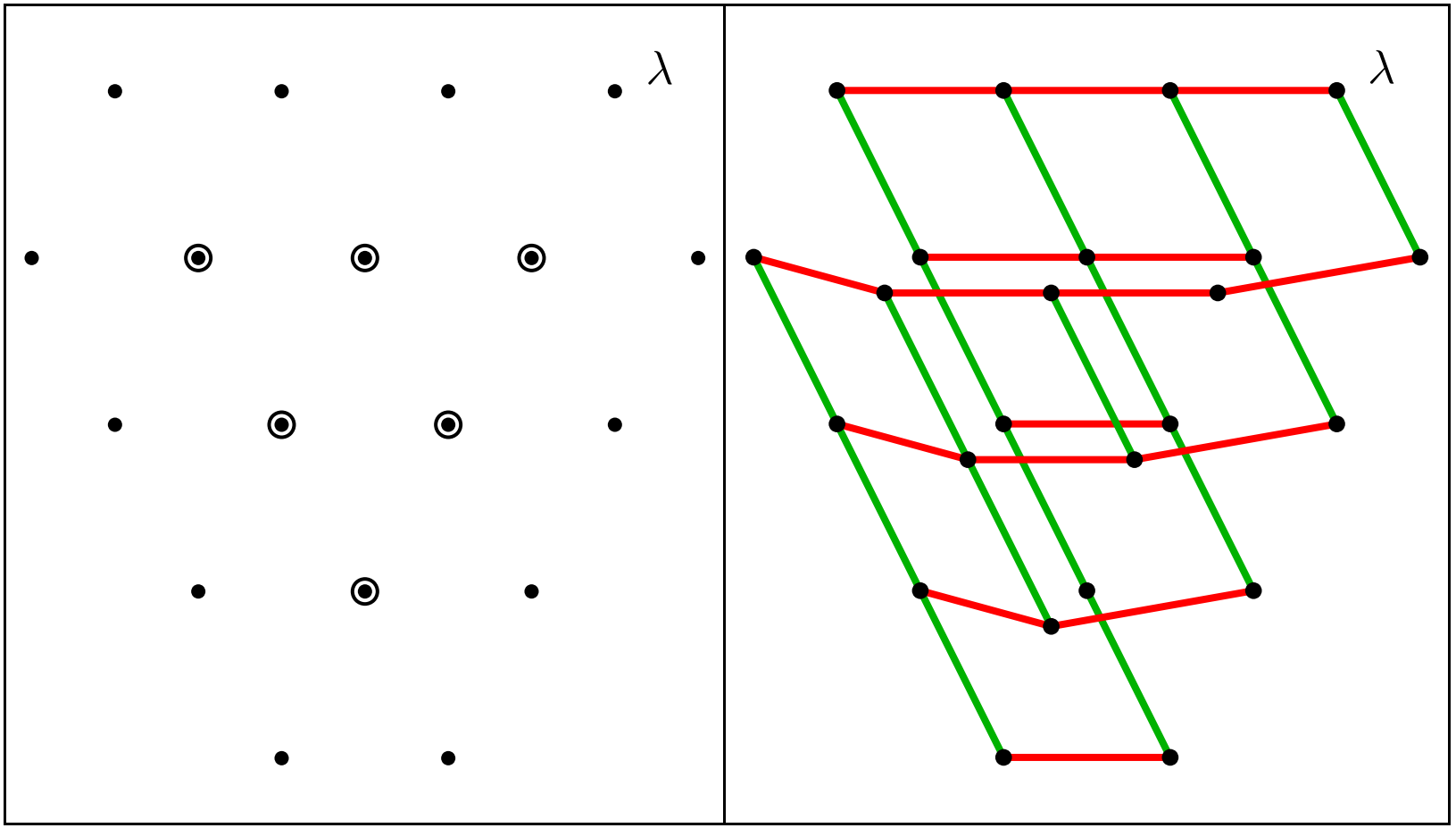
Lie Methods in Sage

by Daniel Bump



Follow this talk interactively

If you want to follow along interactively, grab the file `singalong.sage` from the web page:

`http://sporadic.stanford.edu/bump/sagedays/`

Then load the file with the command:

```
sage: iload singalong.sage
```

LiE

- `LiE` is high quality Lie Group software licensed LGPL, developed by Arjeh Cohen at CWI. **However it is not recently maintained.**
- `LiE` can do many things well, up to Kazhdan-Lusztig polynomials.
- Mike Hansen wrote an interface. It works fine.
- There is an **optional `SAGE` package** based on `LiE`.

On March 12, 2008 William Stein wrote:

> `The time stamps for all but one of the files are 8 - 12 years old! There's no way LiE -- as it is now -- is going to become a standard package in Sage (optional is fine, of course). The only way would be if somebody were to officially take over the project and start making releases, etc.`

Native Lie group code in SAGE

- There is already code for many very typical Lie group calculations.
- SAGE knows about all root systems and Cartan types.
- SAGE can do computations in the Weyl group, conjugation of weights, etc.
- SAGE can do tensor products of irreducible representations.
- SAGE can currently do branching rules to maximal subgroups except reducible ones. Thus $U(4) \to U(3)$ but not $U(4) \to U(2) \times U(2)$.
- There is also substantial support for related symmetric function theory and combinatorics, particularly crystal bases.
- `combinat/root_system/weyl_group.py`
- `combinat/root_system/weyl_characters.py`

