

ABSTRACT

WATSON, ROBERT L. Lifting Automorphisms from Root Systems to Lie Algebras. (Under the direction of Dr. Aloysius G. Helminck).

In 1996 and 2000 A.G. Helminck gave the first algorithms for computing some of the structure of symmetric spaces. In this thesis we extend these results by designing algorithms for other aspects of the structure of local symmetric spaces. We begin with an involution on the root system. We would like to understand how this involution describes an involution on the Lie algebra. To do so, we consider the concept of *lifting*. We say an involution θ on the root system Φ can be lifted to an involution $\bar{\theta}$ on the algebra if we can find $\bar{\theta}$ so that $\bar{\theta}|_{\Phi} = \theta$. Success gives rise to a method to compute local symmetric spaces.

Accomplishing this task requires effort on multiple fronts. On a small scale we consider a *correction vector*. A correction vector lives in the toral subalgebra of the Lie algebra. A result due to Steinberg establishes a unique Lie algebra automorphism that can always be defined. We can modify this map with the correction vector so that it becomes an involution.

On a large scale, computing the correction vector is too cumbersome. We will show how to “break apart” larger involutions on the root system by projecting the roots into the local symmetric space, then “extracting” specific sub-systems. We can correct the involution on each sub-system, then “glue” the pieces together to form the involution on the whole algebra. This process not only vastly improves the timing of the lifting process, but also gives rise to an argument that any involution on the root system can be lifted.

We then present an entire computer package (written for Mathematica) for working with local symmetric spaces. This package includes the algorithms we devise, as well as “helper” algorithms which are necessary for implementation.

Lifting Automorphisms from Root Systems to Lie Algebras

by
Robert L. Watson

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Mathematics

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Naihuan Jing

Dr. Amassa Fauntleroy

Dr. Aloysius G. Helminck
Chair of Advisory Committee

Dr. Ernie L. Stitzinger

DEDICATION

To Elizabeth, with whom I have shared this and all of life's journeys.

BIOGRAPHY

Robert L. Watson's current incarnation began on 9 April, 1983 in Fort Worth, Texas, USA. During his stint as a daffodil in 1653, he had pondered a proof for Fermat's Little Theorem (daffodils, as we all should know, have ample time to themselves for thinking). Unfortunately he was snipped before he could complete his work. During his next incarnation he was successful in proving the theorem. But alas! No reputable mathematics journals would accept manuscripts from basset hounds. Even to this day it is true. He is still bitter about it.

Robert went on to receive his elementary education from Bear Creek Elementary School in Houston, TX, where he majored in multiplication flashcard drills. A few years later he completed his stay at the Katy Independent School District in Houston, TX, and went on to study at the University of Tulsa. He completed his Bachelor of Science degree in Mathematics, and Master of Science degree December 2004 and May 2006 respectively. In the Autumn of 2006 he entered the Ph.D. program in Mathematics with a concentration in Symbolic Computation and Lie Theory at North Carolina State University at Raleigh.

Robert heroically volunteered all of his free time to exploring the depths of black holes and reporting back on his findings. He can safely volunteer without fear of actually being sent to a black hole, as everyone knows graduate students have no spare time. This keeps him hard at work. His second biggest fear is that he'll suddenly run out of math and have to fulfill his obligation to explore black holes. His first biggest fear is that one day he'll wake up and suddenly have to take the world seriously.

ACKNOWLEDGMENTS

First of all I would like to thank my wife, Elizabeth, who has supported my dreams since the day we first met (and never seemed to mind the floating mass of textbooks that accompanied me wherever we went). I would also like to thank my family for their never-ending encouragement and support:

Melissa, my mother, whose advice on teaching four year-olds is frighteningly useful (sometimes we don't grow up - we just grow bigger faces).

Tom, my father, who when I was little was always happy to keep me up an hour past bedtime to answer any questions I had about the world.

Joe, my brother, who adds a philosophical perspective to my life.

My grandmothers: Hazel, with whom I have shared many of my experiences,

and Joanne who was always happy to open up her home when I needed a retreat from my busy life in Tulsa.

and finally Trey, who has taught me more about the world than some may believe.

Of course, special thanks goes to my advisor, Dr. Aloysius "Loek" Helminck, whose advice and encouragement has helped me in many ways. I am fortunate to have such an excellent mentor.

I would also like to thank the research group for their help and encouragement: John Absher, Kate Brenneman, Catherine Buell, Kyle Thompson, and Qiang Wang. Also, best of luck to the new-comers in their endeavors: John Hutchens, Emma Norbrothen.

Further, I would like to thank Dr. Ernest "Stitz" Stitzinger, Dr. Naihuan Jing, and Dr. Amassa Fauntleroy for expressing interest in my work; Dr. Alina Duca and Dr. John Griggs for helping to shape my teaching and encouraging my endeavors in the classroom; and Denise Seabrooks, whose help since my very first day at NCSU has been much appreciated.

Finally I would like to thank Dr. William Coberly and Dr. Christian Constanda at the University of Tulsa, whom have helped pave the mathematical foundations of all my successes.

TABLE OF CONTENTS

LIST OF TABLES	xiii
LIST OF FIGURES	xv
Chapter 1 Introduction	1
1.1 A Brief Introduction to Symmetric Spaces	1
1.2 Recovering the Action of an Involutorial Automorphism on the Lie Algebra	3
1.3 A Brief Overview	3
Chapter 2 Preliminary Topics in Lie Algebra	5
2.1 The Root Space Decomposition	5
2.2 Root Systems	6
2.3 Root Systems in \mathbb{R}^n	9
2.4 Cartan Matrices and Dynkin Diagrams	11
2.5 Bases of the Root Systems	13
2.6 Weyl Groups and the Longest Element	14
2.7 Chevalley Constants	15
2.8 A Library of Identities on Chevalley Constants	16
2.9 Example: Root Space Decomposition of $\mathfrak{sl}_n(\mathbb{C})$	20
Chapter 3 Systems of Multivariate Polynomial Equations	23
3.1 Multivariate Division	23
3.2 Monomial Ordering	24
3.3 Groebner Bases	25
3.4 Application to Systems of Multivariate Polynomials	26
3.5 Existence of a Solution	28
Chapter 4 Involutorial Automorphisms With a Maximal (-1)-Eigenspace .	29
4.1 Classification of the Local Symmetric Spaces	29
4.2 The Structure Constants	31
4.3 The Lifting Condition	34
4.4 The Correction Vector	36
4.5 Computation of the Correction Vector(s)	37
4.6 Correction Vectors for the Involutorial Automorphisms with (-1)-Eigenspace	46
Chapter 5 Admissibility	49
5.1 Root Projections	49
5.2 θ -Normality	51
5.3 Admissibility Criteria	51
5.4 An Algorithm to Check for Admissibility	52

Chapter 6	Relations Between Structure Constants and the Weyl Group..	54
6.1	Structure Constants for Roots Fixed by θ	56
6.2	General Notes on the Action of $w_0(\theta)$ on Projecting Roots	57
6.3	Action of $w_0(\theta)$ on Projecting Roots Over Type A	57
6.4	Action of $w_0(\theta)$ on Projecting Roots Over Type B	61
6.5	Action of $w_0(\theta)$ on Projecting Roots Over Type C	64
6.6	Action of $w_0(\theta)$ on Projecting Roots Over Type D	69
6.7	Identities on the Structure Constants over a Root System of Type A	74
6.8	Identities on the Structure Constants over a Root System of Type B	77
6.9	Identities on the Structure Constants over a Root System of Type C	80
6.10	Identities on the Structure Constants over a Root System of Type D	81
Chapter 7	Classification of the 1-Consistent Involutorial Helminck Dia-	
	grams	86
7.1	1-Consistency	87
7.2	Restricted Rank One Automorphisms	88
7.3	The Restricted Rank One Decomposition of an Involution	88
7.4	Constructing Involutorial Automorphisms of Higher Rank From Restricted Rank One Automorphisms	91
7.5	Involutions With Restricted Rank One	95
7.6	Restricted Rank and 1-Consistency	96
7.7	A Classification Scheme for 1-Consistent Helminck Diagrams	98
7.8	Computation of the 1-Consistency Property	105
Chapter 8	Classification of the Involutions on The Root System Which Lift	108
8.1	Examining the Lifting Condition	109
8.2	Involutions of Restricted Rank One	109
8.3	An Algorithm for Lifting Involutions of Restricted Rank One	113
8.4	Reduction to Restricted Rank One	118
8.5	Constructing Automorphisms for Higher Restricted Ranks	119
8.6	The Gluing Mechanism: $\bar{\theta}$ Involution Construction	123
8.7	Involution Merge	124
8.8	The Polarity of the Involution $\bar{\theta}$	127
8.9	An Illustration of $\bar{\theta}$ Involution Construction	133
8.10	An Improved Lifting Algorithm	137
Chapter 9	Supporting Algorithms and Notes on Implementation	140
9.1	Retrieving the Action of an Involution on the Root System From the Helminck Diagram	140
9.2	Identifying the Type of the Restricted Root System	146
9.3	Identifying the Table Entry in Table C.1	148
9.4	Determining if a Helminck Diagram Describes an Involution on the Roots .	152
9.5	Representation of Involutions on the Lie Algebra	156

Chapter 10 Summary of Conclusions and Future Work.....	158
10.1 Conclusions	158
10.2 Some Questions to Address	159
Chapter 11 Programming Interface for Symbolic Computation in LiE Groups and Symmetric Spaces (Version 1.1 Manual)	160
11.1 Introduction	160
11.2 System Requirements	161
11.3 Organization	162
11.4 Descriptions of Packages	162
11.4.1 Root System and Lie Algebra Package (PI-ROOT)	162
11.4.2 Chevalley Structure Package (PI-CHEVY)	163
11.4.3 Weyl Package (PI-WEYL)	163
11.4.4 Group Action Package (PI-GAP)	163
11.4.5 Local Symmetric Spaces Package (PI-LOSS)	164
11.5 Definitions and Data Structures	164
11.6 List of Procedures	169
11.6.1 Root Systems and Lie Algebra (PI-ROOT) Primary	169
11.6.2 Root Systems and Lie Algebra (PI-ROOT) Diagram	177
11.6.3 Root Systems and Lie Algebra (PI-ROOT) Internal	178
11.6.4 Chevalley Structure Package (PI-CHEVY) Primary	183
11.6.5 Weyl Package (PI-WEYL) Primary	185
11.6.6 Weyl Package (PI-WEYL) Internal	185
11.6.7 Group Action Package (PI-GAP) Primary	185
11.6.8 Group Action Package (PI-GAP) Diagram	188
11.6.9 Group Action Package (PI-GAP) Internal	189
11.6.10 Local Symmetric Spaces Package (PI-LOSS) Primary	190
11.6.11 Local Symmetric Spaces Package (PI-LOSS) Diagram	198
11.6.12 Local Symmetric Spaces Package (PI-LOSS) Internal	198
11.7 Tutorial and Examples	199
11.8 Copyright	208
Chapter 12 Programming Interface for Symbolic Computation in LiE Groups and Symmetric Spaces (Version 1.1.0 Source)	209
12.1 Notes on Construction of The Mathematica Lie Algebra and Local Symmet- ric Spaces Package	209
12.2 Root System and Lie Algebra Package (Primary)	210
12.2.1 adMatrix	210
12.2.2 adMatrixDiagonal	212
12.2.3 AllRootBasisConnected	213
12.2.4 ApplyRootMap	214
12.2.5 BasisCoefficients	215
12.2.6 CartanMatrix	216
12.2.7 CartanToRootSystem	217

12.2.8	ChevalleyLookup	218
12.2.9	CoRoot	219
12.2.10	cvMinimalPolynomialList	220
12.2.11	e	221
12.2.12	eForm	222
12.2.13	FundamentalDominantWeights	223
12.2.14	gInvolutionListFormToMatrix	224
12.2.15	gMergeInvolutions	226
12.2.16	HighestRoot	228
12.2.17	IdentifyRootSystem	229
12.2.18	InnerProduct	230
12.2.19	KleinChevalley	231
12.2.20	LieBracket	234
12.2.21	LinearOperatorMatrix	235
12.2.22	LinearOperatorOrder	236
12.2.23	MakeBasis	237
12.2.24	MakeRootBasis	238
12.2.25	OperatorMatrixFromFunction	239
12.2.26	PositiveRootSystem	240
12.2.27	PrintMatrixArray	242
12.2.28	Reflect	243
12.2.29	RestrictedRootAut	245
12.2.30	RootAlphaForm	246
12.2.31	RootBase	247
12.2.32	RootBasisConnectedSet	248
12.2.33	RootBasisConnectedQ	250
12.2.34	RootBasisQ	251
12.2.35	RootCoBase	252
12.2.36	RootDecomposition	253
12.2.37	RootFunctional	255
12.2.38	RootHeight	256
12.2.39	RootLessQ	257
12.2.40	RootSplit	258
12.2.41	RootString	259
12.2.42	RootStringBounds	260
12.2.43	RootSumPath	261
12.2.44	RootSystem	262
12.2.45	RootToString	263
12.2.46	StructureConstantsFromBasis	264
12.2.47	StructureConstantsLookup	267
12.3	Root System and Lie Algebra Package (Diagram)	267
12.3.1	DrawRootSystem	267
12.3.2	DynkinDiagram	271

12.3.3	gInvolutionDiagram	272
12.4	Root System and Lie Algebra Package (Internal)	272
12.4.1	BasisCoefficientsMatrix	272
12.4.2	BasisCoefficientsVector	274
12.4.3	BasisToRootSystem	275
12.4.4	BlockAssemble	276
12.4.5	BlockList	277
12.4.6	ByBasisSort	278
12.4.7	CartanMatrixFromBasis	279
12.4.8	CartanNorm	280
12.4.9	CartanToRootSystemSimple	281
12.4.10	Diagonalize	284
12.4.11	DynkinData	285
12.4.12	DynkinDataNaturalOrdering	288
12.4.13	DynkinEdgeCodes	291
12.4.14	DynkinEdgeCons	293
12.4.15	DynkinEdgeConsNestedForm	294
12.4.16	DynkinHeight	295
12.4.17	DynkinPoints	296
12.4.18	DynkinOrientation	298
12.4.19	DynkinToCartan	300
12.4.20	DynkinWidth	301
12.4.21	FindPivots	302
12.4.22	GroebnerBackSolver	303
12.4.23	GroebnerOneSolution	305
12.4.24	IrreducibleRootInput	306
12.4.25	LieMultTable	307
12.4.26	LittleDynk	308
12.4.27	MatrixMinimalPolynomial	311
12.4.28	MatrixNorm	312
12.4.29	RootInput	313
12.4.30	RootInputQ	314
12.4.31	RootListFormQ	315
12.4.32	RootStringFormQ	316
12.4.33	RootSystemFromBasis	317
12.4.34	RowSwap	319
12.4.35	SimpleRootBase	320
12.4.36	StringToRoot	322
12.4.37	TakeElements	324
12.4.38	TakeRows	325
12.4.39	VectorPad	326
12.4.40	ZeroesAbove	327
12.4.41	ZeroesBelow	328

12.4.42	ZeroesLeft	329
12.4.43	ZeroesRight	330
12.5	Chevalley Structure Package (Primary)	330
12.5.1	ExtraSpecialPairs	330
12.5.2	ExtraSpecialPairQ	332
12.5.3	SpecialPairs	334
12.5.4	SpecialPairQ	335
12.6	Weyl Package (Primary)	336
12.6.1	LongestElement	336
12.6.2	ReflectWeyl	337
12.6.3	WeylCompare	338
12.6.4	WeylLength	339
12.6.5	WeylReduce	340
12.7	Weyl Package (Internal)	340
12.7.1	InteriorPoint	341
12.7.2	FundamentalChamber	342
12.8	Group Action Package (Primary)	342
12.8.1	ArchesListInvolution	342
12.8.2	DiskList	346
12.8.3	EigenspaceProject	347
12.8.4	EmbeddedRootGroups	348
12.8.5	EmbeddedRootIndices	349
12.8.6	EmbeddedRootSystems	350
12.8.7	FixedBasis	351
12.8.8	FixedRootQ	353
12.8.9	FixedRoots	354
12.8.10	IsRootAutOrder	355
12.8.11	wInvolution	356
12.8.12	wInvolutionAction	359
12.9	Group Action Package (Diagram)	360
12.9.1	DynkinPointsTeX	360
12.9.2	HelminckDiagram	362
12.9.3	HelminckDiagramTeX	364
12.10	Group Action Package (Internal)	366
12.10.1	HelminckDiagramSTeX	366
12.10.2	ThetaComponents	370
12.11	Local Symmetric Spaces Package (Primary)	372
12.11.1	AlignPolarities	372
12.11.2	ApplyRootInvolution	375
12.11.3	ApplyRootInvolutionBasis	376
12.11.4	ComplementRoot	377
12.11.5	CorrectionVector	378
12.11.6	DiagramInvolution	379

12.11.7	InvBasisTable	380
12.11.8	InvBasisTableAlphaForm	381
12.11.9	InvBTRaw	382
12.11.10	InvolutionPolarity	383
12.11.11	LocalBasis	384
12.11.12	LocalProject	385
12.11.13	OneCorrectionVector	386
12.11.14	OrthoComplement	388
12.11.15	RankOneComponentBasis	389
12.11.16	RankOneLocalBasis	391
12.11.17	RankOneRootLift	392
12.11.18	ReduceRestrictedRank	396
12.11.19	RestrictedRankOneBasis	398
12.11.20	RestrictedRankOneDecomp	399
12.11.21	RestrictedRankOneSystem	401
12.11.22	RestrictedRootBasis	403
12.11.23	RestrictedRootRank	405
12.11.24	RestrictedRootSystem	406
12.11.25	RestrictedRootSystemType	407
12.11.26	RootCriticalValues	408
12.11.27	RootInvolution	409
12.11.28	RRDLift	413
12.11.29	RROITable	415
12.11.30	RROITableEntry	416
12.11.31	SimpleRootLift	419
12.11.32	SteinbergThetaDelta	424
12.11.33	SwitchPolarity	425
12.11.34	ThetaStable	427
12.12	Local Symmetric Spaces Package (Diagram)	427
12.12.1	RankOneDecompDiagram	427
12.12.2	ReduceRestrictedRankDiagram	429
12.12.3	RestrictedRankOneDiagram	430
12.12.4	RestrictedRootDiagram	432
12.13	Local Symmetric Spaces Package (Internal)	433
12.13.1	EigenspaceProject (Additional Definitions)	434
12.13.2	FixedRootQ (Additional Definitions)	435
12.13.3	FixedRoots (Additional Definitions)	436
12.13.4	FixedBasis (Additional Definitions)	437
12.13.5	gInvolutionDiagram (Additional Definitions)	438
12.13.6	gInvolutionListFormToMatrix (Additional Definitions)	439
12.13.7	DiagramInvolutionSimple	440

Bibliography	441
-------------------------------	------------

Appendices.....	442
Appendix A: Dynkin Diagrams and Cartan Matrices	443
Appendix B: Helminck Diagrams For Involutorial Automorphisms with a Maximal (-1)-Eigenspace	449
Appendix C: Helminck Diagrams for Involutions of Restricted Rank One . .	453

LIST OF TABLES

Table 2.1	Possible Angles Between Pairs of Roots	11
Table 2.2	Usual Root System Bases	13
Table 6.1	CPU Timings For Lifting θ	55
Table 7.1	1-Consistency States for G2, θ Involution	99
Table 7.2	1-Consistency States for F4, θ Involution	107
Table B.1	Helminck Diagrams For Involutorial Automorphisms with a Maximal (-1)- Eigenspace.....	449
Table B.2	Correction Vectors in the DIIIb Case	452
Table C.1	Helminck Diagrams for Involutions of Restricted Rank One	453
Table C.2	Correction Vector for Type 3	455
Table C.3	Correction Vector for Type 7, $n = 4$	455
Table C.4	Correction Vector for Type 8, $n = 3$	455
Table C.5	Correction Vector for Type 11, $n = 6$	455
Table C.6	Correction Vector for Type 12	456

Table C.7	Correction Vector for Type 13	456
Table C.8	Correction Vector for Type 14	456
Table C.9	Correction Vector for Type 15	457
Table C.10	Correction Vector for Type 17	457
Table C.11	Correction Vector for Type 18	457

LIST OF FIGURES

Figure 4.1	Helminck Diagram for DIIIb	30
Figure 6.1	CPU Timings For Lifting θ	55
Figure 6.2	Helminck Diagram for Restricted Rank One Involution of Type 5	74
Figure 6.3	Helminck Diagram for Restricted Rank One Involution of Type 6	78
Figure 6.4	Helminck Diagram for Restricted Rank One Involution of Type 9	80
Figure 6.5	Helminck Diagram for Restricted Rank One Involution of Type 10	82
Figure A.1	Cartan Matrix for Type A	443
Figure A.2	Cartan Matrix for Type B	444
Figure A.3	Cartan Matrix for Type C	444
Figure A.4	Cartan Matrix for Type D	445
Figure A.5	Cartan Matrix for Type E_6	445
Figure A.6	Cartan Matrix for Type E_7	445
Figure A.7	Cartan Matrix for Type E_8	446
Figure A.8	Cartan Matrix for Type F_4	446
Figure A.9	Cartan Matrix for Type G_2	446
Figure A.10	Dynkin Diagram for A_n	447
Figure A.11	Dynkin Diagram for B_n	447
Figure A.12	Dynkin Diagram for C_n	447
Figure A.13	Dynkin Diagram for D_n	447
Figure A.14	Dynkin Diagram for E_6	447

Figure A.15 Dynkin Diagram for E_7	447
Figure A.16 Dynkin Diagram for E_8	448
Figure A.17 Dynkin Diagram for F_4	448
Figure A.18 Dynkin Diagram for G_2	448

Chapter 1

Introduction

1.1 A Brief Introduction to Symmetric Spaces

Symmetric spaces can be used to describe a variety of symmetries in nature. We think of them as “nice” spaces acted on by a group of symmetries or motions (a Lie group). They are of importance in many areas of science. In particular, they play a large role in mathematics and physics. They have been studied for over a century - initially over the real numbers, but also in other fields such as \mathbb{C} and the \mathfrak{p} -adics. Within mathematics we see their application in differential geometry, singularity theory, the cohomology of arithmetic subgroups, number theory, and representation theory to name a few.

The study of symmetric spaces is a branch of Lie theory. Hence, much of the structure of symmetric spaces can be learned by examining the underlying Lie Algebras and root systems. Symmetric spaces can be defined by an involution on a group. Let G be a semisimple algebraic group defined over an algebraically closed field of non-zero characteristic. Let $\bar{\theta} \in \text{Aut}(G)$ be an involution. (i.e. $\bar{\theta}^2 = 1$). Let K be the fixed point group of $\bar{\theta}$. Let

$$P = \{\chi\bar{\theta}(\chi)^{-1} \mid \chi \in G\}$$

P is known as a *symmetric space*. Note $P \cong G/K$.

The involution $\bar{\theta}$ induces an involution θ on the roots of G . We can learn a great deal about the symmetric space by looking closely at this involution. In particular, we would like to recover as much of the structure of P starting with only knowing how θ acts

on the roots.

We start by considering the underlying Lie algebra of G . Synonymous with studying the structure of a Lie Group by examining its Lie Algebra, we can study the structure of a symmetric space by considering its *local symmetric space*. Let \mathfrak{g} be the Lie Algebra of G . Let $d\bar{\theta} \in \text{Aut}(\mathfrak{g})$ be the involutorial automorphism induced by $\bar{\theta}$ on \mathfrak{g} . By abuse of notation, we write $\bar{\theta}$ for $d\bar{\theta}$. $\bar{\theta}$ induces the same involution θ . Let

$$\mathfrak{k} = \{X \in \mathfrak{g} \mid \bar{\theta}(X) = X\}$$

We define the *local symmetric space* of \mathfrak{g} relative to $\bar{\theta}$, \mathfrak{p} , as

$$\mathfrak{p} = \{X \in \mathfrak{g} \mid \bar{\theta}(X) = -X\} \quad (1.1)$$

\mathfrak{k} and \mathfrak{p} are the tangent spaces in the identity of K and P . \mathfrak{k} , the $+1$ eigenspace relative to $\bar{\theta}$, is a subalgebra of \mathfrak{g} . \mathfrak{p} , the -1 eigenspace relative to $\bar{\theta}$, is not a subalgebra of \mathfrak{g} . However, we can decompose $\mathfrak{g} = \mathfrak{k} \oplus \mathfrak{p}$. Relative to the Killing Form we have $\mathfrak{k} \perp \mathfrak{p}$.

For characteristic $\neq 2$, results obtained concerning the local symmetric space will also correspond to the symmetric space. Hence, we can begin to learn about P by studying \mathfrak{p} . Our primary task will be to recover as much as possible the structure of \mathfrak{p} with initial knowledge of the roots and θ .

In this thesis we are concerned with the behavior of involutorial automorphisms on the roots of a Lie algebra. We want to describe how the involution relates to the corresponding involution on the algebra itself. In particular, we wish to recover the action $\bar{\theta}$ in a fashion suitable for computation.

We then turn our attention to the roots of \mathfrak{g} . Recall the *roots* of \mathfrak{g} are the functionals for which each vector in \mathfrak{g} not in \mathfrak{t} , the toral subalgebra, are associated. In our case, \mathfrak{t} is maximal, and each root is associated with precisely one vector in $\mathfrak{g} \setminus \mathfrak{t}$. For the Lie Bracket $[\cdot, \cdot]$, the roots α, β and their corresponding vectors X_α, X_β satisfy the relationships

$$[X_\alpha, X_\beta] = N_{\alpha, \beta} X_{\alpha+\beta} \quad \text{where } N_{\alpha, \beta} \text{ is a scalar}$$

$$[H_\alpha, X_\beta] = \beta(H_\alpha) X_\beta \quad \text{where } H_\alpha \in \mathfrak{t}$$

Let Φ denote the set of roots of \mathfrak{g} . For $\bar{\theta} \in \text{Aut}(\mathfrak{g})$ and $\theta \in \text{Aut}(\Phi)$ we also have the relationship

$$\bar{\theta}(X_\alpha) = c_{\alpha,\theta} X_{\theta(\alpha)}$$

The constants $c_{\alpha,\theta}$ are the *structure constants* of θ .

1.2 Recovering the Action of an Involutorial Automorphism on the Lie Algebra

We say $\theta \in \text{Aut}(\Phi)$ can be *lifted* to an automorphism in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$ if there is a $\bar{\theta} \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ such that $\bar{\theta}|_{\mathfrak{t}} = \theta$. Steinberg proved that this can always be done [4].

Suppose θ is an involutorial automorphism. Our primary goal will be to determine if θ can be lifted to an automorphism in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$ of the same order. This is not guaranteed. In some cases lifting is “automatic,” that is, depending on the structure constants $c_{\alpha,\theta}$ (see Definition 4.2), $\bar{\theta}$ may already be an involutorial automorphism. Our first task is to determine if we are so lucky.

In the case that we are not, we must modify $\bar{\theta}$ in such a way that we still have an automorphism. $\bar{\theta}$ can only be modified with an element of $\text{ad}(\mathfrak{t})$, because we do not want to lose the condition that $\bar{\theta}|_{\mathfrak{t}} = \theta$. So our second task, should lifting not be “automatic,” is to determine some vector $H \in \mathfrak{t}$ so that we have the correct element of $\text{ad}(\mathfrak{t})$.

Finding the *correction vector* $H \in \mathfrak{t}$ is computationally intensive. The “straight-forward” approach to be introduced in Chapter 4 is sufficient for small cases, but is too inefficient for larger cases. Most of our efforts will be spent describing a “divide-and-conquer” approach which uses the original algorithm for the small pieces. In the process we’ll describe how an involution on the roots can always be lifted.

1.3 A Brief Overview

We begin in Chapter 2 by establishing the notation and some key ideas we frequently will refer to. In particular, we will set up a “library” of identities which will prove useful establishing the “divide-and-conquer” approach to finding the correction vector.

In Chapter 3 we establish important computational tools. We will spend most of our efforts describing Groebner bases, which our first lifting algorithm will require.

In Chapter 4 we will give our first algorithm which accomplishes all tasks describes above. That is, we will determine if lifting is necessary, and if so, the correction vector. However, since we will rely on Groebner bases (the construction of which takes exponential time), it will prove to be extraordinarily inefficient for even modest sized (exceeding 10 simple roots) root systems.

In Chapter 5 we will extend the above algorithm to establish the *admissibility* of an involution on the roots. This algorithm will have the same drawbacks as that of which we discuss in Chapter 4.

In Chapter 6 we will take a step back from our computational efforts and reflect on the Weyl group ¹. The results derived will lay the foundation for the proceeding work.

In Chapter 7 we seek an alternative procedure to determine if lifting is necessary. This procedure will rely only on the *Helminck diagram*, a diagram which extends the Dynkin diagram with additional information concerning θ . In the process of establishing this procedure, we discuss key ideas concerning how a diagram and its corresponding involution on the roots can be “decomposed”. We will call this decomposition the *restricted rank one decomposition*.

In Chapter 8 we extend our discussion from Chapter 7 and determine how involutions on the Lie algebra can be “decomposed.” We propose an improved lifting algorithm which works on each component from our restricted rank one decomposition, then “glues” the components together again.

In Chapter 9 we provide supporting algorithms. These algorithms are not part of the main theories and ideas we will discuss. However, they are needed if one is to implement our algorithms. They provide the key components so that one may build a complete system for constructing involutions in local symmetric spaces from involutions on the roots.

Finally, in Chapter 12 we give Mathematica source code for a local symmetric spaces package. Our package provides enough procedures to implement all algorithms described. Chapter 11 gives a complete manual for installation and operation of the package.

¹Pun fully intended

Chapter 2

Preliminary Topics in Lie Algebra

The goal of this chapter is to discuss relevant elementary topics in Lie Algebra, and establish the environment in which we'll be working in. We will first review some basic concepts (primarily to establish the notation). Our basic reference for this aim will be Humphreys' *Introduction to Lie Algebras and Representation Theory* [1]. His notation and terminology shall be used.

We will follow this discussion with some “advanced” concepts - primarily those concerning Chevalley constants - which will be useful for the latter chapters. We have two primary sources for this discussion: the papers of S. Klein [11] and Vavilov, Nikolai and Eugene Plotkin [13]. We will conclude with an example that illustrates some issues we will need to address in Chapter 4.

A separate preliminary chapter on computational techniques will follow. We'll begin with the root space decomposition.

2.1 The Root Space Decomposition

Let \mathfrak{g} be a non-zero, semisimple Lie Algebra over an algebraically closed field F . This will be the case for the entirety of our discussion. Let $\mathfrak{t} \subset \mathfrak{g}$ be a Cartan Subalgebra. Then the **root space decomposition** of \mathfrak{g} with respect to \mathfrak{t} is given by

$$\mathfrak{g} = \mathfrak{t} \oplus \sum_{\alpha \in \Phi(\mathfrak{t})} \mathfrak{g}_{\alpha} \tag{2.1}$$

Where

$$\mathfrak{g}_\alpha = \{X \in \mathfrak{g} \mid [H, X] = \alpha(H)X \quad \forall H \in \mathfrak{t}\}$$

and

$$\Phi(\mathfrak{t}) = \{\alpha \in \mathfrak{t}^* \mid \alpha \neq 0, \mathfrak{g}_\alpha \neq 0\}$$

The existence of such a decomposition arrives as follows. Given that \mathfrak{g} is not nilpotent, we can find some element $x \in \mathfrak{g}$ whose semisimple part in the abstract Jordan decomposition is nonzero. Recall the abstract Jordan decomposition lets us write $x = x_s + x_n$ for $x \in \mathfrak{g}$, where x_s is the ad-semisimple part, and x_n is the ad-nilpotent part. Hence, we can find \mathfrak{t} , a maximal subalgebra (called *toral*) in \mathfrak{g} . Note that \mathfrak{t} is abelian. Hence, $\text{ad}_{\mathfrak{g}} \mathfrak{t}$ is simultaneously diagonalizable. Thus, \mathfrak{g} is the direct sum of the subspaces \mathfrak{g}_α where α ranges over \mathfrak{t}^* .

