Master of Arts Thesis

Nondegenerate Solutions to the Classical Yang-Baxter Equation, and Checking Solutions with `SINGULAR`.

by

Lisa Kierans

Supervisor: Dr. Bernd Kreussler

Submitted to Mary Immaculate College, University of Limerick,
November 2010

Abstract

This thesis investigates the nondegenerate solutions to the classical Yang-Baxter equation. Such solutions were investigated by A.A Belavin and V.G Drinfeld. In their seminal paper [4], they concluded that solutions to the classical Yang-Baxter equation, for finite-dimensional simple Lie algebras over the complex field $\mathbb{C}$ possessing this additional nondegeneracy property, fall into three classes up to equivalence: elliptic, trigonometric or rational. In this thesis, we outline the necessary preliminaries to formulate the classical Yang-Baxter equation in order to then reformulate the proofs of Belavin and Drinfeld in a coordinate-free language. We then briefly look at the associative Yang-Baxter equation and its relation to the classical equation. Finally, we produce a `SINGULAR` library named `ybe.lib` which offers the user a means to test whether or not an expression is a solution to the classical or associative Yang-Baxter equation.

Acknowledgements

My deepest thanks go to Bernd Kreussler who has very kindly and patiently guided me throughout my postgraduate studies. Bernd has always helped me to see a clearer and bigger picture and has set an excellent example for doing research as well as teaching. I would also like to thank my family and friends, who form an important group of people outside of Mathematics, for their encouragement and faith in me.

Introduction

Norwegian Sophus Lie (1842–1899) was a brilliant mathematician who began a new field of study by introducing what were later named Lie algebras. He developed the theory of “finite and continuous groups” and in 1873, outlined a general theory of not necessarily commutative groups. In principle, Lie’s theory reduced problems on Lie groups, of an analytic nature to algebraic problems on Lie algebras, and this led to an intrinsic study of Lie algebras. However, Lie did not pursue this and it was left to other mathematicians to go in that direction. The first fundamental contributions were due to Wilhelm Killing whose work from 1886-1890 focused on the classification of simple Lie algebras. When Killing started, he was aware of two infinite classes, the Lie algebras of the special linear group $\mathsf{SL}(n, \mathbb{C})$ and of the orthogonal groups $\mathsf{O}(n, \mathbb{C})$. Two years later he obtained the classification we have today: four infinite classes (the classical Lie algebras) and only finitely many other Lie algebras, now called the exceptional Lie algebras and denoted E_6, E_7, E_8, F_4, G_2 [7].

Elie Cartan’s thesis was the second fundamental contribution to Lie algebra theory. There he proved the existence of all the exceptional simple Lie algebras. Furthermore, a basic result in Cartan’s thesis is a criterion for a Lie algebra to be *semisimple* (i.e. the direct sum of (commuting) simple non-commutative subalgebras): namely, the Killing form is nondegenerate. Cartan continued to study Lie algebras and produced many important papers on the theory of semisimple Lie algebras. Other major contributors to the advance in the theory of semisimple Lie algebras include Hermann Weyl, Harish-Chandra Mehrota, Henri Poincaré, and Friedrich Engel to name but a few [14]. The study of Lie algebras has advanced much since then, in part because of their interest to physicists. Applications were known as early as the 1920s, with one of the earliest being the description of the electron configuration of atoms.

The Yang-Baxter equation is an equation which was first introduced in the field of statistical mechanics. It takes its name from the independent works of C.N. Yang from 1968, and R.J. Baxter from 1971. It refers to a principle in integrable systems taking the form of local equivalence transformations which appear in a variety of contexts, including electric networks, knot theory and braid groups, and spin systems.

In 1967, in studying integrable systems in quantum mechanics, C.N. Yang wrote down the matrix equation

$$A(u)B(u+v)A(v) = B(v)A(u+v)B(u) \tag{1}$$

and gave an explicit solution in which $A(u)$ and $B(u)$ are rational functions of u. This equation was also found by R.J. Baxter in 1972 when he studied a different integrable problem in classical statistical mechanics. Equation (1) has been extensively studied and has been named the *Yang-Baxter equation* [9]. At an early stage the Yang-Baxter equation appeared in several different guises in the literature, and sometimes its solutions have preceeded the equation.

The Yang-Baxter equation is a highly nonlinear equation, and is very difficult to be solved generally. It can be solved more easily if it is taken to its limit as one of its parameters tends to zero. Solving this limiting equation may prove helpful in finding new solutions of the Yang-Baxter equation. This equation is called the *classical Yang-Baxter equation* (CYBE) and it is much easier to be solved than the Yang-Baxter equation. The CYBE was first introduced by E.K. Sklyanin [22]. Compared to the Yang-Baxter equation, it represents an important and simplified case since it can be formulated in the language of Lie algebras. One of the directions of study in this domain is the classification of solutions in the case of a simple complex Lie algebra.

In [4], A.A. Belavin and V.G. Drinfeld investigate the nondegenerate solutions of the CYBE for a finite-dimensional, simple Lie algebras over the complex field $\mathbb{C}$. The authors prove that the poles of a nondegenerate solution form a discrete subgroup $\Gamma \subset \mathbb{C}$ and listed all solutions for rank $\Gamma = 2$ (elliptic) and rank $\Gamma = 1$ (trigonometric). Concerning the first class, the authors reduce the problem of finding nondegenerate elliptic solutions to the one of describing triples (L, A_1, A_2), where L is a simple Lie algebra, A_1 and A_2 are commuting automorphisms of L of finite order, not having common fixed nonzero vectors. Moreover, they prove that if such triples exist then there is an isomorphism $L \cong \mathsf{sl}(n)$. Belavin and Drinfeld also succeeded in classifying the trigonometric solutions using the data from the Dynkin diagram. Regarding rational solutions, (solutions with rank $\Gamma = 0$) the authors give several examples associated with Frobenius subalgebras of L and provide arguments in favour of the idea that there are too many rational solutions to try to list them. However, this problem was solved by A. Stolin who reduced the problem of listing "non-trivial" rational solutions of the CYBE to the classification of quasi-Frobenius subalgebras of L. There are a number of articles on rational solutions to the CYBE provided by Stolin, including [24, 25].

This thesis is motivated by the remarkable work of Belavin and Drinfeld in the 1980's. There are two main aims to the work of this thesis. The first aim is to reformulate the proofs of the theorems of Belavin and Drinfeld in a coordinate-free language. The second aim is to develop a computer program which enables the user to verify whether or not an expression satisfies the classical or *associative* Yang-Baxter equations. The *associative Yang-Baxter equation* (AYBE) was introduced by M. Aguiar in [1, 2] and again independently by A. Polishchuk in [19]. The algebraic meaning of the AYBE is beyond the scope of this thesis but is explained in [1, 2] and uses the notion of infinitesimal bialgebras.

The outline of the thesis is as follows: Chapters 1 and 2 serve as an introduction to the area of Lie algebras. In Chapter 1 we provide a brief overview of the fundamental concepts of Lie algebras, representations, the Killing form and Cartan subalgebras. Chapter 2 provides further background information, now focusing on root systems and Dynkin diagrams as well as the Universal Enveloping Algebra and the Casimir operator. Chapters 3 through 5 detail the core computations performed for the main theorems of Belavin and Drinfeld that this thesis focuses on. In Chapter 3, following the complex analysis preliminaries required, we consider a linear map from $L \otimes L \to \mathsf{Hom}(L, L)$ and a map from $L \otimes L \otimes L \to \mathsf{Hom}(L \otimes L, L)$, both of which are induced by the Killing form κ on the simple Lie algebra L. The remainder of this chapter is devoted to proving basic properties of these maps which are required in later chapters. In Chapter 4 the ideas from the preceding chapters are gathered to give a detailed account of the results found in [4]. A new feature of the present work is the language used to obtain these results, we replace the proofs of Belavin and Drinfeld, which use basis elements of the Lie algebra, with a language which uses the linear maps outlined above. The aim of this chapter is to reformulate the proof in [4] of the equivalence of 4 characterisations of nondegeneracy. Chapter 5 presents the main theorem we wish to prove: that nondegenerate solutions fall into three classes up to equivalence: elliptic, trigonometric and rational. We also give a brief outline of elliptic, trigonometric, and rational solutions with several examples of the latter two types of solution. Chapter 6 introduces the notion of the associative Yang-Baxter equation. We give a brief introduction and some examples of solutions to this equation. Solutions stated here and in Chapter 5 are utilised in Chapter 7, where we present a detailed description of the computer program produced. We begin with an overview of `SINGULAR`, the programming language used, we then give a description of the `ybe.lib` library with some sample code, and discuss any problems we encountered during implementation. Finally, Appendix 1 includes the code of our `SINGULAR` library `ybe.lib`.

Contents

Chapter 1

Lie Algebras

We will begin by defining Lie algebras and giving some typical examples to which we shall refer to throughout this thesis. We then introduce some fundamental concepts in Lie algebras which will be extensively used throughout this thesis. This chapter also introduces simple and semisimple Lie algebras, and the Killing form. The main source for this chapter is the book [11], which we have mostly followed. Our main interest lies in the structure of finite-dimensional simple and semisimple Lie algebras over the complex field $\mathbb{C}$. In order to investigate the structure of these Lie algebras we need to understand fully the above concepts. Unless otherwise stated, all Lie algebras in this thesis should be taken to be finite dimensional.

1.1 Introduction to Lie Algebras

Let V be an n-dimensional vector space over a field F. A *bilinear form* on V is a map

$$(-,-) : V \times V \to F$$

such that

$$\begin{aligned}(\lambda_1 v_1 + \lambda_2 v_2, w) &= \lambda_1(v_1, w) + \lambda_2(v_2, w),\\ (v, \mu_1 w_1 + \mu_2 w_2) &= \mu_1(v, w_1) + \mu_2(v, w_2),\end{aligned}$$

for all $v, w, v_i, w_i \in V$ and $\lambda_i, \mu_i \in F$.

Definition 1.1.1. Let V be an n-dimensional vector space over F. The *dual space* of V, denoted V^* is the set of all linear maps from V to F.

Let F be a field of characteristic zero. Throughout this thesis F will be $\mathbb{R}$ or $\mathbb{C}$. There are two classes of algebras over F with a bilinear product that have particularly good properties and so are very useful: *associative algebras*, where multiplication satisfies the associativity axiom $(ab)c = a(bc)$, and *Lie algebras*, which are the main focus of the next few chapters of this thesis.

Definition 1.1.2. A *Lie algebra* over F is an F-vector space L, together with a bilinear map, the *Lie bracket*

$$L \times L \to L, \qquad (x, y) \mapsto [x, y],$$

satisfying the following properties:

$$[x, x] = 0 \quad \text{for all } x \in L, \tag{L1}$$

$$[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0 \quad \text{for all } x, y, z \in L \tag{L2}$$

Condition (L1) implies

$$0 = [x + y, x + y] = [x, x] + [x, y] + [y, x] + [y, y] = 0.$$

Therefore the Lie bracket is *skew-symmetric*, that is

$$[x, y] = -[y, x] \quad \text{for all } x, y \in L.$$

Condition (L2) is called the *Jacobi Identity.*

Remark 1.1.3. Let A be an associative algebra with $x, y \in A$. Defining $[x, y] := x \cdot y - y \cdot x$ makes A a Lie algebra.

Let us now look at some examples of Lie algebras.

Example 1.1.4. *Any vector space V has a Lie bracket defined by $[x, y] = 0$ for all $x, y \in V$. This is the* abelian *Lie algebra structure on V. In particular, the field F may be regarded as a one-dimensional abelian Lie algebra.*

Example 1.1.5. *Let V be a finite dimensional vector space over F, and denote by $\mathsf{End}(V)$ the set of all linear transformations $V \to V$. Then $\mathsf{End}(V)$ is a Lie algebra called the* general linear algebra*, denoted $\mathsf{gl}(V)$. The bracket operation $[-, -]$ is defined by*

$$[x, y] = x \circ y - y \circ x, \qquad \textit{for all } x, y \in \mathsf{gl}(V)$$

where $\circ$ denotes the composition of maps.
If we choose a basis for V, we may represent $\mathsf{gl}(V)$ *with the set of $n \times n$ matrices over F, denoted* $\mathsf{gl}(n, F)$*. A basis for* $\mathsf{gl}(n, F)$ *consists of the* matrix units e_{ij} *for $1 \leq i, j \leq n$. Here e_{ij} is the $n \times n$ matrix which has 1 in the ij-th position and 0's elsewhere. The dimension of* $\mathsf{gl}(n, F)$ *is n^2.*

Remark 1.1.6. It can be shown that

$$[e_{ij}, e_{kl}] = \delta_{jk} e_{il} - \delta_{il} e_{kj},$$

where δ is the Kronecker delta, defined by $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. This formula is often used when calculating in $\mathsf{gl}(n, F)$.

Example 1.1.7. *Let* $\mathsf{b}(n, F)$ *be the set of upper triangular matrices in* $\mathsf{gl}(n, F)$*, and let* $\mathsf{n}(n, F)$ *be the set of strictly upper triangular matrices in* $\mathsf{gl}(n, F)$*. These are both Lie algebras with the same Lie bracket as* $\mathsf{gl}(n, F)$*.*

Example 1.1.8. *Define* $\mathsf{sl}(n, F) = \{x \in \mathsf{gl}(n, F) \mid \mathsf{tr}(x) = 0\}$*, where* $\mathsf{tr}(x) = \sum x_{ii}$*, the sum of the diagonal elements of the matrix x. This is a Lie algebra since for any $x, y \in$* $\mathsf{sl}(n, F)$

$$\mathsf{tr}([x, y]) = \mathsf{tr}(xy - yx) = \mathsf{tr}(xy) - \mathsf{tr}(yx) = 0,$$

because $\mathsf{tr}(ab) = \mathsf{tr}(ba)$ *for all $a, b \in$* $\mathsf{gl}(n, F)$*.*
$\mathsf{sl}(n, F)$ *is called the* special linear algebra*, and has dimension $n^2 - 1$. As a vector space,* $\mathsf{sl}(n, F)$ *has a basis consisting of the e_{ij} for $i \neq j$ together with $e_{ii} - e_{i+1,i+1}$ for $1 \leq i \leq n - 1$.*

Remark 1.1.9. Sometimes the e_{ij} with $i < j$ are called *raising operators* and the e_{ij} with $i > j$ are called *lowering operators*

1.2 Fundamental Concepts

In order to construct Lie algebras we must first choose a basis $x_1, \dots x_n$ of L. Then

$$[x_i, x_j] = \sum_{k=1}^{n} a_{ij}^k x_k.$$

The a_{ij}^k are called the *structure constants*. The structure constants satisfy the following equations:

$$a_{ii}^k = 0 = a_{ij}^k + a_{ji}^k,$$

$$\sum_k (a_{ij}^k a_{kl}^m + a_{jl}^k a_{ki}^m + a_{li}^k a_{kj}^m) = 0.$$

This follows from the antisymmetry of the Lie bracket and the Jacobi identity (L2). We emphasise that the a_{ij}^k depend on the choice of basis of L: Different bases will, in general, give different structure constants.

Definition 1.2.1. A *Lie subalgebra* of a Lie algebra L is a vector subspace $A \subseteq L$ such that

$$[x, y] \in A \quad \text{for all } x, y \in A.$$

That is, a subspace of L which is closed under the Lie bracket.

One can easily see that Lie subalgebras are Lie algebras in their own right.

Definition 1.2.2. An *ideal* of a Lie algebra L is a subspace I of L such that

$$[x, y] \in I \qquad \text{for all } x \in L, y \in I.$$

Example 1.2.3. $\mathsf{sl}(n, F)$ *is an ideal of* $\mathsf{gl}(n, F)$*, and* $\mathsf{n}(n, F)$ *is an ideal of* $\mathsf{b}(n, F)$*.*

Definition 1.2.4. The *centre* of L is an ideal in L defined to be

$$Z(L) := \{x \in L \,|\, [x, y] = 0 \text{ for all } y \in L\}.$$

Example 1.2.5. *The centre of a* $\mathsf{gl}(n, F)$ *consists of scalar multiples of the identity matrix.*

Definition 1.2.6. Let I be an ideal of a Lie algebra L. The *centraliser* of I is defined to be

$$C_L(I) := \{x \in L \,|\, [x, a] = 0 \text{ for all } a \in I\}.$$

Proposition 1.2.7. *If I and J are ideals then the product of ideals*

$$[I, J] := \mathsf{span}\{[x, y] \,|\, x \in I, y \in J\}$$

is an ideal of L.

Proof. $[I, J]$ is by definition a subspace, so we only need to check that $[L, [I, J]] \subseteq [I, J]$. If $x \in I, y \in J$, and $u \in L$, the the Jacobi identity gives

$$[u, [x, y]] = [x, [u, y]] + [[u, x], y].$$

Here $[u, y] \in J$ as J is an ideal, so $[x, [u, y]] \in [I, J]$. Similarly, $[[u, x], y] \in [I, J]$. Therefore, their sum belongs to $[I, J]$. Since $[L, [I, J]]$ is spanned by elements of this form, we conclude that $[L, [I, J]] \subseteq [I, J]$. □

Definition 1.2.8. If L_1 and L_2 are Lie algebras over a field F, a map $\varphi : L_1 \to L_2$ is a *homomorphism* if φ is a linear map and

$$\varphi([x,y]) = [\varphi(x), \varphi(y)] \quad \text{for all } x, y \in L_1.$$

Isomorphisms and automorphisms are defined in the usual way.

Example 1.2.9. *An extremely important homomorphism is the* adjoint homomorphism. *If L is a Lie algebra, we define*

$$\mathsf{ad} : L \to \mathsf{gl}(L)$$

by $(\mathsf{ad}\ x)(y) := [x,y]$ for $x, y \in L$. To show that ad is a homomorphism we can prove that $\mathsf{ad}([x,y])(z) = [\mathsf{ad}(x), \mathsf{ad}(y)](z)$:

$$\begin{aligned}
[\mathsf{ad}(x), \mathsf{ad}(y)](z) &= \mathsf{ad}(x) \circ \mathsf{ad}(y)(z) - \mathsf{ad}(y) \circ \mathsf{ad}(x)(z) \\
&= \mathsf{ad}(x)([y,z]) - \mathsf{ad}(y)([x,z]) \\
&= [x,[y,z]] - [y,[x,z]] \\
&= [x,[y,z]] + [y,[z,x]] \\
&= [[x,y],z] \textit{(This is due to the Jacobi identity)} \\
&= \mathsf{ad}([x,y])(z).
\end{aligned}$$

Definition 1.2.10. A *derivation* of a Lie algebra L is an operator $D : L \to L$ that satisfies

$$D[x,y] = [Dx, y] + [x, Dy] \quad \text{for all } x, y \in L.$$

Remark 1.2.11. Let $\mathsf{Der}\, A$ be the set of derivations of A. $\mathsf{Der}\, A$ is a Lie subalgebra of $\mathsf{gl}(A)$.

Example 1.2.12. *Let L be a Lie algebra and let $x \in L$. The map $\mathsf{ad}\, x : L \to L$ is a derivation of L since*

$$\begin{aligned}
(\mathsf{ad}\, x)[y,z] = [x,[y,z]] &= [[x,y],z] + [y,[x,z]] \\
&= [(\mathsf{ad}\, x)(y), z] + [y, (\mathsf{ad}\, x)(z)].
\end{aligned}$$

Remark 1.2.13. The adjoint maps are called *inner derivations.* All other derivations are *outer.*

1.3 Simple and Semisimple Lie algebras

This section introduces simple and semisimple Lie algebras. Every semisimple Lie algebra is a sum of simple ideals, each of which can be treated as a separate simple Lie algebra. Simple and semisimple Lie algebras are one of the most widely studied classes of algebras as their representation theory is very well understood and there is a classification of simple Lie algebras (see Chapter 3). Throughout this section we will assume that L is defined over $\mathbb{C}$.

Definition 1.3.1. The *derived algebra* L' of a Lie algebra L is the ideal $[L, L]$ generated by $[a, b]$ for all $a, b \in L$.

Remark 1.3.2. The Lie algebra $L = \mathsf{sl}(2)$ of 2×2 matrices with zero trace, has as a basis

$$e = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad f = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad h = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

We can check that

$$[e, f] = h, \quad [h, e] = 2e, \quad [h, f] = -2f.$$

Therefore, $[\mathsf{sl}(2), \mathsf{sl}(2)] = \mathsf{sl}(2)$, that is, $L = L'$. Up to isomorphism, $\mathsf{sl}(2)$ is the only 3-dimensional complex Lie algebra with this property.

The derived algebra is an important ideal of L and we can use it repeatedly to construct the *derived series* of L:

$$L^{(1)} = [L, L], \quad L^{(r+1)} = [L^{(r)}, L^{(r)}], \text{ that is, } L^{(r+1)} = L^{(r)\prime}$$

with $L \supseteq L^{(1)} \supseteq L^{(2)} \supseteq \ldots$. As the product of ideals is an ideal, by induction we can show that the $L^{(r)}$ are ideals of L (the first inclusion is due to the Jacobi identity):

$$[L^{(r+1)}, L] = [[L^{(r)}, L^{(r)}], L] \subset [[L^{(r)}, L], L^{(r)}] \subset [L^{(r)}, L^{(r)}] = L^{(r+1)}.$$

Definition 1.3.3. The Lie algebra L is said to be *solvable* if for some $m \geq 1$ we have $L^{(m)} = 0$.

We can also iterate the construction of the derived algebra in another way:

$$L' = L^1 = [L, L], \quad \text{and } L^{r+1} = [L, L^r] \text{ for } k \geq 2$$

with $L \supseteq L^1 \supseteq L^2 \supseteq \ldots$. This means that L^r is generated by iterated brackets of $r + 1$ elements $[a_1[a_2[a_3[\ldots[a_r, a_{r+1}]]]]]$. This series is called the *lower central series.*

Definition 1.3.4. The Lie algebra L is said to be *nilpotent* if for some $m \geq 1$ we have $L^m = 0$.

Theorem 1.3.5. *A Lie algebra L is nilpotent if and only if for all $x \in L$ the linear map $\mathsf{ad}\, x : L \to L$ is nilpotent.*

Proof. For a proof see [11] Theorem 6.3. □

Lemma 1.3.6. *The Lie algebra $\mathsf{b}(n)$ is solvable.*

Proof. We want to show that for $L = \mathsf{b}(n)$, $L^{(t+1)} = 0$ for some t. We do this by induction on $t \geq 1$. For the inductive step, we assume that $L^{(t)}$ is generated by $G_t := \{e_{ab} \,|\, a \leq b - 2^{t-1}\}$. Accordingly, we have to show that for all $e_{ab}, e_{cd} \in G_t$ we have at least one of $\pm[e_{ab}, e_{cd}]$ is in G_{t+1} or zero otherwise, and for all $e_{ab} \in G_{t+1}$ there exists $e_{ij}, e_{kl} \in G_t$ such that $[e_{ij}, e_{kl}] = e_{ab}$. To start the induction, we look at the case $t = 1$:

(i) Let $e_{ab}, e_{cd} \in L$, that is, $a \leq b$ and $c \leq d$. We find

$$[e_{ab}, e_{cd}] = \begin{cases} e_{ad} & \text{when } b = c \text{ and } a \neq d, \\ -e_{cb} & \text{when } a = d \text{ and } b \neq c, \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

It follows that $a \leq b$ and $c \leq d$ along with $a \neq d$ and $b \neq c$ respectively, means that e_{ad} or $-e_{cd}$ are in G_1. If $b = c$ we have $a \leq b = c \leq d$ and $a =\neq$, and so $[e_{ab}, e_{cd}] = e_{ad}$ with $a < d$. If $a = d$ we have $c \leq d = a \leq b$ and $b \neq c$, and so $[e_{ab}, e_{cd}] = -e_{cb}$ with $c < b$.

(ii) For each $e_{ab} \in G_1$, $[e_{aa}, e_{ab}] = e_{ab}$.

Therefore, $L^{(1)}$ is generated by $G_1 = \{e_{ab} \,|\, a \leq b - 1\}$. For the inductive step we calculate G_{t+1}:

(i) Let $e_{ab}, e_{cd} \in G_t$, that is, $a \leq b - 2^{t-1}$ and $c \leq d - 2^{t-1}$. Then, if nonzero, $[e_{ab}, e_{cd}]$ is equal to e_{ad} or to $-e_{cb}$ according to Equations (1.1). If $b = c$ we have $a + 2^{t-1} \leq b = c \leq d - 2^{t-1}$ and so $a \leq d - 2^t$. If $a = d$ we have $c + 2^{t-1} \leq d = a \leq b - 2^{t-1}$ and so $c \leq b - 2^t$. Therefore, all nonzero $\pm[e_{ab}, e_{cd}]$ are in G_{t+1}.

(ii) If $e_{ab} \in G_{t+1}$ then $a \leq b - 2^k$ or $2^k \leq b - a$, so for $j = a + 2^{t-1}$ we have $[e_{aj}, e_{jb}] = e_{ab}$.

Hence, $L^{(k+1)}$ is generated by $G_{t+1} = \{e_{ab} \,|\, a \leq b - 2^k\}$. If $e_{ab} \in L$, then $n > b - a \geq 0$. For large t we have $2^t > n$ hence $G_t = \emptyset$ and $L^{(t)} = 0$. □

Remark 1.3.7. From Remark 1.3.2 it is clear that for $\mathsf{sl}(2)$, $\mathsf{sl}(2)^{(m)} = \mathsf{sl}(2)$ for all $m \geq 1$, so $\mathsf{sl}(2)$ is not solvable.

Definition 1.3.8. The largest solvable ideal of a Lie algebra is said to be the *radical* of L and is denoted $\mathsf{rad}\, L$.

Remark 1.3.9. The last non-zero term of the derived series of the radical is an abelian ideal of L. Therefore, the radical of L is zero if and only if L has no non-zero abelian ideals.

Definition 1.3.10. Lie algebras possessing no non-zero abelian ideals are called *semisimple*.

Remark 1.3.11. As a semisimple Lie algebra has no non-zero abelian ideals, it cannot be solvable.

These Lie algebras and their representations will be the main focus of our investigation into Lie algebras.

Definition 1.3.12. The Lie algebra L is *simple* if its only ideals are 0 and itself and it is not abelian.

Remark 1.3.13. If L is simple, then $[L, L]$ is a nonzero ideal in L and consequently $[L, L] = L$. Therefore, L is not solvable.

Proposition 1.3.14. *For each $n \geq 2$, $\mathsf{sl}(n, \mathbb{C})$ is simple*

The basis elements for $\mathsf{sl}(n, \mathbb{C})$ outlined in Example 1.1.8 are convenient for Lie algebra calculations because the Lie bracket of any x with an e_{ij} has few nonzero entries. This enables us to take any nonzero element of an ideal J and manipulate it to find a nonzero multiple of each basis element in J, thus showing that $\mathsf{sl}(n, \mathbb{C})$ has no nontrivial ideals. The proof is as follows:

Proof. ([23] Section 6.4) Let $x = (x_{ij})$ be any $n \times n$ matrix, then $[x, e_{ij}]$ has all columns zero except the j-th, which is occupied by the i-th column of x, and $-e_{ij}x$ has all rows zero except the i-th, which is occupied by -(row j) of x.

Therefore, since $[x, e_{ij}] = xe_{ij} - e_{ij}x$, we have

$$\text{column } j \text{ of } [x, e_{ij}] = \begin{pmatrix} x_{1i} \\ \vdots \\ x_{i-1,i} \\ x_{ii} - x_{jj} \\ x_{i+1,1} \\ \vdots \\ x_{ni} \end{pmatrix}$$

and

$$\text{row } i \text{ of } [x, e_{ij}] = \begin{pmatrix} -x_{j1} & \dots & -x_{j,j-1} & x_{ii} - x_{jj} & -x_{j,j+1} & \dots & -x_{jn} \end{pmatrix}$$

and all other entries of $[x, e_{ij}]$ are zero. In the (i, j)-position, where the shifted row and column cross, we get the element $x_{ii} - x_{jj}$.
We now use such bracketing to show that an ideal J with a nonzero member x includes all the basis elements of $\mathsf{sl}(n, \mathbb{C})$, so $J = \mathsf{sl}(n, \mathbb{C})$.

Case (i) x has nonzero entry x_{ji} for some $i \neq j$. Multiply $[x, e_{ij}]$ by e_{ij} on the right. This destroys all columns except the ith, whose only nonzero term is $-x_{ji}$ in the (i, i)-position, moving it to the (i, j)-position (because column i is moved to column j position).
Now, multiply $[x, e_{ij}]$ by $-e_{ij}$ on the left. This destroys all rows except the jth, whose only nonzero element is x_{ji} at the (j, j)-position, moving it to the (i, j)-position and changing its sign (because row j is moved to row i position with a sign change).
It follows that $[x, e_{ij}]e_{ij} - e_{ij}[x, e_{ij}] = [[x, e_{ij}], e_{ij}]$ contains the nonzero element $-2x_{ji}$ at the (i, j)-position, and zeros elsewhere. Thus the ideal J containing x also contains e_{ij}. By further bracketing we can show that *all* the basis elements of $\mathsf{sl}(n, \mathbb{C})$ are in J.

Case (ii) All the nonzero entries of x are among $x_{11}, x_{22}, \dots, x_{nn}$.
Not all these elements are equal (otherwise $\mathsf{tr}(x) \neq 0$), so we can choose i and j such that $x_{ii} - x_{jj} \neq 0$. Now, for this x, the calculations of $[x, e_{ij}]$ gives

$$[x, e_{ij}] = (x_{ii} - x_{jj})e_{ij}.$$

Thus J includes a nonzero multiple of e_{ij}, and hence e_{ij} itself. We can now repeat the rest of the argument in Case (i) to conclude that $J = \mathsf{sl}(n, \mathbb{C})$.

so, $\mathsf{sl}(n, \mathbb{C})$ is simple. □

Definition 1.3.15. Let V be a vector space. A non-zero vector $v \in V$ such that $x(v) = \lambda v$ is said to be an *eigenvector* of x with corresponding *eigenvalue* λ. The *eigenspace* for eigenvalue λ is the vector subspace $\{v \in V \mid x(v) = \lambda v\}$.

We can generalise these notions of eigenvalues and eigenvectors. Let A be a subalgebra of $\mathsf{gl}(V)$, then $v \in V$ is an eigenvector for A if v is an eigenvector for every element of A. That is, $a(v) \in \mathsf{span}\{v\}$ for every $a \in A$. However, different elements of A act with different eigenvalues. We can specify the *eigenvalues* of elements of A by giving a function $\lambda : A \to F$. The corresponding *eigenspace* is then

$$V_\lambda := \{v \in V \mid a(v) = \lambda(a)v \text{ for all } a \in A\}.$$

Lemma 1.3.16. *Let L be a simple complex Lie algebra. Let $\psi \in \mathsf{End}(L)$, then $[\psi, \mathsf{ad}(h)] = 0$ for all $h \in L$ if and only if $\psi \in \mathbb{C} \cdot \mathbb{1}_L$.*

Proof. First let us assume that $\psi \in \lambda \cdot \mathbb{1}_L$, $\lambda \in \mathbb{C}$, then from Example 1.2.5 we can see that $[\psi, \mathsf{ad}(h)] = 0$ for all $h \in L$.

Now, let us assume that $[\psi, \mathsf{ad}(h)] = 0$ for all $h \in L$. Then as L is a complex vector space, there exists a nonzero $x_0 \in L$ which is an eigenvector of ψ, that is, $\psi(x_0) = \lambda x_0$, $\lambda \in \mathbb{C}$. Let

$$\{0\} \neq V_\lambda = \{x \in L \mid \psi(x) = \lambda x\} \subset L$$

be the eigenspace for λ. We will now show that V_λ is an ideal in L, and therefore equal to L (because L is simple). To show this we must prove that $\psi([x, h]) = \lambda([x, h])$ for all $x \in V_\lambda$, $h \in L$. We know from the assumption that $(\psi \circ \mathsf{ad}(h))(x) = (\mathsf{ad}(h) \circ \psi)(x)$, that is, for all $x \in L$, $\psi([h, x]) = [h, \psi(x)]$. If $x \in V_\lambda$ then $\psi(x) = \lambda x$. Therefore, $\psi([h, x]) = [h, \lambda x] = \lambda[h, x]$. This proves that $V_\lambda = L$, therefore $\psi(x) = \lambda x$ for all $x \in L$, and so $\psi = \lambda \cdot \mathbb{1}_L$. □

Theorem 1.3.17. *Let L be a simple Lie algebra. Suppose that ψ is an automorphism of L. Then there exists a nonzero element $x \in L$ such that $\psi(x) = x$. That is, $(\psi - \mathbb{1}_L)(x) = 0$.*

Proof. See [6] Theorem 9.2. □

1.4 Representations and Modules of Lie Algebras

To examine the ways in which an abstract Lie algebra can be viewed concretely, we can use representations. Representations are very useful tools for analysing the structure of Lie algebras by mapping them to matrix Lie algebras where we can apply linear algebra techniques to study the image and gain information about the abstract Lie algebra. In this section we aim to introduce the notions of representations and modules of Lie algebras and prove a very useful lemma called Schur's Lemma.

Definition 1.4.1. Let L be a Lie algebra over a field F. A *representation of L* is a Lie algebra homomorphism

$$\varphi : L \to \mathsf{gl}(V)$$

where V is a finite-dimensional vector space over F.

For brevity, we will sometimes omit to mention the homomorphism and just say that V is a representation of L. We can now fix a basis of V and write the linear transformations of V provided by elements of L as matrices. A representation is called *faithful* if its kernel is zero, it is called *irreducible* if it has no nontrivial invariant subspaces. The dimension of a representation is by definition the dimension of the vector space V

Example 1.4.2. *From Example 1.2.9 we know that the adjoint map* ad *is a homomorphism* $\mathsf{ad} : L \to \mathsf{gl}(L)$. *Therefore,* ad *provides a representation of L with $V = L$ called the adjoint representation.*

The adjoint representation plays an important role in the study of Lie algebras.

Remark 1.4.3. By the definition of the map ad, one has that $Z(L) = \ker(\mathsf{ad})$. Hence, the adjoint representation of a Lie algebra L is faithful precisely when $Z(L) = 0$.

Remark 1.4.4. A semisimple Lie algebra cannot have a non-trivial centre, so the adjoint representation of a semisimple Lie algebra is faithful.

Example 1.4.5. *To compute the adjoint representation of* $\mathsf{sl}(2,\mathbb{C})$ *we will use the basis given in Remark 1.3.2. Take, for example, the map* $\mathsf{ad}\, h$*:*

$$\begin{aligned}\mathsf{ad}\, h(h) &= [h,h] = 0\cdot h + 0\cdot e + 0\cdot f\\ \mathsf{ad}\, h(e) &= [h,e] = 0\cdot h + 2\cdot e + 0\cdot f\\ \mathsf{ad}\, h(f) &= [h,f] = 0\cdot h + 0\cdot e - 2\cdot f.\end{aligned}$$

Similar calculations for $\mathsf{ad}\, e$ *and* $\mathsf{ad}\, f$ *show that the matrix representations of* $\mathsf{sl}(2,\mathbb{C})$ *are:*

$$\mathsf{ad}\, h = \begin{pmatrix}0&0&0\\0&2&0\\0&0&-2\end{pmatrix}\quad \mathsf{ad}\, e = \begin{pmatrix}0&0&1\\-2&0&0\\0&0&0\end{pmatrix}\quad \mathsf{ad}\, f = \begin{pmatrix}0&-1&0\\0&0&0\\2&0&0\end{pmatrix}.$$

Definition 1.4.6. Suppose that L is a Lie algebra over a field F. A *Lie module* for L, or an *L-module*, is a finite-dimensional F-vector space V together with a map

$$L\times V\to V\quad (x,v)\mapsto x\cdot v$$

satisfying the conditions

$$(\lambda x+\mu y)\cdot v = \lambda(x\cdot v)+\mu(y\cdot v) \tag{M1}$$

$$x\cdot(\lambda v+\mu w) = \lambda(x\cdot v)+\mu(x\cdot w) \tag{M2}$$

$$[x,y]\cdot v = x\cdot(y\cdot v) - y\cdot(x\cdot v) \tag{M3}$$

for all $x,y\in L$, $v,w,\in V$, and $\lambda,\mu\in F$.

The Lie module V is said to be *irreducible*, or *simple*, if it is non-zero and it has no submodules other than 0 and V.

Remark 1.4.7. If L is a simple Lie algebra, then L viewed as an L-module via the adjoint representation is irreducible.

Definition 1.4.8. If V is an L-module such that $V = U\oplus W$, where both U and W are L-submodules of V, we say that V is the *direct sum* of the L-modules U and W. The module V is said to be *indecomposable* if there are no non-zero submodules U and W such that $V = U\oplus W$.

Clearly, an irreducible module is indecomposable. The converse does not usually hold. One of the best ways to understand the structure of a module for a Lie algebra is to look at the homomorphisms between it and other modules.

Lemma 1.4.9 (Schur's Lemma)**.** *Let L be a complex Lie algebra and let S be a finite-dimensional irreducible L-module. A map $\theta : S \to S$ is an L-module homomorphism if and only if θ is a scalar multiple of the identity transformation; that is, $\theta = \lambda \mathbb{1}_S$ for some $\lambda \in \mathbb{C}$.*

Proof. ([11] Lemma 7.13) The"if" direction should be clear. For the "only if" direction, suppose that $\theta : S \to S$ is an L-module homomorphism. Then θ is, in particular, a linear map of a complex vector space, and so it must have an eigenvector $v \in S, v \neq 0$ with eigenvalue, say λ. Now $\theta - \lambda \mathbb{1}_S$ is also and L-module homomorphism, with $(\theta - \lambda \mathbb{1}_S)(v) = 0$. Therefore $v \in \ker(\theta - \lambda \mathbb{1}_S)$, and so it is a non-zero submodule of S. As S is irreducible $S = \ker(\theta - \lambda \mathbb{1}_S)$; that is, $\theta = \lambda \mathbb{1}_S$. □

1.5 The Root Space Decomposition of L

The Root Space Decomposition (RSD) of a semisimple Lie algebras reveals a lot about structural infomation of the Lie algebra. The starting point in the construction of the RSD is the identification of a certain abelian subalgebra of L called the Cartan subalgebra. Much of the structure theory of semisimple Lie algebras is based on the fact that every semisimple Lie algebra over $\mathbb{C}$ contains a Cartan subalgebra.

Theorem 1.5.1. *Let L be a complex semisimple Lie algebra. Each $x \in L$ can be written uniquely as $x = d+n$ (the* Jordan Decomposition *of x) where $d, n \in L$ are such that $\mathsf{ad}\, d$ is diagonalisable, $\mathsf{ad}\, n$ is nilpotent, and $[d, n] = 0$. Furthermore, if $y \in L$ commutes with x, then $[d, y] = 0$ and $[n, y] = 0$.*

Proof. See [11] Theorem 9.15 □

Remark 1.5.2. If $n = 0$ then we say that x is a *semisimple element* of L.

Definition 1.5.3. A Lie subalgebra H of a Lie algebra L is called a *Cartan subalgebra* if H is abelian and every element $h \in H$ is semisimple.

Proposition 1.5.4. *Let L be a complex semisimple Lie algebra. Then L has a non-zero Cartan subalgebra.*

Proof. We first need to show that L must contain semisimple elements. If $x \in L$ has Jordan decomposition $x = d + n$, then by Theorem 1.5.1, $d, n \in L$. If d, the semisimple part, were always zero, then by Theorem 1.3.5, L would be nilpotent and therefore solvable. Hence, we can find a non-zero semisimple element $d \in L$. We can obtain a non-zero Cartan subalgebra of L by taking any subalgebra which contains d and which is maximal subject to being abelian and consisting of semisimple elements. This subalgebra must exist because L is finite-dimensional. □

Lemma 1.5.5. *If H is a Cartan subalgebra of a complex semisimple Lie algebra L, then $H = C_L(H)$.*

Proof. For a proof see Section 10.2 of [11] □

Cartan subalgebras are our main tool in uncovering the structure and the classification of semisimple Lie algebras. Suppose that L is a complex semisimple Lie algebra containing an abelian subalgebra H consisting of semisimple elements. Then L has a basis of common eigenvectors for the elements of $\mathsf{ad}\, H$. Given a common eigenvector $x \in L$, the eigenvalues are given by the associated *weight*, $\alpha : H \to \mathbb{C}$, defined by

$$(\mathsf{ad}\, h)x = \alpha(h)x \text{ for all } h \in H.$$

Weights are elements of the dual space H^*. For each $\alpha \in H^*$, let

$$L_\alpha := \{x \in L \,|\, [h, x] = \alpha(h)x \text{ for all } h \in H\}$$

denote the corresponding weight space. One of these weight spaces is the zero weight space:

$$L_0 = \{z \in L \,|\, [h, z] = 0 \text{ for all } h \in H\}.$$

This is the same as the centraliser of H in L (Definition 1.2.6). From Lemma 1.5.5 we have $H = C_L(H)$, so the direct sum decomposition of L into weight spaces for H may be written as

$$L = H \oplus \bigoplus_{\alpha \in \Phi} L_\alpha \tag{1.2}$$

where Φ is the set of $\alpha \in H^*$ such that $\alpha \neq 0$ and $L_\alpha \neq 0$. Since L is finite-dimensional, Φ is finite. The elements of Φ are called the *roots* of L with respect to h, and the L_α are called the *root spaces*. The direct sum decomposition above, Equation (1.2), is the *root space decomposition*.

Example 1.5.6. *Consider* $\mathsf{sl}(3, \mathbb{C})$*. We will work with the following basis:*

$$x_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad y_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad h_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$
$$x_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad y_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad h_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$
$$x_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad y_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

The subspace $\mathsf{span}\{x_1, y_1, h_1\}$ *is a subalgebra of* $\mathsf{sl}(3, \mathbb{C})$ *isomorphic to* $\mathsf{sl}(2, \mathbb{C})$ *(this can be seen if you ignore the 3rd row and 3rd column). Similarly,*
$\mathsf{span}\{x_2, y_2, h_2\}$ *is a subalgebra of* $\mathsf{sl}(3, \mathbb{C})$ *isomorphic to* $\mathsf{sl}(2, \mathbb{C})$*. Thus, using the results of Remark 1.3.2 we have the following commutator relations:*

$$\begin{array}{ll} [h_1, x_1] = 2x_1 & [h_2, x_2] = 2x_2 \\ [h_1, y_1] = -2y_1 & [h_2, y_2] = -2y_2 \\ [x_1, y_1] = h_1 & [x_2, y_2] = h_2. \end{array}$$

For $\mathsf{sl}(3, \mathbb{C})$ *we know that the Cartan subalgebra consists of elements* h_1 *and* h_2*. Therefore, we can construct the matrix* $\mathsf{ad}\, h$ *for* $h = ah_1 + bh_2$ *as follows:*

$$\begin{aligned} \mathsf{ad}(ah_1 + bh_2)x_1 &= [ah_1 + bh_2, x_1] = (2a - b)x_1 \\ \mathsf{ad}(ah_1 + bh_2)y_1 &= (-2a + b)y_1 \\ \mathsf{ad}(ah_1 + bh_2)h_1 &= 0h_1 \\ \mathsf{ad}(ah_1 + bh_2)x_2 &= (-a + 2b)x_2 \\ \mathsf{ad}(ah_1 + bh_2)y_2 &= (a - 2b)y_2 \\ \mathsf{ad}(ah_1 + bh_2)h_2 &= 0h_2 \\ \mathsf{ad}(ah_1 + bh_2)x_3 &= (a + b)x_3 \\ \mathsf{ad}(ah_1 + bh_2)y_3 &= (-a - b)y_3. \end{aligned}$$

The matrix looks like

$$\mathsf{ad}\, h = \begin{pmatrix} 2a - b & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2a + b & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -a + 2b & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a - 2b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a + b & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -a - b \end{pmatrix}.$$

1.6 The Killing Form

In order to analyse roots and root spaces in more detail, we need to define a Euclidean inner product on the root spaces. This is provided by the Killing form on L. The Killing form of a Lie algebra is the trace form corresponding to the adjoint representation. The requirement that the Killing form be nondegenerate poses strong restrictions on the structure of L. In particular, we will show that this is true if and only if L is semisimple. We also show that all derivations of L must be of the form $\mathsf{ad}\, x$ for some $x \in L$, and we will look at a linear map φ which is induced by a symmetric, bilinear, associative form β. Throughout this section, unless otherwise stated, L is a semisimple Lie algebra over $\mathbb{C}$.

Remark 1.6.1. Let $x, y, z \in \mathsf{gl}(V)$ and let V be finite dimensional. Then

$$\mathsf{tr}([x, y]z) = \mathsf{tr}(xyz - yxz) = \mathsf{tr}(xyz - xzy) = \mathsf{tr}(x[y, z]).$$

Recall that the adjoint representation associates to an element $x \in L$ a linear transformation $\mathsf{ad}(x) : L \to L$.

Definition 1.6.2. The *Killing form* on L is the symmetric bilinear form defined by

$$\kappa(x, y) := \mathsf{tr}(\mathsf{ad}\ x \circ \mathsf{ad}\ y) \quad \text{for all } x, y, \in L,$$

that is, the trace of the composition of linear transformations $\mathsf{ad}(x)$ and $\mathsf{ad}(y)$, sending $a \in L$ to $[x, [y, a]]$.

The Killing form is bilinear because ad is linear, the composition of maps is bilinear, and tr is linear. It is symmetric because $\mathsf{tr}(ab) = \mathsf{tr}(ba)$ for linear maps a and b. Another very important property of the Killing form is its *associativity*, which states that for all $x, y, z \in L$, we have

$$\kappa([x, y], z) = \kappa(x, [y, z]).$$

This follows from the identity for trace mentioned above. Equivalently,

$$-\kappa([y, x], z) = \kappa(x, [y, z]).$$

much of the usefulness of κ is due to this identity. We shall see that the properties of the Killing form of a Lie algebra L say a lot about L.

Lemma 1.6.3. *Let I be an ideal in L. If $x, y \in I$, then $\kappa_I(x, y) = \kappa(x, y)$*

Proof. See [11] Lemma 9.8 □

Definition 1.6.4. Define the kernel of a symmetric bilinear form β on a finite-dimensional complex vector space V to be

$$\ker \beta := \{v \in V \mid \beta(v, w) = 0, \text{ for all } w \in V\}$$

This is a vector subspace of V. A symmentric bilinear form is called *nondegenerate* if its kernel is zero, that is, if there is no non-zero vector $v \in V$ such that $\beta(v, x) = 0$ for all $x \in V$. We also define the *orthogonal complement* of a subspace A relative to β to be

$$A^\perp := \{v \in V \mid \beta(v, w) = 0, \text{ for all } w \in A\}.$$

Remark 1.6.5. If I is an ideal of L and $\beta = \kappa$ is the Killing form, then, due to the associativity of κ, $I^\perp$ is also an ideal. This implies that $L^\perp$ is an ideal of L.