Groups, Lie algebras and enveloping algebras

The following have **equivalent categories of finite-dim'l rep'ns.**

group Lie algebra or enveloping algebra	**notation (this page)**	**example**	**note**
a simply-connected compact Lie group (automatically semsimple)	G	$\mathrm{SU}(n)$	
a noncompact real form of G	G'	$\mathrm{SL}(n,\mathbb{R})$	
complex form of G or G' (a complex analytic group)	$G_{\mathbb{C}}$	$\mathrm{SL}(n,\mathbb{C})$	analytic representations
semisimple real Lie algebra	$\mathfrak{g} = \mathrm{Lie}(G)$ $\mathfrak{g}' = \mathrm{Lie}(G')$	$\mathfrak{su}(n)$ $\mathfrak{sl}(n,\mathbb{R})$	$\pi\colon \mathfrak{g} \longrightarrow \mathrm{End}(V)$ is $\mathbb{R}$-linear (V a complex VS)
semisimple complex Lie algebra	$\mathfrak{g}_{\mathbb{C}} = \mathrm{Lie}(G_{\mathbb{C}})$ $\mathfrak{g}_{\mathbb{C}} \cong \mathbb{C} \otimes \mathfrak{g}$ $\cong \mathbb{C} \otimes \mathfrak{g}'$	$\mathfrak{sl}(n,\mathbb{C})$	$\pi\colon \mathfrak{g}_{\mathbb{C}} \longrightarrow \mathrm{End}(V)$ is $\mathbb{C}$-linear
enveloping algebra	$U(\mathfrak{g}_{\mathbb{C}})$		
quantum group	$U_q(\mathfrak{g}_{\mathbb{C}})$	$q \in \mathbb{C}^\times$	(q not a root of 1)

Maximal Tori and their characters

- Let G be a **complex analytic group**. (Assume reductive.) Let T be a maximal torus, $\Lambda = X^*(T)$ be its group of analytic characters. Then $T \cong (\mathbb{C}^\times)^r$ for some r and $\Lambda \cong \mathbb{Z}^r$.

Example 1: $G = \mathrm{GL}_n(\mathbb{C})$. Then T is the diagonal subgroup and $X^*(T) \cong \mathbb{Z}^n$. If $\lambda = (\lambda_1, \cdots, \lambda_n)$ then λ is identified with the rational character

$$\boldsymbol{t} = \begin{pmatrix} t_1 & & \\ & \ddots & \\ & & t_n \end{pmatrix} \longmapsto \prod t_i^{\lambda_i}.$$

Example 2: $G = \mathrm{SL}_n(\mathbb{C})$. Again T is the diagonal subgroup but now if

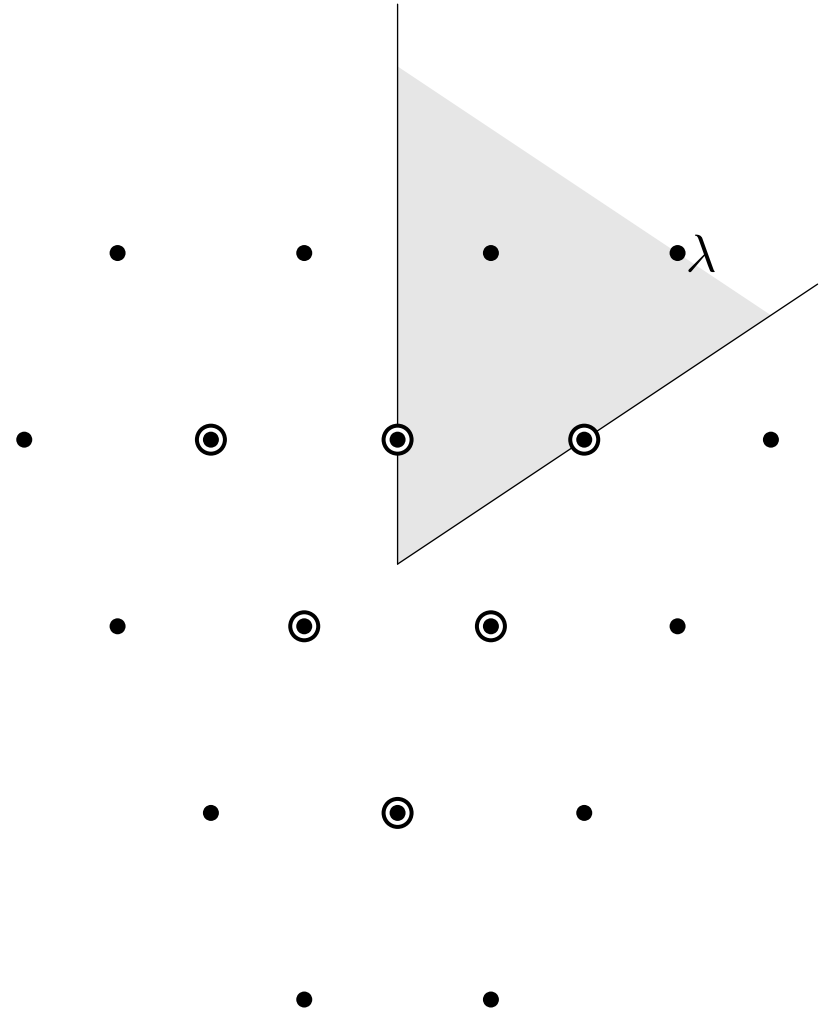
$$\lambda \in \mathbb{Z}^\Delta = \{(d, \cdots, d) | \, d \in \mathbb{Z}\} \subseteq \mathbb{Z}^n$$

then $\prod t_i^{\lambda_i} = \det(\boldsymbol{t})^d = 1$, so $X^*(T) \cong \mathbb{Z}^n / \mathbb{Z}^d \cong \mathbb{Z}^{n-1}$.

Weights

Let $\Lambda = X^*(T)$ be the group of rational characters. Then $\Lambda \cong \mathbb{Z}^r$.

