{1}{2},0\right)\right]\oplus$$

$$\left[\frac{1}{2}\times 0\times\mathbf{6}\times\left(\frac{1}{2},0\right)\right]\oplus\left[0\times\frac{1}{2}\times\mathbf{6}\times\left(0,\frac{1}{2}\right)\right]\Bigg|_{SU(2)\times SU(2)\times SU(4)\times SO(4)}. \tag{3.3}$$

Note, that the decomposition of the weight diagram over the irreducible representations can be done using the same algorithm. The only change that the sub-algebra is algebra it-self. The command `CGDecomposition` returns the decomposition of product of representation over irreducible representations, i.e. Clebsch-Gordan decomposition. The argument is weight diagrams of representations to make the product, split by commas. Let us demonstrate it on the following textbook examples.

in `SetGroup[SU[3]]`

in `quarks = BuildRep[{1, 0}];`

in `antiquarks = BuildRep[{0, 1}];`

in `CGDecomposition[quarks, antiquarks]`

out `{{1,1}, {0,0}}`

in `CGDecomposition[quarks, quarks, quarks]`

out `{{3,0}, {1,1}, {1,1}, {0,0}}`

We restore the well-know $SU(3)$ formulae:

$$\mathbf{3} \otimes \overline{\mathbf{3}} = \mathbf{8} \oplus \mathbf{1}, \tag{3.4}$$

$$\mathbf{3} \otimes \mathbf{3} \otimes \mathbf{3} = \mathbf{10} \oplus \mathbf{8} \oplus \mathbf{8} \oplus \mathbf{1}. \tag{3.5}$$

The algorithms used inside the programme are based on the GT-patterns and therefore they are much faster for classical groups, and quite slow for the exceptional groups. For example, to restore the $E(8)$ formula:

$$\mathbf{248} \otimes \mathbf{248} = \mathbf{27000} \oplus \mathbf{30380} \oplus \mathbf{3875} \oplus \mathbf{248} \oplus \mathbf{1} \tag{3.6}$$

takes about 8 hours on the average laptop. Whereas the calculation of $SU(9)$ formula:

$$\mathbf{240} \otimes \mathbf{240} = \mathbf{10692} \oplus \mathbf{9240} \oplus \mathbf{4950} \oplus \mathbf{11088} \oplus \mathbf{11088} \oplus \mathbf{4620} \oplus \mathbf{2520} \oplus \mathbf{3402} \tag{3.7}$$

takes three and half seconds on the same laptop.

F. CGCoefficients

The Clebsch-Gordan (CG) coefficients are used to perform the explicit form of the direct product decomposition onto irreducible representations:

$$|r, s\rangle = \sum_{s_1, s_2} C^{r,s}_{s_1,s_2} |r_1, s_1\rangle |r_2, s_2\rangle, \tag{3.8}$$

where s, s_1, s_2 enumerates states within the corresponded multiplet (r, r_1, r_2). The CG coefficient is non-zero if the $w(s_1) + w(s_2) = w(s)$, where w is the weight vector of the state. The CG coefficients satisfies the following orthonormality conditions:

$$\sum_{s_1,s_2} C^{r,s}_{s_1,s_2} \left(C^{r',s'}_{s_1,s_2}\right)^* = \delta_{s,s'}\delta_{r,r'}, \qquad \sum_{r,s} C^{r,s}_{s_1,s_2} \left(C^{r,s}_{s'_1,s'_2}\right)^* = \delta_{s_1,s'_1}\delta_{s_2,s'_2}. \tag{3.9}$$

The CG coefficients play important role in the applications. But there are limited number of tables of CG coefficients, and programmes for generating them, except the case of most popular groups $SU(2)$ and $SU(3)$. Here, we present the command for the calculation of CG coefficients only for the SU groups.

In general the evaluation of the CG coefficients is not a very difficult algebraic task. The main problem points are the following: the universal enumeration of the states within the representation and the phase convention for eldest CG coefficients.

The first of these problems can be solved for the classical groups (i.e. of A, B, C and D types) by the notion of Gelfand-Tsetlin (GT) patterns, see refs [3]. The GT patterns provides the universal (within the type of algebra) form of state enumerating, i.e. every state has the unique patterns. The action of shifting operators can be also formulates throw the patterns (see same refs.). We collect the expressions, which we use with in the package in the appendix.