Example 1.6.6. *As an example of the Killing form, consider again* $\mathsf{sl}(3, \mathbb{C})$*. Using the results from Example 1.5.6, the corresponding table of commutators reads, where for example,* $[x_2, x_1] = -x_3$*:*

	x_1	x_2	x_3	y_1	y_2	y_3	h_1	h_2
x_1	0	$-x_3$	0	$-h_1$	0	y_2	$2x_1$	$-x_1$
x_2		0	0	0	$-h_2$	$-y_1$	$-x_2$	$2x_2$
x_3			0	x_2	$-x_1$	$-(h_1 + h_2)$	x_3	x_3
y_1				0	y_3	0	$-2y_1$	y_1
y_2					0	0	y_2	$-2y_2$
y_3						0	$-y_3$	$-y_3$
h_1							0	0
h_2								0

From this table we can easily calculate the matrix for each basis element under the adjoint representation. Taking the Killing form between two of our basis elements, only a few are non-zero:

$$\begin{array}{lll} \kappa(x_1, y_1) = 6, & \kappa(x_2, y_2) = 6, & \kappa(x_3, y_3) = 6 \\ \kappa(h_1, h_2) = -6, & \kappa(h_1, h_1) = 12, & \kappa(h_2, h_2) = 12. \end{array}$$

Therefore, the Killing form of $\mathsf{sl}(3,\mathbb{C})$ *is:*

$$\kappa = 6 \cdot \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Definition 1.6.7. Let L be a Lie algebra. Let $\psi : L \to L$ be a homomorphism. We will denote the "adjoint" of ψ to be the linear map ψ^* which satisfies $\kappa(\psi(x), y) = \kappa(x, \psi^*(y))$.

Remark 1.6.8. The adjoint map ψ^* is unique as κ is nondegenerate.

We can use the Killing form and the definition of "adjoint" above to prove the following Lemma which will prove useful in later chapters:

Lemma 1.6.9. *Let L be a simple Lie algebra. Let ψ be an endomorphism of L as a Lie algebra. Then* $\det \psi \in \{0, 1, -1\}$.

Proof. Suppose that $\psi \neq 0$. As $\psi : L \to L$ is an endomorphism, then $\ker \psi$ is an ideal in L different from L. But L is simple, therefore, $\ker \psi = 0$ and so ψ is an automorphism. As ψ is an endomorphism, we know,

$$\psi([x, y]) = [\psi(x), \psi(y)],$$

which can be written as,

$$(\psi \circ \mathsf{ad}(x))(y) = (\mathsf{ad}(\psi(x)) \circ \psi)(y)$$

therefore,

$$\begin{aligned} \psi \circ \mathsf{ad}(x) &= \mathsf{ad}(\psi(x)) \circ \psi \\ \psi \circ \mathsf{ad}(x) \circ \mathsf{ad}(y) &= \mathsf{ad}(\psi(x)) \circ \psi \circ \mathsf{ad}(y) \\ &= \mathsf{ad}(\psi(x)) \circ \mathsf{ad}(\psi(y)) \circ \psi \end{aligned}$$

and so, $\psi \circ \mathsf{ad}(x) \circ \mathsf{ad}(y) \circ \psi^{-1} = \mathsf{ad}(\psi(x)) \circ \mathsf{ad}(\psi(y))$. We want to show that ψ preserves the Killing form κ. We know that $\kappa(x, y) := \mathsf{tr}(\mathsf{ad}(x) \circ \mathsf{ad}(y))$ and from $\mathsf{tr}(ABA^{-1}) = \mathsf{tr}(B)$, we have

$$\mathsf{tr}(\psi \circ \mathsf{ad}(x) \circ \mathsf{ad}(y) \circ \psi^{-1}) = \mathsf{tr}(\mathsf{ad}(x) \circ \mathsf{ad}(y)) = \mathsf{tr}(\mathsf{ad}(\psi(x)) \circ \mathsf{ad}(\psi(y)))$$

Hence, we can write

$$\kappa(x,y) = \kappa(\psi(x),\psi(y)) \tag{1.3}$$

that is, ψ preserves the Killing form. Now if we let $y = \psi^{-1}(\bar{y})$ in Equation (1.3) we obtain

$$\kappa(x,\psi^{-1}(\bar{y})) = \kappa(\psi(x),\bar{y}).$$

That is, $\psi^{-1} = \psi^*$ the adjoint map. We know that $\psi \circ \psi^{-1} = \mathbb{1}_L = \psi \circ \psi^*$. Then $1 = \det(\psi)\det(\psi^*) = (\det(\psi))^2$. Therefore, $\det \psi = \pm 1$. □

Cartan's criterion for semisimplicity yield a straightforward way of checking whether a given Lie algebra is semisimple.

Theorem 1.6.10 (Cartan's First Criterion)**.** *The complex Lie algebra L is solvable if and only if $\kappa(x,y) = 0$ for all $x \in L$ and $y \in L'$.*

Proof. See [21] Section 1.9 □

Cartan's first criterion characterises solvable Lie algebras as those for which the derived Lie algebra L' is contained in the kernel of the Killing form of L. The opposite case are the algebras for which the Killing form is nondegenerate.

The following remark proves useful in proving Cartan's second criterion:

Remark 1.6.11. Let V be a finite-dimensional complex vector space and let $x : V \to V$ be a linear map. Theorem 6.1 in [11] states that we can always find a basis of V in which x is represented by an upper triangular matrix. This implies that a nilpotent map may be represented by a strictly upper triangular matrix and so has trace zero.

Theorem 1.6.12 (Cartan's Second Criterion)**.** *A Lie algebra L is semisimple if and only if its Killing form is nondegenerate.*

Proof. ([11] Theorem 9.9) If L is not semisimple, then it has a non-zero abelian ideal A. We must show that A is in the kernel of the Killing form. If $x \in L$, $a \in A$ we have that $\mathsf{ad}\, x$ is a linear map that preserves A, and $\mathsf{ad}\, a$ is a linear map which maps L into A and maps A to 0. Hence, the composite map $\mathsf{ad}\, x \circ \mathsf{ad}\, a$ maps L into A and is 0 on A, so $(\mathsf{ad}\, x\, \mathsf{ad}\, a)^2 = 0$. We know from Remark 1.6.11 that nilpotent maps have trace 0 and so $\kappa(x,a) = 0$, that is A is in the kernel of the Killing form. Conversely, let A be the kernel of the Killing form. A is an ideal because

$$\kappa([x,a],y) = -\kappa([a,x],y) = -\kappa(a,[x,y]) = 0, \quad a \in A, x,y \in L.$$

Next we consider the elements of $\mathsf{ad}\,A$. They form a Lie subalgebra with $\kappa(a,b) = 0$ for all $a, b \in \mathsf{ad}\,A$. Therefore by Cartan's first criterion, Theorem 1.6.10, $\mathsf{ad}\,A$ is solvable. The kernel of the adjoint representation is the centre of L (see Remark 1.4.3) so if $\mathsf{ad}\,A$ is solvable , then so is A. We have found a nonzero solvable ideal. □

We will now use Theorem 1.6.12 to show that the only derivations of a complex semisimple Lie algebra are those of the form $\mathsf{ad}\,x$ for $x \in L$. More precisely, we have

Proposition 1.6.13. *If L is a finite-dimensional complex semisimple Lie algebra, then* $\mathsf{ad}\,L = \mathsf{Der}\,L$.

Proof. ([11] Proposition 9.13) We showed in Example 1.2.12 that for each $x \in L$ the linear map $\mathsf{ad}\,x$ is a derivation of L, so ad is a Lie algebra homomorphism from L to $\mathsf{Der}\,L$. Moreover, if δ is a derivation of L and $x, y \in L$, then

$$\begin{aligned}[\delta, \mathsf{ad}\,x]y &= (\delta \circ \mathsf{ad}\,x)(y) - (\mathsf{ad}\,x \circ \delta)(y)\\ &= \delta[x,y] - \mathsf{ad}\,x(\delta y)\\ &= [\delta x, y] + [x, \delta y] - [x, \delta y]\\ &= \mathsf{ad}(\delta x)y.\end{aligned}$$

Thus the image of $\mathsf{ad} : L \to \mathsf{Der}\,L$ is an ideal of $\mathsf{Der}\,L$. This much is true for any Lie algebra.

Now we bring in our assumption that L is complex and semisimple. First, note that $\mathsf{ad} : L \to \mathsf{Der}\,L$ is one-to-one, as $\ker \mathsf{ad} = Z(L) = 0$, so the Lie algebra $M := \mathsf{ad}\,L$ is isomorphic to L and therefore it is semisimple as well. To show that $M = \mathsf{Der}\,L$, we use the Killing form on the Lie algebra $\mathsf{Der}\,L$. If M is properly contained in $\mathsf{Der}\,L$ then $M^\perp \neq 0$, so it is sufficient to prove that $M^\perp = 0$. As M is an ideal of $\mathsf{Der}\,L$, the Killing form κ_M of M is the restriction of the Killing form on $\mathsf{Der}\,L$ by Lemma 1.6.3. As M is semisimple, by Cartan's Second Criterion, κ_M is nondegenerate, so $M^\perp \cap M = 0$ and hence $[M^\perp, M] = 0$. But we saw above that

$$[\delta, \mathsf{ad}\,x] = \mathsf{ad}(\delta x) \tag{1.4}$$

so, for all $x \in L$, we have $\delta(x) = 0$ for $\delta \in M^\perp$. In other words, $\delta = 0$. □

The following Lemma on derivations will prove useful in later chapters:

Lemma 1.6.14. *Let A be a linear operator in L. Let B be a bilinear form $B : L \otimes L \to \mathbb{C}$ such that $B(x, y) = \kappa(Ax, y)$. Then A is a derivation and $A^* + A = 0$ if and only if B is skew-symmetric and satisfies the following equation:*

$$B([x, y], z) + B([y, z], x) + B([z, x], y) = 0. \tag{1.5}$$

Proof. First we prove that B is skew-symmetric if and only if $A^* + A = 0$. B is skew-symmetric if and only if $B(x, y) + B(y, x) = 0$ we find

$$\begin{aligned} B(x, y) + B(y, x) &= \kappa(Ax, y) + \kappa(Ay, x) \\ &= \kappa(Ax, y) + \kappa(y, A^*x) \\ &= \kappa((A + A^*)x, y). \end{aligned}$$

This is equal to zero if and only if $A + A^* = 0$. We know from Definition 1.2.10 that A is a derivation if and only if

$$A[x, y] - [Ax, y] - [x, Ay] = 0 \quad \text{for all } x, y \in L.$$

Because κ is nondegenerate, this is true if and only if

$$\kappa(A[x, y], z) - \kappa(Ax, [y, z]) - \kappa([x, Ay], z) = 0 \quad \text{for all } x, y, z \in L$$

because κ is associative, this is true if and only if

$$\kappa(A[x, y], z) - \kappa(Ax, [y, z]) + \kappa(Ay, [x, z]) = 0.$$

Using the definition of B we find that the above is equivalent to

$$B([x, y], z) - B(x, [y, z]) + B(y, [x, z]) = 0$$

and because B is skew-symmetric, this is equal to Equation (1.5). □

Remark 1.6.15. Let β be a symmetric nondegenerate bilinear form on a vector space V, and let $\{v_1, \dots, v_n\}$ be a basis for V. Then the *dual basis* of V relative to β is defined to be $\{w_1, \dots, w_n\}$ satisfying $\beta(v_i, w_j) = \delta_{ij}$.

Proposition 1.6.16. *Let L be a Lie algebra over $\mathbb{C}$, and let β be a symmetric, associative, bilinear form on L. Then β induces a linear map*

$$\varphi : L \to L^*, \quad \varphi(x) = \beta(x, -).$$

Proof. We must show that $\varphi(x+y) = \varphi(x)+\varphi(y)$ and that $\varphi(\lambda x) = \lambda\varphi(x)$ for all $x, y \in L$, $\lambda \in \mathbb{C}$. From the bilinearity of β we find,

$$\varphi(x+y) = \beta(x+y,-) = \beta(x,-)+\beta(y,-) = \varphi(x)+\varphi(y)$$

also, from bilinearity

$$\varphi(\lambda x) = \beta(\lambda x,-) = \lambda\beta(x,-) = \lambda\varphi(x).$$

Thus, φ is a linear map. □

Lemma 1.6.17. *Let L be a Lie algebra over $\mathbb{C}$ and let β be defined as in Proposition 1.6.16. If β is nondegenerate, then the map φ is an isomorphism of vector spaces between L and L^* and is also an isomorphism of L-modules*

Proof. Is is clear that φ is an isomorphism between the vector spaces of L and L^* because β is nondegenerate. The dual space L^* is an L-module by defining $(x\cdot\xi)(v) = -\xi(x\cdot v)$ for $\xi \in L^*$, $x, v \in L$. Therefore, we need to show that

$$\varphi(x\cdot v)(y) = x\cdot\varphi(v)(y) = -\varphi(v)(x\cdot y).$$

We may write this as $\varphi([x,v])(y) = \beta([x,v],y) = -\beta([v,x],y)$. From the associativity of β this is equal to $-\beta(v,[x,y]) = -\varphi(v)([x,y])$ as required. □

Proposition 1.6.18. *Let L be a simple Lie algebra over $\mathbb{C}$ with Killing form κ. A symmetric, associative, nondegenerate, bilinear form on L is uniquely determined up to scalar. In particular, it is a scalar multiple of κ.*

Proof. Let β be any other symmetric associative nondegenerate bilinear form on L. From Lemma 1.6.17 we know that β and κ induce isomorphisms of L-modules $\varphi_\beta : L \to L^*$ and $\varphi_\kappa : L \to L^*$. By composing these isomorphisms as follows:

$$\varphi_\beta^{-1}\circ\varphi_\kappa : L \to L$$

we obtain an isomorphism from L to L. Because L is simple, L viewed as an L-module via the adjoint representation is simple (that is, irreducible). Therefore, by Schur's Lemma, Lemma 1.4.9, $\varphi : L \to L$ is an L-module homomorphism if and only if φ is a scalar multiple of the identity, that is, $\varphi = \lambda\mathbb{1}_L$ for some $\lambda \in \mathbb{C}$. Therefore, for some $\lambda \in \mathbb{C}$ we have

$$\varphi_\beta^{-1}\circ\varphi_\kappa = \lambda\mathbb{1}_L = \lambda\varphi_\beta^{-1}\circ\varphi_\beta.$$

We can apply φ_β to each side to obtain $\varphi_\kappa = \lambda\varphi_\beta$ Using the definitions of the maps φ_β and φ_κ we find that

$$\kappa(x,y) = \varphi_\kappa(x)(y) = \lambda\varphi_\beta(x)(y) = \lambda\beta(x,y).$$

□

1.7 Roots of a Semisimple Lie algebra

In this section we carry out an investigation into the roots of a semisimple Lie algebra L. The roots of a semisimple Lie algebra are the Lie algebra weights $\alpha \in H^*$ occurring in its adjoint map. The set of roots form the root system and are completely determined by L. Here we focus on the main properties of the roots and among other things we prove that if $\alpha \in \Phi$ is a root then $-\alpha$ is also a root.

Lemma 1.7.1. *Suppose that $\alpha, \beta \in H^*$. Then*

(i) $[L_\alpha, L_\beta] \subseteq L_{\alpha+\beta}$,

(ii) *If $\alpha+\beta \neq 0$, then $\kappa(L_\alpha, L_\beta) = 0$, that is, L_α is orthogonal to L_β relative to the Killing form of L,*

(iii) *The restriction of κ to H is nondegenerate.*

Proof. ([11] Lemma 10.1)

(i) Take $x \in L_\alpha$ and $y \in L_\beta$. We must show that $[x,y]$, if nonzero, is an eigenvector for each $\operatorname{ad} h \in H$, with eigenvalue $\alpha(h)+\beta(h)$. Using the Jacobi identity we get

$$\begin{aligned}[h,[x,y]] = [[h,x],y] + [x,[h,y]] &= [\alpha(h)x,y] + [x,\beta(h)y] \\ &= \alpha(h)[x,y] + \beta(h)[x,y] \\ &= (\alpha+\beta)(h)[x,y].\end{aligned}$$

(ii) Let $h \in H$ be such that $(\alpha+\beta)(h) \neq 0$. Then for $x \in L_\alpha$ and $y \in L_\beta$, we have, using the associativity of the Killing form

$$\begin{aligned}\alpha(h)\kappa(x,y) = \kappa([h,x],y) &= -\kappa([x,h],y) \\ &= -\kappa(x,[h,y]) = -\beta(h)\kappa(x,y)\end{aligned}$$

and hence,

$$\begin{aligned} \alpha(h)\kappa(x,y) + \beta(h)\kappa(x,y) &= 0 \\ (\alpha+\beta)(h)\kappa(x,y) &= 0. \end{aligned}$$

Since by assumption $(\alpha+\beta)(h) \neq 0$, we must have $\kappa(x,y) = 0$.

(iii) Suppose that $z \in H$ and $\kappa(z,x) = 0$ for all $x \in H$. By (ii), we know that H is perpendicular to L_α for all $\alpha \neq 0$. If $x \in L$, then by the RSD, Equation (1.2), we can write x as

$$x = x_0 + \sum_{\alpha \in \phi} x_\alpha$$

with $x_\alpha \in L_\alpha$. By linearity, $\kappa(z,x) = 0$ for all $x \in L$. Since κ is nondegenerate, it follows that $z = 0$, as required.

□

Remark 1.7.2. The Killing form enables us to make a connection between the Cartan subalgebra H and its dual H^*. Given $h \in H$, let θ_h denote the map $\theta_h \in H^*$ defined by

$$\theta_h(k) = \kappa(h,k) \text{ for all } k \in H.$$

Recall from Lemma 1.6.17 that this map is an isomorphism. In particular, to each $\alpha \in H^*$ we associate its *root vector* $t_\alpha \in H$ defined as an element of H such that

$$\alpha(h) = \kappa(t_\alpha, h) \text{ for any } h \in H. \tag{1.6}$$

The root vector t_α exists and is unique by the nondegeneracy of the restriction of the Killing form to H (Lemma 1.7.1). This unique connection between H and H^* occurs not for all Lie algebras but only for the class of semisimple Lie algebras. (Recall that for semisimple Lie algebras the Killing form is nondegenerate. This means, in particular, that if $\kappa(a,b) = 0$ for every $b \in H$, then $a = 0$).

Example 1.7.3. *This relationship between H and H^* can be illustrated using $\mathsf{sl}(3)$. Referring to equations in Example 1.5.6, we designate three non-zero roots by*

$$\begin{aligned} \alpha_1(ah_1 + bh_2) &= 2a - b \\ \alpha_2(ah_1 + bh_2) &= -a + 2b \\ \alpha_3(ah_1 + bh_2) &= a + b. \end{aligned} \tag{1.7}$$

The other non-zero roots are the negative of these. Now we determine the elements in H corresponding to α_1, α_2 and α_3. Each of these h's is to lie in H and is thus of the form

$$h_{\alpha_i} = c_i h_1 + d_i h_2$$

Using the previously computed values of the Killing form, we see that

$$\begin{aligned}\kappa(h_{\alpha_i}, h_1) &= 12c_i - 6d_i\\ \kappa(h_{\alpha_i}, h_2) &= -6c_i + 12d_i.\end{aligned}$$

To determine the coefficients c_i and d_i, we combine the definition of t_{α_1} in Equation (1.6) with the expression for the roots in Equations (1.7)

$$\begin{aligned}\alpha_1(h_1) &= 2 = \kappa(t_{\alpha_1}, h_1) = 12c_1 - 6d_1\\ \alpha_1(h_2) &= -1 = \kappa(t_{\alpha_1}, h_2) = -6c_1 + 12d_1\\ \alpha_2(h_1) &= -1 = \kappa(t_{\alpha_2}, h_1) = 12c_2 - 6d_2\\ \alpha_2(h_2) &= 2 = \kappa(t_{\alpha_2}, h_2) = -6c_2 + 12d_2\\ \alpha_3(h_1) &= 1 = \kappa(t_{\alpha_3}, h_1) = 12c_3 - 6d_3\\ \alpha_3(h_2) &= 1 = \kappa(t_{\alpha_3}, h_2) = -6c_3 + 12d_3.\end{aligned}$$

Thus we find the elements of H which correspond to the various roots:

$$t_{\alpha_1} = \frac{1}{6}h_1; \quad t_{\alpha_2} = \frac{1}{6}h_2; \quad t_{\alpha_3} = \frac{1}{6}(h_1 + h_2).$$

Proposition 1.7.4. *Let $x, y : V \to V$ be linear maps from a complex vector space V to itself. Suppose that x and y both commute with $[x, y]$. Then $[x, y]$ is a nilpotent map.*

Proof. See [11] Proposition 5.7. □

Proposition 1.7.5. *Let $\alpha \in \Phi$. The root spaces $L_{\pm\alpha}$ are 1-dimensional. Moreover, the only multiples of α which lie in Φ are $\pm\alpha$.*

Proof. See [11] Proposition 10.9. □

Lemma 1.7.6. *Suppose that $\alpha \in \Phi$ and that x is a nonzero element in L_α. Then $-\alpha$ is a root and there exists $y \in L_{-\alpha}$ such that $\mathsf{span}\{x, y, [x, y]\}$ is a Lie subalgebra of L isomorphic to $\mathsf{sl}(2, \mathbb{C})$.*

Proof. ([11] Lemma 10.5) First we claim that there is some $y \in L_{-\alpha}$ such that $\kappa(x,y) \neq 0$ and $[x,y] \neq 0$. Since κ is nondegenerate, there is some $w \in L$ such that $\kappa(x,w) \neq 0$. Write $w = y_0 + \sum_{\beta\in\Phi} y_\beta$ with $y_0 \in L_0$ and $y_\beta \in L_\beta$. When we expand $\kappa(x,w)$, we find by Lemma 1.7.1 part (ii) the only way a non-zero term can occur is if $-\alpha$ is a root and $y_{-\alpha} \neq 0$, so we may take $y = y_{-\alpha}$. Now α is nonzero so there is some $t \in H$ such that $\alpha(t) \neq 0$. For this t, we have

$$\kappa(t,[x,y]) = \kappa([t,x],y) = \alpha(t)\kappa(x,y) \neq 0$$

and so $[x,y] \neq 0$. Let $S := \mathsf{span}\{x, y, [x,y]\}$. By Lemma 1.7.1 (i), $[x,y]$ lies in $L_0 = H$. As x and y are simultaneous eigenvectors for all elements of $\mathsf{ad}\, H$, and so in particular for $\mathsf{ad}[x,y]$ this shows that S is a Lie subalgebra of L. It remains to show that S is isomorphic to $\mathsf{sl}(2,\mathbb{C})$.

Let $h := [x,y] \in S$. We claim that $\alpha(h) \neq 0$. If not, then $[h,x] = \alpha(h)x = 0$; similarly, $[h,y] = -\alpha(h)y = 0$, so $\mathsf{ad}\, h : L \to L$ commutes with $\mathsf{ad}\, x : L \to L$ and $\mathsf{ad}\, y : L \to L$. From Proposition 1.7.4 we know that this means that $\mathsf{ad}\, h : L \to L$ is a nilpotent map. However, as H is a Cartan subalgebra, h is semisimple. The only element of L that is both semisimple and nilpotent is 0, so $h = 0$, a contradiction.

Thus S is a 3-dimensional complex Lie algebra with $S' = S$. From Remark 1.3.2, S is isomorphic to $\mathsf{sl}(2,\mathbb{C})$. □

Remark 1.7.7. We can now associate to each root $\alpha \in \Phi$ a Lie subalgebra of L isomorphic to $\mathsf{sl}(2,\mathbb{C})$ so that we can apply our results from representations of $\mathsf{sl}(2,\mathbb{C})$ to deduce several strong results on the structure of L. A standard basis for this Lie algebra is $\{e_\alpha, f_\alpha, h_\alpha\}$ such that:

1. $e_\alpha \in L_\alpha$, $f_\alpha \in L_{-\alpha}$, $h_\alpha \in H$ and $\alpha(h_\alpha) = 2$.

2. The map $\theta : \mathsf{sl}(\alpha) \to \mathsf{sl}(2,\mathbb{C})$ defined by $\theta(e_\alpha) = e$, $\theta(f_\alpha) = f$, $\theta(h_\alpha) = h$ is a Lie algebra isomorphism.

Lemma 1.7.8. *Let $\alpha \in \Phi$. If $x \in L_\alpha$ and $y \in L_{-\alpha}$, then $[x,y] = \kappa(x,y)t_\alpha$. In particular, $h_\alpha = [e_\alpha, f_\alpha] \in \mathsf{span}\{t_\alpha\}$.*

Proof. ([11] Lemma 10.6) For $h \in H$, we have

$$\kappa(h,[x,y]) = \kappa([h,x],y) = \alpha(h)\kappa(x,y) = \kappa(t_\alpha,h)\kappa(x,y).$$

We can look at $\kappa(x,y)$ as a scalar and rewrite the last equality of the above equation as

$$\kappa(h,[x,y]) = \kappa(h,\kappa(x,y)t_\alpha).$$

This shows that $[x, y] - \kappa(x, y)t_\alpha$ is perpendicular to all $h \in H$, and hence it is zero as κ restricted to H is nondegenerate. $\square$

Example 1.7.9. *Consider* $\mathsf{sl}(n, \mathbb{C})$. *Recall that* $(\mathsf{ad}\, h)e_{ij} = (\varepsilon_i - \varepsilon_j)(h)e_{ij}$. *We let* $\alpha_{ij} = \varepsilon_i - \varepsilon_j$ *so*

$$(\mathsf{ad}\, h)e_{ij} = \alpha_{ij}(h)e_{ij}$$

and the root space decomposition is

$$\mathsf{sl}(n, \mathbb{C}) = H \oplus \bigoplus_{i \neq j} \mathbb{C} \cdot e_{ij}$$

using $e_{ij}e_{kl} = \delta_{jk}e_{il}$ *one verifies that* $[e_{ij}, e_{ji}] = e_{ii} - e_{jj}$ *and that*

$$e_{ij} \mapsto x, \ e_{ji} \mapsto y, \ e_{ii} - e_{jj} \mapsto h$$

defines an isomorphism of $S_{ij} := S_{\alpha_{ij}}$ *with* $\mathsf{sl}(2, \mathbb{C})$.

Chapter 2

Cartan Matrices and Dynkin Diagrams

In this chapter, we continue our investigation into the roots of a semisimple Lie algebra. We use the Killing form to define an inner product on the Euclidean space E. We then can define a root system $R \subset E$ and a base for a root system $B \subset R$. This leads us into Cartan matrices and Dynkin diagrams, of which we give explicitly for the classical Lie algebras. Finally, we look at the universal enveloping algebra and the Casimir operator for Lie algebras. We note that this material is not new, and we have mostly followed the book [11].

2.1 Cartan Subalgebras as Inner Product Spaces

In this section we show that the roots of L all lie in a real vector subspace of H^* and that the Killing form induces an inner product on the space. Proposition 1.7.5 shows that if $\alpha \in \phi$ then the only multiples of $\alpha \in \phi$ are $\pm\alpha$. On the other hand, there must be roots, as otherwise the root space decomposition Equation (1.2) would imply that $L = H$ was abelian.

Lemma 2.1.1. *For each $\alpha \in \Phi$ we have*

1. $t_\alpha = \frac{h_\alpha}{\kappa(e_\alpha, f_\alpha)}$,
2. $h_\alpha = \frac{2t_\alpha}{\kappa(t_\alpha, t_\alpha)}$,
3. $\kappa(t_\alpha, t_\alpha)\kappa(h_\alpha, h_\alpha) = 4$.

Proof. For (1) we use Lemma 1.7.8 applied with $x = e_\alpha$, $y = f_\alpha$, then

$$[e_\alpha, f_\alpha] = h_\alpha = \kappa(e_\alpha, f_\alpha)t_\alpha.$$

From Remark 1.7.7 we can show (2): we know that $\alpha(h_\alpha) = 2$ and from (1) $h_\alpha = \kappa(e_\alpha, f_\alpha)t_\alpha$ and so we have

$$2 = \kappa(t_\alpha, h_\alpha) = \kappa(t_\alpha, \kappa(e_\alpha, f_\alpha)t_\alpha)$$

which implies that $\kappa(e_\alpha, f_\alpha)\kappa(t_\alpha, t_\alpha) = 2$. But from (1) $\kappa(e_\alpha, f_\alpha) = \frac{h_\alpha}{t_\alpha}$ so we have

$$h_\alpha = \frac{2t_\alpha}{\kappa(t_\alpha, t_\alpha)}.$$

Finally we prove (3) directly,

$$\kappa(h_\alpha, h_\alpha) = \kappa\left(\frac{2t_\alpha}{\kappa(t_\alpha, t_\alpha)}, \frac{2t_\alpha}{\kappa(t_\alpha, t_\alpha)}\right) = \frac{4\kappa(t_\alpha, t_\alpha)}{\kappa(t_\alpha, t_\alpha)\kappa(t_\alpha, t_\alpha)} = \frac{4}{\kappa(t_\alpha, t_\alpha)}.$$

□

Definition 2.1.2. Denote by E the *real* subspace of H^* spanned by the roots $\alpha_1, \dots, \alpha_l$ and containing all the roots of ϕ. E does not depend on our particular choice of basis.

Recall again from Lemma 1.6.17 that the Killing form defines an isomorphism $H \cong H^*$ sending x to the linear form $\kappa(x, -)$ and under this isomorphism $t_\alpha \in H$ corresponds to $\alpha \in H^*$. Let us write $(-, -)$ for the positive definite symmetric bilinear form on H^* which corresponds to the Killing form under this isomorphism. Then we have

$$(\alpha, \beta) = \kappa(t_\alpha, t_\beta) \in \mathbb{Q}.$$

Hence, for $\beta \in \Phi$ we have

$$\begin{aligned}\beta(h_\alpha) &= \kappa(t_\beta, h_\alpha) \\ &= \frac{2\kappa(h_\beta, t_\alpha)}{\kappa(e_\beta, f_\beta)\kappa(t_\alpha, t_\alpha)} \\ &= \frac{2\kappa(\frac{2t_\beta}{\kappa(t_\beta,t_\beta)}, t_\alpha)\kappa(t_\beta, t_\beta)}{\kappa(t_\alpha, t_\alpha)} \\ &= \frac{2\kappa(t_\beta, t_\alpha)}{\kappa(t_\alpha, t_\alpha)} \\ &= \frac{2(\beta, \alpha)}{(\alpha, \alpha)}.\end{aligned} \tag{2.1}$$

This form $(-,-)$ is a real-valued inner product on E. We will write

$$\langle \beta, \alpha \rangle := \frac{2(\beta,\alpha)}{(\alpha,\alpha)}, \tag{2.2}$$

note that this is only linear in respect to its first variable, β.

The essential properties of the roots of complex semisimple Lie algebras may be captured in the idea of an abstract "root system". These root systems can be used to classify the complex semisimple Lie algebras.

Definition 2.1.3. A subset R of a real inner-product space E is a *root system* if it satisfies the following axioms

(R1) R is finite, it spans E, and it does not contain zero.

(R2) If $\alpha \in R$, then the only scalar multiples of α in R are $\pm\alpha$.

(R3) If $\alpha \in R$, then the reflection s_α in the hyperplane orthogonal to α sends R into itself.

(R4) If $\alpha, \beta \in R$, then $\langle \beta, \alpha \rangle \in \mathbb{Z}$

The elements of R are called *roots*.

In [11] Section 11.2 it was shown that for any two nonzero vectors α and β in E, the permissable angles between them are quite restrictive. This, in turn, limits the possibilities for simple Lie algebras. The importance of the concept of a root system is due to the fact that the isomorphism classes of complex semisimple Lie algebras bijectively correspond to equivalence classes of root systems. Moreover, for a given complex semisimple Lie algebra the various choices of a Cartan subalgebra give rise to equivalent root systems.

Definition 2.1.4. A root system R is *irreducible* if R cannot be partitioned into two disjoint subsets $R = R_1 \cup R_2$ such that every element of R_1 is orthogonal to every element of R_2.

Definition 2.1.5. Let R be a root system in the real inner-product space E. A subset B of R is a *base* for the root system R if

(B1) B is a vector space basis for E, and

(B2) every $\beta \in R$ can be written as $\beta = \sum_{\alpha \in B} k_\alpha \alpha$ with $k_\alpha \in \mathbb{Z}$, where all the nonzero coefficients k_α have the same sign.

We also define *positive roots* to be those roots where all the coefficients k_α are positive, and *negative roots* to be those roots where all the coefficients k_α are negative.

Theorem 2.1.6. *Every root system has a base.*

Proof. See [11] Theorem 11.10 □

Let R^+ denote the set of all positive roots in a root system R with respect to a base B, and let R^- be the set of all negative roots. Then $R = R^+ \cup R^-$, a disjoint union. The set B is contained in R^+; the elements of B are called *simple roots.* A simple root is a positive root which cannot be written as the sum of two positive roots.

Example 2.1.7. *Consider* $\mathsf{sl}(3)$. *The roots are*

$$\begin{aligned}\alpha_1(ah_1 + bh_2) &= 2a - b\\ \alpha_2(ah_1 + bh_2) &= -a + 2b\\ \alpha_3(ah_1 + bh_2) &= a + b\end{aligned}$$

and the negative of these roots.
Suppose we select as a basis for H^* the roots α_1 and α_3, in that order. Now since $\alpha_2 = \alpha_3 - \alpha_1$, α_2 is negative. The positive roots are α_1, $-\alpha_2$, and α_3. Now $\alpha_1 = \alpha_3 + (-\alpha_2)$ so α_1 is the sum of two positive roots, therefore is not simple. The simple roots are $-\alpha_2$ and α_3, and $\alpha_2 > \alpha_3$. This depends on the original ordering of the basis.

2.2 Cartan Matrices and Dynkin Diagrams

In this section we describe a generalised Cartan matrix. The Cartan matrix of a simple Lie algebra is constructed using the simple roots of the Lie algebra. Conversely, given a Cartan matrix, one can recover its corresponding Lie algebra. We also describe a certain graph, called a Dynkin diagram, associated to each Lie algebra which gives the same information as the Cartan matrix.

Definition 2.2.1. Let B be a base in a root system R. Fix an order on the elements of B, say $(\alpha_1, \dots, \alpha_l)$. The matrix

$$(c_{ij}) = \langle \alpha_i, \alpha_j \rangle = \left(\frac{2(\alpha_i, \alpha_j)}{(\alpha_j, \alpha_j)} \right)_{1 \le i,j \le n} = \begin{pmatrix} 2\frac{(\alpha_1,\alpha_1)}{|\alpha_1|^2} & 2\frac{(\alpha_1,\alpha_2)}{|\alpha_2|^2} & 2\frac{(\alpha_1,\alpha_3)}{|\alpha_3|^2} & \cdots \\ 2\frac{(\alpha_2,\alpha_1)}{|\alpha_1|^2} & 2\frac{(\alpha_2,\alpha_2)}{|\alpha_2|^2} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

is called the *Cartan matrix* of R.

We form the Cartan matrix from the simple roots. The Cartan matrix summarizes all the properties of the simple Lie algebra to which it corresponds and depends only on the ordering adopted with our chosen base B and not on the base itself. The dimension of the Cartan subalgebra H is the same as that of H^*, the root space. This dimension is called the *rank* of the algebra.

The diagonal entries of the Cartan matrix are always 2. Since the scalar product of two different simple roots is non-positive, the off-diagonal elements can only be $0, -1, -2$, or -3, since the angles between the simple roots are right or obtuse (see [11] Section 11.2).

Example 2.2.2. *For rank* $R = 2$, $C = \begin{pmatrix} 2 & -a \\ -b & 2 \end{pmatrix}$ *where either* $a = b = 0$ *or both* a *and* b *are non-negative integers.*
Since $\det C = 4 - ab > 0$*, the Cartan matrices for root systems of rank 2 are:*

$$A_1 \times A_1 : \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}; \; A_2 : \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}; \; B_2 : \begin{pmatrix} 2 & -2 \\ -1 & 2 \end{pmatrix}; \; G_2 : \begin{pmatrix} 2 & -1 \\ -3 & 2 \end{pmatrix}.$$

Example 2.2.3. *Consider again* $\mathsf{sl}(3)$*. For simplicity take the positive basis to be* α_1 *and* α_2*. Then since* $\alpha_3 = \alpha_1 + \alpha_2$ *the simple roots are also* α_1 *and* α_2*. The relevant scalar products are*

$$(\alpha_1, \alpha_1) = \frac{1}{3}$$
$$(\alpha_1, \alpha_2) = -\frac{1}{6}$$
$$(\alpha_2, \alpha_2) = \frac{1}{3}.$$

From this we compute the Cartan matrix

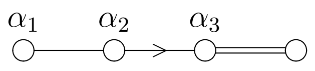
$$\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

Any root system gives rise to a Cartan matrix. So even before studying each of the simple algebras in detail, we know in advance that they exist, provided that we know that the corresponding root systems exist. We can also describe the Cartan matrix of a given root system R by a certain graph.

Definition 2.2.4. The *Dynkin diagram* of a semisimple Lie algebra is a graph whose vertices correspond to the simple roots α. The number of edges joining the vertices α_i and α_j is $c_{ij}c_{ji}$. In addition, if α_i and α_j have different lengths, these edges are marked with an arrow directed towards the shorter root.

Remark 2.2.5. Simple Lie algebras correspond to connected Dynkin diagrams and any finite-dimensional simple Lie algebra yields one of a very restricted set of Dynkin diagrams (and hence Cartan matrices).

Example 2.2.6. *Consider the Dynkin diagram:*



The way to read this diagram is as follows: each node in the diagram stands for a simple root $\alpha_1, \alpha_2, \alpha_3$. It is clear how to discover the Cartan integers c_{ij} from the Dynkin diagram. The Cartan matrix of the above diagram is determined by noting that $c_{13} = c_{31} = 0$ since the first and third dots are not connected. Since one line connects the first and second points, we must have $c_{12} = c_{21} = -1$. The second and third points are connected by two lines so $c_{23}c_{32} = 2$, Since the third root is smaller that the second, it must be that $c_{23} = -2$ while $c_{32} = -1$. Thus we have

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -2 \\ 0 & -1 & 2 \end{pmatrix}.$$

The definition of a semisimple Lie algebra does not seem very restrictive, so the fact that the complex semisimple Lie algebras are determined, up to isomorphism, by their Dynkin diagrams is quite remarkable. However, complex semisimple Lie algebras with different Dynkin diagrams are not isomorphic.

Proposition 2.2.7. *Let L be a complex semisimple Lie algebra with Cartan subalgebra H and root system ϕ. If ϕ is irreducible, then L is simple.*

Proof. ([11] Proposition 12.4) By the root space decomposition, we may write L as

$$L = H \oplus \bigoplus_{\alpha \in \phi} L_\alpha.$$

Suppose that L has a proper non-zero ideal I. Since H consists of semisimple elements, it acts diagonalisably on I, and so I has a basis of common eigenvectors for the elements of $\mathsf{ad}\, H$. From Proposition 1.7.5 we know that each root space L_α is 1-dimensional, this implies that

$$I = H_1 \oplus \bigoplus_{\alpha \in \phi_1} L_\alpha$$

for some subspace H_1 of $H = L_0$ and some subset ϕ_1 of ϕ. Similarly, we have

$$I^\perp = H_2 \oplus \bigoplus_{\alpha \in \phi_2} L_\alpha,$$

where $I^\perp$ is the perpendicular space to I with respect to the Killing form. As $I \oplus I^\perp = L$, we must have $H_1 \oplus H_2 = H$, $\phi_1 \cap \phi_2 = \{0\}$, and $\phi_1 \cup \phi_2 = \phi$. If ϕ_2 is empty, then $L_\alpha \subseteq I$ for all $\alpha \in \phi$. As L is generated by its root spaces, this implies that $I = L$, a contradiction. Similarly, ϕ_1 is not empty. Now, given $\alpha \in \phi_1$ and $\beta \in \phi_2$, we have

$$\langle \alpha, \beta \rangle = \alpha(h_\beta) = 0$$

as $\alpha(h_\beta)e_\alpha = [h_\beta, e_\alpha] \in I^\perp \cap I = 0$, so $(\alpha, \beta) = 0$ for all $\alpha \in \phi_1$ and $\beta \in \phi_2$, which shows that ϕ is reducible. □

The converse is also true. See [11] Proposition 14.2 for a proof.

2.3 The Classical Lie Algebras

We shall now see that with five exceptions, every finite-dimensional simple Lie algebra over $\mathbb{C}$ is isomorphic to one of the *classical Lie algebras:*

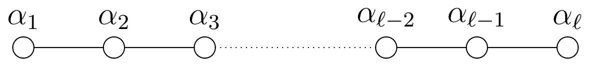
$$\mathsf{sl}(n, \mathbb{C}), \quad \mathsf{so}(n, \mathbb{C}), \quad \mathsf{sp}(2n, \mathbb{C}).$$

The five exceptional Lie algebras are known as E_6, E_7, E_8, F_4 and G_2. For an explicit construction of the Cartan subalgebra, root vectors and roots for the classical Lie algebras see [11] Chapter 12. Here we will simply list them and give the Cartan matrices and Dynkin diagrams:

We have already introduced the family of special linear Lie algebras, $\mathsf{sl}(\ell+1,\mathbb{C})$ consisting of all traceless $n \times n$ matrices. We say that the root system of $\mathsf{sl}(\ell+1,\mathbb{C})$ has *type* A_ℓ. The Cartan matrix is

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & 2 & -1 & \dots & \dots & \dots & 0 \\ 0 & -1 & 2 & \dots & \dots & \dots & 0 \\ \vdots & \ddots & & \ddots & & & \vdots \\ \vdots & \ddots & & \ddots & & & -1 \\ 0 & \dots & & \dots & -1 & 2 & -1 \\ 0 & \dots & & \dots & 0 & -1 & 2 \end{pmatrix}$$

and the Dynkin diagram is:



This diagram is connected, so L is simple.
The remaining algebras can be defined as certain subalgebras of $\mathsf{gl}(n,\mathbb{C})$. If $S \in \mathsf{gl}(n,\mathbb{C})$, then we can define a Lie subalgebra of $\mathsf{gl}(n,\mathbb{C})$ by

$$\mathsf{gl}_S(n,F) = \{x \in \mathsf{gl}(n,F) \mid x^t S = -Sx\}.$$

Assume first of all that $n = 2\ell$. Take S to be the matrix with $\ell \times \ell$ blocks:

$$S = \begin{pmatrix} 0 & I_\ell \\ I_\ell & 0 \end{pmatrix}.$$

We define $\mathsf{so}(2\ell,\mathbb{C}) = \mathsf{gl}_S(2\ell,\mathbb{C})$. When $n = 2\ell+1$, we take

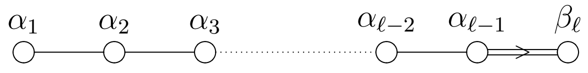
$$S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & I_\ell \\ 0 & I_\ell & 0 \end{pmatrix}$$

and define $\mathsf{so}(2\ell+1,\mathbb{C}) = \mathsf{gl}_S(2\ell+1,\mathbb{C})$. These Lie algebras are known as the *orthogonal Lie algebras* on even and odd dimensional spaces (the structures for the even and odd cases are different).

For $\mathsf{so}(2\ell+1,\mathbb{C})$ the Cartan matrix is:

$$C = \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & 2 & -1 & 0 \\ \vdots & \ddots & \ddots & -1 & 2 & -2 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix}$$

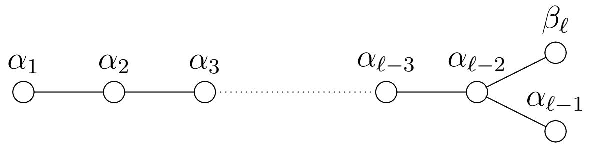
and so we get the Dynkin diagram:



As the Dynkin diagram is connected, ϕ is irreducible and so L is simple. The roots system of $\mathsf{so}(2\ell+1,\mathbb{C})$ is said to have *type* B_ℓ. For the even case $\mathsf{so}(2\ell,\mathbb{C})$ we have the following Cartan matrix:

$$C = \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & 2 & -1 & -1 \\ \vdots & \ddots & \ddots & -1 & 2 & 0 \\ 0 & \dots & \dots & -1 & 0 & 2 \end{pmatrix}$$

and the corresponding Dynkin diagram is:



As this diagram is connected, the Lie algebra is simple. When $\ell = 3$, the Dynkin diagram is the same as that of A_3, the root system for $\mathsf{sl}(3,\mathbb{C})$, so we might expect that $\mathsf{so}(6,\mathbb{C})$ should be isomorphic to $\mathsf{sl}(4,\mathbb{C})$. This is indeed the case. For $\ell \geq 4$ the root system of $\mathsf{so}(2\ell,\mathbb{C})$ is said to have *type* D_ℓ.

The Lie algebras $\mathsf{sp}(n,\mathbb{C})$ are only defined for even n. If $n = 2\ell$, we take

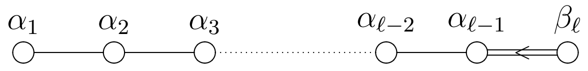
$$S = \begin{pmatrix} 0 & I_\ell \\ -I_\ell & 0 \end{pmatrix}$$

and define $\mathsf{sp}(2\ell,\mathbb{C}) = \mathsf{gl}_S(2\ell,\mathbb{C})$. These Lie algebras are known as the *sympletic Lie algebras.*

The Cartan matrix is:

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & \dots & \dots & 0 \\ 0 & -1 & \ddots & \ddots & & 0 \\ \vdots & \vdots & \vdots & & -1 & 0 \\ & \vdots & \vdots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -2 & 2 \end{pmatrix}.$$

The Dynkin diagram of ϕ is



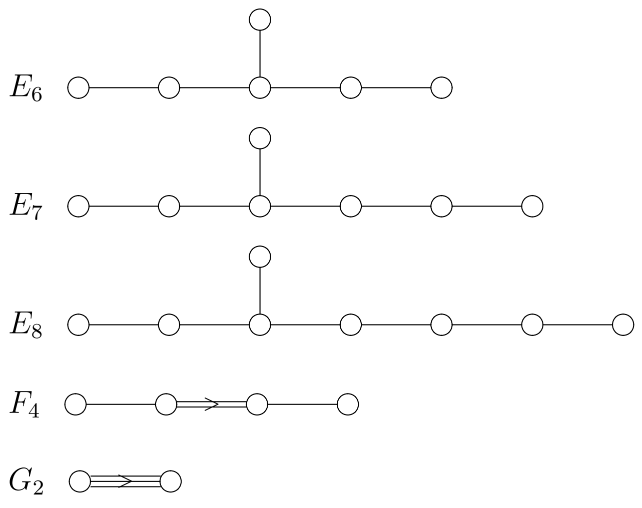
which is connected, so L is simple.

The root system of $\mathsf{sp}(2\ell, \mathbb{C})$ is said to have *type* C_ℓ. Since the root systems C_2 and B_2 have the same Dynkin diagram, we might expect that the Lie algebras $\mathsf{sp}(4, \mathbb{C})$ and $\mathsf{so}(5, \mathbb{C})$ would be isomorphic. This is the case. Two different Cartan subalgebras of L cannot give different root systems.

Lie algebra of types A_ℓ, B_ℓ, C_ℓ, and D_ℓ include all of the classical root systems. Explicit constructions of the exceptional Lie algebras are much more complicated and can be found in [11] Section 13.2.

A root system is determined up to isomorphism by its Dynkin diagram. Therefore, the problem of finding all the root systems can be reduced to the problem of finding all Dynkin diagrams.

