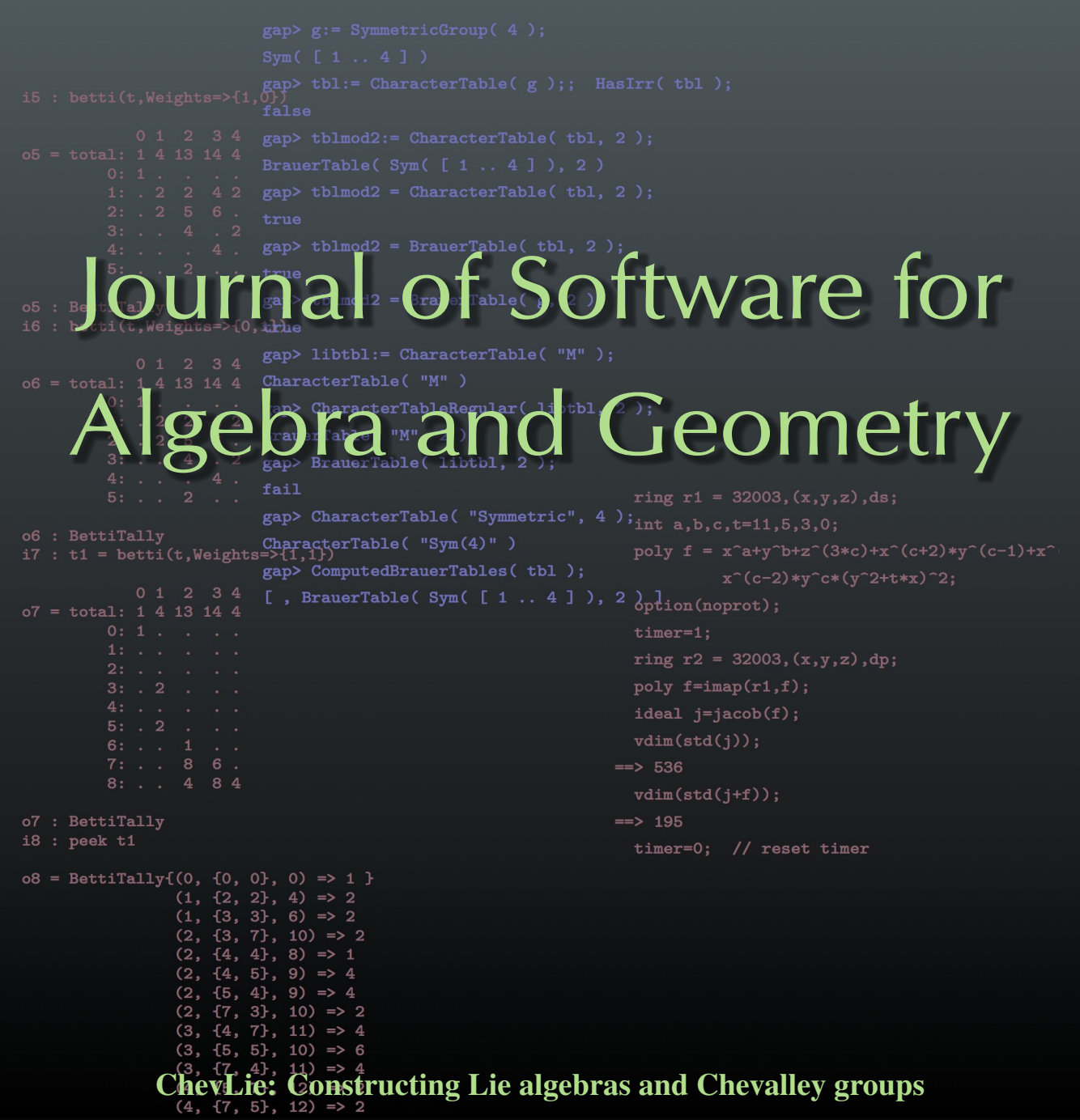

Journal of Software for Algebra and Geometry

ChevLie: Constructing Lie algebras and Chevalley groups

MEINOLF GECK

vol 10 2020

JSAG **10** (2020), 41–49
https://doi.org/10.2140/jsag.2020.10.41

The Journal of Software for
Algebra and Geometry

ChevLie: Constructing Lie algebras and Chevalley groups

MEINOLF GECK

ABSTRACT: We present ChevLie-1.1, a module for Julia and, ultimately, the emerging OSCAR system. It provides functions for constructing simple Lie algebras and the corresponding Chevalley groups (of adjoint or other types), using a recently established approach via Lusztig's "canonical bases". These programs, combined with the Julia interface to SINGULAR, supply an efficient, user-friendly way to establish a key part of a new characterisation of Lusztig's "special" nilpotent orbits in simple Lie algebras.