The second problem consist in the following: the CG coefficients are defined ambiguously up to the rotation which leaves conditions (3.9) invariant. Practically it is enough to fix the ambiguity for the any given (r, s). We do for the eldest state in r in the following way: fix the CG coefficient with the greater[1] $w(s_1)$ to be real and positive. The additional ambiguity arise in the case if there are N equivalent representations in the CG decomposition. Then one can rotate the states between these equivalent representations, $|r, s\rangle \to U_{rr'}|r', s\rangle$, where U is $N \times N$ unitary matrix. There are many ways to fix this ambiguity, we chouse the simplest. We take first N CG coefficients with the greatest

[1] We call the w_1 greater then w_2 if the first component of $w_1 - w_2$ (in the Dynkin indices) is positive (and w_1 less then w_2 if it is negative). If the the first component of $w_1 - w_2$ is zero, we compare the second and so on.

s_1, and for the r_i $(i = 1..N)$ put all of them, except i'th to be zero. Such convention is not convenient, but the user can perform the unitary rotation and chouse the convenient for him/her convention.

The algorithm is described with many details in ref. [5]. The algorithm contains two main parts. The first is to define the CG coefficients for the eldest states. For that one acts by the rising operators onto eldest states, and solve the homogenous system of linear equations together with normalization condition (3.9). Here, for the equivalent representation the N free parameters appear. We fix them and phase ambiguity as discussed above. The second part is consist in the definition of the rest CG coefficients. For that one takes the CG decomposition of the eldest state and act by the lowering operators, and again produce CG decomposition of unknown states. Such procedure results to the linear equation system which defines rest coefficients.

The command `CGCoefficients` takes two arguments and options. The arguments are the Dynkin labels of the representations r_1 and r_2. The result of calculation is saved to five variables `CGC`, `RulesForReps`, `RulesForStates`, `RulesForStates1` and `RulesForStates2`. The variable `CGC` contains the 4-index matrix $C[[r, s, s_1, s_2]]$ of CG coefficients, where the first index corresponds to the result representation r (the interpretation of the indices is in the variable `RulesForReps`), the second index corresponds to the state s in r (the interpretation of the indices is in the variable `RulesForStates`$[[r]]$), the third (fourth) index corresponds to the state $s_1(s_2)$ in r_1 (r_2) (the interpretation of the indices is in the variable `RulesForStates1`(`RulesForStates2`)). Also there is the visual output which form is controlled by option `OutputMethod`.

The option `OutputMethod` for `CGCoefficients` has three variants, `None`, `List`, `Tables`(default). Using this option one can regulate the additional output of `CGCoefficients` command (since the main output is the values of variables `CGC`, `RulesForReps`, etc). The variant `List` formats the output as the list of all non-zero CG coefficients with explicit form of states and their values. The variant `Tables` formats the output as the collection of tables for every representation and final state (rows are states of r_1 and columns are states of r_2).

If one does need all CG coefficients but only for one representation r, option `OnlyRep` can be used. It has the form of Dynkin indices for the desired representation. If there are several equivalent representations within decomposition all of them will be computed.

The example of application of the command can be found in the example files [7]

G. PlotRep2D & PlotRep3D

Often the simplest way to understand the properties of representation is to look at picture of its weight diagram. There are two commands in the package which draw weight diagrams: `PlotRep2D` and `PlotRep3D`, which plots the projections of weight diagrams onto 2-and 3- dimension plane, respectively. There are two obligatory arguments for these commands: The first is weight diagram (e.g. calculated by the `BuildRep`), the second is list of 2 (or 3) linearly independent vectors, which define the 2(or 3)-dimension plane.