Theorem 2.3.1. *Any irreducible root system is one of the classical root systems* A_n, $n \geq 1$, B_n, $n \geq 2$, C_n, $n \geq 3$, D_n, $n \geq 4$, *or one of the exceptional roots systems* G_2, F_4, E_6, E_7, E_8*:*



Proof. For a proof see [11] Chapter 13. □

2.4 The Universal Enveloping Algebra

For any Lie algebra L, one can construct its universal enveloping algebra $U(L)$ which passes from the non-associative structure of L to a unital associative algebra which captures the important properties of L. In this section we give a formal definition of the universal enveloping algebra and explain how it is constructed.

If A and B are vector spaces, we can form their tensor product as vector spaces. We define a product structure on $A \otimes B$ by defining

$$(a_1 \otimes b_1)(a_2 \otimes b_2) := a_1a_2 \otimes b_1b_2. \tag{2.3}$$

If in addition, both A and B have unit elements, then $1\otimes 1$ is a unit element for $A \otimes B$. We have an isomorphism of A into $A \otimes B$ given by $a \mapsto a \otimes 1$ when both A and B are associative algebras with units. Similarly for B. Note that

$$(a \otimes 1) \cdot (1 \otimes b) = a \otimes b = (1 \otimes b) \cdot (a \otimes 1).$$

Recall that a representation ρ assigns to any element $x \in L$ a linear operator $\rho(x)$. Sometimes it is useful to view a representation of a Lie algebra as a representation of an associative algebra. This can be done using the associated *universal enveloping algebra.*

Definition 2.4.1. A *universal enveloping algebra* is an associative unitary algebra $U(L)$ with a Lie algebra homomorphism $\varphi : L \to U(L)$ satisfying $\varphi([x, y]) = \varphi(x)\varphi(y) - \varphi(y)\varphi(x)$ for all $x, y \in L$ and such that for any Lie algebra homomorphism $f : L \to A$ into a unitary associative algebra A (which also satisfies $f([x, y]) = f(x)f(y) - f(y)f(x)$) there also exists a unique homomorphism $\psi : U(L) \to A$ such that the following diagram commutes:

$$\begin{tikzcd}
L \arrow[r, "f"] \arrow[dr, "\varphi"'] & A \\
& U(L) \arrow[u, "\psi"']
\end{tikzcd}$$

that is, $f = \psi \circ \varphi$.

Let V be a finite-dimensional vector space over F. In order to construct the universal enveloping algebra, we must first define the *tensor algebra* on V, $T(V) := \bigoplus_{i=0}^{\infty} T^iV$, where

$$\begin{aligned}
T^0 &= F \\
T^1 &= V \\
T^2 &= V \otimes V \\
T^m &= V \otimes V \otimes \ldots \otimes V (m \text{ times}).
\end{aligned}$$

If we take $V = L$ to be a Lie algebra, and let I be the two sided ideal in $T(L)$ generated by the elements $[x, y] - x \otimes y + y \otimes x$ then

$$U(L) := T(L)/I$$

is a universal enveloping algebra for L.

Let $q : T(L) \to U(L)$ be a linear map. For any basis $\{x_j \mid j \in J\}$ of L, the images $q(x_{j_1} \otimes \ldots \otimes x_{j_n})$ in L of tensor monomials $x_{j_1} \otimes \ldots \otimes x_{j_n}$ span the enveloping algebra over F, since they span the tensor algebra. Suppose J is ordered. Using the Lie bracket $[\,,\,]$ we can do a certain amount of rearranging of the x_{j_k} in a monomial. We anticipate that everything in $U(L)$ can be written to be a sum of monomials $x_{j_1} \cdot \ldots \cdot x_{j_n}$ where $j_1 \leq j_2 \leq \ldots \leq j_n$. A monomial in which the indices possess this ordering is a *standard monomial.*

Theorem 2.4.2 (Poincaré-Birkhoff-Witt)**.** *The standard monomials form a basis of $U(L)$.*

Proof. See [12]. □

Therefore, for any Lie algebra L with an ordered basis $x_1, x_2, \ldots$ the Poincaré-Birkhoff-Witt Theorem gives that the universal enveloping algebra $U(L)$ of L has a basis $x_1^{a_1} \ldots x_s^{a_s}$, $a \geq 0$ and the defining relations of $U(L)$ are the same as the relations $[x_i, x_j] = \sum a_{ij}^k x_k$, $a_{ij}^k \in F$ between the basis elements of L.

Example 2.4.3. *Let $L = sl(2, \mathbb{C})$ with its usual basis, f, e, h. We know the structure constants and therefore we can calculate in the algebra $U(L)$. We write $\mathbf{f}, \mathbf{e}, \mathbf{h}$ for the generators of $U(L)$. $U(L)$ contains all polynomials in $\mathbf{e}$, $\mathbf{f}$ and $\mathbf{h}$. But in addition, $U(L)$ contains products of these elements. We can use the relations $\mathbf{ef} - \mathbf{fe} = \mathbf{h}$, $\mathbf{he} - \mathbf{eh} = \mathbf{2e}$, and $\mathbf{hf} - \mathbf{fh} = -\mathbf{2f}$, valid in $U(L)$. Therefore, the associative algebra $U(\mathsf{sl}(2, \mathbb{C}))$ has as a vector space basis*

$$\{\mathbf{f}^a \mathbf{h}^b \mathbf{e}^c | a, b, c \geq 0\}.$$

Remark 2.4.4. Let L be a Lie algebra. The universal enveloping algebra of a L is unique (up to isomorphism).

2.5 Casimir Operator

We now look at the Casimir operator associated to a Lie algebra representation. Let L be a simple Lie algebra over $\mathbb{C}$. Let V be a faithful

L-module with associated representation $\rho : L \to \mathsf{End}(V)$. Let $B_\rho(x, y) := \mathsf{tr}(\rho(x) \circ \rho(y))$ be a symmetric and associative bilinear form on L. We call B_ρ the *trace form* associated to the representation ρ. As L is simple, $\ker(B_\rho) = 0$ which in turn implies that B_ρ is nondegenerate (Note that $B_{\mathsf{ad}} = \kappa$). The Casimir operator is an element of the centre of the universal enveloping algebra of the Lie algebra L, it is an important element of $U(L)$ as it can be easily written down. In this section we will investigate some general properties of the Casimir operator of a semisimple Lie algebra L, and we will show, using Schur's Lemma, that the Casimir operator is proportional to the identity.

Definition 2.5.1. Let $\{x_i\}$ be a basis of L with $\{y_i\}$ its dual basis with respect to B_ρ. The *Casimir element* Ω over the Lie algebra L with respect to ρ is defined by

$$\Omega = \sum_{i=1}^{n} x_i y_i \in U(L) \tag{2.4}$$

and the *Casimir operator* of the representation ρ is defined by

$$\Omega_\rho = \rho(\Omega) = \sum_{i=1}^{n} \rho(x_i)\rho(y_i). \tag{2.5}$$

Lemma 2.5.2. *Let Ω_ρ be the Casimir operator of ρ. Let $\{x_i\}$ and $\{y_i\}$ be dual bases with respect to B_ρ. Then $\mathsf{tr}(\Omega_\rho) = \dim L$.*

Proof. $\mathsf{tr}(\Omega_\rho) = \sum_i \mathsf{tr}\big(\rho(x_i) \circ \rho(y_j)\big) = \sum_i B_\rho(x_i, y_i) = \dim L$. $\square$

Remark 2.5.3. Recall the definition of the universal enveloping algebra (Definition 2.4.1). Let $\rho : L \to \mathsf{End}(V)$ be a representation and let $A = \mathsf{End}(V)$ be the unitary associative algebra given in Definition 2.4.1. We obtain an algebra homomorphism $\bar{\rho} : U(L) \to \mathsf{End}(V)$ given by $\bar{\rho}(w_1 \cdot \ldots \cdot w_k) = \rho(w_1) \circ \rho(w_2) \circ \ldots \circ \rho(w_k)$ (where, for example, $w_1 \cdot w_2 \in U(L)$ denotes the class of $w_1 \otimes w_2 \in \tau(L)$ in $U(L) = \tau(L)/I$).

Remark 2.5.4. Again, let $\{x_i\}$ be a basis of L with a dual basis with respect to B_ρ, $\{y_i\}$. Define $t := \sum_i x_i \otimes y_i \in L \otimes L$ to be a type of Casimir element. As B_ρ is nondegenerate, it induces an isomorphism $\Psi : L \otimes L \to \mathsf{End}(L)$ given by $\Psi(a \otimes b) = B_\rho(b, -)a$. Since $\sum_i B_\rho(y_i, x) \cdot x_i = x$ for all $x \in L$, then $\Psi(t) = \mathbb{1}_L$, hence t does not depend on the choice of basis $\{x_i\}$. This implies that Ω and Ω_ρ do not depend on the choice of basis. However, t and Ω depend on the bilinear form B_ρ, and Ω_ρ depends on ρ.

Lemma 2.5.5. *For each $x \in L$ with $[x, x_i] = \sum_j a_{ij} x_j$ and $[x, y_i] = \sum_j b_{ij} y_j$ we have $a_{ik} = -b_{ki}$.*

Proof. We can write $a_{ik} = \sum_j a_{ij}\delta_{jk} = \sum_j a_{ij} B_\rho(x_j, y_k) = B_\rho\big([x, x_i], y_k\big)$ $= -B_\rho\big(x_i, [x, y_k]\big)$ which is true by the associativity of B_ρ. This is equal to $-\sum_j b_{kj} B_\rho(x_i, y_j) = -b_{ki}$. □

Lemma 2.5.6. *For all $x \in L$ we have $[\rho(x), \Omega_\rho] = 0$.*

Proof. We know that if $x, y, z \in \mathsf{gl}(V)$, then $[x, yz] = [x, y]z + y[x, z]$. Using this and Equation (2.4) we have $[x, \Omega] = \sum_i [x, x_i y_i] = \sum_i [x, x_i] y_i + \sum_i x_i [x, y_i]$. We can write this as $\sum_{ij} a_{ij} x_j y_i + \sum_{ij} b_{ij} x_i y_j$. From Lemma 2.5.5 this is equal to zero. Applying the algebra homomorphism $\bar{\rho}$ to $[x, \Omega]$ we find that $[\rho(x), \Omega_\rho] = 0$ as required. □

If $\rho = \mathsf{ad} : L \to \mathsf{End}(L)$, then $B_\rho = \kappa$ and we have $[\Omega_{\mathsf{ad}}, \mathsf{ad}(x)] = 0$ for all $x \in L$. Hence, from Lemma 1.3.16 we have $\Omega_{\mathsf{ad}} \in \mathbb{C} \cdot \mathbb{1}_L$. As we have $\mathsf{tr}(\Omega_{\mathsf{ad}}) = \dim L$ from Lemma 2.5.2, we obtain $\Omega_{\mathsf{ad}} = \mathbb{1}_L$. Over $\mathbb{C}$, for simple Lie algebra L and faithful irreducible finite dimensional representation $\rho : L \to \mathsf{End}(V)$ we can use Schur's Lemma, Lemma 1.4.9, to obtain

$$\Omega_\rho = \frac{\dim L}{\dim V} \cdot \mathbb{1}_V. \tag{2.6}$$

Example 2.5.7. *Let $L = \mathsf{sl}(2, \mathbb{C})$ and consider the natural representation $\phi : L \to \mathsf{gl}(V)$, $\dim V = 2$. The dual basis with respect to the trace form of the standard basis $\{e, f, h\}$ is $\{f, e, \frac{h}{2}\}$.*
So,

$$\Omega_\phi = e \circ f + f \circ e + \frac{1}{2} h \circ h = \begin{pmatrix} \frac{3}{2} & 0 \\ 0 & \frac{3}{2} \end{pmatrix} = \frac{3}{2} \cdot 1_V, \quad V = \mathbb{C}^2.$$

Example 2.5.8. *Again, let $L = \mathsf{sl}(2, \mathbb{C})$ and consider the adjoint representation $\mathsf{ad} : L \to \mathsf{gl}(L)$, $\dim(L) = 3$. The dual basis with respect to the Killing form is $\{\frac{f}{4}, \frac{e}{4}, \frac{h}{8}\}$. We have,*

$$\Omega_{\mathsf{ad}} = \frac{1}{4} \mathsf{ad}\, e \circ \mathsf{ad}\, f + \frac{1}{4} \mathsf{ad}\, f \circ \mathsf{ad}\, e + \frac{1}{8} \mathsf{ad}\, h \circ \mathsf{ad}\, h.$$

We use the results of Example 1.4.5 to obtain

$$\Omega_{\mathsf{ad}} = \frac{1}{4} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \frac{1}{4} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix} + \frac{1}{4} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} = \mathbb{1}_V,\ V = \mathbb{C}^3.$$

Remark 2.5.9. Let e_{ij} be the $n \times n$ matrix with 1 at the ij-th position and zero at all other positions. Let $h_i = e_{ii} - e_{i+1,i+1}$ for $1 \leq i < n$, and $h_i^* = \frac{1}{n}((n-i)e_{11} + (n-i)e_{22} + \ldots + (n-i)e_{ii} - ie_{i+1,i+1} - \ldots - ie_{nn})$. Then the *Casimir element* Ω associated to the natural representation ϕ in $\mathsf{sl}(n)$ with respect to the trace form is given by

$$\Omega_\phi = \sum_{1 \leq i < n} h_i h_i^* + \sum_{i \neq j} e_{ij} e_{ji}. \tag{2.7}$$

Chapter 3

Preliminaries to the Classical Yang-Baxter Equation

In this chapter we introduce the maps $\varphi_1 : L \otimes L \to \mathsf{Hom}(L, L)$ and $\varphi_2 : L \otimes L \otimes L \to \mathsf{Hom}(L \otimes L, L)$, which are induced by the Killing form κ on L where L is a simple finite-dimensional Lie algebra over $\mathbb{C}$. These maps will be our main tool in reformulating the proofs of Belavin and Drinfeld into a coordinate-free language. We prove several propositions which will be used extensively throughout the following chapters. First however, we must take a brief look at some required preliminaries in complex analysis.

3.1 Complex Analysis Preliminaries

We know that functions, holomorphic in a disc, can be represented there by a power series.

Theorem 3.1.1. *Let f be holomorphic on a closed disc $\bar{D}(z_0, R)$, $R > 0$. Let C_R be the circle bounding the disc. Then f has a power series expansion*

$$f(z) = \sum a_n (z - z_0)^n$$

whose coefficients a_n are given by the formula:

$$a_n = \frac{1}{2\pi i} \int_{C_R} \frac{f(z)}{(z - z_0)^{n+1}} dz.$$

Proof. For a proof see [17] Section III Theorem 7.3. □

A somewhat similar representation can be derived for functions holomorphic in an annulus $R_1 < |z - z_0| < R_2$. These two-sided power series are known as *Laurent expansions.*

Theorem 3.1.2. *Let r, R be positive numbers with $0 \leq r < R$. Let A be the annulus consisting of all complex numbers z such that $r < |z| < R$, and let f be a holomorphic function on A. Let $r < s < S < R$. Then f has a Laurent expansion*

$$f(z) = \sum_{-\infty}^{\infty} a_n z^n$$

which converges absolutely and uniformly on $s \leq |z| \leq S$. Let C_R and C_r be the circles of radius R and r, respectively. Then the coefficients a_n are obtained by the formula:

$$a_n = \frac{1}{2\pi i} \int_{C_R} \frac{f(z)}{z^{n+1}} dz \quad \textit{if } n \geq 0,$$
$$a_n = \frac{1}{2\pi i} \int_{C_r} \frac{f(z)}{z^{n+1}} dz \quad \textit{if } n < 0.$$

Proof. For a proof see [17] Section V, Theorem 2.1. □

Definition 3.1.3. Suppose that U is an open subset of the complex numbers $\mathbb{C}$, and the point $z_0 \in U$. Let f be a complex differentiable function defined in some neighbourhood around z_0. If f is not defined at z_0 but has values defined on $U \backslash \{z_0\}$ then z_0 is an *isolated singularity* of f. The singularity z_0 is called

1. *removable* if there exists a holomorphic function g defined on all U such that $f(z) = g(z)$ for all $z \in U \setminus \{z_0\}$. The function g is a continuous replacement for the function f,

2. a *pole* if there exists a holomorphic function g defined on U and a natural number n such that $f(z) = g(z)/(z - z_0)^n$ for all $z \in Z \backslash \{z_0\}$. The derivative at a pole may or may not exist. If $g(z_0) \neq 0$, then we say that z_0 is a pole of order n,

3. *essential* if z_0 is neither removable nor a pole. The point z_0 is an essential singularity if and only if the Laurent expansion has infinitely many powers of negative degree.

In the preceding chapters, the singularities we will examine will be of type 2, poles. A pole of a function f is said to be *simple* if it is of order 1. If f has a simple pole at z_0 then $a_{-1} = \lim_{z \to z_0}(z - z_0)f(z)$.

Example 3.1.4. *The function $\frac{1}{z}$ has a simple pole at the origin.*

Example 3.1.5. *The function $\frac{1}{\sin z}$ has a simple pole at the origin. This is because*

$$\sin z = z - \frac{z^3}{3!} + \frac{z^5}{5!} - \cdots$$

Therefore,

$$\frac{1}{\sin z} = \frac{1}{z} \cdot \frac{1}{1 - (\frac{z^2}{3!} - \frac{z^4}{5!} + \cdots)}.$$

We know that $\frac{1}{1-h} = 1 + h + h^2 + \cdots$ for $|h| < 1$, and as the series $\left|\left(\frac{z^2}{3!} - \frac{z^4}{5!} + \cdots\right)\right| = |\frac{\sin z}{z} - 1| < \epsilon$ for all $\epsilon > 0$ and $\delta > 0$ if $|z| < \delta$ we can write this as

$$\frac{1}{\sin z} = \frac{1}{z}(1 + \text{ higher terms}).$$

Definition 3.1.6. Let $f : U \to \mathbb{C}$ be defined on an open subset $U \subset \mathbb{C}$ except for a discrete set of points S which are poles. Then we say that f is *meromorphic* on U. We say that f is *meromorphic at a point* z_0 if f is meromorphic on some open set U containing z_0.

Example 3.1.7. *Let $P(z)$ be a polynomial. Then $f(z) = \frac{1}{P(z)}$ is a meromorphic function.*

Remark 3.1.8. If z_0 is a pole of f, then there exists an integer n such that $(z - z_0)^n f(z)$ is holomorphic in a neighbourhood of z_0.

Remark 3.1.9. Let f be a meromorphic function on an open subset $U \subset \mathbb{C}$ such that there exists an open disc $D \subset U$ with $f|_D \equiv 0$. Then $f \equiv 0$ on all of U. This is known as the *Identity Theorem* or the *Uniqueness Principle*. This theorem can be extended from $\mathbb{C}$ to $\mathbb{C}^n$, $n > 1$ [20].

Definition 3.1.10. Suppose z_0 is an isolated singularity of f with Laurent series expansion $\sum_{k \in \mathbb{Z}} a_k (z - z_0)^k$ in a punctured disc about z_0. Then a_{-1} is the *residue of f at* z_0, denoted by $\mathsf{Res}(f, z_0)$.

Example 3.1.11. *The residue of $(\sin z)/z^2$ at $z = 0$ is found using the equation*

$$\begin{aligned} \frac{\sin z}{z^2} &= \frac{1}{z^2}\left(z - \frac{z^3}{3!} + \cdots\right) \\ &= \frac{1}{z} + \text{higher terms.} \end{aligned}$$

The desired residue is 1.

Example 3.1.12. *The residue of $\frac{e^z}{z^5}$ at $z = 0$ is found by*

$$\begin{aligned}\frac{e^z}{z^5} &= \frac{1}{z^5} \cdot \left(1 + z + \frac{z^2}{2!} + \cdots\right) \\ &= \frac{1}{z^5} + \frac{1}{z^4} + \frac{1}{2!z^3} + \frac{1}{3!z^2} + \frac{1}{4!z} + \cdots\end{aligned}$$

and the desired residue is $\frac{1}{24}$.

Cauchy's Residue Theorem is used to evaluate contour integrals (integrals where the function to be integrated is evaluated along a curve) of analytic functions over closed curves that have one or more poles inside the contour.

Theorem 3.1.13. *(Cauchy Residue Theorem) Let $f(z)$ be analytic inside and on a simple closed contour C, except for a finite number of isolated singular points $z_1, \dots, z_n$ located inside C. Then*

$$\int_C f(z)dz = 2\pi i \sum_{j=1}^{n} a_j \tag{3.1}$$

where a_j is the residue of $f(z)$ at $z = z_j$.

Definition 3.1.14. An elliptic function is a meromorphic function f defined on $\mathbb{C}$ for which there exists two nonzero complex numbers ω_1 and ω_2 with ω_1/ω_2 not real such that

$$f(z + \omega_1) = f(z + \omega_2) = f(z) \quad \text{for all } z \in \mathbb{C}$$

wherever $f(z)$ is defined. That is,

$$f(z + m\omega_1 + n\omega_2) = f(z) \quad \text{for all } z \in \mathbb{C},\ m, n \in \mathbb{Z}.$$

Remark 3.1.15. Fix ω_1 and ω_2. The corresponding elliptic functions form a field. The parallelogram with vertices $0, \omega_1, \omega_2, \omega_1 + \omega_2$ is called the *fundamental parallelogram* Π. Any $\mathbb{C}$-translate $\alpha + \Pi$ of Π is a *period parallelogram.* We can conclude the following: For any period parallelogram $\alpha + \Pi$, any point in $\mathbb{C}$ is congruent modulo $\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$ to one and only one point of $\alpha + \Pi$.

Theorem 3.1.16. *If $f(z)$ is an elliptic function with no poles on the boundary C of a period parallelogram $\alpha + \Pi$, then the sum of the residues of $f(z)$ in $\alpha + \Pi$ is zero.*

Proof. For a proof see [16] Proposition 6.3. □

3.2 Isomorphisms induced by the Killing form

In this section we prove several propositions which will be used extensively in later chapters. The strategy we employ here, whereby we introduce two linear maps φ_1 and φ_2, is to prove the commutativity of several diagrams with useful results. Throughout this section we will assume that L is a finite-dimensional simple Lie algebra over the complex field $\mathbb{C}$.

Recall from Theorem 1.6.12 that as L is simple, the Killing form κ is nondegenerate. From Lemma 1.6.17 this means that κ induces an isomorphism $\varphi : L \to L^*$.

Definition 3.2.1. Let the linear map $\varphi : L \to L^*$ be defined by $\varphi(x) = \kappa(x, -)$.

1. We define the map $\varphi_1 = \mathbb{1}_L \otimes \varphi : L \otimes L \to \mathsf{Hom}(L, L) \cong L \otimes L^*$ by $\varphi_1(a \otimes b) = \kappa(b, -)a$. That is, $\varphi_1(a \otimes b)(p) = \kappa(b, p)a$ for all $a, b, p \in L$.
2. We define the map $\varphi_2 = \mathbb{1}_L \otimes \varphi \otimes \varphi : L \otimes L \otimes L \to \mathsf{Hom}(L \otimes L, L) \cong L^* \otimes L^* \otimes L$ by $\varphi_2(a \otimes b \otimes c) = \kappa(b, -)\kappa(c, -)a$. That is, $\varphi_2(a \otimes b \otimes c)(p \otimes q) = \kappa(b, p)\kappa(c, q)a$ for all $a, b, c, p, q \in L$

Definition 3.2.2. We use Remark 2.5.4 to define an element $t \in L \otimes L$ by $\varphi_1(t) = \mathbb{1}_L$. This tensor t is called the *Casimir element.*

Lemma 3.2.3. *Let $\{I_\mu\}$ be an orthonormal basis in L with respect to the Killing form. The Casimir element can be written*

$$t = \sum_\mu I_\mu \otimes I_\mu \in L \otimes L.$$

Proof. As $\{I_\mu\}$ is an orthonormal basis we know that $\kappa(I_\alpha, I_\beta) = \delta_{\alpha\beta}$. From Definition 3.2.2 we know that $\varphi_1(t) = \mathbb{1}_L$. We calculate $\varphi_1(\sum_\mu I_\mu \otimes I_\mu)(v) = \sum_\mu \kappa(I_\mu, v)I_\mu = v$. Hence, $\varphi_1(\sum_\mu I_\mu \otimes I_\mu) = \mathbb{1}_L = \varphi_1(t)$. As φ_1 is an isomorphism we must have $\sum_\mu I_\mu \otimes I_\mu = t$. □

Proposition 3.2.4. *Let $f, g \in \mathsf{Hom}(L, L)$, and let f^* be the adjoint map of f with respect to the Killing form (see Definition 1.6.7). Let $\sigma : \mathsf{Hom}(L, L) \to \mathsf{Hom}(L, L)$ be given by $\sigma(h) = f \circ h \circ g^*$ for $h \in L$. Then the following diagram is commutative:*

$$\begin{tikzcd}
L \otimes L \arrow[r, "\varphi_1"] \arrow[d, "f\otimes g"'] & \mathsf{Hom}(L, L) \arrow[d, "\sigma"] \\
L \otimes L \arrow[r, "\varphi_1"] & \mathsf{Hom}(L, L)
\end{tikzcd}$$

Proof. For $a, b, x \in L$ we have $\varphi_1((f \otimes g)(a \otimes b))(x) = \varphi_1(f(a) \otimes g(b))(x)$. Using Definition 3.2.1 we can write this as $\kappa(g(b), x)f(a) = f(\kappa(b, g^*(x))a)$ which is equal to $f(\varphi_1(a \otimes b)(g^*(x))) = \sigma(\varphi_1(a \otimes b))(x)$ as required. $\square$

Lemma 3.2.5. *Let $A, B, C \in \mathsf{End}(L)$. The following diagram is commutative:*

$$\begin{tikzcd}
L \otimes L \otimes L \arrow[r, "\varphi_2"] \arrow[d, "A\otimes B\otimes C"'] & \mathsf{Hom}(L \otimes L, L) \arrow[d, "\sigma"] \\
L \otimes L \otimes L \arrow[r, "\varphi_2"] & \mathsf{Hom}(L \otimes L, L)
\end{tikzcd}$$

Where σ is the map which sends $\psi \in \mathsf{Hom}(L \otimes L, L)$ to $A \circ \psi \circ (B^* \otimes C^*)$.

Proof. We calculate

$$\sigma\big(\varphi_2(x \otimes y \otimes z)\big)(p \otimes q) = A \circ \varphi_2(x \otimes y \otimes z) \circ (B^* \otimes C^*)(p \otimes q)$$

which is equal to $A\big(\kappa(y, B^*(p))\kappa(z, C^*(q))x\big) = A\big(\kappa(B(y), p)\kappa(C(z), q)x\big)$. This can be written as $\kappa(B(y), p)\kappa(C(z), q)A(x)$ and using Definition 3.2.1 we know that this is equal to $\varphi_2(A(x) \otimes B(y) \otimes C(z))(p \otimes q)$ or $\Bigg(\varphi_2\big((A \otimes B \otimes C)(x \otimes y \otimes z)\big)\Bigg)(p \otimes q)$. $\square$

Recall again the definition of the adjoint of a map (Definition 1.6.7). Let us denote by $* : \mathsf{Hom}(L, L) \to \mathsf{Hom}(L, L)$ the map which send f to its adjoint f^*.

Proposition 3.2.6. *Let $\tau : L \otimes L \to L \otimes L$ denote the swapping map $\tau(a \otimes b) = b \otimes a$. The following diagram is commutative:*

$$\begin{tikzcd}
L \otimes L \arrow[r, "\varphi_1"] \arrow[d, "\tau"'] & \mathsf{Hom}(L, L) \arrow[d, "*"] \\
L \otimes L \arrow[r, "\varphi_1"] & \mathsf{Hom}(L, L)
\end{tikzcd}$$

Proof. Let $f = \varphi_1(a \otimes b) = \kappa(b, -)a$ and let $g = \varphi_1(\tau(a \otimes b)) = \kappa(a, -)b$. Then, $\kappa(f(u), v) = \kappa(\kappa(b, u)a, v)$ for all $u, v \in L$. This is equal to $\kappa(u, b)\kappa(a, v)$ $= \kappa(u, \kappa(a, v)b) = \kappa(u, g(v))$. Therefore, $g = f^*$, that is $\varphi_1(\tau(A)) = (\varphi_1(A))^*$ for all $A \in L \otimes L$. $\square$

Let A be an associative algebra with unit containing L (for example the universal enveloping algebra). Then $A \otimes A$ and $A \otimes A \otimes A$ are also associative algebras. Multiplication in these algebras is defined by $(a \otimes b)(p \otimes q) = ap \otimes bq$ and $(a \otimes b \otimes c)(p \otimes q \otimes r) = ap \otimes bq \otimes cr$ respectively, for all $a, b, c, p, q, r \in A$. If we apply Remark 1.1.3 to $A \otimes A$ and $A \otimes A \otimes A$ we obtain

1. $[a \otimes b, p \otimes q] = (a \otimes b)(p \otimes q) - (p \otimes q)(a \otimes b) = ap \otimes bq - pa \otimes qb$. Similarly,

2. $[a \otimes b \otimes c, p \otimes q \otimes r] = ap \otimes bq \otimes cr - pa \otimes qb \otimes rc$

for all $a, b, c, p, q, r \in A$.

Remark 3.2.7. Let $a, b, c \in L$ and let $a \otimes b, 1 \otimes c \in A \otimes A$. Then $[a \otimes b, 1 \otimes c] = a \otimes [b, c] \in L \otimes L$.

Definition 3.2.8. Define the linear maps $\phi_{12}, \phi_{13}, \phi_{23} : A \otimes A \to A \otimes A \otimes A$ by: $\phi_{12}(a \otimes b) = a \otimes b \otimes 1$, $\phi_{13}(a \otimes b) = a \otimes 1 \otimes b$ and $\phi_{23}(a \otimes b) = 1 \otimes a \otimes b$. For brevity we will write $(a \otimes b)^{ij}$ for $\phi_{ij}(a \otimes b)$.

Example 3.2.9. *1.*

$$\begin{aligned}\big[(a \otimes b)^{12}, (c \otimes d)^{13}\big] &= [a \otimes b \otimes 1, c \otimes 1 \otimes d] \\ &= (a \otimes b \otimes 1)(c \otimes 1 \otimes d) - (c \otimes 1 \otimes d)(a \otimes b \otimes 1) \\ &= a \cdot c \otimes b \cdot 1 \otimes 1 \cdot d - c \cdot a \otimes 1 \cdot b \otimes d \cdot 1 \\ &= (ac - ca) \otimes b \otimes d \\ &= [a, c] \otimes b \otimes d.\end{aligned}$$

2. Similarly,

$$\big[(a \otimes b)^{12}, (c \otimes d)^{23}\big] = a \otimes [b, c] \otimes d$$

3. and,

$$\big[(a \otimes b)^{13}, (c \otimes d)^{23}\big] = a \otimes c \otimes [b, d].$$

Proposition 3.2.10. *Let $a, b \in L$. Let $\phi : L \to L$ be an automorphism of Lie algebras, that is, $\phi\big([a, b]\big) = [\phi(a), \phi(b)]$. For $c \in L$,*

$$[a \otimes b, \phi(c) \otimes 1 + 1 \otimes c] = 0$$

is equivalent to

$$[\phi^{-1}(a) \otimes b, c \otimes 1 + 1 \otimes c] = 0.$$

Proof. $[a \otimes b, \phi(c) \otimes 1] + [a \otimes b, 1 \otimes c] = [a, \phi(c)] \otimes b + a \otimes [b, c] = 0$. If we apply $\phi^{-1} \otimes 1$ to this equation we find $[\phi^{-1}(a), c] \otimes b + \phi^{-1}(a) \otimes [b, c] = 0$ or, $[\phi^{-1}(a) \otimes b, c \otimes 1] + [\phi^{-1}(a) \otimes b, 1 \otimes c] = 0$, that is, $[\phi^{-1}(a) \otimes b, c \otimes 1 + 1 \otimes c] = 0$. □

Proposition 3.2.11. *Let $a, b, c \in L$. Then, for all x in L we have*

1. $[\varphi_1(a \otimes b)(x), c] = \varphi_1\big([a \otimes b, c \otimes 1]\big)(x)$.

2. $\varphi_1(a \otimes b)[x, c] = -\varphi_1\big([a \otimes b, 1 \otimes c]\big)(x)$.

Proof.
1. $[\varphi_1(a \otimes b)(x), c] = [\kappa(b, x)a, c] = \kappa(b, x)[a, c] = \varphi_1([a, c] \otimes b)(x) = \varphi_1([a \otimes b, c \otimes 1])(x)$.

2. $\varphi_1(a \otimes b)[x, c] = \kappa(b, [x, c])a = -\kappa([b, c], x)a = -\varphi_1(a \otimes [b, c])(x) = -\varphi_1([a \otimes b, 1 \otimes c])(x)$.

□

Proposition 3.2.12. *Let $A \in L\otimes L$ and consider the linear maps $\sigma_1, \sigma_2, \sigma_3 : L \otimes L \to L \otimes L \otimes L$ where $\sigma_1(B) = [A^{12}, B^{13}]$, $\sigma_2(B) = [A^{12}, B^{23}]$ and $\sigma_3(B) = [A^{13}, B^{23}]$. The following diagrams are commutative:*

1.
$$\begin{tikzcd} L \otimes L \arrow[r, "\varphi_1"] \arrow[d, "\sigma_1"'] & \mathsf{Hom}(L, L) \arrow[d, "\widetilde{\sigma}_1"] \\ L \otimes L \otimes L \arrow[r, "\varphi_2"] & \mathsf{Hom}(L \otimes L, L) \end{tikzcd}$$

where $\widetilde{\sigma}_1(f)(x \otimes y) = [\varphi_1(A)(x), f(y)]$ for $f \in \mathsf{Hom}(L, L)$.

2.
$$\begin{tikzcd} L \otimes L \arrow[r, "\varphi_1"] \arrow[d, "\sigma_2"'] & \mathsf{Hom}(L, L) \arrow[d, "\widetilde{\sigma}_2"] \\ L \otimes L \otimes L \arrow[r, "\varphi_2"] & \mathsf{Hom}(L \otimes L, L) \end{tikzcd}$$

where $\widetilde{\sigma}_2(f)(x \otimes y) = -\varphi_1(A)([x, f(y)])$ for $f \in \mathsf{Hom}(L, L)$.

3.
$$\begin{tikzcd} L \otimes L \arrow[r, "\varphi_1"] \arrow[d, "\sigma_3"] & \mathsf{Hom}(L, L) \arrow[d, "\widetilde{\sigma}_3"] \\ L \otimes L \otimes L \arrow[r, "\varphi_2"] & \mathsf{Hom}(L \otimes L, L) \end{tikzcd}$$

with $\widetilde{\sigma}_3(f)(x \otimes y) = \varphi_1(A)([f^*(x), y])$ for $f \in \mathsf{Hom}(L, L)$.

Proof. Without loss of generality, we can assume that $A = a \otimes b$. For $B = c \otimes d \in L \otimes L$ we have, for all $x, y \in L$,

1. $\varphi_2(\sigma_1(B))(x \otimes y) = \varphi_2\big([(a \otimes b)^{12}, (c \otimes d)^{13}]\big)(x \otimes y)$. From Example 3.2.9 part (1) this can be written as $\varphi_2([a, c] \otimes b \otimes d)(x \otimes y) = [\kappa(b, x)a, \kappa(d, y)c]$, which can be simplified using Definition 3.2.1 to $[\varphi_1(A)(x), \varphi_1(B)(y)] = \widetilde{\sigma}_1(\varphi_1(B))(x \otimes y)$

2. Similarly, we have $\varphi_2(\sigma_2(B))(x\otimes y) = \varphi_2([(a\otimes b)^{12},(c\otimes d)^{23}])(x\otimes y)$. Using Example 3.2.9 part (2) we can write this as $\varphi_2(a\otimes[b,c]\otimes d)(x\otimes y)$ which is equal to $-\kappa(b,[x,c])\kappa(d,y)a = -\kappa(b,[x,\kappa(d,y)c])a$ or $-\varphi_1(A)([x,\varphi_1(B)(y)]) = \widetilde{\sigma}_2(\varphi_1(B))(x\otimes y)$
3. We have $\varphi_2(\sigma_3(B))(x\otimes y) = \varphi_2([(a\otimes b)^{13},(c\otimes d)^{23}])(x\otimes y)$. We can use Example 3.2.9 part (3) to write this as $\varphi_2(a\otimes c\otimes[b,d])(x\otimes y) = \kappa(b,[\kappa(c,x)d,y])a = \kappa(b[\varphi_1(\tau B)(x),y])a$. This gives $\varphi_1(A)([\varphi_1(B)^*(x),y])$ which is equal to $\widetilde{\sigma}_3(\varphi_1(B))(x\otimes y)$ as required.

□

Definition 3.2.13. We define the map $\mathcal{L}: L\otimes L\to L$ to be the Lie bracket. That is, $\mathcal{L}(x\otimes y)=[x,y]$.

Proposition 3.2.14. *Let $x,y\in L$. Let t be the Casimir element. Then $\varphi_2([t^{12},t^{13}])(x\otimes y)=\mathcal{L}(x\otimes y)=[x,y]$*

Proof. From Proposition 3.2.12 we can see that

$$\varphi_2([t^{12},t^{13}])(x\otimes y) = [\varphi_1(t)(x),\varphi_1(t)(y)] = [x,y].$$

□

Remark 3.2.15. The three identities in Proposition 3.2.12 can be rewritten as follows,

$$\begin{aligned}
\varphi_2([A^{12},B^{13}]) &= \mathcal{L}\circ(\varphi_1(A)\otimes\varphi_1(B)),\\
\varphi_2([A^{12},B^{23}]) &= -\varphi_1(A)\circ\mathcal{L}\circ(\mathbb{1}_L\otimes\varphi_1(B)),\\
\varphi_2([A^{13},B^{23}]) &= \varphi_1(A)\circ\mathcal{L}\circ(\varphi_1(\tau(B))\otimes\mathbb{1}_L).
\end{aligned}$$

Remark 3.2.16. From Proposition 3.2.4 we obtain for all $A\in L\otimes L$:

$$\varphi_1\Big(\big(\varphi_1(A)\otimes\mathbb{1}_L\big)(t)\Big) = \varphi_1(A)\circ\varphi_1(t) = \varphi_1(A).$$

Therefore, $\big(\varphi_1(A)\otimes\mathbb{1}_L\big)(t)=A$. Similarly, we can show $\big(\mathbb{1}_L\otimes\varphi_1(\tau A)\big)(t)=A$. Hence,

$$\begin{aligned}
A^{12} &= \big(\varphi_1(A)\otimes\mathbb{1}_L\otimes\mathbb{1}_L\big)t^{12} = \big(\mathbb{1}_L\otimes\varphi_1(\tau A)\otimes\mathbb{1}_L\big)t^{12},\\
A^{13} &= \big(\varphi_1(A)\otimes\mathbb{1}_L\otimes\mathbb{1}_L\big)t^{13} = \big(\mathbb{1}_L\otimes\mathbb{1}_L\otimes\varphi_1(\tau A)\big)t^{13},\\
A^{23} &= \big(\mathbb{1}_L\otimes\varphi_1(A)\otimes\mathbb{1}_L\big)t^{23} = \big(\mathbb{1}_L\otimes\mathbb{1}_L\otimes\varphi_1(\tau A)\big)t^{23}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
[A^{12}, B^{13}] &= \Big[\big(\mathbb{1}_L \otimes \varphi_1(\tau A) \otimes \mathbb{1}_L\big)t^{12}, \big(\mathbb{1}_L \otimes \mathbb{1}_L \otimes \varphi_1(\tau B)\big)t^{13}\Big] \\
&= \big(\mathbb{1}_L \otimes \varphi_1(\tau A) \otimes \varphi_1(\tau B)\big)[t^{12}, t^{13}]. \\
[A^{12}, B^{23}] &= \Big[\big(\varphi_1(A) \otimes \mathbb{1}_L \otimes \mathbb{1}_L\big)t^{12}, \big(\mathbb{1}_L \otimes \mathbb{1}_L \otimes \varphi_1(\tau B)\big)t^{23}\Big] \\
&= \big(\varphi_1(A) \otimes \mathbb{1}_L \otimes \varphi_1(\tau B)\big)[t^{12}, t^{23}]. \\
[A^{13}, B^{23}] &= \Big[\big(\varphi_1(A) \otimes \mathbb{1}_L \otimes \mathbb{1}_L\big)t^{13}, \big(\mathbb{1}_L \otimes \varphi_1(B) \otimes \mathbb{1}_L\big)t^{23}\Big] \\
&= \big(\varphi_1(A) \otimes \varphi_1(B) \otimes \mathbb{1}_L\big)[t^{12}, t^{23}].
\end{aligned}$$

Corollary 3.2.17. *Let t be the Casimir element. For all $A \in L \otimes L$ we have*

$$\begin{aligned}
[t^{12}, A^{13} + A^{23}] &= 0, \\
[A^{12} + A^{13}, t^{23}] &= 0.
\end{aligned}$$

Proof. Using Proposition 3.2.12 for all $x, y \in L$ we find

$$\begin{aligned}
\varphi_2([t^{12}, A^{13} + A^{23}])(x \otimes y) &= \varphi_2([t^{12}, A^{13}])(x \otimes y) + \varphi_2([t^{12}, A^{23}])(x \otimes y) \\
&= [\varphi_1(t)(x), \varphi_1(A)(y)] - \varphi_1(t)[x, \varphi_1(A)(y)] \\
&= [x, \varphi_1(A)(y)] - [x, \varphi_1(A)(y)] = 0.
\end{aligned}$$

Similarly, we find that $[A^{12} + A^{13}, t^{23}] = 0$. □

Corollary 3.2.18. *Let t be the Casimir element. Then,*

$$[t^{12}, t^{13}] = -[t^{12}, t^{23}] = [t^{13}, t^{23}].$$

Proof. From Corollary 3.2.17 with $A = t$ we have $[t^{12}, t^{13} + t^{23}] = 0$ and $[t^{12} + t^{13}, t^{23}] = 0$. Then, $[t^{12}, t^{13}] = -[t^{12}, t^{23}]$ and $-[t^{12}, t^{23}] = [t^{13}, t^{23}]$. □

Corollary 3.2.19. *Let t be the Casimir element and let $B \in L \otimes L$. Then,*

$$\begin{aligned}
&[t^{12}, B^{13}] = 0 \textit{ implies that } B = 0, \\
&[t^{12}, B^{23}] = 0 \textit{ implies that } B = 0, \\
&[t^{13}, B^{23}] = 0 \textit{ implies that } B = 0, \\
&[t^{23}, B^{12}] = 0 \textit{ implies that } B = 0.
\end{aligned}$$

Proof. From part (1) of Proposition 3.2.12 we obtain, $\varphi_2\big([t^{12}, B^{13}]\big)(x \otimes y) = [x, \varphi_1(B)(y)]$ for all $x, y \in L$. If $[t^{12}, B^{13}] = 0$, then $[x, \varphi_1(B)(y)] = 0$ for all $x, y \in L$. That is, $\varphi_1(B)(y) \in Z(L)$ for all $y \in L$. But as L is simple, $Z(L) = 0$ so $\varphi_1(B) = 0$ and as φ_1 is injective, we have $B = 0$. We also have,

$$\varphi_2\big([t^{12}, B^{23}]\big)(x \otimes y) = -\big[x, \varphi_1(B)(y)\big],$$
$$\varphi_2\big([t^{13}, B^{23}]\big)(x \otimes y) = \big[\varphi_1(\tau B)(x), y\big].$$

Similar calculations allow us to conclude that if these equations equal zero, then $B = 0$. Finally, We calculate $\varphi_2\big([X^{12}, t^{23}]\big)(x \otimes y) = -\varphi_1(X)[x, \varphi_1(t)(y)]$ $= -\varphi_1(X)[x, y] = 0$ for all $x, y \in L$. As L is simple and φ_1 is an isomorphism, we must have $X = 0$. $\square$

Definition 3.2.20. A constant $r_0 \in L \otimes L$ is called *nondegenerate* if $\varphi_1(r_0)$ is an isomorphism.

Definition 3.2.21. A meromorphic function $r : U \to L \otimes L$, $U \subseteq \mathbb{C}$ is called nondegenerate if there exists $u_0 \in U$ such that r is holomorphic at u_0 and $r(u_0)$ is nondegenerate in the sense of Definition 3.2.20.

Remark 3.2.22. The function r being nondegenerate means that the meromorphic function $\det(\varphi_1(r(u)))$ for $u \in U$ is not identically zero on U. Therefore, it has only isolated zeros.

Lemma 3.2.23. *Let $A \in L \otimes L$. If A is nondegenerate, then $[A^{12}, A^{13}] \neq 0$.*

Proof. If A is nondegenerate, then from Definition 3.2.20, $\varphi_1(A)$ is an isomorphism. This implies that $\mathsf{im}\big(\varphi_1(A)\big) = L$. From Proposition 3.2.12 part (1) we have $\varphi_2\big([A^{12}, A^{13}]\big)(x \otimes y) = [\varphi_1(A)(x), \varphi_1(A)(y)]$ for all $x, y \in L$. If we assume that $[A^{12}, A^{13}] = 0$ we obtain now $[X, Y] = 0$ for all $X, Y \in \mathsf{im}\big(\varphi_1(A)\big) = L$, that is, L is abelian. But L is simple, therefore, $[A^{12}, A^{13}] \neq 0$. $\square$

Corollary 3.2.24. *If $A \in L \otimes L$ with $[A^{12}, A^{13}] = 0$ then there exists a vector subspace $V \subset L$, $V \neq L$ such that $A \in V \otimes L$.*

Proof. Let $V := \mathsf{im}\big(\varphi_1(A)\big) \subset L$. From Lemma 3.2.23 we have that A is degenerate, that is, $\varphi_1(A)$ is not an isomorphism. Therefore, $V \neq L$. From Remark 3.2.16 we have $A = (\varphi_1(A) \otimes \mathbb{1}_L)(t) \in \mathsf{im}\big(\varphi_1(A)\big) \otimes L = V \otimes L$ as required. $\square$

Proposition 3.2.25. *Let L be a simple Lie algebra over $\mathbb{C}$. Let $A \in L \otimes L$, then $[A, 1 \otimes h + h \otimes 1] = 0$ for all $h \in L$ if and only if $A = \lambda \cdot t$ for some $\lambda \in \mathbb{C}$, where t is the Casimir element.*

Proof. Let us rewrite $[1 \otimes h + h \otimes 1, A]$ as $(\mathbb{1}_L \otimes \mathsf{ad}(h) + \mathsf{ad}(h) \otimes \mathbb{1}_L)(A)$. We can now use Lemma 3.2.4 to obtain

$$\varphi_1\big((\mathbb{1}_L \otimes \mathsf{ad}(h) + \mathsf{ad}(h) \otimes \mathbb{1}_L)(A)\big) = \varphi_1(A) \circ \mathsf{ad}(h)^* + \mathsf{ad}(h) \circ \varphi_1(A). \quad (3.2)$$

We need to calculate $\mathsf{ad}(h)^*$. We know that $\kappa([h,u],v) = -\kappa([u,h],v) = -\kappa(u,[h,v])$. So $\mathsf{ad}(h)^* = -\mathsf{ad}(h)$. Using this in Equation (3.2) we find

$$\begin{aligned}\varphi_1\big((\mathbb{1}_L \otimes \mathsf{ad}(h) + \mathsf{ad}(h) \otimes \mathbb{1}_L)(A)\big) &= -\varphi_1(A) \circ \mathsf{ad}(h) + \mathsf{ad}(h) \circ \varphi_1(A) \\ &= [\mathsf{ad}(h), \varphi_1(A)].\end{aligned}$$

Therefore, $\varphi_1\big([A, 1 \otimes h + h \otimes 1]\big) = [\varphi_1(A), \mathsf{ad}(h)]$ and $[A, 1 \otimes h + h \otimes 1] = 0$ for all $h \in L$, if and only if $[\varphi_1(A), \mathsf{ad}(h)] = 0$ for all $h \in L$. We know that $\varphi_1(A) \in \mathsf{End}(L)$ and so we can apply Lemma 1.3.16 to give us $\varphi_1(A) = \lambda \cdot \mathbb{1}_L = \lambda \cdot \varphi_1(t)$. Therefore, $A = \lambda t$, $\lambda \in \mathbb{C}$ as required. $\square$

Chapter 4

Nondegenerate Solutions to the CYBE

In [6] and [4] A.A. Belavin and V.G.Drinfeld investigated the nondegenerate solutions to the classical Yang-Baxter equation (CYBE), for finite-dimensional, simple Lie algebras over $\mathbb{C}$. In this chapter we will discuss the CYBE in more detail. Our aim is to prove Belavin and Drinfeld's characteristisation of non-degeneracy. We have already studied in some detail in Chapter 3 the linear maps φ_1 and φ_2. Here we will use these maps to rewrite the proofs of Belavin and Drinfeld in a coordinate-free language. Throughout this chapter we will assume that L is a finite-dimensional simple Lie algebra over the complex field $\mathbb{C}$.