1. THE ϵ-CANONICAL CHEVALLEY BASIS OF A LIE ALGEBRA. Let $\mathfrak{g}$ be a finite-dimensional, simple Lie algebra over $\mathbb{C}$. By the classical Cartan–Killing theory (see [Humphreys 1978]), one can associate with $\mathfrak{g}$ a Dynkin diagram Γ (that is, one of the graphs in Figure 1 below) and $\mathfrak{g}$ is, up to isomorphism, uniquely determined by Γ. Conversely, an elegant way to construct a Lie algebra $\mathfrak{g}$ corresponding to such a diagram Γ is given by taking the quotient of the free Lie algebra on generators $\{e_i, f_i \mid i \in I\}$ (where I is an index set for the nodes of Γ) by the ideal generated by the Serre relations in [Humphreys 1978, (18.1)] (which only depend on Γ). The general theory then shows that $\mathfrak{g}$ has a basis

$$B = \{h_i \mid i \in I\} \cup \{e_\alpha \mid \alpha \in \Phi\},$$

where the elements $h_i := [e_i, f_i]$ $(i \in I)$ span a Cartan subalgebra $\mathfrak{h} \subseteq \mathfrak{g}$, the set Φ is the root system determined by Γ, and the e_α are chosen such that $[h_i, e_\alpha] \in \mathbb{C}e_\alpha$ for all $i \in I$; the e_α are unique up to nonzero scalar multiples.

Now, for practical purposes, it is convenient to fix a choice of the elements e_α. A general scheme for making such a choice is described in [Cohen et al. 2004, §3], for example; however, there is a certain amount of arbitrariness to it. Various ad hoc choices can also be found in the literature; see, for example, [Mizuno 1980, Table 12]. Here, we wish to advertise the fact that a natural choice for the e_α is provided by Lusztig's work on "canonical bases".

For this purpose, we fix a function $\epsilon : I \to \{\pm 1\}$ such that $\epsilon(i) = -\epsilon(j)$ whenever $i \neq j$ in I are joined by an edge in the Dynkin diagram Γ. Note that such functions exist since there are no closed paths in the diagram Γ. Since Γ is a connected graph, there are exactly two such functions; if ϵ is one of them,

MSC2010: primary 20G40; secondary 17B45.
Keywords: Lie algebras, canonical bases, weighted Dynkin diagrams.
`ChevLie` version 1.1

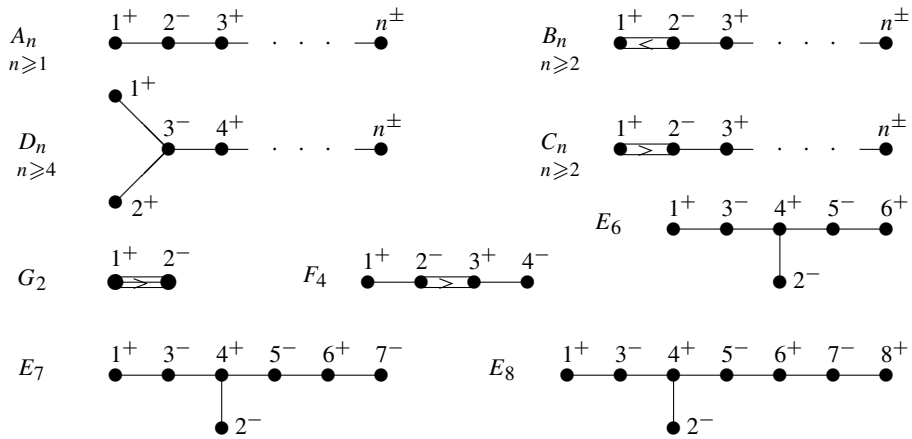

Figure 1. Dynkin diagrams.