The `PlotRep`-commands has many different options. Some of them are standard *Mathematica* options for graphics, these are `PointStyle LinkStyle MultiplicitiesStyle AxesStyle`, which, obviously, responsible for drawing style of vertices, edges, etc. Some of them just switch on/off the drawing of particular components, these are `ShowMultiplicities`, `ShowAxes`, `ShowLinks`. And the last option `LinksList` is responsible for the plotting of edges between weight-points. For example, the figures 1,-2,-3 differs only by this option. The program which gives these figures is following:

in `SetGroup[SU[4]]`

in `rep = BuildRep[{1, 0, 1}];`

in `PlotRep3D[rep, SimpleRoots]`

out see FIG.1

in `PlotRep3D[rep, SimpleRoots, LinksList->{SimpleRoots[[1]], SimpleRoots[[2]]}]`

out see FIG.2

in `PlotRep3D[rep, SimpleRoots, LinksList->{SimpleRoots[[1]], SimpleRoots[[3]]}]`

out see FIG.3

Note, that in this example the rank of the group is 3, therefore, the natural space for projection is formed by simple roots. Also, note, that by definition `LinksList` is `PositiveRoots`, that is why for the FIG.1 we did not put any options at all.

The first argument should be the list of form `{{{..},..},{{..},..},...}`, like one which is generated by `BuildRep` command. The multiplicities which are plot on the figure are the numbers which stand next to weight vector. For example, the figure 5 can be obtained by the following operations:

in `SetGroup[SU[3]]`

in `rep = BuildRep[{1, 0}];`

out `{{{2/3,-1/3,-1/3}, 1}, {{-1/3,2/3,-1/3},  1}, {{-1/3,-1/3,2/3}, 1}}`

in `newrep={{{2/3,-1/3,-1/3}, "u"}, {{-1/3,2/3,-1/3},  "d"}, {{-1/3,-1/3,2/3}, "s"}}`

in `PlotRep2D[newrep, SimpleRoots, ShowAxes -> False, MultiplicitiesStyle -> {Bold, 27}, PointStyle -> {Large, Red}]`

out see FIG.5

Another example is the picture on the title page, where plot the projection of the 352-plet of $SO(12)$ group onto the Coxeter plane. Also we marked out the fundamental 32-plet (blue) and 12-plet (green). The text of the program is following:

in `SetGroup[SO[16]]`

in `rep1 = BuildRep[{1,0,0,0,0,1}];` (this is the 352-plet)

in `rep2 = BuildRep[{0,0,0,0,0,1}];` (this is the 32-plet)

in `rep3 = BuildRep[{1,0,0,0,0,0}];` (this is the 12-plet)

in `pic1 = PlotRep2D[rep1, CoxeterPlane, ShowMultiplicities->False, PointStyle->{0.0001, Red}, ShowAxes->False, LinkStyle->{Thickness[0.001], Black, Opacity[0.15]}];`

in `pic2=PlotRep2D[rep2, CoxeterPlane, ShowMultiplicities->False, PointStyle->{0.015, Black}, ShowAxes->False, LinkStyle->{Opacity[0.3], Thickness[0.0025], RGBColor[0,0,0.5]}];`

in `pic3=PlotRep2D[rep3, CoxeterPlane, ShowMultiplicities->False, PointStyle->{0.015, Black}, ShowAxes->False, LinkStyle->{Opacity[0.8], Thickness[0.003], RGBColor[0, 0.8, 0.5]}];`

in `Show[pic1, pic2, pic3];`

out see title page.

Although this example is senseless from scientific point of view, but it demonstrate all main regimes of `PlotRep`-commands.

IV. CONCLUSION

APPENDIX A: GELFAND-TSETLIN PATTERNS

The Gelfand-Tsetlin patterns (GT-patterns) enumerate the states within the given multiplet. The presented here variant is the short compilation of some results the articles [3], rewritten in the form which is used in package.

1. $SU(n) = A_{n-1}$ type

The GT-pattern for the A_{n-1} type of group has the form:

$$\begin{array}{ccccccccc} \lambda_{n1} & & \lambda_{n2} & & \cdots & & \cdots & & \lambda_{nn} \\ & \lambda_{n-1,1} & & \lambda_{n-1,2} & & \cdots & & \lambda_{n-1,n-1} & \\ & & \cdots & & \cdots & & \cdots & & \\ & & & \lambda_{21} & & \lambda_{22} & & & \\ & & & & \lambda_{11} & & & & \end{array} \tag{A1}$$

where number in every neighbor lines satisfies "snake"-rule:

$$\lambda_{k,1} \geqslant \lambda_{k-1,1} \geqslant \lambda_{k,2} \geqslant \lambda_{k-1,2} \geqslant \ldots \geqslant \lambda_{k,k-1} \geqslant \lambda_{k-1,k-1} \geqslant \lambda_{k,k}.$$