- Elements of $\Lambda \cong \mathbb{Z}^r$ are called **weights**
- $\mathbb{R} \otimes X^*(T) \cong \mathbb{R}^r$ has a fundamental domain $\mathcal{C}^+$ for the Weyl group W called the **positive Weyl chamber**. Weights in $\mathcal{C}^+$ are **dominant**.
- It is useful to embed Λ in $\mathbb{R}^r$ and consider weights as lattice points.
- If $(\pi,\ V)$ is a representation then restricting to T, the module V decomposes into a direct sum of weight eigenspaces $V(\mu)$ with multiplicity $m(\mu)$ for weight μ.
- There is a unique highest weight λ wrt partial order. We have $\lambda \in \mathcal{C}^+$ and $m(\lambda) = 1$.
- $V \longleftrightarrow \lambda$ gives a **bijection** between irreducible representations and weights λ in $\mathcal{C}^+$.



Here are the weights of an irreducible V for $G = \mathrm{SL}_3$, showing multiplicities and the pos. Weyl chamber.

- $m(\mu) = 1$
- $m(\mu) = 2$
- $\mathcal{C}^+$

Legend

Roots

- G acts on itself by conjugation.
- Hence it acts on its Lie algebra $\mathfrak{g}$.
- The nonzero weights in $\mathfrak{g}$ are called **roots**.
- If α is a root the eigenspace $\mathfrak{g}(\alpha)$ is one dimensional. Let X_α be a generator.
- If (π, V) is any rep'n, μ a weight, $\pi(X_\alpha)$ maps $V(\mu)$ to $V(\mu+\alpha)$.
- The set Φ of roots is called the **root system**.
- Φ is partitioned into **positive roots** Φ^+ and negative roots Φ^-.
- If $\alpha \in \Phi^+$ cannot be expressed as a sum over other positive roots then α is called **simple**. Let $\alpha_1, \cdots, \alpha_r$ be the simple roots.
- There are **fundamental weights** $\varepsilon_1, \cdots, \varepsilon_r$ such that $\langle \varepsilon_i, \alpha_j \rangle = \delta_{ij}$.
- There reflection in the hyperplane perpendicular to a simple root α_i is called a **simple reflection**. They generate the **Weyl group.**

Cartan Types

Root systems are classified by their **Cartan types**. The irreducible types:

A_r	SL_{r+1} or GL_{r+1}
B_r	$\mathrm{SO}(2r+1)$ or $\mathrm{spin}(2r+1)$
C_r	$\mathrm{Sp}(2r)$
D_r	$\mathrm{SO}(2r)$ or $\mathrm{spin}(2r)$
G_2, F_4, E_6, E_7, E_8	exceptional groups
affine types	affine Kac-Moody Lie algebras

The affine Cartan types are implemented in SAGE and are used in the crystal code. They do not correspond to Lie groups.

```
sage: t = CartanType("E8"); t
['E', 8]
sage: t.root_system()
Root system of type ['E', 8]
sage: R = RootSystem("E8"); R
Root system of type ['E', 8]
sage: R.cartan_type()
['E', 8]
sage: t == R.cartan_type() and R == t.root_system()
True
```

Root systems in SAGE

- The ambient vector space (`L` in the next example) has many useful attributes and methods. The simple roots, fundamental weights, Weyl group are all attributes or methods of `L`.

```
sage: R = RootSystem("F4"); R
Root system of type ['F', 4]

sage: L = R.ambient_space(); L
Ambient space of the Root system of type ['F', 4]

sage: [L.simple_root(i) for i in [1,2,3,4]]
[(0, 1, -1, 0), (0, 0, 1, -1), (0, 0, 0, 1),
(1/2, -1/2, -1/2, -1/2)]

sage: W = L.weyl_group(); W
Weyl Group of type ['F', 4] (as a matrix group acting on the
ambient space)
```

GL_n versus SL_n

- For root systems of type A_r there is a choice of coding the ambient vector space to be $r+1$ dimensional or r dimensional.
- `SAGE` makes the ambient vector space $r+1$ dimensional.
- This means that A_r is the root system of GL_{r+1}, not SL_{r+1}.
- GL_{r+1} is reductive but not semisimple.
- Thus if $r=2$ the fundamental weights are $(1,0,0)$ and $(1,1,0)$.
- Not $\left(\frac{2}{3},-\frac{1}{3},-\frac{1}{3}\right)$ and $\left(\frac{1}{3},\frac{1}{3},-\frac{2}{3}\right)$.
- $\rho=(2,1,0)$ not $(1,0,-1)$. Works in all formulae (e.g. Weyl character formula) but it's not "half the sum of the positive roots."
- Advantage: many things are simpler.
- It's Bourbaki-safe.
- Important in some situations: e.g. GL_n is often a Levi subgroup.

GL_n versus SL_n, continued

For example if you want to work with SL_3, the character `ad` of the adjoint representation is realized as follows:

```
sage: A2=WeylCharacterRing("A2")
sage: ad=A2(1,0,-1); ad
A2(1,0,-1)
```

If you are working on GL_3, the character with highest weight $(2, 1, 0)$ is realized as follows:

```
sage: sym=A2(2,1,0); sym
A2(2,1,0)
```

We have `det=(1,1,1)` so `A2(2,1,0)=A2(1,0,-1)` $\otimes$ `det`. Thus on SL_3 these representations are the same. But on GL_3 you want to distinguish them.

The characters `A2(2,1,0)` and `A2(1,0,-1)` which are the same on SL_3 but not on GL_3 naturally behave similarly. For comparison we decompose their tensor squares into irreducibles.

```
sage: ad*ad
A2(0,0,0) + 2*A2(1,0,-1) + A2(1,1,-2) + A2(2,-1,-1) + A2(2,0,-2)
sage: sym*sym
A2(2,2,2) + 2*A2(3,2,1) + A2(3,3,0) + A2(4,1,1) + A2(4,2,0)
```

If $G = \mathrm{SL}_3$ these decompositions are really the same.

If you want to be an SL_3 purist, you may encode any character of SL_3 as `A2(x,y,z)` where `x+y+z` $= 0$. But then the standard representation is `A2(2/3,-1/3,-1/3)`.