then the other one is $-\epsilon$. In Figure 1, we have specified such a function ϵ by attaching a sign to each label. Now, having fixed ϵ, we set

$$u_i := -\epsilon(i)h_i \in \mathfrak{h} \qquad \text{for } i \in I.$$

Then it is shown in [Lusztig 1990a; 1990b; 2017] (see also [Geck 2017b] for an alternative proof) that there is a natural choice for the e_α such that the matrices of the linear operators

$$\mathrm{ad}(e_i) : \mathfrak{g} \to \mathfrak{g} \quad \text{and} \quad \mathrm{ad}(f_i) : \mathfrak{g} \to \mathfrak{g} \qquad (i \in I)$$

with respect to the basis $B^\epsilon = \{u_i \mid i \in I\} \cup \{e_\alpha \mid \alpha \in \Phi\}$ of $\mathfrak{g}$ have all their entries in $\mathbb{Z}_{\geqslant 0}$. In order to indicate the dependence on ϵ, we shall write $u_i = u_i^\epsilon$ for $i \in I$ and $e_\alpha = e_\alpha^\epsilon$ for $\alpha \in \Phi$. If we replace ϵ by $-\epsilon$, then $u_i^{-\epsilon} = -u_i^\epsilon$ for $i \in I$ and $e_\alpha^{-\epsilon} = -e_\alpha^\epsilon$ for $\alpha \in \Phi$. Thus, we have $B^{-\epsilon} = -B^\epsilon$.

There is a simple recursive algorithm for constructing the elements e_α^ϵ. First, we need some notation. Given $\alpha, \beta \in \Phi$, $\beta \neq \pm\alpha$, we define integers $p = p_{\alpha,\beta} \geqslant 0$ and $q = q_{\alpha,\beta} \geqslant 0$ by the condition that

$$\beta - q\alpha, \ldots, \beta - \alpha, \beta, \beta + \alpha, \ldots, \beta + p\alpha$$

are all contained in Φ, but $\beta - (q+1)\alpha \notin \Phi$ and $\beta + (p+1)\alpha \notin \Phi$. Furthermore, for each $i \in I$, there is a unique $\alpha_i \in \Phi$ such that e_{α_i} is a scalar multiple of e_i. Then $\Pi = \{\alpha_i \mid i \in I\}$ is a system of simple roots for Φ. Every $\alpha \in \Phi$ can be written uniquely as $\alpha = \sum_{i\in I} n_i\alpha_i$ where either all $n_i \in \mathbb{Z}_{\geqslant 0}$ or all $n_i \in \mathbb{Z}_{\leqslant 0}$. We set $\mathrm{ht}(\alpha) := \sum_{i\in I} n_i$, the "height" of α. Note also that $\Phi = -\Phi$.

We now proceed as follows. If $\mathrm{ht}(\alpha) = 1$, then $\alpha = \alpha_i$ where $i \in I$. In this case, we set $e_{\alpha_i}^\epsilon := \epsilon(i)e_i$ and $e_{-\alpha_i}^\epsilon := -\epsilon(i)f_i$. Now assume that $\mathrm{ht}(\alpha) > 1$ and that $e_{\pm\beta}^\epsilon$ has been already defined for all $\beta \in \Phi$ with $0 < \mathrm{ht}(\beta) < \mathrm{ht}(\alpha)$. Then there is some $i \in I$ such that $\beta := \alpha - \alpha_i \in \Phi$. In this case, $e_{\pm\alpha}^\epsilon$ are defined by

$$[e_{\alpha_i}^\epsilon, e_\beta^\epsilon] = \epsilon(i)(q_{\alpha_i,\beta} + 1)e_\alpha^\epsilon \quad \text{and} \quad [e_{-\alpha_i}^\epsilon, e_{-\beta}^\epsilon] = -\epsilon(i)(q_{\alpha_i,\beta} + 1)e_{-\alpha}^\epsilon.$$

(Note that, if $\mathrm{ht}(\alpha) > 1$, then there may be several $i \in I$ such that $\alpha - \alpha_i \in \Phi$; but the whole point of the

construction is that the resulting set $\{e_\alpha^\epsilon \mid \alpha \in \Phi\}$ does not depend on the choice of those i.) By [Geck 2017b, §5], we have

$$[e_\alpha^\epsilon, e_\beta^\epsilon] = \pm(q_{\alpha,\beta}+1)e_{\alpha+\beta}^\epsilon \qquad \text{whenever } \alpha, \beta, \alpha+\beta \in \Phi.$$

Thus, $B^\epsilon = \{u_i^\epsilon \mid i \in I\} \cup \{e_\alpha^\epsilon \mid \alpha \in \Phi\}$ is a particular Chevalley basis for $\mathfrak{g}$ in the sense of [Humphreys 1978, §25]; it may be called the "ϵ-canonical Chevalley basis" of $\mathfrak{g}$.