The first line is reproduces the usual Young-like notation of the representation. It can be obtained from the Dynkin labels of the eldest weight as follows

$$(k_1, k_2, ..., k_{n-1}) \;\rightarrow\; [\sum_{j=1}^{n-1} k_j,\;\; \sum_{j=2}^{n-1} k_j, ...,\;\; k_{n-2}+k_{n-1},\;\; k_{n-1}, 0] \;=\; [\lambda_{n1}, \lambda_{n2}, \ldots, \lambda_{nn}]. \tag{A2}$$

In order to obtain the Dynkin indices for the state's weight one has to multiply the vector of numbers (sum of every line in pattern)

$$\{v_1, v_2, \ldots, v_n\} = \{\sum_{i=1}^{n} \lambda_{ni},\;\; \sum_{i=1}^{n-1} \lambda_{n-1,i}, \ldots, \lambda_{11}\},$$

by the $(n-1) \times n$ Cartan-like matrix:

$$\begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ k_{n-2} \\ k_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 & -1 & 2 \\ 0 & 0 & 0 & \ldots & -1 & 2 & -1 \\ 0 & 0 & 0 & \ldots & 2 & -1 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & -1 & 2 & \ldots & 0 & 0 & 0 \\ -1 & 2 & -1 & \ldots & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix}.$$

Thus, one sees that states are degenerate if they vectors v coincides.

The GT-patterns by construction contains information about the decomposition of the $SU(n)$ multiplet onto the multiplets belonging to the groups of the chain $SU(n) \to SU(n-1) \to SU(n-2) \to .. .$ Among all states enumerated by the pattern with the upper line λ_n all GT-patterns with the same $(k+1)$ upper lines belongs to the same multiplet of $SU(n-k)$. The eldest weight of the corresponded multiplet can be found from the line λ_{n-k}.

2. $SP(n) = C_n$ type

The GT patterns for the SP groups was introduced in ref. [**?**]. Here for convenience we change the definition of pattern: $(\lambda^{(')}_{k,i})_{[?\ ]} = (-\lambda^{(')}_{k,k-i+1})_{\text{here}}$.

The GT-pattern for the C_n groups has $2n$ lines:

$$\begin{array}{ccccccccccc}
\lambda_{n1} & & \lambda_{n2} & & \lambda_{n3} & & \ldots & & \lambda_{nn} \\
& \lambda'_{n1} & & \lambda'_{n2} & & & \ldots\ldots & & \lambda'_{nn} \\
& & \lambda_{n-1,1} & & \lambda_{n-1,2} & & \ldots & & \lambda_{n-1,n-1} \\
& & & \lambda'_{n-1,1} & & & \ldots & & \lambda'_{n-1,n-1} \\
& & & & & & \ldots\ldots\ldots & & \\
& & & & & & \lambda_{21} & \lambda_{22} & \\
& & & & & & & \lambda'_{21} & \lambda'_{22} \\
& & & & & & & & \lambda_{11} \\
& & & & & & & & & \lambda'_{11}
\end{array} \tag{A3}$$

where every $\lambda \geqslant 0$ and every two lines satisfies "snake"-rule:

$$\lambda_{k,1} \geqslant \lambda'_{k,1} \geqslant \lambda_{k,2} \geqslant \lambda'_{k,2} \geqslant \ldots \geqslant \lambda_{k,k} \geqslant \lambda'_{k,k} \geqslant 0,$$

$$\lambda'_{k,1} \geqslant \lambda_{k-1,1} \geqslant \lambda'_{k,2} \geqslant \lambda_{k-1,2} \geqslant \ldots \geqslant \lambda_{k-1,k-1} \geqslant \lambda'_{k,k}.$$

The upper line is the eldest weight of the representation in the canonical realization, alternatively it can be obtained from the Dynkin labels by following relation:

$$(k_1, k_2, ..., k_n) \;\rightarrow\; [\sum_{j=1}^{n} k_j,\;\; \sum_{j=2}^{n} k_j, ...,\;\; k_{n-1}+k_n,\;\; k_n] \;=\; [\lambda_{n1}, \lambda_{n2}, \ldots, \lambda_{nn}]. \tag{A4}$$