Sage does not get confused by `A2(2/3,-1/3,-1/3)`.

```
sage: f1=A2(2/3,-1/3,-1/3)
sage: f1*f1
A2(1/3,1/3,-2/3) + A2(4/3,-2/3,-2/3)
```

But isn't it easier to work with $(1,0,0)$?

```
sage: f1=A2(1,0,0)
sage: f1*f1
A2(1,1,0) + A2(2,0,0)
```

Even though `SAGE` will accept `A2(2/3,-1/3,-1/3)`, internally it "knows" the first fundamental representation as `A2(1,0,0)`.

```
sage: A2.lattice().fundamental_weights()
Finite family {1: (1, 0, 0), 2: (1, 1, 0)}
```

The Weyl is a gap matrix group

The simple reflections are the generators of this Coxeter group.

```
sage: W = L.weyl_group(); W
Weyl Group of type ['F', 4] (as a matrix group acting on the
ambient space)
sage: W.simple_reflections()
[[1 0 0 0]
[0 0 1 0]
[0 1 0 0]
[0 0 0 1],
[1 0 0 0]
[0 1 0 0]
[0 0 0 1]
[0 0 1 0],
    ...
[ 1/2 1/2 1/2 1/2]
[ 1/2 1/2 -1/2 -1/2]
[ 1/2 -1/2 1/2 -1/2]
[ 1/2 -1/2 -1/2 1/2]]
```

The length of a Weyl group element

```
sage: [s1,s2,s3,s4] = [W.simple_reflection(i) for i in [1,2,3,4]]

sage: w = s1*s2*s3*s4; w
[ 1/2 1/2 1/2 1/2]
[-1/2 1/2 1/2 -1/2]
[ 1/2 1/2 -1/2 -1/2]
[ 1/2 -1/2 1/2 -1/2]
```

- The length $l(w)$ is the smallest number of simple reflections into which w may be factored.
- It is the number of $\alpha \in \Phi^+$ such that $w(\alpha) \in \Phi^-$.

```
sage: w.length()
4
sage: [a for a in L.positive_roots() if w.action(a) not in
       L.positive_roots()]
[(1, -1, 0, 0), (1, 0, -1, 0), (1/2, -1/2, -1/2, 1/2),
  (1/2, -1/2, -1/2, -1/2)]
```

Longest element, Cartan matrix

- The length of the longest element is the number of positive roots.

```
sage: w0 = W.long_element(); w0
[-1  0  0  0]
[ 0 -1  0  0]
[ 0  0 -1  0]
[ 0  0  0 -1]
sage: w0.length()
24
sage: len(L.positive_roots())
24
```

- The Cartan matrix is the matrix of inner products $\frac{2\langle \alpha_i, \alpha_j \rangle}{\langle \alpha_i, \alpha_i \rangle}$.

```
sage: R.cartan_matrix()
[ 2 -1  0  0]
[-1  2 -1  0]
[ 0 -2  2 -1]
[ 0  0 -1  2]
```

Dynkin Diagram

- There are two conventions for ordering of the simple roots: Bourbaki and Dynkin. `Sage` and `LiE` both follow Bourbaki.

```
sage: D = RootSystem("E7").dynkin_diagram(); D
Dynkin diagram of type ['E', 7]
```

- `D.plot()` is not very helpful. The best way to view the Dynkin diagram is to use the optional package LiE.
- We should change the `__repr__` method of `DynkinDiagram_class` to make such ascii art.

```
sage: lie.diagram("F4")
O---O=>=O---O 1 2 3 4 F4
sage: lie.diagram("E8")
        O 2
        |
        |
O---O---O---O---O---O---O
1   3   4   5   6   7   8
E8
```

Weyl Character rings

Given a dominant weight λ, there is a unique irreducible module $V(\lambda)$ with highest weight λ. Let χ_λ denote its character.

- The characters χ_λ span a ring.
- The addition corresponds to direct sum of modules and the multiplication corresponds to tensor product.
- The ring is implemented as an instance of class `WeylCharacterRing`.
- `weyl_character_ring.py` should be converted to Cython.

We create this ring for B_3, i.e. spin(7).

```
sage: B3 = WeylCharacterRing("B3", cache=true)
sage: L = B3.lattice(); L
Ambient space of the Root system of type ['B', 3]
```

Caching will speed things up. The method producing the ambient space is called `lattice` for historical reasons.

Characters

- Characters are elements of a Weyl Character ring.
- They are instances of class `WeylCharacter`

We name the fundamental weights `fw1`, `fw2` and `fw3`, and their characters `chi1`, `chi2` and `chi3`.

```
sage: [fw1,fw2,fw3] = [L.fundamental_weights()[i] for i in [1,2,3]]
sage: [chi1,chi2,chi3] = [B3(x) for x in [fw1,fw2,fw3]]
sage: fw1, chi1, chi1.degree()
((1, 0, 0), B3(1,0,0), 7)
sage: chi1.parent()
The Weyl Character Ring of Type [B,3] with Integer Ring
coefficients
```

`chi1` is the character of the standard (orthogonal) representation.

```
sage: fw3, chi3, chi3.degree()
((1/2, 1/2, 1/2), B3(1/2,1/2,1/2), 8)
```

The eight dimensional spin representation. Half integral rep's are rep's of spin(7) that do not factor through the homomorphism spin(7) $\longrightarrow$ SO(7).