2. THE Julia MODULE ChevLie-1.1. The ChevLie package was originally developed in [GAP] (see, e.g., [Geck 2020, §4] for a short description) and then rewritten and extended for Julia [Julia 2017]. On a Linux system, one can just load the ChevLie package into a Julia session:

```
julia> include("chevlie1r1.jl"); using .ChevLie
```

The central command in this module is the Julia constructor `LieAlg`, with various fields containing basic information about a Lie algebra of a given type (a Julia symbol like `:g`) and rank (a positive integer); just type `?LieAlg` for further details and examples. In particular, there are fields holding the matrices (with entries in $\mathbb{Z}_{\geqslant 0}$, as discussed in Section 1) for the operators $\mathrm{ad}(e_i)$ and $\mathrm{ad}(f_i)$.

```
julia> lie=LieAlg(:g,2)
#I dim = 14
LieAlg('G2')
julia> lie.cartan     # the Cartan matrix given by the diagram
2x2 Array{Int8,2}:
  2  -1
 -3   2
julia> println(lie.epsilon)   # values of the epsilon function
Int8[1, -1]
julia> println(lie.roots)     # the roots of the Lie algebra
Array{Int8,1}[[1, 0], [0, 1], [1, 1], [1, 2], [1, 3], [2, 3],
    [-1, 0], [0, -1], [-1, -1], [-1, -2], [-1, -3], [-2, -3]]
julia> size(lie.e_i[1])       # matrix representing ad(e_1)
(14, 14)
```

(The conventions for labelling the Dynkin diagrams are those in Figure 1, which are the same as in CHEVIE [Geck et al. 1996].) The complete list of all matrices for the linear operators $\mathrm{ad}(e_\alpha^\epsilon) : \mathfrak{g} \to \mathfrak{g}$ is obtained by the function `canchevbasis`. Even when $\dim \mathfrak{g}$ becomes large, this should cause no problems with computer memory, because (1) the matrices are extremely sparse (and stored as `SparseArray`) and (2) the entries are small integers (and stored as `Int8`). Once these matrices are available, one can easily compute the corresponding structure constants $N_{\alpha,\beta}$ such that $[e_\alpha^\epsilon, e_\beta^\epsilon] = N_{\alpha,\beta}e_{\alpha+\beta}^\epsilon$ for $\alpha, \beta, \alpha+\beta \in \Phi$. (One just needs to work out one nonzero entry in the matrix of the Lie bracket.) This is done by the function `structconst`. There is some very basic functionality for working with the corresponding Weyl group; see, e.g., `allwords`, `reflsubgrp`. (Much more functionality is available in [Gapjm.jl].)

We mention some further useful functions.

`rep_minuscule`: returns a tuple of matrices representing the generators e_i, f_i, h_i in a highest weight representation with a given minuscule weight; the available minuscule weights are contained in the field `minuscule` of a Lie algebra. The representing matrices are formed using the "canonical" models discussed in [Geck 2017a].

`chevrootelt`: returns a generator, usually denoted $x_\alpha(t)$ (where $\alpha \in \Phi$ and t is an element in a field), for the Chevalley group associated with a Lie algebra and a given representation of it (adjoint or minuscule). See [Geck 2017b, §5] and [Geck 2017a, §5] for further details. This function (or rather, its precursor in the GAP version of our package) has been extensively applied in [Geck 2020]. There is also a function `collect_chevrootelts` that applies Chevalley's commutator relations in order to collect an arbitrary product of $x_\alpha(t)$'s (for positive α's) into a normal form.

`cross_regular`: returns a set of representatives (given by "Steinberg's cross section", see [Humphreys 1995, §4.15]) of the conjugacy classes of regular elements in a finite Chevalley group of simply connected type. Via the Jordan decomposition, this also yields a set of representatives of the semisimple conjugacy classes.

The online help (e.g., `?structconst`, `?rep_minuscule`, `?chevrootelt` and so on) contains further details, examples, and also references to related functions.