4.1 The Classical Yang-Baxter Equation

For a function $r : U \times U \to L \otimes L$ we consider $r^{ij} : U \times U \to A \otimes A \otimes A$, where A is a unital associative algebra with unit (see page 50), defined by $r^{ij}(a,b) = (r(a,b))^{ij}$, $1 \leq i < j \leq 3$.

Remark 4.1.1. In terms of a basis $\{I_\mu\}$ of L we can write

$$r(u,v) = \sum_{\mu\nu} r_{\mu\nu}(u,v) I_\mu \otimes I_\nu,$$

where $r_{\mu\nu}$ is a meromorphic function. We can then obtain a *matrix of the* *coefficients of $r(u,v)$:*

$$r(u,v) = \begin{pmatrix} r_{11}(u,v) & r_{12}(u,v) & \dots & r_{1n}(u,v) \\ \vdots & \vdots & \dots & \vdots \\ r_{n1}(u,v) & r_{n2}(u,v) & \dots & r_{nn}(u,v) \end{pmatrix}.$$

Definition 4.1.2. The *classical Yang-Baxter equation* (CYBE) is the functional equation

$$[r^{12}(u_1,u_2), r^{13}(u_1,u_3)] + [r^{12}(u_1,u_2), r^{23}(u_2,u_3)] + [r^{13}(u_1,u_3), r^{23}(u_2,u_3)] = 0. \quad (4.1)$$

Remark 4.1.3. Solutions r to the classical Yang-Baxter equation are frequently called *classical r-matrices* (or simply r-matrices). They play an important role in mathematical physics, representation theory, integrable systems and statistical mechanics.

4.2 Equivalence of Solutions

Given a solution r of the CYBE, Equation (4.1), we can obtain a solution which is equivalent to this one in two ways. In this section we prove two propositions which obtain equivalent solutions to the CYBE and give a version of the CYBE which depends on the difference of the parameters only.

Proposition 4.2.1. *If $r(u_1,u_2)$ is a solution of the classical Yang-Baxter equation and f is a function with values in $\mathsf{Aut}(L)$, then $\tilde{r}(u_1,u_2) = (f(u_1) \otimes f(u_2)\ )(r(u_1,u_2))$ is also a solution.*

Proof. We know that

$$[r^{12}(u_1,u_2), r^{13}(u_1,u_3)] + [r^{12}(u_1,u_2), r^{23}(u_2,u_3)] + [r^{13}(u_1,u_3), r^{23}(u_2,u_3)] = 0 \quad (4.2)$$

and we must show that

$$\Big[\big((f(u_1)\otimes f(u_2))r(u_1,u_2)\big)^{12}, \big((f(u_1)\otimes f(u_3))r(u_1,u_3)\big)^{13}\Big] + \Big[\big((f(u_1)\otimes f(u_2))r(u_1,u_2)\big)^{12}, \big((f(u_2)\otimes f(u_3))r(u_2,u_3)\big)^{23}\Big] + \Big[\big((f(u_1)\otimes f(u_3))r(u_1,u_3)\big)^{13}, \big((f(u_2)\otimes f(u_3))r(u_2,u_3)\big)^{23}\Big] = 0. \quad (4.3)$$

Applying φ_2 to Equation (4.2) and using Proposition 3.2.12 we see that for all $x, y \in L$ we have,

$$\Big[\varphi_1\big(r(u_1,u_2)\big)(x), \varphi_1\big(r(u_1,u_3)\big)(y)\Big] - \varphi_1\big(r(u_1,u_2)\big)\big[x, \varphi_1(r(u_2,u_3)(y)\big] \\ + \varphi_1\big(r(u_1,u_3)\big)\big[\varphi_1(r(u_2,u_3))^*(x), y\big] = 0$$

and similarly Equation (4.3) is equivalent to

$$\Big[\varphi_1\Big(\big(f(u_1)\otimes f(u_2)\big)r(u_1,u_2)\Big)(x), \varphi_1\Big(\big(f(u_1)\otimes f(u_3)\big)r(u_1,u_3)\Big)(y)\Big] \\ - \varphi_1\Big(\big(f(u_1)\otimes f(u_2)\big)r(u_1,u_2)\Big)\Big[x, \varphi_1\Big(\big(f(u_2)\otimes f(u_3)\big)r(u_2,u_3)\Big)(y)\Big] \\ + \varphi_1\Big(\big(f(u_1)\otimes f(u_3)\big)r(u_1,u_3)\Big)\Big[\varphi_1\Big(\big(f(u_2)\otimes f(u_3)\big)r(u_2,u_3)\Big)^*(x), y\Big] = 0. \tag{4.4}$$

If we look at the first bracket of Equation (4.4) and using Lemma 3.2.4 we find that,

$$\Big[\varphi_1\Big(\big(f(u_1)\otimes f(u_2)\big)r(u_1,u_2)\Big)(x), \varphi_1\Big(\big(f(u_1)\otimes f(u_3)\big)r(u_1,u_3)\Big)(y)\Big] \\ = \big[\big(f(u_1)\circ\varphi_1(r(u_1,u_2))\circ f(u_2)^*\big)(x), \big(f(u_1)\circ\varphi_1(r(u_1,u_3))\circ f(u_3)^*\big)(y)\big].$$

Because $f(u_1) \in \mathsf{Aut}(L)$ we can write this as

$$= f(u_1)\Big[\big(\varphi_1(r(u_1,u_2))\circ f(u_2)^*\big)(x), \big(\varphi_1(r(u_1,u_3))\circ f(u_3)^*\big)(y)\Big].$$

From Proposition 3.2.12 part (1) we can write this as

$$= f(u_1)\Big(\varphi_2\big([r^{12}(u_1,u_2), r^{13}(u_1,u_3)]\big)\Big(\big(f(u_2)^*\otimes f(u_3)^*\big)(x\otimes y)\Big)\Big).$$

Now we use Lemma 3.2.5 to obtain,

$$= \varphi_2\Big(\big(f(u_1)\otimes f(u_2)\otimes f(u_3)\big)\big([r^{12}(u_1,u_2), r^{13}(u_1,u_3)]\big)\Big)(x\otimes y).$$

Computing the second and third bracket similarly, but using the identity $f(u)^*\circ f(u) = \mathbb{1}_L$ from the proof of Lemma 1.6.9, we find that the left-hand-side of Equation (4.4) is equivalent to

$$\varphi_2(f(u_1)\otimes f(u_2)\otimes f(u_3))\Big([r^{12}(u_1,u_2), r^{13}(u_1,u_3)] \\ + [r^{12}(u_1,u_2), r^{23}(u_2,u_3)] + [r^{13}(u_1,u_3), r^{23}(u_2,u_3)]\Big)$$

which is equal to zero by assumption of the theorem. Therefore, $\big(f(u_1)\otimes f(u_2)\big)r(u_1,u_2)$ is a solution of CYBE, Equation (4.1). □

We now define an equivalence relation on solutions to the CYBE as follows: Solutions r and $\tilde{r}$ are equivalent if there exists a function $f : U \to \mathsf{Aut}(L)$ such that $\tilde{r}(u_1, u_2) = \big(f(u_1) \otimes f(u_2)\big)\big(r(u_1, u_2)\big)$:

1. reflexivity holds if we let $f(u) = \mathbb{1}_L$ for all $u \in U$.

2. the relation is symmetric as $f(u)^{-1}$ is also an automorphism of L, so we can apply $f(u_1)^{-1} \otimes f(u_2)^{-1}$ to both sides of the relation to obtain $(f(u_1)^{-1} \otimes f(u_2)^{-1})\tilde{r}(u_1, u_2) = r(u_1, u_2)$.

3. to show that the relation is transitive, we let $\bar{r}(u_1, u_2) = \big(g(u_1) \otimes g(u_2)\big)\ \tilde{r}(u_1, u_2)$. Then $\bar{r}(u_1, u_2) = \big(g(u_1) \otimes g(u_2)\big)\big(f(u_1) \otimes f(u_2)\big)\big(r(u_1, u_2)\big)$ which is equal to $\big(g(u_1)f(u_1) \otimes g(u_2)f(u_2)\big)\big(r(u_1, u_2)\big)$.

Definition 4.2.2. The function r is said to be *invariant* with respect to $h \in L$ if $[h \otimes 1 + 1 \otimes h, r] = 0$. A solution which is invariant with respect to every element of a subalgebra $H \subset L$ is said to be invariant with respect to that subalgebra.

We are now ready to present the second method of obtaining equivalent solutions to the CYBE, Equation (4.1):

Proposition 4.2.3. *Let r be a solution of CYBE, Equation (4.1), invariant with respect to a subalgebra $H \subset L$, and let the tensor $r_0 \in H \otimes H$ satisfy*

$$[r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}] = 0$$

$$r_0 = -\tau(r_0)$$

then the function $\tilde{r} := r + r_0$ is also a solution to Equation (4.1)

Proof. First, we will calculate the first bracket:

$$\begin{aligned}[r^{12}(u_1, u_2) + r_0^{12}, r^{13}(u_1, u_3) + r_0^{13}] = [r^{12}(u_1, u_2), r^{13}(u_1, u_3)] \\ + [r^{12}(u_1, u_2), r_0^{13}] + [r_0^{12}, r^{13}(u_1, u_3)] + [r_0^{12}, r_0^{13}].\end{aligned}$$

Similarly, the second bracket can be expanded to

$$\begin{aligned}[r^{12}(u_1, u_2) + r_0^{12}, r^{23}(u_2, u_3) + r_0^{23}] = [r^{12}(u_1, u_2), r^{23}(u_2, u_3)] \\ + [r^{12}(u_1, u_2), r_0^{23}] + [r_0^{12}, r^{23}(u_2, u_3)] + [r_0^{12}, r_0^{23}]\end{aligned}$$

and the third bracket can be expanded to

$$\begin{aligned}[r^{13}(u_1,u_3)+r_0^{13}, r^{23}(u_2,u_3)+r_0^{23}] = {} & [r^{13}(u_1,u_3), r^{23}(u_2,u_3)] \\ & + [r^{13}(u_1,u_3), r_0^{23}] + [r_0^{13}, r^{23}(u_2,u_3)] + [r_0^{13}, r_0^{23}].\end{aligned}$$

From the assumptions of the theorem, the first and fourth Lie brackets of each expansion added together are zero. We are left with

$$\begin{aligned}[r^{12}(u_1,u_2), r_0^{13}] + [r_0^{12}, r^{13}(u_1,u_3)] + [r^{12}(u_1,u_2), r_0^{23}] + [r_0^{12}, r^{23}(u_2,u_3)] \\ + [r^{13}(u_1,u_3), r_0^{23}] + [r_0^{13}, r^{23}(u_2,u_3)].\end{aligned}$$

This simplifies to

$$[r^{12}(u_1,u_2), r_0^{13}+r_0^{23}] + [r^{13}(u_1,u_3), r_0^{23}-r_0^{12}] - [r^{23}(u_2,u_3), r_0^{12}+r_0^{13}]$$

or,

$$[r^{12}(u_1,u_2), r_0^{13}+r_0^{23}] + [r^{13}(u_1,u_3), r_0^{23}+(\tau r_0)^{12}] - [r^{23}(u_2,u_3), r_0^{12}+r_0^{13}].$$

We can calculate,

$$\begin{aligned}&\varphi_2\big([r^{12}(u_1,u_2), r_0^{13}+r_0^{23}]\big) \\ &\quad = \Big[\varphi_1\big(r(u_1,u_2)\big)(x), \varphi_1(r_0)(y)\Big] - \varphi_1\big(r(u_1,u_2)\big)\Big[x, \varphi_1(r_0)(y)\Big] \\ &\quad = -\Big[\varphi_1(r_0)(y), \varphi_1\big(r(u_1,u_2)\big)(x)\Big] + \varphi_1\big(r(u_1,u_2)\big)\Big[\varphi_1(r_0)(y), x\Big] \\ &\quad = -\Big(\mathsf{ad}\big(\varphi_1(r_0)(y)\big)\circ\varphi_1\big(r(u_1,u_2)\big)\Big)(x) \\ &\qquad\quad + \Big(\varphi_1\big(r(u_1,u_2)\big)\circ\mathsf{ad}\big(\varphi_1(r_0)(y)\big)\Big)(x) \\ &\quad = \Big[\varphi_1\big(r(u_1,u_2)\big), \mathsf{ad}\big(\varphi_1(r_0)(y)\big)\Big].\end{aligned}$$

Without loss of generality, we can let $r_0 = a \otimes b$ with $a, b \in H$. Then $\varphi_1(r_0)(x) = \kappa(b,x)a$ is in H for all $x \in L$. The assumption of the Proposition gives, $[h \otimes 1 + 1 \otimes h, r(u_1,u_2)] = 0$, from the proof of Proposition 3.2.25 we know that implies that $[\varphi_1\big(r(u_1,u_2)\big), \mathsf{ad}(h)] = 0$ for all $h \in H$. Hence, $[r^{12}(u_1,u_2), r_0^{13}+r_0^{23}] = 0$. Similarly it can be shown that $[r^{13}(u_1,u_3), r_0^{23}+(\tau r_0)^{12}] = 0$ and $[r^{23}(u_2,u_3), r_0^{12}+r_0^{13}] = 0$ □

Definition 4.2.4. Solutions where $r(u_1,u_2) = -\tau(r(u_2,u_1))$ are known as *unitary* solutions.

By a theorem of Belavin and Drinfeld [5], each nondegenerate solution $r(u,v)$ to the CYBE, is equivalent to a solution $\tilde{r}(u_1,u_2)$ which depends only on u_1-u_2. From now on we will consider only solutions depending on u_1-u_2, only after Proposition 5.2.11 will we revert back to the 2 parameter case. If r depends on the difference, u_1-u_2, only, we write $r(u_1-u_2)$ for $r(u_1,u_2)$ and consider r as a function of one variable. In this case, $r(u_1,u_2)$ is denoted by $r(u_1-u_2)$ and CYBE, Equation (4.1), can be written as

$$[r^{12}(u), r^{13}(u+v)] + [r^{12}(u), r^{23}(v)] + [r^{13}(u+v), r^{23}(v)] = 0 \tag{4.5}$$

and such a solution is unitary if and only if $r(u) = -\tau(r(-u))$.

From here on out, unless otherwise stated, we refer to Equation (4.5) as the CYBE.

4.3 Properties of Constant Solutions to CYBE

In this section we investigate constant solutions to the CYBE. When we assume that $r \equiv r_0 \in L \otimes L$, we get the simpler constant coefficient equation

$$[r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}] = 0. \tag{4.6}$$

The aim of this section is to show that Equation (4.6) does not have nondegenerate solutions. Throughout this section we will let $T = \varphi_1(r_0)$

Lemma 4.3.1. *Let $r_0 \in L \otimes L$. Then*

1. *$r_0 + \tau(r_0) = t$ is equivalent to $T + T^* = \mathbb{1}_L$.*
2. *Equation (4.6) is equivalent to*

$$[T(x), T(y)] - T\Big([x, T(y)] - [T^*(x), y]\Big) = 0. \tag{4.7}$$

3. *If $T + T^* = \mathbb{1}_L$, then Equation (4.6) is equivalent to*

$$[T(x), T(y)] = T\Big([x, T(y)] + [T(x), y] - [x, y]\Big). \tag{4.8}$$

Proof. 1. Because $\varphi_1 : L \otimes L \to \mathsf{End}(L)$ is an isomorphism and $\varphi_1(t) = \mathbb{1}_L$, we obtain $r_0 + \tau(r_0) = t$ if and only if $\varphi_1(r_0) + \varphi_1(\tau(r_0)) = \mathbb{1}_L$, From Proposition 3.2.6 this can be written, $\varphi_1(r_0) + \varphi_1(r_0)^* = \mathbb{1}_L$ or $T + T^* = \mathbb{1}_L$.

2. As before, let t be the Casimir element. Using Remark 3.2.16 we can obtain

$$\begin{aligned}
[r_0^{12}, r_0^{13}] &= (\mathbb{1}_L \otimes T^* \otimes T^*)[t^{12}, t^{13}], \\
[r_0^{12}, r_0^{23}] &= (T \otimes \mathbb{1}_L \otimes T^*)[t^{12}, t^{23}], \\
[r_0^{13}, r_0^{23}] &= (T \otimes T \otimes \mathbb{1}_L)[t^{13}, t^{23}].
\end{aligned}$$

Recall from Corollary 3.2.18 that $[t^{12}, t^{13}] = -[t^{12}, t^{13}] = [t^{13}, t^{23}]$. We can now apply φ_2 to Equation (4.6) to find

$$\begin{aligned}
&\varphi_2\big([r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}]\big) \\
&\quad = \varphi_2\big((\mathbb{1}_L \otimes T^* \otimes T^* - T \otimes \mathbb{1}_L \otimes T^* + T \otimes T \otimes \mathbb{1}_L)([t^{12}, t^{13}])\big).
\end{aligned} \tag{4.9}$$

We know from Proposition 3.2.14 that $\varphi_2([t^{12}, t^{13}]) = \mathcal{L}$. Using this and the results from Lemma 3.2.5 we can write Equation (4.9) as

$$= \mathcal{L} \circ (T \otimes T) - T \circ \mathcal{L} \circ (\mathbb{1}_L \otimes T) + T \circ \mathcal{L} \circ (T^* \otimes \mathbb{1}_L).$$

Therefore, Equation (4.6) holds if and only if, for all $x, y \in L$ we have

$$[T(x), T(y)] - T([x, T(y)]) + T([T^*(x), y]) = 0. \tag{4.10}$$

3. We know from part (2) that Equation (4.6) is equivalent to Equation (4.7), by assumption we have that $T^* = \mathbb{1}_L - T$. So we have,

$$[T(x), T(y)] = T\big([x, T(y)] + [T(x), y] - [x, y]\big).$$

□

Proposition 4.3.2. *Let the tensor $r_0 \in L \otimes L$ be nondegenerate with $r_0 = -\tau(r_0)$. Let $S \in \mathsf{End}(L)$ be the inverse of T. Then r_0 satisfies Equation (4.6) if and only if S is a derivation of L.*

Proof. From Proposition 4.3.1 part (2) we know that r_0 being a solution to Equation (4.6) is equivalent to

$$[T(x), T(y)] - T[x, T(y)] + T[T^*(x), y] = 0$$

for all $x, y \in L$. By definition $S = T^{-1}$ and because r_0 is unitary we have $T^* = -T$. S is an isomorphism so we can replace x with $S(x)$ and replace y with $S(y)$ to obtain

$$[x, y] - T[S(x), y] - T[x, S(y)] = 0$$

for all $x, y \in L$. Applying S to this we find

$$S[x,y] - [S(x),y] - [x,S(y)] = 0$$

for all $x, y \in L$. That is, S is a derivation. □

Proposition 4.3.3. *A tensor $r_0 \in L \otimes L$ satisfying the following system of equations:*

$$r_0 + \tau(r_0) = t, \tag{4.11}$$

$$[r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}] = 0. \tag{4.12}$$

is degenerate.

Proof. From Lemma 4.3.1 we know that Equation (4.11) is equivalent to $T + T^* = \mathbb{1}_L$ and Equation (4.12) is equivalent to Equation (4.7). We have

$$(T-1)[T(x),T(y)] = T\big([T(x),T(y)]\big) - [T(x),T(y)]$$

from Equation (4.7) this is equal to

$$= T\big([T(x),T(y)]\big) - T\big([x,T(y)]\big) + T\big([T^*(x),y]\big).$$

From Equation (4.11) we can replace T^* with $\mathbb{1}_L - T$ to obtain a more symmetric expression:

$$\begin{aligned} &= T\big([T(x),T(y)]\big) - T\big([x,T(y)]\big) + T\big([x,y]\big) - T\big([T(x),y]\big) \\ &= T\big([T(x),(T-\mathbb{1}_L)(y)]\big) - T\big([x,(T-\mathbb{1}_L)(y)]\big) \\ &= T\big([(T-\mathbb{1}_L)(x),(T-\mathbb{1}_L)(y)]\big). \end{aligned} \tag{4.13}$$

Assuming that $r_0 \in L \otimes L$ is nondegenerate, then $\det(T) \neq 0$. This means that $\det(T^*) \neq 0$ and so $\det(T - \mathbb{1}_L) \neq 0$. We define $\theta : L \to L$ by $\theta = T \circ (T - \mathbb{1}_L)^{-1}$. Therefore, $\theta - \mathbb{1}_L = T(T-\mathbb{1}_L)^{-1} - \mathbb{1}_L = T(T-\mathbb{1}_L)^{-1} - (T-\mathbb{1}_L)\big((T-\mathbb{1}_L)^{-1}\big) = (T-\mathbb{1}_L)^{-1}$. If we apply $(T-\mathbb{1}_L)^{-1}$ to Equation (4.13) we find

$$[T(x),T(y)] = (T-\mathbb{1}_L)^{-1} \circ T\big([(T-\mathbb{1}_L)(x),(T-\mathbb{1}_L)(y)]\big). \tag{4.14}$$

But $\theta = T \circ (T-\mathbb{1}_L)^{-1} = (T-\mathbb{1}_L)^{-1} \circ T$ and if we let $x' = (T-\mathbb{1}_L)(x)$ and $y' = (T-\mathbb{1}_L)y$ we can write Equation (4.14) as

$$[\theta(x'),\theta(y')] = \theta\big([x',y']\big).$$

That is, θ is a Lie algebra automorphism. We know that $\det((T-\mathbb{1}_L)^{-1}) \neq 0$ and so $\det(\theta - \mathbb{1}_L) \neq 0$. But Theorem 1.3.17 shows that there is no automorphism θ such that $\det(\theta - \mathbb{1}_L) \neq 0$. Therefore, if T satisfies Equations (4.11) and (4.12), then $\det(T) = 0$, that is, r_0 is degenerate. □

We will now show that nondegenerate constant solutions to Equation (4.6), are automatically unitary.

Proposition 4.3.4. *Let $r_0 \in L \otimes L$ be a nondegenerate solution of Equation (4.6). Then r_0 also satisfies*

$$r_0 + \tau(r_0) = 0. \tag{4.15}$$

Proof. We know that Equation (4.6) holds, and from Lemma 4.3.1 we know that this is equivalent to Equation (4.7). Interchanging x and y in Equation (4.7) we obtain for all $y, z \in L$

$$[T(y), T(x)] - T[y, T(x)] + T[T^*(y), x] = 0$$

using the skew-symmetry of the Lie bracket on the above equation we have

$$-[T(x), T(y)] + T[T(x), y] - T[x, T^*(y)] = 0. \tag{4.16}$$

Adding (4.7) and (4.16) we obtain

$$T\left([(T + T^*)(x), y]\right) = T\left([x, (T + T^*)(y)]\right).$$

Because T is an isomorphism we have

$$[(T + T^*)(x), y] = [x, (T + T^*)(y)].$$

For $z \in L$ we can apply the map $\kappa(-, z)$ to this equation to obtain

$$\kappa\big([(T + T^*)(x), y], z\big) = \kappa\big([x, (T + T^*)(y)], z\big).$$

Because the Killing form is associative, and $(T + T^*)^* = T^* + T$ we have,

$$\begin{aligned}
\kappa\big([(T + T^*)(x), y], z\big) &= \kappa\big([x, (T + T^*)(y)], z\big) \\
\kappa\big((T + T^*)(x), [y, z]\big) &= \kappa\big(x, [(T + T^*)(y), z]\big) \\
\kappa\big(x, (T + T^*)[z, y]\big) &= \kappa\big(x, [z, (T + T^*)(y)]\big)
\end{aligned}$$

for all $x \in L$. Therefore,

$$(T + T^*)[z, y] = [z, (T + T^*)(y)]$$

for all $y \in L$. That is,

$$(T + T^*) \circ \mathsf{ad}(z) = \mathsf{ad}(z) \circ (T + T^*)$$

or,

$$[T + T^*, \mathsf{ad}(z)] = 0$$

for all $z \in L$. From Proposition 3.2.25 this implies that $T + T^* = \lambda \cdot \mathbb{1}_L$ or $r + \tau(r) = \lambda \cdot t$ for some $\lambda \in \mathbb{C}$. If $\lambda \neq 0$ from Proposition 4.3.3 this equation along with Equation (4.6) implies that $r \in L \otimes L$ is degenerate. Therefore, by the assumptions of the proposition, $\lambda = 0$ □

Corollary 4.3.5. *Let L be a simple Lie algebra. Then Equation (4.6) does not have nondegenerate solutions.*

Proof. Let us assume that $r_0 \in L \otimes L$ is a nondegenerate solution to Equation (4.6). By Proposition 4.3.4 it is unitary, that is, $r_0 + \tau(r_0) = 0$. As r_0 is nondegenerate, T is an isomorphism, therefore by Proposition 4.3.2, $S = T^{-1}$ is a derivation. As L is simple, Proposition 1.6.13 implies that there exists $a \in L$ such that $S = \mathsf{ad}(a)$. But then $a \in \ker(\mathsf{ad}(a))$ as $[a, a] = 0$, hence $\mathsf{ad}(a)$ cannot be an isomorphism, and r_0 is degenerate. $\square$

4.4 Characterisation of Nondegenerate Solutions

For a solution r to the CYBE, meromorphic in some disc $U \subseteq \mathbb{C}$ and centred at zero, being nondegenerate implies that it also satisfies three more equivalent conditions. In this section we aim to prove this equivalence by reproducing the proof of Belavin and Drinfeld [4] in a coordinate-free language.

Proposition 4.4.1. *Let r be a solution to the CYBE, Equation (4.5), which is holomorphic in a disc U containing zero and let there exist a $u_0 \in U$ such that the tensor $r(u_0)$ is nondegenerate. Then the tensor $r(0)$, is also nondegenerate.*

Proof. Setting $v = 0$ in Equation (4.5), we obtain

$$[r^{12}(u), r^{13}(u)] + [r^{12}(u) + r^{13}(u), r^{23}(0)] = 0$$

So,

$$\varphi_2\big([r^{12}(u), r^{13}(u)]\big) + \varphi_2\big([r^{12}(u) + r^{13}(u), r^{23}(0)]\big) = 0.$$

Using Remark 3.2.15 we obtain for all $x, y \in L$

$$\Big[\varphi_1(r(u))(x), \varphi_1(r(u))(y)\Big] = \varphi_1\big(r(u)\big)\Big([x, \varphi_1(r(0))(y)] - [\varphi_1(r(0))^*(x), y]\Big). \tag{4.17}$$

By the condition of the proposition $\varphi_1\big(r(u_0)\big)$ is an isomorphism. Therefore for each $u \in U$, there exists a linear map $\psi_u : L \to L$ such that $\psi_u \circ \varphi_1(r(u_0)) = \varphi_1(r(u))$. For ease of reading we will write $X = \varphi_1(r(u_0))(x)$ and $Y = \varphi_1(r(u_0))(y)$. As $\varphi_1(r(u_0))$ is an isomorphism, each $X, Y \in L$ can be written in this form with appropriate $x, y \in L$. We will also write $I =$

$[x, \varphi_1(r(0))(y)] - [\varphi_1(r(0))^*(x), y]$ and letting $u = u_0$ we see that Equation (4.17) can be written

$$[X, Y] = \varphi_1(r(u_0))(I). \tag{4.18}$$

Applying ψ_u to Equation (4.18) we obtain

$$\psi_u([X, Y]) = \psi_u\big(\varphi_1(r(u_0))(I)\big) = \varphi_1(r(u))(I) = \varphi_1(r(u))\big([x, \varphi_1(r(0))(y)] \\ - [\varphi_1(r(0))^*(x), y]\big).$$

From Equation (4.17) we find

$$\psi_u([X, Y]) = \Big[\varphi_1(r(u))(x), \varphi_1(r(u))(y)\Big] = [\psi_u(X), \psi_u(Y)]$$

for all $X, Y \in \mathsf{im}(\varphi_1(r(u_0)) = L$, $u \in U$. Therefore, ψ_u is an endomorphism of L as a Lie algebra. Lemma 1.6.9 then gives that $\det(\psi_u) \in \{0, -1, 1\}$. We have $\psi_u = \varphi_1(r(u)) \circ \varphi_1(r(u_0))^{-1}$ and $\psi_{u_0} = \varphi_1(r(u_0)) \circ \varphi_1(r(u_0))^{-1} = 1$. Therefore, $\det(\psi_{u_0}) = 1$. Due to the continuity of r we must have $\det(\psi_u) = 1$ for any u. In particular $\det \psi_0 = 1$ and $r(0)$ is nondegenerate. □

Proposition 4.4.2. *Let $A, B \in L \otimes L$ and let $V := \mathsf{im}\big(\varphi_1(B)\big)$. Define $M := \{x \in L \,|\, [x, v] \in V$ for all $v \in V\}$. Then*

1. *$M \subset L$ is a Lie subalgebra.*
2. *if $[A^{12}, B^{23}] \in L \otimes V \otimes L$ then $A \in L \otimes M$,*
3. *if $[B^{12}, A^{13}] \in V \otimes L \otimes L$ then $A \in M \otimes L$.*

Proof.
1. For $x, y \in M, v \in V$ we have $[[x, y], v] = [x, [y, v]] + [y, [v, x]]$ due to the Jacobi identity. But $[y, v] \in V$ and $[v, x] \in V$ so $[x, [y, v]]$ and $[y, [v, x]]$ are both in V. So their sum is also in V.
2. Define a map $\tau_{213}(a \otimes b \otimes c) = b \otimes a \otimes c$. Then $\tau_{213}([A^{12}, B^{23}]) = [(\tau A)^{12}, B^{13}] \in V \otimes L \otimes L$. Clearly, for $\xi \in V \otimes L \otimes L$ we have $\varphi_2(\xi) : L \otimes L \to V$ as $\varphi_2(a \otimes b \otimes c)(x \otimes y) = \kappa(b, x)\kappa(c, y)a$. Hence, for all $x, y \in L$ we have $\varphi_2\big([(\tau A)^{12}, B^{13}]\big)(x \otimes y) \in V$, that is (using Remark 3.2.15), $[\varphi_1(\tau A)(x), \varphi_1(B)(y)] \in V$ for all $x, y \in L$. Now, $V = \mathsf{im}(\varphi_1(B))$ implies $B \in V \otimes L$. Hence, $\varphi_1(\tau A)(x) \in M$ for all $x \in L$, that is, $A \in L \otimes M$ as required.
3. Similarly, if $[B^{12}, A^{13}] \in V \otimes L \otimes L$, then $\mathsf{im}\big(\varphi_2([B^{12}, A^{13}])\big) \subset V$ and again using Remark 3.2.15, for all $x, y \in L$ we have $[\varphi_1(B)(x), \varphi_1(A)(y)] \in V$. As $V = \mathsf{im}(\varphi_1(B))$ this means that $[\varphi_1(A)(y), v] \in V$ for all $v \in V$, that is $\varphi_1(A)(y) \in M$ for all $y \in L$. Hence, $A \in M \otimes L$.

□

Remark 4.4.3. For any subspaces $V, W \subset L$, we have $L \otimes W \cap V \otimes L = V \otimes W$.

Proof. Let $\{v_1, \ldots, v_r\}$ be a basis of V and $\{w_1, \ldots, w_s\}$ be a basis of W. We can extend these bases to two different bases for L, say $L = v_1, \ldots, v_r, v_{r+1}, \ldots, v_n = V \oplus L_1$ and $L = w_1, \ldots, w_s, w_{s+1}, \ldots, w_n = W \oplus L_2$. Now we can write

$$L \otimes W = (V \oplus L_1) \otimes W \cong (V \otimes W) \oplus (L_1 \otimes W)$$
$$V \otimes L = V \otimes (W \oplus L_2) \cong (V \otimes W) \oplus (V \otimes L_2).$$

So $V \otimes W$ is in the intersection. □

We are now ready to formulate a proof for the following theorem, which was given in [4].

Theorem 4.4.4. *Let L be a finite-dimensional simple Lie algebra over $\mathbb{C}$. Let r be a solution of CYBE in the class of meromorphic functions, defined in some disk $U \subseteq \mathbb{C}$ with centre at zero. Then the following four conditions are equivalent:*

- *(A) the function r has at least one pole, and there does not exist a neighbourhood $U' \subset U$ of zero and a Lie subalgebra $L_1 \subset L$ such that $r(u) \in L_1 \otimes L_1$ for all $u \in U'$ at which r is holomorphic;*
- *(B) all the poles of r are simple and at $u = 0$ the residue is of the form $\lambda \cdot t$, $\lambda \in \mathbb{C}$;*
- *(C) r has for $u = 0$ a simple pole with residue of the form $\lambda \cdot t$, $\lambda \in \mathbb{C}$.*
- *(D) r is nondegenerate.*

Proof. Recall that L denotes a finite-dimensional simple Lie algebra over $\mathbb{C}$. We will fix the nondegenerate invariant bilinear form on L to be κ.

(A) $\Rightarrow$ (B) To prove this, suppose that r has a pole of order $k \geq 1$ at $u = \gamma$, and set $\rho = \lim_{u \to \gamma} (u - \gamma)^k r(u)$. Multiplying both sides of Equation (4.5) by $(v - \gamma)^k$ we have

$$(v-\gamma)^k [r^{12}(u), r^{13}(u+v)] + (v-\gamma)^k [r^{12}(u), r^{23}(v)] + (v-\gamma)^k [r^{13}(u+v), r^{23}(v)] = 0$$

and taking the limit as v tends to γ, we get

$$[r^{12}(u), \rho^{23}] + [r^{13}(u+\gamma), \rho^{23}] = 0 \tag{4.19}$$

for $u \in U$ with $u+\gamma \in U$. In exactly the same way, multiplying by $(u-\gamma)^k$ and letting u tend to γ in Equation (4.5), we get

$$[\rho^{12}, r^{13}(v+\gamma)] + [\rho^{12}, r^{23}(v)] = 0 \tag{4.20}$$

for $v \in U$ with $v+\gamma \in U$. We will now show from these equations that $[\rho^{12}, \rho^{13}] \neq 0$. First, Suppose that $V \subset L$ is the smallest vector space such that $\rho \in V \otimes L$, that is $V = \mathsf{im}(\varphi_1(\rho))$, and set $M = \{x \in L \,|\, [x, V] \subset V\}$. M is a Lie subalgebra. Using Example 3.2.9 part (3) we obtain $[r^{13}(u+\gamma), \rho^{23}] \in L \otimes V \otimes L$ as $\rho \in V \otimes L$. It follows from Equation (4.19) that $[r^{12}(u), \rho^{23}] \in L \otimes V \otimes L$. From Proposition 4.4.2 part (1) this means that $r(u) \in L \otimes M$ if $u \in U$ and $u+\gamma \in U$. From Example 3.2.9 part (2) we see that $[\rho^{12}, r^{23}(v)] \in V \otimes L \otimes L$, hence from Equation (4.20) we have $[\rho^{12}, r^{13}(v+\gamma)] \in V \otimes L \otimes L$. Using Proposition 4.4.2 part (2) we have $r(v+\gamma) \in M \otimes L$ if $v \in U$ and $v+\gamma \in U$.

For some small disc $U' \subset U$ centred at zero we must have $u', u'+\gamma$ and $u'-\gamma \in U$, therefore, for all $u \in U'$ we have $r(u) \in L \otimes M \cap M \otimes L$. From Remark 4.4.3 we have $r(u) \in M \otimes M$. By the assumptions of (A) we must have $M = L$, that is, $[L, V] \subset V$. This implies that V is an ideal in L. But L is simple, and so $V = \mathsf{im}(\varphi_1(\rho)) = L$ (that is, $\varphi_1(\rho)$ is surjective, hence ρ is nondegenerate). From Lemma 3.2.23 we conclude that $[\rho^{12}, \rho^{13}]$ cannot be equal to zero.

It follows from this that the function r has, for $u = 0$, a pole of order not less than k: otherwise, multiplying Equation (4.20) by v^k and letting v tend to 0, we would have $[\rho^{12}, \rho^{13}] = 0$. It remains to prove that the order of the pole of r for $u = 0$ does not exceed one and $\lim_{u\to 0} u \cdot r(u) = \lambda t$. Setting $\gamma = 0$, let $\theta = \rho$, that is $\theta = \lim_{u \to 0} u^\ell r(u)$ with ℓ being the order of the pole at zero, therefore from what we have shown above, $[\theta^{12}, \theta^{13}] \neq 0$. Suppose

$$r(u) = \frac{\theta}{u^\ell} + \frac{A}{u^{\ell-1}} + \sum_{i=2-\ell}^{\infty} X_i u^i, \quad \theta \neq 0.$$

If $\ell \geq 1$, then fixing v close to zero such that $r(u)$ has no pole at $u = v$ we find

$$r(u+v) = \sum_{i\geq 0} f_i(v) u^i$$

(or $r(u) = \sum_{i\geq 0} f_i(v)(u-v)^i$ by Taylors Theorem). With $f_0(v) = r(v)$, and $f_1(v) = \left.\frac{dr(u)}{du}\right|_{u=v}$. Now fixing v and treating u as a variable in Equation (4.5) we wish to compare the coefficient of $u^{\ell-1}$ in the Laurent series at the point $u = 0$ of the left side of Equation (4.5). To obtain this we need

$$\begin{aligned}\Big[r^{12}(u), r^{13}(u+v)\Big] + \Big[r^{13}(u+v), r^{23}(v)\Big] + \Big[r^{12}(u), r^{23}(v)\Big] &= \\ \Big[\frac{1}{u^\ell}\theta^{12} + \frac{1}{u^{\ell-1}}A^{12} + \cdots, f_0(v)^{13} + f_1(v)^{13}u + \cdots\Big] & \\ + \Big[f_0(v)^{13} + f_1(v)^{13}u + \cdots, f_0^{23}(v)\Big] & \\ + \Big[\frac{1}{u^\ell}\theta^{12} + \frac{1}{u^{\ell-1}}A^{12} + \cdots, f_0^{23}(v)\Big]&.\end{aligned}$$

The expression on the right-hand side of this equation can be written as

$$\frac{1}{u^\ell}\Big[\theta^{12}, f_0(v)^{13} + f_0(v)^{23}\Big] + \frac{1}{u^{\ell-1}}\Big(\Big[A^{12}, f_0(v)^{13} + f_0(v)^{23}\Big] + \Big[\theta^{12}, f_1(v)^{13}\Big]\Big) + \cdots$$

If $\ell > 1$ we have $\ell - 1 \geq 1$, therefore,

$$\big[A^{12}, f_0(v)^{13} + f_0(v)^{23}\big] + \big[\theta^{12}, f_1(v)^{13}\big] = 0. \tag{4.21}$$

Multiplying Equation (4.21) by $v^{\ell+1}$ and taking the limit of this new equation as $v \to 0$ we obtain with

$$\begin{aligned}f_0(v) &= r(v) = \frac{1}{v^\ell}\theta + \frac{1}{v^{\ell-1}}A + \cdots \\ f_1(v) &= \frac{dr(u)}{du}\Big|_{u=v} = \frac{-\ell}{v^{\ell+1}}\theta + \frac{-(\ell-1)}{v^\ell}A + \cdots\end{aligned}$$

that

$$\lim_{v\to 0} v^{\ell+1} f_0(v) = \lim_{v\to 0}(v\theta + v^2 A + \ldots) = 0$$

and

$$\lim_{v\to 0} v^{\ell+1} f_1(v) = \lim_{v\to 0}(-\ell\theta - (\ell-1)vA + \ldots) = -\ell\theta$$

that is, $[\theta^{12}, \theta^{13}] = 0$, which contradicts what we have proved in the above paragraph. Thus, $\ell = 1$. As we have shown, $\ell \geq k$, this implies that $k = 1$ also.

We now need to show that θ from the previous paragraph is a multiple of t. From the conditions of part (A) of the theorem we know that there does not exist a disc $U' \subset U$ around zero such that there is a Lie subalgebra $L_1 \subset L$ with $r(u) \in L_1 \otimes L_1$ for all $u \in U'$ at which r is holomorphic. From Equations (4.19) and (4.20) with $\gamma = 0$ we have, for all $u \in U$,

$$[r^{12}(u) + r^{13}(u), \theta^{23}] = 0 \tag{4.22}$$

$$[\theta^{12}, r^{13}(u) + r^{23}(u)] = 0. \tag{4.23}$$

We define the cyclic permutation operator τ_{231} by $\tau_{231}(a \otimes b \otimes c) = b \otimes c \otimes a$. We obtain from Equation (4.22)

$$\tau_{231}\big([r^{12}(u) + r^{13}(u), \theta^{23}]\big) = [(\tau r)^{13}(u) + (\tau r)^{23}(u), \theta^{12}] = 0. \tag{4.24}$$

(Note that this is a version of Equation (4.23) with r replaced by $\tau(r)$). Now, let $L_1 := \{z \in L \,|\, [\mathsf{ad}(z), \varphi_1(\theta)] = 0\}$, that is, $z \in L_1$ if and only if for all $x \in L$ we have $[z, \varphi_1(\theta)(x)] - \varphi_1(\theta)\big([z, x]\big) = 0$. From Proposition 3.2.12 and Remark 3.2.15 we obtain from Equation (4.23), for all $x, y \in L$ where $A = r(u)$:

$$\begin{aligned}\varphi_2\big([\theta^{12}, A^{13} + A^{23}]\big)(x \otimes y) = {} & [\varphi_1(\theta)(x), \varphi_1(A)(y)] \\ & - \varphi_1(\theta)\big([x, \varphi_1(A)(y)]\big) = 0.\end{aligned}$$

Hence, $\varphi_1(A)(y) \in L_1$ for all $y \in L$. This means that $\mathsf{im}(\varphi_1(A)) \subset L_1$ and hence $A = r(u) \in L_1 \otimes L$. From Equation (4.24) we obtain $(\tau(r))(u) \in L_1 \otimes L$, or $r(u) \in L \otimes L_1$. Hence, $r(u) \in L_1 \otimes L_1$ for all $u \in U$. By assumption this implies that $L_1 = L$, hence, $[\mathsf{ad}(z), \varphi_1(\theta)] = 0$ for all $z \in L$, that is, $\varphi_1(\theta) \in \mathbb{C}\mathbb{1}_L$. From Lemma 1.3.16 it follows that θ is proportional to t.

(B) $\Rightarrow$ (C) (C) follows trivially from (B).

(C) $\Rightarrow$ (D) Let us assume that r has a simple pole at $u = 0$ with residue λt, $\lambda \in \mathbb{C}$. This means that $\lim_{u\to 0} u \cdot r(u) = \lambda t$ and $\lambda \neq 0$. As φ_1 is linear, $\lambda \mathbb{1}_L = \varphi_1(\lambda t) = \lim_{u \to 0} u \cdot \varphi_1(r(u))$. Hence, for nonzero λ, $\det(\lambda \mathbb{1}_L) \neq 0$ and so $\det(\lambda \mathbb{1}_L) = \lim_{u\to 0} \det\big(u \cdot \varphi_1(r(u))\big)$. Moreover, as r is continuous, for $u \neq 0$ sufficiently small, $\det\big(\varphi_1(r(u))\big) \neq 0$. Therefore, from Remark 3.2.22, r is nondegenerate.

(D) $\Rightarrow$ (A) In order to show this we will prove that there does not exist a nondegenerate solution r to Equation (4.5) which is holomorphic in U. First let us assume that r is holomorphic in U and that there exists a $u_0 \in U$ such that $r(u_0)$ is nondegenerate. Then from Proposition

4.4.1 we know that the tensor $r(0)$ is also nondegenerate. It is clear that if r is a solution of Equation (4.5), holomorphic for $u = 0$, then the $r(0)$, a constant not depending on u, satisfies the relation (4.6). From Corollary 4.3.5 $r(0)$ is degenerate and so for nondegenerate r, r must have at least one pole.

In order to prove the second part of condition (A) we will assume that there exists a subalgebra $L_1 \subset L$ and an open set $U' \subset U$ containing the origin such that $r(u) \in L_1 \otimes L_1$ for all $u \in U'$ at which r is holomorphic. Then $\mathsf{im}\big(\varphi_1(r(u))\big) \subset L_1$ for all $u \in U'$. If $L_1 \neq L$ then $r(u)$ is degenerate on U' as this would mean that $\varphi_1(r(u))$ is not an isomorphism. But as r is nondegenerate, by Remark 3.2.22 $\det\big(\varphi_1(r(u))\big)$ is not identically zero on U, and in particular, $\det\big(\varphi_1(r(u))\big)$ is not identically zero on U'. Therefore we must have $L_1 = L$.

□

4.5 Properties of Nondegenerate Solutions

We will now focus on some general properties of nondegenerate solutions to the CYBE. We show that these functions are not only meromorphic on a disc contained in $\mathbb{C}$, but are meromorphic on the entire complex plane. We will also look at the set of poles, Γ, of the function r. Throughout this section we will assume that $\lim_{u\to 0} u \cdot r(u) = t$.