The linear functionals α are known as the *roots* of \mathfrak{g} with respect to \mathfrak{t} , and $\{\mathfrak{g}_\alpha \mid \alpha \in \Phi(\mathfrak{t})\}$ the *root space* associated with α .

Note that \mathfrak{g}_0 is the centralizer of \mathfrak{t} . Because in our case \mathfrak{t} is maximal, we write $\mathfrak{g}_0 = \mathfrak{t}$. Note 0 is not regarded as a root, and \mathfrak{g}_0 is not regarded as a root space.

Hence, for every $\alpha \in \Phi(\mathfrak{t})$ we have $\dim(\mathfrak{g}_\alpha) = 1$. We designate $\{X_\alpha\}$, $X_\alpha \in \mathfrak{g}$ as a basis for \mathfrak{g}_α . X_α is known as the *root vector* associated with α . For our purposes, we can think of the root as a “label” for the root vector which describes how it interacts with other vectors. We will want to pay close attention to how the vectors in \mathfrak{g} are “labelled” by the roots. In particular, we’ll soon introduce the notion of a Chevalley basis. The Chevalley basis gives us an ideal basis for \mathfrak{g} and “labelling” of these vectors by the roots.

2.2 Root Systems

As suggested at the close of the previous section, understanding how the roots of a Lie Algebra relate to each other will reveal much of the structure of the Lie Algebra itself. In fact, with regards to the local symmetric space, this is the very subject of our entire discussion! To begin, let us summarize some of the important properties. Please refer to Humphreys’ text for a detailed development of these conclusions.

Proposition 2.2.1 (Humphreys. [1], §8.5).

1. $\alpha \in \Phi$ implies $\dim \mathfrak{g}_\alpha = 1$.
2. If $\alpha \in \Phi$, then the only scalar multiples of α which are roots are α and $-\alpha$.
3. If $\alpha, \beta \in \Phi$ then $\beta(H_\alpha) \in \mathbb{Z}$, and $\beta - \beta(H_\alpha)\alpha \in \Phi$.
4. If $\alpha, \beta, \alpha + \beta \in \Phi$, then $[\mathfrak{g}_\alpha, \mathfrak{g}_\beta] = \mathfrak{g}_{\alpha+\beta}$.
5. Let $\alpha, \beta \in \Phi, \beta \neq \pm\alpha$. Let r, q be (respectively) the largest integers for which $\beta - r\alpha$, $\beta + q\alpha$ are roots. Then all $\beta + i\alpha \in \Phi, (-r \leq i \leq q)$, and $\beta(H_\alpha) = r - q$.
6. \mathfrak{g} is generated (as Lie algebras) by the root spaces \mathfrak{g}_α .

The proposition enables us to understand how vectors in \mathfrak{g} relate by associating each vector with a root. To see how the roots themselves relate to one another, it will be very beneficial to describe them as vectors in \mathbb{R}^n . Doing so will enable us to use the geometry of \mathbb{R}^n to examine symmetries and combinatorial relations of our vectors.

This task is surprisingly tricky to accomplish, and requires a bit of work. We will want to describe the roots as vectors in a real Euclidean space \mathbf{E} . To do so, we will need to define the appropriate inner product. We begin with the familiar Killing Form.

Definition 2.2.2 (Killing Form). *Let \mathfrak{g} be a Lie Algebra over a field F . Define a bilinear form κ on \mathfrak{g} as follows.*

$$\kappa : \mathfrak{g} \times \mathfrak{g} \rightarrow F$$

$$\kappa(X, Y) = \text{tr}(\text{ad } X \cdot \text{ad } Y)$$

Clearly κ is a symmetric bilinear form, and is associative with respect to the Lie Bracket. With regards to κ 's action on the torus, we have the following

Proposition 2.2.3 (Humphreys. [1], §8.2). *The restriction of κ to \mathfrak{t} is nondegenerate*

This proposition enables us to identify \mathfrak{t}^* with \mathfrak{t} by means of an isomorphism. The relation identifies each element $\alpha \in \mathfrak{t}$ with a unique vector $t_\alpha \in \mathfrak{t}$ satisfying $\alpha(t) = \kappa(t_\alpha, t)$ for all $t \in \mathfrak{t}$. We have the isomorphism

$$\tau : \mathfrak{t}^* \rightarrow \mathfrak{t}$$

$$\tau(\alpha) = t_\alpha$$

Hence if we apply τ to our roots, we have $\tau(\Phi(\mathfrak{t})) = \{t_\alpha \mid \alpha \in \Phi(\mathfrak{t})\}$. This allows us to relate the killing form to \mathfrak{t}^* in the following way:

Definition 2.2.4. *Let $\alpha, \beta \in \mathfrak{t}^*$. Define a bilinear form on \mathfrak{t}^* as follows*

$$(\cdot, \cdot) : \mathfrak{t}^* \times \mathfrak{t}^* \rightarrow \kappa$$

$$(\alpha, \beta) = \kappa(t_\alpha, t_\beta)$$

We have that (\cdot, \cdot) is a symmetric bilinear form (as κ is), and is non-degenerate by Proposition 2.2.3.

The following proposition enables us to begin constructing the Euclidean Space \mathbf{E} .

Proposition 2.2.5 (Humphreys. [1], §8.3). $\Phi(\mathfrak{t})$ spans \mathfrak{t}^* .

We then pick a basis for \mathfrak{t}^* . Let $\Delta = \{\alpha_1, \alpha_2, \dots, \alpha_n\} \subset \Phi(\mathfrak{t})$ be our basis for \mathfrak{t}^* . Proposition 2.2.1-(3). leads us to a second useful conclusion.

Proposition 2.2.6 (Humphreys. [1], §8.5). *If $\alpha, \beta \in \Phi(\mathfrak{t})$, then $\frac{2(\beta, \alpha)}{(\alpha, \alpha)} \in \mathbb{Z}$, and $\beta - \frac{2(\beta, \alpha)}{(\alpha, \alpha)}\alpha \in \Phi(\mathfrak{t})$.*

In light of Proposition 2.2.1-(3), we refer to the integers $\frac{2(\beta, \alpha)}{(\alpha, \alpha)}$ as the *Cartan Integers*.

It turns out for any $\beta \in \Phi$, we can write β uniquely as $\beta = \sum_{i=1}^n k_i \alpha_i$. In fact, $k_i \in \mathbb{Q}$. Let $\mathbf{E}_{\mathbb{Q}}$ denote the \mathbb{Q} -span of α . We have the following statement.

Theorem 2.2.7 (Humphreys. [1], §8.5).

1. Each $\alpha \in \Phi(\mathfrak{t})$ is contained in $\mathbf{E}_{\mathbb{Q}}$.
2. The \mathbb{Q} -dimension of $\mathbf{E}_{\mathbb{Q}}$ equals $\dim \mathfrak{t}^*$.

The assertion allows us to view our symmetric bilinear form (\cdot, \cdot) over $\mathbf{E}_{\mathbb{Q}}$. Another result due to Humphreys is as follows.

Proposition 2.2.8 (Humphreys. [1] §8.5).

1. $(\cdot, \cdot)|_{\mathbf{E}_{\mathbb{Q}}}$ is symmetric, positive-definite
2. For any $\alpha, \beta \in \mathbf{E}_{\mathbb{Q}}$, $(\alpha, \beta) \in \mathbb{Q}$.

We finish our construction by extending the base field from \mathbb{Q} to \mathbb{R} . Let $\mathbf{E} = \mathbb{R} \otimes_{\mathbb{Q}} \mathbf{E}_{\mathbb{Q}}$. The form (\cdot, \cdot) extends to \mathbf{E} and is positive-definite. Hence \mathbf{E} is a Euclidean Space, Φ contains a basis of \mathbf{E} , and $\dim_{\mathbb{R}} \mathbf{E} = n$.

The construction allows us to study the roots by means of their geometric properties via our real Euclidean inner product space. To summarize the construction, we have:

Theorem 2.2.9 (Humphreys. [1], §8.5). *Let \mathbf{E} be as defined above, with form (\cdot, \cdot) . Let \mathfrak{g} be a Lie Algebra with maximal torus \mathfrak{t} and roots $\Phi(\mathfrak{t})$. Then*

1. $\Phi(\mathfrak{t})$ spans \mathbf{E} .
2. $0 \notin \mathfrak{t}$.
3. The only multiples of $\alpha \in \Phi(\mathfrak{t})$ are ± 1 .
4. For all $\alpha, \beta \in \Phi(\mathfrak{t})$, $\beta - \frac{2(\beta, \alpha)}{(\alpha, \alpha)}\alpha \in \Phi(\mathfrak{t})$.
5. For all $\alpha, \beta \in \Phi(\mathfrak{t})$, $\frac{2(\beta, \alpha)}{(\alpha, \alpha)}\alpha \in \mathbb{Z}$.

2.3 Root Systems in \mathbb{R}^n

The stage is now set to begin our geometric description of the roots. Fix a Euclidean space \mathbf{E} with a symmetric positive-definite bilinear form (\cdot, \cdot) . We first define the length of a vector and the angle between two vectors in the usual way:

Definition 2.3.1.

1. Let $\alpha \in \mathbf{E}$. Then $|\alpha| = \sqrt{(\alpha, \alpha)}$.
2. Let $\alpha, \beta \in \mathbf{E}$, and θ be the angle between α and β . Then $\cos(\theta) = \frac{(\alpha, \beta)}{(\alpha, \alpha)}$.

A *reflection* in \mathbf{E} is an invertible linear transformation constructed in a particular way. Given a root α , we fix a *reflecting hyperplane* $P_{\alpha} = \{\beta \in \mathbf{E} \mid (\beta, \alpha) = 0\}$. Any root orthogonal to P_{α} is sent into its negative. We then define a reflection explicitly:

Definition 2.3.2. Let $\alpha, \beta \in \mathbf{E}$. A reflection about the root α , denoted s_α , is defined as

$$s_\alpha(\beta) = \beta - \frac{2(\beta, \alpha)}{(\alpha, \alpha)}\alpha \quad (2.2)$$

For convenience, we usually write

$$\langle \beta, \alpha \rangle = \frac{2(\beta, \alpha)}{(\alpha, \alpha)} \quad (2.3)$$

Then we have in place of equation 2.2

$$s_\alpha(\beta) = \beta - \langle \beta, \alpha \rangle \alpha \quad (2.4)$$

We are now suitably equipped to define a root system in \mathbb{R}^n .

Definition 2.3.3. A subset Φ of the Euclidean space \mathbf{E} is called a root system in \mathbf{E} if the following five axioms are satisfied.

1. Φ is finite and spans \mathbf{E} .
2. $0 \notin \Phi$.
3. For $\alpha \in \Phi$, if $k\alpha \in \Phi$ then $k = \pm 1$.
4. If $\alpha \in \Phi$, then s_α leaves Φ invariant.
5. If $\alpha, \beta \in \Phi$ then $\langle \beta, \alpha \rangle \in \mathbf{Z}$.

The fifth axiom imposes severe restrictions on the possible angles between pairs of roots. Considering Definition 2.3.1-(2). we have

$$\langle \beta, \alpha \rangle \cdot \langle \alpha, \beta \rangle = 4 \cos^2(\theta) \quad (2.5)$$

We note that $\langle \beta, \alpha \rangle$ and $\langle \alpha, \beta \rangle$ have the same sign. Since $\cos^2(\theta) \in [0, 1]$, then there is a finite number of angles between pairs of roots. We can let $|\beta| \geq |\alpha|$ without loss of generality. Then the following table summarizes the possible angles between pairs of roots that preserves the condition $\langle \beta, \alpha \rangle \in \mathbf{Z}$.

Table 2.1: Possible Angles Between Pairs of Roots

$\langle \alpha, \beta \rangle$	$\langle \beta, \alpha \rangle$	θ	$ \beta ^2/ \alpha ^2$
0	0	$\pi/2$	Undetermined
1	1	$\pi/3$	1
-1	-1	$2\pi/3$	1
1	2	$\pi/4$	2
-1	-2	$3\pi/4$	2
1	3	$\pi/6$	3
-1	-3	$5\pi/6$	3

As a remark, for a particular root system we are concerned more with the angle between vectors than the vectors themselves. Hence, let Φ be a root system in Euclidean space \mathbf{E} with a given bilinear form. Then any form which leaves the Cartan integers invariant maintains the structure of the root system up to scale.

As a second remark, we have the following theorem in Humphreys [1].

Theorem 2.3.4. $\Phi(t)$ is a root system in \mathbf{E} .

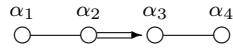
This theorem is of enormous importance. It gives that every nonzero finite dimensional semisimple Lie algebra \mathfrak{g} is characterized up to isomorphism by its root system. By means of the root system we hope to study its corresponding algebra. We next show that there are a finite number of classes of root systems. Hence, there are a finite number of classes of nonzero finite dimensional semisimple Lie algebras.

2.4 Cartan Matrices and Dynkin Diagrams

An indispensable tool for understanding the structure of the root system will be its Cartan matrix. Fix an ordering of the simple roots (those which are not the sum of any other roots). The matrix whose (i, j) entry is given by $(\langle \alpha_i, \alpha_j, \rangle)$ is the Cartan matrix for the root system. The matrix depends on the ordering chosen. For our discussions we will use the ordering given in Humphreys. However, our algorithms will work for any chosen ordering.

A second invaluable tool is the Dynkin diagram. The Dynkin diagram extends the Coxeter graph with additional information concerning the roots. The roots α_i and α_j are joined by $\langle \alpha_i, \alpha_j \rangle \langle \alpha_j, \alpha_i \rangle$ number of edges. The possible values are 0, 1, 2, or 3. In the case of a double or triple bond (2 or 3), an arrow points to the shorter of the two roots.

The Cartan matrix and Dynkin diagram can be easily determined from each other - an idea we will make frequent use of in our algorithms. In particular, consider the roots α_i and α_j . In the case of a single bond, both the (i, j) and (j, i) entries hold the value -1 . In the case of a double or triple bond, either the (i, j) entry or (j, i) entry holds the value -2 (double bond), or -3 (triple bond). The other entry holds the value -1 . The value which holds -1 falls on the row which the arrow points to. For instance, given the diagram



we can deduce the Cartan matrix in the following way. First, all entries on the diagonal hold the value 2. Because we have single bonds between the pairs of roots (α_1, α_2) and (α_3, α_4) , the $(1, 2)$, $(2, 1)$, $(3, 4)$, and $(4, 3)$ entries hold the value -1 . For the pair (α_2, α_3) , the arrow points toward α_3 . Hence, the $(3, 2)$ entry (the entry on row 3) holds the value -1 . The $(2, 3)$ entry holds the value -2 . Remaining entries hold the value 0. We have the Cartan matrix

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

The seven classification classes of possible Dynkin diagrams are listed in Humphreys [1]. Because the diagrams classify the possible root systems, they classify the possible nonzero finite dimensional semisimple Lie algebras. The ordering of the roots does not matter - we will have the same classifications. However, because we will frequently rely on these diagrams and matrices, we will want to pick one ordering for consistency. For reference, the Cartan matrices and Dynkin diagrams relevant to our ordering are given in Appendix A.

2.5 Bases of the Root Systems

Considering our geometric description of the root systems, we are now able to establish bases for each system in terms of orthonormal vectors which span \mathbb{R}^n . The inner product is the usual one

$$\langle e_i, e_j \rangle = \delta_{i,j}$$

where

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{else} \end{cases}$$

Our usual basis for each of the root systems will be as follows.

Table 2.2: Usual Root System Bases

Name	Basis
A_n	$e_i - e_{i+1}, i = 1 \dots n$
B_n	$e_i - e_{i+1}, i = 1 \dots n - 1;$ e_n
C_n	$e_i - e_{i+1}, i = 1 \dots n - 1;$ $2e_n$
D_n	$e_i - e_{i+1}, i = 1 \dots n - 1;$ $e_{n-1} + e_n$
E_6	$\{\frac{1}{2}(e_1 - e_2 - e_3 - e_4 - e_5 - e_6 - e_7 + e_8), e_1 + e_2,$ $-e_1 + e_2, -e_2 + e_3, -e_3 + e_4, -e_4 + e_5\}$
E_7	$\{\frac{1}{2}(e_1 - e_2 - e_3 - e_4 - e_5 - e_6 - e_7 + e_8), e_1 + e_2,$ $-e_1 + e_2, -e_2 + e_3, -e_3 + e_4, -e_4 + e_5, -e_5 + e_6\}$
E_8	$\{\frac{1}{2}(e_1 - e_2 - e_3 - e_4 - e_5 - e_6 - e_7 + e_8), e_1 + e_2,$ $-e_1 + e_2, -e_2 + e_3, -e_3 + e_4, -e_4 + e_5, -e_5 + e_6, -e_6 + e_7\}$
F_4	$\{e_2 - e_3, e_3 - e_4, e_4, \frac{1}{2}(e_1 - e_2 - e_3 - e_4)\}$
G_2	$\{e_1 - e_2, -2e_1 + e_2 + e_3\}$

It should be emphasized that this is not the only choice of bases. Indeed, a question we will need to later address is how one can identify which root system a given set of vectors describes. A procedure, using Cartan matrices, can be found in Algorithms 9.1.1 and 9.1.2.

As an example, consider the basis $\Delta = \{\frac{1}{2}(e_1 - e_2 + e_6 - e_7), \frac{1}{2}(e_2 - e_6)\}$. Let α_i denote the i^{th} entry, and let α_i^\vee denote the co-root of α_i . Then the matrix formed by $[(\alpha_i, \alpha_j^\vee)]_{i,j=1,2}$ yields

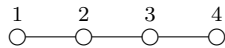
$$\begin{pmatrix} 2 & -2 \\ -1 & 2 \end{pmatrix}$$

which is precisely the Cartan Matrix for a root system of type B_2 .

2.6 Weyl Groups and the Longest Element

The subgroup of $GL(\mathbf{E})$ generated by the reflections s_α gives the Weyl group (denoted by W). The longest element, denoted w_0 , denotes the longest element of the Weyl group which cannot be expressed as a shorter element. For example, consider the root system A_2 , with two basis roots α and β . The longest element of the Weyl group is $s_\alpha s_\beta s_\alpha$, which has three elements. Because this is the longest element, any expression which is longer can be expressed equivalently as a shorter element. $s_\alpha s_\beta s_\alpha s_\beta$ has four elements, but is equivalent to $s_\beta s_\alpha$.

The longest element is of particular interest to us in that it acts invariantly on an “embedded root system.” An embedded root system is constructed by looking at the root system described by part of a Dynkin diagram. Consider the Dynkin diagram for the root system A_4 .



We can look at the sub-diagram constructed by removing the first and last roots.



This is an A_2 root system embedded in A_4 . The longest element of this system with respect to the two basis roots is $w_0 = s_{\alpha_2}s_{\alpha_3}s_{\alpha_2}$.

The longest element has the property that roots of A_4 which also lie in the embedded root system will be mapped by w_0 to another root in the embedded root system. For instance,

$$w_0(\alpha_2) = \alpha_3 \in \text{embedded } A_2$$

$$w_0(\alpha_3) = \alpha_2 \in \text{embedded } A_2$$

This follows from the fact that w_0 maps Δ to $-\Delta$. So if Δ is a basis for the roots in the embedded root system, all root in the larger root system that are precisely linear combinations of embedded basis elements must be mapped to the embedded system by w_0 .

2.7 Chevalley Constants

The advantage to identifying each basis vector in \mathfrak{g} with the correct root is that we can quickly calculate the Lie bracket given two vectors. this is done in the following manner. The relation between the root system and its corresponding Lie algebra gives us that $[X_\alpha, X_\beta] = kX_{\alpha+\beta}$, where k is some constant. In general there are few cases where we can tell the value of k knowing only the roots. However, it is possible to choose a basis for \mathfrak{g} so that this is possible. We will often want to take a *Chevalley basis* of \mathfrak{g} , for which the constant k has special properties. These properties allow us to quickly determine its value for all pairs of root vectors. To construct a Chevalley basis, we choose root vectors satisfying several properties. In particular, we have the following definition for this constant:

Definition 2.7.1. *If $\alpha + \beta$ is a root and p is the greatest integer such that $\beta - p\alpha$ is a root, then $[X_\alpha, X_\beta] = N_{\alpha,\beta}X_{\alpha+\beta}$ where $N_{\alpha,\beta} = \pm(p+1)$.*

We have the fundamental properties:

1. $[H_{\alpha_i}, H_{\alpha_j}] = 0$
2. $[H_{\alpha_i}, X_{\alpha_j}] = \langle \alpha_j, \alpha_i \rangle X_{\alpha_j}$
3. $[X_\alpha, X_{-\alpha}] = H_\alpha$
4. $N_{\alpha,-\beta} = -N_{-\alpha,\beta}$

2.8 A Library of Identities on Chevalley Constants

Some of our algorithms (and justification of the underlying theory) will depend on clever manipulation of the Chevalley constants. In particular, there may be more than one Chevalley basis for a given Lie algebra. We'll often want to manipulate the Chevalley constants so that the particular choice of Chevalley basis becomes arbitrary. For this reason we provide a list of the most useful identities. Enough are provided so that the particular Chevalley constants we need can be computed from the root system. Our references for this collection are the works of N. Vavilov and E. Plotkin [13], and S. Klein [11].

The proceeding set can be found in [13]. The first three identities help establish the rest that follow. The first immediately follows from the definition of the Chevalley constants. The second two allow us to manipulate the order of the roots in the subscripts. We have the following.

Proposition 2.8.1. *Let $\alpha, \beta \in \Phi$ and $\alpha + \beta \neq 0$ then*

1. *If $\alpha + \beta \notin \Phi$ then*

$$N_{\alpha,\beta} = 0 \tag{2.6}$$

2. *If $\alpha + \beta \in \Phi$ then*

$$N_{\alpha,\beta} = -N_{\beta,\alpha} \tag{2.7}$$

3. *If $\alpha + \beta \in \Phi$ then*

$$N_{\alpha,-\beta} = N_{\beta,-\alpha} \tag{2.8}$$

The next proposition gives us a foundation for determining specific values of the Chevalley constants. Following from the definition of the Chevalley constants 2.7.1 (namely, $N_{\alpha,\beta} = \pm(p+1)$), we have

Proposition 2.8.2. *Let $\alpha, \beta \in \Phi$, $\alpha + \beta \in \Phi$, and $\alpha + \beta \neq 0$ then*

$$N_{\alpha,\beta} N_{-\alpha,-\beta} = -(p+1)^2 \tag{2.9}$$

Next, the Jacobi identity provides us two more “advanced” relations.

Proposition 2.8.3. *Let $\alpha, \beta, \gamma, \delta \in \Phi$. Then*

1. *If $\alpha + \beta + \gamma = 0$ then*

$$\frac{N_{\alpha,\beta}}{(\gamma,\gamma)} = \frac{N_{\beta,\gamma}}{(\alpha,\alpha)} = \frac{N_{\gamma,\alpha}}{(\beta,\beta)} \quad (2.10)$$

2. *If $\alpha + \beta + \gamma + \delta = 0$ then*

$$\frac{N_{\alpha,\beta}N_{\gamma,\delta}}{(\alpha+\beta,\alpha+\beta)} + \frac{N_{\beta,\gamma}N_{\alpha,\delta}}{(\beta+\gamma,\beta+\gamma)} + \frac{N_{\gamma,\alpha}N_{\beta,\delta}}{(\gamma+\alpha,\gamma+\alpha)} = 0 \quad (2.11)$$

The next two results concern the matrix formed by the Chevalley constants. We obtain this matrix by letting the (i, j) entry take the value N_{α_i, α_j} for $\alpha \in \Delta$, or $\alpha \in \Phi$. First we have:

Proposition 2.8.4. *From Equation 2.7 we have that the matrix*

$$N^+ = [N_{\alpha_i, \alpha_j}]_{i,j=1 \dots |\Phi^+|} \quad (2.12)$$

where $\alpha_i, \alpha_j \in \Phi^+$ is anti-symmetric.

A second proposition immediately follows, giving the Chevalley constants concerning the negative roots.

Proposition 2.8.5. *Following Equations 2.8 and 2.10, the matrix*

$$N^- = [N_{\alpha_i, \alpha_j}]_{i,j=1 \dots |\Phi^-|} \quad (2.13)$$

where $\alpha_i, \alpha_j \in \Phi^-$ can be expressed via N^+ .

More can be said about Equations 2.10 and 2.11 if all roots have the same length. If this condition is met, we have:

Proposition 2.8.6. *Let all roots have the same length. Then,*

1. *If $\alpha + \beta + \gamma = 0$ then*

$$N_{\alpha,\beta} = N_{\beta,\gamma} = N_{\gamma,\alpha} \quad (2.14)$$

2. *If $\alpha + \beta + \gamma + \delta = 0$ then*

$$N_{\alpha,\beta} N_{\gamma,\delta} + N_{\beta,\gamma} N_{\alpha,\delta} + N_{\gamma,\alpha} N_{\beta,\delta} = 0 \quad (2.15)$$

The next three identities will prove useful when establishing relationships between an automorphism on the roots and its corresponding automorphism on the Lie algebra. If we call the preceding set of Chevalley relations the “basic” set, then what follows is the “advanced”, as making use of them requires clever re-arrangement of the roots.

Proposition 2.8.7.

1.

$$N_{\beta,\gamma} N_{\alpha,\beta+\gamma} = N_{\alpha+\beta,\gamma} N_{\alpha,\beta} \quad (2.16)$$

2.

$$N_{\alpha,\beta} = \begin{cases} -N_{\alpha_i,\beta-\alpha_i} N_{\beta-\alpha_i,\alpha} & \text{if } \beta - \alpha_i \in \Phi^+; \\ N_{\alpha_i,\alpha-\alpha_i} N_{\alpha-\alpha_i,\beta} & \text{if } \alpha - \alpha_i \in \Phi^+. \end{cases} \quad (2.17)$$

3.

$$N_{\alpha,-\beta} = \begin{cases} N_{\alpha-\beta,\beta} \frac{(\alpha-\beta,\alpha-\beta)}{(\alpha,\alpha)} & \text{if } \alpha - \beta \in \Phi^+; \\ N_{\beta-\alpha,\beta} \frac{(\beta-\alpha,\beta-\alpha)}{(\beta,\beta)} & \text{if } \beta - \alpha \in \Phi^+. \end{cases} \quad (2.18)$$

Providing a means to begin calculating numerical values for the Chevalley constants, let $\Delta = \{\alpha_1, \dots, \alpha_n\}$ be a basis for the roots of \mathfrak{g} . We have the following scheme in [14]. Set

$$N_{\alpha_i, \beta} = 1 \text{ if } \alpha_i + \beta \in \Phi^+ \quad (2.19)$$

provided there is no $j < i$ such that $\alpha_i + \beta = \alpha_j + \bar{\beta}$ for some $\bar{\beta} \in \Phi^+$.

An alternative means to compute the values of some Chevalley constants (via the roots) can be related to Equation 2.9. Two claims will help us. First follows from [12]:

Lemma 2.8.8 (Knapp, A.). *Let $\{\alpha_2 + k\alpha_1 \mid -p \leq k \leq q\}$ be the α_1 -string through α_2 . Then we have*

$$N_{\alpha_1, \alpha_2}^2 = \frac{q(1+p)}{2} \|\alpha_1\|^2$$

Second from [11].

Proposition 2.8.9 (Klein. Proposition 3.5-(2)). *For every non-simple, positive root $\alpha \in \Phi^+ - \Delta$, fix a decomposition $\alpha = \beta_1 + \beta_2$ such that $\beta_1, \beta_2 \in \Phi^+$. Then there exists a Chevalley basis $\{X_\alpha\}$ with the property that for all $\alpha \in \Phi^+$ we have $N_{\beta_1, \beta_2} > 0$.*

The implication of Proposition 2.8.9 is that there is a Chevalley basis such that we can compute some of the Chevalley constants (those for which the decomposition of the proposition can be found) in the following way.

1. for the non-simple positive roots $\alpha \in \Phi^+ - \Delta$, fix a decomposition $\alpha = \beta_1 + \beta_2$, $\beta_1, \beta_2 \in \Phi^+$.
2. Let $\{\beta_2 + k\beta_1 \mid -p \leq k \leq q\}$ be the β_1 -string through β_2 .
3. Then N_{β_1, β_2} is given by

$$N_{\beta_1, \beta_2} = \sqrt{\frac{q(1+p)}{2}} \|\beta_1\| \quad (2.20)$$

Another result in Klein [11] extends Equation 2.18. It is possible to construct a Chevalley basis so that we have the following.

Lemma 2.8.10 (Klein. Equation 3.5).

$$N_{\alpha, -\beta} = -N_{-\alpha, \beta} = \begin{cases} N_{\beta-\alpha, \alpha} & \text{if } \beta - \alpha \in \Phi^+; \\ N_{\alpha-\beta, \beta} & \text{if } \beta - \alpha \in -\Phi^+ \text{ and } N_{-\alpha, -\beta} = -N_{\alpha, \beta}; \\ 0 & \text{else.} \end{cases} \quad (2.21)$$

2.9 Example: Root Space Decomposition of $\mathfrak{sl}_n(\mathbb{C})$

Let $\mathfrak{g} = \mathfrak{sl}_n(\mathbb{C})$. Recall $\mathfrak{sl}_n(\mathbb{C})$ is the semisimple Lie Algebra of $n \times n$ matrices over \mathbb{C} with trace zero. The Lie Bracket $[\cdot, \cdot]$ on \mathfrak{g} is the commutator:

$$[X, Y] = XY - YX \quad \forall X, Y \in \mathfrak{g}$$

Let $\mathfrak{t} \subset \mathfrak{g}$ be the set of diagonal matrices.

For $i = 1 \dots n$, let e_i be the functionals on \mathfrak{t} given by

$$e_i\left(\begin{pmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \end{pmatrix}\right) = a_i$$

We can compute that the set of roots of \mathfrak{g} is given by

$$\Phi(\mathfrak{t}) = \{e_i - e_j \mid i, j = 1 \dots n, i \neq j\}$$

Define $\alpha_i = e_i - e_{i+1}$. Then $\Delta = \{\alpha_1, \alpha_2, \dots, \alpha_{n-1}\}$ is a base for $\Phi(\mathfrak{t})$. Denote by Φ the set of roots $\Phi(\mathfrak{t})$.

Consider the inner product $\langle e_i, e_j \rangle = \delta_{i,j}$. Then the reader can verify that $\langle \alpha_i, \alpha_j \rangle$ yields precisely the (i, j) entry in the Cartan Matrix for A_{n-1} . Hence, \mathfrak{g} is a Lie Algebra of type A_{n-1} .

Let $E_{i,j}$ denote the $n \times n$ matrix with 1 in the (i, j) entry and zero elsewhere. Pick $\{H_{\alpha_i} = E_{i,i} - E_{i+1,i+1}\}$ as a basis for \mathfrak{t} . We have the relation $H_{\alpha+\beta} = H_{\alpha} + H_{\beta}$

for every $\alpha, \beta \in \Phi$. Then for $\alpha = k_1\alpha_1 + \dots + k_{n-1}\alpha_{n-1}$, $\alpha \in \Phi, \alpha_i \in \Delta$, we have $H_\alpha = k_1H_{\alpha_1} + \dots + k_{n-1}H_{\alpha_{n-1}}$.

Now that we have defined a basis for \mathfrak{t} , it is the remaining task to determine the root vectors X_α with regards to \mathfrak{t} . Recall $[H_{\alpha_i}, X_{\alpha_i}] = \langle i, i \rangle X_{\alpha_i}$, where $\langle i, j \rangle$ is the (i, j) entry in the Cartan Matrix. Then $\langle i, i \rangle = 2$. Thus, for each H_{α_i} we compute the 2-Eigenspace of $\text{ad}(H_{\alpha_i})$. Since \mathfrak{t} is maximal, the dimension of the 2-Eigenspace is 1, and a basis for this space is given by $\{E_{i,i+1}\}$. Therefore we have $X_{\alpha_i} = E_{i,i+1}$ for all $\alpha_i \in \Delta$.