3. AN APPLICATION: SPECIAL NILPOTENT ORBITS. Let $\mathfrak{g}$ be a simple Lie algebra, and let $B^\epsilon = \{u_i^\epsilon \mid i \in I\} \cup \{e_\alpha^\epsilon \mid \alpha \in \Phi\}$ be the basis constructed in Section 1. The classical Dynkin–Kostant theory (see [Humphreys 1995, §7.6]) shows that the nilpotent orbits in $\mathfrak{g}$ are classified by weighted Dynkin diagrams; these are certain maps $d : I \to \{0, 1, 2\}$. In ChevLie, the complete list of all such weighted diagrams for $\mathfrak{g}$ is returned by the function `weighted_dynkin_diagrams`, which simply implements the explicitly known descriptions of those diagrams in all cases (see [Carter 1985, §13.1]).

```
julia> lie=LieAlg(:f,4); wdd=weighted_dynkin_diagrams(lie)
#I dim = 52
(Array{Int64,1}[[0, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 1], [0, 1, 0, 0],
  [2, 0, 0, 0], [0, 0, 0, 2], [0, 0, 1, 0], [2, 0, 0, 1], [0, 1, 0, 1],
  [1, 0, 1, 0], [0, 2, 0, 0], [2, 2, 0, 0], [1, 0, 1, 2], [0, 2, 0, 2],
  [2, 2, 0, 2], [2, 2, 2, 2]],
 [24, 16, 13, 10, 9, 9, 7, 6, 6, 5, 4, 3, 3, 2, 1, 0])
```

(The result is a tuple: the first component is the list of all d; the second component is the list of the numbers $\dim \mathcal{B}_e$ where $\mathcal{B}_e$ is the variety of Borel subalgebras containing a nilpotent element e in the orbit parametrised by d.)

Now let us fix such a weighted diagram $d : I \to \{0, 1, 2\}$. We extend d to a function on Φ by setting $d(\alpha) := \sum_{i \in I} n_i d(i)$, where $\alpha \in \Phi$ and $\alpha = \sum_{i \in I} n_i \alpha_i$ with $n_i \in \mathbb{Z}$ for all $i \in I$ (see Section 1). We set

$$\Phi_j := \{\alpha \in \Phi \mid d(\alpha) = j\} \quad \text{and} \quad \mathfrak{g}_d(j) := \langle e_\alpha \mid \alpha \in \Phi_j \rangle_{\mathbb{C}} \qquad (j = 1, 2).$$

Then an element in the nilpotent orbit parametrised by d is contained in $\mathfrak{g}_d(2)$. Furthermore, we have $[\mathfrak{g}_d(1), \mathfrak{g}_d(1)] \subseteq \mathfrak{g}_d(2)$. Hence, given any linear map $\lambda : g_d(2) \to \mathbb{C}$, we obtain an alternating bilinear form

$$\sigma_\lambda : \mathfrak{g}_d(1) \times \mathfrak{g}_d(1) \to \mathbb{C}, \qquad (x, y) \mapsto \lambda([x, y]).$$

For sufficiently general λ, this bilinear form is known to be nondegenerate (see, e.g., [Geck 2018, Remark 3.5]). We can reformulate this property as follows.

The condition is empty if $\mathfrak{g}_d(1) = \{0\}$. Assume now that $\mathfrak{g}_d(1) \neq \{0\}$. Let $\Phi_1 = \{\beta_1, \ldots, \beta_n\}$ and $\Phi_2 = \{\gamma_1, \ldots, \gamma_m\}$. Given $\lambda : \mathfrak{g}_d(2) \to \mathbb{C}$, we denote by $\mathcal{G}_\lambda \in M_n(\mathbb{C})$ the Gram matrix of σ_λ with respect to the basis $\{e^\epsilon_{\beta_1}, \ldots, e^\epsilon_{\beta_n}\}$ of $\mathfrak{g}_d(1)$. We set $x_l := \lambda(e^\epsilon_{\gamma_l})$ for $1 \leqslant l \leqslant m$. If $1 \leqslant i, j \leqslant n$ and $\beta_i + \beta_j \in \Phi$, then let $l(i, j) \in \{1, \ldots, m\}$ be such that $\gamma_{l(i,j)} = \beta_i + \beta_j$. Thus, we have

$$(\mathcal{G}_\lambda)_{ij} = \sigma_\lambda(e^\epsilon_{\beta_i}, e^\epsilon_{\beta_j}) = \begin{cases} N_{\beta_i,\beta_j} x_{l(i,j)} & \text{if } \beta_i + \beta_j \in \Phi, \\ 0 & \text{otherwise.} \end{cases}$$