Proposition 4.5.1. *Let r be a nondegenerate solution to the CYBE, meromorphic on a disc $u \subset \mathbb{C}$. Then $r(u+v)$ is a rational function of $r(u)$ and $r(v)$ for all $u, v, u+v \in U$.*

Proof. We know that

$$[r^{12}(u), r^{13}(u+v)] + [r^{12}(u), r^{23}(v)] + [r^{13}(u+v), r^{23}(v)] = 0.$$

Let $\{I_\mu\}$ be a basis of L and using the identities $r(u) = \sum_{\mu,\nu} r_{\mu\nu}(u) I_\mu \otimes I_\nu$ and $[I_r, I_s] = \sum_\lambda a^\lambda_{rs} I_\lambda$ we can see that Equation (4.5) holds if and only if

$$\sum_{\alpha,\beta} r_{\alpha k}(u) r_{\beta \ell}(u+v) a^j_{\alpha\beta} + \sum_{\mu,\nu} r_{j\mu}(u) r_{\nu\ell}(v) a^k_{\mu\nu} + \\ \sum_{\sigma,\rho} r_{j\sigma}(u+v) r_{k\rho}(v) a^\ell_{\sigma\rho} = 0 \quad (4.25)$$

for all $j, k, \ell \in \{1, \ldots, n\}$. If $r(u)$ and $r(v)$ are known, we let $\sum_{\mu,\nu} r_{j\mu}(u) r_{\nu\ell}(v) a^k_{\mu\nu} = -D$ where D depends on $j, k, \ell \in \{1, \ldots, n\}$, and we can write Equation (4.25) as

$$\sum_{\alpha,\beta} D^{jk\ell}_{\beta\rho} r_{\beta\rho}(u+v) = D$$

for all $j, k, \ell \in \{1, \ldots, n\}$ where D and $D_{\alpha\beta}$ depend linearly on all the $r_{\alpha k}(u)$ and $r_{k\rho}(v)$. That is, Equation (4.5) is an inhomogeneous system of linear equations in the elements of the tensor $r(u+v)$ with n^3 equations and n^2 unknowns. A solution to such a system is expressed as a rational function of the coefficients if the associated homogeneous system in nondegenerate, that is, it has only the trivial solution. In our case, the homogeneous system is

$$[r^{12}(u) - r^{23}(v), Z^{13}] = 0. \tag{4.26}$$

If $u \neq 0$ is sufficiently small, then for $u = v$ Equation (4.26) is equivalent to

$$[u \cdot r^{12}(u) - u \cdot r^{23}(u), Z^{13}] = 0. \tag{4.27}$$

Taking the limit as u approaches zero Equation (4.27) takes the form

$$[t^{12} - t^{23}, Z^{13}] = 0. \tag{4.28}$$

We need to show that this equation is only true when $Z = 0$. It will then follow that for small u, Equation (4.27) is true only for $Z = 0$.

Applying φ_2 to Equation (4.28) we have, for all $x, y \in L$

$$\big[\varphi_1(t)(x), \varphi_1(Z)(y)\big] + \varphi_1(Z)\big[\varphi_1(t)^*(x), y\big] = 0.$$

Because $\varphi_1(t) = \varphi_1(t)^* = \mathbb{1}_L$ we have,

$$\big[x, \varphi_1(Z)(y)\big] + \varphi_1(Z)\big[x, y\big] = 0 \tag{4.29}$$

that is,

$$\big(\mathsf{ad}(x) \circ \varphi_1(Z)\big)(y) + \big(\varphi_1(Z) \circ \mathsf{ad}(x)\big)(y) = 0.$$

Because $-\mathsf{ad}(x) = \mathsf{ad}(x)^*$ we can write this as

$$\big(\mathsf{ad}(x) \circ \varphi_1(Z)\big)(y) - \big(\varphi_1(Z) \circ \mathsf{ad}(x)^*\big)(y) = 0.$$

Using Lemma 3.2.4 we can write this as

$$\varphi_1\big((\mathsf{ad}(x) \otimes \mathbb{1}_L - \mathbb{1}_L \otimes \mathsf{ad}(x))(Z)\big) = 0$$

or,

$$\varphi_1\big([x\otimes 1-1\otimes x, Z]\big)=0 \tag{4.30}$$

for all $x\in L$. Direct computation gives $\big[[x,y]\otimes 1+1\otimes[x,y],Z\big]=\big[[x\otimes 1-1\otimes x, y\otimes 1-1\otimes y],Z\big]$, and from Equation (4.30) and the Jacobi identity it follows that

$$\big[[x\otimes 1-1\otimes x, y\otimes 1-1\otimes y],Z\big]=0 \tag{4.31}$$

for any $x,y\in L$. Since elements of the form $[x,y]$, because of the simplicity of L, generate L as a vector space, it follows from Equation (4.31) that $[x\otimes 1+1\otimes x, Z]=0$ for all $x\in L$. (From Equation (4.30) we see that $\varphi_1([x\otimes 1,Z])=\varphi_1([1\otimes x,Z])$ so we have $[x\otimes 1,Z])=0$, that is, $\mathsf{ad}(x)\circ\varphi_1(Z)=0$). From Proposition 3.2.25 this means that $Z=\lambda t$, $\lambda\in\mathbb{C}$ From Equation (4.29) we can now write $[x,\lambda\cdot y]+\lambda[x,y]=0$ or $2\lambda[x,y]=0$. As L is simple, $\lambda=0$ and consequently $Z=0$. Therefore, $r(u)$ is a solution to the CYBE if and only if the equation $r(u+v)=R(r(u),r(v))$ holds for some rational function R and for all $(u,v)\in\widetilde{U}=\{(u,v)\in\mathbb{C}^2\,|,\ u\in U, v\in U, u+v\in U\}\subset\mathbb{C}^2$. This means that the meromorphic function $F(u,v)=r(u+v)-R(r(u),r(v))$, which is defined on an open subset $V\subset\mathbb{C}^2$, vanishes identically on $\tilde{U}$. The Identity Theorem, (Remark 3.1.9), implies that $F\equiv 0$ on V. □

Corollary 4.5.2. *$r(u-v)$ is a rational function of $r(u)$ and $r(v)$.*

Proof. We know from Theorem 4.5.1 that $r(u+v)=R(r(u),r(v))$ and as r is unitary we have $r(u-v)=R(r(u),-\tau(r(v)))$. But the map $-\tau: L\otimes L\to L\otimes L$ is linear, so $r(u-v)$ is also a rational function of $r(u)$ and $r(v)$. □

Corollary 4.5.3. *Let r be a nondegenerate solution to the CYBE defined on a disc U with centre zero. Then r extends meromorphically to the whole complex plane $\mathbb{C}$.*

Proof. We know that r is defined on a disc $U=D(0,\rho)\subset\mathbb{C}$. If r does not extend mermorphically to $\mathbb{C}$, there exists $\epsilon>0$ and $\rho_1\geq\rho>\epsilon$ such that r extends to $D(0,\rho_1)$ but not to $D(0,\rho_1+\varepsilon)$. If we let $Y(u,v)=[r^{12}(u),r^{13}(u+v)]+[r^{12}(u),r^{23}(v)]+[r^{13}(u+v),r^{23}(v)]$, then $Y(u,v)\equiv 0$ on $D(0,\rho)\subset D(0,\rho_1)$, therefore, from the Identity Theorem, $Y(u,v)\equiv 0$ on $\tilde{V}(\rho_1)$ and so r satisfies the CYBE in $D(0,\rho_1)$. By Proposition 4.5.1 there is a rational function R such that $r(u+v)=R(r(u),r(v))$ for all $u,v\in D(0,\rho_1)$ with $u+v\in D(0,\rho_1)$.

We now fix $v \in D(0,\varepsilon) \subset D(0,\rho_1)$ and define a meromorphic function f on $D(v,\rho_1)$ by $f(u+v) = R(r(u), r(v))$. That is, $f(u) = R(r(u-v), r(v))$ for all $u \in D(v,\rho_1)$. This means that for all $u \in D(v,\rho_1) \cap D(0,\rho_1)$, we have $f(u) = r(u)$, that is, $f(u)$ extends $r(u)$ to $D(v,\rho_1) \cup D(0,\rho_1)$. If we vary v we can extend $r(u)$ to $\bigcup D(v,\rho_1) = D(0,\rho_1+\varepsilon)$ which contradicts our assumption. Hence, $r(u)$ extends meromorphically to $\mathbb{C}$. $\square$

Proposition 4.5.4. *Let r be a nondegenerate solution to the CYBE, meromorphic on $\mathbb{C}$. Let Γ denote the set of poles of r and let $\gamma \in \Gamma$. Then there exists an $A_\gamma \in \mathsf{Aut}(L)$ such that*

$$r(u+\gamma) = (A_\gamma \otimes \mathbb{1}_L) r(u) \tag{4.32}$$

for all $u \in \mathbb{C}$.

Proof. Again, set $\rho = \lim_{u \to \gamma}(u-\gamma) r(u)$. Let $A_\gamma := \varphi_1(\rho) \in \mathsf{End}(L)$. From Lemma 3.2.4 we obtain,

$$(A_\gamma \otimes \mathbb{1}_L)(t) = (\mathbb{1}_L \otimes A_\gamma^*)(t) = \rho.$$

As all the poles are simple, we can multiply Equation (4.5) by $(u-\gamma)$ and take its limit as $u \to \gamma$:

$$[\rho^{12}, r^{13}(v+\gamma)] + [\rho^{12}, r^{23}(v)] = 0 \tag{4.33}$$

for all $v \in \mathbb{C}$. Multiplying by v and taking the limit as $v \to 0$ we find,

$$[\rho^{12}, \rho^{13}] + [\rho^{12}, t^{23}] = 0. \tag{4.34}$$

We have,

$$\begin{aligned}
[\rho^{12}, \rho^{13}] &= \left[\left((\mathbb{1}_L \otimes A_\gamma^*)t\right)^{12}, \left((\mathbb{1}_L \otimes A_\gamma^*)t\right)^{13} \right] \\
&= \left[(\mathbb{1}_L \otimes A_\gamma^* \otimes \mathbb{1}_L)t^{12}, (\mathbb{1}_L \otimes \mathbb{1}_L \otimes A_\gamma^*)t^{13} \right] \\
&= (\mathbb{1}_L \otimes A_\gamma^* \otimes A_\gamma^*)[t^{12}, t^{13}]
\end{aligned}$$

and also,

$$\begin{aligned}
[\rho^{12}, t^{23}] &= \left[\left((A_\gamma \otimes \mathbb{1}_L)t\right)^{12}, t^{23} \right] \\
&= (A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)[t^{12}, t^{23}] \\
&= -(A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)[t^{12}, t^{13}].
\end{aligned}$$

Therefore, Equation (4.34) is equivalent to

$$(A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)[t^{12}, t^{13}] = (\mathbb{1}_L \otimes A_\gamma^* \otimes A_\gamma^*)[t^{12}, t^{13}].$$

Applying φ_2 to this equation and using Lemma 3.2.5 we find,

$$A_\gamma \circ \varphi_1([t^{12}, t^{13}])(x \otimes y) = \varphi_2([t^{12}, t^{13}]) \circ (A_\gamma(x) \otimes A_\gamma(y)). \tag{4.35}$$

From Proposition 3.2.14 we know that $\varphi_2\big([t^{12}, t^{13}]\big) = \mathcal{L}$, the Lie bracket, so Equation (4.35) can be written

$$A_\gamma[x, y] = [A_\gamma(x), A_\gamma(y)]$$

that is, $A_\gamma : L \to L$ is a Lie algebra homomorphism, and $A_\gamma \neq 0$ as $\rho \neq 0$ (γ is a pole of order 1). Because L is simple and $\ker(A_\gamma) \subset L$ is an ideal, A_γ is an automorphism.

From Equation (4.33) we have $[\rho^{12}, r^{13}(v + \gamma)] = -[\rho^{12}, r^{23}(v)]$ for all $v \in \mathbb{C}$. But $\rho^{12} = (A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)t^{12}$ so we can write, $[\rho^{12}, r^{13}(v + \gamma)] = -(A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)[t^{12}, r^{23}(v)]$. Using the identity $[t^{12}, r^{13}(v)] = -[t^{12}, r^{23}(v)]$ (Corollary 3.2.17) we obtain $[(A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)t^{12}, r^{13}(v + \gamma)] - (A_\gamma \otimes \mathbb{1}_L \otimes \mathbb{1}_L)[t^{12}, r^{13}(v)] = 0$. Applying $(A_\gamma^{-1} \otimes \mathbb{1}_L \otimes \mathbb{1}_L)$ to this equation and using the fact that A_γ is a Lie algebra homomorphism we obtain

$$[t^{12}, \big((A_\gamma^{-1} \otimes \mathbb{1}_L)r(v + \gamma)\big)^{13}] - [t^{12}, r^{13}(v)] = 0$$

or,

$$\big[t^{12}, \big(r(v) - (A_\gamma^{-1} \otimes \mathbb{1}_L)r(v + \gamma)\big)^{13}\big] = 0. \tag{4.36}$$

From Remark 3.2.19 we know that this implies

$$r(v) = (A_\gamma^{-1} \otimes \mathbb{1}_L)r(v + \gamma)$$

that is,

$$r(v + \gamma) = (A_\gamma \otimes \mathbb{1}_L)r(v).$$

□

Proposition 4.5.5. *Let r be a nondegenerate solution of the CYBE meromorphic on $\mathbb{C}$. Then r satisfies the unitary condition $r(u) = -\tau(r(-u))$.*

Proof. We have, for all $u_1, u_2, u_3 \in \mathbb{C}$:

$$[r^{12}(u_1 - u_2), r^{13}(u_1 - u_3)] + [r^{12}(u_1 - u_2), r^{23}(u_2 - u_3)] + [r^{13}(u_1 - u_3), r^{23}(u_2 - u_3)] = 0. \tag{4.37}$$

Applying the map τ_{213} defined earlier and interchanging u_1 and u_2 we get

$$[(\tau(r))^{12}(u_2-u_1), r^{23}(u_2-u_3)] + [(\tau(r))^{12}(u_2-u_1), r^{13}(u_1-u_3)] \\ + [r^{23}(u_2-u_3), r^{13}(u_1-u_3)] = 0 \quad (4.38)$$

adding Equations (4.37) and (4.38) we obtain

$$[r^{12}(u_1-u_2), r^{13}(u_1-u_3)] + [r^{12}(u_1-u_2), r^{23}(u_2-u_3)] \\ + [r^{13}(u_1-u_3), r^{23}(u_2-u_3)] + [(\tau(r(u_2-u_1))^{12}, r^{23}(u_2-u_3)] \\ + [\tau(r(u_2-u_1))^{12}, r^{13}(u_1-u_3)] + [r^{23}(u_2-u_3), r^{13}(u_1-u_3)] = 0.$$

Simplifying we obtain

$$[r^{12}(u_1-u_2) + \tau(r(u_2-u_1))^{12}, r^{13}(u_1-u_3) + r^{23}(u_2-u_3)] = 0.$$

We now fix u_1 and u_2, multiply by (u_2-u_3) and take the limit as u_3 approaches u_2,

$$[r^{12}(u_1-u_2) + \tau(r(u_2-u_1))^{12}, t^{23}] = 0.$$

Now, letting $X = r(u_1-u_2) + \tau(r(u_2-u_1))$ we have $[X^{12}, t^{23}] = 0$. From Remark 3.2.19 this means that $X = 0$. Thus, we have proved that

$$r(u_1-u_2) + \tau(r(u_2-u_1)) = 0 \quad (4.39)$$

that is,

$$r(u) = -\tau(r(-u)) \quad (4.40)$$

for all $u \in \mathbb{C}$. □

Proposition 4.5.6. *Let r be a nondegenerate solution of the CYBE meromorphic in $\mathbb{C}$. Let Γ denote the set of poles of r with $\gamma \in \Gamma$ and let $A_\gamma \in \mathsf{Aut}(L)$ such that Equation (4.32) holds. Then,*

1. *Γ is a discrete subgroup relative to the addition of $\mathbb{C}$*
2. *$A_{\gamma_1+\gamma_2} = A_{\gamma_1} \circ A_{\gamma_2}$ for any $\gamma_1, \gamma_2 \in \Gamma$,*
3. *$A_\gamma^* = A_{-\gamma} = A_\gamma^{-1}$,*
4. *$r(u+\gamma) = (1 \otimes A_\gamma^{-1})r(u)$,*
5. *$(A_\gamma \otimes A_\gamma)r(u) = r(u)$.*

Proof.

1. In order to show that Γ is a discrete subgroup, we need only show that it is an additive subgroup as poles of a meromorphic function are discrete. Let $\gamma, \gamma' \in \Gamma$. The function r has a pole at $u = \gamma'$, hence $(A_\gamma \otimes 1)r(u)$ has a pole at $u = \gamma'$. From Equation (4.32) it follows that $r(u+\gamma)$ has a pole at $u = \gamma'$. That is, $\gamma + \gamma' \in \Gamma$. Since r satisfies the unitary condition, we have $-\gamma \in \Gamma$, and thus Γ is an additive subgroup.

2. From Theorem 4.5.4 we have $(A_\gamma \otimes 1)r(u) = r(u+\gamma)$ for all $\gamma \in \Gamma$. This means in particular, $(A_{\gamma_1} \otimes 1)r(u) = r(u+\gamma_1)$. Hence,

$$\begin{aligned}\lim_{u\to\gamma_2}(A_{\gamma_1}\otimes 1)(u-\gamma_2)r(u) &= \lim_{u\to\gamma_2}(u-\gamma_2)r(u+\gamma_1)\\ &= \lim_{u+\gamma_1\to\gamma_1+\gamma_2}(u+\gamma_1-(\gamma_1+\gamma_2))r(u+\gamma_1).\end{aligned}$$

   That is,

$$(A_{\gamma_1}\otimes 1)\rho_{\gamma_2} = \rho_{\gamma_1+\gamma_2}. \tag{4.41}$$

   If we let $\gamma_1 = \gamma$ and $\gamma_2 = 0$ in Equation (4.41) we find $\rho_\gamma = (A_\gamma \otimes 1)\rho_0$ and as $\rho_0 = t$, $\rho_\gamma = (A_\gamma \otimes 1)t$. Therefore, we can write Equation (4.41) as

$$\begin{aligned}(A_{\gamma_1}\otimes 1)\circ(A_{\gamma_2}\otimes 1)t &= (A_{\gamma_1+\gamma_2}\otimes 1)t\\ \big((A_{\gamma_1}\circ A_{\gamma_2})\otimes 1\big)t &= (A_{\gamma_1+\gamma_2}\otimes 1)t.\end{aligned}$$

   Applying φ_1 to both sides of this equation and using Proposition 3.2.4 we obtain $A_{\gamma_1}\circ A_{\gamma_2} = A_{\gamma_1+\gamma_2}$ as desired.

3. As before, we will let $\rho_\gamma = \lim_{u\to\gamma}(u-\gamma)r(u)$. We then have

$$\tau(\rho_\gamma) = \lim_{u\to\gamma}(u-\gamma)\tau(r(u)).$$

   From Proposition 4.5.5 we know that r is unitary therefore,

$$\tau(\rho_\gamma) = -\lim_{u\to\gamma}(u-\gamma)r(-u)$$

   letting $v = -u$ we have,

$$\tau(\rho_\gamma) = \lim_{v\to-\gamma}(v+\gamma)r(v) = \rho_{-\gamma}.$$

   Hence, $A^*_\gamma = \varphi_1(\tau(\rho_\gamma)) = \varphi_1(\rho_{-\gamma}) = A_{-\gamma}$. Now, from $\rho_0 = \lim_{u\to 0} u\cdot r(u) = t$ and the definition of A_γ we have $A_0 = \varphi_1(\rho_0) = \varphi_1(t) = \mathbb{1}_L$. From part (1) above we have $A_\gamma \circ A_{-\gamma} = \mathbb{1}_L$. That is, $A_{-\gamma} = A^{-1}_\gamma$ for all $\gamma \in \Gamma$.

4. Using Equation (4.32) with $u := -v - \gamma$ we obtain $r(-v) = (A_\gamma \otimes \mathbb{1}_L)(r(-v-\gamma))$, that is $-r(-v) = (A_\gamma \otimes \mathbb{1}_L)(-r(-(v+\gamma)))$. Because r is unitary, we can write this as $\tau(r(v)) = (A_\gamma \otimes \mathbb{1}_L)\tau(r(v+\gamma))$ or $\tau\big((1 \otimes A_\gamma)r(v+\gamma)\big) = \tau(r(u))$ which gives us $(1 \otimes A_\gamma)r(u+\gamma) = r(u)$ or, $r(u+\gamma) = (1 \otimes A_\gamma^{-1})r(u)$.

5. $(A_\gamma \otimes A_\gamma)r(u) = (1 \otimes A_\gamma) \circ (A_\gamma \otimes \mathbb{1}_L)r(u) = (1 \otimes A_\gamma)r(u+\gamma) = r(u)$.

□

Chapter 5

Rational, Trigonometric and Elliptic Solutions to the CYBE

In 1982, A.A. Belavin and V.G. Drinfeld suceeded in catagorising solutions r to the CYBE meromorphic in a neighbourhood U of the origin and possessing an additional nondegeneracy property [4]. They found that, up to equivalence, these solutions fell into one of three types: elliptic, rational, or trigonometric, depending on the rank of the set of poles Γ. In this chapter, we reformulate their proof of this theorem, and then briefly look at each solution type. Throughout this chapter we assume that r is a nondegenerate solution to the CYBE, meromorphic on $\mathbb{C}$, and that $t = \lim_{u\to 0} u \cdot r(u)$.

5.1 Elliptic Solutions

Recall from the complex analysis preliminaries the definition of an elliptic function (Definition 3.1.14). We wish to show that if Γ has rank 2, then r is an elliptic function. First we need the following Lemma:

Lemma 5.1.1. *Let $H \subset \mathsf{Aut}(L)$ be an infinite abelian subgroup. Then there exists a nonzero $a \in L$ such that $h(a) = a$ for any $h \in H$.*

Proof. For a proof see [6] Theorem 9.1. □

Proposition 5.1.2. *Let Γ have rank 2. Then*

1. *There is no nonzero $a \in L$ such that $A_\gamma(a) = a$ for any $\gamma \in \Gamma$,*

2. *there is an n such that $A_\gamma^n = \mathbb{1}_L$ for $\gamma \in \Gamma$.*

Proof. 1. Assume that $a \in L$, $a \neq 0$, $A_\gamma(a) = a$ for $\gamma \in \Gamma$. Define the linear map $\phi : \mathbb{C} \to L$ by $\phi(u) = \varphi_1\big(\tau(r(u))\big)(a)$. Then $\phi(u+\gamma) = \varphi_1\big(\tau(r(u+\gamma))\big)(a)$. However, from Proposition 4.5.4 we have $r(u+\gamma) = (A_\gamma \otimes \mathbb{1}_L)r(u)$ so $\tau(r(u+\gamma)) = (\mathbb{1}_L \otimes A_\gamma)\tau(r(u))$. Using Proposition 3.2.4 we can write this as $\varphi_1\big(\tau(r(u+\gamma))\big)(a) = \varphi_1\big(\tau(r(u))\big) \circ A_\gamma^*$. From Proposition 4.5.6 part (3) we know that $A_\gamma^* = A_\gamma^{-1}$ and as $A_\gamma(a) = a$ we obtain $A_\gamma^{-1}(a) = a$. Hence, $\varphi_1\big(\tau(r(u+\gamma))\big)(a) = \varphi_1\big(\tau(r(u))\big)(a)$, that is, the function ϕ is periodic for shifts by $\gamma \in \Gamma$. Because $r(u)$ has a simple pole at $u = 0$, then $\phi(u)$ also has a simple pole at $u = 0$ and has no other poles in the period parallelogram $\alpha + \Pi$. By the Residue Theorem, Theorem 3.1.13, the sum of the residues of $\phi(u) = \frac{1}{2\pi i}\int_C \phi(u)du$ which is equal to $\mathsf{Res}(\phi, 0)$. But the integrals over opposite sides of $\alpha + \Pi$ cancel (since ϕ is the same at congruent points and du introduces a minus sign on one of the two matching sides), and so the integral is zero. This contradicts our assumption and so proves assertion (1), there does not exist an $a \in L$ such that $A_\gamma(a) = a$ for any $\gamma \in \Gamma$.

2. Let H be the abelian subgroup generated by A_γ, $H = \{A_\gamma \,|\, \gamma \in \Gamma\}$. From part (1) we know that there does not exist a nonzero $a \in L$ such that $A_\gamma(a) = a$ for any $\gamma \in \Gamma$. Hence, from Lemma 5.1.1, H is a finite group and so must have finite order. This implies that there exists an integer n such that $A_\gamma^n = \mathbb{1}_L$.

□

Corollary 5.1.3. *If the rank of Γ is 2, then r is an elliptic function.*

Proof. From Proposition 4.5.4 we know that $(A_\gamma \otimes \mathbb{1}_L)(r(u)) = r(u+\gamma)$ for all $u \in \mathbb{C}$ and for all $\gamma \in \Gamma$. Then $(A_\gamma^n \otimes \mathbb{1}_L)(r(u)) = r(u+n\gamma)$. For $\gamma_1, \gamma_2 \in \Gamma$ we have $A_{\gamma_1}^{n_1} = \mathbb{1}_L$ and $A_{\gamma_2}^{n_2} = \mathbb{1}_L$ so we must have $r(u) = r(u+n_1\gamma_1)$ and $r(u) = r(u+n_2\gamma_2)$ for all $u \in \mathbb{C}$. □

In [4] it was shown that finding the nondegenerate elliptic solutions to the CYBE reduces to describing triples (L, A_1, A_2), where L is a simple Lie algebra, A_1 and A_2 are commuting automorphisms of L of finite order, hot having fixed nonzero vectors.

Proposition 5.1.4. *Let A_1 and A_2 be commuting automorphisms of L, where there does not exist a nonzero $x \in L$ such that $A_1(x) = A_2(x) = x$. Then $L \cong \mathsf{sl}(n)$.*

Proof. For a proof see [6] Theorem 9.3. □

As a consequence of Proposition 5.1.4, elliptic solutions exist only for $L = \mathsf{sl}(n)$. All nondegenerate elliptic solutions can be found in [3].

5.2 Rational and Trigonometric Solutions

In order to give Belavin and Drinfeld's proof of the classification of rational and trigonometric solutions, we need to remind the notion of quasi-Abelian functions, and the addition theorem of Myrberg. We then show that if Γ has rank 0, then r is equivalent to a solution which is a rational function of u, and if Γ has rank 1, then r is equivalent to a solution of the form $f(e^{ku}) \subset L \otimes L$.

Remark 5.2.1. A function of n complex variables is said to be *Abelian* if it is meromorphic in the complex space $\mathbb{C}^n$ and has $2n$ periods that are linearly independent over the field of real numbers

An Abelian function is a generalisation of the concept of an elliptic function of one complex variable, to the case of several complex variables.

Definition 5.2.2. A meromorphic function X on an n-dimensional complex vector space W is called *quasi-Abelian* if there exists a coordinate system $\mathbf{z}_1, \dots, \mathbf{z}_n$ in the space W, integers $a, b, c \geq 0$, $a + b + c = n$ and vectors $\gamma_1, \dots, \gamma_{2c} \in W$ such that

1. for fixed $\mathbf{z}_{a+b+1}, \dots, \mathbf{z}_n$, $X(\mathbf{z}_1, \dots, \mathbf{z}_n)$ is a rational function of $\mathbf{z}_1, \dots, \mathbf{z}_a$, $e^{\mathbf{z}_{a+1}}, \dots, e^{\mathbf{z}_{a+b}}$;
2. the vectors γ_i are periods of X;
3. the vectors $\bar{\gamma}_i \in \mathbb{C}^c$, formed by the last c coordinates of the vectors γ_i, are linearly independent over $\mathbb{R}$.

Remark 5.2.3. The quasiabelian functions include the abelian functions and degenerate forms of them. For $n = 1$, there are three types of quasi-Abelian functions: elliptic $(a = b = 0, c = 1)$; rational $(a = 1, b = c = 0)$ and rational on e^z $(a = c = 0, b = 1)$.

Theorem 5.2.4 (Myrberg's Theorem)**.** *Let $f(u)$ be a meromorphic vector-valued function of one complex variable with values in $\mathbb{C}^m$ satisfying*

$$f(u + v) = P(f(u), f(v))$$
$$f(u - v) = Q(f(u), f(v))$$

where P and Q are rational functions. Then there exists a natural number n, a vector $\mathfrak{a} \in \mathbb{C}^n$ and a quasi-Abelian function $\bar{F}$ of n variables such that

1. *$f(u) = \bar{F}(u\mathfrak{a})$*
2. *the identities*

$$\bar{F}(\mathbf{x}+\mathbf{y}) = P(\bar{F}(\mathbf{x}), \bar{F}(\mathbf{y}))$$
$$\bar{F}(\mathbf{x}-\mathbf{y}) = Q(\bar{F}(\mathbf{x}), \bar{F}(\mathbf{y}))$$

are satisfied.

Proof. For a proof see [18] and [6] Theorem 11.1. □

Myrberg's theorem generalises the theorem of Weierstrass on functions satisfying an algebraic addition theorem to the case of vector-valued functions. The classical Weierstrass theorem asserts that, if a function f is meromorphic over the whole complex plane, and satisfies the functional equation

$$P(f(u), f(v), f(u+v)) = 0$$

where P is a nonzero polynomial, then the function f is either elliptic, rational, or trigonometric.

From Theorem 4.5.1 and Corollary 4.5.2 we can use Myrberg's theorem to show that there exists an $n > 0$, a quasi-Abelian function on n variables $X : \mathbb{C}^n \to L \otimes L$ and a vector $\mathfrak{a} \in \mathbb{C}^n$ such that $r(u) = X(u\mathfrak{a})$. Moreover, we can choose X so as to satisfy the CYBE, that is,

$$[X^{12}(\mathbf{u}), X^{13}(\mathbf{u}+\mathbf{z})] + [X^{12}(\mathbf{u}), X^{23}(\mathbf{z})] + [X^{13}(\mathbf{u}+\mathbf{z}), X^{23}(\mathbf{z})] = 0 \quad (5.1)$$

with $\mathbf{u}, \mathbf{z} \in \mathbb{C}^n$.

Proposition 5.2.5. *Let $X(\mathbf{z}) = Y(\mathbf{z})/f(\mathbf{z})$ for holomorphic functions $Y : U \to L \otimes L$ and $f : U \to \mathbb{C}$. Without loss of generality we can assume that $S := \{\mathbf{z} \in \mathbb{C}^n \mid f(\mathbf{z}) = 0, Y(\mathbf{z}) \neq 0\}$ is not empty. Suppose that $\mathbf{h} \in S$, that is, $\mathbf{h}$ is a pole of X. Then there exists $\Phi(\mathbf{h}) \in \mathsf{Aut}(L)$ and $c(\mathbf{h}) \in \mathbb{C} \setminus \{0\}$ such that*

$$Y(\mathbf{h}) = (c(\mathbf{h})\Phi(\mathbf{h}) \otimes \mathbb{1}_L)(t), \quad (5.2)$$

$$X(\mathbf{z}+\mathbf{h}) = (\Phi(\mathbf{h}) \otimes \mathbb{1}_L)X(\mathbf{z}), \quad \mathbf{z} \in \mathbb{C}^n \quad (5.3)$$

Proof. Let X be chosen so that Equation (5.1) holds. Multiplying this equation by $f(\mathbf{u})$ we have

$$[Y^{12}(\mathbf{u}), X^{13}(\mathbf{u}+\mathbf{z})] + [Y^{12}(\mathbf{u}), X^{23}(\mathbf{z})] + f(\mathbf{u})[X^{13}(\mathbf{u}+\mathbf{z}), X^{23}(\mathbf{z})] = 0$$

and taking this equation to its limit as $\mathbf{u} \to \mathbf{h}$,

$$[Y^{12}(\mathbf{h}), X^{13}(\mathbf{h}+\mathbf{z})] + [Y^{12}(\mathbf{h}), X^{23}(\mathbf{z})] = 0$$

for all $\mathbf{z} \in \mathbb{C}^n$. If we apply the linear map φ_2 to this equation we obtain

$$\big[\varphi_1\big(Y(\mathbf{h})\big)(x), \varphi_1\big(X(\mathbf{h}+\mathbf{z})\big)(y)\big] - \varphi_1\big(Y(\mathbf{h})\big)\big[x, \varphi_1\big(X(\mathbf{z})\big)(y)\big] = 0.$$

For the sake of brevity, we will write $T_{\mathbf{z}}$ for $\varphi_1(X(\mathbf{z}))$ and $T_{\mathbf{h}+\mathbf{z}}$ for $\varphi_1(X(\mathbf{h}+\mathbf{z}))$. Therefore, we can write

$$[\varphi_1(Y(\mathbf{h}))(x), T_{\mathbf{h}+\mathbf{z}}(y)] - \varphi_1(Y(\mathbf{h}))[x, T_{\mathbf{z}}] = 0.$$

As $Y(\mathbf{h}) \in L \otimes L$, we can use Proposition 3.2.11 to obtain

$$\varphi_1\big([Y(\mathbf{h}), T_{\mathbf{z}+\mathbf{h}}(y) \otimes 1]\big)(x) + \varphi_1\big([Y(\mathbf{h}), 1 \otimes T_{\mathbf{z}}(y)]\big)(x) = 0.$$

Because φ_1 is an isomorphism, this implies that

$$[Y(\mathbf{h}), T_{\mathbf{z}+\mathbf{h}} \otimes 1 + 1 \otimes T_{\mathbf{z}}(y)] = 0 \tag{5.4}$$

for all $y \in L$. Let $W = \{\mathbf{z} \in \mathbb{C}^n \,|\, X$ holomorphic at $\mathbf{z}$ and $\mathbf{z}+\mathbf{h}$; $X(\mathbf{z}), X(\mathbf{z} + \mathbf{h})$ are nondegenerate$\}$ and suppose that $\mathbf{z} \in W$. Let $\ell \subset L \oplus L$ be the subgroup generated by $(T_{\mathbf{z}+\mathbf{h}}(y), T_{\mathbf{z}}(y))$, $y \in L$. Then the projections $\pi_{1,2} : \ell \to L$ are surjective. We have $\ker(\pi_2) = \{(x,y) \,|\, (x,y) \in \ell, y = 0\} = \{(x,0) \in \ell\} \subset L \oplus 0 = L$. As L is simple, $\ker(\pi_2) = 0$ or L. If $\ker(\pi_2) = L$ we have $\dim \ell = \dim L + \dim L$ as π_2 is surjective. Hence, $\dim \ell = \dim(L \oplus L)$, that is, $\ell = L \oplus L$. By assumption, this is not possible. Therefore, we must have $\ker(\pi_2) = 0$, that is, π_2 is an isomorphism of Lie algebras. Hence there exists a Lie algebra automorphism $\Phi(\mathbf{h},\mathbf{z}) = \pi_1 \circ \pi_2^{-1} : L \to L$, for all $\mathbf{z} \in W$ with $\mathbf{h} \in S$. We know that for $(x,y) \in \ell$, $x = \pi_1(x,y)$ and $(x,y) = \pi_2^{-1}(y)$. Therefore, $\pi_1(\pi_2^{-1}(y)) = x = \Phi(y)$. So $\ell = \{\big(\Phi(\mathbf{h},\mathbf{z})(y), y\big) \,|\, y \in L\}$. From Equation (5.4) we now have $[Y(\mathbf{h}), \Phi(\mathbf{h},\mathbf{z})T_{\mathbf{z}}(y) \otimes 1 + 1 \otimes T_{\mathbf{z}}(y)] = 0$ and from Proposition 3.2.10 we can write this as $[(\Phi(\mathbf{h},\mathbf{z})^{-1} \otimes 1)(Y(h)), T_{\mathbf{z}}(y) \otimes 1 + 1 \otimes T_{\mathbf{z}}(y)] = 0$, with $T_{\mathbf{z}}(y) \in L$. Using Proposition 3.2.25 this implies that $(\Phi(\mathbf{h},\mathbf{z})^{-1} \otimes \mathbb{1}_L)(Y(\mathbf{h})) = c(\mathbf{h},\mathbf{z}) \cdot t$ with $c(\mathbf{h},\mathbf{z}) \in \mathbb{C} \backslash \{0\}$. Hence,

$$Y(\mathbf{h}) = c(\mathbf{h},\mathbf{z})(\Phi(\mathbf{h},\mathbf{z}) \otimes \mathbb{1}_L)(t) \tag{5.5}$$

From Remark 3.2.16, $\varphi_1(Y(\mathbf{h})) = c(\mathbf{h},\mathbf{z})\Phi(\mathbf{h},\mathbf{z})$. As $\Phi(\mathbf{h},\mathbf{z})$ is an isomorphism of Lie algebras, then $\Phi(\mathbf{h},\mathbf{z})[x,y] = [\Phi(\mathbf{h},\mathbf{z})(x),\Phi(\mathbf{h},\mathbf{z})(y)]$. This holds only when $c(\mathbf{h},\mathbf{z}) = 1$, and so $c(\mathbf{h},\mathbf{z})$ and $\Phi(\mathbf{h},\mathbf{z})$ are in fact independent of $\mathbf{z}$ and Equation (5.2) is satisfied.

We know from above that $x = \Phi(y)$ so $T_{\mathbf{z}+\mathbf{h}} = \Phi(\mathbf{h})T_{\mathbf{z}}$ for all $\mathbf{z} \in W$ with $\mathbf{h} \in S$. This holds for $\mathbf{z} \in \mathbb{C}^n$ due to analytic continuation. So, $\varphi_1(X(\mathbf{h}+\mathbf{z})) = \Phi(\mathbf{h}) \circ \varphi_1(X(\mathbf{z})) = \varphi_1((\Phi(\mathbf{h}) \otimes \mathbb{1}_L)(X(\mathbf{z})))$, from which we obtain $X(\mathbf{h}+\mathbf{z}) = (\Phi(\mathbf{h}) \otimes \mathbb{1}_L)X(\mathbf{z})$ for all$\mathbf{z} \in \mathbb{C}^n$ and Equation (5.3) is satisfied. $\square$

Proposition 5.2.6. *There exists an $(n-1)$-dimensional vector subspace $V \subset \mathbb{C}^n$ and a holomorphic homomorphism $\psi : V \to \mathsf{Aut}(L)$ such that for any $\mathbf{z} \in \mathbb{C}^n$, $\mathbf{h} \in V$*

$$X(\mathbf{z}+\mathbf{h}) = (\psi(\mathbf{h}) \otimes \mathbb{1}_L)X(\mathbf{z}) \tag{5.6}$$

$$(\psi(\mathbf{h}) \otimes \psi(\mathbf{h}))X(\mathbf{z}) = X(\mathbf{z}). \tag{5.7}$$

Proof. Again suppose that $X(\mathbf{z}) = Y(\mathbf{z})/f(\mathbf{z})$, where Y and f are holomorphic functions. Let S be the set of poles of X and let J be the subgroup of $\mathbb{C}^n$ generated by S. Because $T_{\mathbf{z}_0+\mathbf{h}} = \Phi(\mathbf{h}) \circ T_{\mathbf{z}_0}$ for $\mathbf{h}_1, \mathbf{h}_2 \in S$, if $\mathbf{h}_1 + \mathbf{h}_2 \in S$, we have $\Phi(\mathbf{h}_1+\mathbf{h}_2) = \Phi(\mathbf{h}_1) \circ \Phi(\mathbf{h}_2)$. We can extend the homomorphism Φ from Proposition 5.2.5 to $\psi : J \to \mathsf{Aut}(L)$ by defining $\Phi(\mathbf{h}_1+\mathbf{h}_2) := \Phi(\mathbf{h}_1) \circ \Phi(\mathbf{h}_2)$ even if $\mathbf{h}_1 + \mathbf{h}_2 \notin S$. We then have $X(\mathbf{z}+\mathbf{h}) = (\psi(\mathbf{h}) \otimes \mathbb{1}_L)X(\mathbf{z})$, that is, $T_{\mathbf{z}_0+\mathbf{h}} = \psi(\mathbf{h}) \circ T_{\mathbf{z}_0}$ and this proves Equation (5.6).

The set of poles of X is invariant under translations by elements of J. Therefore, $J \neq \mathbb{C}^n$. Since S is an analytic subset of codimension 1, J is a union of (at most) countably many parallel affine hyperplanes. In part, it contains a hyperplane V going through zero, hence $V \subset J$. From above we have $\varphi_1(Y(\mathbf{h})) = c(\mathbf{h})\Phi(\mathbf{h})$. Now we prove that ψ is holomorphic at $\mathbf{h}_0 \in V$. Because X is nondegenerate, there exists $\mathbf{z}_0 \in \mathbb{C}^n$ such that X is holomorphic at $\mathbf{z}_0$ and at $\mathbf{z}_0 + \mathbf{h}_0$ and $X(\mathbf{z}_0)$ and $X(\mathbf{z}_0+\mathbf{h})$ are nondegenerate. In particular, $T_{\mathbf{z}_0}$ is an isomorphism. Therefore, $\psi(\mathbf{h}) = T_{\mathbf{z}_0+\mathbf{h}} \circ T_{\mathbf{z}_0}^{-1}$ is holomorphic for $\mathbf{h}$ in a neighbourhood of $\mathbf{h}_0$.

We now need to show that X satisfies the unitarity condition. We can use the same arguments as in the proof for Proposition 4.5.5, however we must be aware that $f(\mathbf{u_2}-\mathbf{u_3}) \cdot X^{23}(\mathbf{u_2}-\mathbf{u_3}) = Y^{23}(\mathbf{u_2}-\mathbf{u_3})$ is a holomorphic function, so $\lim_{\mathbf{u_3}\to\mathbf{u_2}} Y^{23}(\mathbf{u_2}-\mathbf{u_3}) = Y^{23}(0) = t^{23}$.

From Equation (5.6) we know that $X(\mathbf{z}+\mathbf{h}) = (\psi(\mathbf{h}) \otimes \mathbb{1}_L)(X(\mathbf{z}))$. If we apply φ_1 to this equation we find $\varphi_1\big(X(\mathbf{z}+\mathbf{h})\big) = \varphi_1\big((\psi(\mathbf{h}) \otimes \mathbb{1}_L)(X(\mathbf{z}))\big)$. From Proposition 3.2.4 the right hand side of this equation is equal to $\psi(\mathbf{h}) \circ \varphi_1(X(\mathbf{z}))$. We can use Proposition 3.2.4 to obtain $(\psi(\mathbf{h}) \otimes \psi(\mathbf{h}))X(\mathbf{z}) =$

$\psi(\mathbf{h}) \circ \varphi_1(X(\mathbf{z})) \circ \psi(\mathbf{h})^*$. This is equal to $\varphi_1(X(\mathbf{z}+\mathbf{h})) \circ \psi(\mathbf{h})^*$. As X is unitary, this is equal to $-\varphi_1(X(-\mathbf{z}-\mathbf{h}))^* \circ \psi(\mathbf{h})^* =$
$-(\psi(\mathbf{h}) \circ \varphi_1(X(-\mathbf{z}-\mathbf{h}))^*(\varphi_1(X(-\mathbf{z})))^* = \varphi_1(X(\mathbf{z}))$ as required. $\square$

Lemma 5.2.7. *Suppose that $\mathfrak{a} \in \mathbb{C}^n$ such that $r(u) = X(u\mathfrak{a})$. If $\tilde{\mathfrak{a}} - \mathfrak{a} \in V$, then $\tilde{r}(u) := X(u\mathfrak{a})$ is a solution to the CYBE equivalent to $r(u)$.*

Proof. We know that $\tilde{r}(u_1 - u_2) = X(u_1\tilde{\mathfrak{a}} - u_2\tilde{\mathfrak{a}})$. Let $\mathbf{h} = \tilde{\mathfrak{a}} - \mathfrak{a}$, hence, $\tilde{\mathfrak{a}} = \mathbf{h} + \mathfrak{a}$. We can write $\tilde{r}(u_1 - u_2) = X(u_1\mathbf{h} + u_1\mathfrak{a} - u_2\mathbf{h} - u_2\mathfrak{a}) = X((u_1\mathfrak{a} - u_2\mathfrak{a}) + (u_1\mathbf{h} - u_2\mathbf{h}))$. From Proposition 5.2.6 part (1) we can write this as $\tilde{r}(u_1 - u_2) = (\psi(u_1\mathbf{h} - u_2\mathbf{h}) \otimes \mathbb{1}_L)X((u_1 - u_2)\mathfrak{a})$, with ψ having values in $\mathsf{Aut}(L)$, and from part (2) of Proposition 5.2.6 we obtain,

$$\begin{aligned}
\tilde{r}(u_1 - u_2) &= (\psi(u_1\mathbf{h} - u_2\mathbf{h}) \otimes \mathbb{1}_L)(\psi(u_2\mathbf{h}) \otimes \psi(u_2\mathbf{h}))X((u_1 - u_2)\mathfrak{a}) \\
&= \big(\psi(u_1\mathbf{h} - u_2\mathbf{h})\psi(u_2\mathbf{h}) \otimes \psi(u_2\mathbf{h})\big)r(u_1 - u_2) \\
&= \big(\psi(u_1\mathbf{h}) \otimes \psi(u_2\mathbf{h})\big)r(u_1 - u_2).
\end{aligned}$$

From Proposition 4.2.1 this is also a solution to the CYBE. $\square$

Theorem 5.2.8. *If Γ has rank 1, then r is equivalent to a solution $\tilde{r}$ of the form $f(e^{ku})$, $k \in \mathbb{C}$, where f is a rational function. If rank $\Gamma = 0$, then r is equivalent to a rational solution.*

Proof. Suppose that $a, b, c; (\mathbf{z}_1, \ldots, \mathbf{z}_n), \gamma_1, \ldots, \gamma_{2c}$ have the same meaning as in the definition of quasi-Abelianness. We will denote by $\mathbf{e}_1, \ldots, \mathbf{e}_n$ be the basis vectors in $\mathbb{C}^n$ corresponding to the coordinate system $(\mathbf{z}_1, \ldots, \mathbf{z}_n)$, that is $\mathbf{z} = \sum \mathbf{z}_i\, \mathbf{e}_i$. Let $V' = \mathbb{C}\mathbf{e}_1 \oplus \ldots \oplus \mathbb{C}\mathbf{e}_{p+q}$ be a subspace in $\mathbb{C}^n$. We represent γ_i in the form $\delta_i\mathfrak{a} + \mathbf{h}_i$, $\delta_i \in \mathbb{C}$, $\mathbf{h}_i \in V$. As the $\gamma_\mathbf{i}$ are periods of X we have $X(\gamma_i) = X(0)$, that is, $\gamma_i \in \tilde{S}$. As $V \subset H$, with H the subgroup generated by $S \subset \tilde{S}$, for all $\mathbf{h} \in V$, $x \in \Gamma$ we have $\mathbf{h} + x \in \tilde{S}$. Hence, $\gamma_i = \delta_i\mathfrak{a} + \mathbf{h}_i \in \tilde{S}$ if and only if $\delta_i\mathfrak{a} \in \tilde{S}$, and $X(\delta_i\mathfrak{a}) = r(\delta_i)$, so $\delta_i \in \Gamma$.

We assume that the rank of Γ is either 0 or 1. If $V' \subset V$ we would have $\gamma_1, \ldots, \gamma_{2c}$ generating $\mathbb{C}^n/V$ as a vector space over $\mathbb{R}$ and as $\mathbb{C}^n/V \cong \mathbb{C}$ then $\delta_1, \ldots, \delta_{2c}$ would generate $\mathbb{C}$ as a vector space over $\mathbb{R}$. But $\delta_1, \ldots, \delta_{2c} \in \Gamma$ so this is impossible. As V' is not a subset of V, there exists $1 \leq i \leq a + b$ such that $\mathbf{e}_\mathbf{i} \notin V'$. Hence, we can write $\mathfrak{a}$ in the form $k\mathbf{e}_\mathbf{i} + \mathbf{h}$ for some $k \in \mathbb{C}$, $\mathbf{h} \in V$. We set $\tilde{r}(u) = X(uk\mathbf{e}_\mathbf{i})$. By Lemma 5.2.7 $\tilde{r}$ is a solution to the CYBE equivalent to r. Hence, if $i \leq a$ then $\tilde{r} \sim R$ for some rational function R, and if $i > a$, then $\tilde{r}(u) \sim R(e^{ku})$ for some rational function R. The set of poles of $\tilde{r}$ is Γ. Therefore, if the rank of Γ is 1, the function $\tilde{r}$ cannot be rational, while if $\Gamma = \{0\}$, then $\tilde{r}$ cannot have the form $R(e^{ku})$ where R is rational. $\square$

Remark 5.2.9. Rational solutions are equivalent to an r-matrix which is a rational function of u. Trigonometric solutions are equivalent to an r-matrix of the form $f(e^{ku})$, where f is a rational function with values in $L \otimes L$.