In order to obtain the Dynkin indices for the state's weight one has to multiply the vector of numbers (sum of every line in pattern)

$$\{v_1, v_1', v_2, \ldots, v_n'\} = \{\sum_{i=1}^{n} \lambda_{ni},\ \sum_{i=1}^{n} \lambda'_{ni},\ \sum_{i=1}^{n-1} \lambda_{n-1,i}, \ldots, \lambda'_{11}\},$$

by the $n \times (2n)$ matrix:

$$\begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix} = \begin{pmatrix} -1 & 2 & 0 & -2 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & -2 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_1' \\ v_2 \\ \vdots \\ v_n \\ v_n' \end{pmatrix}.$$

The GT-patterns by construction contains information about the decomposition of the C_n multiplet onto the multiplets belonging to the groups of the chain $C_n \to C_{n-1} \to C_{n-2} \to \ldots$. Among all states enumerated by the pattern with the upper line λ_n all GT-patterns with the same $(2k+1)$ upper lines belongs to the same multiplet of C_{n-k}. The eldest weight of the corresponded multiplet can be found from the line λ_{n-k}.

3. $SO(2n) = D_n$ type

The GT-pattern for the D_n groups has $2n-1$ lines:

$$\begin{array}{ccccccccccc} \lambda_{n1} & & \lambda_{n2} & & & \cdots\cdots & & & \lambda_{nn} \\ & \lambda'_{n-1,1} & & \lambda'_{n-1,2} & & \cdots & & \lambda'_{n-1,n-1} & \\ & & \lambda_{n-1,1} & & & \cdots & & \lambda_{n-1,n-1} & \\ & & & & & \cdots\cdots\cdots & & & \\ & & & & & \lambda_{21} & & \lambda_{22} & \\ & & & & & & \lambda'_{11} & & \\ & & & & & & & \lambda_{11} & \end{array} \tag{A5}$$

where every λ's are simultaneously integers or half-integers, and every two lines satisfies "snake"-rule:

$$\lambda_{k,1} \geqslant \lambda'_{k-1,1} \geqslant \lambda_{k,2} \geqslant \lambda'_{k-1,2} \geqslant \ldots \geqslant \lambda'_{k-1,k-1} \geqslant |\lambda_{k,k}| \geqslant 0.$$

$$\lambda'_{k,1} \geqslant \lambda_{k,1} \geqslant \lambda'_{k,2} \geqslant \lambda_{k,2} \geqslant \ldots \geqslant \lambda'_{k,k} \geqslant |\lambda_{k,k}| \geqslant 0.$$

The upper line is the eldest weight of the representation in the canonical realization, alternatively it can be obtained from the Dynkin labels by following relation:

$$(k_1, k_2, ..., k_n) \ \to\ [\sum_{j=1}^{n-2} k_j + \frac{k_{n-1}+k_n}{2},\ \sum_{j=2}^{n-2} k_j + \frac{k_{n-1}+k_n}{2}, ..., \frac{k_{n-1}+k_n}{2}, \frac{k_n - k_{n-1}}{2}] \ =\ [\lambda_{n1}, \lambda_{n2}, \ldots, \lambda_{nn}]. \tag{A6}$$

In order to obtain the Dynkin indices for the state's weight one has to multiply the vector of numbers

$$\{v_1, v_1', v_2, v_2' \ldots, v_n'\} = \{\sum_{i=1}^{n} \lambda_{ni},\ \sum_{i=0}^{n-1} \lambda'_{n-1,i},\ \sum_{i=1}^{n-1} \lambda_{n-1,i}, \sum_{i=0}^{n-2} \lambda'_{n-2,i} \ldots, \lambda'_{11} + \lambda'_{10}, \lambda_{11}, \lambda_{11}\},$$

where $\lambda'_{k,0} = \min(\lambda_{k+1,k+1}, \lambda_{k,k})$, by the $n \times (2n)$ matrix:

$$\begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix} = \begin{pmatrix} -1 & 2 & 0 & -2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & -2 & 1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -1 & 2 & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -1 & 2 & -2 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_1' \\ v_2 \\ \vdots \\ v_n \\ v_n' \end{pmatrix}.$$

The matrix is differs form those for C-case only by the last row.