Then σ_λ is nondegenerate if and only if $\det(\mathcal{G}_\lambda) \neq 0$. Now, in the above description, we may replace the values x_l by (commuting) indeterminates X_l over $\mathbb{Z}$. Then we obtain a well-defined matrix

$$\hat{\mathcal{G}}_d = \big(f_{ij}\big) \in M_n(\mathbb{Z}[X_1, \ldots, X_m])$$

such that, for any $\lambda : \mathfrak{g}_d(1) \to \mathbb{C}$, the matrix $\mathcal{G}_\lambda$ is obtained by specialising the indeterminate X_l to $x_l = \lambda(e_{\gamma_l}) \in \mathbb{C}$ for $1 \leqslant l \leqslant m$. We call $\hat{\mathcal{G}}_d$ the *generic Gram matrix* associated with d. (Note that $\hat{\mathcal{G}}_d$ depends on the choice of ϵ; but if we replace ϵ by $-\epsilon$, then $\hat{\mathcal{G}}_d$ is replaced by $-\hat{\mathcal{G}}_d$.) The fact that σ_λ is nondegenerate for sufficiently general λ now simply means that the polynomial

$$\hat{\delta}_d := \det(\hat{\mathcal{G}}_d) \in \mathbb{Z}[X_1, \ldots, X_m]$$

is nonzero. We say that d is *δ-special* if $\hat{\mathcal{G}}_d$ is "integrally nondegenerate", that is, there exist integers $z_1, \ldots, z_m \in \mathbb{Z}$ such that $\hat{\delta}_d(z_1, \ldots, z_m) = \pm 1$.

Conjecture 3.1 [Geck 2018, 4.10]. *Let $\mathcal{O}$ be the nilpotent orbit corresponding to d. Then d is δ-special if and only if $\mathcal{O}$ is "special" in the sense of Lusztig* [*1997*].

According to Lusztig [1997], special nilpotent orbits play a key role in several problems in representation theory, but they are "often regarded as rather mysterious objects". So the interest of the above conjecture lies in the fact that it would provide an intrinsic characterisation in terms of an integrality condition. The above conjecture is now known to hold (see [Geck 2018] and [Dong and Yang 2019]). The proof for Lie algebras of exceptional type relies on explicit computations, but [Geck 2018] contains no details about how this can be done concretely on a computer. As these computations are now part of a proof of a general result, we believe it is useful to provide these details, where we use ChevLie and the Julia interface to SINGULAR [Greuel and Pfister 2002], available via [OSCAR].

First, the ChevLie function `generic_gram_wdd` returns $\hat{\mathcal{G}}_d$ for a given weighted diagram d. (This uses the functions discussed in Section 2.)

```
julia> d=[1,0,1,2]; gr=generic_gram_wdd(lie,d)
(Any[0 0 0 :(1x2);
     0 0 :(1x2) :(2x3);
     0 :(-1x2) 0 0;
     :(-1x2) :(-2x3) 0 0], [1, 3, 5, 6], [4, 8, 9])
```

(The result is a triple: the second and third component contains the lists of roots with $d(\alpha) = 1$ and $d(\alpha) = 2$, respectively. The first component is the 4×4-matrix $\hat{\mathcal{G}}_d$; its entries are formed using Julia symbols `x1`, `x2`, `x3`.) The function `eval_gram_wdd` evaluates the symbols `x1`, `x2`, `x3` to actual elements in a field or a ring.

```
julia> println(eval_gram_wdd(gr,[1,1,1]))      # all X_i->1
[0 0 0 1; 0 0 1 2; 0 -1 0 0; -1 -2 0 0]
```

Now we are ready to verify Conjecture 3.1 for a given weighted diagram d. Suppose first that d is special in the sense of Lusztig. (Explicit lists of such d can be found in [Carter 1985, §13.4].) The strategy for verifying that d is δ-special is explained in [Geck 2018, Corollary 4.11]: we simply run through all possible vectors of values $(z_1, \ldots, z_m) \in \{0, 1\}^m$ (starting with the vector $1, 1, \ldots, 1$ and then increasing step by step the number of zeroes) until we find one such that $\det(\hat{\mathcal{G}}_d(z_1, \ldots, z_m)) = \pm 1$. Of course, the number of vectors $(z_1, \ldots, z_m)$ to test can quickly become enormous. In a sense, we are lucky because it turns out that, in all cases that we need to consider, this search is successful just after a few steps. In ChevLie, this procedure is implemented in the function `gram_wdd_search`:

```
julia> println(gram_wdd_search(lie,wdd[1][3],1))
#I [0, 0, 0, 1]            # the diagram under consideration
#I prime=1, dim g(1)=8, dim g(2)=7, k = 0 1
# ---> search successful for characterstic exponent 1
Rational{Int64}[0//1, 1//1, 1//1, 1//1, 1//1, 1//1, 1//1]
```

(Here, `k = 0 1 ...` indicates the number of zeroes in the vectors $(z_1, \ldots, z_m)$.) Thus, the above d (which is special by [Carter 1985, §13,4]) is also δ-special; an integer vector as required is given by $(z_1, z_2, z_3, z_4, z_5, z_6, z_7) = (0, 1, 1, 1, 1, 1, 1)$.