The simplest rational solution is shown in the following proposition:

Proposition 5.2.10. *The following r satisfies the CYBE:*

$$r(u) = \frac{t}{u}. \tag{5.8}$$

Proof. If we substitute $r(u)$ from Equation (5.8) into the CYBE we must show that

$$\frac{1}{u(u+v)}[t^{12}, t^{13}] + \frac{1}{uv}[t^{12}, t^{23}] + \frac{1}{(u+v)v}[t^{13}, t^{23}] = 0.$$

Using Corollary 3.2.18 we can write the left-hand side of this equation as

$$\left(\frac{1}{u(u+v)} - \frac{1}{uv} + \frac{1}{(u+v)v}\right)[t^{12}, t^{13}]$$

or,

$$\frac{v-(u+v)+v}{uv(u+v)}[t^{12}, t^{13}] = 0$$

as required. □

This was the first solution ever found, explicitly we can write this for $\mathsf{sl}(2)$ as

$$r_{rat}(z) = \frac{1}{z}\left(\frac{1}{2}h \otimes h + e \otimes f + f \otimes e\right). \tag{5.9}$$

Proposition 5.2.11. *Let $r(u) = \frac{t}{u} + r_0$, with $r_0 \in L \otimes L$. Then r is a solution to the CYBE if and only if r_0 is unitary and satisfies Equation (4.6).*

Proof. If we put $r(u)$ into the CYBE, we obtain

$$\left[\frac{t^{12}}{u} + r_0^{12}, \frac{t^{13}}{u+v} + r_0^{13}\right] + \left[\frac{t^{12}}{u} + r_0^{12}, \frac{t^{23}}{v} + r_0^{23}\right] + \left[\frac{t^{13}}{u+v} + r_0^{13}, \frac{t^{23}}{v} + r_0^{23}\right] = 0. \tag{5.10}$$

From Proposition 5.2.10 the nonzero terms of Equation (5.10) are:

$$\begin{aligned}[]
[r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}] + \\
\frac{1}{u}[t^{12}, r_0^{13}] + \frac{1}{u+v}[r_0^{12}, t^{13}] + \frac{1}{u}[t^{12}, r_0^{23}] + \frac{1}{v}[r_0^{12}, t^{23}] \\
+ \frac{1}{u+v}[t^{13}, r_0^{23}] + \frac{1}{v}[r_0^{13}, t^{23}] = 0.
\end{aligned}$$

Simplifying, we obtain

$$\begin{aligned}[]
[r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}] + \frac{1}{u}[t^{12}, r_0^{13} + r_0^{23}] \\
+ \frac{1}{u+v}[r_0^{12} - r_0^{23}, t^{13}] + \frac{1}{v}[r_0^{12} + r_0^{13}, t^{23}] = 0.
\end{aligned}$$

By Corollary 3.2.17, the fourth and sixth terms are zero and we are left with

$$[r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}] + + \frac{1}{u+v}[r_0^{12} - r_0^{23}, t^{13}] = 0. \tag{5.11}$$

Now, if r_0 satisfies Equation (4.6) the first three terms also vanish. Applying φ_2 to the remaining term we find that it is equal to $[\varphi_1(r_0)(x), \varphi_1(t)(y)] + \varphi_1(t)[\varphi_1(\tau(r_0))(x), y]$. As $\varphi_1(t) = \mathbb{1}_L$ this can be written as $[\varphi_1(r_0)(x), y] + [\varphi_1(\tau(r_0))(x), y] = [\varphi_1(r_0 + \tau(r))(x), y]$. But r_0 is unitary so this terms is also equal to zero. Looking again at Equation (5.11) we must show that if this equation holds, then r_0 must be a unitary solution to Equation (4.6). We obtain from Equation (5.11) that

$$-\frac{1}{u+v}\left[r_0^{12} - r_0^{23}, t^{13}\right] = [r_0^{12}, r_0^{13}] + [r_0^{12}, r_0^{23}] + [r_0^{13}, r_0^{23}].$$

The left-hand-side of this equation depends on u and v but the right side does not. The only way they can be equal then, is if they are both zero. □

In fact, the degree of the polynomial part of a rational solution to the CYBE can be estimated. In 1984 Drinfeld conjectured that any rational solution $r(u, v)$ to the CYBE taking values in L is equivalent to one of the form:

$$r(u, v) = t/(u - v) + g(u, v), \tag{5.12}$$

where g is a polynomial in u, v and $\deg_u g = \deg_v g \leq 1$. This conjecture was proved by Stolin in [25] for $L = \mathsf{sl}(n)$.

Nondegenerate rational solutions were completely classified by Stolin. In [25] he gives all “nontrivial” rational solutions for $\mathsf{sl}(2)$, $\mathsf{sl}(3)$, $\mathsf{sl}(4)$ and several series of examples in the general case. While in [24] he lists all constant solutions to the CYBE for the function with values in $\mathsf{sl}(2)$ and $\mathsf{sl}(3)$ and gives an algorithm which allows one to obtain all constant solutions for L.

Example 5.2.12. *An example of a rational solution in* $\mathsf{sl}(2)$ *given in [19] is:*

$$r_{rat}(z) = \frac{1+z}{4(1-z)} h \otimes h + \frac{f \otimes e + ze \otimes f}{1-z}. \tag{5.13}$$

Example 5.2.13. *Another example of a rational solution in* $\mathsf{sl}(2)$ *given in [19]:*

$$r_{rat}(z) = \frac{1}{z}(\frac{1}{2}h \otimes h + e \otimes f + f \otimes e) + z(f \otimes h + h \otimes f) - z^3 f \otimes f. \tag{5.14}$$

For $\mathsf{sl}(2)$ Stolin gave the following 2 parametric solution:

$$r_{rat}(u,v) = \frac{t}{u-v} - \frac{u}{2} f \otimes h + \frac{v}{2} h \otimes f \tag{5.15}$$

which is gauge equivalent to the above solution, Equation (5.14).

In describing trigonometric solutions important roles are played by the concepts of Coxeter automorphisms and simple weights which we do not cover in the scope of this thesis.

Recall the definitions of roots and the root space decomposition from Section 1.5. The classical trigonometric solutions are obtained as follows. Let
$L = H \oplus \bigoplus_{\alpha \in \phi} L_\alpha$ be the root space decomposition (Equation (1.2)). Choose $x \in L_\alpha$ and $y \in L_{-\alpha}$ so that $\kappa(x,y) = 1$ and let

$$r_0 = \sum_{\alpha > 0} (x \otimes y - y \otimes x).$$

Then

$$r(u) = r_0 - t + \frac{2t}{1-e^u}$$

is a trigonometric solution.

The following are examples of trigonometric solutions in $\mathsf{sl}(2)$;

Example 5.2.14. *(given in [8])*

$$r_{trg}(z) = \frac{\cos(z)}{2\sin(z)} h \otimes h + \frac{1}{\sin(z)}\big(e \otimes f + f \otimes e\big) + \sin(z) f \otimes f. \tag{5.16}$$

Example 5.2.15. *(given in [19])*

$$r_{trg}(z) = \frac{1+e^z}{4(1-e^z)} h \otimes h + \frac{1}{1-e^z} f \otimes e + \frac{e^z}{1-e^z} e \otimes f \tag{5.17}$$

Chapter 6

The Associative Yang-Baxter Equation

In this chapter we introduce the notion of associative r-matrices, which were introduced in [1] and [2], and again independently in [19]. We aim to give a brief overview of the AYBE and explain the relation between classical and associative Yang-Baxter equations.

6.1 The Associative Yang-Baxter Equation

Let A be an associative algebra. In the case where there is no dependence on variables, the *associative Yang-Baxter Equation* for r over A is the equation

$$r^{12}r^{13} - r^{23}r^{13} + r^{13}r^{23} = 0$$

where r is a meromorphic function of two variables (u, v) in a neighbourhood of $(0, 0)$ taking values in $A \otimes A$, where $A = \mathsf{Mat}(n, \mathbb{C})$ is the matrix algebra. The algebraic meaning of this equation is explained in [1, 2].

For the purposes of this thesis, we use a construction of Polishchuk, the AYBE with parameters:

$$\begin{aligned} r^{12}(u_3, u_2; y_1, y_2)r^{13}(u_1, u_3; y_1, y_3) - r^{23}(u_1, u_3; y_2, y_3)r^{12}(u_1, u_2; y_1, y_2) \\ + r^{13}(u_1, u_2; y_1, y_3)r^{23}(u_2, u_3; y_2, y_3) = 0. \quad (6.1) \end{aligned}$$

A solution is called unitary if

$$r(v_1, v_2; y_1, y_2) = -\tau(r(v_2, v_1; y_2, y_1)).$$

Assume a unitary solution $r(v_1, v_2; y_1, y_2)$ of the AYBE depends on the difference $u = u_1 - u_2$, $v = u_2 - u_3$ of the first pair of parameters only. Then Equation (6.1) can be rewritten as

$$r^{12}(-v; y_1, y_2)r^{13}(u+v; y_1, y_3) - r^{23}(u+v; y_2, y_3)r^{12}(u; y_1, y_2) \\ + r^{13}(u; y_1, y_3)r^{23}(v; y_2, y_3) = 0 \quad (6.2)$$

where r is a meromorphic function of two complex variables with values in $A \otimes A$.

Example 6.1.1. *An example of a solution of the AYBE, Equation (6.2), taken from [8] is:*

$$r(v; y_1, y_2) = \frac{1}{v}\mathbb{1} \otimes \mathbb{1} + \frac{2}{y_2 - y_1}\big(e_{11} \otimes e_{11} + e_{22} \otimes e_{22} + e_{12} \otimes e_{21} + e_{21} \otimes e_{12}\big) \\ + (v - y_1)e_{21} \otimes h + (v + y_2)h \otimes e_{21} + v(v - y_1)(v + y_2)e_{21} \otimes e_{21} \quad (6.3)$$

where $\mathbb{1} = e_{11} + e_{22}$.

Finally, assume that a solution to the AYBE, Equation (6.1), has the form $r(v_1, v_2; y_1, y_2) = r(v_2 - v_1; y_2 - y_1)$. Then the AYBE can be rewritten as

$$r^{12}(-v; x)r^{13}(u+v; x+y) - r^{23}(u+v; y)r^{12}(u; x) \\ + r^{13}(u; x+y)r^{23}(v; y) = 0. \quad (6.4)$$

Example 6.1.2. *An example of a solution of this type, taken from [8], is:*

$$r(v, y) = \frac{1}{2v}\mathbb{1} \otimes \mathbb{1} + \frac{1}{y}(e_{11} \otimes e_{11} + e_{22} \otimes e_{22} + e_{12} \otimes e_{21} + e_{21} \otimes e_{12}). \quad (6.5)$$

6.2 The Relation Between the AYBE and the CYBE

One special case studied in [19] is where $A = \mathsf{Mat}(n, \mathbb{C})$ a solution r to the AYBE also satisfies the unitarity condition

$$r(u, v) = -\tau(r(-u, -v))$$

and has a Laurent expansion near $u = 0$ of the form

$$r(u, v) = \frac{\mathbb{1} \otimes \mathbb{1}}{u} + r_0(v) + ur_1(v) + O(u^2).$$

In this case, it was shown that $r_0(v)$ satisfies the CYBE, that is,

$$[r_0^{12}(u), r_0^{13}(u+v)] + [r_0^{12}(u), r_0^{23}(v)] + [r_0^{13}(u+v), r_0^{23}(v)] = 0$$

and the unitarity condition

$$\tau(r(-u)) = -r(u)$$

holds. This follows from the fact that even without the Laurent condition, when the limit $\bar{r}(u) = (\mathsf{pr} \otimes \mathsf{pr}) r(u,v)\,|_{u=0}$ exists (pr is the projection away from the identity to traceless matrices), it is a unitary solution of the CYBE:

Proposition 6.2.1. *Let $A = L$ and $\mathsf{pr} : L \to L'$ the orthogonal projection with respect to the standard form $(B, C) = \mathsf{tr}(BC)$. If r is a unitary solution to the AYBE, and the limit $\bar{r}(u) = [(\mathsf{pr} \otimes \mathsf{pr}) r(u,v)]\,|_{u=0}$ exists, then $\bar{r}$ is a unitary solution to the CYBE.*

Proof. ([19] Lemma 1.2) Note that the unitarity of $\bar{r}$ follows from the unitarity of r: $r(u,v) = -\tau(r(-u,-v))$, therefore, $\bar{r}(u) = -[(\mathsf{pr} \otimes \mathsf{pr})\tau(r(-u,-v))] = -\tau\big([(\mathsf{pr} \otimes \mathsf{pr}) r(-u,-v)]\big)$. Substituting $-\tau(r(-u,-v)) = r(u,v)$ into Equation (6.4) we have,

$$\begin{aligned} - r^{13}(u+v, x+y)\big(\tau r(v,-x)\big)^{12} + \big(\tau r(-u,-x)\big)^{12} r^{23}(u+v, y) \\ + r^{23}(v,y) r^{13}(u, x+y) = 0. \end{aligned} \quad (6.6)$$

If we apply the map τ_{213}, defined in the proof for Proposition 4.4.2 part (2), to Equation (6.2) we obtain,

$$\begin{aligned} - r^{12}(v,-x) r^{23}(u+v, w+x) + r^{13}(u+v, x) r^{12}(-u,-w) \\ + r^{23}(u, w+x) r^{13}(v,x) = 0 \end{aligned}$$

We now make the linear change of variables given by $u \mapsto v$, $v \mapsto u$, $w \mapsto -w$ and $x \mapsto w + x$,

$$r^{13}(u+v, w+x) r^{12}(-v, w) - r^{12}(u,w) r^{23}(u+v, x) + r^{23}(v,x) r^{13}(u, w+x) = 0. \quad (6.7)$$

If we subtract Equation (6.7) from Equation (6.4) we get

$$\begin{aligned} [r^{12}(-v,w), r^{13}(u+v, w+x)] + [r^{12}(u,w), r^{23}(u+v, x)] \\ + [r^{13}(u, w+x), r^{23}(v,x)] = 0. \end{aligned} \quad (6.8)$$

We can apply the map $\mathsf{pr} \otimes \mathsf{pr} \otimes \mathsf{pr}$ to obtain

$$\begin{aligned}[\big((\mathsf{pr}\otimes\mathsf{pr})r(-v,w)\big)^{12}, \big((\mathsf{pr}\otimes\mathsf{pr})r(u+v,w+x)\big)^{13}]+ \\ [\big((\mathsf{pr}\otimes\mathsf{pr})r(u,w)\big)^{12}, \big((\mathsf{pr}\otimes\mathsf{pr})r(u+v,x)\big)^{23}]+ \\ [\big((\mathsf{pr}\otimes\mathsf{pr})r(u,w+x)\big)^{13}, \big((\mathsf{pr}\otimes\mathsf{pr})r(v,x)\big)^{23}] = 0, \quad (6.9)\end{aligned}$$

letting $u = v$ and taking the limit as $u \to 0$ we find that $\bar{r}(u)$ satisfies the CYBE. □

Remark 6.2.2. It is not known whether for every unitary nondegenerate solution $\bar{r}$ to the CYBE there exists a unitary solution to the AYBE of the form $\frac{\mathbb{1}\otimes\mathbb{1}}{u} + r_0(v) + ur_1(v) + O(u^2)$ such that $(\mathsf{pr}\otimes\mathsf{pr})(r_0(v)) = \bar{r}(v)$.

Corollary 6.2.3. *If r has a Laurent expansion at $u = 0$ of the form*

$$r(u,v) = \frac{\mathbb{1}\otimes\mathbb{1}}{u} + r_0(v) + ur_1(v) + O(u^2) \tag{6.10}$$

and is an associative solution to the AYBE, then $r_0(v)$ is a solution to the CYBE.

Proof. This follows from Equation (6.8) since $\mathbb{1}$ commutes with anything. □

Polishchuk conjectured that for $A = \mathsf{Mat}(n, \mathbb{C})$, the equivalence of the Belavin and Drinfeld classification holds, that is, all nondegenerate solutions of the AYBE are equivalent to either elliptic, or trigonometric, or rational solutions.

Chapter 7

The `ybe.lib` Library

One of the main goals of the work presented in this thesis was to produce a piece of software which performs the calculations necessary to check whether or not a given expression is a solution to the CYBE or the AYBE. In this chapter, we outline the main elements of the `ybe.lib` library, the piece of software produced. We first give a brief summary of the computer algebra system `SINGULAR`, we then discuss some of the more frequently used `SINGULAR` commands, this material is from [13] and [10]. We also summarize the challenges in implementing the `ybe.lib` library. The chapter also includes a number of examples of our `SINGULAR` code.

7.1 What is `SINGULAR`?

`SINGULAR` is a computer algebra system for polynomial computations, with special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory. It is free and open-source under the **GNU General Public Licence**. It can be downloaded from its homepage

`www.singular.uni-kl.de`

`SINGULAR` consists of a kernel written in C/C++ programming language, and libraries written in `SINGULAR`'s programming language. These libraries improve the internal functionality provided by the kernel and are easily changed and extended by the user. `SINGULAR` can either be run in a text terminal or within Emacs and is operating system independent.

We will now give some general information for users not familiar with SINGULAR. Once SINGULAR is started, it awaits an input after the prompt >. Every statement is terminated by a semicolon ;. Starting a line with a double backslash // denotes a comment and the rest of the line is ignored in calculations.

SINGULAR is a special purpose system for polynomial computations as it aims to enable the user to compute effectively in a polynomial ring as well as in the localisation of a polynomial ring at a maximal ideal. Hence, for almost all computations to be carried out in SINGULAR, a ring has to be defined first. If no ring has been created, only integer and string operations are available. SINGULAR offers very many different commutative rings and a class of non-commutative rings. In the following, ring always means a *commutative ring.* The rings in SINGULAR are either

1. polynomial rings over a field
2. localisations of polynomial rings, or
3. quotient rings with respect to an ideal.

To calculate with objects as polynomials, ideals, matrices, modules, and polynomial vectors a ring has to be defined first:

```
> ring r = 0, (x,y,z), dp;
```

The definition of a ring consists of three parts, the first part determines the ground field, the second part determines the names of the ring variables, and the third part determines the monomial ordering to be used. The above example declares a ring called r with a ground field of characteristic 0 (that is, rational numbers) and ring variables called x,y and z. The dp at the end means that degree reverse lexicographical ordering is used. All rings come equipped with a monimial order. See [13] Section 1.2 for a more detailed discussion of monomial orderings.

Defining a ring makes this ring the current active basering. However, if we want to calculate in a previously defined ring, we use the function setring. Once a ring is active, we can define polynomials. Ideals are represented as lists of polynomials which generate the ideal. A monomial x^2 can be entered either as x^2 or x2.

It is possible to define procedures which combine several commands to form a new one. Procedures are defined with the keyword proc followed by a name and an optional parameter list with specified types. A procedure may return an object by using the command `return` or can export an object to be used outside the procedure using the command `export`.

The distribution of `SINGULAR` contains several libraries, which extend the functionality of `SINGULAR`. Each of these libraries is a collection of useful procedures based on the kernel commands. The command `help all.lib;` lists all libraries together with a one-line explanation.

Libraries are loaded with the command `LIB` followed by the library name in inverted commas. For example `LIB "ybe.lib";`. The procedures in a library have a help part, which can be displayed by typing `help` followed by the procedure name. Examples can also be displayed by typing `example` followed by the procedure name. Also, the library itself has a help section, which shows a list of all the functions available for the user, which are contained in the library.

The following is a brief overview of the most important data types we have used in our `SINGULAR` library:

`int`: Variables of type `int` represent integers. Some useful `int` operations and `int` related functions are:

- `++` changes its operand to its successor.
- `%` integer modulo (the remainder of the division)
- `char` returns the characteristic of the coefficient field of a ring.
- `nvars` returns the number of variables of a ring.
- `rvar` returns the number of the variable if the name/poly is a ring variable of the basering or if the string is the name of a ring variable of the basering; returns 0 if not. Hence the return value of rvar can also be used in a boolean context to check whether the variable exists.
- `size` (of list, string) returns the length, i.e., the number of characters, entries or elements
- `var` var(n) returns the n-th ring variable

`list` Lists are arrays whose elements can be of any type (including ring and qring).

`map` Maps are ring maps from a preimage ring into the basering

`poly` Polynomials are the basic data for all main algorithms in `SINGULAR`. Polynomials can only be defined or accessed with respect to a basering, which determines the coefficient type, the names of the indeterminants and the monomial ordering.

`ring` Rings are used to describe properties of polynomials, ideals, etc. Almost all computations in `SINGULAR` require a basering. The following are some ring related functions we have used:

- `charstr` returns the description of the coefficient field of a ring.
- `ordstr` returns the description of the monomial ordering of the ring.
- `setring` changes the basering to another (already defined) ring
- `varstr` returns the list of the names of the ring variables as a string or the name of the n-th ring variable, where n is given by the int expression. If the ring name is omitted, the basering is used, thus `varstr(n)` is equivalent to `varstr(basering,n)`.

`string` Variables of type string are used for output (almost every type can be "converted" to string) and for creating new commands at runtime. String constants consist of a sequence of ANY characters between a starting `"` and a closing `"` Strings are especially useful to define new rings inside procedures as they may be used to execute Singular commands using the function `execute`. The result is the same as if the commands were written on the command line.

Some important `SINGULAR` commands which we use extensively in our program are the following:

`def`: Objects may be defined without a specific type: they get their type from the first assignment to them.

`proc`: Procedures are sequences of `SINGULAR` commands in a special format. They are used to extend the set of `SINGULAR` commands with user defined commands. Once a procedure is defined it can be used like any other `SINGULAR` command.

`imap`: identity map on common subrings. It is the map between rings and qrings with compatible ground fields which is the identity on variables and parameters of the same name and 0 otherwise.

`defined`: `defined(name)` returns a value $\neq 0$ (TRUE) if there is a user-defined object with this name, and 0 (FALSE) otherwise.

`execute`: executes a string containing a sequence of `SINGULAR` commands.

`execute`: returns the first position of a substring in a string or 0 (if not found). Starts the search at the position given in the (optional) third argument.

`kill`: deletes objects.

`lead`: `lead(I)` returns the leading term(s) of a polynomial.

`leadcoef`: `leadcoef(f)` returns the leading coefficient of a polynomial with respect to the monomial ordering.

`ringlist`: `ringlist(r)` decomposes a ring/qring into a list of four components:

a) the field description in the following format:

   for $\mathbb{Q}$, $\mathbb{Z}/p$: the characteristic (0 or prime number)

   for real/complex: the characteristic (always 0), the precision (2 integers), the name of the imaginary unit.

b) the names of the variables (a list of L string, L[i] is the name of the i-th variable.

c) the monomial ordering

d) the quotient ideal.

`setring`: `setring(S)` changes the basering to the (already defined) ring S.

`subst`: `subst(f,x,m)` substitutes the ring variable x for the term m.

A sequence of commands surrounded by curly brackets is called a *block*. Blocks are used in `SINGULAR` to define procedures and to collect commands belonging to `if, else, for` and `while` statements.

The `for` is for repetitive, conditional execution of a command block. In `for(i=1; i<7; i++)`, the initial command `i=1` is executed first. Then the boolean expression `i<7` is evaluated. If its value is TRUE the block is executed, otherwise the `for` statement is complete. After each execution of the block, the iterate command `i++` is executed and the boolean expression is evaluated. This is repeated until the boolean expression evaluates to FALSE. The command `break;` leaves the innermost `for` construct.

The `if` executes true block if the boolean condition is true. If the `if` statement is followed by an `else` statement and the boolean condition is false, then false block is executed. The command `if(i==1)` executes the command block only if the boolean expression $i = 1$ is TRUE and does nothing otherwise. The command `if(i==1){..}else{..}` executes the `if` command if the boolean expression $i = 1$ is TRUE and executes the `else` command otherwise.

The `while` is repetitive, conditional execution of a block. In `while(i<7)`, the boolean expression `i<7` is evaluated and if its value is TRUE, the block is executed. This is repeated until the boolean expression evaluates to FALSE. The command `break` leaves the innermost `while` construction.

7.2 The `ybe.lib` library

A `SINGULAR` library is a collection of `SINGULAR` procedures in a file. `SINGULAR` reads a library with the command `LIB`. After loading the library its procedures can be used like any built-in `SINGULAR` function. We use several in-built `SINGULAR` libraries in our `ybe.lib` library: the library `latex.lib` contains procedures for typesetting of Singular objects in LaTeX; and the library `ncalg.lib` contains commands for defining important Lie algebras.

The `ybe.lib` library is the result of the current research. The library is written in `SINGULAR`. Singular was the most obvious option as it is open source and is specifically designed for polynomial computations. `SINGULAR` can also facilitate calculations in non-commutative algebras.

The main idea is that the user can input an expression in order to check whether or not it is a solution to the CYBE or the AYBE. The user should be able to check the value of the left-hand-side of the CYBE or AYBE for any expression entered. The framework of the library should also allow the user to convert this value to LaTeX format. Apart from checking solutions, the user will also have access to a procedure which can calculate the Casimir element (see Definition 2.5.1) for $\mathsf{sl}(n)$ and two procedures which give rational solutions to the CYBE with 2 parameters.

The library is organised as follows: An `info-` and `version-` appear at the beginning of the library before the first procedure definition. We begin with an overview of the functions of the library and then list all procedures in the library with a short description of the function of each procedure. The procedures are in four separate categories, first are procedures relating to checking solutions to the CYBE, the next section has procedures relating to checking solutions to the AYBE (some procedures are common to both of these), in the next section we have procedures which allow us to convert the value of the left-hand-side of the CYBE/AYBE to LaTeX format, and finally we have procedures which are related to calculating the Casimir element of a given Lie algebra. Each procedure which is not accessible by users is declared `static`. Each procedure which is not declared `static` has an example and help section.

7.3 Implementation Details of `ybe.lib`

In this section we aim to give a detailed description of all the procedures in our `ybe.lib` library. We also explain the process each procedure goes through and how it helps to establish whether or not an expression is a solution to the CYBE/AYBE.

Recall again the classical Yang-Baxter equation:

$$[r^{12}(u), r^{13}(u+v)] + [r^{12}(u), r^{23}(v)] + [r^{13}(u+v), r^{23}(v)] = 0 \tag{7.1}$$

and the associative Yang-Baxter equation:

$$r^{12}(-v,w)r^{13}(u+v,w+x) - r^{23}(u+v,x)r^{12}(u,w) + r^{13}(u,w+x)r^{23}(v,x) = 0. \tag{7.2}$$

When we produced our `ybe.lib` library, we used the following calculations: Let L be a Lie algebra, and let r be an $L \otimes L$-valued function. In terms of a basis $\{I_\mu\}$ of L, write

$$r(u) = \sum_{\mu\nu} r_{\mu\nu}(u) I_\mu \otimes I_\nu$$

with $\mathbb{C}$-valued functions $r_{\mu\nu}(u)$. Recall from Definition 3.2.8 the linear maps $(a \otimes b)^{ij}$ with $a, b \in L$ and $1 \le i < j \le 3$. So we have $r^{12}(u) = \sum r_{\mu\nu} I_\mu \otimes I_\nu \otimes 1 \in U(L) \otimes U(L) \otimes U(L)$ and so on, where $U(L)$ denotes the universal enveloping algebra (see Definition 2.4.1). We know that the defining relations of $U(L)$ are the same as the relations between the basis elements of the Lie algebra L, therefore we note, for example, that in calculating the first term in the CYBE, Equation (7.1), we use the operation of commutation in L:

$$\begin{aligned}
[r^{12}(u), r^{13}(v)] &= \Big[\Big(\sum_{\mu\nu} r_{\mu\nu}(u) I_\mu \otimes I_\nu\Big)^{12}, \Big(\sum_{\sigma\rho} r_{\sigma\rho}(u+v) I_\sigma \otimes I_\rho\Big)^{13}\Big] \\
&= \sum_{\mu\nu\sigma\rho} r_{\mu\nu}(u) r_{\sigma\rho}(u+v)\Big[I_\mu \otimes I_\nu \otimes 1, I_\sigma \otimes 1 \otimes I_\rho\Big] \\
&= \sum_{\mu\nu\sigma\rho} r_{\mu\nu}(u) r_{\sigma\rho}(u+v)\Big((I_\mu \otimes I_\nu \otimes 1)(I_\sigma \otimes 1 \otimes I_\rho) \\
&\qquad - (I_\sigma \otimes 1 \otimes I_\rho)(I_\mu \otimes I_\nu \otimes 1)\Big) \\
&= \sum_{\mu\nu\sigma\rho} r_{\mu\nu}(u) r_{\sigma\rho}(u+v)(I_\mu \cdot I_\sigma \otimes I_\nu \cdot 1 \otimes 1 \cdot I_\rho \\
&\qquad I_\sigma \cdot I_\mu \otimes 1 \cdot I_\nu \otimes I_\rho \cdot 1) \\
&= \sum_{\mu\nu\sigma\rho} r_{\mu\nu}(u) r_{\sigma\rho}(u+v)\Big([I_\mu, I_\sigma] \otimes I_\nu \otimes I_\rho\Big).
\end{aligned}$$

Similarly,

$$\begin{aligned}
[r^{12}(u), r^{23}(v)] &= \sum_{\mu\nu\sigma\rho} r_{\mu\nu}(u) r_{\sigma\rho}(v)\Big(I_\mu \otimes [I_\nu, I_\sigma] \otimes I_\rho\Big) \\
[r^{13}(u), r^{23}(v)] &= \sum_{\mu\nu\sigma\rho} r_{\mu\nu}(u+v) r_{\sigma\rho}(v)\Big(I_\mu \otimes I_\sigma \otimes [I_\nu, I_\rho]\Big).
\end{aligned}$$

Therefore, we needed to create a ring with the basis elements of L and variables u and v.

Remark 7.3.1. Note that for solutions of the form $r(u, v)$ we must create a ring with variables u_1, u_2 and u_3.

Remark 7.3.2. Please note that the `SINGULAR` relations for $\mathfrak{sl}(n)$ do not correspond exactly to those given in Example 1.6.6. In `SINGULAR`, using $\mathfrak{sl}(5)$ as an example, the basis elements would correspond to the following matrix:

$$\begin{pmatrix}
 & x_4 & x_7 & x_9 & x_{10} \\
y_4 & & x_3 & x_6 & x_8 \\
y_7 & y_3 & & x_2 & x_5 \\
y_9 & y_6 & y_2 & & x_1 \\
y_{10} & y_8 & y_5 & y_1 &
\end{pmatrix}$$

with $h_1 = e_{44} - e_{55}, \ldots, h_4 = e_{11} - e_{22}$.

In calculating the AYBE, Equation (7.2), we used the rules of matrix products:

$$\begin{aligned}
r^{13}(u+v,&y+x)r^{12}(-v,y)\\
&=\sum_{ijklrspq} r_{ijkl}(u+v,y+x)r_{rspq}(-v,y)\big(e_{ij}\otimes 1\otimes e_{kl}\big)\big(e_{rs}\otimes e_{pq}\otimes 1\big)\\
&=\sum_{ijklrspq} r_{ijkl}(u+v,y+x)r_{rspq}(-v,y)\big(e_{ij}\cdot e_{rs}\otimes e_{pq}\otimes e_{kl}\big)\\
&=\sum_{ijklrspq} r_{ijkl}(u+v,y+x)r_{rspq}(-v,y)\delta_{jr}\big(e_{is}\otimes e_{pq}\otimes e_{kl}\big).
\end{aligned}$$

Similarly,

$$\begin{aligned}
r^{12}(u,y)r^{23}(u+v,x) &= \sum_{ijklrspq} r_{ijkl}(u,y)r_{rspq}(u+v,x)\delta_{lr}\big(e_{ij}\otimes e_{ks}\otimes e_{pq}\big),\\
r^{23}(v,x)r^{13}(u,y+x) &= \sum_{ijklrspq} r_{ijkl}(v,x)r_{rspq}(u,y+x)\delta_{lp}\big(e_{rs}\otimes e_{ij}\otimes e_{kp}\big).
\end{aligned}$$

Therefore, we needed to include all basis elements of the associative algebra in question, along with variables u, v, y, and x.

Remark 7.3.3. Note that for the 3 parametric associative solution, we need to include variables u, v, x_1, x_2, and x_3.

In order to check whether or not an expression is a solution to the YBE the process the user must undertake is as follows: First they must define a ring, as `SINGULAR` cannot perform calculations unless a ring has been defined. The user must then set this ring as the basering using the `SINGULAR` command `setring`. The next step is to input the expression to be checked. Numerous expressions can be checked once the ring they are in has been set as the basering.

We will now describe some of the procedures of `ybe.lib` in more detail. The procedure `createRingRational` creates what we call a ‘rational ring’ that is, a ring within which we can carry out calculations to test for rational solutions. The user inputs the Lie algebra that the expression to be checked is in, between inverted commas, for example “$\mathsf{sl}(3)$”. This procedure exports this inputted ring as the ‘original ring’, note that this ring contains all basis elements of the Lie algebra L. What is returned is the ’original ring’ with required parameters `u` and `v` included, plus the variable `z` included to the variable list. We include the `u` and `v` as parameters and not variables because they will be used in the denominator of internal calculations and `SINGULAR` does not allow variables to be used in the denominator. The variable `z` is from the expression entered $r(z)$. For example, in $\mathsf{sl}(2)$ we create the following ring:

```
> def ratRing=createRingRational("sl(2)");
> ratRing;
characteristic :  0
2 parameter :  u v
minpoly :  0
number of vars :  4
block 1 :  ordering dp
:  names e f h z
block 2 :  ordering C
>
```

Note that this *commutative* ring is $\mathbb{Q}(u,v)[e,f,h,z]$.

The procedures createRingRational2par and createRingTrig have similar inputs and outputs with suitable parameters and variables included. For example, in $\mathsf{sl}(3)$ we can create the following 3 parametric 'rational ring':

```
> def ratRing2par=createRingRational2par("sl(3)");
> ratRing2par;
characteristic :  0
3 parameter :  u1 u2 u3
minpoly :  0
number of vars :  10
block 1 :  ordering dp
:  names x(1) x(2) x(3) y(1) y(2) y(3) h(1) h(2) z_1 z_2
block 2 :  ordering C
>
```

which is the commutative ring $\mathbb{Q}(u_1,u_2,u_3)[x_1,x_2,x_3,y_1,y_2,y_3\ ,h_1,h_2,z_1,z_2]$, and the following 'trigonometric ring':

```
> def trigRing=createRingTrig("sl(3)");
> trigRing;
characteristic :  0
2 parameter :  u v
minpoly :  0
number of vars :  15
block 1 :  ordering dp
```

```
:  names x(1) x(2) x(3) y(1) y(2) y(3) h(1) h(2) z sin cos
cos_u sin_u cos_v sin_v I
block 2 :  ordering C
```

is the commutative ring $\mathbb{Q}(u,v)[x_1,x_2,x_3,y_1,y_2,y_3,h_1,h_2,z,\sin,\cos,\sin(u),$ $\cos(u),\sin(v),\cos(v),I]$. Note that the `I` in the variable string of the 'trigonometric ring' is the variable that the user should enter for the complex number i. It is converted to this complex i by introducing the ideal I^2+1 at a later stage.

For the AYBE we create the following ring for checking solutions in $\mathsf{sl}(2)$ with 3 parameters:

```
> def assocRing=createRingA3par(2);
> assocRing;
characteristic :  0
5 parameter :  u v w_1 w_2 w_3
minpoly :  0
number of vars :  7
block 1 :  ordering dp
:  names e_1_1 e_1_2 e_2_1 e_2_2 z y_1 y_2
block 2 :  ordering C
```

which is the ring $\mathbb{Q}(u,v,w_1,w_2,w_3)[e_{11},e_{12},e_{21},e_{22},z,y_1,y_2]$. These procedures use the `static` internal procedure `internalnoncalg` and `internalAssocRing`.

The main procedures for checking solutions are `IsSolutionCYBE`, and `IsSolutionAYBE`. The user must break down each element of the expression and input it as a list in the form: numerator, denominator, tensor 1, tensor 2. For example, the rational expression in $\mathsf{sl}(2)$, $\frac{1}{z}(e\otimes f+f\otimes e)$ should be entered as:

```
>IsSolutionCYBE(1,z,e,f,\
1,z,f,e);
```

and the trigonometric expression in $\mathsf{sl}(2)$,

$$\frac{\cos(z)}{2\sin(z)}h\otimes h+\frac{1}{\sin(z)}e\otimes f+\frac{1}{\sin(z)}f\otimes e+\sin(z)f\otimes f$$

should be entered as:

```
>IsSolutionCYBE(cos,2*sin,h,h,\
1,sin,e,f,\
1,sin,f,e,\
sin,1,f,f);
```

The process involved in calculating whether or not the expression input is a solution is as follows:

1. The procedure performs an initial check to determine which ring the expression is contained in. We first set the ring type to 'rational ring' with one parameter, we use the `SINGULAR` command `find` to look for either `z_1` or `sin` in the variable string of the basering. If either of these are found then the ring type is changed accordingly. For 'associative rings' the procedure is very similar; the ring type is initially set to the three parameter associative ring, but there is an `x` in the variable string of the basering it is changed to the 2 parameter 'associative ring'.

2. The user-inputted list is split into four separate lists by the procedure `MakeLists`.

3. A new ring is created to be used internally. This ring has additional variables which allow us to mimic the rules of tensor product.

4. The four lists created are mapped into this internal ring.

5. The procedure `totalAllBrackets` or `totalAYBE` is used to give the value of the left-hand-side of the CYBE or AYBE. These procedures take each of the three elements in Equation (7.1) or (7.2) and works out their value. The results are then added/subtracted according to the equation being tested.

6. If the returned value of the above procedure is zero, then `SINGULAR` prints the text "`Is a solution to the CYBE`" or "`Is a solution to the  AYBE`". However, if the value is not zero, then this value is converted to a string of the same format type as the input: numerator, denominator, tensor1, tensor2, tensor3, linebreak, etc. and is exported so that it can be printed, if required, by the procedure `showValue`. `SINGULAR` then prints the text "`Is not a solution to the CYBE`" or "`Is not a solution to the AYBE`".

For checking whether expressions are solutions to the CYBE, the procedure `totalAllBrackets` performs all the main computations. The input is the list of four lists made by the procedure `MakeLists`, a list of numerators, a list of denominators, a list of all the first tensors, and a list of all the second tensors. Firstly, the procedure takes every combination of the double tensors and uses the procedure `nonCommutativePart` to perform the calculations shown above, that is, for $a \otimes b, c \otimes d \in L \otimes L$:

$$[(a \otimes b)^{12}, (c \otimes d)^{13}] = [a, c] \otimes b \otimes d, \tag{7.3}$$
$$[(a \otimes b)^{12}, (c \otimes d)^{23}] = a \otimes [b, c] \otimes d, \tag{7.4}$$
$$[(a \otimes b)^{13}, (c \otimes d)^{23}] = a \otimes c \otimes [b, d].. \tag{7.5}$$

The Lie bracket in each part is also calculated in this procedure. The results are given in the form of a list of three variables.

For example, for the expression $r(z) = \frac{1}{z}(e \otimes f + f \otimes h)$, calculations in the procedure `nonCommutativePart` would include:

1. `nonCommutativePart[1](e,f,e,f);` calculates the Lie bracket in Equation (7.3) and returns:
   `0,f,f.`

2. `nonCommutativePart[2](f,h,e,f);` calculates the Lie bracket in Equation (7.4) and returns:
   `f,2e,f.`

3. `nonCommutativePart[3](e,f,f,h);` calculates the Lie bracket in Equation (7.5) and returns:
   `e,f,2f.`

The procedure `MakeTensor` then takes these lists and returns a polynomial which mimics the rules of tensor product.

Taking the same examples as above, we would obtain the following:

1. `MakeTensor(0,f,f);` creates the polynomial:
   `0.`

2. `MakeTensor(f,2e,f);` creates the polynomial:
   $2{\cdot}\mathtt{f}_1{\cdot}\mathtt{e}_2{\cdot}\mathtt{f}_3$.

3. `MakeTensor(3,f,2f);` creates the polynomial:
   $2{\cdot}\mathtt{e}_1{\cdot}\mathtt{f}_2{\cdot}\mathtt{f}_3$.

This process involved the procedure `MakeTensor` is explained in more detail in Section 7.4. We then take into consideration the coefficients of each tensor. The variable 'z' is replaced by the appropriate parameters, `u`,`v`, or `u+v` using the `SINGULAR` command `subst`. These steps are taken separately for each bracket from Equations (7.3-7.5) and are then added together. If the expression entered is a solution to the CYBE then the total should be zero.

For checking expressions in the AYBE, the procedure `totalAYBE` works in a similar manner, the difference being that the procedure `matrixMult` is used in place of `MakeTensor`. This procedure mimics the rules of matrix multiplication. `SINGULAR` contains a procedure for multiplying matrices, but this needs the matrices to be input explicitly, whereas in our associative procedure, we use variables $e_{11}, e_{12}, \dots, e_{nn}$ but do not explicitly define each variable. The procedure `matrixMult` takes the position each variable lies in the variable string of the ring to calculate the integers i and j from e_{ij}. We then use the rule $e_{ij} \cdot e_{kl} = \delta_{jk} e_{il}$.

The procedure `showValue` allows the user to see the value of the left-hand-side of the CYBE or AYBE of last expression checked. This does not work for trigonometric solutions. This procedure must be used directly after checking an expression using `IsSolutionCYBE` or `IsSolutionAYBE`. These two procedures export the value of the left-hand-side of the YBE for expressions which are not solutions. The procedure `showValue` simply returns this value. For example,

```
> setring ratRing;
> IsSolutionCYBE(1,z,e,f,\
.  1,z,f,h);
Is not a solution to the CYBE
> showValue();
2,(u2+uv),f,e,f
2,(uv+v2),e,f,f
-1,(uv),e,h,f
-1,(u2+uv),h,h,f
-2,(uv),f,f,h
1,(u2+uv),h,f,h
```

The procedure `casimirEl` computes the Casimir element for $L = \mathsf{sl}(n)$ with respect to the *trace form*. This procedure uses the expression for the Casimir element of $\mathsf{sl}(n)$ given in Remark 2.5.9 and returns Equation (2.7). From Proposition 5.2.10 we know that $\frac{t}{u}$ is a solution to the CYBE, the procedure `casimirZTest` returns this expression in the appropriate format to be entered directly into our procedure `IsSolutionCYBE` when the 'rational ring' has been set. Furthermore, the expression $\frac{t}{u_1 - u_2}$ is a solution to the rational CYBE with 2 parameters, the procedure `casimirZTest2par` returns this expression in the appropriate format to be entered directly into our procedure `IsSolutionCYBE` when the 'rational ring' with 2 parameters has been set. For example,

```
> def ratRing=createRingRational("sl(2)");
> setring ratRing;
> IsSolutionCYBE(casimirZTest(2));
Is a solution to the CYBE
```

In order to test our library for the CYBE, we used rational solutions of type $r(u,v) == t/(u-v) + g(u,v)$ (Equation (5.12)) which were provided by Thilo Henrich [15]. Solutions of this type for $\mathsf{sl}(n)$ can be found using the following equation:

$$\begin{aligned} r(u,v) = \frac{t}{u-v} &+ u\left[e_{12} \otimes h_1^* - \sum_{j=3}^{n} e_{1,j} \otimes \left(\sum_{k=1}^{n-j+1} e_{j+k-1,k+1} \right)\right] \\ &- v\left[h_1^* \otimes e_{12} - \sum_{j=3}^{n} \left(\sum_{k=1}^{n-j+1} e_{j+k-1,k+1} \right) \otimes e_{1,j}\right] \\ &+ \left[\sum_{j=2}^{n-1} e_{1,j} \otimes \left(\sum_{k=1}^{n-j} e_{j+k,k+1} \right)\right. \\ &+ \left. \sum_{i=2}^{n-1} e_{i,i+1} \otimes h_i^* - \sum_{i=2}^{n-2} \left(\sum_{k=2}^{n-i} e_{i,i+k} \otimes \left(\sum_{l=1}^{n-i-k+1} e_{i+k+l-1,l+i} \right) \right)\right] \\ &+ \left[\sum_{j=2}^{n-1} \left(\sum_{k=1}^{n-j} e_{j+k,k+1} \right) \otimes e_{1,j}\right. \\ &+ \left. \sum_{i=2}^{n-1} h_i^* \otimes e_{i,i+1} - \sum_{i=2}^{n-2} \left(\sum_{k=2}^{n-i} \left(\sum_{l=1}^{n-i-k+1} e_{i+k+l-1,l+i} \right) \otimes e_{i,i+k} \right)\right]. \end{aligned} \tag{7.6}$$

or the following equation:

$$\begin{aligned} r(u,v) = \frac{t}{u-v} &+ v\Bigg[h_{n-1}^* \otimes e_{n,n-1} + \sum_{i=1}^{n-2}\Bigg(\sum_{j=1}^{i} e_{j,j+n-i-1}\Bigg) \otimes e_{n,i}\Bigg] \\ &- u\Bigg[e_{n,n-1} \otimes h_{n-1}^* + \sum_{i=1}^{n-2} e_{n,i} \otimes \Bigg(\sum_{j=1}^{i} e_{j,j+n-i-1}\Bigg)\Bigg] \\ &+ \Bigg[-\sum_{i=2}^{n-1}\Bigg(\sum_{j=1}^{i-1} e_{j,j+n-i}\Bigg) \otimes e_{n,i} + \sum_{i=1}^{n-2} h_i^* \otimes e_{i+1,i} \\ &+ \sum_{i=3}^{n-1}\Bigg(\sum_{j=1}^{i-2}\Bigg(\sum_{k=1}^{j} e_{k,k+i-j-1}\Bigg) \otimes e_{i,j}\Bigg)\Bigg] \\ &- \Bigg[-\sum_{i=2}^{n-1} e_{n,i} \otimes \Bigg(\sum_{j=1}^{i-1} e_{j,j+n-i}\Bigg) \\ &+ \sum_{i=1}^{n-2} e_{i+1,i} \otimes h_i^* + \sum_{i=3}^{n-1}\Bigg(\sum_{j=1}^{i-2} e_{i,j} \otimes \Bigg(\sum_{k=1}^{j} e_{k,k+i-j-1}\Bigg)\Bigg)\Bigg]. \end{aligned} \tag{7.7}$$

To check the validity of the above equations, we produced two procedures `ratSolType1` and `ratSolType2` whose outputs give the right-hand-side of Equations (7.6) and (7.7) respectively. Note that we had to keep in mind Remark 7.3.2 when producing these procedures, so there are certain sign changes to ensure that the results are correct. The output of each of these two procedures is in the form of a list which can be input directly into our procedure for testing solutions to the CYBE. For example,

```
> def ratRing2par4=createRingRational2par("sl(4)");
> setring ratRing2par4;
> list L1=casimirZTest2par(4);
> list L2=ratSolType1(4);
> list L3=ratSolType2(4);
> list L=L1+L2;
> IsSolutionCYBE(L);
Is a solution to the CYBE
> L=L1+L3;
> IsSolutionCYBE(L);
Is a solution to the CYBE
>
```

There are also a number of procedures in `ybe.lib` designed to be used if the value of an expression which is not a solution is required in LaTeX format. The procedure `texYBE` converts the value of the left-hand-side of the CYBE/AYBE to a LaTeX format. This can be printed on screen or saved to a user-named file. For example,

```
> IsSolutionCYBE(1,z_2-z_1,h,e,\
.  z2,1,e,f);
Is not a solution to the CYBE
> string s=showValue();
> texYBE("",s);
```

prints the expression:

$$-\frac{2}{u_1 \cdot u_2 - u_1 \cdot u_3 - u_2^2 + u_2 \cdot u_3} h \otimes e \otimes e \\ - \frac{(2 \cdot u_1 \cdot u_2 - 2 \cdot u_2^2)}{u_1 \cdot u_2 - u_1 \cdot u_3 - u_2 \cdot u_3 + u_3^2} e \otimes f \otimes e \\ - \frac{(2 \cdot u_3)}{u_1 - u_2} e \otimes e \otimes f - \frac{(u_2 \cdot u_3)}{1} e \otimes h \otimes f \\ - \frac{(u_3)}{u_1 - u_3} h \otimes e \otimes h + \frac{u_3}{u_2 - u_3} e \otimes h \otimes h.$$

Using the solutions given in Chapters 5 and 6, we can test our program, and achieve the following results:

Rational Solutions

```
> def ratRing=createRingRational("sl(2)");
> setring ratRing;
IsSolutionCYBE(1,2z,h,h,\
1,z,e,f,\
1,z,f,e);
Is a solution to the CYBE
```

(*Note that this is Equation (5.9)*).

```
> IsSolutionCYBE(1+z,4-4z,h,h,\
1,1-z,f,e,\
z,1-z,e,f);
Is a solution to the CYBE
```

Note that this is Equation (5.13)

```
> def ratRing5=createRingRational("sl(5)");
> setring ratRing5;
> IsSolutionCYBE(casimirZTest(5));
```

```
Is a solution to the CYBE
```
(*Note that this is a solution of type* $\frac{t}{u}$)

Rational Solutions with 2 parameters

```
> def ratRing2par=createRingRational("sl(2)");
> setring ratRing2par;
> IsSolutionCYBE(casimirZTest2par(2));
Is a solution to the CYBE
```
(*Note that this is the solution* $\frac{t}{u-v}$).

```
> list L1=casimirZTest2par(2);
> list L2=-z_1,2,f,h,z2,2,h,f;
> list L=L1+L2;
> IsSolutionCYBE(L);
Is a solution to the CYBE
```
(*Note that this is Equation (5.15)*)

```
> def ratRing2par5=createRingRational2par("sl(5)");
> setring ratRing2par5;
> list L1=casimirZTest2par(5);
> list L2=ratSolType1(5);
> list L=L1+L2;
> IsSolutionCYBE(L);
Is a solution to the CYBE
>
```

Trigonometric Solutions

```
> def trigRing=createRingTrig("sl(2)");
> setring trigRing;
> IsSolutionCYBE(cos,2*sin,h,h,\
1,sin,e,f,\
1,sin,f,e,\
sin,1,f,f);
Is a solution to the CYBE
```
(*Note that this is Equation (5.16)).*

```
> IsSolutionCYBE(1+cos+I*sin,4-4*cos-4*I*sin,h,h,\
1,1-cos-I*sin,f,e,\
cos+I*sin,1-cos-I*sin,e,f);
```

```
Is a solution to the CYBE
```

(Note that this is Equation (5.17)).