The GT-patterns by construction contains information about the decomposition of the D_n multiplet onto the multiplets belonging to the groups of the chain $D_n \to D_{n-1} \to D_{n-2} \to .. .$ Among all states enumerated by the pattern with the upper line λ_n all GT-patterns with the same $(2k+1)$ upper lines belongs to the same multiplet of C_{n-k}. The eldest weight of the corresponded multiplet can be found from the line λ_{n-k}.

4. $SO(2n+1) = B_n$ type

The GT-pattern for the B_n groups has the form similar to C_n pattern but with addition column σ:

$$\begin{array}{ccccccccccc}
\lambda_{n1} & & \lambda_{n2} & & \lambda_{n3} & & \cdots & & \lambda_{nn} & & \sigma_n \\
& \lambda'_{n1} & & \lambda'_{n2} & & & \cdots\cdots & & & \lambda'_{nn} & \\
& & \lambda_{n-1,1} & & \lambda_{n-1,2} & & \cdots & & \lambda_{n-1,n-1} & & \sigma_{n-1} \\
& & & \lambda'_{n-1,1} & & & \cdots & & & \lambda'_{n-1,n-1} & \\
& & & & & & \cdots\cdots\cdots & & & & \\
& & & & & \lambda_{21} & & \lambda_{22} & & & \sigma_2 \\
& & & & & & \lambda'_{21} & & \lambda'_{22} & & \\
& & & & & & & \lambda_{11} & & & \sigma_1 \\
& & & & & & & & \lambda'_{11} & &
\end{array} \tag{A7}$$

where all λ simultaneously integers or half-integers, every σ is 0 or 1, and every two lines satisfies "snake"-rule:

$$\lambda_{k,1} \geqslant \lambda'_{k,1} \geqslant \lambda_{k,2} \geqslant \lambda'_{k,2} \geqslant \ldots \geqslant \lambda_{k,k} \geqslant \lambda'_{k,k} \geqslant [0 \text{ or } \sigma_k]\,,$$

$$\lambda'_{k,1} \geqslant \lambda_{k-1,1} \geqslant \lambda'_{k,2} \geqslant \lambda_{k-1,2} \geqslant \ldots \geqslant \lambda_{k-1,k-1} \geqslant \lambda'_{k,k},$$

where in the first line one chouse 0 for half-integer λ's, and σ_k for integer numbers.

The upper line is the eldest weight of the representation in the canonical realization, alternatively it can be obtained from the Dynkin labels by following relation:

$$(k_1, k_2, ..., k_n) \;\to\; [\sum_{j=1}^{n-1} k_j + \frac{k_n}{2},\; \sum_{j=2}^{n-1} k_j + \frac{k_n}{2}, ..., \frac{k_n}{2}] \;=\; [\lambda_{n1}, \lambda_{n2}, \ldots, \lambda_{nn}]. \tag{A8}$$

In order to obtain the Dynkin indices for the state's weight one has to multiply the vector of numbers (sum of every line in pattern)

$$\{v_1, v'_1, v_2, \ldots, v'_n\} = \{\sum_{i=1}^{n} \lambda_{ni},\; \sum_{i=1}^{n} \lambda'_{ni},\; \sum_{i=1}^{n-1} \lambda_{n-1,i}, \ldots, \lambda'_{11}\},$$

by the $n \times (2n)$ matrix (the matrix differs from C only by the last row) and subtract vector of σ's:

$$\begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix} = \begin{pmatrix} -1 & 2 & 0 & -2 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & -2 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -2 & 4 \end{pmatrix} \begin{pmatrix} v_1 \\ v'_1 \\ v_2 \\ \vdots \\ v_n \\ v'_n \end{pmatrix} - \begin{pmatrix} \sigma_1 - \sigma_2 \\ \sigma_2 - \sigma_3 \\ \sigma_3 - \sigma_4 \\ \vdots \\ \sigma_{n-1} - \sigma_n \\ 2\sigma_n \end{pmatrix}.$$

The GT-patterns by construction contains information about the decomposition of the B_n multiplet onto the multiplets belonging to the groups of the chain $B_n \to B_{n-1} \to B_{n-2} \to .. .$ Among all states enumerated by the pattern with the upper line λ_n all GT-patterns with the same $(2k+1)$ upper lines (do not taking into account the σ-column) belongs to the same multiplet of C_{n-k}. The eldest weight of the corresponded multiplet can be found from the line λ_{n-k}.