Conversely, assume that d is not special in the sense of Lusztig. Then the first idea would be to try to determine the polynomial $\hat{\delta}_d$ explicitly. Some simplification can be achieved since $\hat{\mathcal{G}}_d$ is skew-symmetric; consequently, $\hat{\delta}_\delta$ is the square of the pfaffian of $\hat{\mathcal{G}}_d$. For example:

```
julia> import Singular       # requires singular.jl, see [18]
julia> R,x=Singular.PolynomialRing(Singular.QQ,
                 ["x"*string(i) for i in 1:5]; ordering=:lex)
julia> gr=generic_gram_wdd(lie,wdd[1][9]);   # d=[0, 1, 0, 1]
julia> pfaffian(eval_gram_wdd(gr,x))
-3*x1*x4^2*x5 + 3*x2*x3*x4^2
```

Thus, we conclude that the above d (which is not special by [Carter 1985, §13,4]) is not δ-special either, since $\hat{\delta}_d(z_1, z_2, z_3, z_4, z_5)$ will be divisible by 3 for all $z_i \in \mathbb{Z}$.

Algorithm 1. A Julia/SINGULAR function for proving Proposition 5.10 of [Geck 2018].

```
function prop510(lie,d,pr)
  gr=generic_gram_wdd(lie,d)
  n,m=length(gr[2]),length(gr[3])
  str=prod([string(i) for i in d])
  print("#I ",str,", dim g(1), g(2) = ",n,", ",m," -> syz ")
  if n==0 || m==0    # trivial cases
    println("= 0"); return gr
  end
  xi=["x"*string(i) for i in 1:m]
  F=Singular.Fp(pr)              # field Z/pZ
  R,x=Singular.PolynomialRing(F,xi; ordering=:lex)
  gr1=eval_gram_wdd(gr,x)
  vecs=[Singular.vector(R,[gr1[i,j] for j in 1:n]...) for i in 1:n]
  sy=Singular.syz(Singular.Module(R,vecs...))
  if Singular.iszero(sy)==true
    println("= 0")
  else
    println("not 0 (",Singular.ngens(sy)," gens)")
  end
  return sy
end
```

The above method works in most cases but there are weighted diagrams in type E_8 where, e.g., $\hat{\mathcal{G}}_d \in M_{30}(\mathbb{Z}[X_1, \ldots, X_{30}])$ and the computation of the pfaffian becomes infeasible. But the above example suggests that the reason for d not being δ-special might be the fact that the polynomial $\hat{\delta}_d$ becomes 0 upon reduction modulo a suitable prime p. (It turns out in the end that this is correct.) A. Steel and U. Thiel pointed out that this question can be approached by computing syzygies (via Groebner bases), instead of directly trying to compute $\hat{\delta}_d$; see [Greuel and Pfister 2002, §2.8.7] for general background.

In ChevLie, this can be realised through the function `prop510` in Algorithm 1, which uses the interface `singular.jl`. In that function, we first assign the generic Gram matrix $\hat{\mathcal{G}}_d$ to the variable `gr`. Then we form the polynomial ring $R = \mathbb{F}_p[X_1, \ldots, X_m]$ where $m = \dim \mathfrak{g}_d(2)$ and p is given by the argument `pr`. The symbols in $\hat{\mathcal{G}}_d$ are evaluated to the variables X_i, which yields the new matrix `gr1` over R. We turn the rows of that matrix into SINGULAR vectors and, finally, apply the SINGULAR function `syz` to the module spanned by those row vectors. If `syz` returns nonzero vectors, then we will have $\hat{\delta}_d \equiv 0 \pmod{p}$.

Now we can also handle the hard cases in type E_8, which was first done by Steel and Thiel; see

[Geck 2018, Proposition 5.10]. We simply need to inspect the output of the following code (which takes less than a minute to run).

```
julia> lie=LieAlg(:e,8); wdd=weighted_dynkin_diagrams(lie)
julia> p2=[prop510(lie,d,2) for d in wdd[1]];  # prime p=2
julia> p3=[prop510(lie,d,3) for d in wdd[1]];  # prime p=3
julia> p5=[prop510(lie,d,5) for d in wdd[1]];  # prime p=5
```

In each case where d is not special, one verifies that there is a prime $p \in \{2, 3, 5\}$ such that `prop510` prints "`syz not 0`" and, hence, d is not δ-special either.