To find the root vectors for the non-basis roots, we can make use of the fact that $[X_\alpha, X_\beta]$ is a multiple of $X_{\alpha+\beta}$ for $\alpha, \beta \in \Phi$, or we can repeat the same Eigenspace computation for all $H_\alpha, \alpha \in \Phi$. To save time in the computation, note that $\text{ad } H_\alpha + \text{ad } H_\beta = \text{ad } H_{\alpha+\beta}$ for $\alpha, \beta \in \Phi$. To determine the root vectors for the negative roots, we repeat the computations above, using the (-2) -Eigenspaces instead. We then arrive at the following root space decomposition:

$$\mathfrak{g} = \text{span}(H_{\alpha_i}) \oplus \text{span}(M)$$

$$M = \begin{pmatrix} 0 & X_{\alpha_1} & X_{\alpha_1+\alpha_2} & \cdots & X_{\alpha_1+\dots+\alpha_{n-2}} & X_{\alpha_1+\dots+\alpha_{n-1}} \\ X_{-\alpha_1} & 0 & X_{\alpha_2} & \cdots & X_{\alpha_2+\dots+\alpha_{n-2}} & X_{\alpha_2+\dots+\alpha_{n-1}} \\ X_{-\alpha_1-\alpha_2} & X_{-\alpha_2} & 0 & \cdots & X_{\alpha_3+\dots+\alpha_{n-2}} & X_{\alpha_3+\dots+\alpha_{n-1}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ X_{-\alpha_1-\dots-\alpha_{n-2}} & X_{-\alpha_2-\dots-\alpha_{n-2}} & X_{-\alpha_3-\dots-\alpha_{n-2}} & \cdots & 0 & X_{\alpha_{n-1}} \\ X_{-\alpha_1-\dots-\alpha_{n-1}} & X_{-\alpha_2-\dots-\alpha_{n-1}} & X_{-\alpha_3-\dots-\alpha_{n-1}} & \cdots & X_{-\alpha_{n-1}} & 0 \end{pmatrix} \quad (2.22)$$

Recall the constants $N_{\alpha,\beta}$ satisfy the relation $[X_\alpha, X_\beta] = N_{\alpha,\beta}X_{\alpha+\beta}$, where $N_{\alpha,\beta} = 0$ if and only if $\alpha+\beta$ is not a root. We can then use the root space decomposition to compute these constants. In the case of $\mathfrak{sl}_3(\mathbb{C})$ we get the non-zero constants

Root α	Root β	Constant $N_{\alpha,\beta}$
α_1	α_2	1
α_1	$-\alpha_1 - \alpha_2$	-1
α_2	α_1	-1
α_2	$-\alpha_1 - \alpha_2$	1
$\alpha_1 + \alpha_2$	$-\alpha_1$	-1
$\alpha_1 + \alpha_2$	$-\alpha_2$	1
$-\alpha_1$	$\alpha_1 + \alpha_2$	1
$-\alpha_1$	$-\alpha_2$	-1
$-\alpha_2$	$\alpha_1 + \alpha_2$	-1
$-\alpha_2$	$-\alpha_1$	1
$-\alpha_1 - \alpha_2$	α_1	1
$-\alpha_1 - \alpha_2$	α_2	-1

Chapter 3

Systems of Multivariate Polynomial Equations

The goal of this chapter is to discuss computational techniques we'll rely heavily upon. In particular, we thoroughly discuss solutions to systems of multivariate polynomial equations. Once described, we will have everything we need to determine if an automorphism of the roots can be lifted to one of the Lie Algebra. Our basic references will be Von Zur Gathen and Gerhard's *Modern Computer Algebra* [2], and Cox, Little, and O'Shea's *Ideals, Varieties, and Algorithms* [3].

We will rely on Groebner bases to solve these systems. Groebner bases are a powerful computational tool. Unfortunately, algorithms which compute these bases tend to be quite slow. Once we establish how Groebner bases are relevant to our goals, we will next need to establish how to rely on them as little as possible.

3.1 Multivariate Division

Our first task is to introduce a procedure for division of multivariate polynomials analogous to long division. In the single variable case, division of polynomial $f(x)$ by $g(x)$ is accomplished in the following manner:

1. Divide the leading term of $f(x)$ by the leading term of $g(x)$. Add the resulting monomial $m(x)$, to the quotient..

2. Multiply each term of $g(x)$ by $m(x)$. Subtract this result from $f(x)$.
3. Repeat until no divisions are possible. The remaining terms of $f(x)$ compose the remainder.

A similar scheme for division with multivariate polynomials is given below. The purpose of a *monomial order* will be discussed in the following section.

Algorithm 3.1.1 (Multivariate Polynomial Division).

Input Nonzero polynomials $f, f_1, \dots, f_s \in F[x_1, \dots, x_n]$, where F is a field, and \prec , a monomial order on $F[x_1, \dots, x_n]$.

Output $q_1, \dots, q_s, r \in F[x_1, \dots, x_n]$ such that $f = q_1 f_1 + \dots + q_s f_s + r$, and no monomial r is divisible by the leading term (LT) of any f_1, \dots, f_s .

1. $r := 0$; $p := f$; $q_i := 0$ for all $i = 1, \dots, s$.

2. **while** $p \neq 0$ **do**

if $\text{LT}(f_i)$ divides $\text{LT}(p)$ for any i

then $q_i := q_i + \frac{\text{LT}(p)}{\text{LT}(f_i)}$; $p := p - f_i \frac{\text{LT}(p)}{\text{LT}(f_i)}$

else $r := r + \text{LT}(p)$; $p := p - \text{LT}(p)$

3. **return** q_1, \dots, q_s, r

The algorithm must terminate because after each pass through step 2, the *multidegree* of f has decreased, where the multidegree of f is the maximum (with respect to the order \prec) n -tuple formed by the powers of x_i in each monomial of f .

3.2 Monomial Ordering

The leading term of a multivariate polynomial is ambiguous without defining an order on the monomials. For instance, suppose $f(x) = 3x^3y^2 + 2x^2y^5$. Should the first term be considered the leading term because the power of the first variable (x) is higher, or should the latter term be considered leading because the sum of powers is higher?

To settle the issue, we define a monomial order as follows:

Definition 3.2.1. A total order \prec on \mathbb{N}^n is a **monomial order** in $F[x_1, \dots, x_n]$ if it satisfies the following two conditions.

1. $\alpha \prec \beta$ if $\alpha + \gamma \prec \beta + \gamma$ for all $\alpha, \beta, \gamma \in \mathbb{N}^n$
2. \prec is a well order

Two common orders are lexicographical, and graded lexicographical. In lexicographical order, $\alpha \prec \beta$ if the left-most entry in $\alpha - \beta$ is negative. Hence, to determine if $\alpha \prec \beta$, we look at the left-most entry. If α 's is the lowest, then $\alpha \prec \beta$. If there is a tie, we move to the second left-most entry, or third, etc. In graded lexicographical order, we compare the sums of the entries of α and β . In the case of a tie, we fall back to lexicographical order.

Hence, for $f(x) = 3x^3y^2 + 2x^2y^5$, we have the tuples $(3, 2)$ and $(2, 5)$. Using lexicographical order, $(2, 5) \prec (3, 2)$ and so $3x^3y^2$ is the leading term. Using graded lexicographical order, $(3, 2) \prec (2, 5)$ and $2x^2y^5$ is the leading term.

3.3 Groebner Bases

With multivariate polynomial division we are now able to construct a *Groebner Basis*. Let F be an algebraically closed field, and $I = \langle f_1, \dots, f_s \rangle$ a polynomial ring over F . Our task is to find a finite subset $G = \{g_1, \dots, g_t\}$ such that $\langle \text{LT}(g_1), \dots, \text{LT}(g_t) \rangle = \langle \text{LT}(I) \rangle$.

G satisfying the above criteria describes a Groebner Basis. Equivalently, we can define a Groebner Basis as follows:

Definition 3.3.1. A basis G of an ideal I in a polynomial ring over an algebraically closed field F is a **Groebner Basis** if it satisfies

1. The ideal given by the leading terms of polynomials in I is itself generated by the leading terms of the basis G
2. The leading term of any polynomial in I is divisible by the leading term of some polynomial in the basis G
3. Multivariate division of any polynomial in the polynomial ring R by G gives a unique remainder.

4. *Multivariate division of any polynomial in the ideal I by G gives 0*

The classic algorithm for the computation of such a set G was given by Buchberger. His algorithm can be viewed as a generalization of the Euclidean Algorithm for GCD computation, and Gaussian Elimination for linear systems. It is described below.

First, define the **S-Polynomial** of two polynomials f and g as follows.

$$S(f, g) = \frac{x^\gamma}{\text{LT}(f)}f - \frac{x^\gamma}{\text{LT}(g)}g$$

where $x^\gamma = \text{LCM}(\text{LM}(f), \text{LM}(g))$

The S stands for “subtraction”, or “syzygy.” It has the effect of “cancelling” the leading terms of f and g . The objective is to remove polynomials whose leading terms are not in the ideal generated by leading terms of G .

Let \bar{f}^G denote the remainder of polynomial f divided by polynomials in G .

Algorithm 3.3.2 (Buchberger’s Algorithm).

Input $F = \{f_1, \dots, f_s\}$

Output A Groebner Basis $G = \{g_1, \dots, g_t\}$ for I .

1. $G := F; G' := \{\}$

2. **while** $G' \neq G$ **do**

$G' := G$

for each pair (p, q) of polynomials in G' , $p \neq q$, **do**

$S := \overline{S(p, q)}^{G'}$

if $S \neq 0$ **then** $G := G \cup \{S\}$

3.4 Application to Systems of Multivariate Polynomials

Consider, for example, the ideal $I = \langle xy + x^2, x + y + z^3, 2x - y \rangle$. One can compute the following Groebner Basis $G = \{3z + z^6, 3y + 2z^3, 3x + z^3\}$. The attractive property of G is that one polynomial contains only the variable z . Each subsequent polynomial introduces, for the first time, a new variable.

What this implies is the following. Suppose we have a system of polynomial equations $f_i = 0, i = 1, \dots, s$. To solve this system, we consider the ideal $I = \langle f_1, f_2, \dots, f_s \rangle$. Computation of a Groebner Basis for this ideal will not change the solution set, because each polynomial in G is a combination of polynomials in I . But G has the property that we can use back-substitution.

To take advantage of this property, we introduce elimination ideals.

Definition 3.4.1. *Let F be a field and $I = \langle f_1, \dots, f_s \rangle \subset F[x_1, \dots, x_n]$. Then the l^{th} elimination ideal of I is given as $I_l = I \cap F[x_l, \dots, x_n]$.*

The above definition implies that, to find I_l , we remove all polynomials except those for which only the variables x_l, \dots, x_n are present. If we were to do the same for the polynomials in G , we would be left with a Groebner Basis for I_l as shown in the following theorem.

Theorem 3.4.2. *Let I be an ideal and G a Groebner Basis for I . Then*

1. I_l is an ideal for all $l = 1, \dots, n$
2. $G_l = G \cap F[x_1, \dots, x_n]$ is a Groebner Basis for I_l

The theorem implies we can solve a system of equations $f_i(x_1, x_2, \dots, x_n) - b_i = 0$ in the following manner.

Algorithm 3.4.3 (Solutions to Systems of Multivariate Polynomial Equations).

Input Polynomials g_1, \dots, g_s , where $g_i = f_i(x_1, x_2, \dots, x_n) - b_i$

Output All solutions to the system $f_i(x_1, x_2, \dots, x_n) - b_i = 0$

1. Compute G_n ; Solve the system $g_i = 0$.
2. Compute G_{n-1} . For each solution, solve the system $g_i = 0$.
3. Repeat for all $G_i, i = n, \dots, 1$.

3.5 Existence of a Solution

The existence of a solution is more relevant to our interests than the solution itself. Since a system of equations has no solution if and only if the corresponding Groebner Basis contains 1, it suffices to check this condition. However, it will prove useful to introduce a *reduced Groebner Basis*.

Definition 3.5.1. *A reduced Groebner Basis for an ideal I is a Groebner Basis G of I which satisfies*

1. *For all $p \in G$, the coefficient of the leading term of p is 1*
2. *For all $p \in G$, no monomial of p lies in $\langle \text{LT}(G) - \{p\} \rangle$*

For any Groebner Basis G , a reduced Groebner Basis \bar{G} can be computed in the following manner.

1. For all $p \in G$, if $\text{LT}(p) \in \langle \text{LT}(G) - \{p\} \rangle$, then remove p from G .
2. For all $p \in G$, if some non-leading term in p is in $\langle \text{LT}(G) - \{p\} \rangle$, then replace p with the remainder of p divided by the set $G - \{p\}$.

Step 2 above ensures the resulting \bar{G} satisfies 3.5.1-(2). Hence, for every Groebner Basis, a reduced Groebner Basis can be found. Fast Algorithms (e.g. Faugère F4 Algorithm) exist for the computation of a Reduced Groebner Basis.

If a Groebner Basis contains 1, then $1 \in \langle \text{LT}(G) - \{1\} \rangle$. Hence, the reduced Groebner Basis is $\{1\}$. The following lemma will provide us with a mechanism for determining the existence of a solution to systems of multivariate polynomial equations.

Lemma 3.5.2. *A system of polynomial equations $f_i(x_1, x_2, \dots, x_n) - b_i = 0$ has a solution if and only if the reduced Groebner Basis for the ideal $\langle f_1, f_2, \dots, f_s \rangle$ does not reduce to $\{1\}$.*

Chapter 4

Involutorial Automorphisms With a Maximal (-1)-Eigenspace

In 1988 Helminck [5] classified the local symmetric spaces over algebraically closed fields. He found 24 cases which correspond to unique isomorphisms of involutions of Lie Algebras with a maximal (-1)-eigenspace. Recall the local symmetric space for an arbitrary Lie Algebra \mathfrak{g} and involution $\bar{\theta} \in \text{Aut}(\mathfrak{g})$ is defined in Equation 1.1 as

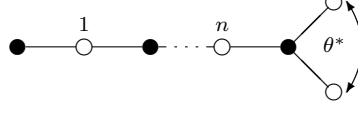
$$\mathfrak{p} = \{X \in \mathfrak{g} \mid \bar{\theta}(X) = -X\}$$

The primary goal of this chapter is to study the structure of \mathfrak{p} by understanding how the involution on the roots of \mathfrak{g} relates to $\bar{\theta}$. Specifically, for an involution $\theta \in \text{Aut}(\Phi)$, we want to determine how to modify $\bar{\theta}$ so that $\bar{\theta}|_{\mathfrak{t}} = \theta$. When the context is clear, we write θ for $\bar{\theta}$. Studying this problem will lead to an algorithm for considering any involutorial automorphism. We'll refer to this algorithm as the *lifting algorithm* throughout our discussion.

4.1 Classification of the Local Symmetric Spaces

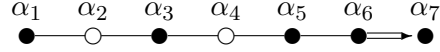
Helminck borrows the Dynkin diagram from finite dimensional semisimple theory. He amends the diagram with additional information from which we can represent the action of the involution on the roots of \mathfrak{g} . We call this diagram a *Helminck Diagram*.

Figure 4.1: Helminck Diagram for DIIIb



Colored black are those roots which are fixed by θ . Colored white are the roots which project down to roots in \mathfrak{p} . The arrows represent the action of θ^* , the diagram automorphism. In the diagram we omit showing the action on the roots denoted by black dots, as they are fixed. The local symmetric spaces are classified by the Helminck Diagrams. The 24 cases are given in Table B.1.

The action of θ can be recovered in the following way. We first look at the “embedded root systems” found in the Helminck Diagrams. These are the root systems associated with the Dynkin Diagrams formed by the black dots. For instance, in the diagram



we have the root systems B_3 formed by the string of black dots encompassing the roots α_5 , α_6 , and α_7 . We also have the root system A_1 twice (formed by the roots α_1 and α_3). Thus, the embedded root system is $A_1 \times A_1 \times B_3$. $w_0(\theta)$ is the longest element of the Weyl group of the embedded root system. The longest element of B_3 with respect to the roots α_5 , α_6 , and α_7 is $s_{\alpha_5}s_{\alpha_6}s_{\alpha_5}s_{\alpha_7}s_{\alpha_6}s_{\alpha_5}s_{\alpha_7}s_{\alpha_6}s_{\alpha_7}$. The longest element of A_1 with respect to the root α_1 is s_{α_1} . The longest element of A_1 with respect to the root α_3 is s_{α_3} . Hence, $w_0(\theta) = s_{\alpha_1}s_{\alpha_3}s_{\alpha_5}s_{\alpha_6}s_{\alpha_5}s_{\alpha_7}s_{\alpha_6}s_{\alpha_5}s_{\alpha_7}s_{\alpha_6}s_{\alpha_7}$

In [5], Helminck showed $\theta^* = -\text{id} \circ \theta \circ w_0(\theta)$. We have the following theorem.

Theorem 4.1.1.

$$\theta^* = \begin{cases} \text{id}; \\ \langle \text{Dynkin Diagram automorphism of order } 2 \rangle \end{cases}$$

Since θ is an involution then $\theta^{-1} = \theta$. We have the same remark for θ^* . Then we can recover the induced involution on the root system from the Helminck Diagram by noting

$$\theta = -\text{id} \circ \theta^* \circ w_0(\theta) \quad (4.1)$$

For the example diagram above we have θ^* is the identity mapping. Hence

$$\theta = -w_0(\theta) = -s_{\alpha_1} s_{\alpha_3} s_{\alpha_5} s_{\alpha_6} s_{\alpha_5} s_{\alpha_7} s_{\alpha_6} s_{\alpha_5} s_{\alpha_7} s_{\alpha_6} s_{\alpha_7}$$

The action of θ on the roots is given in the following matrix

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

Note that we have the identity mapping when we restrict θ to the roots denoted by black dots.

4.2 The Structure Constants

For X_α the root vector in \mathfrak{g} associated with the root α , we have the following

$$\bar{\theta}(X_\alpha) = c_{\alpha, \bar{\theta}} X_{\theta(\alpha)} \quad (4.2)$$

We call the constants $c_{\alpha, \bar{\theta}}$ the *structure constants* of $\bar{\theta}$ with respect to \mathfrak{g} . The prerequisite step to running the lifting algorithm is to determine these constants.

Definition 4.2.1. Let θ_Δ be the unique automorphism in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$ such that

$$\theta_\Delta(X_\alpha) = X_{\theta(\alpha)} \quad \text{for all the basis roots } \alpha \in \Delta$$

The existence and uniqueness of such an automorphism is given in [5]. Note that the definition implies that for all $\alpha \in \Delta$ we have $c_{\alpha, \theta_\Delta} = 1$.

We now establish two identities which will prove useful

Lemma 4.2.2. For $\alpha, \beta \in \Phi$, $c_{\alpha, \theta}$ as in equation 4.2, and $N_{\alpha, \beta}$ as in definition 2.7.1 we have

1.

$$c_{-\alpha, \theta} = (c_{\alpha, \theta})^{-1} \quad (4.3)$$

2.

$$c_{\alpha+\beta, \theta} = \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \theta} c_{\beta, \theta} \quad (4.4)$$

if $\alpha + \beta$ is a root of \mathfrak{g} .

Proof. To show the first statement, first recall θ is a Lie Algebra homomorphism. Then $\theta([X_\alpha, X_{-\alpha}]) = [\theta(X_\alpha), \theta(X_{-\alpha})] = c_{\alpha, \theta} c_{\alpha, -\theta} [X_{\theta(\alpha)}, X_{\theta(-\alpha)}] = c_{\alpha, \theta} c_{\alpha, -\theta} H_{\theta(\alpha)}$.

We also have $\theta([X_\alpha, X_{-\alpha}]) = \theta(H_\alpha) = H_{\theta(\alpha)}$. Then $c_{\alpha, \theta} c_{\alpha, -\theta} = 1$ and so $c_{-\alpha, \theta} = (c_{\alpha, \theta})^{-1}$.

To prove the second statement, we first recall that from Definition 4.2 we have $\theta(X_{\alpha+\beta}) = c_{\alpha+\beta, \theta} X_{\theta(\alpha+\beta)}$. Recall that we have $[X_\alpha, X_\beta] = N_{\alpha, \beta} X_{\alpha+\beta}$ if $\alpha + \beta$ is a root. Then $X_{\alpha+\beta} = \frac{1}{N_{\alpha, \beta}} [X_\alpha, X_\beta]$. Since θ is a Lie Algebra homomorphism, we write

$$\begin{aligned} \theta(X_{\alpha+\beta}) &= \theta\left(\frac{1}{N_{\alpha, \beta}} [X_\alpha, X_\beta]\right) \\ &= \frac{1}{N_{\alpha, \beta}} [\theta(X_\alpha), \theta(X_\beta)] \\ &= \frac{1}{N_{\alpha, \beta}} [c_{\alpha, \theta} X_{\theta(\alpha)}, c_{\alpha, \beta} X_{\theta(\beta)}] \\ &= \frac{1}{N_{\alpha, \beta}} c_{\alpha, \theta} c_{\alpha, \beta} [X_{\theta(\alpha)}, X_{\theta(\beta)}] \\ &= \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \theta} c_{\alpha, \beta} X_{\theta(\alpha+\beta)} \end{aligned}$$

$$\text{Then } c_{\alpha+\beta, \theta} = \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \theta} c_{\beta, \theta}.$$

□

The two preceding lemmas lead us to a mechanism by which we can compute these structure constants. If we know the structure constants $c_{\alpha, \theta}$ for the basis roots, then we

can find the structure constants for all roots. The manner in which this is done is similar to how one constructs the root system from the basis roots. The scheme to compute the roots is given in [6]. A modified version of this algorithm is as follows

Algorithm 4.2.3 (Computation of the Structure Constants).

Input Δ , the basis roots, and $c_{\alpha,\theta}$ for all $\alpha \in \Delta$.

Output $c_{\alpha,\theta}$ for all $\alpha \in \Phi$.

1. $n := 1$; $\Phi := \Delta$; $s := 1$.

2. **while** $|\Phi| > s$

$s := |\Phi|$

for every element $\alpha = \sum_{\beta_i \in \Delta} k_i \beta_i \in \Phi$ and every element of $\delta \in \Delta$ **do**

$h := \sum k_i$

if $h = n$ **then**

Compute r , the highest integer so that $\alpha - r\delta$ is in Φ

$q := r - \sum k_i M_{i,j}$, where $M_{i,j}$ is the (i,j) entry in the Cartan Matrix

if $q > 0$ **then**

$\Phi := \Phi \cup (\alpha + \delta)$

$c_{\alpha+\delta,\theta} := \frac{N_{\theta(\delta),\theta(\alpha)}}{N_{\delta,\alpha}} c_{\alpha,\theta} c_{\delta,\theta}$

$c_{-(\alpha+\delta),\theta} := (c_{\alpha+\delta,\theta})^{-1}$

3. **return** $c_{\alpha,\theta}$ for all $\alpha \in \Phi$

Theorem 4.2.4. Algorithm 4.2.3 computes all $c_{\alpha,\theta}$ for all $\alpha \in \Phi$

Proof. Proof that Algorithm 4.2.3 generates all the roots from the basis roots can be found in [6]. Every time a new root is found, it is found by adding two previously known roots. Hence, by computing the structure constant at the same time, for every new root found, we find its corresponding structure constant. Because the algorithm computes all the roots, it computes all the structure constants. \square

As a remark, all we need to supply to the algorithm are the structure constants for the basis roots. From the remark made in Definition 4.2.1, we have $c_{\alpha,\theta_\Delta} = 1$ for all $\alpha \in \Delta$. Hence, we can compute $c_{\alpha,\theta_\Delta} = 1$ for all $\alpha \in \Delta$.

4.3 The Lifting Condition

The goal of this section is to describe the conditions for which $\theta \in \text{Aut}(\Phi)$ can be lifted in such a way that is suitable for computation. We begin with the following identity.

Lemma 4.3.1. *Let $H \in \mathfrak{t}$ and θ_Δ as defined in definition 4.2.1. Then*

$$\theta_\Delta \text{ad } H = \text{ad}(\theta_\Delta H) \theta_\Delta$$

and

$$\theta_\Delta \text{ad}(\theta_\Delta H) = \text{ad } H \theta_\Delta$$

Proof. Let $X \in \mathfrak{g}$.

$$\begin{aligned} \theta_\Delta \text{ad } H X &= \theta_\Delta [H, X] \\ &= [\theta_\Delta H, \theta_\Delta X] \\ &= \text{ad}(\theta_\Delta H) \theta_\Delta X \end{aligned}$$

$$\theta_\Delta \text{ad } H = \text{ad}(\theta_\Delta H) \theta_\Delta$$

Take $H = \theta_\Delta H$ to write

$$\theta_\Delta \text{ad}(\theta_\Delta H) = \text{ad}(\theta_\Delta^2 H) \theta_\Delta$$

Since θ_Δ is an involution on \mathfrak{t} , we get

$$\theta_\Delta \text{ad}(\theta_\Delta H) = \text{ad } H \theta_\Delta$$

□

In [5], Helminck characterized the involutions of Φ which can be lifted. The following proposition is obtained.

Proposition 4.3.2. *Let Δ be a basis of Φ and $\theta \in \text{Aut}(\Phi)$ be an involution. Then the following are equivalent.*

1. θ can be lifted.
2. There exists a vector $H \in \mathfrak{t}$ such that $\theta = \theta_\Delta \text{ad}(H)$ is an involution.
3. $\alpha(H) \theta(\alpha)(H) c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} = 1$ for all $\alpha \in \Delta$.

Proof. That (1) and (2) are equivalent follows from the definition.

To show the equivalence of (2) and (3), let $X_\alpha \in \mathfrak{g}$, where $\alpha \in \Phi$. We assume $\theta^2 = (\theta_\Delta \text{ad}(H))^2$. Then

$$\theta_\Delta \text{ad}(H) \theta_\Delta \text{ad}(H) X_\alpha = \theta^2 X_\alpha$$

As θ is an involution, we have

$$\theta_\Delta \text{ad}(H) \theta_\Delta \text{ad}(H) X_\alpha = X_\alpha$$

Use lemma 4.3.1 to write

$$\begin{aligned} \text{ad}(\theta_\Delta H) \theta_\Delta^2 \text{ad}(H) X_\alpha &= X_\alpha \\ \text{ad}(\theta_\Delta H) \text{ad}(\theta_\Delta^2 H) \theta_\Delta^2 X_\alpha &= X_\alpha \\ \text{ad}(\theta_\Delta H) \text{ad} H \theta_\Delta^2 X_\alpha &= X_\alpha \end{aligned}$$

Using definition 4.2, note

$$\theta_\Delta^2 X_\alpha = \theta_\Delta \theta_\Delta X_\alpha = c_{\alpha, \theta_\Delta} \theta_\Delta X_{\theta(\alpha)} = c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} X_{\theta^2(\alpha)} = c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} X_\alpha$$

Then

$$\begin{aligned} \text{ad}(\theta_\Delta H) \text{ad} H \theta_\Delta^2 X_\alpha &= c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} \text{ad}(\theta_\Delta H) \text{ad}(\theta_\Delta^2 H) X_\alpha \\ &= c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} \alpha(H) \text{ad}(\theta_\Delta H) X_\alpha \\ &= c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} \alpha(H) \alpha(\theta_\Delta H) X_\alpha \end{aligned}$$

Hence,

$$\begin{aligned}
X_\alpha &= c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} \alpha(H) \alpha(\theta_\Delta H) X_\alpha \\
1 &= c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} \alpha(H) \alpha(\theta_\Delta H)
\end{aligned}$$

□

4.4 The Correction Vector

Proposition 4.3.2 leads us to a mechanism with which we can determine if θ can be lifted. θ can be lifted if we can find a vector $H \in \mathfrak{t}$ satisfying statement 4.3.2-(3). We call this vector the *correction vector*. For any given $\theta \in \text{Aut}(\Phi)$, there are three possible outcomes.

1. θ can be lifted. That is, we can compute a vector H satisfying statement 4.3.2-(3).
2. θ fails to lift. This is the case when such a vector H does not exist.
3. θ is already an involution on the Lie Algebra. A discussion on this case proceeds.

Under certain conditions it may not be necessary to compute a correction vector. This is the case when the structure constants are “well-behaved,” as stated below.

Lemma 4.4.1. $\theta \in \text{Aut}(\Phi)$ already lends itself to an involutorial automorphism on \mathfrak{g} if $c_{\theta(\alpha), \theta_\Delta} = 1$ for all $\alpha \in \Delta$.

Proof. Let θ_Δ be defined as in 4.2.1 and let α be a root in Δ .

$$\theta_\Delta^2(X_\alpha) = \theta_\Delta(\theta_\Delta(X_\alpha)) = c_{\alpha, \theta_\Delta} \theta_\Delta(X_{\theta(\alpha)}) = c_{\alpha, \theta_\Delta} c_{\theta(\alpha), \theta_\Delta} X_{\theta^2(\alpha)} = c_{\alpha, \theta_\Delta} c_{\theta(\alpha), \theta_\Delta} X_\alpha$$

$c_{\alpha, \theta_\Delta} = 1$ by definition of θ_Δ , and $c_{\theta(\alpha), \theta_\Delta} = 1$ by assumption. Then we have $\theta_\Delta^2 X_\alpha = X_\alpha$ for all $\alpha \in \Delta$. Hence, $\theta_\Delta^2 = 1$ and θ_Δ is an involution. Note that $\theta_\Delta|_\Phi = \theta$, which indicates θ can be lifted to an involutorial automorphism on $\text{Aut}(\mathfrak{g}, \mathfrak{t})$. □

For cases where Lemma 4.4.1 applies there is no further work to be done. We already have an involution on \mathfrak{g} . Since the process of computing the correction vector requires computing the structure constants, a quick check to see that Lemma 4.4.1 applies may save time. For cases where the lemma does not apply, we will have to proceed with the rest of the procedure.

4.5 Computation of the Correction Vector(s)

The goal of this section is to define an algorithm which will generate the correction vector. We assume that at this point, we've already verified that Lemma 4.4.1 does not apply. The first step is to interpret 4.3.2 in such a way that we have a system of equations - the solution of which will provide us with the correction vector.

First we write

$$H = x_1 H_{\alpha_1} + x_2 H_{\alpha_2} + \dots + x_n H_{\alpha_n} \quad (4.5)$$

where \mathfrak{t} is of dimension n , $H \in \mathfrak{t}$, and $H_{\alpha_i} \in \mathfrak{t}$ satisfies $[X_{\alpha_i}, X_{-\alpha_i}] = H_{\alpha_i}$ (that is, H_{α_i} is a basis vector for \mathfrak{t}). The constants x_i are the coordinates of H in \mathfrak{t} with respect to the basis $\{H_{\alpha_i}\}$.

Let α be a root in Φ . Then

$$\alpha(H) = \alpha(x_1 H_{\alpha_1} + x_2 H_{\alpha_2} + \dots + x_n H_{\alpha_n})$$

Since the roots of \mathfrak{g} are linear functionals, we can write

$$\alpha(H) = x_1 \alpha(H_{\alpha_1}) + x_2 \alpha(H_{\alpha_2}) + \dots + x_n \alpha(H_{\alpha_n})$$

where $\alpha(H_{\alpha_i})$ are elements of our algebraically closed ground field F .