---

[1] M. A. A. van Leeuwen, A. M. Cohen and B. Lisser, *LiE, A Package for Lie Group Computations*, Computer Algebra Nederland, Amsterdam, ISBN 90-74116-02-7, 1992
http://www-math.univ-poitiers.fr/ maavl/LiE/

[2] I. M.Gelfand, M.L.Tsetlin, Dokl. Akad. Nauk SSSR **71**, 825828 (1950) (Russian).
English transl. in: Gelfand, I. M. Collected papers. Vol II (1988)
I. M.Gelfand, M.L.Tsetlin, Dokl. Akad. Nauk SSSR **71**, 10171020 (1950)(Russian).
English transl. in: Gelfand, I. M. Collected papers. Vol II, (1988)
[3] A.I. Molev, Handbook of Algebra, Volume 4, 2006, Pages 109-170 arXiv:math/0211289 [math.RT].
A. I. Molev, J. Phys. A: Math. Gen. **33** (2000), 4143-4168. arXiv:math/9909034v1 [math.QA]
A. I. Molev, Adv. Studies in Pure Math. **28** (2000), 221-240. arXiv:math/9902060v1 [math.RT]
A. I. Molev, Comm. Math. Phys. **201** (1999), 591-618. arXiv:math/9804127v1 [math.QA]
[4] Bourbaki vol.2
[5] A. Alex, M. Kalus, A. Huckleberry and J. von Delft, arXiv:1009.0437 [math-ph].
[6] M. Fischler, J. Math. Phys. **22** (1981) 637.
G. Girardi, A. Sciarrino and P. Sorba, J. Phys. A **16** (1983) 2609.
G. Girardi, A. Sciarrino and P. Sorba, Physica **114A** (1982) 365.
G. Girardi, A. Sciarrino and P. Sorba, J. Phys. A **15** (1982) 1119.
K. Koike, I. Terada, J.Algebra, **107**, 2, (1987), 466-511.
[7] The example notebooks and the lates version of the package can be found on: www.ruhr-uni-bochum.de/tp2/...

TABLE II: Full list of commands in version 1.0

Command[variables]	Usage of global variables	Description
`SetGroup[NameOfAlgerbra]`	—	Sets the global variables.
`CartanMatrix[SimpleRoots]`	No	Returns the Cartan matrix for the given set of simple roots.
`FundamentalWeight[SimpleRoots]`	No	Returns the system of fundamental weights for the given set of simple roots.
`RootSystem[SimpleRoots]`	No	Returns the complete system of roots for the given system of simple roots, in the form `{{PositiveRoots},{NegativeRoots}}`.
`DynkinDiagram`	Yes	Draw the affine Dynkin diagram for the group under consideration.
`CoxeterPlane`	Yes	Returns two vectors (in numerical form), which define the Coxeter plane in the roots space.
`RepDimension[rep,opts___]`	Yes	Returns the dimension of the group representation specified by argument. The argument can be Dynkin labels of the eldest weight(default) or eldest weight, which is regulated by the option `RepNotation`. It is based on Weyl formula (2.10)
`BuildRep[rep,opts___]`	Yes	Returns the weight diagram for the group representation specified by argument, in the form `{{{weight},degeneracy},...}`. The argument can be Dynkin labels of the eldest weight(default) or eldest weight, which is regulated by the option `RepNotation`. The algorithm is based on Freidantel formula (2.11).
`Representation[rep,SimpleRoots,opts___]`	No	Returns the weight diagram of the representation for group with given set of simple roots. The representation is specified by Dynkin indices of its eldest weight(default) or eldest weight, which is regulated by the option `RepNotation`. The algorithm is based on Freidantel formula (2.11).
`DynkinLabels[elem,SimpleRoots___]`*	Yes/No	Transforms the element of weight space to Dynkin labels with respect to the given simple roots. In the absence of the second argument uses current set group.
`Weight[elem,opts___]`*	Yes	Transforms the element of weight space to weight vector.
`PhysicalNormalizationMatrix[Group,Rank,opts___]`	No	Returns the rotation-rescaling matrix for the weight space from "canonical" realization of [4] of the root system to the physical notation (see sect.C).
`PhysicalNormalization[elem,opts___]`*	Yes	Rotates and normalizes the element of package from "canonical" realization of [4] to the physical notation (see sect.C).
`RepProduct[Rep1,Rep2]`	No	Returns the weight diagram for the Kronecker product of two representations. The arguments are weight diagrams of representations.
`DecomposeRep[Rep,SystemOfRoots,opts___]`	No	Decompose given multiplet with respect to the given roots. Returns the list of Dynkin indices for eldest weights of sub-multiplets. For details see sect.D.
`CGDecomposition[Rep1,Rep2__,opts___]`	Yes	Decompose the direct product of representations over irreducible representations. Returns the list of Dynkin indices for eldest weights of irreducible representations. For details see sect.D.
`CGCoefficients[rep1,rep2__,opts___]`	Yes	Returns the Clebsch-Gordan coefficients. Works only with SU (A) group. For details see sect.E.
`PlotRep2D[Rep,Plane,opts___]`	Yes/No	Plot the projection of the given weight diagram onto the plane defined by two vectors. For details see sect.F.
`PlotRep3D[Rep,Space,opts___]`	Yes/No	Plot the projection of the given weight diagram onto the 3D space defined by three vectors. For details see sect.F.