The original verification in [Geck 2018] relied on a "naive" interface between GAP and SINGULAR: using the older GAP version of ChevLie one could produce the matrices $\hat{\mathcal{G}}_d$ inside GAP, write them to a file in a format that could subsequently be read into SINGULAR, and finally perform the required syzygy computations in SINGULAR. With the new module ChevLie, all this can be done conveniently within Julia, together with the interface `singular.jl`. And everyone who wishes to do so, can easily verify the code and the computations.

SUPPLEMENT. The online supplement contains version 1.1 of `ChevLie`.

REFERENCES.

[Carter 1985] R. W. Carter, *Finite groups of Lie type: Conjugacy classes and complex characters*, John Wiley & Sons, New York, 1985. MR Zbl

[Cohen et al. 2004] A. M. Cohen, S. H. Murray, and D. E. Taylor, "Computing in groups of Lie type", *Math. Comp.* **73**:247 (2004), 1477–1498. MR

[Dong and Yang 2019] J. Dong and G. Yang, "Geck's Conjecture and the generalized Gelfand–Graev representations in bad characteristic", preprint, 2019. arXiv

[GAP] The GAP Group, "GAP – Groups, Algorithms, and Programming", available at https://www.gap-system.org.

[Gapjm.jl] J. Michel, "Experimental port of some GAP functionality to Julia", Julia package, available at https://github.com/jmichel7/Gapjm.jl.

[Geck 2017a] M. Geck, "Minuscule weights and Chevalley groups", pp. 159–176 in *Finite simple groups: thirty years of the atlas and beyond*, edited by M. Bhargava et al., Contemp. Math. **694**, Amer. Math. Soc., Providence, RI, 2017. MR

[Geck 2017b] M. Geck, "On the construction of semisimple Lie algebras and Chevalley groups", *Proc. Amer. Math. Soc.* **145**:8 (2017), 3233–3247. MR Zbl

[Geck 2018] M. Geck, "Generalised Gelfand–Graev representations in bad characteristic?", 2018. arXiv

[Geck 2020] M. Geck, "Computing Green functions in small characteristic", *J. Algebra* (2020).

[Geck et al. 1996] M. Geck, G. Hiss, F. Lübeck, G. Malle, and G. Pfeiffer, "CHEVIE – a system for computing and processing generic character tables", *Appl. Algebra Engrg. Comm. Comput.* **7**:3 (1996), 175–210. MR Zbl

[Greuel and Pfister 2002] G.-M. Greuel and G. Pfister, *A* **Singular** *introduction to commutative algebra*, Springer, 2002. MR Zbl

[Humphreys 1978] J. E. Humphreys, *Introduction to Lie algebras and representation theory*, Graduate Texts in Mathematics **9**, Springer, 1978. MR

[Humphreys 1995] J. E. Humphreys, *Conjugacy classes in semisimple algebraic groups*, Mathematical Surveys and Monographs **43**, American Mathematical Society, Providence, RI, 1995. MR Zbl

[Julia 2017] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing", *SIAM Review* **59**:1 (2017), 65–98. MR Zbl

[Lusztig 1990a] G. Lusztig, "On quantum groups", *J. Algebra* **131**:2 (1990), 466–475. MR Zbl

[Lusztig 1990b] G. Lusztig, "Quantum groups at roots of 1", *Geom. Dedicata* **35**:1-3 (1990), 89–113. MR Zbl

[Lusztig 1997] G. Lusztig, "Notes on unipotent classes", *Asian J. Math.* **1**:1 (1997), 194–207. MR

[Lusztig 2017] G. Lusztig, "The canonical basis of the quantum adjoint representation", *J. Comb. Algebra* **1**:1 (2017), 45–57. MR Zbl

[Mizuno 1980] K. Mizuno, "The conjugate classes of unipotent elements of the Chevalley groups E_7 and E_8", *Tokyo J. Math.* **3**:2 (1980), 391–461. MR Zbl

[OSCAR] "OSCAR – A comprehensive open source computer algebra system for computations in algebra, geometry, and number theory", available at https://oscar.computeralgebra.de/.

RECEIVED: 13 Jan 2020 REVISED: 17 Mar 2020 ACCEPTED: 21 Apr 2020

MEINOLF GECK:

meinolf.geck@mathematik.uni-stuttgart.de
IAZ – Lehrstuhl für Algebra, Universität Stuttgart, Stuttgart, Germany