Let $\theta(\alpha) = \sum_i k_i \alpha_i$ for $\alpha_i \in \Delta$. Then note that we also have

$$\begin{aligned} \alpha(\theta(H)) &= \alpha(\theta(x_1 H_{\alpha_1} + x_2 H_{\alpha_2} + \dots + x_n H_{\alpha_n})) \\ &= \alpha(x_1 H_{\theta(\alpha_1)} + x_2 H_{\theta(\alpha_2)} + \dots + x_n H_{\theta(\alpha_n)}) \\ &= x_1 \alpha(H_{\theta(\alpha_1)}) + x_2 \alpha(H_{\theta(\alpha_2)}) + \dots + x_n \alpha(H_{\theta(\alpha_n)}) \end{aligned}$$

where

$$\begin{aligned}
\alpha(H_{\theta(\alpha_i)}) &= \alpha(H_{k_1\alpha_1+\dots+k_n\alpha_n}) \\
&= \alpha(k_1H_{\alpha_1} + \dots + k_nH_{\alpha_n}) \\
&= k_1\alpha(H_{\alpha_1}) + \dots + k_n\alpha(H_{\alpha_n})
\end{aligned}$$

We can then define the following polynomials

Definition 4.5.1.

$$F_{p,i} := x_1\alpha_i(H_{\theta^p(\alpha_1)}) + x_2\alpha_i(H_{\theta^p(\alpha_2)}) + \dots + x_n\alpha_i(H_{\theta^p(\alpha_n)})$$

where $p \in \mathbb{Z}_{\geq 0}$, $i \in \{0, 1, \dots, n\}$, and θ^0 denotes the identity mapping

Of interest to us in the involution case are the polynomials $F_{0,i}$ and $F_{1,i}$. These are

$$\begin{aligned}
F_{0,i} &= x_1\alpha_i(H_{\alpha_1}) + x_2\alpha_i(H_{\alpha_2}) + \dots + x_n\alpha_i(H_{\alpha_n}) \\
F_{1,i} &= x_1\alpha_i(H_{\theta(\alpha_1)}) + x_2\alpha_i(H_{\theta(\alpha_2)}) + \dots + x_n\alpha_i(H_{\theta(\alpha_n)})
\end{aligned} \tag{4.6}$$

If we have a Chevalley basis for \mathfrak{g} , more can be said. Recall that a Chevalley basis gives us

$$[H_{\alpha_k}, X_\alpha] = \alpha(H_{\alpha_k})X_\alpha$$

and

$$\alpha(H_{\alpha_k}) = \langle \alpha, \alpha_k \rangle$$

Then

$$\alpha(H) = \sum_{k=1}^n x_k \langle \alpha, \alpha_k \rangle$$

We re-define our “F-polynomials” as follows.

Definition 4.5.2.

$$F_{p,i} := \sum_{k=1}^n x_k \langle \theta^p(\alpha_i), \alpha_k \rangle$$

where $p \in \mathbb{Z}_{\geq 0}$, $i \in \{0, 1, \dots, n\}$, and θ^0 denotes the identity mapping

Then of immediate interest to us are

$$\begin{aligned} F_{0,i} &= \sum_{k=1}^n x_k \langle \alpha_i, \alpha_k \rangle \\ F_{1,i} &= \sum_{k=1}^n x_k \langle \theta(\alpha_i), \alpha_k \rangle \end{aligned} \tag{4.7}$$

For every α_i , $i = 1 \dots n$, we replace $\alpha_i(H)$ with $F_{0,i}$ and $\alpha_i(\theta(H))$ with $F_{1,i}$ in Proposition 4.3.2-(3). This yields a system of n multivariate polynomial equations in the variables x_i . Finding a solution implies θ can be lifted. The solution gives the coordinates of the correction vector with respect to the basis of \mathfrak{t} . The full algorithm is described as follows.

Algorithm 4.5.3 (Check Lifting Algorithm, Order 2).

Input θ , an involutorial automorphism in $\text{Aut}(\Phi)$; \mathfrak{g} , a Lie Algebra with roots Φ , having basis roots Δ

Output **TRUE** if θ can be lifted, **N/A** if the correction vector is not necessary, or **FALSE** if the correction vector is necessary, but cannot be found.

1. Call Algorithm 4.2.3 or Algorithm 7.8.1 to compute the structure constants $c_{\alpha,\theta}$ for every $\alpha \in \Phi$
2. **if** $c_{\theta(\alpha),\theta_\Delta} = 1$ for all $\alpha \in \Delta$ **then return** **N/A**
3. **for** $i = 1$ to n

$$f_i := c_{\alpha,\theta_\Delta} c_{\theta(\alpha),\theta_\Delta} F_{0,i} F_{1,i} - 1$$
4. $I := \langle f_1, \dots, f_n \rangle$
5. Compute G , the reduced Groebner Basis for I .
6. **if** $G \neq \{1\}$ **then return** **TRUE**
- else return** **FALSE**

Theorem 4.5.4. Algorithm 4.5.3 works correctly

Proof. Recall that $F_{0,i}$ is precisely $\alpha_i(H)$, and $F_{1,i}$ is $\alpha_i(\theta(H))$.

Computation of a reduced Groebner Basis for the polynomials reduces to $\{1\}$ if and only if the system

$$c_{\alpha,\theta} c_{\theta(\alpha),\theta} F_{0,i} F_{1,i} - 1 = 0 \quad i = 1 \dots n$$

is unsolvable. Hence, if $G \neq \{1\}$, then there is a vector H satisfying proposition 4.3.2-(3). \square

For the purposes of doing computations in local symmetric spaces, we may wish to retrieve vectors H satisfying Proposition 4.3.2. We modify the preceding algorithm to return these vector(s).

Algorithm 4.5.5 (Lifting Algorithm, Order 2).

Input θ_Δ as in Definition 4.2.1; θ , an involutorial automorphism in $\text{Aut}(\Phi)$; \mathfrak{g} , a Lie Algebra with roots Φ , having basis roots Δ

Output **N/A** if θ_Δ is already an involution on \mathfrak{g} ; All possible sets of coordinates of a correction vector $H \in \mathfrak{t}$ so that $\theta_\Delta \text{ad}(H)$ is an involution on \mathfrak{g} ; or **FAIL** if such a vector H is needed, but cannot be found.

1. Call Algorithm 4.2.3 or Algorithm 7.8.1 to compute the structure constants $c_{\alpha,\theta}$ for every $\alpha \in \Phi$
2. **if** $c_{\theta(\alpha),\theta_\Delta} = 1$ for all $\alpha \in \Delta$ **then return** **N/A**
3. **for** $i = 1$ to n

$$f_i := c_{\alpha,\theta_\Delta} c_{\theta(\alpha),\theta_\Delta} F_{0,i} F_{1,i} - 1$$
4. $I := \langle f_1, \dots, f_n \rangle$
5. Compute G , the reduced Groebner Basis for I .
6. **if** $G = \{1\}$ **then return** **FAIL**
7. **return** output of Algorithm 3.4.3 with $g_1, \dots, g_s \in G$ as the arguments.

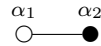
Theorem 4.5.6. Algorithm 4.5.5 works correctly

Proof. Justification for the first six steps has been discussed in the previous algorithm. That the last step completes the process follows immediately from Algorithm 3.4.3. \square

To illustrate the above procedure, let us consider a simple example.

Example 4.5.7. Lifting an Involution on the roots of $\mathfrak{sl}_3(\mathbb{C})$ to an Involution on the Algebra

Consider the Helminck diagram describing an involution on the roots of $\mathfrak{sl}_3(\mathbb{C})$



We have no diagram automorphism, hence $\theta^* = \text{id}$. The embedded root system is A_1 with respect to α_2 . Hence, $w_0(\theta) = s_{\alpha_2}$. We have

$$\theta = -s_{\alpha_2}$$

and hence

$$\theta(\alpha_1) = -\alpha_1 - \alpha_2$$

$$\theta(\alpha_2) = \alpha_2$$

An ordered basis for the Lie algebra $\mathfrak{sl}_3(\mathbb{C})$ is

$$\mathfrak{g} = \text{span}\{E_{1,1} - E_{2,2}, E_{2,2} - E_{3,3}, E_{1,1} - E_{2,2}, E_{1,2}, E_{2,3}, E_{1,3}, E_{2,1}, E_{3,2}, E_{3,1}\} \quad (4.8)$$

We compute a Chevalley basis for this algebra and obtain the following assignments for the root vectors

Root	Root Vector X_α	Torus Vector H_α
α_1	$E_{1,2}$	$E_{1,1} - E_{2,2}$
α_2	$E_{2,3}$	$E_{2,2} - E_{3,3}$
$\alpha_1 + \alpha_2$	$E_{1,3}$	$E_{1,1} - E_{3,3}$
$-\alpha_1$	$E_{2,1}$	$E_{2,2} - E_{1,1}$
$-\alpha_2$	$E_{3,2}$	$E_{3,3} - E_{2,2}$
$-\alpha_1 - \alpha_2$	$E_{3,1}$	$E_{3,3} - E_{1,1}$

We can then compute the Chevalley constants as given:

Root α	Root β	Constant $N_{\alpha,\beta}$
α_1	α_2	1
α_1	$-\alpha_1 - \alpha_2$	-1
α_2	α_1	-1
α_2	$-\alpha_1 - \alpha_2$	1
$\alpha_1 + \alpha_2$	$-\alpha_1$	-1
$\alpha_1 + \alpha_2$	$-\alpha_2$	1
$-\alpha_1$	$\alpha_1 + \alpha_2$	1
$-\alpha_1$	$-\alpha_2$	-1
$-\alpha_2$	$\alpha_1 + \alpha_2$	-1
$-\alpha_2$	$-\alpha_1$	1
$-\alpha_1 - \alpha_2$	α_1	1
$-\alpha_1 - \alpha_2$	α_2	-1

We have that θ_Δ is a Lie algebra automorphism. Next we want to determine if it is an involution. To compute the structure constants, we run through Algorithm 4.2.3 and obtain the following.

Root α	c_{α,θ_Δ}
α_1	1
α_2	1
$\alpha_1 + \alpha_2$	-1
$-\alpha_1$	1
$-\alpha_2$	1
$-\alpha_1 - \alpha_2$	-1

If θ_Δ is an involution, then we need $c_{\theta(\alpha),\theta_\Delta} = 1$ for $\alpha = \alpha_1$ and α_2 . However, $\theta(\alpha_1) = -\alpha_1 - \alpha_2$, and hence, $c_{\theta(\alpha_1),\theta_\Delta} = c_{-\alpha_1 - \alpha_2,\theta_\Delta} = -1$. We have that θ_Δ is not an involution. To illustrate, below is the matrix for θ_Δ relative to the ordered basis given in 4.8.

$$[\theta] = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (4.9)$$

$\theta_\Delta^2 = \text{diag}(1, 1, 1, -1, -1, 1, -1, -1)$. We will need to modify θ_Δ with the right correction vector satisfying Proposition 4.3.2.

As per Equation 4.5 we have $H = x_1 H_{\alpha_1} + x_2 H_{\alpha_2}$. x_1 and x_2 are the coordinates of H relative to the basis for the torus given by $\{H_{\alpha_1}, H_{\alpha_2}\}$.

From Definition 4.5.2 and Equation 4.7 We have

$$F_{0,1} = x_1 \alpha_1(H_{\alpha_1}) + x_2 \alpha_1(H_{\alpha_2})$$

$$F_{1,1} = x_1 \alpha_1(H_{\theta(\alpha_1)}) + x_2 \alpha_1(H_{\theta(\alpha_2)})$$

$$F_{0,2} = x_1 \alpha_2(H_{\alpha_1}) + x_2 \alpha_2(H_{\alpha_2})$$

$$F_{1,2} = x_1 \alpha_2(H_{\theta(\alpha_1)}) + x_2 \alpha_2(H_{\theta(\alpha_2)})$$

$F_{0,1}$ and $F_{0,2}$ simply become

$$F_{0,1} = 2x_1 - x_2$$

$$F_{0,2} = -x_1 + 2x_2$$

For $F_{1,1}$ we have

$$\begin{aligned}
F_{1,1} &= x_1\alpha_1(H_{\theta(\alpha_1)}) + x_2\alpha_1(H_{\theta(\alpha_2)}) \\
&= x_1\alpha_1(H_{-\alpha_1-\alpha_2}) + x_2\alpha_1(H_{\alpha_2}) \\
&= x_1\alpha_1(-H_{\alpha_1} - H_{\alpha_2}) + x_2\alpha_1(H_{\alpha_2}) \\
&= -x_1 - x_2
\end{aligned}$$

For $F_{1,2}$ we have

$$\begin{aligned}
F_{1,2} &= x_1\alpha_2(H_{\theta(\alpha_1)}) + x_2\alpha_2(H_{\theta(\alpha_2)}) \\
&= x_1\alpha_2(H_{-\alpha_1-\alpha_2}) + x_2\alpha_2(H_{\alpha_2}) \\
&= x_1\alpha_2(-H_{\alpha_1} - H_{\alpha_2}) + x_2\alpha_2(H_{\alpha_2}) \\
&= -x_1 + 2x_2
\end{aligned}$$

Hence we have the system

$$\left\{ \begin{array}{l} -(-x_1 - x_2)(2x_1 - x_2) - 1 = 0 \\ (2x_2 - x_1)^2 - 1 = 0 \end{array} \right\} \quad (4.10)$$

which has four solution sets

$$\left\{ x_1 = -\frac{\sqrt{5}}{3}, x_2 = \frac{1}{6}(-3 - \sqrt{5}) \right\} \quad (4.11)$$

$$\left\{ x_1 = -\frac{\sqrt{5}}{3}, x_2 = \frac{1}{6}(3 - \sqrt{5}) \right\} \quad (4.12)$$

$$\left\{ x_1 = \frac{\sqrt{5}}{3}, x_2 = \frac{1}{6}(-3 + \sqrt{5}) \right\} \quad (4.13)$$

$$\left\{ x_1 = \frac{\sqrt{5}}{3}, x_2 = \frac{1}{6} (3 + \sqrt{5}) \right\} \quad (4.14)$$

If we consider the first solution set, then the correction vector is

$$H = \begin{pmatrix} -\frac{\sqrt{5}}{3} & 0 & 0 \\ 0 & \frac{\sqrt{5}}{3} + \frac{1}{6} (-3 - \sqrt{5}) & 0 \\ 0 & 0 & \frac{1}{6} (3 + \sqrt{5}) \end{pmatrix} \quad (4.15)$$

and the corrected involution is

$$[\theta_{\Delta} \text{ad}(H)] = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} (-1 - \sqrt{5}) \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} (-1 + \sqrt{5}) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} (1 + \sqrt{5}) & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} (1 - \sqrt{5}) & 0 & 0 & 0 & 0 \end{pmatrix} \quad (4.16)$$

and $[\theta_{\Delta} \text{ad}(H)]^2 = I$.

4.6 Correction Vectors for the Involutional Automorphisms with (-1)-Eigenspace

We now turn our attention to Local Symmetric Spaces over algebraically closed fields. Recall that we have 24 isomorphism cases in the classification of involutorial automorphisms with a maximal (-1)-eigenspace. Our aim in this section is to establish

1. The action of $\theta \in \text{Aut}(\Phi)$ described by the Helminck Diagrams.
2. The structure constants.
3. If $\theta_\Delta \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ is an involution, or if it needs to be modified with a correction vector.
4. In the latter case above, what the possible correction vectors are.

As a remark, we should expect that under no circumstances in any of the 24 cases will any involutions on the root systems need a correction vector which cannot be found. That is, for the involutions θ described, if the map θ_Δ is not an involution, it can always be modified so that it is.

Diagrams for the 24 cases, and the conclusions, are given in table B.1.

To save some time we make the following claim.

Lemma 4.6.1. *For a given Helminck Diagram, if there are no fixed roots (black dots), then $c_{\theta(\alpha), \theta_\Delta} = 1$ for all $\alpha \in \Delta$.*

Proof. If there are no black dots, then $w_0(\theta) = \text{id}$. Hence, $\theta = -\text{id}$, or $\theta = -\theta^*$ in the case of a diagram automorphism. In both instances θ maps α to another basis root. Thus, $\theta(\alpha) \in \Delta$. Then by the definition of θ_Δ (4.2.1), $c_{\theta(\alpha), \theta_\Delta} = 1$. \square

Immediately we have that θ_Δ is an involution for the cases of AI, AIIIb, CI, DI, EI, EII, EV, EVIII, FI, and G.

For the remaining 14 cases we first compute all the structure constants. The surprising result is that there are only two instances where lifting is necessary: DIIIb and EIII. For all 22 remaining cases, we can take θ_Δ to be an involution on the root system satisfying the lifting condition. We give correction vectors immediately following Table B.1. The results are summarized as follows.

Theorem 4.6.2. *Let θ_Δ be the involution defined by definition 4.2.1, with θ the involution on the root system defined by the Helminck Diagrams in table B.1. Let $H \in \mathfrak{t}$. Then*

1. *The involutions θ described by DIIIb and EIII lift to involutions on the Lie Algebra. A vector H satisfying Proposition 4.3.2 (1) can be found.*
2. *The involutions described by the remaining entries in Table B.1 are already involutions on the Lie Algebra. i.e., the structure constants $c_{\theta(\alpha), \theta} = 1$ for all $\alpha \in \Delta$.*

At first glance it is surprising that in so many cases θ_Δ was an involution. Indeed, if it were not for $DIII_b$, for every classical case θ_Δ would be an involution. There is much more to be said about the conditions which imply θ_Δ is an involution without lifting. This is an issue which we will address in Chapter 7. In this chapter, we shall describe how one can tell, simply by looking at the configuration of black and white dots in a Helminck diagram, whether or not θ_Δ will turn out to be an involution.

Beforehand, however, we would like to establish an algorithm to check for admissibility - which will be a slight extension of that which we just established.

Chapter 5

Admissibility

This short chapter is dedicated to a slight extension of the lifting algorithm we previously established. In the discussion we will introduce several key concepts that will be important to discussion in the subsequent chapters. In particular, we will define restricted roots and projections, how they relate to θ -normality, and how θ -normality relates to *admissibility*. We may be interested in knowing if an involution on the root system is admissible, as it would imply several nice properties.

5.1 Root Projections

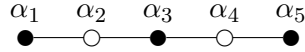
Recall that \mathfrak{p} is defined as the set of $X \in \mathfrak{g}$ such that $\bar{\theta}(X) = -X$, where $\bar{\theta}$ is an involutorial automorphism on \mathfrak{g} . Let \mathfrak{a} be a maximal torus in \mathfrak{p} . We expect the roots of \mathfrak{p} to satisfy $\theta(\alpha) = -\alpha$ for all $\alpha \in \Phi(\mathfrak{a})$. We then define the projection of the roots of \mathfrak{g} onto the roots of \mathfrak{p} via the following

$$\pi(\alpha) = \frac{1}{2}(\alpha - \theta(\alpha)) \tag{5.1}$$

Since the roots denoted by black dots satisfy $\theta(\alpha) = \alpha$, it is easy to see that they project to zero. The roots denoted by white dots then form the *projected root system*. Let us denote by $\bar{\Phi}$ all the roots in the image of π . In [5] we see that $\bar{\Phi}$ is indeed a root system. The root system has basis $\bar{\Delta}$, satisfying

$$\bar{\Delta} = \{\lambda = \pi(\alpha) \mid \alpha \in \Delta\} \quad (5.2)$$

To avoid confusion, we will use α for roots in Φ , and λ for roots in $\bar{\Phi}$. To illustrate how $\bar{\Phi}$ indeed forms a root system, take as an example the Helminck diagram \mathbb{H} below.



We note that \mathbb{H} describes a root system of type A_5 . Hence, we have $\alpha_i = e_i - e_{i+1}$. For α_i with i odd, we have a fixed root, and hence $\pi(\alpha_i) = 0$. For α_i with i even we have

$$\lambda_1 = \pi(\alpha_2) = \frac{1}{2}(e_1 + e_2 - e_3 - e_4)$$

$$\lambda_2 = \pi(\alpha_4) = \frac{1}{2}(e_3 + e_4 - e_5 - e_6)$$

Let λ_i^\vee denote the co-root of λ_i . Then the matrix $[\langle \lambda_i, \lambda_j^\vee \rangle]_{i,j=1,2}$ gives

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

which is precisely the Cartan Matrix for a root system of type A_2 . Hence, $\bar{\Phi}$ is a root system of type A_2 . Since we have $|\bar{\Delta}| = 2$, we say the *restricted rank* of θ is 2. Formally, we define the restricted rank as follows.

Definition 5.1.1 (Restricted Rank). *Let θ be an involution on the root system of \mathfrak{g} , with basis Δ . Then the restricted rank of θ is given by $|\bar{\Delta}|$.*

We denote by Φ_θ the set of restricted roots. That is,

$$\Phi_\theta = \{\pi(\alpha) \mid \alpha \in \Phi\}$$

5.2 θ -Normality

Let $\Phi' = \{\alpha \in \Phi \mid \frac{1}{2}\alpha \notin \Phi\}$ denote the set of indivisible roots. We then define θ -normality as follows:

Definition 5.2.1. *We say Φ is θ -normal if for all $\alpha \in \Phi'$ such that $\theta(\alpha) \neq \alpha$ we have $\theta(\alpha) + \alpha \notin \Phi$.*

One reason we may care about admissibility is due to the following results by Helminck [5]. First, if Φ is θ -normal, then Φ_θ is a root system with Weyl group W_θ , the restricted Weyl group with respect to the action of θ . Then we have the following:

Lemma 5.2.2. *If $\theta \in \text{Aut}(\Phi(\mathfrak{t}))$ is an admissible involution then $\Phi(\mathfrak{t})$ is θ -normal.*

5.3 Admissibility Criteria

From Section 4.1 in [5] we have that θ is *admissible* if and only if it can be lifted to $\bar{\theta} \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ in such a way that $c_{\alpha, \bar{\theta}} = 1$ for all roots fixed by θ . This criteria imposes some additional restraint when compared to Proposition 4.3.2. We can modify the criteria for lifting found in this proposition to give us the basis for a new algorithm. Our new algorithm will check for admissibility via similar methods which we used to check for lifting.

A corollary to Proposition 4.3.2 from [5] gives us the following result.

Corollary 5.3.1. *Let $\theta \in \text{Aut}(\Phi)$ and Δ be a θ -basis of Φ . Then θ is admissible if and only if there is some $H \in \mathfrak{t}$ so that*

1. $\alpha(H) \theta(\alpha)(H) c_{\theta(\alpha), \theta_\Delta} c_{\alpha, \theta_\Delta} = 1$ for all $\alpha \in \Delta$ which are not fixed by θ .
2. $\alpha(H) = 1$ for all $\alpha \in \Delta$ fixed by θ .

We can set up and solve a very similar system of multivariate polynomial equations. This works in the following way. For $\alpha \in \Delta$ not fixed by θ , we set up the equations we used for lifting. That is, we can use the $F_{0,i}$ and $F_{1,i}$ polynomials (Equations 4.7). For these roots we have

$$f_i := c_{\alpha, \theta_\Delta} c_{\theta(\alpha), \theta_\Delta} F_{0,i} F_{1,i} - 1$$

For the roots α which are fixed by θ we have much simpler polynomials. Let us call them “G polynomials” and write

$$G_i := \alpha_i(x_1 H_{\alpha_1} + x_2 H_{\alpha_2} + \dots + x_n H_{\alpha_n}) \quad (5.3)$$

Due to the linearity of α_i we can simplify this expression. We write

$$G_i := x_1 \alpha_i(H_{\alpha_1}) + x_2 \alpha_i(H_{\alpha_2}) + \dots + x_n \alpha_i(H_{\alpha_n}) \quad (5.4)$$

We now have for the roots fixed by θ the polynomial

$$f_i := G_i - 1$$

5.4 An Algorithm to Check for Admissibility

We are now able to present our algorithm. Because we only are concerned with the existence of such a vector H , and not the vector itself, it suffices to only compute the Groebner basis and check if it equals $\{1\}$. If it does not, such a vector H exists, and θ is therefore admissible.

Algorithm 5.4.1 (Admissibility Check).

Input θ , an involutorial automorphism in $\text{Aut}(\Phi)$; \mathfrak{g} , a Lie Algebra with roots Φ , having basis roots Δ

Output **TRUE** if θ can be lifted, **N/A** if the correction vector is not necessary, or **FALSE** if the correction vector is necessary, but cannot be found.

1. Call Algorithm 4.2.3 or Algorithm 7.8.1 to compute the structure constants $c_{\alpha,\theta}$ for every $\alpha \in \Phi$
2. **if** $c_{\theta(\alpha),\theta_\Delta} = 1$ for all $\alpha \in \Delta$ **then return** **N/A**
3. **for** $i = 1$ to n
 - if** α_i corresponds to a white dot **then** $f_i := c_{\alpha,\theta_\Delta} c_{\theta(\alpha),\theta_\Delta} F_{0,i} F_{1,i} - 1$
 - if** α_i corresponds to a black dot **then** $f_i := G_i - 1$
4. $I := \langle f_1, \dots, f_n \rangle$
5. Compute G , the reduced Groebner Basis for I .
6. **if** $G \neq \{1\}$ **then return** **TRUE**
- else return** **FALSE**

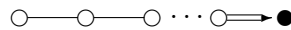
Theorem 5.4.2. Algorithm 5.4.1 works correctly

Proof. That the system of equations corresponds to admissibility is established by Corollary 5.3.1 and [5]. Proof that computation of the reduced Groebner basis establishes that a solution exists is given in Chapter 3. \square

Chapter 6

Relations Between Structure Constants and the Weyl Group

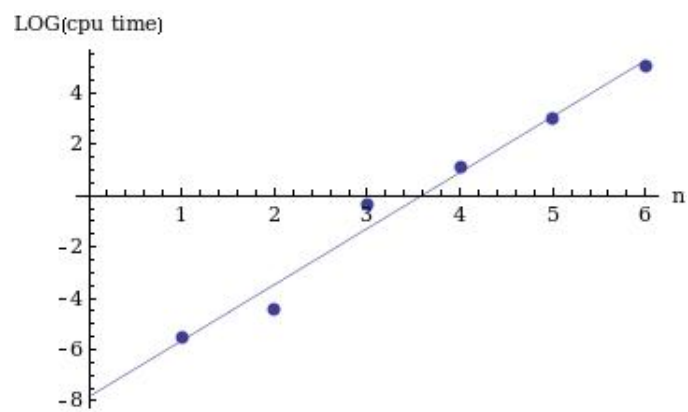
In Chapter 4 we introduced our lifting algorithm. While powerful, it will prove to be quite slow on almost any implementation. Hence, its proper place will be for computation of problems small size, where “size” is given by the number of basis roots (dots) in the Helminck diagram. Consider, for example, the case of lifting θ as induced by the diagram:



where n denotes the number of basis roots. For $n = 1$ we have θ induced over the root system A_1 . For larger n , we have θ induced over the root system B_n . As most computer implementations of Groebner bases rely on Buchberger’s algorithm (or some variant), we should expect an exponential increase in CPU processing time as the number of basis roots (n) increases. Indeed, we obtain the times in the table that follows. The subsequent figure suggests a linear relationship between n and $\log(\text{CPU time})$.

Table 6.1: CPU Timings For Lifting θ

n	CPU sec	LOG(CPU sec)
1	0.004	-5.521
2	0.012	-4.423
3	0.696	-0.362
4	2.984	1.093
5	19.829	2.987
6	153.714	5.035

Figure 6.1: CPU Timings For Lifting θ 

Our primary goal will be to quickly solve larger lifting problems. This will involve breaking up the underlying system of equations by breaking into smaller pieces the Helminck diagram, solving the lifting problem for each piece, and “gluing” the solutions together to give the overall solution. Our basic method for accomplishing this task will be to make use of relationships between the structure constants of θ and the Weyl group of the underlying root system.

In this chapter we will establish some important facts concerning the Weyl group and structure constants. In the chapters that follow we will reflect on our current discussion. In particular, we will be able to determine some information about the structure constants simply by looking at the configuration of black and white dots in the Helminck diagram.

6.1 Structure Constants for Roots Fixed by θ

If α is a root fixed by θ , then $c_{\theta(\alpha),\theta_\Delta} = 1$. This should be clear for the basis roots, because we have $c_{\alpha,\theta_\Delta} = 1$ for all $\alpha \in \Delta$. If α is a fixed root, then $\theta(\alpha) = \alpha$. It then follows that for any non-simple root α , if fixed by θ , yields $c_{\theta(\alpha),\theta_\Delta} = 1$.

Lemma 6.1.1. *Let α be a root in Φ with basis Δ . Let θ be an involution on the roots. Then if $\theta(\alpha) = \alpha$ then $c_{\theta(\alpha),\theta_\Delta} = 1$.*

Proof. Having already established the hypothesis for the basis roots, let us induct on the level of α . Assume for α of level n that $c_{\theta(\alpha_1+\dots+\alpha_n),\theta_\Delta} = 1$. Then

$$\begin{aligned} c_{\theta(\alpha_1+\dots+\alpha_n+\alpha_{n+1}),\theta_\Delta} &= \frac{N_{\theta(\alpha_1+\dots+\alpha_n),\theta(\alpha_{n+1})}}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} c_{\theta(\alpha_1+\dots+\alpha_n),\theta_\Delta} c_{\theta(\alpha_{n+1}),\theta_\Delta} \\ &= \frac{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} c_{\theta(\alpha_1+\dots+\alpha_n),\theta_\Delta} c_{\alpha_{n+1},\theta_\Delta} \\ &= c_{\theta(\alpha_1+\dots+\alpha_n),\theta_\Delta} c_{\alpha_{n+1},\theta_\Delta} \end{aligned}$$

$\alpha_{n+1} \in \Delta$, so $c_{\alpha_{n+1},\theta_\Delta} = 1$. Also $c_{\theta(\alpha_1+\dots+\alpha_n),\theta_\Delta} = 1$ by assumption. Then $c_{\theta(\alpha_1+\dots+\alpha_n+\alpha_{n+1}),\theta_\Delta} = 1$. \square

6.2 General Notes on the Action of $w_0(\theta)$ on Projecting Roots

The action of $w_0(\theta)$ on the roots fixed by θ is simply described. $w_0(\theta)$ maps all fixed roots α to the negative of some other fixed root. In the cases where θ^* , the diagram automorphism, is the identity, then this other root is $-\alpha$. In cases where θ^* is non-trivial, then θ^* maps $w_0(\theta)(\alpha)$ to $-\alpha$.

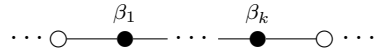
However, the action of $w_0(\theta)$ on the roots which project to some root in the local symmetric space is not so clear. It will be the goal of this section to establish what happens to $w_0(\theta)(\alpha)$, where α is denoted by a white dot.

For all cases where α is denoted by a white dot, and is not neighboring a black dot, the action is simple. For all roots β_i which compose $w_0(\theta)$, we have $(\alpha, \beta_i) = 0$. Hence, $w_0(\theta)(\alpha) = \alpha$.

That said, the cases where α is the “neighbor” of a string of fixed roots are the interesting ones. We will investigate these cases for each of the root systems of the classical type (A, B, C, and D). We are not so interested in the E, F, and G cases, as there are only a finite number of them. Arguments made in subsequent chapters will rely on the discussion we are about to make. For the E, F, and G cases, however, argument by demonstration will be our option of choice.

6.3 Action of $w_0(\theta)$ on Projecting Roots Over Type A

As stated before, the only interesting case is that of α being denoted by a white dot and neighboring a black dot. In this case, $w_0(\theta)$ serves to sum α with all the fixed roots. Let $\{\beta_1, \dots, \beta_k\}$ give the set of all fixed basis roots composing a single A-string.



For the longest element of A_k , denoted by w_0 , we have from [15]

$$w_0 = s_{\beta_1} s_{\beta_2} s_{\beta_1} s_{\beta_3} s_{\beta_2} s_{\beta_1} \dots s_{\beta_k} s_{\beta_{k-1}} \dots s_{\beta_1}$$

For convenience, let us define the following sequence of reflections

$$W(n, m) = s_{\beta_n} s_{\beta_{n-1}} s_{\beta_{n-2}} \dots s_{\beta_{m+1}} s_{\beta_m} \tag{6.1}$$

Then we have for the longest element of A_k :

$$w_0 = W(1, 1) W(2, 1) W(3, 1) \dots W(n, 1) \quad (6.2)$$

Let $\Delta = \{\alpha_1, \dots, \alpha_n\}$ denote the base of a root system of type A, with an embedded A-string from α_k to α_m , $1 \leq k, m \leq n$. Let $\alpha \in \Delta$. If α not a fixed root, nor is it a root not immediately neighboring any fixed roots β , then $(\alpha, \beta) = 0$ and $W(m, k)(\alpha) = \alpha$. Hence, $W(m, k)(\alpha_i) = \alpha_i$ if $i < k - 1$ or $i > m + 1$.