Tensor products

We can compute tensor products as sums of irreducibles.

```
sage: chi1*chi3
B3(1/2,1/2,1/2) + B3(3/2,1/2,1/2)
sage: B3(1/2,1/2,1/2) * B3(3/2,1/2,1/2)
B3(1,0,0) + B3(1,1,0) + B3(1,1,1) + B3(2,0,0) + B3(2,1,0) +
B3(2,1,1)
sage: chi3^5
30*B3(1/2,1/2,1/2) + 40*B3(3/2,1/2,1/2) + 35*B3(3/2,3/2,1/2) +
20*B3(3/2,3/2,3/2) + 14*B3(5/2,1/2,1/2) + 16*B3(5/2,3/2,1/2) +
10*B3(5/2,3/2,3/2) + 5*B3(5/2,5/2,1/2) + 4*B3(5/2,5/2,3/2) +
B3(5/2,5/2,5/2)
```

Weights

We can find the weights of the irreducible representation and their multiplicities.

```
sage: B3(2,1,0).degree()
105
sage: B3(2,1,0).mlist()
[[(1, 0, 2), 1],
[(1, -1, 1), 2],
[(1, 0, -1), 2],
[(-2, 0, 1), 1],
        ...
```

- These are computed using the Freudenthal multiplicity formula.
- The function `irreducible_character_freudenthal` underlies all everything.
- Optimizing it would be a Good Thing.

Weight Rings

If you want to work directly with weights, you may associate with a WeylCharacter ring an auxiliary **weight ring**. Let us define this pair of rings for $D_4 = \mathrm{spin}(8)$.

```
sage: D4=WeylCharacterRing("D4")
sage: d4=WeightRing(D4)
```

- `D4` is the character ring of G and `d4` is the group algebra of $X^*(T)$.
- By default the coefficient ring is $\mathbb{Z}$ but you can choose another ring.
- If μ is a weight, `d4(`μ`)` is μ as an element of the weight ring, which is the character ring of $X^*(T)$.
- By contrast `D4(`λ`)` is only defined if λ is dominant. It is the character with highest weight λ.

Coercion G2 -> g2 (for example)

```
G2 = WeylCharacterRing("G2", base_ring=QQ)
g2 = WeightRing(G2)
L = G2.lattice()
```

We'll need rational coefficients at the end of this example (next page) so the base ring is $\mathbb{Q}$. Any element of the character ring (`G2` in this case) can be coerced into the weight ring. This is a good way to see the weight multiplicities. Here is the seven-dimensional representation of G_2:

```
sage: chi = G2(L.fundamental_weights()[1]); chi
G2(1,0,-1)
sage: g2(chi)
g2(-1,0,1) + g2(-1,1,0) + g2(0,-1,1) + g2(0,0,0) + g2(0,1,-1) +
g2(1,-1,0) + g2(1,0,-1)
```

Coercion g2 -> G2

Any character is W-invariant, so coercion in the other direction is only possible if the weight ring element is W-invariant. To illustrate, we make a W-invariant virtual character by brute force averaging.

```
sage: wt = g2(L.fundamental_weights()[1]); wt
g2(1,0,-1)
sage: vc = sum(wt.weyl_group_action(w) for w in L.weyl_group()); vc
2*g2(-1,0,1) + 2*g2(-1,1,0) + 2*g2(0,-1,1) + 2*g2(0,1,-1) +
2*g2(1,-1,0) + 2*g2(1,0,-1)
sage: vc.character()
-2*G2(0,0,0) + 2*G2(1,0,-1)
```

It is a multiple of 2 since the character we started with is on a wall of the positive Weyl chamber, hence has a nontrivial stabilizer.

```
sage: (1/2)*vc.character()
-G2(0,0,0) + G2(1,0,-1)
```

Another Weight Ring Example

If λ is a dominant weight, the Weyl denominator formula for spin(7) is

$$\sum_{w\in W}(-1)^{l(w)}\mathtt{b3}(\rho)=\prod_{\alpha\in\Phi^+}\left[\mathtt{b3}\left(\frac{\alpha}{2}\right)-\mathtt{b3}\left(-\frac{\alpha}{2}\right)\right],\qquad \rho=\frac{1}{2}\sum_{\alpha\in\Phi^+}\alpha.$$

```
sage: B3 = WeylCharacterRing("B3",cache=true)
sage: b3 = WeightRing(B3)
sage: rho = B3.lattice().rho(); rho
(5/2, 3/2, 1/2)
sage: W = B3.lattice().weyl_group(); W.order()
48
sage: Delta = sum((-1)^w.length()*b3(w.action(rho)) for w in W)
sage: Delta
-b3(-5/2,-3/2,-1/2) + b3(-5/2,-3/2,1/2) + b3(-5/2,-1/2,-3/2) -
...
- b3(5/2,3/2,-1/2) + b3(5/2,3/2,1/2)
sage: wd = prod(b3(x/2)-b3(-x/2) for x in B3.lattice().positive_roots())
sage: wd == Delta
true
```

Branching Rules

If G is a Lie group and H a subgroup, how do irreducible characters of G restrict to H?