* Argument can be weight, list of weights or weight diagram. The command would act onto successful part of expression.

TABLE III: Full list of options used by commands

Option name	Applying commands	Format	Description
RepNotation	RepDimension BuildRep Representation	"DynkinLabels"(default)	The notation of representation is Dynkin labels of its eldest weight.
		"EldestWeight"	The notation of representation is its eldest weight.
InputMethod	CGDecomposition PhysicalNorma -lization Weight	"DynkinLabels"	The input multiplet weights are presented in Dynkin labels. This variant is default for CGDecomposition and Weight.
		"Weights"	The input multiplet weights are presented in canonical realization. This variant is default for PhysicalNormalization.
		"GTPatterns"	Only for Weight. The input weights are presented in the Gelfand-Tsetlin patterns. Available only if the set group is simple-classical group.
OutputMethod	BuildRep Representation	"DynkinLabels"(default)	The weights in the output presented by their Dynkin indices.
		"Weights"	The weights in the output presented by coordinates in the weight space. The realization of the weight space is fixed by "NormalizationMethod" option, by definition "Canonical" is used.
		"GTPatterns"	Only for BuildRep. The output states are presented as Gelfand-Tsetlin patterns. Available only if the set group is simple-classical group.
OutputMethod	CGCoefficients	"Tables"(default) "List" "None"	see sect. F for descriptions.
NormalizationMethod	PhysicalNorma -lization PhysicalNorma -lizationMatrix BuildRep Representation	"Coxeter"	The normalization of roots is $\sum_{\alpha\succ 0}\boldsymbol{\alpha}\cdot\boldsymbol{\alpha} = \frac{h}{2}$, where h is Coxeter number. This variant is default for PhysicalNormalization and PhysicalNormalizationMatrix.
		"Unity"	The normalization of roots is $\sum_{\alpha\succ 0}\boldsymbol{\alpha}\cdot\boldsymbol{\alpha} = 1$
		"Half"	The normalization of roots is $\sum_{\alpha\succ 0}\boldsymbol{\alpha}\cdot\boldsymbol{\alpha} = \frac{1}{2}$
		"Canonical"	The "canonical" realization of roots by [4]. This variant is default for BuildRep and Representation.
OnlyRep	CGCoefficients	{Dynkin indeces}	The CG coefficients calculates only for representation specified by ToRep. See sect.F for details.
LinksList	PlotRep2D PlotRep3D	{any weights}	The list of links to show on the weight diagram. By default PositiveRoots.
ShowMultiplicities	PlotRep2D PlotRep3D	True(default)/False	In the case of "True" subscribes the multiplicities on weight diagram.
ShowLinks	PlotRep2D PlotRep3D	True(default)/False	In the case of "True" draws the edges specified by LinksList on weight diagram.
ShowAxes	PlotRep2D PlotRep3D	True(def.2D)/False(def.3D)	In the case of "True" draws on the weight diagram the vectors given by the second argument.
PointStyle LinkStyle MultiplicitiesStyle AxesStyle	PlotRep2D PlotRep3D	See sect.G	Set styles for drawing the elements of weight diagram.