Associative Solutions

```
> def assocRing2=createRingA2par(2);
> def assocRing3=createRingA3par(2);
> setring assocRing2;
> IsSolutionAYBE(1,2z,e_1_1,e_1_1,\
1,2z,e_1_1,e_2_2,\
1,2z,e_2_2,e_1_1,\
1,2z,e_2_2,e_2_2,\
1,w,e_1_1,e_1_1,\
1,w,e_2_2,e_2_2,\
1,w,e_1_2,e_2_1,\
1,w,e_2_1,e_1_2);
Is a solution to the AYBE
```

(Note that this is Equation (6.5)).

```
> setring assocRing3;
> IsSolutionAYBE(1,2z,e_1_1,e_1_1,\
1,2z,e_1_1,e_2_2,\
1,2z,e_2_2,e_1_1,\
1,2z,e_2_2,e_2_2,\
1,y_2-y_1,e_1_1,e_1_1,\
1,y_2-y_1,e_2_2,e_2_2,\
1,y_2-y_1,e_1_2,e_2_1,\
1,y_2-y_1,e_2_1,e_1_2,\
z-y_1,2,e_2_1,e_1_1,\
-z+y_1,2,e_2_1,e_2_2,\
z+y_2,2,e_1_1,e_2_1,\
-z-y_2,2,e_2_2,e_2_1,\
-z*(z-y_1)*(z+y_2),2,e_2_1,e_2_1);
Is a solution to the AYBE
```

(Note that this is Equation (6.3)).

7.4 Implementation Issues

This section presents the main difficulties arising during the conception and the implementation of `ybe.lib`. Several implementation issues were faced and solved. A major problem we encountered was in mimicking the tensor product between variables. The `SINGULAR` command for tensor product `tensor(A,B);` returns the tensor product of the matrices A and B. This is not suitable for our procedures as we do not need to compute the value of tensor $a \otimes b \otimes c$, we only need the structure and rules of triple tensors. We overcame this problem by creating a new internal ring to use for all calculations. In this ring we included the variables $\Lambda = \{X(i) \,|\, X \in \text{ basis of } L,\, i = 1, 2, 3\}$, that is for $\mathfrak{sl}(2)$, the solution ring is:

```
> def ratRing=createRingRational("sl(2)");
> ratRing;
characteristic :  0
2 parameter :  u v
minpoly :  0
number of vars :  4
block 1 :  ordering dp
:  names e f h z
block 2 :  ordering C
>
```

and the internal ring within which we work is:

```
> internalRing;
characteristic :  0
2 parameter :  u v
minpoly :  0
number of vars :  13
block 1 :  ordering dp
:  names e f h e(1) f(1) h(1) e(2) f(2) h(2) e(3) f(3) h(3) z
block 2 :  ordering C
>
```

The strategy we employed here is based on the following. We know that the user inputs an expression $r(z) \in L \otimes L \subset U(L) \otimes U(L)$, so our ring which we create for the user must contain all basis elements of L. Furthermore, even though $r^{12}(u)$, $r^{13}(u + v)$ and $r^{23}(v)$ are all elements of

$U(L) \otimes U(L) \otimes U(L)$, the Lie bracket of each of these elements with each other is in $L \otimes L \otimes L$ (Recall Example 3.2.9). If we let Π be the vector subspace generated by the elements $X(1) \cdot Y(2) \cdot Z(3)$ with X, Y, Z basis elements in L, then as a vector space, $L \otimes L \otimes L \cong \Pi \subset \mathbb{C}[\Lambda]$.

We then used the procedure `MakeTensor` to convert a list of three variables into a 'tensor product'. We did this by mapping each of the variables to a variable of the same name but with a bracketed number indicating the position it must take in the triple tensor. We then use the product of these three variables. For example, the list `f,e,h` in $\mathsf{sl}(2)$ would be converted to the polynomial `f(1)*e(2)*h(3)`. The position of the new variables in the variable string of our ring is important, we need all the variables with `(1)` to come first, then all the variables with `(2)` and finally all the variables with `(3)`. This ensures that the polynomial will always have the variables in chronological order. This method also overcame the problem of equating $e \otimes 2f \otimes h = 2e \otimes f \otimes h$ as the rules for product ensure that any coefficient is moved to the front of the polynomial.

We also encountered several problems during the development of the procedure for trigonometric solutions. The biggest challenge was to implement the trigonometric addition rules, $\cos(u+v) = \cos(u)\cos(v) - \sin(u)\sin(v)$ and $\sin(u+v) = \sin(u)\cos(v) + \cos(u)\sin(v)$. We discovered that the easiest and best way to solve the problem of the addition formulae was to create a procedure called `trigRules` which substitutes an expression for $\cos(u+v)$ or $\sin(u+v)$ with the expression on the right hand side. In order for the program to work properly, we also needed to have the procedure recognise that $\cos(z)^2 + \sin(z)^2 = 1$, We implemented this rule by using the `SINGULAR` command `ringlist` which allowed us to change the parts of a previously defined ring. We simply added a quotient ideal $\sin(z)^2 + \cos(z)^2 - 1$. We used the same method to overcome the problem of having a complex variable `I` in the 'trigonometric ring'. In `SINGULAR`, the ring definition for complex variable 'i' does not allow the user to include any more parameters. Hence, we needed to create the complex number 'i' as a variable `I` and add an ideal to the internal ring `I`$\hat{}2 + 1$. Similar to the case for the addition formulae for sine and cosine, we needed to use this complex number in the denominator of coefficients, so we had to substitute it for a parameter and then back to a variable again when working with the numerator only.

We also came across some implementation challenges which we could not solve. The first challenge was the need to define a ring and *then* set this ring as the basering. The user must be informed of the requirement to enter the command

```
>setring [ringname];
```

Another problem that we failed to overcome was the programs inability to give the value of the left-hand-side of the CYBE for trigonometric solutions. When calculating trigonometric solutions we need to work in yet another internal ring to calculate the CYBE. As we have mentioned before, `SINGULAR` does not allow the use of variables in the denominator, so we created a new ring with the complex number 'i' and the sine and cosine functions for u, v, and $u + v$ as *parameters*. We could then substitute these parameters for the variables in the calculation. However, procedures defined in `SINGULAR` must use the same ring for input and output, and the procedure `totalAllBrackets` returns a polynomial, so it was impossible to give the correct value for trigonometric solutions. We know that if an expression is a solution to the CYBE then the coefficient of each triple tensor must be zero. This happens only when the numerator of the coefficient is zero. So for trigonometric expressions we look at the numerator of each coefficient only and disregard the denominator.

7.5 Conclusion and Future Work

The main goal of this research was to develop a tool which enables the user to perform lengthy calculations of the CYBE or the AYBE using a computer program. The examples we have performed so far should produce evidence enough that the program successfully calculates whether or not a given expression is a solution to the CYBE or the AYBE. However, we have also identified some problems and limitations. More specifically, the need to have the user set the ring before computations can take place, and the fact that this program cannot calculate the value of the left-hand-side of the CYBE for trigonometric solutions.

In terms of future work, it is certainly worth experimenting on alternative methods of computation that would allow the user to see the value of the left-hand-side of the trigonometric CYBE. Although the library `ybe.lib` seems to produce promising results, there are still some aspects that should be optimized: computations on larger dimensional Lie algebras take some time. For the solution $r(z) = t/z$ in $\mathsf{sl}(9)$ the procedure takes 13 minutes and 31 seconds to give us a result (using SONY VAIO, AMD Athlon II X2P320 processor, 4GB RAM).

```
> def ratRing=createRingRational("sl(9)");
> setring ratRing;
> IsSolutionCYBE(casimirZTest(9));
Is a solution to the CYBE
```

Other directions of future work on the `ybe.lib` library may include a procedure for testing trigonometric solutions to the AYBE.

Bibliography

[1] Aguiar, M. Infinitesimal Hopf Algebras. *Contemporary Mathematics*, 267:1-30, 2000.

[2] Aguiar, M. On the Associative Analog of Lie Bialgebras. *Journal of Algebra*, 244:492-532, 2001.

[3] Belavin, A.A., Dynaminal Symmetry of Integrable Quantum systems. *Nucl. Physics*, B180:189-200, 1981.

[4] Belavin, A.A., Drinfeld, V. G. Solutions of the Classical Yang-Baxter Equation for Simple Lie algebras. *Functional Analysis & its Applications*, 16:159-180, 1981.

[5] Belavin, A.A., Drinfeld, V.G. Classical Yang-Baxter Equation for Simple Lie algebras. *Journal of Functional Analysis*, 17:69-70, 1982.

[6] Belavin, A.A., Drinfeld,V.G. *Triangle Equations and Simple Lie Algebras.* Harwood Academic Publishers, 1998.

[7] Borel, A. *Essays in the History of Lie groups and Algebraic Groups.* The American Mathematical Society, 2001.

[8] Burban, I., Kreussler, B. Vector Bundles on Degenerations of Elliptic Curves and Yang-Baxter Equations. *arXiv: math/0708.1685*, 2009.

[9] Chari, V., Pressley, A. *A Guide to Quantum Groups.* Cambridge University Press, 1994.

[10] Decker, W., Greuel, G.M., Pfister, G., Schönemann, H. Singular 3-1-1 - A computer algebra system for polynomial computations. 2010. Retrieved from http://www.singular.uni-kl.de.

[11] Erdmann, K., Wildon, M.J. *Introduction to Lie algebras.* Springer Undergraduate Mathematical Series, 2006.

[12] Garrett, P. *Poincaré-Birkhoff-Witt Theorem*, 2005. retrieved from www.math.umn.edu/garrett/ on 24/08/2009.

[13] Greuel, G.M., Pfister, G. *A Singular Introduction to Commutative Algebra.* Springer, 2008.

[14] Hawkins, T. *Emergence of the Theory of Lie Groups: An Essay in the History of Mathematics 1869-1926.* Springer-Varlag, 2000.

[15] Henrich, T. Private Correspondance. University Bonn, Autumn 2010.

[16] Knapp, A.W. *Elliptic Curves.* Princeton University Press, 1992.

[17] Lang, S. *Complex Analysis.* Springer-Verlag, 1999.

[18] Myrberg, P.J. *Über Systeme analytischer Functionen welche ein Additionstheorem besitzen: Preisschrift gekrönt und herausgegeben von der Fürstlich Jablonowskischen Gesellschaft zu Leipzig.* 1922.

[19] Polishchuk, A. Classical Yang-Baxter equation and the A_∞-constraint. *arXiv: math/0008156v1*, 2000.

[20] Range, M.R. *Complex Analysis: A Brief Tour into Higher Dimensions.* The American Mathematical Monthly. 110:89-108, 2003.

[21] Samelson, H. *Notes on Lie Algebras*, 1989. retrieved from www.math.cornell.edu/~hatcher/other/Samelson-LieAlg.pdf on 12/03/2009.

[22] Skylyanin, E.K. Quantum version of the method of inverse scattering problem. *Translated from Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR* 95:55-128, 1980.

[23] Stillwell, J. *Naive Lie Theory.* Springer Science and Business Media, 2008.

[24] Stolin, A. Constant Solutions of Yang-Baxter Equation for $\mathsf{sl}(2)$ and $\mathsf{sl}(3)$. *Math. Scand.*, 69:81-88, 1991.

[25] Stolin, A. On Rational Solutions of Yang-Baxter Equation for $\mathsf{sl}(n)$. *Math. Scand.*, 69:57-80, 1991.