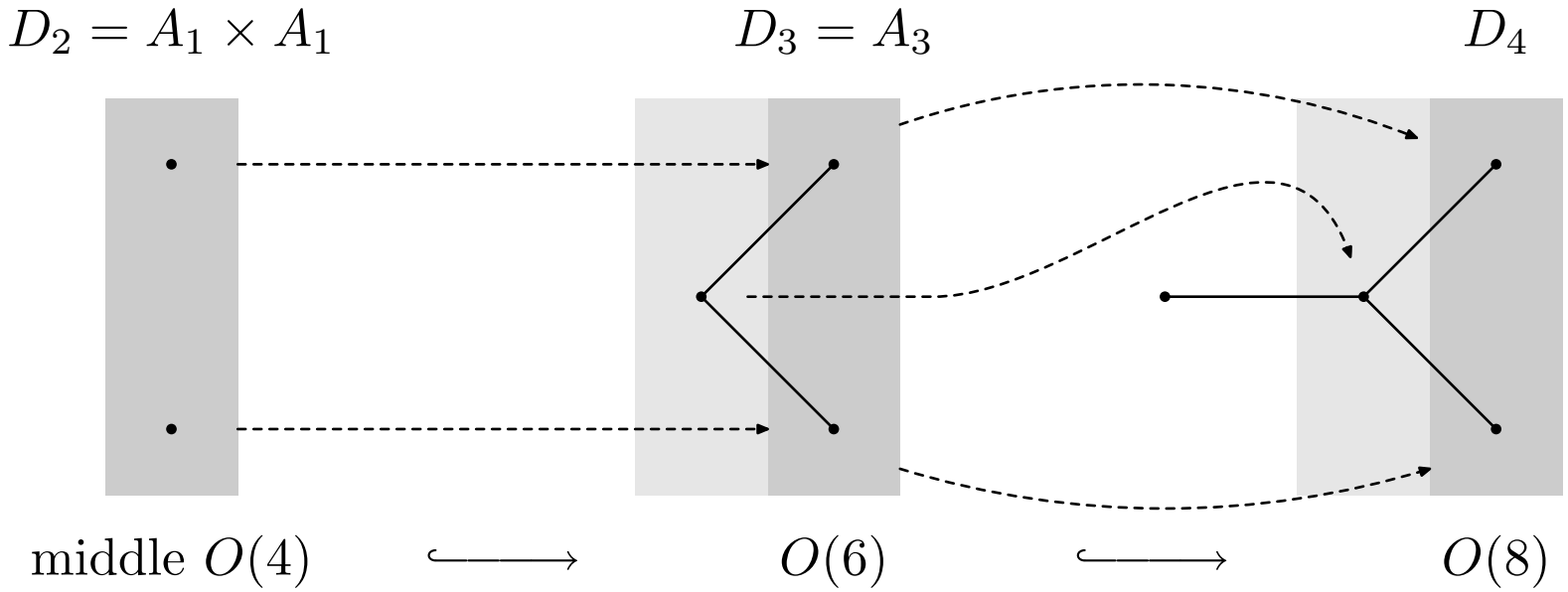
- Branching rules are a standard and important type of computation.
- One reduces immediately to H maximal.
- Maximal subgroups were classified by Dynkin.
- `SAGE` knows about branching rules with **two restrictions.**
- ***H* must be of irreducible type.**

Thus `SAGE` knows the $\mathrm{GL}(4) \to \mathrm{GL}(3)$ rule but not $\mathrm{GL}(4) \to \mathrm{GL}(2) \times \mathrm{GL}(2)$.

- This could be fixed. (A couple days work.) Sage already knows about reducible root systems but the rules have to be coded.
- ***G* and *H* are not exceptional groups.**
- These restrictions could be fixed with **a couple of day's work.**
- Maximal subgroups can be read off from the Dynkin diagram.
- Type `branch_weyl_character?` for help. Or read on ...

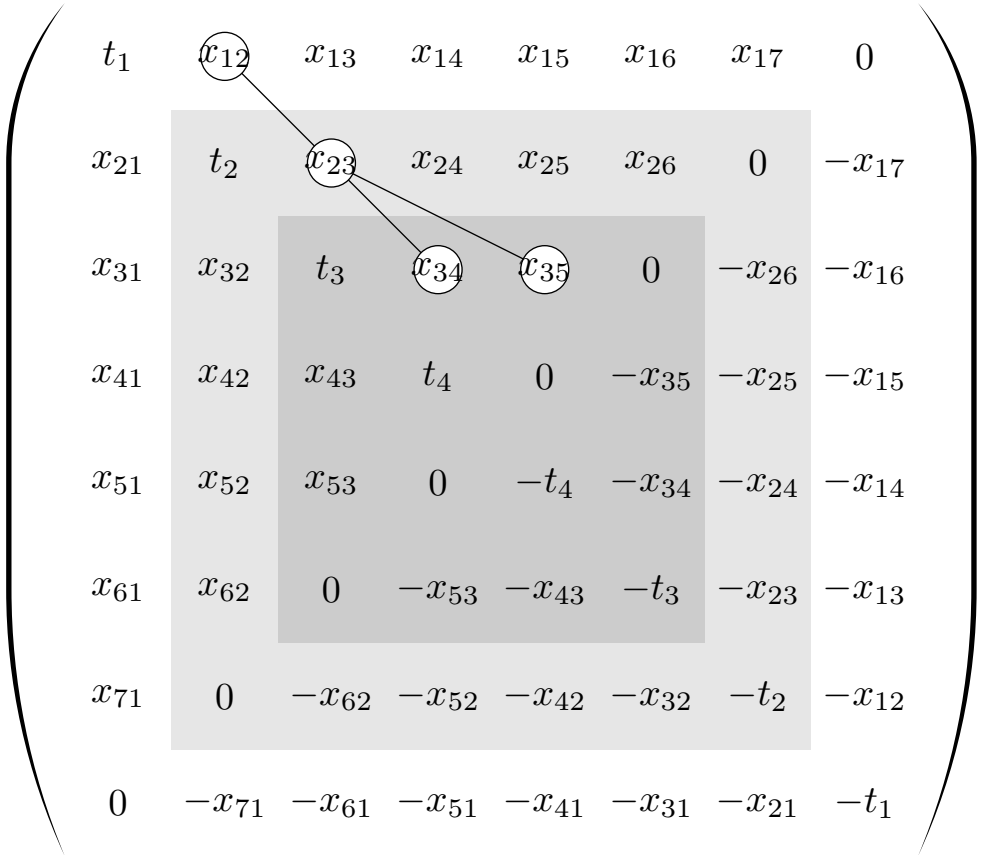
Levi Subgroups

Eliminating one node in the Dynkin diagram produces the Dynkin diagram of a Levi subgroup. Thus we envision the inclusions $O(4) \to O(6) \to O(8)$:



- The actual Levi subgroup of $O(2r)$ is $\mathrm{GL}_1 \times O(2r-2)$.
- These Dynkin diagrams illustrate accidental isomorphisms $D_3 = A_3$ (that is, $\mathrm{spin}(6) = \mathrm{SL}_3$) and $D_2 = A_1 \times A_1$ ($\mathrm{spin}(4) = \mathrm{SL}_2 \times \mathrm{SL}_2$).