Now for a root immediately to the right of the A-string, we have

$$\begin{aligned} W(m, k)(\alpha_{m+1}) &= s_{\alpha_m} s_{\alpha_{m-1}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{m+1}) \\ &= s_{\alpha_m} s_{\alpha_{m-1}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{m+1}) \\ &= s_{\alpha_m}(\alpha_{m+1}) \\ &= \alpha_m + \alpha_{m+1} \end{aligned} \quad (6.3)$$

Then we have

$$\begin{aligned} W(m-1, k)(\alpha_m + \alpha_{m+1}) &= s_{\alpha_{m-1}} s_{\alpha_{m-2}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_m + \alpha_{m+1}) \\ &= s_{\alpha_{m-1}} s_{\alpha_{m-2}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_m) + \\ &\quad s_{\alpha_{m-1}} s_{\alpha_{m-2}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{m+1}) \\ &= s_{\alpha_{m-1}} s_{\alpha_{m-2}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_m) + \alpha_{m+1} \\ &= s_{\alpha_{m-1}}(\alpha_m) + \alpha_{m+1} \\ &= \alpha_{m-1} + \alpha_m + \alpha_{m+1} \end{aligned} \quad (6.4)$$

If we continue applying the result of $W(m-i, k)$ to $W(m-(i+1), k)$, we slowly build a string of the sum of all roots $\alpha_{m-(i+1)} + \dots + \alpha_{m+1}$. The suggestion is that $w_0(\alpha_{m+1}) = \alpha_k + \alpha_{k+1} + \dots + \alpha_{m+1}$.

Lemma 6.3.1. *If $\{\alpha_k, \alpha_{k+1}, \dots, \alpha_m\}$ gives an embedded A-string with longest element w_0 , then $w_0(\alpha_{m+1}) = \alpha_k + \alpha_{k+1} + \dots + \alpha_{m+1}$.*

Proof. If our embedded A-string is of length one, then we have

$$w_0(\alpha_{m+1}) = s_{\alpha_m}(\alpha_{m+1}) = \alpha_m + \alpha_{m+1}.$$

Let us proceed via induction. Assume $w_0(\alpha_{m+1}) = \alpha_k + \alpha_{k+1} + \dots + \alpha_{m+1}$. Then we have

$$W(k, k) W(k+1, k) \dots W(m-1, k) W(m, k)(\alpha_{m+1}) = \alpha_k + \alpha_{k+1} + \dots + \alpha_{m+1} \quad (6.5)$$

We now add one fixed root to our A-string. Do this by coloring black the dot corresponding to α_{k-1} . Call the longest element of the new A-string v_0 . Then

$$v_0 = W(k-1, k-1) W(k, k-1) \dots W(m-1, k-1) W(m, k-1)$$

Since $(\alpha_{m+1}, \alpha_i) = 0$ for $i < m$ then

$$\begin{aligned} v_0(\alpha_{m+1}) &= W(k-1, k-1) W(k, k-1) \dots W(m-1, k-1) W(m, k-1)(\alpha_{m+1}) \\ &= W(k-1, k-1) W(k, k) \dots W(m-1, k) W(m, k)(\alpha_{m+1}) \\ &= W(k-1, k-1) w_0(\alpha_{m+1}) \\ &= W(k-1, k-1) (\alpha_k + \alpha_{k+1} + \dots + \alpha_{m+1}) \\ &= \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{m+1} \end{aligned}$$

□

Now if α is a root bordering the A-string on the left, we have similar behavior. We have

$$\begin{aligned} s_{\alpha_i}(\alpha_{i-1} + \alpha_i + \alpha_{i+1}) &= \alpha_{i-1} + \alpha_i - \alpha_i + \alpha_i + \alpha_{i+1} \\ &= \alpha_{i-1} + \alpha_i + \alpha_{i+1} \end{aligned} \quad (6.6)$$

Hence, it will be the case that

$$s_{\alpha_m} s_{\alpha_{m-1}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{k+1} + \dots + \alpha_{m-1}) = \alpha_{k+1} + \dots + \alpha_{m-1} \quad (6.7)$$

We have

$$\begin{aligned} W(m, k)(\alpha_{k-1}) &= s_{\alpha_m} s_{\alpha_{m-1}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{k-1}) \\ &= \alpha_{k-1} + \alpha_k + \dots + \alpha_{m-1} + \alpha_m \end{aligned} \quad (6.8)$$

And hence, from Equation 6.7,

$$W(m-1, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{m-1} + \alpha_m) = \alpha_{k-1} + \alpha_k + \dots + \alpha_{m-1} + \alpha_m \quad (6.9)$$

We can now make the following claim.

Lemma 6.3.2. *If $\{\alpha_k, \alpha_{k+1}, \dots, \alpha_m\}$ gives an embedded A -string with longest element w_0 , then $w_0(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_m$.*

Proof. We start with

$$w_0(\alpha_{k-1}) = W(k, k) W(k+1, k) \dots W(m, k)(\alpha_{k-1})$$

By Equation 6.8 we have

$$w_0(\alpha_{k-1}) = W(k, k) W(k+1, k) \dots W(m-1, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{m-1} + \alpha_m)$$

And by Equation 6.7 we have

$$w_0(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \dots + \alpha_{m-1} + \alpha_m$$

□

We can now make a general conclusion.

Corollary 6.3.3. *If $\{\alpha_k, \alpha_{k+1}, \dots, \alpha_m\}$ gives an embedded A -string with longest element w_0 , and α_i is a root denoted by a white dot and neighboring the A -string, then $w_0(\alpha_i) = \alpha_i + \alpha_k + \alpha_{k+1} + \dots + \alpha_m$.*

Proof immediately follows Lemmas 6.3.1 and 6.3.2.

6.4 Action of $w_0(\theta)$ on Projecting Roots Over Type B

Within a Helminck diagram of type B, the only configuration which gives α neighboring an embedded B-string is if the B-string lies to the right (with respect to the diagram below) of α .



For the longest element of B_k , we have from [15]

$$w_0 = s_{\beta_1} s_{\beta_2} s_{\beta_1} s_{\beta_3} s_{\beta_2} s_{\beta_1} \dots s_{\beta_k} s_{\beta_{k-1}} \dots s_{\beta_1} s_{\beta_k} s_{\beta_{k-1}} \dots s_{\beta_2} s_{\beta_k} s_{\beta_{k-1}} \dots s_{\beta_3} \dots s_{\beta_k} \quad (6.10)$$

Written in terms of $W(m, k)$ as per Equation 6.1, we have

$$w_0 = W(1, 1) W(2, 1) W(3, 1) \dots W(n, 1) W(n, 2) W(n, 3) \dots W(n, n) \quad (6.11)$$

Because we are now working over type B, the action of $W(m, k)$ on the roots is slightly different than that of the action over type A. However, we still have $s_{\alpha_i}(\alpha_j) = \alpha_j$ if $|i - j| > 1$.

Let $\Delta = \{\alpha_1, \dots, \alpha_n\}$ denote the base of a root system of type B, with an embedded B-string from α_k to α_m , $1 \leq k$. In order to have an embedded B-string, we have $m = n$. Let $\alpha \in \Delta$. If α not a fixed root, nor is it a root not immediately neighboring any fixed roots β , then $W(m, k)(\alpha) = \alpha$. Hence, $W(m, k)(\alpha_i) = \alpha_i$ if $i < k - 1$.

If $i = k - 1$ then $W(n, j)(\alpha_i) = \alpha_i$ for $j = 2 \dots n$. Hence, we can eliminate from w_0 the components $W(n, 2), W(n, 3), \dots, W(n, n)$. In effect, the action of w_0 on the root neighboring the B-string is precisely that of the longest element of an A-string with the same length as the B-string. It should be clarified that the action of the longest element of an A-string over the root system of type B is different. What we can do for the neighbor root α_{k-1} is write

$$\begin{aligned}
w_0(\alpha_{k-1}) &= W(k, k) W(k+1, k) W(k+2, k) \dots W(n, k) \\
&\quad W(n, k+1) W(n, k+2) \dots W(n, n)(\alpha_{k-1}) \\
&= W(k, k) W(k+1, k) W(k+2, k) \dots W(n, k)(\alpha_{k-1})
\end{aligned} \tag{6.12}$$

From the $n \times n$ Cartan matrix over type B we can write

$$s_{\alpha_i}(\alpha_j) = \begin{cases} -\alpha_j & \text{if } i = j; \\ \alpha_i + \alpha_j & \text{if } |i - j| = 1 \text{ and } i \neq n; \\ \alpha_i + 2\alpha_j & \text{if } |i - j| = 1 \text{ and } i = n; \\ \alpha_j & \text{else.} \end{cases} \tag{6.13}$$

Then the action of $W(i, k)$ is the same over the root system of type B as it is over the root system of type A with the exception of the case $i = n$. For this case we have

$$\begin{aligned}
W(n, k)(\alpha_{k-1}) &= s_{\alpha_n} s_{\alpha_{n-1}} s_{\alpha_{n-2}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{k-1}) \\
&= s_{\alpha_n}(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1}) \\
&= \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + 2\alpha_n
\end{aligned} \tag{6.14}$$

Then we have

$$\begin{aligned}
W(n-1, k)(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + 2\alpha_n) &= \\
W(n-1, k)(\alpha_{k-1}) + W(n-1, k)(\alpha_k) + W(n-1, k)(\alpha_{k+1}) + \dots + \\
W(n-1, k)(\alpha_{n-1}) + 2W(n-1, k)(\alpha_n) &= \\
\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + 2\alpha_{n-1} + 2\alpha_n
\end{aligned} \tag{6.15}$$

And then

$$\begin{aligned}
W(n-2, k)(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + 2\alpha_n) &= \\
\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + 2\alpha_{n-2} + 2\alpha_{n-1} + 2\alpha_n
\end{aligned} \tag{6.16}$$

If we continue applying $W(i, k)$ to the result of $W(i+1, k)$, decreasing until $i = k$, the suggestion is that we have

$$\alpha_{k-1} + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{n-2} + 2\alpha_{n-1} + 2\alpha_n$$

Let us make the following claim.

Lemma 6.4.1. *If $\{\alpha_k, \alpha_{k+1}, \dots, \alpha_m\}$ gives an embedded B-string with longest element w_0 , and α_i is a root denoted by a white dot and neighboring the B-string, then $w_0(\alpha_i) = \alpha_i + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_m$, where $i = k - 1$.*

Proof. From Equation 6.14 we have

$$W(n, k)(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + 2\alpha_n$$

Which, for notational convenience we shall write as

$$W(n - 0, k)(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + 2\alpha_{n-0}$$

Then let us then show that

$$\begin{aligned} W(n - i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \dots + 2\alpha_{n-i} + \dots + 2\alpha_n & \end{aligned} \quad (6.17)$$

where $i > 0$. Let us first write Equation 6.17 as

$$\begin{aligned} W(n - i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_n) &= \\ W(n - i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + \dots + \alpha_n) &+ W(n - i, k)(\alpha_{n-i+1} + \dots + \alpha_n) \end{aligned} \quad (6.18)$$

Now because $k < n$ (as a root system of type B must have at least two basis roots), and $i > 0$, then the Cartan matrix gives us $W(n - i, k)$ acts on all roots as it would if defined over a root system of type A. Hence

$$W(n-i, k)(\alpha_{k-1} + \dots + \alpha_n) = \alpha_{k-1} + \dots + \alpha_n \quad (6.19)$$

and

$$\begin{aligned} W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_n) &= s_{\alpha_{n-i}} s_{\alpha_{n-i-1}} \dots s_{\alpha_k}(\alpha_{n-i+1} + \dots + \alpha_n) \\ &= s_{\alpha_{n-i}}(\alpha_{n-i+1} + \dots + \alpha_n) \\ &= \alpha_{n-i} + \alpha_{n-i+1} + \dots + \alpha_n \end{aligned} \quad (6.20)$$

Then from Equations 6.19 and 6.20 we can write

$$\begin{aligned} W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_n) &= \\ W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + \dots + \alpha_n) + W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i-1} + 2\alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_n & \end{aligned} \quad (6.21)$$

The effect is that by applying $W(n-i, k)$ for $i = 1$ to $n-k$ in incrementing order of i , we add to

$$\alpha_{k-1} + \alpha_k + \dots + \alpha_n$$

the root α_{n-i} . Application of $W(n-i, k)$ in this way is precisely the action of w_0 on α_{k-1} . Hence

$$w_0(\alpha_{k-1}) = \alpha_{k-1} + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_m$$

□

6.5 Action of $w_0(\theta)$ on Projecting Roots Over Type C

Within a Helminck diagram of type C, the only configuration which gives α neighboring an embedded C-string is if the C-string lies to the right (with respect to the diagram

below) of α . We are primarily interested in the configuration which gives an A-string of length 1 to the left.



Root systems of types B and C have same the longest element. Hence, recall the longest element over type B is given in Equation 6.11.

$$w_0 = W(1, 1) W(2, 1) W(3, 1) \dots W(n, 1) W(n, 2) W(n, 3) \dots W(n, n)$$

Note we are now working over type C, so the action of $W(m, k)$ on the roots is slightly different than that of the action over type B we previously discussed. We still have $s_{\alpha_i}(\alpha_j) = \alpha_j$ if $|i - j| > 1$.

Let $\Delta = \{\alpha_1, \dots, \alpha_n\}$ denote the base of a root system of type C, with an embedded C-string from α_k to α_m , $1 \leq k$. As in the B-string case, in order to have an embedded C-string, we have $m = n$. Let $\alpha \in \Delta$. If α not a fixed root, nor is it a root not immediately neighboring any fixed roots β , then $W(m, k)(\alpha) = \alpha$. Hence, $W(m, k)(\alpha_i) = \alpha_i$ if $i < k - 1$.

As with the B-string case, if $i = k - 1$ then $W(n, j)(\alpha_i) = \alpha_i$ for $j = 2 \dots n$. Hence, we can eliminate from w_0 the components $W(n, 2), W(n, 3), \dots, W(n, n)$. The effect is the same as previously stated. The action of w_0 on the root neighboring the C-string is precisely that of the longest element of an A-string with the same length as the C-string. Again, it should be clarified that the action of the longest element of an A-string over the root system of type C is different from both cases of over a system of type A or B. For the neighbor root α_{k-1} we write

$$\begin{aligned} w_0(\alpha_{k-1}) &= W(k, k) W(k+1, k) W(k+2, k) \dots W(n, k) \\ &\quad W(n, k+1) W(n, k+2) \dots W(n, n)(\alpha_{k-1}) \\ &= W(k, k) W(k+1, k) W(k+2, k) \dots W(n, k)(\alpha_{k-1}) \end{aligned} \tag{6.22}$$

From the $n \times n$ Cartan matrix over type C we can write

$$s_{\alpha_i}(\alpha_j) = \begin{cases} -\alpha_j & \text{if } i = j; \\ \alpha_i + \alpha_j & \text{if } |i - j| = 1 \text{ and } i \neq n - 1; \\ \alpha_i + 2\alpha_j & \text{if } |i - j| = 1 \text{ and } i = n - 1; \\ \alpha_j & \text{else.} \end{cases} \quad (6.23)$$

Then the action of $W(i, k)$ is the same over the root system of type C as it is over the root system of type A with the exception of the case $i = n - 1$. For this case we have

$$\begin{aligned} W(n, k)(\alpha_{k-1}) &= s_{\alpha_n} s_{\alpha_{n-1}} s_{\alpha_{n-2}} \dots s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{k-1}) \\ &= s_{\alpha_n} s_{\alpha_{n-1}}(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-2}) \\ &= s_{\alpha_n}(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-2} + \alpha_{n-1}) \\ &= \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + \alpha_n \end{aligned} \quad (6.24)$$

Then we have

$$\begin{aligned} W(n-1, k)(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + \alpha_n) &= \\ W(n-1, k)(\alpha_{k-1}) + W(n-1, k)(\alpha_k) + W(n-1, k)(\alpha_{k+1}) + \dots + \\ W(n-1, k)(\alpha_{n-1}) + W(n-1, k)(\alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + 2\alpha_{n-1} + \alpha_n \end{aligned} \quad (6.25)$$

And then

$$\begin{aligned} W(n-2, k)(\alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-1} + 2\alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + 2\alpha_{n-2} + 2\alpha_{n-1} + \alpha_n \end{aligned} \quad (6.26)$$

If we continue applying $W(i, k)$ to the result of $W(i+1, k)$, decreasing until $i = k$, the suggestion is that we have

$$\alpha_{k-1} + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{n-2} + 2\alpha_{n-1} + \alpha_n$$

Formally we state the following.

Lemma 6.5.1. *If $\{\alpha_k, \alpha_{k+1}, \dots, \alpha_m\}$ gives an embedded C-string with longest element w_0 , and α_i is a root denoted by a white dot and neighboring the C-string, then $w_0(\alpha_i) = \alpha_i + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{m-1} + \alpha_m$, where $i = k - 1$.*

Proof. From Equation 6.24 we have

$$W(n, k)(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-2} + \alpha_{n-1} + \alpha_n$$

Which, for notational convenience we shall write as

$$W(n - 0, k)(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-2} + \alpha_{n-1} + \alpha_{n-0}$$

Following from Equation 6.25 we have

$$W(n - 1, k)(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \alpha_{k+1} + \dots + \alpha_{n-2} + 2\alpha_{n-1} + \alpha_{n-0}$$

Then let us then show that

$$\begin{aligned} W(n - i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-1} + \alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \dots + 2\alpha_{n-i} + \dots + 2\alpha_{n-1} + \alpha_n & \end{aligned} \quad (6.27)$$

where $i > 1$. Let us first write Equation 6.27 as

$$\begin{aligned} W(n - i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-1} + \alpha_n) &= \\ W(n - i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + \dots + \alpha_n) + W(n - i, k)(\alpha_{n-i+1} + \dots + \alpha_{n-1}) & \end{aligned} \quad (6.28)$$

Now because $k < n$ (as a root system of type C must have at least three basis roots), and $i > 1$, then the Cartan matrix gives us $W(n - i, k)$ acts on all roots as it would if defined over a root system of type C. Hence

$$W(n - i, k)(\alpha_{k-1} + \dots + \alpha_n) = \alpha_{k-1} + \dots + \alpha_n \quad (6.29)$$

and

$$\begin{aligned}
W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_{n-1}) &= s_{\alpha_{n-i}} s_{\alpha_{n-i-1}} \dots s_{\alpha_k}(\alpha_{n-i+1} + \dots + \alpha_{n-1}) \\
&= s_{\alpha_{n-i}}(\alpha_{n-i+1} + \dots + \alpha_{n-1}) \\
&= \alpha_{n-i} + \alpha_{n-i+1} + \dots + \alpha_{n-1}
\end{aligned} \tag{6.30}$$

Then from Equations 6.29 and 6.30 we can write

$$\begin{aligned}
&W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-1} + \alpha_n) = \\
W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + \dots + \alpha_n) &+ W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_{n-1}) = \\
&\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i-1} + 2\alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-1} + \alpha_n
\end{aligned} \tag{6.31}$$

The effect is that by applying $W(n-i, k)$ for $i = 1$ to $n-k-1$ in incrementing order of i , we add to

$$\alpha_{k-1} + \alpha_k + \dots + \alpha_n$$

the root α_{n-i} . Application of $W(n-i, k)$ in this way is precisely the action of w_0 on α_{k-1} . Hence

$$w_0(\alpha_{k-1}) = \alpha_{k-1} + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{n-1} + \alpha_n$$

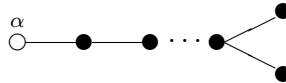
□

As a final remark, the involution θ induced by the diagram at the start of this section includes an A-string of length one. Hence, the embedded root system is $A_1 \times C_{n-2}$. The longest element of A_1 is s_{α_1} . So we have

$$w_0(\alpha_2) = \alpha_1 + \alpha_2 + 2\alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n \tag{6.32}$$

6.6 Action of $w_0(\theta)$ on Projecting Roots Over Type D

Within a Helminck diagram of type D, the only configuration which gives α neighboring an embedded D-string is if the D-string lies to the right (with respect to the diagram below) of α . We are primarily interested in the configuration described below.



The longest element of a root system of type D can be conveniently written in one of two slightly varying ways. The exact expression will depend on whether the number of basis roots n is even or odd. However, one “scheme” to describe both variations can be written. First recall Equation 6.1.

$$W(n, m) = s_{\beta_n} s_{\beta_{n-1}} s_{\beta_{n-2}} \cdots s_{\beta_{m+1}} s_{\beta_m}$$

We will introduce a second shorthand describing a similar series of reflections.

$$Y(n, m) = s_{\beta_n} s_{\beta_{n-2}} \cdots s_{\beta_{m+1}} s_{\beta_m} \tag{6.33}$$

This series is the same as described via $W(n, m)$ except the $s_{\beta_{n-1}}$ component has been omitted. Based on [15], we then have for n even

$$\begin{aligned} w_0 = & W(1, 1)W(2, 1) \cdots W(n-1, 1) \\ & Y(n, 1)W(n-1, 2)Y(n, 3)W(n-1, 4) \cdots W(n-1, n-2)Y(n, n-1) \end{aligned} \tag{6.34}$$

The pattern starts in a similar manner as w_0 does for the previous types. Specifically, we have the usual $W(1, 1)W(2, 1) \cdots$ pattern. However, this component terminates at $W(n-1, 1)$. Afterward, we alternate $Y(n, i)$ and $W(n-1, i)$ components, incrementing i from 1 on each step. The pattern stops when i reaches $n-1$. This must be so because the next component would be $W(n-1, n)$, which is not defined. The fact that $Y(n, n-1)$

is the last step is due to n being even. Should n be odd, we would have the variant with $W(n-1, n-1)$ being last. This is as follows:

$$\begin{aligned} w_0 = & W(1, 1)W(2, 1) \dots W(n-1, 1) \\ & Y(n, 1)W(n-1, 2)Y(n, 3)W(n-1, 4) \dots W(n-1, n-3)Y(n, n-2) \\ & W(n-1, n-1) \end{aligned} \quad (6.35)$$

Next we would like to describe how the $W(m, k)$ and $Y(m, k)$ components act on the roots. Due to the orientation of our Dynkin diagrams for our root systems, we must only “step cautiously” for the roots near the “Y split” at the right end. That is, when our reflections involve the last three roots. For the other roots, we have the usual properties associated with the A type root systems. Namely, we still have $s_{\alpha_i}(\alpha_j) = \alpha_j$ if $|i - j| > 1$, but also requiring $i < n - 2$ and $j < n - 2$. We always have $s_{\alpha_i}(\alpha_j) = \alpha_j$ if $|i - j| > 2$.

Let $\Delta = \{\alpha_1, \dots, \alpha_n\}$ denote the base of a root system of type D, with an embedded D-string from α_k to α_m , $1 \leq k$. In order to have an embedded D-string, we have $m = n$. Let $\alpha \in \Delta$. If α not a fixed root, nor is it a root not immediately neighboring any fixed roots β , then $W(m, k)(\alpha) = \alpha$. Hence, $W(m, k)(\alpha_i) = \alpha_i$ if $i < k - 1$. (note that the neighboring root is always a distance of at least two from the “split” because an embedded D-string must include at least four roots).

With the condition $i = k - 1$, we would like to eliminate from w_0 as many components as possible. Thankfully, because this root lies to the “left” of the “split”, then we can peel away many of the components that lie in the $W()Y()W()Y()$ alternating part. In particular, for any component for which the second argument (of $W()$ or $Y()$) exceeds 1, we have $s_{\alpha_i}(\alpha_j) = \alpha_j$, $|i - j| > 1$, $i < n - 2$, and $j < n - 2$ for all reflections in the $W()$ or $Y()$ component. Hence, we eliminate everything to the right of (but not including) the $Y(n, 1)$ term of the sequence. For our purposes, we can write

$$w_0 = W(k, k)W(k+1, k) \dots W(n-1, k)Y(n, k) \quad (6.36)$$

From the $n \times n$ Cartan matrix over type D we can write

$$s_{\alpha_i}(\alpha_j) = \begin{cases} -\alpha_j & \text{if } i = j; \\ \alpha_i + \alpha_j & \text{if } (|i - j| = 1 \text{ and } i \neq n - 1) \\ & \text{or } (i = n - 2 \text{ and } j = n) \\ & \text{or } (i = n \text{ and } j = n - 2); \\ \alpha_j & \text{else.} \end{cases} \quad (6.37)$$

Hence, the action of $Y(n, k)$ on α_{k-1} is given below. For all reflections except those cases involving the roots $n - 2$, $n - 1$, or n , we can work as though we were in a system of type A.

$$\begin{aligned} Y(n, k)(\alpha_{k-1}) &= s_{\alpha_n} s_{\alpha_{n-2}} s_{\alpha_{n-3}} \cdots s_{\alpha_{k+2}} s_{\alpha_{k+1}} s_{\alpha_k}(\alpha_{k-1}) \\ &= s_{\alpha_n} s_{\alpha_{n-2}} s_{\alpha_{n-3}} \cdots s_{\alpha_{k+2}} s_{\alpha_{k+1}}(\alpha_{k-1} + \alpha_k) \\ &= s_{\alpha_n} s_{\alpha_{n-2}}(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3}) \\ &= s_{\alpha_n}(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2}) \\ &= \alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_n \end{aligned} \quad (6.38)$$

If we take this result and apply it to $W(n - 1, k)$ we get

$$\begin{aligned} W(n - 1, k)(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_n) &= \\ s_{\alpha_{n-1}} s_{\alpha_{n-2}} \cdots s_{\alpha_k}(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_n) &= \\ s_{\alpha_{n-1}}(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n & \end{aligned} \quad (6.39)$$

Now we apply $W(n - 2, k)$ to the result above.

$$\begin{aligned} W(n - 2, k)(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\ s_{\alpha_{n-2}} s_{\alpha_{n-3}} \cdots s_{\alpha_k}(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\ s_{\alpha_{n-2}}(\alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\ \alpha_{k-1} + \alpha_k + \cdots + \alpha_{n-3} + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n & \end{aligned} \quad (6.40)$$

Finally, apply $W(n - 3, k)$ to the result above.

$$\begin{aligned}
W(n-3, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-3} + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\
s_{\alpha_{n-3}} s_{\alpha_{n-4}} \dots s_{\alpha_k}(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\
s_{\alpha_{n-3}}(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\
\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-4} + 2\alpha_{n-3} + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n &
\end{aligned} \tag{6.41}$$

If we continue applying $W(i, k)$ to the result of $W(i+1, k)$, decreasing until $i = k$, the suggestion is that we have

$$\alpha_{k-1} + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n$$

Formally we state the following.

Lemma 6.6.1. *If $\{\alpha_k, \alpha_{k+1}, \dots, \alpha_m\}$ gives an embedded C -string with longest element w_0 , and α_i is a root denoted by a white dot and neighboring the D -string, then $w_0(\alpha_i) = \alpha_i + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{m-2} + \alpha_{m-1} + \alpha_m$, where $i = k-1$.*

Proof. From Equation 6.38 we have

$$Y(n, k)(\alpha_{k-1}) = \alpha_{k-1} + \alpha_k + \dots + \alpha_{n-3} + \alpha_{n-2} + \alpha_n$$

then from Equation 6.39 we have

$$\begin{aligned}
W(n-1, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-3} + \alpha_{n-2} + \alpha_n) &= \\
\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-3} + \alpha_{n-2} + \alpha_{n-1} + \alpha_n &
\end{aligned}$$

Now that we have filled in the “hole” left by the omission of α_{n-1} from Equation 6.38, we want to work our way down the rest of the terms $W(n-i, k)$ where i increments from 2 to k . We will establish that

$$\begin{aligned}
W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\
\alpha_{k-1} + \alpha_k + \dots + 2\alpha_{n-i} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n &
\end{aligned} \tag{6.42}$$

where $i > 1$.

Let us first write Equation 6.42 as

$$\begin{aligned} W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\ W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + \dots + \alpha_n) + W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_{n-2}) & \end{aligned} \quad (6.43)$$

Now because $k < n - 2$ (as a root system of type D must have at least four basis roots), and $i > 1$, then the Cartan matrix gives us $W(n-i, k)$ acts on all roots as it would if defined over a root system of type D. Hence

$$W(n-i, k)(\alpha_{k-1} + \dots + \alpha_n) = \alpha_{k-1} + \dots + \alpha_n \quad (6.44)$$

and

$$\begin{aligned} W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_{n-1}) &= s_{\alpha_{n-i}} s_{\alpha_{n-i-1}} \dots s_{\alpha_k}(\alpha_{n-i+1} + \dots + \alpha_{n-2}) \\ &= s_{\alpha_{n-i}}(\alpha_{n-i+1} + \dots + \alpha_{n-2}) \\ &= \alpha_{n-i} + \alpha_{n-i+1} + \dots + \alpha_{n-2} \end{aligned} \quad (6.45)$$

Then from Equations 6.44 and 6.45 we can write

$$\begin{aligned} W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n) &= \\ W(n-i, k)(\alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i} + \dots + \alpha_n) + W(n-i, k)(\alpha_{n-i+1} + \dots + \alpha_{n-2}) &= \\ \alpha_{k-1} + \alpha_k + \dots + \alpha_{n-i-1} + 2\alpha_{n-i} + 2\alpha_{n-i+1} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n & \end{aligned} \quad (6.46)$$

The effect is that by applying $W(n-i, k)$ for $i = 1$ to $n - k - 1$ in incrementing order of i , we add to

$$\alpha_{k-1} + \alpha_k + \dots + \alpha_n$$

the root α_{n-i} . Application of $W(n-i, k)$ in this way is precisely the action of w_0 on α_{k-1} . Hence

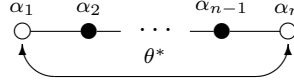
$$w_0(\alpha_{k-1}) = \alpha_{k-1} + 2\alpha_k + 2\alpha_{k+1} + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_m$$

□

6.7 Identities on the Structure Constants over a Root System of Type A

Consider the Helminck diagram over a root system of type A depicted below, which induces an involution θ :

Figure 6.2: Helminck Diagram for Restricted Rank One Involution of Type 5



The structure constant $c_{\theta(\alpha_1), \theta_\Delta}$ will be of particular interest in upcoming discussion. From Corollary 6.3.3 we know

$$w_0(\theta)(\alpha_1) = \alpha_1 + \alpha_2 + \dots + \alpha_{n-1}$$

and hence

$$\theta(\alpha_1) = -\alpha_2 - \alpha_3 - \dots - \alpha_{n-1} - \alpha_n$$

Then

$$c_{\theta(\alpha_1), \theta_\Delta} = c_{-\alpha_2 - \alpha_3 - \dots - \alpha_{n-1} - \alpha_n, \theta_\Delta} \quad (6.47)$$

Because $c_{\alpha, \theta_\Delta} = c_{-\alpha, \theta_\Delta}^{-1}$ then

$$c_{\theta(\alpha_1),\theta_\Delta} = c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1}+\alpha_n,\theta_\Delta} \quad (6.48)$$

We now claim that the value of this structure constant is always 1. To do so, we make use of the identities in Section 2.8 and Lemma 4.2.2.