Appendices

Appendix A

The ybe.lib Source Code

%

```
info="

   LIBRARY: ybe.lib
   AUTHOR: Lisa Kierans

   OVERVIEW:
   A library for testing possible solutions to the classical Yang-Baxter %
      equation of
   type rational, rational with two parameters, and trigonometric, and the
   associative Yang-Baxter equation of type 2 and 3 parameters.  The library %
      also
   includes a procedure for converting the value of the left hand side of the %
      YBE
   for non-solutions to LaTeX format. We can also compute the Casimir operator %
      for
   sl(n) and can check our procedures for the CYBE using the Casimir test %
      procedure.

   PROCEDURES FOR CREATING REQUIRED RINGS:

   createRingRational creates a ring from the Lie algebra entered with
                          additional parameter and variables.
   createRingRational2par  similar to rational except ring can handle two %
      parameters
   createRingTrig          creates a ring from the Lie algebra entered with
                          additional parameters and trig. variables
   createRingA2par         creates a ring with 2 parameters and variables %
      which are
                          matrix basis elements
   createRingA3par         creates a ring with 3 parameters and variables %
      which are
```

```
                        matrix basis elements

PROCEDURES FOR CYBE:

IsSolutionCYBE         checks if entered list is a solution to CYBE


PROCEDURES FOR THE AYBE:

IssolutionAYBE         checks if entered list is a solution to the AYBE


PROCEDURES FOR THE YBE:

texYBE                 converts the value of expressions that are not %
    solutions
                       to LaTex format.
showValue              shows the value of expressions that are not solutions.


PROCEDURES INVOLVING THE CASIMIR ELEMENT:

casimirZTest           returns a list corresponding to the rational %
    solution t/z,
                       where t is the Casimir element.
casimirZTest2par       returns a list corresponding to the rational solution
                       t/(z_1-z_2), where t is the Casimir element.
casimirEl              returns the Casimir element of some sl(n).

GENERAL PROCEDURES:

MakeLists
MakeTensor
NonCommutativePart
trigRules
totalAllBrackets
internalSolutionCYBE
parseCoefficient
matrixMult
commBracket
totalAYBE
dualBasis

KEYWORDS: Lie algebras, Classical Yang-Baxter Equation, Associative %
    Yang-Baxter
Equation, Casimir element.
";

////////////////////////////////////////////////////////////////////////////////
LIB "ncalg.lib";
LIB "latex.lib";
////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////
//PROCEDURES WHICH CREATE THE REQUIRED RING TO WORK IN FROM THE LIE ALGEBRA
////////////////////////////////////////////////////////////////////////////////

proc createRingRational(string LieAlg)
"USAGE:    createRingRational("LieAlg");LieAlg is a string.
 ASSUME:   - LieAlg is one of the pre-defined noncommutative algebras from %
    the
             library ncalg.lib
           - this is the Lie algebra which the possible solution is %
    contained in.
 RETURN:   The ring which is inputted is exported to be used globally.  A new
           ring is returned.  This new ring is similar to the non-commutative
           ring entered but is a commutative ring with additional %
    parameters 'u'
           and 'v'.  It also has a number of additional variables: each %
    variable
           from the original ring plus each of these variables with (1),(2) %
    and
           (3).  The variable 'z' is also added.
 REMARK:   The new ring is exported but cannot be set as the basering.  The %
    user
           must do this in order to proceed to checking solutions to CYBE.
 EXAMPLE:  example createRingRational; shows an example."
{

  def ratRing=internalnoncalg(1,LieAlg);
  return(ratRing);
}
example
{
  "EXAMPLE:";
  echo=2;
  def ratRing=createRingRational("sl(2)");
  setring ratRing;
  ratRing;
}

proc createRingRational2par(string LieAlg)
"USAGE:    createRingRational2PAR("LieAlg");LieAlg is a string.
 ASSUME:   - LieAlg is one of the pre-defined noncommutative algebras from %
    the
             library ncalg.lib
           - this is the Lie algebra which the possible solution is %
    contained in.
 RETURN:   The ring which is inputted is exported to be used globally.  A new
           ring is returned.  This new ring is similar to the non-commutative
           ring entered but is a commutative ring with additional parameters
           'u_1', 'u_2' and 'u_3'.  It also has a number of additional %
    variables:
           each variable from the original ring plus each of these variables
           with (1),(2) and (3).  The variables 'z_1' and 'z_2' are also %
    added.
 REMARK:   The new ring is exported but cannot be set as the basering.  The %
    user
           must do this in order to proceed to checking solutions to CYBE.
 EXAMPLE:  example createRingRational2par; shows an example."
{
  def ratRing2par=internalnoncalg(2,LieAlg);
  return(ratRing2par);
}
example
{
  "EXAMPLE:";
  echo=2;
  def ratRing2par=createRingRational("sl(2)");
  setring ratRing2par;
  ratRing2par;
}

proc createRingTrig(string LieAlg)
"USAGE:    createRingTrig("LieAlg");LieAlg is a string.
 ASSUME:   - LieAlg is one of the pre-defined noncommutative algebras from %
    the
             library ncalg.lib
           - this is the Lie algebra which the possible solution is %
    contained in.
 RETURN:   The ring which is inputted is exported to be used globally.  A new
           ring is returned.  This new ring is similar to the non-commutative
           ring entered but is a commutative ring with additional %
    parameters 'u'
           and 'v'.  It also has a number of additional variables: each %
    variable
           from the original ring plus each of these variables with (1),(2) %
    and
           (3).  The variables %
    'z','cos','sin','cos_u','cos_v','sin_u','sin_v'
           are also added.
 REMARK:   The new ring is exported but cannot be set as the basering.  The %
    user
           must do this in order to proceed to checking solutions to CYBE.
 EXAMPLE:  example createRingTrig; shows an example."
{
  def trigRing=internalnoncalg(3,LieAlg);
  return(trigRing);
}
example
{
  "EXAMPLE";
  echo=2;
  def trigRing=createRingTrig("sl(2)");
  setring trigRing;
  trigRing;
```

```
}

//The following procedure is used in each case: Rational, Rational-2 %
   parameter,
//and Trigonometric.

static proc internalnoncalg(int n,string LieAlg)
"USAGE:    internalnoncalg(n,"LieAlg"); n is an integer, LieAlg is a string.
 ASSUME:   LieAlg is taken from the procedures createRingRational/
           createRingRational2par/createRingTrig.  'n' is 1 for rational
           solutions, 2 for rational solutions with 2 parameters, or 3 for
           trigonometric solutions.
 RETURN:   a new ring with the characteristic and variables of the
           noncommutative ring entered along with several new characteristics
           and variables which are necessary in order to carry out %
   calculations.
 NOTE:     it is called from createRingRational,createRingRational2par, and
           createRingTrig" .
{
  if(defined(originalRing)==1){kill originalRing;}
         //we must first define the 'original ring' to be the %
   noncommutative Lie
        //algebra which was specified by the user.
execute("def originalRing=makeU"+LieAlg);
//for rational solutions:
if(n==1)
{
  //a ring is created with two new characters and the list of variables
  //created above, plus the variable 'z' is required so that the user
  //can input 'z' when giving the expression to check, the 'u' and 'v'
  //are required as they will appear in the value of the CYBE.
  execute("ring ringCreated=("+charstr(originalRing)+",u,v),("+varstr
          (originalRing)+",z),("+ordstr(originalRing)+");");
}
//for rational solutions with two variables:
if(n==2)
{
  //a ring is created with three new characters and the list of variables
  //created above, plus 'z_1' and 'z_2'.  Again, 'z_1' and 'z_2' can be
  //entered by the user, and 'u_1','u_2' and 'u_3' will be used in the
  //output value.
  execute("ring ringCreated=("+charstr(originalRing)+",u_1,u_2,u_3),("
          +varstr(originalRing)+",z_1,z_2),("+ordstr(originalRing)+");");

}
//for trigonometric solutions:
if(n==3)
{
  //a ring is created with two new characters along with additional
  //variables 'z',and the sine and cosine functions in 'z', 'u', and 'v'.
  execute("ring ringCreated=("+charstr(originalRing)+",u,v),("+varstr
          (originalRing)+",z,sin,cos,cos_u,sin_u,cos_v,sin_v,I),("+ordstr
          (originalRing)+");");
}
//we need to export the original ring so that we can call it in later
//procedures.
export(originalRing);
//we ring we have created is returned so that the user can then set this
//ring and check a possible solution which includes the new variables
//added.
return(ringCreated);
}

///////////////////////////////////////////////////////////////////////////
//PROCEDURES WHICH ARE USED TO CHECK SOLUTIONS
///////////////////////////////////////////////////////////////////////////

static  proc MakeLists(list #)
"USAGE:     MakeLists(#); # is a list.
 ASSUME:    # is divisable by 4, entries must follow the pattern numerator,
            denominator, first component of tensor, second component of %
    tensor.
 RETURN:    Four lists, a list of numerators, a list of denominators, a %
    list of
            the first components (a(i)) and a list of the second components
            (b(i)).
 NOTE:      it is called from internalSolutionCYBE."
{
        //define a counter and the lists to be returned.
int count=1; list num, den, a, b; int i;
//this loop goes through all entered data and groups it into four lists
  for(i=1; i<=size(#); i=i+4)
{
num[count] =#[i];
den[count]=#[i+1];
a[count]=#[i+2];
b[count]=#[i+3];
count++;
}
return(num,den,a,b);
}

//we now need a procedure which incorporates the rules of the Lie bracket %
    of two
//elements of L'tensor'L'tensor'L.

static proc MakeTensor(list Tensors,RING)
"USAGE:     MakeTensor(a,b,c,RING); a,b,c are polynomials, RING is the %
    basering.
 ASSUME:    input is 3 single variable from the variables entered by the %
    user.
            These are common to both the ring that the user sets to begin %
    with,
            and the internal ring used for calculations.  That is, the %
```

```
   variables
            without bracketed numbers.
 RETURN:    gives each element entered as a product but with first element %
   +(1),
            second element +(2) and third element +(3).  For example
            a(1)*b(2)*c(3).
 NOTE:      it is called from totalAllBrackets."
{
        //create lists for first, second and third components of our 'tensor
        //product'.
list firstComponent,secondComponent,thirdComponent; int i; int count=1;

//this for loop adds the correct variables to each list.  We start our
//counting at the variable after the last variable common to both rings.
for(i=1; i<=nvars(originalRing); i=i+1)
{
        //this list includes all the variables with (1):
firstComponent[count]=varstr(RING,(nvars(originalRing)+i));
//this list includes all the variables with (2):
secondComponent[count]=varstr(RING,((nvars(originalRing)*2)+i));
//this list includes all the variables with (3):
thirdComponent[count]=varstr(RING,((nvars(originalRing)*3)+i));
count++;
}
//we will now define three polynomials to be the three variables input
//into the procedure.
poly firstPolynomial=Tensors[1];
poly secondPolynomial=Tensors[2];
poly thirdPolynomial=Tensors[3];
//we use the command 'map' to map the entered variables to the same
//variable but with a (1) on the first variable entered, a (2) on the
//second variable entered and a (3) on the third variable entered.
execute("map firstMap=RING,"+string(firstComponent)+";");
execute("map secondMap=RING,"+string(secondComponent)+";");
execute("map thirdMap=RING,"+string(thirdComponent)+";");
//we now return the product of these three new variables.  Note that our
//ordering plays an important role as we need the bracketed numbers to
//ascend chronologically.  This also ensures that any integer
//coefficient is placed in front of the three variables. This mimics the
//rules for tensor product.
return(firstMap(firstPolynomial)*secondMap(secondPolynomial)*thirdMap
                                                           %
   (thirdPolynomial));
}
//the next procedure is where we make use of our noncommutative ring.  We use
//this procedure to calculate the three Lie brackets of CYBE.  We enter the
//elements of the tensors to be bracketed as a list of 4 variables.

static proc NonCommutativePart(poly a,poly b,poly c,poly d)
"USAGE:     NonCommutativePart(a(i),b(i),a(j),b(j)); a(i) and b(i) are the
            elements of the first tensor and a(j) and b(j) are the elements %
   from
```

```
           the second tensor.
 ASSUME:    variables entered are single variables from those common to both
           rings.
 RETURN:    returns a list of 3 lists with 3 elements in each list.  List 1 %
    has:
           [a(i),a(j)],b(i),b(j).  List 2 has: a(i),[b(i),a(j)],b(j). List 3
           has: a(i),a(j),[b(i),b(j)].
 NOTE:      it is called from totalAllBrackets."
{
        //define our basering with a name to be used in the procedure.
        def SolRing=basering;
//create a list from the inputted variables.
        list INPUT=a,b,c,d;
//now change to basering to the  noncommutative ring exported from
//earlier procedure.
setring originalRing;
//then we create a list which maps each of the entered variables to the
//same variable in the original ring.(Recall that variables entered must
//be from those common to both rings).
list L=imap(SolRing,INPUT);
poly LieBracket; list A1213, A1223, A1323;
//the bracket command in Singular computes the Lie bracket between two
//variables.
LieBracket=bracket(L[1],L[3]);
//we now create a list of three elements based on the fact that
//[(a'tensor'b)^(12),(c'tensor'd)^(13)]=[a,c]'tensor'b'tensor'd
A1213=LieBracket,L[2],L[4];
LieBracket=bracket(L[2],L[3]);
//similarly, we create a list of three elements for the second bracket
//of CYBE: [(a'tensor'b)^(12),(c'tensor'd)^(23)]=a'tensor'[b,c]'tensor'd
A1223=L[1],LieBracket,L[4];
LieBracket=bracket(L[2],L[4]);
//we alse create a list of three elements for the third bracket of CYBE:
//[(a'tensor'b)^(13),(c'tensor'd)^(23)]=a'tensor'c'tensor'[b,d]
A1323=L[1],L[3],LieBracket;
//we must change back to the commutative ring as we need to return
//values in the same ring from which they were entered.
setring SolRing;
list result;
//we will create a list of lists to map our results to the commutative
//ring.
result[1]=imap(originalRing,A1213);
result[2]=imap(originalRing,A1223);
result[3]=imap(originalRing,A1323);
//we return a list of three lists.
return(result);
}
//we now need to create a procedure that can cope with the rules required for
//trigonometric solutions.  The identities: %
    sin(u+v)=sin(u)*cos(v)+cos(u)*sin(v)
//and cos(u+v)=cos(u)*cos(v)-sin(u)*sin(v) are incorporated using this %
    procedure.
```

```
//We also overcome the problem of having variables in the denominator here.
static proc trigRules(poly Total)
"USAGE:     trigRules(Total); Total a polynomial.
 ASSUME:    Total is a polynomial from the procedure totalAllBrackets in the
            temporary trigonometric ring set up there.
 RETURN:    A polynomial incorporating the trigonometric addition formulae.
 REMARK:    The returned polynomial is not the correct value for the
            trigonometric CYBE as we have to disregard the denominator of %
    each
            coefficient for the procedure to work.
 NOTE:      it is called from totalAllBrackets."
{
        //introduce necessary variables.
        int i; number Numerat; poly trigVariables=0; poly trig,reducetrig;
//we create a loop which takes each element of the polynomial entered
//one-by-one
for(i=1; i<=size(Total); i++)
        {
  //define the numerator of the coefficient of the first term in the
  //polynomial 'Total'
  Numerat=numerator(leadcoef(Total));
  //then take this numerator and substitute for the parameters, the
  //correct variables but incorporating the addition formulae for sine
  //and cosine.
  trig=subst(Numerat,cu,cos_u,cv,cos_v,su,sin_u,sv,sin_v,sadd,
            %
    (sin_u*cos_v+cos_u*sin_v),cadd,(cos_u*cos_v-sin_u*sin_v),complexi,I);
  //the SINGULAR command 'reduce' reduces the polynomial 'trig' to its
  //normal form with respect to the ideal representing the trigonometric
  //rule sin^2+cos^2=1.
  reducetrig=reduce(trig,std(0));
  //create a new polynomial, the addition of these elements.
  trigVariables=trigVariables+reducetrig*leadmonom(Total);
  //subtract the leading term from the polynomial 'Total' so that we can
  //now work on the next term, i.e. the new leading term.
  Total=Total-lead(Total);
}
//what is returned is the polynomial entered disregarding all denominators
//and incorporating all trigonometric rules which apply.
return(trigVariables);
}

static proc totalAllBrackets(int TYPE, list Num, list Den, list A, list B)
"USAGE:     totalAllBrackets(TYPE,Num,Den,A,B); TYPE is a number; Num,Den,A,B
            are lists.
 ASSUME:    TYPE can be 1,2 or 3 depending on ring.  Num is the list of
            numerators for each tensor, Den is the list of denominators for %
    each
            tensor, A is the list of  first tensor elements and B is the %
    list of
            second tensor elements.
 RETURN:    The value of the CYBE is returned.
 NOTE:      it is called from internalSolCYBE."
{
  //create a list of the lists entered.
  list INPUT=Num,Den,A,B;
  //if we are working in the Trigonometric ring, we need to work in a ring %
    where
  //sin,cos,sin(u+v) and cos(u+v) are parameters so they can be part of the
  //denominator of coefficients.
if(TYPE==3)
  {
    //the SINGULAR procedure 'ringlist' creates a list of all the rings
    //properties.
    list baseRing=ringlist(internalRing);
    //the first object in this list, _[1], is the characteristic of the
    //ring.  We need to change this to have additional parameters
    //symbolising cos(u),cos(v),sin(u),sin(v),sin(u+v) and cos(u+v).
    baseRing[1]=list(0,list("u","v","cu","cv","su","sv","sadd","cadd",
                                         %
    "complexi"),list(list("lp",1)),ideal(0));
    //define a temporary trigonometric ring to have this new character
    //string.
    def tempTrigRing=ring(baseRing);
    setring tempTrigRing;
    //we must now incorporate the formulas for cos^2+sin^2 as an ideal
    //in this temporary ring.
    ideal trigRule=sin_u^2+cos_u^2-1,sin_v^2+cos_v^2-1, I^2+1;
    //and now define our 'solution ring' to be the quotient ring of the
    //temporary ring modulo the ideal 'trigRule'.
    qring SolRing=std(trigRule);
  }
//if we are not working in the trigonometric ring then we define the
//ring to be called 'SolRing' so we can continue using one ring name for
// all original ring types.
else{def SolRing=internalRing;}
setring SolRing;
//introduce necessary variables.
 int i,j; poly r12r13,r12r23,r13r23,numer1,denom1,numer2,denom2, Total;
 poly T1=0; poly T2=0; poly T3=0; list Tensors, INPUT1,num,den,a,b;
 //we have changed rings within the procedure so we need to map our
 //input to this new ring.
INPUT1=imap(internalRing,INPUT);
num=INPUT1[1]; den=INPUT1[2]; a=INPUT1[3]; b=INPUT1[4];
//we now create two loops that will go through all combinations of pairs
//of tensors (a(i),b(i),a(j),b(j)).
for(i=1; i<=size(a); i=i+1)
{
for(j=1; j<=size(a); j=j+1)
{
  //define polynomials to be each element in the list of
  //numerators and denominators.
  numer1=num[i]; denom1=den[i]; numer2=num[j]; denom2=den[j];
  //this list gives the three variables computed from the first
  //Lie bracket of the CYBE.
  Tensors=NonCommutativePart(a[i],b[i],a[j],b[j])[1];
  //this list makes these variables into a 'tensor product'.
  r12r13=MakeTensor(Tensors,SolRing);
  //if we are working in a rational ring:
  if(TYPE==1)
    {
      //we multiply by the coefficient of each tensor,
      //substituting 'u' for 'z' in the first tensor, and 'u+v'
      //for 'z' in the second tensor.
      r12r13=r12r13*(subst(numer1,z,u)/subst(denom1,z,u))*
                       (subst(numer2,z,u+v)/subst(denom2,z,u+v));
    }
          //if we are working in a 2 parameter rational ring:
  if(TYPE==2)
    {
      //we multiply by the coefficient of each tensor,
      //substituting 'u_1' for 'z_1' and 'u_2' for 'z_2' in the first
      //tensor, and 'u_1' for 'z_1' and 'u_3' for 'z_2' in the second
      //tensor.
      r12r13=r12r13*(subst(numer1,z_1,u_1,z_2,u_2)/subst
     (denom1,z_1,u_1,z_2,u_2))*(subst(numer2,z_1,u_1,z_2,u_3)
      /subst(denom2,z_1,u_1,z_2,u_3));
    }
  //if we are working in a trigonometric ring:
  if(TYPE==3)
   {
 //we multiply by the coefficient of each tensor, substituting
 //'u' for 'z',cos and sin of 'u' for cos and sin in the first
 //tensor, and 'u+v' for 'z', cos and sin of (u+v) for cos and
 //sin in the second tensor.
      r12r13=r12r13*(subst(numer1,z,u,cos,cu,sin,su,I,complexi)
   %
    /subst(denom1,z,u,cos,cu,sin,su,I,complexi))*(subst(numer2,z,u+v,cos,cadd,sin,
              %
    sadd,I,complexi)/subst(denom2,z,u+v,cos,cadd,sin,sadd,I,complexi));
    }
  //this polynomial is the addition of all elements of the first
  //bracket of the CYBE.
  T1=T1+r12r13;
  //similarly, we calculate the second bracket of the CYBE.
  Tensors=NonCommutativePart(a[i],b[i],a[j],b[j])[2];
  r12r23=MakeTensor(Tensors,SolRing);
  if(TYPE==1)
    {
      r12r23=r12r23*(subst(numer1,z,u)/subst(denom1,z,u))*
                             (subst(numer2,z,v)/subst(denom2,z,v));
    }
  if(TYPE==2)
    {
      r12r23=r12r23*(subst(numer1,z_1,u_1,z_2,u_2)/subst(denom1,z_1,
                %
    u_1,z_2,u_2))*(subst(numer2,z_1,u_2,z_2,u_3)/subst(denom2,z_1,u_2,z_2,u_3));
    }
  if(TYPE==3)
    {
      r12r23=r12r23*(subst(numer1,z,u,cos,cu,sin,su,I,complexi)
       %
    /subst(denom1,z,u,cos,cu,sin,su,I,complexi))*(subst(numer2,z,v,cos,cv,sin,
                   %
    sv,I,complexi)/subst(denom2,z,v,cos,cv,sin,sv,I,complexi));
    }
  //this polynomial is the addition of all elements of the second
  //bracket of the CYBE
  T2=T2+r12r23;
  //we use the same method to obtain the third bracket of the
  //CYBE
  Tensors=NonCommutativePart(a[i],b[i],a[j],b[j])[3];
  r13r23=MakeTensor(Tensors,SolRing);
  if(TYPE==1)
    {
      r13r23=r13r23*(subst(numer1,z,u+v)/subst(denom1,z,u+v))*
                                        %
    (subst(numer2,z,v)/subst(denom2,z,v));
    }
  if(TYPE==2)
    {
      r13r23=r13r23*(subst(numer1,z_1,u_1,z_2,u_3)/subst(denom1,z_1,
               %
    u_1,z_2,u_3))*(subst(numer2,z_1,u_2,z_2,u_3)/subst(denom2,z_1,u_2,z_2,u_3));
    }
  if(TYPE==3)
    {
      r13r23=r13r23*(subst(numer1,z,u+v,cos,cadd,sin,sadd,I,
complexi)/subst(denom1,z,u+v,cos,cadd,sin,sadd,I,complexi))*(subst(numer2,z,v, %
           %
    cos,cv,sin,sv,I,complexi)/subst(denom2,z,v,cos,cv,sin,sv,I,complexi));
    }
  //this polynomial is the addition of all elements of the second
  //bracket of the CYBE
  T3=T3+r13r23;
}
}
//this polynomial is the total value of the CYBE for the expression
//entered by the user.
Total=T1+T2+T3;
//we must now incorporate the trigonometric rules for sin(u+v) and
//cos(u+v) using the procedure trigRules.
   if(TYPE==3)
   {
     Total=trigRules(Total);
   }
   //we need to return a value in the original basering defined at the %
    beginning
   //of the procedure.
     setring internalRing;
     //and map the total value to a polynomial in this ring
     poly TOTAL=imap(SolRing,Total);
     return(TOTAL);
}

///////////////////////////////////////////////////////////////////////////////
//PROCEDURES FOR THE CYBE
///////////////////////////////////////////////////////////////////////////////

static proc internalSolutionCYBE(int TYPE, list #)
"USAGE:     internalSolutionCYBE(TYPE,#); TYPE is a number, # is a list.
 ASSUME:    TYPE is 1,2 or 3 depending on the type of ring we are working %
    in. #
            is the list of the possible solution to be checked in the format:
            numerator,denominator, first tensor,second tensor.
 RETURN:    a string of text which tells the user whether the inputted %
    variables
            are a solution to the CYBE or not.  IF the expression is not a
            solution to the CYBE, then the value is exported.
 NOTE:      it is called from IsSolutionCYBE."
{
  //because we need to export 'internalRing','lhsOfEquation', we must %
    undefine
  //them if they are already defined in order to avoid SINGULAR printing a
  //message "redefining ..."
 if(defined(internalRing)==1){kill internalRing; }
 if(defined(lhsOfEquation)==1){kill lhsOfEquation; }
 //we first need to create a ring to be used internally.  This ring will %
    make it
 //possible to mimic the rules of tensor product of Lie algebras.
        //we will create a list of new variables from the variables of the
        //original ring. This list 'new_vars' will be the variable list for %
    our
        //new internal ring.
list new_vars; int count=1;int i,j;
//this loop runs through each variable from the original noncommutatuve
//ring and adds it to the list first (n variables).
for(i=1; i<=nvars(originalRing); i=i+1)
{
        //the SINGULAR procedure 'varstr' gives the name of the i-th
        //ring variable
new_vars[count]=varstr(originalRing,i);
count++;
}
//we now need two 'for' loops to add 3n new variables made from each of
//the original variables with a (1),(2), and (3) added to each one.  The
//first loop orders the variables correctly for use in preceeding
//procedures.
for(j=1; j<=3; j++)
{
```

```
for(i=1; i<=nvars(originalRing); i++)
{
new_vars[count]=varstr(originalRing,i)+"("+string(j)+")";
count++;
}
}
//we now need to make four lists out of the variables entered by the
//user to separate the numerator, denominator, 1st tensor, and 2nd tensor
list listOfPossibleSol=MakeLists(#);
//define the basering as 'SolutionRing'.  This will be the ring any
//output is given in and has the same name for each solution type.
def SolutionRing=basering;
//use the SINGULAR command 'ringlist' to create a list of the rings
//properties.
list newRing=ringlist(SolutionRing);
//we now use a 'for' loop to add all the new variables created above to this
   //SolutionRing.
  for(i=1; i<=size(new_vars); i=i+1)
    {
      newRing[2][i]=new_vars[i];
    }
//if we are working in the rational ring add only the variable 'z' again.
  if(TYPE==1){ list extraVars=z;}
  //if we are working in the 2 parameter rational ring, we must create a list
     //with the appropriate variables to add them to our new ring.
  if(TYPE==2){  list extraVars=z_1,z_2;}
  //if we are working in the trigonometric ring, we must add the %
    appropriate variables
  if(TYPE==3){  list extraVars=z,sin,cos,sin_u,cos_u,sin_v,cos_v,I;}
  newRing[2]=newRing[2]+extraVars;
  //rename this new ring 'internalRing'.  It will be used for all internal
     //calculations
  def internalRing=ring(newRing);
setring internalRing;
list listOfPossibleSolution=imap(SolutionRing,listOfPossibleSol);
//we need to export this internal ring so that it is defined globally and %
    can be called in any
//internal procedure.
export(internalRing);
//the polynomial 'TOTAL' is the value of the CYBE.
poly TOTAL=totalAllBrackets(TYPE,listOfPossibleSolution[1],
listOfPossibleSolution[2],listOfPossibleSolution[3],listOfPossibleSolution[4]);

//if the expression that was checked is a solution to the CYBE
if(TOTAL==0)
{
        //we must change back to our solution ring before we can return %
    anything.
        setring SolutionRing;
return("Is a solution to the CYBE");
}
//if the expression that was checked is not a solution to the CYBE
```

```
else
{

  //save the size of the polynomial 'TOTAL'.
  def p=size(TOTAL);
  //save the number of variables in the original (non-commutative) ring.
  def b=nvars(originalRing);
  //create several objects which are lists of integers
  intvec T,tensor1,tensor2,tensor3; count=1;
  //we need a loop that continues until the polynomial TOTAL is equal to %
    zero.
  while(TOTAL<>0)
    {
      //the SINGULAR command 'leadexp' returns the exponent vector of the %
    leading
      //monomial of a polynomial.  The result of the following command will %
    be a
      //vector of size 4b (b is from above).  This vector will have 1's %
    where the
      //first, second and third tensors are found.
      T=leadexp(TOTAL);
      //to find the first tensor, we look at the vector given by the %
    elements of
      //the variable string with a (1) added.
      tensor1=T[(b+1)..2*b];
      //to find the second tensor, we look at the vector given by the %
    elements of
      //the variable string with a (2) added.
      tensor2=T[(2*b+1)..3*b];
      //to find the third tensor, we look at the vector given by the %
    elements of
      //the variable string with a (3) added.
      tensor3=T[(3*b+1)..4*b];
      //we now create a number of lists.  Each list will have numerator, %
    denominator,
      //tensor1, tensor2, and tensor3 from each term of the polynomial %
    TOTAL.  The
      //tensors will be from the original list (i.e. without brackets).
      //The SINGULAR command 'monomial' effectively reverses the command %
    'leadexp'.
      //What we have done here is factorised the polynomial TOTAL into lists.
      execute("list Factorised(count)=numerator(leadcoef(TOTAL)),
                    denominator(leadcoef(TOTAL)),"
    +string(monomial(tensor1))+","
    +string(monomial(tensor2))+","
    +string(monomial(tensor3))+";");

      count++;
      //we need to subtract the leading term from TOTAL.
    TOTAL=TOTAL-lead(TOTAL);
  }
  //so that we can return values, we need to convert back to the Solution %
```

```
    ring
  setring SolutionRing;
  //we define a string to hold the value of non-solutions
  string lhsOfEquation="";
  //we cannot return values of trigonometric expressions
  if(TYPE==3)
    {
      lhsOfEquation=lhsOfEquation+"Cannot return the value of
                                                   trigonometric %
    expressions";
    }
  else
    {
      //we need a 'for' loop that is the size of the polynomial TOTAL.
      //This is the same as the number of lists defined above called
      //'Factorised(i)'
      for(i=1; i<=number(p); i=i+1)
{
  //we need to undefine the lists 'solInputRing(i)' if they have
  //been previously defined to avoid getting a message from
  //SINGULAR "redefining ..."
  if(defined(solInInputRing(i))==1){kill solInInputRing(i);}
  //create a number of lists which map the lists defined above
  //to the solution ring
  list solInInputRing(i)=imap(internalRing,Factorised(i));
  export(solInInputRing(i));
  //the value of the non-solution is made into a string.  It is
  //in the format: numerator, denominator, tensor1, tensor2,
  //tensor3, [linebreak] etc.
  lhsOfEquation=lhsOfEquation+string(solInInputRing(i))+newline;
}
    }
        //this string is then exported to be used in the procedure %
    'showValue'
export(lhsOfEquation);
return("Is not a solution to the CYBE");
      }
}

proc IsSolutionCYBE(list #)
"USAGE:     IsSolutionCYBE(#); # is a list.
 ASSUME:    # is the list of the possible solution to be checked in the %
    format:
            numerator,denominator, first tensor,second tensor.  The %
    elements of
            this list are from the basering created.
 RETURN:    a string of text which tells the user whether the inputted %
    variables
            are a solution to the CYBE or not.  If the expression is not a
            solution to the CYBE, if the expression is not a solution, then %
    the
            value is exported.
 EXAMPLE:   example IsSolutionCYBE; shows an example."
{
  //the input must be grouped into lists of size 4.
  if(size(#)%4!=0){ERROR("The input must be a list entered as: numerator,
          denominator,variable1,variable2.  See help YBE for more %
    information");}
  else
    {
       int TYPE=1;
       if(find(varstr(basering),"z_1")){TYPE=2;}
       if(find(varstr(basering),"sin")){TYPE=3;}
        //create a list of the list entered.  We need to have a name for this
       //list so it can be used in the procedure 'internalSolutionAYBE'.
      list SOL=#;
      return(internalSolutionCYBE(TYPE,SOL));
    }
}
example
{
  "EXAMPLE:";
  echo=2;
  def ratRing=createRingRational("sl(2)");
  setring ratRing;
  IsSolutionCYBE(1,z,e,f,\
  1,z,f,e,\
  1,z,1/2h,h);
  def trigRing=createRingTrig("sl(2)");
  setring trigRing;
  IsSolutionCYBE(cos,2*sin,h,h,\
  1,sin,e,f,\
  1,sin,f,e,\
  sin,1,f,f);
}

////////////////////////////////////////////////////////////////////////////
//PROCEDURES FOR THE AYBE
////////////////////////////////////////////////////////////////////////////

//this procedure creates a ring with variables e_1_1,....e_1_n,.....e_n_n + %
    each
//variable with (1),(2),and(3) as in previous procedures

proc createRingA3par(int sqrtDim)
"USAGE:     createRingA3par(sqrtDim); sqrtDim is an integer.
 ASSUME:    n^2 is the dimension of the ring to be created.
 RETURN:    a ring is returned with variables: the (n x n) basis elements; %
    z, y_1,
            y_2, plus the parameters: u; v; x_1; x_2; x_3.
 REMARK:    The new ring is exported but cannot be set as the basering.  The
            user must do this in order to proceed to checking solutions to %
    AYBE.
 EXAMPLE:   example createRingA3par; shows an example."
```

```
{
  def assocRing3par=internalAssocRing(1,sqrtDim);
  return(assocRing3par);
}
example
{
  "EXAMPLE:"; echo=2;
  def assocRing3=createRingA3par(3);
  setring assocRing3;
  assocRing3;
}

proc createRingA2par(int sqrtDim)
"USAGE:     createRingA2par(sqrtDim); sqrtDim is an integer.
 ASSUME:    n^2 is the dimension of the ring to be created.
 RETURN:    a ring is returned with the variables:(n x n) basis elements; %
    z,w,
            plus the parameters: u; v; x; y.
 REMARK:    The new ring is exported but cannot be set as the basering.  The
            user must do this in order to proceed to checking solutions to %
    AYBE.
 EXAMPLE:   example createRingA2par; shows an example."
{
  def assocRing2par=internalAssocRing(2,sqrtDim);
  return(assocRing2par);
}
example
{
  "EXAMPLE:"; echo=2;
  def assocRing2=createRingA2par(3);
  setring assocRing2;
  assocRing2;
}

static proc internalAssocRing(int n, int sqrtDim)
"USAGE:    internalAssocRing(n,sqrtDim); n and sqrtDim are integers.
 ASSUME:   sqrtDim is taken from the procedures %
    createRingA2par/createRingA3par
           n is 1 for 3 parametric associative solutions, 2 is for 2 %
    parametric
           associative solutions.
 RETURN:   a new ring  with several new characteristics and variables which %
    are
           necessary in order to carry out calculations.
 NOTE:     it is called from createRingA2par and createRingA3par" .
{

      //because we export the original ring and the value 'n' we need to
      //undefine it (if defined) at the beginning of the procedure.
      if(defined(dimension)==1){kill dimension;}
      if(defined(originalRing)==1){kill originalRing;}
        //we make provisions for entering an integer <=1.
if(sqrtDim<2)
{
print("incorrect input");
return(0);
}
//introduce necessary variables.
int i,j; string originalVariables="";
//we need to create a string with (n x n) variables to become our
//variable list.
for(i=1; i<=sqrtDim; i++)
{
for(j=1; j<=sqrtDim; j++)
{
  //if this is not the first element of the variable list, we
  //add a comma.
if(originalVariables!="")
  {
    originalVariables= originalVariables + ", ";
  }
//add each element of our variable string
originalVariables=originalVariables + "e_"+string(i)+"_"
                                                                    %
    +string(j);
}
}
// define original ring as ring with char 0 and variables from string
//created above
execute("ring originalRing=(0),("+originalVariables+"),dp;");
execute("ring assocRing2par=(0,u,v,x,y),("+originalVariables+",z,w),dp;"
                                                                           %
      );
execute("ring assocRing3par=(0,u,v,x_1,x_2,x_3), ("+originalVariables+",z,
                                                                  %
    y_1,y_2),dp;");
//we export the original ring and the value of 'n' to be used globally,
//and return this new ring .
export(originalRing);
int dimension=sqrtDim;
export(dimension);
if(n==1){return(assocRing3par); }
if(n==2){return(assocRing2par); }
}

static proc matrixMult
"USAGE:     matrixMult(e_r_s,e_k_l); e_r_s and e_k_l are polynomials, n is an
            integer.
 ASSUME:    e_r_s and e_k_l are single variable polynomials, which are basis
            elements of the
            associative algebra we are working in.'n' is globally defined %
    in the
            procedure createRingAYBE.
 RETURN:    returns the value of the two matrices multiplied.
 NOTE:      called from procedure commBracket."
{
        poly matrixmult;  int %
    variablePosition_1,variablePosition_2,S,L,r,s,k,l;
        //we use the position of each variable in the ring's variable %
    string to
//determine its value.
variablePosition_1=rvar(#[1]);
//the value of 's' is the remainder on division by 'n'.
S=variablePosition_1%dimension;
//if the remainder is zero then the value of 's' is equal to 'n'.
if(S==0){s=dimension;}
//otherwise
else {s=S;}
//the value of 'r' is found using the following formula:
r=(variablePosition_1-s)/dimension + 1;
//we now use the position of the second variable to find its value.
variablePosition_2=rvar(#[2]);
L=variablePosition_2%dimension;
if(L==0){l=dimension;}
else {l=L;}
//the following formula gives us the value of 'k'
k=(variablePosition_2-l)/dimension + 1;
//using the rules of matrix multiplication, we know that e_r_s*e_k_l=
//e_r_l IF s=k
if(s==k)
{
execute("matrixmult = e_"+string(r)+"_"+string(l)+";");
}
// otherwise the value is zero
else
{
matrixmult = 0;
}
//we return this polynomial.
return(matrixmult);
}

static proc commBracket(poly a,poly b,poly c,poly d)
"USAGE:    commBracket(a(i),b(i),a(j),b(j)); a(i) and b(i) are elements of %
    the
          first tensor and a(j) and b(j) are elements of the second tensor.
 ASSUME:   variables entered are single variables from those common to both
          rings.
 RETURN:   returns a list of three lists with three elements in each list.  %
    List
          1 has a(i)*a(j), b(i),b(j). List 2 has a(i),b(i)*a(j),b(j). List 3
          has a(i),a(j),b(i)*b(j).
 NOTE:     it is called from totalAYBE."
{
     list A1223, A1312, A2313; poly MM;
//we work out the first bracket of the AYBE
```

```
MM=matrixMult(b,c);
     A1223=a,MM,d;
//then the second bracket of the AYBE
MM=matrixMult(a,c);
     A1312=MM,d,b;
//then the third bracket of the AYBE
MM=matrixMult(b,d);
     A2313=c,a,MM;
list result;
     result[1]=A1223;
     result[2]=A1312;
     result[3]=A2313;
     return(result);
}

static proc totalAYBE(int TYPE, list num,list den,list a, list b)
"USAGE:     totalAYBE(TYPE,num,den,a,b); TYPE is a number; num, den, a and b
            are all lists.
 ASSUME:    TYPE can be 1 or 2 depending on the ring we are working in.  %
    num is
            the list of numerators, den is the list of denominators, a and %
    b are
            the tensors.  Input is from the procedure makeLists.
 RETURN:    the value of the AYBE is returned.
 NOTE:      it is called from internalAYBE."
{
        //as the procedure makeTensor uses a ring in the input, we must %
    define
        //our basering def solRing=internalRing;
//introduce necessary variables.
int i,j; poly r12r23,r13r12,r23r13,numer1,denom1,numer2,denom2, Total;
        poly T1=0; poly T2=0; poly T3=0;list Tensors;
//we will now create two loops that will go through all the combinations
//of pairs of tensors.
for(i=1; i<=size(a); i=i+1)
{
for(j=1; j<=size(a); j=j+1)
{
        //set the numerator and denominator for each loop.
numer1=num[i]; denom1=den[i]; numer2=num[j];
                        denom2=den[j];
//this list gives the three variables computed from the
//first bracket of the AYBE.
Tensors=commBracket(a[i],b[i],a[j],b[j])[1];
r12r23=MakeTensor(Tensors,internalRing);
//if we are working in a 3 parameter ring:
if(TYPE==1)
  {
    r12r23=r12r23*(subst(numer1,z,u,y_1,x_1,y_2,x_2)/subst
                %
    (denom1,z,u,y_1,x_1,y_2,x_2))*(subst(numer2,z,u+v,y_1,x_2,y_2,x_3)/subst
                                                        %
```

```
      (denom2,z,u+v,y_1,x_2,y_2,x_3));
    }
  //if we are working in a 2 parameter ring
  if(TYPE==2)
    {
      r12r23=r12r23*(subst(numer1,z,u,w,x)/subst(denom1,z,
                           %
      u,w,x))*(subst(numer2,z,u+v,w,y)/subst(denom2,z,u+v,w,y));
    }
  T1=T1+r12r23;
  Tensors=commBracket(a[i],b[i],a[j],b[j])[2];
  //this list makes the variables into a tensor product
  //format.
  r13r12=MakeTensor(Tensors,internalRing);
  //if we are working in a 3 parameter associative ring:
  if(TYPE==1)
    {
      //we multiply by the coefficient of each tensor with
      //appropriate substitution.
      r13r12=r13r12*(subst(numer1,z,u+v,y_1,x_1,y_2,x_3)/
            %
      subst(denom1,z,u+v,y_1,x_1,y_2,x_3))*(subst(numer2,z,-v,y_1,x_1,y_2,x_2)/subst(
                                                           %
      denom2,z,-v,y_1,x_1,y_2,x_2));
    }
  //if we are working in a 2 parameter associative ring:
  if(TYPE==2)
    {
      //we multiply by the coefficient of each tensor with
      //appropriate substitution.
      r13r12=r13r12*(subst(numer1,z,u+v,w,x+y)/subst
            %
      (denom1,z,u+v,w,x+y))*(subst(numer2,z,-v,w,x)/subst(denom2,z,-v,w,x));
    }
  //the polynomial T1 is the addition of all elements of
  //the first bracket of the AYBE
  T2=T2+r13r12;
  //Similarly, we find the value of the second bracket of
  //the AYBE and the value of the third bracket of the AYBE
  Tensors=commBracket(a[i],b[i],a[j],b[j])[3];
  r23r13=MakeTensor(Tensors,internalRing);
  if(TYPE==1)
    {
      r23r13=r23r13*(subst(numer1,z,v,y_1,x_2,y_2,x_3)/subst
                 %
      (denom1,z,v,y_1,x_2,y_2,x_3))*(subst(numer2,z,u,y_1,x_1,y_2,x_3)/subst(
                                                            %
      denom2,z,u,y_1,x_1,y_2,x_3));
    }
  if(TYPE==2)
    {
      r23r13=r23r13*(subst(numer1,z,v,w,y)/subst(denom1,z,
                        %
     v,w,y))*(subst(numer2,z,u,w,x+y)/subst(denom2,z,u,w,x+y));
    }
  T3=T3+r23r13;
  }
  }
  //we add the above three polynomials together to give us the total value
  //of the AYBE
  Total=T1-T2-T3;
  return(Total);
  }

  static proc internalSolutionAYBE(int TYPE, list #)
  "USAGE:     internalSolutionAYBE(TYPE,#); TYPE is a number, # is a list.
   ASSUME:    TYPE is either 1 or 2 depending on the type of ring we are %
     working
              in.  # is the list of the possible solution in the format: %
     numerator,
              denominator tensor1, tensor2.  The elements of this list are from
              the variable list of the associative ring created.
   RETURN:    a string of text which tells the user whether or not the %
     expression
              entered as a list is a solution to the AYBE.  If the expression %
     is
              not a solution, then the value of the LHS of the AYBE is %
     exported.
   NOTE:      it is called from IsSolutionAYBE."
  {

    //as we export 'lhsOfEquation' and 'internalRing', we must undefine them, %
     if
    //defined, at the beginning of the procedure:
    if(defined(lhsOfEquation)==1){kill lhsOfEquation; }
    if(defined(internalRing)==1){kill internalRing;}
          //we first need to create a ring to be used internally.  This ring %
     will
          //make it possible to mimic the rules of tensor product of algebras.
          //we will create a list of new variables from the variables of the
          //original ring .  This list 'new_vars' will be the variable list for
          //our new internal ring.
          list new_vars; int count=1;int i,j;
         //this loop runs through each variable from the original ring and adds
  //it to the list first.
  for(i=1; i<=nvars(originalRing); i=i+1)
  {
    //the SINGULAR procedure 'varstr' gives the name of the i-th ring
    //variable
  new_vars[count]=varstr(originalRing,i);
  count++;
  }
          //we now need two 'for' loops to add 3n new variables made from %
     each of
```

```
//the original variables with a (1),(2), and (3) added to each one.  The
//first loop orders the variables correctly for use in preceeding
//procedures.
for(j=1; j<=3; j++)
{
for(i=1; i<=nvars(originalRing); i++)
{
new_vars[count]=varstr(originalRing,i)+"("+string(j)+")";
count++;
}
}
//we now define a new ring to be used internally with the basis elements
// plus each one with a (1),(2) or (3) added.
//we now need to make four lists out of the variables entered by the user
//to separate the numerator, denominator, 1st tensor, and 2nd tensor
list listOfPossibleSol=MakeLists(#);
//define the basering as 'SolutionRing'.  This will be the ring any
//output is given in and has the same name for each solution type.
def SolutionRing=basering;
//create a list of the properties of this list.
list newRing=ringlist(SolutionRing);
//use a 'for' loop to add all the new variables created above to this
//SolutionRing
for(i=1; i<=size(new_vars); i=i+1)
    {
      newRing[2][i]=new_vars[i];
    }
    //now add additional variables again
//if we are working in 3 parameter associative ring:
if(TYPE==1){  list extraVars=z,y_1,y_2;}
if(TYPE==2){  list extraVars=z,w;}
newRing[2]=newRing[2]+extraVars;
//we will call this ring 'internalRing' so that the ring name is
//universal to each of the solution types.
def internalRing=ring(newRing);
setring internalRing;
//map our user-inputted list to this new ring
list listOfPossibleSolution=imap(SolutionRing,listOfPossibleSol);
export(internalRing);
//the polynomial 'TOTAL' is the value of the AYBE.
poly TOTAL=totalAYBE(TYPE,listOfPossibleSolution[1],
listOfPossibleSolution[2],listOfPossibleSolution[3],listOfPossibleSolution[4]);
//if the expression checked is a solution to the AYBE
if(TOTAL==0)
{
        setring SolutionRing;
return("Is a solution to the AYBE");
}
//if the expression checked is not a solution to the AYBE
else
{
  //save the size of the polynomial 'TOTAL'.
```

```
def p=size(TOTAL);
//save the number of variables in the original ring.
def b=dimension*dimension;
//create several objects which are lists of integers
intvec T,tensor1,tensor2,tensor3; count=1;
//we need a loop that continues until the polynomial TOTAL is equal to
//zero.
while(TOTAL<>0)
  {
    //the SINGULAR command 'leadexp' returns the exponent vector of
    //the leading monomial of a polynomial.  The result of the
    //following command will be a vector of size 4b (b is from above).
    //This vector will have 1's where the first, second and third
    //tensors are found.
    T=leadexp(TOTAL);
    //to find the first tensor, we look at the vector given by the
    //elements of the variable string with a (1) added.
    tensor1=T[(b+1)..2*b];
    //to find the second tensor, we look at the vector given by the
    //elements of the variable string with a (2) added.
    tensor2=T[(2*b+1)..3*b];
    //to find the third tensor, we look at the vector given by the
    //elements of the variable string with a (3) added.
    tensor3=T[(3*b+1)..4*b];
    //we now create a number of lists.  Each list will have numerator,
    //denominator, tensor1, tensor2, and tensor3 from each term of the
    //polynomial TOTAL.  The tensors will be from the original list
    //(i.e. without brackets). The SINGULAR command 'monomial'
    //effectively reverses the command 'leadexp'. What we have done
    //here is factorised the polynomial TOTAL into lists.
    execute("list Factorised(count)=numerator(leadcoef(TOTAL)),
                   denominator(leadcoef(TOTAL)),"
  +string(monomial(tensor1))+","
  +string(monomial(tensor2))+","
  +string(monomial(tensor3))+";");
        count++;
    //we need to subtract the leading term from TOTAL.
  TOTAL=TOTAL-lead(TOTAL);
  }
//so that we can return values, we need to convert back to the
//associative ring we began with
setring SolutionRing;
//we define a string to hold the value of non-solutions
string lhsOfEquation="";
//we need a 'for' loop that is the size of the polynomial TOTAL.  This
//is the same as the number of lists defined above called
//'Factorised(i)'
for(i=1; i<=number(p); i=i+1)
   {
     //we need to undefine the lists 'solInputRing(i)' if they have
     //been previously defined to avoid getting a message from SINGULAR
     //"redefining ..."
```

```
       if(defined(solInInputRing(i))==1){kill solInInputRing(i);}
       //create a number of lists which map the lists defined above to
       //the solution ring
       list solInInputRing(i)=imap(internalRing,Factorised(i));
       export(solInInputRing(i));
       //the value of the non-solution is made into a string.  It is in
       //the format: numerator, denominator, tensor1, tensor2, tensor3,
       //[linebreak] etc.
       lhsOfEquation=lhsOfEquation+string(solInInputRing(i))+newline;
      }
}
  //this string is then exported to be used in the procedure 'showValue'
   export(lhsOfEquation);
   return("Is not a solution to the AYBE");
}

proc IsSolutionAYBE(list#)
"USAGE:     IsSolutionAYBE(#); # is a list.
 ASSUME:    # is the list of the possible solution to be checked in the %
    format:
            numerator,denominator, first tensor,second tensor. The elements %
    of
            this list are from the associative ring created.
 RETURN:    a string of text which tells the user whether the inputted %
    variables
            are a solution to the AYBE or not.  If the expression is not a
            solution to the AYBE, then the value is exported.
 EXAMPLE:   example IsSolutionAYBE; shows an example."
 {
   //the input must be grouped into lists of size 4.
   if(size(#)%4!=0){ERROR("The input must be a list entered as: numerator,
         denominator,variable1,variable2. See help ybe; for more %
    information");}
   else
     {
       int TYPE;
       if(find(charstr(basering),"x_1")){TYPE=1;}
       if(find(charstr(basering),"y")){TYPE=2;}
        //create a list of the list entered.  We need to have a name for this
       //list so it can be used in the procedure 'internalSolutionAYBE'.
        list SOL=#;
//we return the output of this internal procedure with TYPE=2
return(internalSolutionAYBE(TYPE,SOL));
     }
 }
example
{
  "EXAMPLE:";
  echo=2;
  def assocRing2=createRingA2par(2);
  setring assocRing2;
  IsSolutionAYBE(1,2z,e_1_1,e_1_1,\
    1,2z,e_1_1,e_2_2,\
    1,2z,e_2_2,e_1_1,\
    1,2z,e_2_2,e_2_2,\
    1,w,e_1_1,e_1_1,\
    1,w,e_2_2,e_2_2,\
    1,w,e_1_2,e_2_1,\
    1,w,e_2_1,e_1_2);
  def assocRing3=createRingA3par(2);
  setring assocRing3;
  IsSolutionAYBE(1,2z,e_1_1,e_1_1,\
    1,2z,e_1_1,e_2_2,\
    1,2z,e_2_2,e_1_1,\
    1,2z,e_2_2,e_2_2,\
    1,y_2-y_1,e_1_1,e_1_1,\
    1,y_2-y_1,e_2_2,e_2_2,\
    1,y_2-y_1,e_1_2,e_2_1,\
    1,y_2-y_1,e_2_1,e_1_2,\
    z-y_1,2,e_2_1,e_1_1,\
    -z+y_1,2,e_2_1,e_2_2,\
    z+y_2,2,e_1_1,e_2_1,\
    -z-y_2,2,e_2_2,e_2_1,\
    -z*(z-y_1)*(z+y2),2,e_2_1,e_2_1);
}

///////////////////////////////////////////////////////////////////////////////
//PROCEDURES FOR THE YBE
////////////////////////////////////////////////////////////////////////////////

proc showValue
"USAGE:     showValue();
 ASSUME:    The user has just performed one of the above procedures to check
            whether an expression is a solution to the CYBE.  This procedure
            uses the exported value 'lhsOfEquation'.
 RETURN:    a string which is defined in the procedure %
    'internalSolutionCYBE'.
            This string is the value of a non-solution.
 NOTE:      this procedure gives values of rational solutions only.
 EXAMPLE:   example showValue; shows an example."
 {
      //this procedure returns the string constructed in the procedure
      //'internalSolutionCYBE'
      return(lhsOfEquation);
 }
example
{
  "EXAMPLE:";
  echo=2;
  rationalSolution(1,z_2,f,e,\
   z3,2,h,e);
  showvalue();
}
```

```
//we now introduce some procedures which are required in order to convert the
//string attained in the procedure 'showValue' to LaTex format.

//the following procedure breaks the coefficients down into their %
    components so
//that we can write them in LaTeX format.

static proc parseCoefficient(int TYPE,string s)
"USAGE:     parseCoefficient(TYPE,s); TYPE is a number,s is a string.
 ASSUME:    TYPE is 1 if the basering is the rational ring, TYPE is 2 if the
            basering is any other ring.  The string s is an integer which is
            ended with a '!'
 RETURN:    a string which is in LaTeX format."
{
  int i=1; int b; string TexCoeff;
  //we create a loop that begins at the first element of the string s and  %
    ends
  //when it reaches an "!"
  while(s[i]<>"!")
    {
      //set b to be equal to the position of the element of the string we are
      //working on.
       b=i;
       //this loop takes into consideration the integers on the left hand %
    side
       //of the string adding them all to a string.
      while(s[i]>="0" and s[i]<="9")
      {
        i++;
  TexCoeff=TexCoeff+s[b,i-b];
  b++;
      }
      //if the string contains integers only, we need to exit from this while
      //loop.
      if(s[i]=="!"){break;}
      //otherwise, the string must contain additional parameters.  The %
    first of
      //these needs to be added to our string.
      TexCoeff=TexCoeff+s[i];
      i++;
      if(TYPE==1)
{
  //set b equal to the position of the next element
  b=i;
  //we now look at whether the parameter just added has a power.  If
  //SINGULAR finds integers after this parameter, then it adds to the
  //position.
  while(s[i]!="!" and s[i]>="0" and s[i]<="9"){i++;}
  //now if there is a difference between the position in the string
  //before the above while loop /we know that this must be a power.
  if(i-b)
    {
      //we add brackets to ensure that this is correctly written in
      //LaTeX format.
      TexCoeff=TexCoeff+"^{"+s[b,i-b]+"}";
    }
}
      if(TYPE==2)
{
  if(s[i]=="1" or s[i]=="2" or s[i]=="3")
    {
       TexCoeff=TexCoeff+s[i]; i++;
    }
  if(s[i]=="^")
    {
       i++;
       b=i;
       while(s[i]!="!" and s[i]>="0" and s[i]<="9"){i++;}
       TexCoeff=TexCoeff+"^{"+s[b,i-b]+"}";
     }
  if(s[i]=="*")
    {
      TexCoeff=TexCoeff+"\\cdot ";
      i++;
    }
}
    }
  return(TexCoeff);
}

proc texYBE(string fname,string s)
"USAGE:     (s); s is a string.
 ASSUME:    the string s is from the procedure 'showValue'.
 RETURN:    if fname= then a string is returned, the string s in LaTeX-
            typesetting. Otherwise this string is sent to the file <fname>, %
   and
            nothing is returned.
 NOTE:      In fname the .tex, if not given, is added to the file name.
 EXAMPLE:   example texYBE; shows an example."
{
  //introduce necessary variables.
  int i,j,k,l,m,TYPE; string %
    num,den,fTensor,sTensor,tTensor,sign,monom1,monom2,
  Texnum,Texden,TexSol; number changeSign;
  //we need to know what type of ring we are working in.  TYPE 1 is rational
  //ring, TYPE 2 is any other ring.  The procedure parseCoefficient works
  //differently for each type.
  if(charstr(basering)=="0,u,v"){TYPE=1;}
     else{TYPE=2;}
  //we create a loop that continues until the size of the string s is 1.  %
    This
  //one element is a 'newline' and so not required for the output.
  while(size(s)>1)
    {
      //we use the SINGULAR command 'find' to look for the commas in the %
    string
      i=find(s,",");
      //we create an 'if' statement to disregard any brackets around
      //coefficients
      if(s[1]=="("){num=s[2..(i-2)];}
      //the numerator of the first element of the value is found by taking %
    all
      //elements of the string up to this comma.
      else{num=s[1..(i-1)];}
      //we then look for the next comma in the string
      j=i+(find(s[i+1,size(s)],","));
      //again, we disregard brackets, and take the denominator to be the
      //elements from the first
      //to the second comma.
      if(s[i+1]=="("){den=s[(i+2)..(j-2)];}
   else{den=s[(i+1)..(j-1)];}
      //the three tensors from the tensor product are found using the third %
    and
      //fourth commas and the end of the line.
      k=j+(find(s[j+1,size(s)],","));
      l=k+(find(s[k+1,size(s)],","));
      m=l+(find(s[l+1,size(s)],newline));
      fTensor=s[(j+1)..(k-1)];
      sTensor=s[(k+1)..(l-1)];
      tTensor=s[(l+1)..(m-1)];
      //if we are converting a value from one of the associative rings, we %
    need
      //to overcome the problem of the double superscript in the variable %
    names
      if(find(fTensor,"_"))
{
  fTensor=fTensor[1]+"_{"+fTensor[3]+fTensor[5]+"}";
  sTensor=sTensor[1]+"_{"+sTensor[3]+sTensor[5]+"}";
  tTensor=tTensor[1]+"_{"+tTensor[3]+tTensor[5]+"}";
}
   //we now work on these five elements.
   //we check if the coefficient is positive or negative.
   if(num[1]=="-")
     {
       //if the numerator is negative, we convert it to a polynomial
       execute("changeSign="+num+";");
       //and then multiply this polynomial by -1
       changeSign=changeSign*(-1);
       //we then redefine the numerator
       num=string(changeSign);
       //and save the sign as a 'minus'
       sign="-";
     }
   else{sign="+";}
   //we then take each element of the numerator and convert it to LaTeX
   //format using the procedure 'parseCoefficient'
   while(size(num)!=0)
     {
       //to extract each element of the numerator separately, we use the
       //command 'find' to look for + or - signs, if there are no such
       //signs, then we save the numerator and delete 'num'.
       if(find(num,"+")==0 and find(num,"-")==0){monom1=num;num="";}
       //if there is a + sigm, then we save the first part of the
       //numerator and then delete this first part from 'num'.
       if(find(num,"+")){monom1=num[1..(find(num,"+"))];
       num=num[(find(num,"+")+1)..size(num)];}
       //Similarly, if a - sign is found.
       if(find(num,"-")){monom1=num[1..(find(num,"-"))];
       num=num[(find(num,"-")+1)..size(num)];}
       //we add an exclamation mark to this saved part of the numerator
       monom1=monom1+"!";
       //then we convert it to LaTeX-typesetting and add it to a string.
       Texnum=Texnum+string(parseCoefficient(TYPE,monom1));
     }
   //the same procedure is carried out for each denominator.
   while(size(den)!=0)
     {
               if(find(den,"+")==0 and find(den,"-")==0){monom2=den;den="";}
       if(find(den,"+")){monom2=den[1..(find(den,"+"))];
       den=den[(find(den,"+")+1)..size(den)];}
       if(find(den,"-")){monom2=den[1..(find(den,"-"))];
       den=den[(find(den,"-")+1)..size(den)];}
       monom2=monom2+"!";
       Texden=Texden+string(parseCoefficient(TYPE,monom2));
     }
   //we then begin to build our latex polynomial.
   TexSol=TexSol+sign+"\\frac{"+Texnum+"}{"+Texden+"}"+fTensor+"\\otimes
                                                "+sTensor+"\\otimes %
    "+tTensor;
   Texnum=""; Texden="";
   //at the end of each loop we redefine the string s without the 7
   //expressions already used.
   s=s[(find(s,newline)+1)..size(s)];
    }
  //if the user has not previously instructed SINGULAR not to print $ signs,
  //then we add them to each end of the polynomial.
  if(not(defined(NoDollars))){TexSol="$"+TexSol+"$";}
  //if the user has specified a file name for the text to be printed to,
  if(size(fname))
    {
      //we first check if they have added ".tex" to the file name, and if %
    not,
      //it is added.
      if(size(fname)>4)
{
  if(fname[size(fname)-3,4]!=".tex"){fname=fname+".tex";}
}
      else{fname=fname+".tex";}
      //the LaTeX text is written to the user-named file.
      write(fname,TexSol);
      print("Latex file generated");
    }
  else
    {
      return(TexSol);
    }
}
example
{
  "EXAMPLE:";
  echo=2;
  rationalSolution(1,z,f,e,\
   z_2,3,f,h);
  string s=showValue();
  texCYBE("",s);
}

////////////////////////////////////////////////////////////////////////////////
//CASIMIR ELEMENT PROCEDURES
////////////////////////////////////////////////////////////////////////////////

static
proc dualBasis(int n)
"USAGE:     dualBasis(n); n is an integer.
 ASSUME:    n^2-1 is the dimension of the special linear Lie algebra we are %
    working
            in.
 RETURN:    gives a part of the dual basis of the h(i) in the format of a %
    list.
 NOTE:      To see the complete dual basis of h(i) we must multiply this by %
    1/n
            is called from casimirEl and casimirZTest."
{
if(n<2)
{
print("incorrect input");
return(0);
}

list RESULT; int i,j;
//for sl(2) the dual basis of h is h and no calculations
//are required.
if(n==2)
{
RESULT=h;
}
//otherwise, we create a number of loops to calculate the dual basis
//based on the following formula:
//h(i)*=1/n(sum{j=1..i}(n-i)e_j_j-sum{j=i+1..n}ie_j_j) which
   //we have input as:
```

```
//h(i)*=1/n(sum{j=1..i}j(n-i)h_j+sum{j=i+1..n-1}i(n-j)h_j)
else
{
        //create our lists.
for(i=1; i<=(n-1); i++)
{
execute("list @h("+string(i)+");");
}
//calculate the dual basis of h(i).
for(i=1; i<=(n-1); i++)
{
for(j=1; j<=i; j++)
{
@h(i)[j]=j*(n-i)*h(j);

}
for(j=i+1; j<=(n-1); j++)
{
@h(i)[j]=i*(n-j)*h(j);
}
RESULT[i]=@h(i);
}
}
//we return a list of (n-1) lists, which together make up the dual basis
//of the h(i). Each of these lists has (n-1) number of elements
return(RESULT);
}

proc casimirZTest(int n)
"USAGE:     casimirZTest(n); n is an integer.
 ASSUME:    n^2-1 is the dimension of the special linear Lie algebra we are
            calculating from.
 RETURN:    a list which is the solution to the CYBE 't/z' where 't' is the
            Casimir element of sl(n).  The list is in the correct format so %
    it
            can be put directly into procedures based on the CYBE.
 NOTE:      this procedure works only in the basering sl(n).
 EXAMPLE:   example casimirZTest; shows an example."
{
      list partA, partB, casimirZTest; int i,j;
        //create a list of lists from the above procedure.
      list dualBasisH=dualBasis(n);
//for sl(2) the Casimir element is defined explicitly.
      if(n==2)
  {
    casimirZTest=1,2z,h,h,1,z,e,f,1,z,f,e;
  }
      else
      {
//we will add the numerator and denominator of each tensor to our list
//for use with the CYBE procedure.  We also incorporate the '1/n' which
//is in the formula for the dual basis of the h(i).  We add 4 at each
//loop so that these elements appear in the numerator and denominator
//position for each tensor.
for(i=1; i<=4*(n-1)^2; i=i+4)
{
partA[i]=1;
partA[i+1]=n*z;
}
//we introduce a counter
int count=3;
//this double loop adds each h(i) to its dual found in the procedure
//dualBasis. These elements must go in the position for first and second
//tensor in our list.
for(i=1; i<=(n-1); i++)
{
for(j=1; j<=(n-1); j++)
{
partA[count]=h(i);  count++;
partA[count]=dualBasisH[i][j];  count=count+3;
//we must add 3 to our counter so that these two elements
//appear in the correct position for tensor1 and tensor2
}
}
//the second part of our list is the tensor product of all other basis
//elements with their dual.  Again we need to add the 1/z as numerator
//and denominator for each element.
for(i=1; i<=4*n*(n-1); i=i+4)
{
partB[i]=1;
partB[i+1]=z;
}
//we redefine our counter.
count=3;
//we must add all the variables x(i) and the dual of these variables
//(the y(i))
for(i=1; i<=n*(n-1)/2; i++)
{
partB[count]=x(i); count++;
partB[count]=y(i); count=count+3;
//we must add 3 to our counter so that these two elements appear
//in the correct position for tensor1 and tensor2.
}
for(i=1; i<=n*(n-1)/2; i++)
{
partB[count]=y(i); count++;
partB[count]=x(i); count=count+3;
}
//the addition of the two parts to this list gives us the 1/z times the
//Casimir element.
casimirZTest=partA+partB;
     }
return(casimirZTest);
}
```

```
example
{
  "EXAMPLE:"; echo=2;
  casimirZTest(2);
}

proc casimirZTest2par(int n)
"USAGE:     casimirZTest2par(n); n is an integer.
 ASSUME:    n^2-1 is the dimension of the special linear Lie algebra we are
            calculating from.
 RETURN:    a list which is the solution to the 2 parametric CYBE 't/z_1-z_2'
            where 't' is the Casimir element of sl(n).  The list is in the
            correct format so it can be put directly into procedures based on
            the CYBE.
 NOTE:      this procedure works only in the basering sl(n).
 EXAMPLE:   example casimirZTest2par; shows an example."
{
      list partA, partB, casimirZTest2par; int i,j;
        //create a list of lists from the above procedure.
      list dualBasisH=dualBasis(n);
      if(n==2)
  {
    casimirZTest2par=1,2*z_1-2*z_2,h,h,1,z_1-z_2,e,f,1,z_1-z_2,f,e;
  }
      else
      {
//we will add the numerator and denominator of each tensor to our list
//for use with the CYBE procedure.  We also incorporate the '1/n' which
//is in the formula for the dual basis of the h(i).  We add 4 at each
//loop so that these elements appear in the numerator and denominator
//position for each tensor.
for(i=1; i<=4*(n-1)^2; i=i+4)
{
partA[i]=1;
partA[i+1]=n*z_1-n*z_2;
}
//we introduce a counter
int count=3;
//this double loop adds each h(i) to its dual found in the procedure
//dualBasis. These elements must go in the position for first and second
//tensor in our list.
for(i=1; i<=(n-1); i++)
{
for(j=1; j<=(n-1); j++)
{
partA[count]=h(i);  count++;
partA[count]=dualBasisH[i][j];  count=count+3;
//we must add 3 to our counter so that these two elements
//appear in the correct position for tensor1 and tensor2.
}
}
//the second part of our list is the tensor product of all other basis
//elements with their dual.  Again we need to add the 1/z as numerator
//and denominator for each element.
for(i=1; i<=4*n*(n-1); i=i+4)
{
partB[i]=1;
partB[i+1]=z_1-z_2;
}
//we redefine our counter.
count=3;
//we must add all the variables x(i) and the dual of these variables
//(the y(i))
for(i=1; i<=n*(n-1)/2; i++)
{
partB[count]=x(i); count++;
partB[count]=y(i); count=count+3;
//we must add 3 to our counter so that these two elements appear
//in the correct position for tensor1 and tensor2.
}
for(i=1; i<=n*(n-1)/2; i++)
{
partB[count]=y(i); count++;
partB[count]=x(i); count=count+3;
}
//the addition of the two parts to this list gives us the 1/z times the
//Casimir element.
casimirZTest2par=partA+partB;
     }
return(casimirZTest2par);
}
example
{
  "EXAMPLE:"; echo=2;
  casimirZTest2par(2);
}

proc casimirEl(int n)
"USAGE:     casimirEl(n); n is an integer.
 ASSUME:    n^2-1 is the dimension of the special linear Lie algebra we are
            calculating from.
 RETURN:    a string which is the Casimir element of the special linear %
    algebra
            sl(n).
 EXAMPLE:   example casimirEl; shows an example."
{
        list partA, partB, casimir; int i,j;
        //create a list of lists from the above procedure.
        list dualBasisH=dualBasis(n);
if(n==2)
  {
    casimir=1,2,h,h,1,1,e,f,1,1,f,e;
  }
else
```

```
  {
    //for the first part of our list the numerator is 1 and the
    //denominator is n
    for(i=1; i<=4*(n-1)^2; i=i+4)
      {
partA[i]=1;
partA[i+1]=n;
      }
    int count=3;
    //this double loop adds each h(i) to its dual found in the procedure
    //dualBasis. These elements must go in the position for first and
    //second tensor in our list.
    for(i=1; i<=(n-1); i++)
      {
for(j=1; j<=(n-1); j++)
{
partA[count]=h(i);  count++;
partA[count]=dualBasisH[i][j];  count=count+3;
}
      }
    //for the second part of our list the numerator and denominator are
    //both 1
    for(i=1; i<=4*n*(n-1); i=i+4)
      {
partB[i]=1;
partB[i+1]=1;
      }
    //we redefine our counter.
    count=3;
    //we must add all the variables x(i) and the dual of these variables
    //(the y(i))
    for(i=1; i<=n*(n-1)/2; i++)
      {
partB[count]=x(i); count++;
partB[count]=y(i); count=count+3;
      }
    for(i=1; i<=n*(n-1)/2; i++)
      {
partB[count]=y(i); count++;
partB[count]=x(i); count=count+3;
      }
    //the addition of the two parts to this list gives us the Casimir
    //operator.
    casimir=partA+partB;
  }
    return(casimir);
}
example
{
  "EXAMPLE:"; echo=2;
  def ratRing=createRingRational("sl(2)");
  setring ratRing;
  casimirEl(2);
}

////////////////////////////////////////////////////////////////////////////
//PROCEDURES FOR CHECKING SOLUTIONS WITH 2 PARAMETERS IN sl(n)
///////////////////////////////////////////////////////////////////////////

proc ratSolType1(int n)
"USAGE:     ratSolType1(n); n is an integer.
 ASSUME:    n is the dimension of the expression to be created
 RETURN:    a list based on the equation of Thilo Henrich which is a
            solution to the rational CYBE with 2 parameters.
 EXAMPLE:   example ratSolType1; shows an example."
{
int i,j,k,l;
 int count=1;
 if(n==2){execute("poly e_1_2="+string(e)+";");
   execute("poly e_2_1="+string(f)+";");
   execute("poly h_1="+string(h)+";");
 }
 else{
//create necessary polynomials
   for(i=1; i<=(n-2); i++)
{
       for(k=0; k<=(n-i-1); k++)
       {
       execute("poly %
    e_"+string(k+1)+"_"+string(k+1+i)+"=x("+string(count)+");");
       execute("poly %
    e_"+string(k+1+i)+"_"+string(k+1)+"=y("+string(count)+");");
       count++;
       }
}
execute("poly e_1_"+string(n)+"=x("+string((n^2-n)/2)+");");
execute("poly e_"+string(n)+"_1=y("+string((n^2-n)/2)+");");
 }
 list dualBasisH=dualBasis(n);
list ratSol;
//first bracket
count=1;
for(i=1; i<=(n-1); i++)
{
ratSol[count]=-z_1;count++;
ratSol[count]=n;count++;
ratSol[count]=e_1_2;count++;
ratSol[ count]=dualBasisH[1][i]; count++;
}

for(j=3; j<=n; j++)
{
for(k=1; k<=(n-j+1); k++)
{
```

```
ratSol[count]=-z_1;count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_1_"+string(j)+";"); count++;
execute("ratSol[count]=e_"+string(j+k-1)+"_"+string(k+1)+";"); count++;
}
}
//second bracket
for(i=1; i<=(n-1); i++)
{
ratSol[count]=z_2;count++;
ratSol[count]=n;count++;
ratSol[count]=dualBasisH[1][i]; count++;
ratSol[count]=e_1_2;count++;
}
for(j=3; j<=n; j++)
{
for(k=1; k<=(n-j+1); k++)
{
ratSol[count]=z_2;count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(j+k-1)+"_"+string(k+1)+";"); count++;
execute("ratSol[count]=e_1_"+string(j)+";"); count++;
}
}
//third bracket
for(j=2; j<=(n-1); j++)
{
for(k=1; k<=(n-j); k++)
{
ratSol[count]=1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_1_"+string(j)+";"); count++;
execute("ratSol[count]=e_"+string(j+k)+"_"+string(k+1)+";"); count++;
}
}
for(i=2; i<=(n-1); i++)
{
for(j=1; j<=(n-1); j++)
{
ratSol[count]=1;count++;
ratSol[count]=n;count++;
execute("ratSol[count]=e_"+string(i)+"_"+string(i+1)+";"); count++;
ratSol[count]=dualBasisH[i][j];count++;
}
}
for(i=2; i<=(n-2); i++)
{
for(k=2; k<=(n-i); k++)
{
for(l=1; l<=(n-i-k+1); l++)
{
ratSol[count]=1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(i)+"_"+string(i+k)+";"); count++;
execute("ratSol[count]=e_"+string(i+k+l-1)+"_"+string(l+i)+";"); count++;
}
}
}
//fourth bracket
for(j=2; j<=(n-1); j++)
{
for(k=1; k<=(n-j); k++)
{
ratSol[count]=-1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(j+k)+"_"+string(k+1)+";"); count++;
execute("ratSol[count]=e_1_"+string(j)+";"); count++;
}
}
for(i=2; i<=(n-1); i++)
{
for(j=1; j<=(n-1); j++)
{
ratSol[count]=-1;count++;
ratSol[count]=n;count++;
ratSol[count]=dualBasisH[i][j];count++;
execute("ratSol[count]=e_"+string(i)+"_"+string(i+1)+";"); count++;
}
}
for(i=2; i<=(n-2); i++)
{
for(k=2; k<=(n-i); k++)
{
for(l=1; l<=(n-i-k+1); l++)
{
ratSol[count]=-1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(i+k+l-1)+"_"+string(l+i)+";");count++;
execute("ratSol[count]=e_"+string(i)+"_"+string(i+k)+";"); count++;
}
}
}
return(ratSol);
}
example
{
  "EXAMPLE:"; echo=2;
  def ratRing2par4=createRingRational2par("sl(4)");
  setring ratRing2par4;
  ratSolType1(4);
}

proc ratSolType2(int n)
"USAGE:     ratSolType2(n); n is an integer.
```

```
 ASSUME:    n is the dimension of the expression to be created
 RETURN:    a list based on the equation of Thilo Henrich which is a
            solution to the rational CYBE with 2 parameters.
 EXAMPLE:   example ratSolType2; shows an example."
{
int i,j,k,l;
int count=1;
if(n==2){execute("poly e_1_2="+string(e)+";");
   execute("poly e_2_1="+string(f)+";");
   execute("poly h_1="+string(h)+";");
 }
 else{
//create necessary polynomials
   for(i=1; i<=(n-2); i++)
{
       for(k=0; k<=(n-i-1); k++)
       {
       execute("poly %
    e_"+string(k+1)+"_"+string(k+1+i)+"=x("+string(count)+");");
       execute("poly %
    e_"+string(k+1+i)+"_"+string(k+1)+"=y("+string(count)+");");
       count++;
       }
}
execute("poly e_1_"+string(n)+"=x("+string((n^2-n)/2)+");");
execute("poly e_"+string(n)+"_1=y("+string((n^2-n)/2)+");");
 }
 list dualBasisH=dualBasis(n);
list ratSol;
//first bracket
count=1;
for(i=1; i<=(n-1); i++)
{
ratSol[count]=-z_2;count++;
ratSol[count]=n;count++;
ratSol[count]=dualBasisH[(n-1)][i]; count++;
execute("ratSol[count]=e_"+string(n)+"_"+string(n-1)+";"); count++;
}
for(i=1; i<=(n-2); i++)
  {
    for(j=1; j<=i; j++)
      {
ratSol[count]=z_2;count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(j)+"_"+string(j+n-i-1)+";"); count++;
execute("ratSol[count]=e_"+string(n)+"_"+string(i)+";"); count++;
      }
  }
//second bracket
for(i=1; i<=(n-1); i++)
{
ratSol[count]=z_1;count++;
```

```
ratSol[count]=n;count++;
execute("ratSol[count]=e_"+string(n)+"_"+string(n-1)+";"); count++;
ratSol[count]=dualBasisH[(n-1)][i]; count++;
}
for(i=1; i<=(n-2); i++)
{
for(j=1; j<=i; j++)
{
ratSol[count]=-z_1;count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(n)+"_"+string(i)+";"); count++;
 execute("ratSol[count]=e_"+string(j)+"_"+string(j+n-i-1)+";"); count++;
}
}
//third bracket
for(i=2; i<=(n-1); i++)
{
for(j=1; j<=(i-1); j++)
{
ratSol[count]=-1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(j)+"_"+string(j+n-i)+";"); count++;
execute("ratSol[count]=e_"+string(n)+"_"+string(i)+";"); count++;
}
}
for(i=1; i<=(n-2); i++)
{
for(j=1; j<=(n-1); j++)
{
ratSol[count]=1;count++;
ratSol[count]=n;count++;
ratSol[count]=dualBasisH[i][j];count++;
execute("ratSol[count]=e_"+string(i+1)+"_"+string(i)+";"); count++;
}
}
for(i=3; i<=(n-1); i++)
{
for(j=1; j<=(i-2); j++)
{
for(k=1; k<=j; k++)
{
ratSol[count]=-1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(k)+"_"+string(k+i-j-1)+";"); count++;
execute("ratSol[count]=e_"+string(i)+"_"+string(j)+";"); count++;
}
}
}
//fourth bracket
for(i=2; i<=(n-1); i++)
{
for(j=1; j<=(i-1); j++)
```

```
{
ratSol[count]=1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(n)+"_"+string(i)+";"); count++;
execute("ratSol[count]=e_"+string(j)+"_"+string(j+n-i)+";"); count++;
}
}
for(i=1; i<=(n-2); i++)
{
for(j=1; j<=(n-1); j++)
{
ratSol[count]=-1;count++;
ratSol[count]=n;count++;
execute("ratSol[count]=e_"+string(i+1)+"_"+string(i)+";"); count++;
ratSol[count]=dualBasisH[i][j];count++;
}
}
for(i=3; i<=(n-1); i++)
{
for(j=1; j<=(i-2); j++)
{
for(k=1; k<=j; k++)
{
ratSol[count]=1; count++;
ratSol[count]=1; count++;
execute("ratSol[count]=e_"+string(i)+"_"+string(j)+";");count++;
execute("ratSol[count]=e_"+string(k)+"_"+string(k+i-j-1)+";"); count++;
}
}
}
return(ratSol);
}
example
{
  "EXAMPLE:"; echo=2;
  def ratRing2par4=createRingRational2par("sl(4)");
  setring ratRing2par4;
  ratSolType2(4);
}
```