We can visualize the Dynkin diagram of D_4 superimposed on the four simple roots in the Lie algebra. Eliminating x_{12} passes to the light gray Lie algebra which is $D_3 = \mathrm{SO}(6)$. Then eliminating x_{23} passes to the dark gray Lie algebra which is $D_2 = \mathrm{SO}(4)$.



Branching rules of Levi type

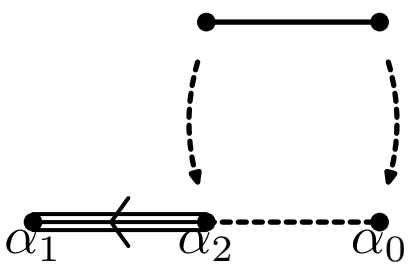
Sage knows branching to Levi subgroups that are irreducible when G, H are not reducible and not exceptional.

```
sage: D4 = WeylCharacterRing(['D',4])
sage: A3 = WeylCharacterRing(['A',3])
sage: D3 = WeylCharacterRing(['D',3])
sage: fun_wts = [wt for wt in D4.lattice().fundamental_weights()]
sage: [D4(w).branch(A3,rule="levi") for w in fun_wts]
[A3(0,0,0,-1) + A3(1,0,0,0), A3(0,0,-1,-1) + A3(0,0,0,0) +
A3(1,0,0,-1) + A3(1,1,0,0), A3(1/2,-1/2,-1/2,-1/2) +
A3(1/2,1/2,1/2,-1/2), A3(-1/2,-1/2,-1/2,-1/2) +
A3(1/2,1/2,-1/2,-1/2) + A3(1/2,1/2,1/2,1/2)]
sage: [D4(w).branch(D3,rule="levi") for w in fun_wts]
[2*D3(0,0,0) + D3(1,0,0), D3(0,0,0) + 2*D3(1,0,0) + D3(1,1,0),
D3(1/2,1/2,-1/2) + D3(1/2,1/2,1/2), D3(1/2,1/2,-1/2) +
D3(1/2,1/2,1/2)]
```

Extended Dynkin Diagrams

The Dynkin diagram can be extended by adjoining a node, corresponding to a negative root α_0, where $-\alpha_0$ is the longest root.

- The Weyl group of the extended diagram is infinite. These are the affine Weyl groups which arise in Kac-Moody theory, Iwahori Hecke algebras, string theory, moonshine and other madness.
- A more mundane application is the enumeration of some maximal subgroups that are not of Levi type.
- If eliminating one node of the extended diagram results in a Dynkin diagram then there is a corresponding maximal subgroup.

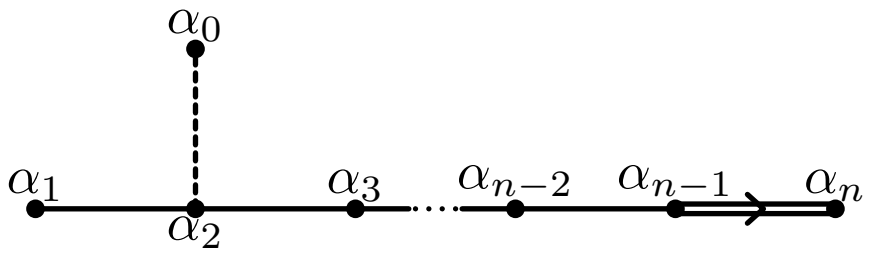


A_2 (ordinary Dynkin diagram)

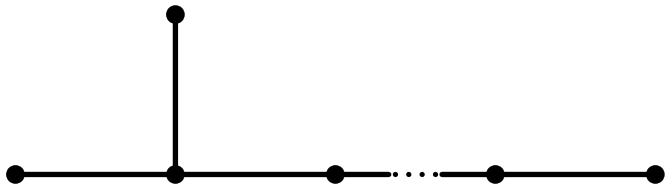
G_2 (extended Dynkin diagram)

Example

Eliminating the node α_n from the B_n **extended** Dynkin diagram:



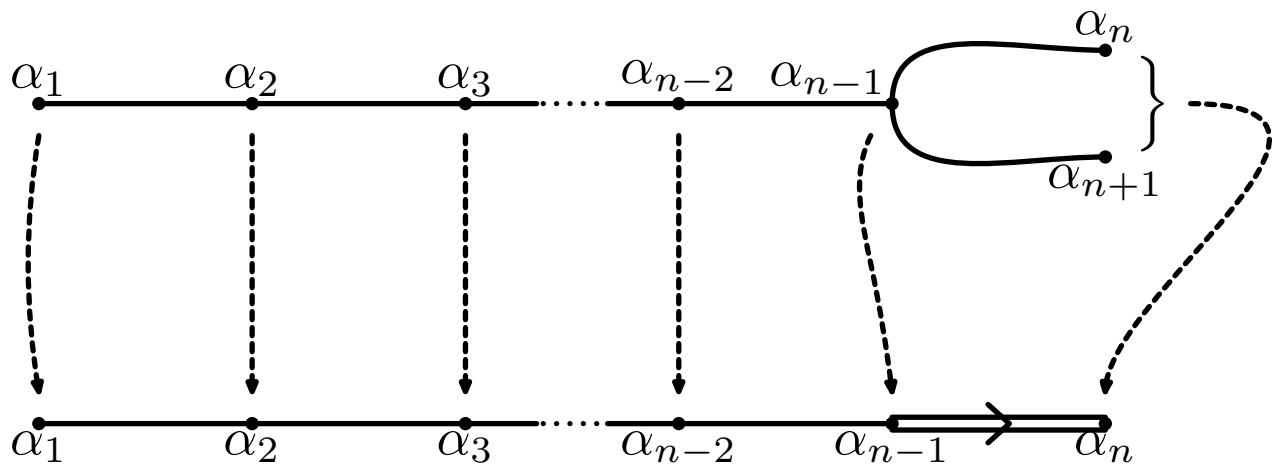
results in the D_n Dynkin diagram, corresponding to the subgroup $\mathrm{spin}(2n)$ of $\mathrm{spin}(2n+1)$:



```
sage: B4 = WeylCharacterRing("B4")
sage: D4 = WeylCharacterRing("D4")
sage: fw = B4.lattice().fundamental_weights()
sage: [B4(x).branch(D4,rule="extended") for x in fw]
[D4(0,0,0,0) + D4(1,0,0,0), D4(1,0,0,0) + D4(1,1,0,0),
D4(1,1,0,0) + D4(1,1,1,0),
D4(1/2,1/2,1/2,-1/2) + D4(1/2,1/2,1/2,1/2)]
```