Lemma 6.7.1. *Let θ be an involution induced by the Helminck diagram in Figure 6.2. Then $c_{\theta(\alpha_1),\theta_\Delta} = c_{\theta(\alpha_n),\theta_\Delta} = 1$*

Proof. Equivalently, we can show $c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1}+\alpha_n,\theta_\Delta} = 1$. We can split this constant as per Lemma 4.2.2-2. Then

$$c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1}+\alpha_n,\theta_\Delta} = c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},\theta_\Delta} c_{\alpha_n,\theta_\Delta} \frac{N_{\theta(\alpha_2+\alpha_3+\dots+\alpha_{n-1}),\theta(\alpha_n)}}{N_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},\alpha_n}}$$

Since $\alpha_2 + \alpha_3 + \dots + \alpha_{n-1}$ is a fixed root, then $c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},\theta_\Delta} = 1$. Also, $\alpha_n \in \Delta$, so $c_{\alpha_n,\theta_\Delta} = 1$. Then we have

$$c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1}+\alpha_n,\theta_\Delta} = \frac{N_{\theta(\alpha_2+\alpha_3+\dots+\alpha_{n-1}),\theta(\alpha_n)}}{N_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},\alpha_n}}$$

which gives

$$\begin{aligned} c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1}+\alpha_n,\theta_\Delta} &= \frac{N_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},-\alpha_1-\alpha_2-\dots-\alpha_{n-1}}}{N_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},\alpha_n}} \\ &= \frac{N_{\alpha_1,\alpha_2+\dots+\alpha_{n-1}}}{N_{\alpha_2+\alpha_3+\dots+\alpha_{n-1},\alpha_n}} \end{aligned} \quad (6.49)$$

via Equation 2.21. We have equality if

$$N_{\alpha_1,\alpha_2+\dots+\alpha_{n-1}} = N_{\alpha_2+\dots+\alpha_{n-1},\theta^*(\alpha_1)}$$

To show equality, we first consider Equation 2.11 with

$$\begin{aligned} \alpha &= \alpha_1 \\ \beta &= \alpha_2 + \dots + \alpha_{n-1} \\ \gamma &= \alpha_n \\ \delta &= -\alpha_1 - \dots - \alpha_n \end{aligned}$$

This gives us that

$$\begin{aligned} N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_n, -\alpha_1 - \dots - \alpha_n} + N_{\alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_1, -\alpha_1 - \dots - \alpha_n} + \\ N_{\alpha_n, \alpha_1} N_{\alpha_2 + \dots + \alpha_{n-1}, -\alpha_1 - \dots - \alpha_n} = 0 \end{aligned}$$

But $\alpha_1 + \alpha_n$ is not a root (since n must be at least three to have a fixed root in the diagram), so $N_{\alpha_n, \alpha_1} = 0$ and

$$N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_n, -\alpha_1 - \dots - \alpha_n} + N_{\alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_1, -\alpha_1 - \dots - \alpha_n} = 0$$

which we write as

$$N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_n, -\alpha_1 - \dots - \alpha_n} = -N_{\alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_1, -\alpha_1 - \dots - \alpha_n}$$

Applying Equation 2.21 we have

$$N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_1 + \dots + \alpha_{n-1}, \alpha_n} = -N_{\alpha_2 + \dots + \alpha_{n-1}, \alpha_n} N_{\alpha_2 + \dots + \alpha_n, \alpha_1} \quad (6.50)$$

Next we want to show that $N_{\alpha_1, \alpha_2 + \dots + \alpha_n} = -N_{\alpha_1 + \dots + \alpha_{n-1}, \alpha_n}$. First consider $N_{\alpha_1, \alpha_2 + \dots + \alpha_n}$. This is equivalent to $-N_{\alpha_2 + \dots + \alpha_n, \alpha_1}$. Apply Equation 2.17 with

$$\begin{aligned} \alpha &= \alpha_2 + \dots + \alpha_n \\ \beta &= \alpha_1 \\ \alpha_i &= \alpha_n \end{aligned}$$

$$\alpha - \alpha_i = \alpha_2 + \dots + \alpha_{n-1}$$

to get

$$\begin{aligned} N_{\alpha_1, \alpha_2 + \dots + \alpha_n} &= -N_{\alpha_2 + \dots + \alpha_n, \alpha_1} \\ &= -N_{\alpha_n, \alpha_2 + \dots + \alpha_{n-1}} N_{\alpha_2 + \dots + \alpha_{n-1}, \alpha_1} \quad rl \\ &= -N_{\alpha_2 + \dots + \alpha_{n-1}, \alpha_n} N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-1}} \end{aligned} \quad (6.51)$$

Then apply to $N_{\alpha_1+\dots+\alpha_{n-1},\alpha_n}$ Equation 2.17 with

$$\begin{aligned}\alpha &= \alpha_1 + \dots + \alpha_{n-1} \\ \beta &= \alpha_n \\ \alpha_i &= \alpha_1 \quad \quad \quad rl\end{aligned}$$

$$\alpha - \alpha_i = \alpha_2 + \dots + \alpha_{n-1}$$

to get

$$N_{\alpha_1+\dots+\alpha_{n-1},\alpha_n} = N_{\alpha_1,\alpha_2+\dots+\alpha_{n-1}} N_{\alpha_2+\dots+\alpha_{n-1},\alpha_n} \quad rl \quad (6.52)$$

The RHS of Equations 6.51 and 6.52 are negatives of each other. Then

$$N_{\alpha_1,\alpha_2+\dots+\alpha_n} = -N_{\alpha_1+\dots+\alpha_{n-1},\alpha_n}$$

so it must follow from Equation 6.50 that

$$N_{\alpha_1,\alpha_2+\dots+\alpha_{n-1}} = N_{\alpha_2+\dots+\alpha_{n-1},\alpha_n}$$

or equivalently

$$N_{\alpha_1,\alpha_2+\dots+\alpha_{n-1}} = -N_{\alpha_n,\alpha_2+\dots+\alpha_{n-1}}$$

Then from Equation 6.49 we can write

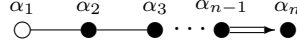
$$c_{\alpha_2+\alpha_3+\dots+\alpha_{n-1}+\alpha_n,\theta_\Delta} = 0$$

□

6.8 Identities on the Structure Constants over a Root System of Type B

Now consider the Helminck diagram over a root system of type B depicted below. Again, we denote the induced involution by θ .

Figure 6.3: Helminck Diagram for Restricted Rank One Involution of Type 6



As with the previous type, the structure constant $c_{\theta(\alpha_1), \theta_\Delta}$ will be of particular interest. From Lemma 6.4.1 we have

$$w_0(\theta)(\alpha_1) = \alpha_1 + 2\alpha_2 + \dots + 2\alpha_n$$

and hence

$$\theta(\alpha_1) = -\alpha_1 - 2\alpha_2 - \dots - 2\alpha_n$$

Then

$$c_{\theta(\alpha_1), \theta_\Delta} = c_{-\alpha_1 - 2\alpha_2 - \dots - 2\alpha_n, \theta_\Delta} \quad (6.53)$$

We now claim that the value of this structure constant is always 1. To do so, we make further use of the identities in Section 2.8 and Lemma 4.2.2.

Lemma 6.8.1. *Let θ be an involution induced by the Helminck diagram in Figure 6.3. Then $c_{\theta(\alpha_1), \theta_\Delta} = 1$*

Proof. Let n be the number of basis roots. The minimum number of basis roots which gives us the appropriate diagram is three. For n at least three we have

$$\begin{aligned} c_{\theta(\alpha_1), \theta_\Delta} &= c_{-\alpha_1 - 2\alpha_2 - \dots - 2\alpha_n, \theta_\Delta} \\ &= c_{\alpha_1 + 2\alpha_2 + \dots + 2\alpha_n, \theta_\Delta} \\ &= \frac{N_{\theta(\alpha_1 + \alpha_2 + \dots + \alpha_n), \theta(\alpha_2 + \dots + \alpha_n)}}{N_{\alpha_1 + \alpha_2 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_n}} c_{\alpha_1 + \alpha_2 + \dots + \alpha_n, \theta_\Delta} c_{\alpha_2 + \dots + \alpha_n, \theta_\Delta} \end{aligned} \quad (6.54)$$

Now we have

$$c_{\alpha_1+\alpha_2+\dots+\alpha_n,\theta_\Delta} = \frac{N_{\theta(\alpha_1),\theta(\alpha_2+\dots+\alpha_n)}}{N_{\alpha_1,\alpha_2+\dots+\alpha_n}} c_{\alpha_1,\theta_\Delta} c_{\alpha_2+\dots+\alpha_n,\theta_\Delta}$$

Since $\alpha_1 \in \Delta$, then $c_{\alpha_1,\theta_\Delta} = 1$. Also $c_{\alpha_2+\dots+\alpha_n,\theta_\Delta}^2 = 1$, so we can continue Equation 6.54 by writing

$$\begin{aligned} c_{\theta(\alpha_1),\theta_\Delta} &= \frac{N_{\theta(\alpha_1+\alpha_2+\dots+\alpha_n),\theta(\alpha_2+\dots+\alpha_n)}}{N_{\alpha_1+\alpha_2+\dots+\alpha_n,\alpha_2+\dots+\alpha_n}} \frac{N_{\theta(\alpha_1),\theta(\alpha_2+\dots+\alpha_n)}}{N_{\alpha_1,\alpha_2+\dots+\alpha_n}} \\ &= \frac{N_{-\alpha_1-\alpha_2-\dots-\alpha_n,\alpha_2+\dots+\alpha_n}}{N_{\alpha_1+\alpha_2+\dots+\alpha_n,\alpha_2+\dots+\alpha_n}} \frac{N_{-\alpha_1-2\alpha_2-\dots-2\alpha_n,\alpha_2+\dots+\alpha_n}}{N_{\alpha_1,\alpha_2+\dots+\alpha_n}} \end{aligned} \quad (6.55)$$

Now we can write

$$N_{-\alpha_1-\alpha_2-\dots-\alpha_n,\alpha_2+\dots+\alpha_n} = -N_{\alpha_2+\dots+\alpha_n,-\alpha_1-\alpha_2-\dots-\alpha_n}$$

and by Equation 2.21, with $\alpha_2 + \dots + \alpha_n$ as α and $\alpha_1 + \alpha_2 + \dots + \alpha_n$ as β we have

$$N_{-\alpha_1-\alpha_2-\dots-\alpha_n,\alpha_2+\dots+\alpha_n} = -N_{\alpha_1,\alpha_2+\dots+\alpha_n}$$

Then we continue Equation 6.55:

$$\begin{aligned} c_{\theta(\alpha_1),\theta_\Delta} &= -\frac{N_{-\alpha_1-2\alpha_2-\dots-2\alpha_n,\alpha_2+\dots+\alpha_n}}{N_{\alpha_1+\alpha_2+\dots+\alpha_n,\alpha_2+\dots+\alpha_n}} \\ &= \frac{N_{\alpha_2+\dots+\alpha_n,-\alpha_1-2\alpha_2-\dots-2\alpha_n}}{N_{\alpha_1+\alpha_2+\dots+\alpha_n,\alpha_2+\dots+\alpha_n}} \end{aligned} \quad (6.56)$$

Appealing to Equation 2.21 once again, with $\alpha_2 + \dots + \alpha_n$ as α and $-\alpha_1 - 2\alpha_2 - \dots - 2\alpha_n$ as $-\beta$, we have

$$N_{\alpha_2+\dots+\alpha_n,-\alpha_1-2\alpha_2-\dots-2\alpha_n} = N_{\alpha_1+\alpha_2+\dots+\alpha_n,\alpha_2+\dots+\alpha_n}$$

Then

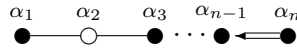
$$c_{\theta(\alpha_1),\theta_\Delta} = 1$$

□

6.9 Identities on the Structure Constants over a Root System of Type C

Consider the Helminck diagram over a root system of type C depicted below. Again, we denote the induced involution by θ .

Figure 6.4: Helminck Diagram for Restricted Rank One Involution of Type 9



The structure constant $c_{\theta(\alpha_2), \theta_\Delta}$ will be of particular interest to us. From Lemma 6.5.1 and Equation 6.32 we have

$$w_0(\alpha_2) = \alpha_1 + \alpha_2 + 2\alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n$$

and hence

$$\theta(\alpha_2) = -\alpha_1 - \alpha_2 - 2\alpha_3 - 2\alpha_4 - \dots - 2\alpha_{n-1} - \alpha_n \quad (6.57)$$

We can show that the value of this structure constant is always 1. To do so, we again appeal to the identities in Section 2.8 and Lemma 4.2.2.

Lemma 6.9.1. *Let θ be an involution induced by the Helminck diagram in Figure 6.4. Then $c_{\theta(\alpha_2), \theta_\Delta} = 1$*

Proof. The smallest case is the condition of there being four roots. We can compute this instance via Algorithm 4.2.3 and obtain 1.

We proceed via induction. Assume the case is so for $n - 1$ roots. For n roots we have

$$\begin{aligned}
c_{\theta(\alpha_2), \theta_\Delta} &= c_{-\alpha_1 - \alpha_2 - 2\alpha_3 - \dots - 2\alpha_{n-1} - \alpha_n} \\
&= c_{\alpha_1 + \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n} \quad rl \\
&= c_{\alpha_1, \theta_\Delta} c_{\alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n, \theta_\Delta} \frac{N_{\theta(\alpha_1), \theta(\alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n)}}{N_{\alpha_1, \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n}}
\end{aligned}$$

Since $\alpha_1 \in \Delta$, then $c_{\alpha_1, \theta_\Delta} = 1$. Then

$$\begin{aligned}
c_{\theta(\alpha_2), \theta_\Delta} &= c_{\alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n, \theta_\Delta} N_{\alpha_1, -\alpha_1 - \alpha_2} N_{\alpha_1, \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n} \\
&= c_{\alpha_3} c_{\alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n, \theta_\Delta} \frac{N_{\alpha_1, -\alpha_1 - \alpha_2}}{N_{\alpha_1, \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n}} \\
&\quad \frac{N_{\theta(\alpha_3), \theta(\alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n)}}{N_{\alpha_3, \alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n}}
\end{aligned}$$

Since $\alpha_3 \in \Delta$, then $c_{\alpha_3, \theta_\Delta} = 1$. By induction hypothesis,

$$c_{\alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n, \theta_\Delta} = 1$$

Then we continue

$$c_{\theta(\alpha_2), \theta_\Delta} = \frac{N_{\alpha_1, -\alpha_1 - \alpha_2}}{N_{\alpha_1, \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n}} \frac{N_{\alpha_3, -\alpha_1 - \alpha_2 - \alpha_3}}{N_{\alpha_3, \alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n}}$$

Via Equation 2.21 we have

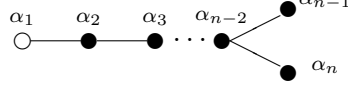
$$\begin{aligned}
c_{\theta(\alpha_2), \theta_\Delta} &= \frac{N_{\alpha_2, \alpha_1}}{N_{\alpha_1, \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n}} \frac{N_{\alpha_1 + \alpha_2, \alpha_3}}{N_{\alpha_3, \alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n}} \\
&= \frac{N_{\alpha_1, \alpha_2}}{N_{\alpha_1, \alpha_2 + 2\alpha_3 + \dots + 2\alpha_{n-1} + \alpha_n}} \frac{N_{\alpha_3, \alpha_1 + \alpha_2}}{N_{\alpha_3, \alpha_2 + \alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-1} + \alpha_n}}
\end{aligned}$$

which gives 1 via Proposition 2.8.9 and Equation 2.19. \square

6.10 Identities on the Structure Constants over a Root System of Type D

Consider the Helminck diagram over a root system of type D depicted below. Denote the induced involution by θ .

Figure 6.5: Helminck Diagram for Restricted Rank One Involution of Type 10



The structure constant $c_{\theta(\alpha_1), \theta_\Delta}$ will be of particular interest to us. From Lemma 6.6.1 we have

$$w_0(\alpha_1) = \alpha_1 + 2\alpha_2 + 2\alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n$$

and hence

$$\theta(\alpha_1) = -\alpha_1 - 2\alpha_2 - 2\alpha_3 - 2\alpha_4 - \dots - 2\alpha_{n-2} - \alpha_{n-1} - \alpha_n \quad (6.58)$$

We can show that the value of this structure constant is always 1. To do so, we again appeal to the identities in Section 2.8 and Lemma 4.2.2.

Lemma 6.10.1. *Let θ be an involution induced by the Helminck diagram in Figure 6.5.*

Then $c_{\theta(\alpha_1), \theta_\Delta} = 1$

Proof.

$$\begin{aligned} c_{\theta(\alpha_1), \theta_\Delta} &= c_{-\alpha_1 - 2\alpha_2 - 2\alpha_3 - 2\alpha_4 - \dots - 2\alpha_{n-2} - \alpha_{n-1} - \alpha_n, \theta_\Delta} \\ &= c_{\alpha_1 + 2\alpha_2 + 2\alpha_3 + 2\alpha_4 + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n, \theta_\Delta} \\ &= c_{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \dots + \alpha_{n-2} + \alpha_{n-1} + \alpha_n, \theta_\Delta} c_{\alpha_2 + \dots + \alpha_{n-2}, \theta_\Delta} \frac{N_{\theta(\alpha_1 + \dots + \alpha_n), \theta(\alpha_2 + \dots + \alpha_{n-2})}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \end{aligned}$$

Since $\theta(\alpha_2 + \dots + \alpha_{n-2}) = \alpha_2 + \dots + \alpha_{n-2}$ then $c_{\alpha_2 + \dots + \alpha_{n-2}, \theta_\Delta} = 1$ Then

$$\begin{aligned} c_{\theta(\alpha_1), \theta_\Delta} &= c_{\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \dots + \alpha_{n-2} + \alpha_{n-1} + \alpha_n, \theta_\Delta} \frac{N_{\theta(\alpha_1 + \dots + \alpha_n), \theta(\alpha_2 + \dots + \alpha_{n-2})}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \\ &= c_{\alpha_1, \theta_\Delta} c_{\alpha_2 + \dots + \alpha_n, \theta_\Delta} \frac{N_{\theta(\alpha_1), \theta(\alpha_2 + \dots + \alpha_n)}}{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}} \frac{N_{\theta(\alpha_1 + \dots + \alpha_n), \theta(\alpha_2 + \dots + \alpha_{n-2})}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \end{aligned}$$

Now $\alpha_1 \in \Delta$, so $c_{\alpha_1, \theta_\Delta} = 1$. Also, $\theta(\alpha_2 + \dots + \alpha_n) = \alpha_2 + \dots + \alpha_n$, so $c_{\alpha_2 + \dots + \alpha_n, \theta_\Delta} = 1$. It then follows that

$$\begin{aligned} c_{\theta(\alpha_1), \theta_\Delta} &= \frac{N_{\theta(\alpha_1), \theta(\alpha_2 + \dots + \alpha_n)}}{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}} \frac{N_{\theta(\alpha_1 + \dots + \alpha_n), \theta(\alpha_2 + \dots + \alpha_{n-2})}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \\ &= \frac{N_{-\alpha_1 - 2\alpha_2 - 2\alpha_3 - \dots - 2\alpha_{n-2} - \alpha_{n-1} - \alpha_n, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}} \frac{N_{-\alpha_1 - \alpha_2 - \dots - \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \\ &= \frac{N_{\alpha_2 + \dots + \alpha_n, -\alpha_1 - 2\alpha_2 - 2\alpha_3 - \dots - 2\alpha_{n-2} - \alpha_{n-1} - \alpha_n}}{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}} \frac{N_{\alpha_2 + \dots + \alpha_{n-2}, -\alpha_1 - \alpha_2 - \dots - \alpha_n}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \end{aligned}$$

With $\alpha_2 + \dots + \alpha_n$ acting as α and $-\alpha_1 - 2\alpha_2 - 2\alpha_3 - \dots - 2\alpha_{n-2} - \alpha_{n-1} - \alpha_n$ as $-\beta$, then via Equation 2.21 we have

$$N_{\alpha_2 + \dots + \alpha_n, -\alpha_1 - 2\alpha_2 - 2\alpha_3 - \dots - 2\alpha_{n-2} - \alpha_{n-1} - \alpha_n} = N_{\alpha_1 + \alpha_2 + \dots + \alpha_{n-2}, \alpha_2 + \dots + \alpha_n}$$

and with $\alpha_2 + \dots + \alpha_{n-2}$ as α and $-\alpha_1 - \dots - \alpha_n$ as $-\beta$ we have

$$N_{\alpha_2 + \dots + \alpha_{n-2}, -\alpha_1 - \alpha_2 - \dots - \alpha_n} = N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-2}}$$

Then

$$c_{\theta(\alpha_1), \theta_\Delta} = \frac{N_{\alpha_1 + \alpha_2 + \dots + \alpha_{n-2}, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}} \frac{N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-2}}}{N_{\alpha_1 + \dots + \alpha_n, \alpha_2 + \dots + \alpha_{n-2}}} \quad (6.59)$$

Now let us consider how we can rewrite the term $N_{\alpha_1, \alpha_2 + \dots + \alpha_n}$. First,

$$N_{\alpha_1, \alpha_2 + \dots + \alpha_n} X_{\alpha_1 + \alpha_2 + \dots + \alpha_n} = [X_{\alpha_1}, X_{\alpha_2 + \dots + \alpha_n}]$$

Let us write X for $X_{\alpha_1 + \alpha_2 + \dots + \alpha_n}$. Hence,

$$N_{\alpha_1, \alpha_2 + \dots + \alpha_n} X = [X_{\alpha_1}, X_{\alpha_2 + \dots + \alpha_n}]$$

Then

$$\frac{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_2 + \dots + \alpha_{n-2}, \alpha_{n-1} + \alpha_n}} X = [X_{\alpha_1}, [X_{\alpha_2 + \dots + \alpha_{n-2}}, X_{\alpha_{n-1} + \alpha_n}]]$$

Following from the Jacobi identity, we have

$$\frac{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_2 + \dots + \alpha_{n-2}, \alpha_{n-1} + \alpha_n}} X = -[X_{\alpha_2 + \dots + \alpha_{n-2}}, [X_{\alpha_{n-1} + \alpha_n}, X_{\alpha_1}]] - [X_{\alpha_{n-1} + \alpha_n}, [X_{\alpha_1}, X_{\alpha_2 + \dots + \alpha_{n-2}}]]$$

But $\alpha_{n-1} + \alpha_n + \alpha_1$ is not a root, so $[X_{\alpha_{n-1} + \alpha_n}, X_{\alpha_1}] = 0$. Then

$$\begin{aligned} \frac{N_{\alpha_1, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_2 + \dots + \alpha_{n-2}, \alpha_{n-1} + \alpha_n}} X &= -[X_{\alpha_{n-1} + \alpha_n}, [X_{\alpha_1}, X_{\alpha_2 + \dots + \alpha_{n-2}}]] \\ &= -N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-2}} [X_{\alpha_{n-1} + \alpha_n}, X_{\alpha_1 + \dots + \alpha_{n-2}}] \\ &= N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-2}} N_{\alpha_{n-1} + \alpha_n, \alpha_1 + \dots + \alpha_{n-2}} X \end{aligned}$$

Hence

$$N_{\alpha_1, \alpha_2 + \dots + \alpha_n} = N_{\alpha_2 + \dots + \alpha_{n-2}, \alpha_{n-1} + \alpha_n} N_{\alpha_1, \alpha_2 + \dots + \alpha_{n-2}} N_{\alpha_{n-1} + \alpha_n, \alpha_1 + \dots + \alpha_{n-2}} \quad (6.60)$$

Next we can rewrite $N_{\alpha_1 + \dots + \alpha_{n-2}, \alpha_2 + \dots + \alpha_n}$ in a similar fashion. Let X denote $X_{\alpha_1 + 2\alpha_2 + \dots + 2\alpha_{n-2} + \alpha_{n-1} + \alpha_n}$. Then

$$N_{\alpha_1 + \dots + \alpha_{n-2}, \alpha_2 + \dots + \alpha_n} X = [X_{\alpha_1 + \dots + \alpha_{n-2}}, X_{\alpha_2 + \dots + \alpha_n}]$$

It follows that

$$\frac{N_{\alpha_1 + \dots + \alpha_{n-2}, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_2 + \dots + \alpha_{n-2}, \alpha_{n-1} + \alpha_n}} X = [X_{\alpha_1 + \dots + \alpha_{n-2}}, [X_{\alpha_2 + \dots + \alpha_{n-2}}, X_{\alpha_{n-1} + \alpha_n}]]$$

From the Jacobi identity we write

$$\begin{aligned} \frac{N_{\alpha_1 + \dots + \alpha_{n-2}, \alpha_2 + \dots + \alpha_n}}{N_{\alpha_2 + \dots + \alpha_{n-2}, \alpha_{n-1} + \alpha_n}} X &= -[X_{\alpha_2 + \dots + \alpha_{n-2}}, [X_{\alpha_{n-1} + \alpha_n}, X_{\alpha_1 + \dots + \alpha_{n-2}}]] \\ &\quad - [X_{\alpha_{n-1} + \alpha_n}, [X_{\alpha_1 + \dots + \alpha_{n-2}}, X_{\alpha_2 + \dots + \alpha_n}]] \end{aligned}$$

which becomes

$$\begin{aligned} \frac{N_{\alpha_1+\dots+\alpha_{n-2}, \alpha_2+\dots+\alpha_n}}{N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_{n-1}+\alpha_n}} X = & -N_{\alpha_{n-1}+\alpha_n, \alpha_1+\dots+\alpha_{n-2}} [X_{\alpha_2+\dots+\alpha_{n-2}}, X_{\alpha_1+\dots+\alpha_n}] \\ & -N_{\alpha_1+\dots+\alpha_{n-2}, \alpha_2+\dots+\alpha_n} [X_{\alpha_{n-1}+\alpha_n}, X_{\alpha_1+2\alpha_2+\dots+2\alpha_{n-2}+\alpha_{n-1}+\alpha_n}] \end{aligned}$$

Since $\alpha_1 + 2\alpha_2 + \dots + 2\alpha_n$ is not a root, then

$$[X_{\alpha_{n-1}+\alpha_n}, X_{\alpha_1+2\alpha_2+\dots+2\alpha_{n-2}+\alpha_{n-1}+\alpha_n}] = 0$$

Then

$$\frac{N_{\alpha_1+\dots+\alpha_{n-2}, \alpha_2+\dots+\alpha_n}}{N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_{n-1}+\alpha_n}} X = -N_{\alpha_{n-1}+\alpha_n, \alpha_1+\dots+\alpha_{n-2}} [X_{\alpha_2+\dots+\alpha_{n-2}}, X_{\alpha_1+\dots+\alpha_n}]$$

Hence

$$\begin{aligned} N_{\alpha_1+\dots+\alpha_{n-2}, \alpha_2+\dots+\alpha_n} X &= -N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_{n-1}+\alpha_n} N_{\alpha_{n-1}+\alpha_n, \alpha_1+\dots+\alpha_{n-2}} \\ &\quad N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_1+\dots+\alpha_n} X \end{aligned}$$

and finally

$$N_{\alpha_1+\dots+\alpha_{n-2}, \alpha_2+\dots+\alpha_n} = N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_{n-1}+\alpha_n} N_{\alpha_{n-1}+\alpha_n, \alpha_1+\dots+\alpha_{n-2}} N_{\alpha_1+\dots+\alpha_n, \alpha_2+\dots+\alpha_{n-2}} \quad (6.61)$$

Using Equations 6.60 and 6.61, we can write Equation 6.59 as

$$\begin{aligned} c_{\theta(\alpha_1), \theta_\Delta} &= \frac{N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_{n-1}+\alpha_n}}{N_{\alpha_2+\dots+\alpha_{n-2}, \alpha_{n-1}+\alpha_n}} \frac{N_{\alpha_{n-1}+\alpha_n, \alpha_1+\dots+\alpha_{n-2}}}{N_{\alpha_1, \alpha_2+\dots+\alpha_{n-2}}} \frac{N_{\alpha_1+\dots+\alpha_n, \alpha_2+\dots+\alpha_{n-2}}}{N_{\alpha_{n-1}+\alpha_n, \alpha_1+\dots+\alpha_{n-2}}} \\ &\quad \times \frac{N_{\alpha_1, \alpha_2+\dots+\alpha_{n-2}}}{N_{\alpha_1+\dots+\alpha_n, \alpha_2+\dots+\alpha_{n-2}}} \\ &= 1 \end{aligned}$$

□

Chapter 7

Classification of the 1-Consistent Involutorial Helminck Diagrams

In Chapter 4 we focused significant attention on the condition of $c_{\alpha, \theta_{\Delta}} = 1$ for all $\alpha \in \Delta$. When working in local symmetric spaces, the condition that θ_{Δ} as in Definition 4.2.1 is quite beneficial. While we left our previous discussion with an algorithm to compute the structure constants, there remains much more to be said. Indeed, it is possible to determine some of the structure constants simply by looking at the configuration of black and white dots in a given Helminck diagram. These structure constants happen to be the ones which we use to determine if θ_{Δ} is an involution.

In order to propose such a scheme for computing the structure constants by way of the Helminck diagram, we need to look closely at several new key ideas. In particular, we will describe how one can “decompose” a diagram into smaller pieces. These pieces represent involutions of restricted rank one. We will then need to describe how to take each individual piece and “glue” them together to form the original involution.

Determining the values of the structure constants this way is a pleasant consequence. The primary motivation, however, is to extend our decomposition and recomposition scheme to involutions on the Lie algebra itself. This will be the subject of Chapter 8. This way we will be able to break apart the long Groebner basis calculation into smaller parts. We can then lift each part, and glue the involutions on each component into the bigger involution on the original algebra.

One additional benefit we will soon see is that we may be spared the necessity

of computing the Chevalley constants. This will arise from the fact that the structure constants do not change as we break the diagrams apart.

7.1 1-Consistency

Let us denote a Helminck Diagram as \mathbb{H} . Let $\mathbb{H}(\theta)$ denote the involution on the roots recovered from \mathbb{H} , and $\mathbb{H}(\theta_\Delta)$ be defined as in definition 4.2.1. Unless otherwise noted, $\mathbb{H}(\theta)$ should be taken to be an involution. The goal of this chapter is to thoroughly examine the cases such that $\mathbb{H}(\theta_\Delta)$ is an involution on \mathfrak{g} . For convenience, we make the following definition.

Definition 7.1.1 (1-Consistent). *Let \mathbb{H} be a Helminck Diagram over a root system with basis Δ . We say \mathbb{H} is 1-consistent if $c_{\theta(\alpha),\theta_\Delta} = 1$ for all $\alpha \in \Delta$.*

Immediately following from the definition we have

1. $\mathbb{H}(\theta_\Delta)$ is an involution if and only if \mathbb{H} is 1-consistent.
2. If \mathbb{H} contains no black dots, then \mathbb{H} is 1-consistent.
3. If \mathbb{H} contains no white dots, then \mathbb{H} is 1-consistent.

Statement 2 is due to Lemma 4.6.1. Statement 3 arises from Lemma 6.1.1. If all the dots are black, then θ fixes every root. Then $c_{\theta(\alpha),\theta_\Delta} = c_{\alpha,\theta_\Delta}$ for all $\alpha \in \Delta$, the basis for the root system. By Definition 4.2.1 we then have $c_{\theta(\alpha),\theta_\Delta} = 1$ for all $\alpha \in \Delta$.

The primary objective for this chapter will be to understand the conditions for which \mathbb{H} is 1-consistent. We will conclude with a classification scheme which can be easily implemented into an algorithm to quickly check if θ_Δ is an involution.

Unless otherwise noted, let $\theta = \mathbb{H}(\theta)$ and $\theta_\Delta = \mathbb{H}(\theta_\Delta)$. As previously stated, the black dots of \mathbb{H} represent those roots which are fixed by θ . The white dots correspond to roots which project to some root in the local symmetric space. Because for the fixed roots α we have $c_{\theta(\alpha),\theta_\Delta} = 1$, we will want to focus our attention on those roots denoted by white dots.

7.2 Restricted Rank One Automorphisms

We now discuss how to build the restricted rank one components of θ . Recall that Φ_θ denotes the set of restricted roots. Then for all λ such that $\frac{1}{2}\lambda \notin \Phi_\theta$, let $\lambda(\Phi)$ denote the set of all roots $\beta \in \Phi$ so that $\pi(\beta)$ is an integral multiple of λ .