Root Folding

- If the Dynkin diagram of G has a symmetry there is a maximal subgroup H.
- The Dynkin diagram of H is obtained from that of G by "folding" some roots together.



Hence the embedding of $B_n = \mathrm{spin}(2n+1)$ into $D_n = \mathrm{spin}(2n+2)$.

- In `SAGE` these embeddings are described as **symmetric type**.

Example

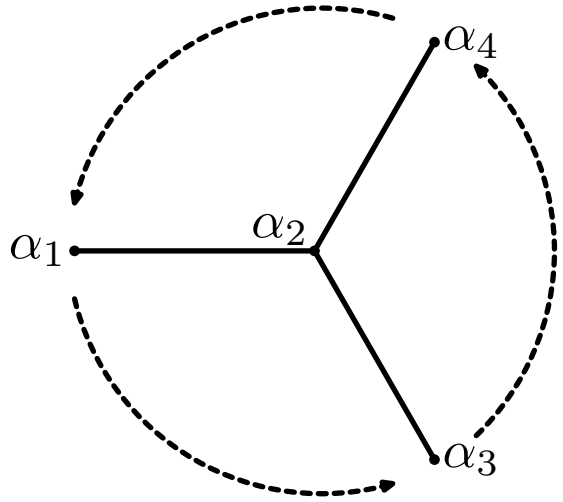
```
sage: D5 = WeylCharacterRing("D5")
sage: B4 = WeylCharacterRing("B4")
sage: fw = D5.lattice().fundamental_weights()
sage: [D5(x).branch(B4,rule="symmetric") for x in fw]
[B4(0,0,0,0) + B4(1,0,0,0), B4(1,0,0,0) + B4(1,1,0,0),
B4(1,1,0,0) + B4(1,1,1,0), B4(1/2,1/2,1/2,1/2),
B4(1/2,1/2,1/2,1/2)]
```

- Recall that the odd orthogonal spin groups ($B_r = \mathrm{spin}(2r+1)$) have one spin representation of degree 2^r.
- The even orthogonal spin groups ($D_r = \mathrm{spin}(2r)$) have two spin representations each of degree 2^{r+1}.
- We can see that the two $\pm$ spin representations of spin(10) both have the same restriction to the spin representation of spin(9). If we further restricted it to spin(8) it would split into the two $\pm$ spin representations of SO(8).

Automorphisms

An automorphism is a special case of a branching rule. If the Dynkin diagram has a symmetry, then the Lie group has an outer automorphism. This happens in types A_r, D_r and E_r. For types A and D `SAGE` knows these automorphisms.

The most spectacular case is type $D_4 = \text{spin}(0)$, which has an extra automorphism of order 8 called triality.



This has a beautiful description in terms of the octonions, which we will not describe. But `SAGE` can compute the effect on characters.

Triality

```
sage: D4 = WeylCharacterRing("D4")
sage: for lamb in D4.lattice().fundamental_weights():
sage:     print D4(lamb), "->", D4(lamb).branch(D4,rule="triality")
D4(1,0,0,0) -> D4(1/2,1/2,1/2,-1/2)
D4(1,1,0,0) -> D4(1,1,0,0)
D4(1/2,1/2,1/2,-1/2) -> D4(1/2,1/2,1/2,1/2)
D4(1/2,1/2,1/2,1/2) -> D4(1,0,0,0)
```

Note that the standard representation and the two spin representations are all eight dimensional. They are permuted by triality.

What remains to be done

Areas for optimization:

- The program is too slow if the rank is at all large.
- In particular, try to speed up `irreducible_character_freudenthal`
- The Weyl group action relies on `gap`. This too can be slow.

Further implementation needed:

Of course many things could be done. But two seem most important to me:

- Branching to all maximal subgroups. Arbitrarily we stopped at non-exceptional groups and irreducible subgroups.
- Ascii art for Dynkin diagrams.

When these tasks are done, I think the Lie group engine in `SAGE` will be quite nice for real-world computations.

Crystals

- Crystals are combinatorial analogs of Lie group representations.
- Precursors to crystals can be seen in combinatorics in tableaux algorithms (Robinson, Schensted, Knuth, Schützenberger, ...).
- There are many parallels between such combinatorial theory and Lie group representation theory. For example if λ is a dominant weight for A_r and then $\dim(V_\lambda)$ is the number of tableaux of shape λ in the alphabet $1, 2, \cdots, r+1$.
- Kashiwara found an explanation of this in quantum groups and defined crystals.
- If λ is a highest weight, there is a crystal $\mathcal{B}(\lambda)$. It is a connected directed colored graph with a map wt: $\mathcal{B}(\lambda) \longrightarrow \Lambda$ (the weight lattice). The number of elements of weight μ is the weight multiplicity $m(\mu)$.

Crystals in SAGE

- SAGE has good support for crystals. You can construct all highest weight crystals for types A_r, B_r, C_r, D_r and G_2.
- The crystal code has already been used extensively by different research groups.