All fixed roots are zero multiples of λ . By construction, the restricted rank of $\Phi(\lambda)$ is one. In [5] Helminck gave all the restricted rank one involution diagrams. These are listed in Table C.1. We can view these diagrams as the “fundamental building blocks” of involutorial automorphisms over both the root system and the corresponding lie algebra. This table will be frequently referred to for this reason.

7.3 The Restricted Rank One Decomposition of an Involution

Our first discussion on the topic of restricted rank one automorphisms centers around how one can view an involution in terms of its restricted rank one components. We then do our computations on these components. Then we reconstruct the original involution (which is the subject of the next section).

There are a surprising number of issues involved in this computation. Our aim will be to resolve enough to present a fully working system. The first task is to determine the basis for each restricted rank one component. We only want the roots denoted by white dots. In most cases this means that, say, for a root α_1 colored white, the only root denoted white in restricted rank one component containing α_1 is α_1 itself. However, this is not always so. Therefore, we need a procedure that captures all cases.

Let θ be the usual involution on the root system. For each root α_i not fixed by θ we compute the projection of α_i into the roots of the local symmetric space. Call this root λ_i :

$$\lambda_i = \pi(\alpha_i) = \frac{1}{2}(\alpha - \theta(\alpha))$$

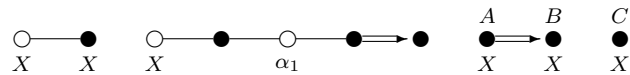
For each root λ_i we compute the set of roots α such that

1. $\pi(\alpha) = k\lambda_i$ for some integer k .
2. α is not fixed by θ .

Call this set Ri . Let Δ_0 denote the set of basis roots fixed by θ . We have

$$Ri = \{\alpha \in (\Delta - \Delta_0) \mid \pi(\alpha) = k\lambda_i, k \in \mathbb{Z}\}$$

Now we merge into Ri the fixed roots joining any $\alpha \in Ri \cap (\Delta - \Delta_0)$. By this we mean the following. Start with α and trace the Helminck diagram through all the black dots. Stop when another white dot is reached, or the end of the irreducible component α resides in is reached. For α_1 in the example diagram below (over $A_2 \times B_5 \times B_2 \times A_1$), the roots which are not in $R1$ are marked with an X .



In the above diagram we repeat this scheme three times, one for each root denoted by white. However, a close study of this example will reveal that the two fixed roots in the far right (composing the B_2 irreducible component) will never be reached.

As we iterate through all the white roots, we want to keep a list of the roots processed. As in the above case, it may be the case that an irreducible component consists entirely of fixed roots. In this situation, the above process would overlook it! When we have processed all white roots, a final step is to inspect the list of all roots which have not been processed. These roots will always be fixed (black). Pick the first unprocessed root. Call it the pivot root. This starts a new Ri set. Merge with this set all the black roots that can be “reached” by tracing along the Helminck diagram starting from pivot root. In the example above we start with A . Root B can be “reached” from A . So $\{A, B\}$ compose one Ri set. There is still one remaining root that has not been processed. Root C is in an Ri set by itself.

The process is summarized as follows.

Algorithm 7.3.1 (Restricted Rank One Decomposition).

Input θ , an involution over the root system with basis Δ .

Output R_i , set of restricted rank one (and zero) subsystems.

1. $\Phi^\dagger = \{\}$

2. **for** every $\alpha_i \in \Delta - \Delta_0$.

$$\lambda_i := \pi(\alpha_i)$$

$$R_i = \{\alpha \in (\Delta - \Delta_0) \mid \pi(\alpha) = k\lambda_i, k \in \mathbb{Z}\}$$

3. **for** every R_i

for every fixed root α which neighbors by any $\beta \in R_i$ or any fixed roots which can be “reached” by tracing from β through another fixed root (or string of fixed roots), compute $R_i = R_i \cup \alpha$.

4. **for** every R_i , $\Phi^\dagger := \Phi^\dagger \cup R_i$

5. **for** every $\alpha_i \in \Delta, \alpha_i \notin \Phi^\dagger$

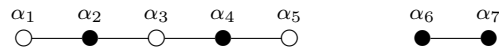
Let $R_i = \{\alpha\}$

for every fixed root β which neighbors α or can be “reached” by tracing from α through another fixed root or string of fixed roots, compute $R_i = R_i \cup \beta$.

6. **return** every R_i .

Example 7.3.2. Restricted rank one decomposition of an involution over Φ

Suppose we have the involution θ induced by the diagram over $A_5 \times A_2$ below.



which has the basis roots

$$\Delta = \{e_1 - e_2, e_2 - e_3, e_3 - e_4, e_4 - e_5, e_5 - e_6, e_7 - e_8, e_8 - e_9\}$$

and α_i is given by the i^{th} entry in Δ .

We compute the projections of the three roots denoted by white dots

$$\lambda_1 = \pi(\alpha_1) = e_1 - \frac{1}{2}e_2 - \frac{1}{2}e_3$$

$$\lambda_2 = \pi(\alpha_3) = \frac{1}{2}e_2 + \frac{1}{2}e_3 - \frac{1}{2}e_4 - \frac{1}{2}e_5$$

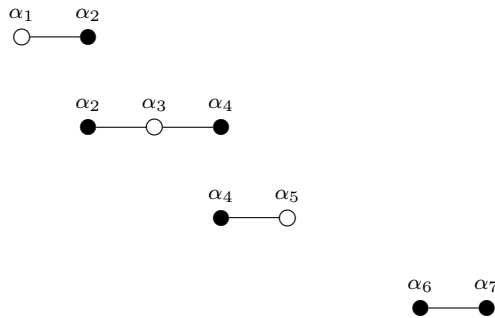
$$\lambda_3 = \pi(\alpha_5) = \frac{1}{2}e_4 + \frac{1}{2}e_5 - e_6$$

Next we iterate through the three λ roots. For λ_1 the set of roots which are not fixed and not perpendicular to λ_1 is $\{\alpha_1\}$. The only fixed root joined to α_1 is α_2 . So our first restricted rank one component $R1$ is $\{\alpha_1, \alpha_2\}$.

The set of roots not fixed and not perpendicular to λ_2 is $\{\alpha_3\}$. There are two fixed roots joined to α_3 , one on each side. Then $R2$ is $\{\alpha_2, \alpha_3, \alpha_4\}$.

Finally, the set of roots not fixed and not perpendicular to λ_3 is $\{\alpha_5\}$. There is only one fixed root joined to α_5 . Then $R3$ is $\{\alpha_4, \alpha_5\}$.

We have only processed α_1 through α_5 . We then choose the first unprocessed root, which is α_6 . This root is joined to α_7 , so we merge them into the same set. After merging all the roots joined to α_6 , there are no unprocessed roots remaining. The restricted rank one decomposition is



7.4 Constructing Involutorial Automorphisms of Higher Rank From Restricted Rank One Automorphisms

In this section we'll demonstrate how to construct an involutorial automorphism on the root system of higher rank from its restricted rank one components. Let θ be an

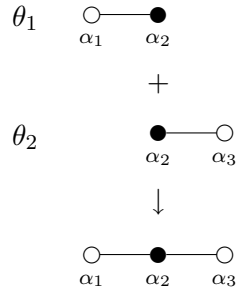
involutorial automorphism on Φ . Suppose we have two roots α_1 and α_2 which are not fixed by θ , and θ is restricted rank two. Let $\lambda_i = \pi(\alpha_i)$. Label the involution induced on $\Phi(\lambda_1)$ by θ_1 . Similarly, let θ_2 be the involution induced on $\Phi(\lambda_2)$. We will show how to “glue” the two involutions together to construct the original θ .

To begin, we make the claim that if a root resides in two different restricted rank-one root systems, then it must be fixed by θ .

Lemma 7.4.1. *If $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$, and $\Phi(\lambda_1) \neq \Phi(\lambda_2)$ then $\theta(\alpha) = \alpha$.*

Proof. We begin with the basis roots. Both $\Phi(\lambda_1)$ and $\Phi(\lambda_2)$ contain all fixed roots (black dots). But they both contain a different white dot, not connected by a diagram automorphism. It follows then, if a basis root is in both systems, it must be represented by a black dot. We have $\forall \delta \in \Delta$, if $\delta \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$ then $\theta(\delta) = \delta$. □

The implication this that the “gluing” process will work as follows. To glue two diagrams together, draw one on top of the other so that the black dots align. Then on the bottom diagram, fill in blank spaces with the white dots that show in the upper diagram. The result looks like this:



Because we’ve seen that two restricted rank-one systems only share fixed roots, then we must have that $\theta_1 = \theta_2$ when the involutions are restricted to the intersection of their two respective systems.

Lemma 7.4.2. *Let θ_1 be the root system induced by the restricted rank-one system $\Phi(\lambda_1)$. Let θ_2 be the root system induced by the restricted rank-one system $\Phi(\lambda_2)$. Then $\theta_1|_{\Phi(\lambda_1) \cap \Phi(\lambda_2)} = \theta_2|_{\Phi(\lambda_1) \cap \Phi(\lambda_2)}$.*

Proof. Pick any $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$. Suppose $\theta_1(\alpha) = \alpha_1$ and $\theta_2(\alpha) = \alpha_2$. Since $\theta(\alpha) = \alpha$ and every $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$, then $\alpha = \alpha_1 = \alpha_2$. \square

In addition, we also have that every root lies in a restricted rank one system. The idea is straightforward. Every root that isn't fixed is contained in its own restricted rank one system.

Lemma 7.4.3. *Let Δ be a basis for Φ . If $\alpha \in \Delta$ then there is a λ_i so that $\alpha \in \Phi(\lambda_i)$.*

Proof. Every fixed root is perpendicular to every λ_i , and hence, a zero multiple. So if α is a fixed root, then $\alpha \in \Phi(\lambda_i)$ for every λ_i . Otherwise, if α is not fixed, then $\alpha \in \Phi(\pi(\alpha))$. \square

Corollary 7.4.4. *Every $\alpha \in \Phi$ can be written as a sum of elements in $\Phi(\lambda_i)$, $i = 1 \dots n$, where n is the number of unique $\Phi(\lambda_i)$ systems. i.e. $\alpha = k_1\alpha_1 + \dots k_s\alpha_s$ where $\alpha_i \in \Phi(\lambda_i)$, $i = 1 \dots n$.*

Proof. This follows immediately from the previous lemma by the linearity of θ . \square

It now makes sense to describe how to “decompose” θ into its restricted rank one components. We write the following for an involution θ of restricted rank k , where θ_i is the involution induced by the Helminck diagram for $\Phi(\lambda_i)$.

$$\theta(\alpha) = \begin{cases} \theta_i(\alpha) & \text{if } \alpha \in \Phi(\lambda_i); \\ \theta_{s_1}(\alpha) = \theta_{s_2}(\alpha) = \dots = \theta_{s_k}(\alpha) & \text{if } \alpha \in \cap_{j=1}^k \Phi(\lambda_{s_j}) \end{cases} \quad (7.1)$$

$\theta(\alpha)$ as defined covers all roots. For roots neither in $\Phi(\lambda_i)$, nor the intersection of any number of restricted systems, we apply the linearity of θ . For example, suppose $\alpha \in \Phi(\lambda_1)$, $\beta \in \Phi(\lambda_2)$, and $\alpha + \beta \notin \Phi(\lambda_1) \cap \Phi(\lambda_2)$. Then $\theta(\alpha + \beta) = \theta(\alpha) + \theta(\beta) = \theta_1(\alpha) + \theta_2(\beta)$.

We refer to the definition of θ in terms of θ_i as the *restricted rank one decomposition* of θ . The restricted rank one decomposition of θ can highlight many important properties of θ itself. In the remaining sections of this chapter we will discuss how to determine the structure constants from the decomposition. In particular, we have for $\alpha \in \Phi(\lambda_i)$,

$$c_{\theta(\alpha), \bar{\theta}} = c_{\theta_i(\alpha), \bar{\theta}}.$$

Relevant to Chapter 8, we have the following claim

Theorem 7.4.5. *Let $\theta_1, \dots, \theta_n$ give the restricted rank one decomposition of θ . Then θ is an involution if and only if θ_i is an involution for all $i = 1 \dots n$.*

Proof. If θ is an involution then $\theta^2(\alpha) = \alpha$ for all $\alpha \in \Phi$. Since $\Phi(\lambda_i) \subset \Phi$ for all i and $\theta(\alpha) = \theta_i(\alpha)$ for all $\alpha \in \Phi(\lambda_i)$, then $\theta_i^2(\alpha) = \alpha$ for all $\alpha \in \Phi(\lambda_i)$.

Now suppose $\theta_i^2 = 1$ for all i . Then for all roots α_i entirely within $\Phi(\lambda_i)$ we have $\theta^2(\alpha_i) = \theta_i^2(\alpha_i) = \alpha_i$.

Let us proceed via an induction argument, and begin with the case $n = 2$. Let $\alpha \in \Phi(\lambda_1)$ and $\beta \in \Phi(\lambda_2)$. Then $\theta^2(\alpha + \beta) = \theta_1^2(\alpha) + \theta_2^2(\beta) = \alpha + \beta$.

Now suppose $\alpha_1 + \dots + \alpha_n \in \Phi$ and $\theta^2(\alpha_1 + \dots + \alpha_n) = \alpha_1 + \dots + \alpha_n$. We have

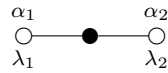
$$\begin{aligned} \theta^2(\alpha_1 + \dots + \alpha_n + \alpha_{n+1}) &= \theta^2(\alpha_1 + \dots + \alpha_n) + \theta^2(\alpha_{n+1}) \\ &= \alpha_1 + \dots + \alpha_n + \alpha_{n+1} \end{aligned}$$

□

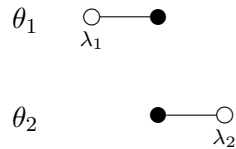
To conclude this section, we'll illustrate the restricted rank one decomposition with a simple example.

Example 7.4.6. *Reconstruction of an involution on the root system by its restricted rank one components*

Let us consider the involution θ on Φ induced by the Helminck diagram below



Where $\lambda_i = \pi(\alpha_i)$. Then $\Phi(\lambda_1)$ and $\Phi(\lambda_2)$ can be represented by the restricted rank-one diagrams, which induce, respectively, involutions θ_1 and θ_2 .



For $\alpha_i \in \Delta$, a basis for Φ , we have

$$\begin{aligned}\theta_1 : \quad & \alpha_1 \rightarrow -\alpha_1 - \alpha_2 \\ & \alpha_2 \rightarrow \alpha_2\end{aligned}$$

$$\begin{aligned}\theta_2 : \quad & \alpha_2 \rightarrow \alpha_2 \\ & \alpha_3 \rightarrow -\alpha_2 - \alpha_3\end{aligned}$$

Define

$$\theta'(\alpha) = \begin{cases} \theta_1(\alpha) & \text{if } \alpha \in \Phi(\lambda_1); \\ \theta_2(\alpha) & \text{if } \alpha \in \Phi(\lambda_2); \\ \theta_1(\alpha) = \theta_2(\alpha) & \text{if } \alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2) \end{cases} \quad (7.2)$$

For the basis roots we have

$$\theta'(\alpha_1) = \theta_1(\alpha_1) = -\alpha_1 - \alpha_2$$

$$\theta'(\alpha_2) = \theta_1(\alpha_2) = \theta_2(\alpha_2) = \alpha_2$$

$$\theta'(\alpha_3) = \theta_2(\alpha_3) = -\alpha_2 - \alpha_3$$

which is precisely the action of θ on the basis roots. By linearity, we have $\theta = \theta'$.

7.5 Involutions With Restricted Rank One

Let θ be an involution with Helminck Diagram \mathbb{H} . In most cases, if \mathbb{H} contains precisely one white dot, then the restricted rank of θ is one. However, if \mathbb{H} designates a diagram automorphism of order 2, then \mathbb{H} may contain two white dots and still designate θ of restricted rank one.

As a brief example, consider the simple case of the root system $A_1 \times A_1$, with basis $\Delta = \{e_1 - e_2, e_3 - e_4\}$. Let $\alpha_1 = e_1 - e_2$ and $\alpha_2 = e_3 - e_4$. Then

$$\lambda_1 = \pi(\alpha_1) = \alpha_1 - \alpha_2 = e_1 - e_2 - e_3 + e_4$$

and

$$\lambda_2 = \pi(\alpha_2) = \alpha_2 - \alpha_1 = e_3 - e_4 - e_1 + e_2$$

Then $\{e_1 - e_2 - e_3 + e_4\}$ gives the basis for Φ_{λ_1} and $\{e_3 - e_4 - e_1 + e_2\}$ gives the basis for Φ_{λ_2} . But as $e_1 - e_2 - e_3 + e_4 = -(e_3 - e_4 - e_1 + e_2)$ then these are really bases of the same system. Hence $\Phi_{\lambda_1} = \Phi_{\lambda_2}$.

Our next step will be to “decompose” a Helminck diagram by computing Φ_{λ_i} for each α_i denoted by a white dot. Each of these restricted rank one systems is denoted by its own Helminck diagram consisting of either one (or in one case, two) white dots. As given by Helminck in [5] and Table C.1, there are only eighteen restricted rank one systems which represent involutions. What we will determine is that a Helminck diagram is 1-consistent if and only if all of its restricted rank one components are. Hence, we can compute 1-consistency via pre-computed table look-ups.

7.6 Restricted Rank and 1-Consistency

Let us begin by formalizing the last statement of the previous section.

Lemma 7.6.1 (Preservation of 1-Consistency via Reduction of Rank). *Let \mathbb{H} be a Helminck diagram defined over a root system Φ . Let λ_i be a restricted root, and $\Phi_{\lambda_i}^\perp$ denote the reduced-rank restricted root system with respect to λ_i . Let \mathbb{H}_{λ_i} be the Helminck diagram for $\Phi_{\lambda_i}^\perp$. Then \mathbb{H} is 1-consistent if and only if \mathbb{H}_{λ_i} is 1-consistent for all $\lambda_i \in \bar{\Delta}$.*

Proof. Because all the fixed roots are perpendicular to λ_i , then all of these roots are present in $\Phi_{\lambda_i}^\perp$. Hence, \mathbb{H} and \mathbb{H}_{λ_i} share the same longest Weyl element $w_0(\theta)$. Reduction of rank preserves symmetry, and so the action of θ^* is preserved for the surviving roots. Hence, the action of θ is preserved.

Clearly reduction of rank does not add new roots to \mathbb{H} , nor are any roots denoted by black dots changed to white dots (or vice-versa). So it remains to show that for a white dot in \mathbb{H} corresponding to α_i , $c_{\theta(\alpha_i), \theta_\Delta} = 1$ for the same root in \mathbb{H}_{λ_i} .

Suppose \mathbb{H} is 1-consistent. Let α_i be a root denoted by a white dot. Then $c_{\theta(\alpha_i), \theta_\Delta} = 1$. Pick a root α_j and reduce rank with respect to this root. Choose $i \neq j$ if possible. If the only option is to eliminate α_i itself, then only fixed roots remain and \mathbb{H}_{λ_i}

is 1-consistent. Otherwise we note θ does not change, and hence the value of $c_{\theta(\alpha_i), \theta_\Delta}$ does not change either. Since this is true for all white roots α_i , then \mathbb{H}_{λ_i} is 1-consistent for all i .

To show the converse, let us assume that $c_{\theta(\alpha_i), \theta_\Delta} \neq 1$ for some α_i . By the same argument as before, the action of θ does not change via reduction of rank. Hence, $c_{\theta(\alpha_i), \theta_\Delta}$ has the same value, and \mathbb{H}_{λ_i} is not 1-consistent. \square

To show the usefulness of this lemma, consider that each reduced restricted rank component is 1-consistent if and only its own reduced restricted rank components are 1-consistent. Hence, we can keep reducing the restricted rank until we break a diagram into restricted rank one components.

Lemma 7.6.2. *Preservation of 1-Consistency via Reduction to Restricted Rank One* Let \mathbb{H} be a Helminck diagram defined over a root system Φ . Let λ_i be a restricted root, and Φ_{λ_i} its restricted rank one root system with respect to λ_i . Let \mathbb{H}_{λ_i} be the Helminck diagram for Φ_{λ_i} . Then \mathbb{H} is 1-consistent if and only if \mathbb{H}_{λ_i} is 1-consistent for all $\lambda_i \in \bar{\Delta}$.

Proof. We proceed by induction over the restricted rank of the involution θ induced by \mathbb{H} . If the restricted rank is one, Lemma 7.6.1 gives us that \mathbb{H} is 1-consistent if and only if \mathbb{H}_{λ_1} is.

Assume the hypothesis is true for restricted rank n . If the restricted rank of θ is $n + 1$, then take all restricted rank n components $\Phi_{\lambda_i}^\downarrow$. Lemma 7.6.1 gives us that \mathbb{H} is 1-consistent if and only if $\mathbb{H}_{\Phi_{\lambda_i}^\downarrow}$ is. By the assumption, $\mathbb{H}_{\Phi_{\lambda_i}^\downarrow}$ is 1-consistent if and only if its restricted rank one components are 1-consistent. \square

In the subsequent section we will classify all the 1-consistent Helminck diagrams. Hence, no Chevalley constants will be needed to compute 1-consistency. First, however, we need to address one issue regarding the classical cases.

Helminck gives a classification of the restricted rank one systems consisting of 18 types, but four represent 1-consistent systems with infinite possibilities for the number of roots. These types are 5, 6, 9, and 10 in Table C.1.

In all four of these cases, the root of interest is the white dot. For the roots denoted by black dots we have $c_{\theta(\alpha), \theta_\Delta} = c_{\alpha, \theta_\Delta}$ and hence, $c_{\theta(\alpha), \theta_\Delta} = 1$.

That the system given in entry 5 of Table C.1 is 1-consistent follows from Lemma 6.7.1.

That the system given in entry 6 of Table C.1 is 1-consistent follows from Lemma 6.8.1.

That the system given in entry 9 of Table C.1 is 1-consistent follows from Lemma 6.9.1.

That the system given in entry 10 of Table C.1 is 1-consistent follows from Lemma 6.10.1.

7.7 A Classification Scheme for 1-Consistent Helminck Diagrams

In an unusual plot twist, justification of a classification scheme for 1-consistent Helminck diagrams is more straightforward in the E, F, and G cases. Because there are a finite number of these diagrams, it is sufficient to simply point to the computed results. For the classical cases, a scheme is necessary. However, there are a large number of E cases - enough that a simple table would prove burdensome. We will provide a scheme for type E in an effort to render implementation less burdensome. For the classical systems, deeper contemplation will be necessary.

We first consider the case of type G. Since there are only four possibilities, it is easy to simply compute a list. The computations are given in Table 7.1, and suffice as a proof via brute force for the following result.

Lemma 7.7.1. *Let \mathbb{H} be a Helminck diagram and $\mathbb{H}(\theta)$ an involution over the root system G_2 . Then \mathbb{H} is 1-consistent if and only if the number of fixed roots is not one.*

Next we will consider all the cases where $\theta^* = \text{id}$. First we consider the root systems of type A. All diagrams have substrings of type A, so we shall hope to extend our results to the other cases. Let \mathbb{H} be a Helminck diagram for a root system of type A. Recall that if we have an A-string of length exceeding 1, then θ is not an involution.

Of the remaining diagrams, we know $c_{\theta(\alpha), \theta_\Delta}$ for α a fixed root (black dot) is 1. This is because $\theta(\alpha) = \alpha$ and so $\theta(\alpha) \in \Delta$, which is 1 by definition 4.2.1.

We turn our attention to the white dots. Whether $c_{\theta(\alpha), \theta_\Delta}$ is +1 or -1 depends on the number of neighboring black dots.

Table 7.1: 1-Consistency States for G2, θ Involution

Diagram	$c_{\theta(\alpha_1), \theta_\Delta}$	$c_{\theta(\alpha_2), \theta_\Delta}$	1-Consistent
$\alpha_1 \quad \alpha_2$ 	1	1	+
$\alpha_1 \quad \alpha_2$ 	-1	1	-
$\alpha_1 \quad \alpha_2$ 	1	-1	-
$\alpha_1 \quad \alpha_2$ 	1	1	+

Lemma 7.7.2. *Let $\theta^* = \text{id}$. Let \mathbb{H} be a Helminck diagram containing an A-string, and let h_i be a dot corresponding to the root α_i .*

1. *If h_i is bordered by one black dot, then $c_{\theta(\alpha_i), \theta_\Delta} = -1$.*
2. *If h_i is bordered by either zero or two black dots, then $c_{\theta(\alpha_i), \theta_\Delta} = 1$.*

Proof. Let $\lambda_i = \pi(\alpha_i)$, and consider the Helminck diagram for the rank-one root system. Call this diagram \mathbb{H}_{λ_i} . If we erase all the black dots which have no bearing on α_i . That is, if $(\alpha_i, \alpha_j) = 0$ then $S_{\alpha_j} \alpha_i = \alpha_i$, and erasing α_j from \mathbb{H}_{λ_i} does not change the action of θ on α_i .

The surviving diagram is a rank-one diagram of type 2 (in case of zero neighboring black dots), type 3 (in case of one neighboring black dot), or type 4 (in case of two neighboring black dots) in Table C.1. If it is of type 3, then as computed, $c_{\theta(\alpha_i), \theta_\Delta} = -1$. Otherwise, we have $c_{\theta(\alpha_i), \theta_\Delta} = 1$. \square

Helminck diagrams with a B-string behave in a slightly peculiar way. The B-string has no effect on its neighbor. However, should a white dot (call it α) be bordering a single black dot on one side, the presence of a B-string on the other side will ensure that $c_{\theta(\alpha), \theta_\Delta} = -1$, since another A-string cannot be placed on the other side to “save” it. Formally, we have as follows:

Lemma 7.7.3. *Let $\theta^* = \text{id}$. Let \mathbb{H} be a Helminck diagram containing a B-string, and let h_i be a dot corresponding to the root α_i . Should h_i be neighboring a B-string of any length, then the B-string has no bearing on the value of $c_{\theta(\alpha_i), \theta_\Delta}$. The structure constant is entirely determined by its other neighbor.*

1. *If h_i is bordered on the other side by one black dot, then $c_{\theta(\alpha_i), \theta_\Delta} = -1$.*

2. *If h_i is bordered on the other side by zero black dots, then $c_{\theta(\alpha_i), \theta_\Delta} = 1$.*

Proof. First we should note that if α_i is bordered on one side by a B-string, then the only possible string that may reside on the other side, if any, is of type A.

Let $\lambda_i = \pi(\alpha_i)$, and consider the Helminck diagram for the rank-one root system. Call the Helminck diagram for the corresponding rank-one root system \mathbb{H}_{λ_i} . As in Lemma 7.7.2, we erase all dots which have no bearing on $c_{\theta(\alpha_i), \theta_\Delta}$. The surviving rank-one root system is of type 6 (in case of zero black dots on the other side), or type 7 (in case of one black dot on the other side) in Table C.1. In the case of type 6, $c_{\theta(\alpha_i), \theta_\Delta} = 1$. In the case of type 7, $c_{\theta(\alpha_i), \theta_\Delta} = -1$. \square

Now let us consider the case of a C-string. It turns out that C-strings act precisely in the same manner as an A-string. The difference is, of course, C-strings may be of any length. However, the same principle as in Lemma 7.7.2 applies. We need only consider the neighbors of a white dot, and in the case of a C-string, we can apply this lemma as if it were an A-string.

Lemma 7.7.4. *Let $\theta^* = \text{id}$. Let \mathbb{H} be a Helminck diagram containing a C-string, and let h_i be a dot corresponding to the root α_i .*

1. *If h_i is bordered by only a C-string of any length, then $c_{\theta(\alpha_i), \theta_\Delta} = -1$.*

2. *If h_i is bordered by both a C-string of any length, and an A-string of length one, then $c_{\theta(\alpha_i), \theta_\Delta} = 1$.*

Proof. We shall construct an argument similar to before. Let $\lambda_i = \pi(\alpha_i)$, and consider the Helminck diagram for the rank-one root system. Call the Helminck diagram for the corresponding rank-one root system \mathbb{H}_{λ_i} . As in Lemma 7.7.2 and 7.7.3, we erase all dots which have no bearing on $c_{\theta(\alpha_i), \theta_\Delta}$.

What remains is a rank-one root system of type 8 (in case of only a C-string neighbor), or type 9 (in case of both a C-string and A-string neighbor) in Table C.1. In the case of type 8 we have $c_{\theta(\alpha_i),\theta_\Delta} = -1$. In the case of type 9 we have $c_{\theta(\alpha_i),\theta_\Delta} = 1$. \square

To wrap up the classical cases, let us consider type D. A D-string behaves in the same manner as a B-string. That is, the D-string has no effect on the value of the neighboring structure constant.

Lemma 7.7.5. *Let $\theta^* = \text{id}$. Let \mathbb{H} be a Helminck diagram containing a D-string, and let h_i be a dot corresponding to the root α_i . Should h_i be neighboring a D-string of any length, then the D-string has no bearing on the value of $c_{\theta(\alpha_i),\theta_\Delta}$. The structure constant is entirely determined by its other neighbor.*

1. *If h_i is bordered on the other side by one black dot, then $c_{\theta(\alpha_i),\theta_\Delta} = -1$.*
2. *If h_i is bordered on the other side by zero black dots, then $c_{\theta(\alpha_i),\theta_\Delta} = 1$.*

Proof. We shall use the rank-one reduction argument again. Let $\lambda_i = \pi(\alpha_i)$, and consider the Helminck diagram for the rank-one root system. Call the Helminck diagram for the corresponding rank-one root system \mathbb{H}_{λ_i} . As in Lemma 7.7.2, 7.7.3, and 7.7.4, we erase all dots which have no bearing on $c_{\theta(\alpha_i),\theta_\Delta}$.

Remaining is a rank-one root system of type 10 (in the case that there are no black dots on the other side), or type 11 (in the case that there is an A-string on the other side) in Table C.1. In the event of type 10, we have $c_{\theta(\alpha_i),\theta_\Delta} = 1$. In the event of type 11 we have $c_{\theta(\alpha_i),\theta_\Delta} = -1$. \square

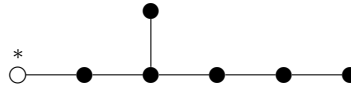
For type F we first point out that there are only three cases amongst the string types to consider. We can not embed a D-string or an E-string. Then we only need to be concerned about an A, B, or C-string. The only case that we do not have an A, B, or C string is in the event of an F-string. But the only way to have an F-string is if there are no white dots. Then we know \mathbb{H} is 1-consistent. Otherwise, we proceed.

The F case is tricky and, in fact, reverses many of the rules we've established in the previous lemmas. As there are only twelve cases where the F diagram is an involution diagram, it is easier to simply provide a list of computed results. However, as a general rule of thumb, the rules for A, B, C, and D strings are reversed. In effect, we only have a

1-consistent Helminck diagram of type F if there is a B-string of length three, or all dots are colored the same. The results are provided in Table 7.2.

To conclude all the cases of Helminck diagrams with $\theta^* = \text{id}$, we need to address the E types. In a Helminck diagram of type E, the only strings are of type A, D, or E.

In most of the cases where we have an A-string or a D-string, we can refer to our previous results. There is one new case, being that which the D-string is “pointing away” from a white dot. Take this to be the case where the white dot is bordering the end of one of the two branches, as illustrated below.



Let α_i be the root corresponding to the white dot “behind” the D-string. In the case that the D-string is of odd length, we do not have an involution diagram. In the case of even length, the structure constant is not one. Note that in the case of a D-string of length four, it is not possible to draw a white dot at the “back” of the string. In this case, the neighboring structure constants are all one.

There is only one case where an E-string may be embedded. If we have a Helminck diagram over the root system E_8 , it is only possible to have an E-string of length seven. Any diagram with an E-string of length six will not be an involution diagram.

Lemma 7.7.6. *Let $\theta^* = \text{id}$. Let \mathbb{H} be a Helminck diagram containing an E-string, and let h_i be a dot corresponding to the root α_i . Then*

1. *Should h_i be neighboring one or two A-strings, then consider Lemma 7.7.2.*
2. *Should h_i be neighboring a D-string, and the D-string is “pointed” toward h_i , then consider Lemma 7.7.5.*
3. *Should h_i be neighboring a D-string, and the D-string is “pointed” away from h_i , then we have $c_{\theta(\alpha_i), \theta_\Delta} = -1$.*
4. *Should h_i be neighboring a E-string of length seven, then $c_{\theta(\alpha_i), \theta_\Delta} = -1$.*

Proof. The first two cases are handled by their respective lemmas. Let $\lambda_i = \pi(\alpha_i)$, and consider the Helminck diagram for the rank-one root system. Call the Helminck diagram

for the corresponding rank-one root system \mathbb{H}_{λ_i} . As in Lemma 7.7.2, 7.7.3, and 7.7.4, we erase all dots which have no bearing on $c_{\theta(\alpha_i), \theta_\Delta}$.

Should we have an embedded D-string of even length, and the root α_i is “behind” the D-string, the remaining rank-one root system is of type 13 in Table C.1. Then we have $c_{\theta(\alpha_i), \theta_\Delta} = -1$.

Should we have an embedded E-string of length seven, the remaining rank-one root system is of type 14 in Table C.1. Then we have $c_{\theta(\alpha_i), \theta_\Delta} = -1$. \square

Next we will consider the case that θ^* is a Dynkin diagram automorphism of order 2. The presence of such θ^* eliminates from consideration diagrams of types B, C, F, G, and $E_{n \neq 6}$. While certain restrictions are no longer imposed (e.g. we may have A-strings of length exceeding one), we require the diagram be symmetric. Certain cases, such as two A-strings of length exceeding one, are still disallowed. In short, if reduction to rank-one does not result in a diagram provided in Table C.1, then we do not have an involution diagram.

Any Helminck diagram that is an involution diagram, has θ^* as assumed, and contains precisely one A-string, is 1-consistent. We see this because reduction to rank one will only result in some diagram of type 5 in Table C.1. Cases where we may have θ^* as assumed, and 1-inconsistent Helminck diagrams do not represent involutions to begin with.

Lemma 7.7.7. *Let θ^* be a Dynkin diagram automorphism of order 2. Let \mathbb{H} be a Helminck diagram containing only an A-string, and let h_i be a dot corresponding to the root α_i . Then $c_{\theta(\alpha_i), \theta_\Delta} = 1$.*

Proof. We use the usual rank-one reduction argument. Let $\lambda_i = \pi(\alpha_i)$, and call the Helminck diagram for the rank-one root system \mathbb{H}_{λ_i} . We erase all dots which have no bearing on $c_{\theta(\alpha_i), \theta_\Delta}$.

Remaining is a rank-one root system of type 5 in Table C.1. Then we have $c_{\theta(\alpha_i), \theta_\Delta} = 1$. \square

In the case of a Helminck diagram of type D, we may have embedded either an A-string or a D-string. In these cases, we actually preserve the properties we had in the case of no diagram automorphism. Nothing more needs to be established, as reduction to rank one results in the same entries in Table C.1. However, we should point out that the

rule determining whether or not we have an involution diagram is “flipped.” In the case of our θ^* , we have an involution diagram if the D-string is of odd length. If the D-string is of even length, we do not have an involution diagram.

We conclude our classification scheme by establishing that, in the case of an E-string, we only have 1-consistency if all roots are colored the same.

Lemma 7.7.8. *Let θ^* be a Dynkin diagram automorphism of order 2. Let \mathbb{H} be an involutorial Helminck diagram of type E_6 . Let h_i be a dot corresponding to the root α_i . Then $c_{\theta(\alpha_i), \theta_\Delta} = 1$ if and only if \mathbb{H} contains no black dots, or no white dots.*

Proof. If \mathbb{H} contains all black or all white dots, then we’ve already established \mathbb{H} is 1-consistent.

As usual, let $\lambda_i = \pi(\alpha_i)$, and consider the Helminck diagram for the rank-one root system. Call the Helminck diagram for the corresponding rank-one root system \mathbb{H}_{λ_i} . Erase all dots which have no bearing on $c_{\theta(\alpha_i), \theta_\Delta}$.

If α_i was in one of the five positions composing the “bottom” string (with respect to how the diagrams are drawn in Table C.1), then we have a rank-one root system of a previously established type. The new case is if α_i is the single “top” dot. In this event, the rank-one root system is of type 12 in Table C.1. Then we have $c_{\theta(\alpha_i), \theta_\Delta} = -1$. \square

We have now covered every irreducible Helminck diagram. We will finish by stating one more claim concerning reducible diagrams. These are diagrams which are constructed from reducible Dynkin diagrams. In the case of a reducible diagram, it is 1-consistent if and only if each irreducible component is.

Theorem 7.7.9. *Let \mathbb{H} be a reducible Helminck diagram with irreducible components $\mathbb{H}_1, \mathbb{H}_2, \dots, \mathbb{H}_n$. Then \mathbb{H} is 1-consistent if and only if \mathbb{H}_i is 1-consistent for all $i = 1 \dots n$.*

Proof. First let us consider the case that there is no diagram automorphism. Then the action of θ is entirely determined by $w_0(\theta)$. If α_i is a root in one irreducible component, and β_i is a root in a different irreducible component, then $(\alpha_i, \beta_i) = 0$. Hence, $S_{\alpha_i}(\beta_i) = \beta_i$ and $S_{\beta_i}(\alpha_i) = \alpha_i$. This means we can “split” $w_0(\theta)$ in the following way: $w_0(\theta) = w_0^\alpha(\theta)w_0^\beta(\theta)$, where $w_0^\alpha(\theta)$ is the longest element of the Weyl group generated by the set of roots corresponding to the first irreducible component, and $w_0^\beta(\theta)$ the longest element with respect to the second component. Therefore, $c_{\theta(\alpha), \theta_\Delta}$ will only consist of structure constants

with respect to the first component, and $c_{\theta(\beta),\theta_\Delta}$ will consist of structure constants with respect to the second component.

In the case of a diagram automorphism between the two components, we remark that the diagram automorphism does not “determine” the value of the structure constant. By this we mean, if $c_{\theta(\alpha),\theta_\Delta} = -1$ and $c_{\theta(\beta),\theta_\Delta} = 1$. and θ^* swaps α and β , then $c_{\theta(\alpha),\theta_\Delta} = 1$ and $c_{\theta(\beta),\theta_\Delta} = -1$. Hence, if for all white dots δ we have $c_{\theta(\delta),\theta_\Delta} = 1$, then swapping two will not change their values. \square

7.8 Computation of the 1-Consistency Property

We will conclude with two a short algorithms. First, we’ll determine if \mathbb{H} is 1-consistent, and therefore, if θ_Δ is an involution on the Lie algebra. Second, we’ll provide an alternate way to compute the structure constants in the case of an involution. This alternative approach will use the classification scheme, and avoid using the Chevalley constants.

A scheme to compute the structure constants can be devised in a variety of ways. We gave one in Section 4.2. See Algorithm 4.2.3. This algorithm is useful if one already has the Chevalley constants. However, if these constants are unknown, we many not want to compute them for the purpose of obtaining the structure constants.

One alternative method is to analyze the pattern of black and white dots in the Helminck diagram by using our classification scheme. We already know the structure constants for the fixed roots are 1. For each basis root corresponding to a white dot we can analyze its neighbors.

A third approach is to use reduction to one of the rank one types in Table C.1. We can obtain the values of the structure constants $c_{\theta(\alpha),\bar{\theta}}$ in the following way.

Algorithm 7.8.1 (Computation of the Structure Constants via Rank One Reduction).

Input Φ , the root system with basis Δ , and θ , involutorial automorphism on the roots.

Output $c_{\theta(\alpha),\bar{\theta}}$ for all $\alpha \in \Delta$.

1. **for** every α_i fixed by θ , let $c_{\theta(\alpha_i),\bar{\theta}} = 1$.

2. **for** every α_i not fixed by θ

Compute $\lambda_i = \pi(\alpha_i)$.

Compute $\Phi(\lambda_i)$.

Use Table C.1 to look up the value of the structure constant corresponding to α_i .

3. **return** $c_{\theta(\alpha_i),\bar{\theta}}$ for all $\alpha_i \in \Delta$.

The algorithm provides an easy way to check for 1-consistency. If it returns all ones, then the corresponding Helminck diagram is 1-consistent.

Algorithm 7.8.2 (Computation of 1-Consistency).

Input \mathbb{H} , Helminck diagram corresponding to θ , the involutorial automorphism on the root system Φ .

Output **TRUE** if \mathbb{H} is 1-consistent. Else, **FALSE** .

1. Call Algorithm 7.8.1 with root system Φ and involution θ .

2. **return** **TRUE** if all ones are returned. Otherwise, return **FALSE** .

Table 7.2: 1-Consistency States for F4, θ Involution

Diagram	$c_{\theta(\alpha_i), \theta_\Delta}$	1-Consistent
α_1 α_2 α_3 α_4 	1 1 1 1	+
α_1 α_2 α_3 α_4 	1 1 -1 1	-
α_1 α_2 α_3 α_4 	1 1 1 -1	-
α_1 α_2 α_3 α_4 	-1 1 -1 1	-
α_1 α_2 α_3 α_4 	-1 1 -1 1	-
α_1 α_2 α_3 α_4 	1 1 1 -1	-
α_1 α_2 α_3 α_4 	-1 1 1 1	-
α_1 α_2 α_3 α_4 	1 -1 1 1	-
α_1 α_2 α_3 α_4 	1 -1 -1 1	-
α_1 α_2 α_3 α_4 	1 -1 1 -1	-
α_1 α_2 α_3 α_4 	1 1 1 1	+
α_1 α_2 α_3 α_4 	1 1 1 1	+

Chapter 8

Classification of the Involutions on The Root System Which Lift

In this chapter we will present a simple classification scheme classifying the involutions which lift.

If θ is an involution on the root system, then θ can be lifted to an involution on the Lie algebra.

This result is a consequence of our primary objective. In a similar manner as our restricted rank one decomposition of θ , we will decompose $\bar{\theta}$, an involution on the Lie algebra. The decomposition works by restricting $\bar{\theta}$ to the torus (and hence, the roots), and projecting the map onto each possible restricted rank one root system.

The goal is to break the problem of lifting an involution on the roots into several pieces. Given θ , we construct restricted rank one involutions θ_i . We lift each θ_i to $\bar{\theta}_i$. Then we “glue” each $\bar{\theta}_i$ together, to give $\bar{\theta}$ in such a way that $\bar{\theta}^2 = 1$ and $\bar{\theta}|_{\mathfrak{t}} = \theta$. Hence, we lift θ to $\bar{\theta}$.

Since computation of Groebner bases can take exponential time, we stand to make significant gains in efficiency. However, this may still not be enough. Some restricted rank one systems can be very large. In particular, types 7, 8, and 11 in Table C.1 are not 1-consistent (need to be lifted), but can be of any length. Hence, they can be of large length and impractical for computation via the “direct” method presented in Chapter 4. We will need to break these maps into even smaller components.

The end result will be a fairly complex, but fast, computation. The “classification

scheme” introduced in this chapter is a result of what will be called the restricted rank one decomposition of $\bar{\theta}$.

8.1 Examining the Lifting Condition

It may be the case that θ_Δ , as defined in Definition 4.2.1, is an involution on the Lie algebra. If not, we will show a correction vector can always be found. However, it will not be entirely useful to think of lifting θ in terms of finding a correction vector. Instead, recall from Section 1.2 that θ can be *lifted* to an automorphism in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$ if there is an automorphism $\bar{\theta} \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ such that $\bar{\theta}|_{\mathfrak{t}} = \theta$.

We will demonstrate how a satisfactory $\bar{\theta}$ can be constructed for any involution θ by considering its restricted rank-one components. This will be done in much the same way that we constructed θ from its restricted rank one components. Proposition 4.3.2 will then guarantee a correction vector can be found, should it be necessary.

8.2 Involutions of Restricted Rank One

By demonstration, it is possible to find correction vectors for all the involutions of restricted rank one of finite size (see Table C.1), should they be necessary. If they are not necessary, take $\bar{\theta} = \theta_\Delta$. Otherwise, take $\bar{\theta} = \theta_\Delta \text{ad}(H)$.

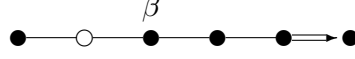
Demonstration satisfies the claim that all restricted rank one involutions can be lifted for most of the types. However, types 7, 8, and 11 are of infinite length, and require correction vectors. For these types we can compute correction vectors for all cases up to $n = 4$. We then proceed by an induction argument to claim that by adding one more fixed root, the involution on the root system still lifts.

Lemma 8.2.1. *If θ is an involution represented by a Helminck diagram of type 7 in Table C.1, then θ lifts to an involutorial automorphism on the Lie algebra.*

Proof. We shall consider an induction argument. A correction vector for θ of the minimum length is given after Table C.1. Consider θ_1 induced by the Helminck diagram over the root system Φ_1 :



and θ induced over the root system Φ by



where θ is constructed by adding one black dot (β) to the diagram of θ_1 . Define $\bar{\theta}$ as follows.

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_1(X_\alpha) & \text{if } \alpha \in \Phi_1; \\ X_\alpha & \text{if } \alpha = \beta. \end{cases} \quad (8.1)$$

Since $\bar{\theta}_1^2(X_\alpha) = X_\alpha$ then $\bar{\theta}^2(X_\alpha) = X_\alpha$ for $\alpha \in \Phi_1$. Also $\bar{\theta}^2(X_\beta) = X_\beta$ since β is fixed by θ . We need to establish $\bar{\theta}^2(X_{\alpha+\beta}) = X_{\alpha+\beta}$.

$$\begin{aligned} \bar{\theta}^2(X_{\alpha+\beta}) &= \frac{1}{N_{\alpha,\beta}} \bar{\theta}^2[X_\alpha, X_\beta] \\ &= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}^2(X_\alpha), \bar{\theta}^2(X_\beta)] \\ &= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} [X_{\theta^2(\alpha)}, X_{\theta^2(\beta)}] \\ &= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} [X_\alpha, X_\beta] \\ &= \frac{N_{\alpha,\beta}}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} \\ &= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} \end{aligned}$$

As $\alpha \in \Phi_1$ then $\bar{\theta}^2(X_\alpha) = \bar{\theta}_1^2(X_\alpha) = c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} X_\alpha = X_\alpha$. Then $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} = 1$

Also, β is fixed by θ , so $c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} = c_{\beta,\bar{\theta}}^2 = 1$.

Then $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} = X_{\alpha+\beta}$ and $\bar{\theta}$ is an involution.

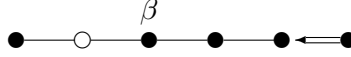
Now let θ_1 be induced by a Helminck diagram of type 7 in C.1 of any length, and induce θ by adding one new black dot β . We construct $\bar{\theta}$ precisely as in Equation 8.1, which is an involution. \square

Lemma 8.2.2. *If θ is an involution represented by a Helminck diagram of type 8 in Table C.1, then θ lifts to an involutorial automorphism on the Lie algebra.*

Proof. We shall again consider an induction argument. A correction vector for θ of the minimum length is given after Table C.1. Consider θ_1 induced by the Helminck diagram over the root system Φ_1 :



and θ induced over the root system Φ by



where θ is constructed by adding one black dot (β) to the diagram of θ_1 . Define $\bar{\theta}$ as follows.

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_1(X_\alpha) & \text{if } \alpha \in \Phi_1; \\ X_\alpha & \text{if } \alpha = \beta. \end{cases} \quad (8.2)$$

Since $\bar{\theta}_1^2(X_\alpha) = X_\alpha$ then $\bar{\theta}^2(X_\alpha) = X_\alpha$ for $\alpha \in \Phi_1$. Also $\bar{\theta}^2(X_\beta) = X_\beta$ since β is fixed by θ . We need to establish $\bar{\theta}^2(X_{\alpha+\beta}) = X_{\alpha+\beta}$.

$$\begin{aligned} \bar{\theta}^2(X_{\alpha+\beta}) &= \frac{1}{N_{\alpha,\beta}} \bar{\theta}^2[X_\alpha, X_\beta] \\ &= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}^2(X_\alpha), \bar{\theta}^2(X_\beta)] \\ &= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} [X_{\theta^2(\alpha)}, X_{\theta^2(\beta)}] \\ &= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} [X_\alpha, X_\beta] \\ &= \frac{N_{\alpha,\beta}}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} \\ &= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} \end{aligned}$$

As $\alpha \in \Phi_1$ then $\bar{\theta}^2(X_\alpha) = \bar{\theta}_1^2(X_\alpha) = c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} X_\alpha = X_\alpha$. Then $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} = 1$

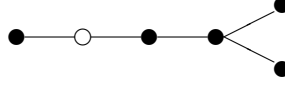
Also, β is fixed by θ , so $c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} = c_{\beta,\bar{\theta}}^2 = 1$.

Then $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} = X_{\alpha+\beta}$ and $\bar{\theta}$ is an involution.

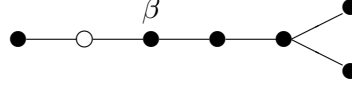
Now let θ_1 be induced by a Helminck diagram of type 8 in C.1 of any length, and induce θ by adding one new black dot β . We construct $\bar{\theta}$ precisely as in Equation 8.2, which is an involution. \square

Lemma 8.2.3. *If θ is an involution represented by a Helminck diagram of type 11 in Table C.1, then θ lifts to an involutorial automorphism on the Lie algebra.*

Proof. We shall once more consider an induction argument. A correction vector for θ of the minimum length is given after Table C.1. Consider θ_1 induced by the Helminck diagram over the root system Φ_1 :



and θ induced over the root system Φ by



where θ is constructed by adding one black dot (β) to the diagram of θ_1 . Define $\bar{\theta}$ as follows.

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_1(X_\alpha) & \text{if } \alpha \in \Phi_1; \\ X_\alpha & \text{if } \alpha = \beta. \end{cases} \quad (8.3)$$

Since $\bar{\theta}_1^2(X_\alpha) = X_\alpha$ then $\bar{\theta}^2(X_\alpha) = X_\alpha$ for $\alpha \in \Phi_1$. Also $\bar{\theta}^2(X_\beta) = X_\beta$ since β is fixed by θ . We need to establish $\bar{\theta}^2(X_{\alpha+\beta}) = X_{\alpha+\beta}$.

$$\begin{aligned}
\bar{\theta}^2(X_{\alpha+\beta}) &= \frac{1}{N_{\alpha,\beta}} \bar{\theta}^2[X_\alpha, X_\beta] \\
&= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}^2(X_\alpha), \bar{\theta}^2(X_\beta)] \\
&= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} [X_{\theta^2(\alpha)}, X_{\theta^2(\beta)}] \\
&= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} [X_\alpha, X_\beta] \\
&= \frac{N_{\alpha,\beta}}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} \\
&= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta}
\end{aligned}$$

As $\alpha \in \Phi_1$ then $\bar{\theta}^2(X_\alpha) = \bar{\theta}_1^2(X_\alpha) = c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} X_\alpha = X_\alpha$. Then $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} = 1$

Also, β is fixed by θ , so $c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} = c_{\beta,\bar{\theta}}^2 = 1$.

Then $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta} = X_{\alpha+\beta}$ and $\bar{\theta}$ is an involution.

Now let θ_1 be induced by a Helminck diagram of type 11 in C.1 of any length, and induce θ by adding one new black dot β . We construct $\bar{\theta}$ precisely as in Equation 8.3, which is an involution. \square

Because the remaining diagrams in Table C.1 are either 1-consistent, or are of finite length, we can now conclude that every restricted rank one involution θ can be lifted.

Lemma 8.2.4. *If θ is an involution of restricted rank one then θ lifts to an involution on the Lie Algebra.*

Proof. For the remaining types not discussed in Lemmas 8.2.1, 8.2.2, and 8.2.3, the Helminck diagram is either of finite length (hence, proof by demonstration), or is 1-consistent (θ_Δ as defined in Definition 4.2.1 is suitable). \square

8.3 An Algorithm for Lifting Involutions of Restricted Rank One

We will now present an algorithm for lifting involutions of restricted rank one. In particular, we only need to focus on those of type 7, 8, or 11 in Table C.1. We make use of

the arguments in the proofs of Lemmas 8.2.1, 8.2.2, and 8.2.3. The motivation will make itself clear as we discuss restricted rank one decomposition in general. Recall our aim is to “build” the involution $\bar{\theta}$ from involutions lifted from the restricted rank one components of θ .

Given θ of restricted rank one and of type 7, 8, or 11 in Table C.1, we first consider the diagram of minimum size (number of roots). For instance, suppose we are given θ of type 7, and of size 6 roots. We start with θ' , an involution described by the Helminck diagram of type 7 and of size 4 roots. This is a smaller system, and can be quickly lifted. Suppose this lifts to $\bar{\theta}'$. We then add one black dot at a time, constructing the involution on the Lie algebra via Equation 8.1.

Our algorithm will make several considerations. We assume at this point the structure constants have been computed. Second, assume our involution on the roots, θ , is of restricted rank one. We know there are 18 possible types. We can use Algorithm 9.3.1 to determine which type we have. As long as the type is not 7, 8, or 11, then we are mostly done. For type not 7, 8, or 11, we may have that θ_Δ , as given in Definition 4.2.1 and with respect to θ is an involution (the corresponding Helminck diagram is 1-consistent). If not, then the number of roots is of finite length, and a suitable involution on the algebra can be quickly computed or pre-computed on some computer algebra system package.

For types 7, 8, or 11, we proceed by starting with the involution of minimum length, and add one fixed root at a time - applying an equation such as Equation 8.1 on each iteration. In the proofs of Lemmas 8.2.1, 8.2.2, and 8.2.3, our new root β was always placed to the right of the one white dot. For consistency and simplicity, we shall always place the new root in this position, and hence can use the same equation for all three types. Our algorithm proceeds as follows.

Algorithm 8.3.1 (Lifting Algorithm for Involutions of Restricted Rank One).

Input θ , an involution of restricted rank one on the root system Φ with basis Δ of n number of roots.

Output $\bar{\theta}$, an involution on the Lie algebra such that $\bar{\theta}|_{\mathfrak{t}} = \theta$ and $\bar{\theta}^2 = 1$

1. Call Algorithm 9.3.1 to determine the type of restricted rank one involution.

2. **if** the corresponding Helminck diagram is 1-consistent, **return** θ_Δ as in Definition 4.2.1.
3. **if** the type is not 7, 8, or 11, call Algorithm 4.5.5 to compute $\bar{\theta}$. **return** $\bar{\theta}$.
4. Remaining is an involution of type 7, 8, or 11. Let $\theta^{(1)}$ be the involution of the same type with the minimum possible size basis, over the subset of Φ denoted $\Phi^{(1)}$.
5. Compute $\bar{\theta}^{(1)}$ by means of Algorithm 4.5.5.
6. **for** $i = 2 \dots n$

Add one new fixed root β to the Helminck diagram for $\theta^{(i-1)}$. For consistency, place this root to the right of the one white dot in the diagram for $\theta^{(i)}$.

Adding one new fixed root in turn adds β to the basis for $\Phi^{(i-1)}$. This gives the root system $\Phi^{(i)}$.

Compute $\bar{\theta}^{(i)}$ via:

$$\bar{\theta}^{(i)}(X_\alpha) = \begin{cases} \bar{\theta}^{(i-1)}(X_\alpha) & \text{if } \alpha \in \Phi^{(i-1)}; \\ X_\alpha & \text{if } \alpha = \beta. \end{cases} \quad (8.4)$$

7. **return** $\bar{\theta}^{(n)}$.

Actually implementing the constructed involution has a tricky point worth exploration. Suppose we have θ_1 an involution on the root system Φ_1 , and that we lifted θ_1 to $\bar{\theta}_1$. Now suppose we constructed $\bar{\theta}$ from $\bar{\theta}_1$. $\bar{\theta}|_{\mathfrak{t}}$ gives θ , an involution over the root system Φ . We have

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_1(X_\alpha) & \text{if } \alpha \in \Phi_1; \\ X_\alpha & \text{if } \alpha = \beta. \end{cases} \quad (8.5)$$

We can easily compute $\bar{\theta}(X_\alpha)$ and $\bar{\theta}(X_\beta)$. What may not be so clear is how to go about computing $\bar{\theta}(X_{\alpha+\beta})$. We need to “split” $X_{\alpha+\beta}$.

A common trick used to “split” root vectors has been to appeal to the identity found in the proof of Lemma 4.2.2:

$$[X_\alpha, X_\beta] = N_{\alpha,\beta} X_{\alpha+\beta}$$

We then have the following.

$$\begin{aligned} \bar{\theta}(X_{\alpha+\beta}) &= \frac{1}{N_{\alpha,\beta}} \bar{\theta}[X_\alpha, X_\beta] \\ &= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}(X_\alpha), \bar{\theta}(X_\beta)] \\ &= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}_1(X_\alpha), X_\beta] \\ &= \frac{1}{N_{\alpha,\beta}} c_{\alpha, \bar{\theta}_1} [X_{\theta_1(\alpha)}, X_\beta] \\ &= \frac{1}{N_{\alpha,\beta}} c_{\alpha, \bar{\theta}} [X_{\theta(\alpha)}, X_\beta] \\ &= \frac{N_{\theta(\alpha), \beta}}{N_{\alpha,\beta}} c_{\alpha, \bar{\theta}} X_{\theta(\alpha+\beta)} \end{aligned}$$

Now recall from Lemma 4.2.2-(2) we have

$$c_{\alpha+\beta, \theta} = \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha,\beta}} c_{\alpha, \theta} c_{\beta, \theta}$$

But the implication of our definition for $\bar{\theta}$ is that $c_{\beta, \theta} = 1$. Also note $\theta(\beta) = \beta$.

Then

$$c_{\alpha+\beta, \theta} = \frac{N_{\theta(\alpha), \beta}}{N_{\alpha,\beta}} c_{\alpha, \theta}$$

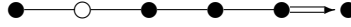
Then we have

$$\bar{\theta}(X_{\alpha+\beta}) = c_{\alpha+\beta, \bar{\theta}} X_{\theta(\alpha+\beta)} \tag{8.6}$$

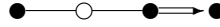
To illustrate the above procedures, let us consider a simple example.

Example 8.3.2. *Constructing an Involution of Restricted Rank One*

Consider the involution θ on the root system Φ induced by the Helminck diagram



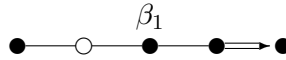
We wish to lift θ to $\bar{\theta} \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$. We first consider the “smaller problem” of lifting $\theta^{(1)}$ induced by the minimum length diagram of type 7 in Table C.1.



We call Algorithm 4.5.5 to lift $\theta^{(1)}$ and obtain $\bar{\theta}^{(1)} = \theta_{\Delta} \text{ad}(x_1 H_1 + x_2 H_2 + x_3 H_3 + x_4 H_4)$ where

$$\begin{aligned} x_1 &= \frac{1}{14} \left(-7 - (5\sqrt{2} + \sqrt{15}) \sqrt{13 - 2\sqrt{30}} \right) \\ x_2 &= -\frac{1}{7} (5\sqrt{2} + \sqrt{15}) \sqrt{13 - 2\sqrt{30}} \\ x_3 &= -\frac{1}{7} (5\sqrt{2} + \sqrt{15}) \sqrt{13 - 2\sqrt{30}} \\ x_4 &= -\sqrt{\frac{1}{2} (13 - 2\sqrt{30})} \end{aligned}$$

Our next task is to add one fixed root at a time to the Helminck diagram for $\theta^{(1)}$ so that the induced involution is our original one. We add a new fixed root β_1 :



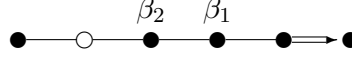
which induces a new involution $\theta^{(2)}$ on the root system. To lift $\theta^{(2)}$ to $\bar{\theta}^{(2)}$ we modify $\bar{\theta}^{(1)}$ as per Equation 8.4. Hence

$$\bar{\theta}^{(2)}(X_{\alpha}) = \begin{cases} \bar{\theta}^{(1)}(X_{\alpha}) & \text{if } \alpha \in \Phi^{(1)}; \\ X_{\alpha} & \text{if } \alpha = \beta_1. \end{cases} \quad (8.7)$$

which gives us

$$\bar{\theta}^{(2)}(X_{\alpha}) = \begin{cases} \theta_{\Delta} \text{ad}(x_1 H_1 + x_2 H_2 + x_3 H_3 + x_4 H_4)(X_{\alpha}) & \text{if } \alpha \in \Phi^{(1)}; \\ X_{\alpha} & \text{if } \alpha = \beta_1. \end{cases} \quad (8.8)$$

Now we add a final black dot to our previous Helminck diagram to obtain the given one. We construct



which induces θ . To lift θ to $\bar{\theta}$ we appeal to Equation 8.8. We have

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}^{(2)}(X_\alpha) & \text{if } \alpha \in \Phi^{(2)}; \\ X_\alpha & \text{if } \alpha = \beta_2. \end{cases} \quad (8.9)$$

which we can write as

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}^{(1)}(X_\alpha) & \text{if } \alpha \in \Phi^{(1)}; \\ X_\alpha & \text{if } \alpha = \beta_1; \\ X_\alpha & \text{if } \alpha = \beta_2. \end{cases} \quad (8.10)$$

and finally

$$\bar{\theta}(X_\alpha) = \begin{cases} \theta_\Delta \operatorname{ad}(x_1 H_1 + x_2 H_2 + x_3 H_3 + x_4 H_4)(X_\alpha) & \text{if } \alpha \in \Phi^{(1)}; \\ X_\alpha & \text{if } \alpha = \beta_1; \\ X_\alpha & \text{if } \alpha = \beta_2. \end{cases} \quad (8.11)$$

8.4 Reduction to Restricted Rank One

Let $\bar{\theta} \in \operatorname{Aut}(\mathfrak{g}, \mathfrak{t})$. We say $\bar{\theta}$ is of restricted rank one if the root system of the algebra is of restricted rank one. We can reduce $\bar{\theta}$ to rank one in a similar manner as we reduced θ . Namely, define

$$\hat{\phi}_\lambda(X_\alpha) = \begin{cases} X_\alpha & \text{if } \alpha \in \Phi(\lambda); \\ 0 & \text{else.} \end{cases} \quad (8.12)$$

and

$$\bar{\theta}_\lambda(X_\alpha) = \begin{cases} \bar{\theta}(X_\alpha) & \text{if } \alpha \in \Phi(\lambda); \\ 0 & \text{else.} \end{cases} \quad (8.13)$$

$\hat{\phi}_\lambda$ projects a Lie algebra to one with the restricted rank-one root system $\Phi(\lambda)$. Call this Lie algebra \mathfrak{g}_λ , with maximal torus \mathfrak{t}_λ . $\bar{\theta}$ describes an automorphism in the restricted rank-one Lie algebra in terms of $\bar{\theta}$. In particular, we preserve the exact action of $\bar{\theta}$ for elements which are in the restricted algebra. Other elements map to zero.

8.5 Constructing Automorphisms for Higher Restricted Ranks

We proceed by demonstrating how, for any Lie algebra, an involution $\bar{\theta}$ can be constructed from involutions of lesser restricted rank. Let \mathbb{H} be a Helminck diagram with white roots α_i , $i = 1 \dots n$. Let $\lambda_i = \pi(\alpha_i)$, and $\Phi(\lambda_i)$ be the restricted rank one root system relative to λ_i .

We denote by $\bar{\theta}|_{\lambda_i}$ the restriction of $\bar{\theta}$ to $\Phi(\lambda_i)$. For convenience, we will write $\bar{\theta}_i$ for $\bar{\theta}|_{\lambda_i}$. We then can decompose $\bar{\theta}$ in a similar way as we decomposed θ . For $X_\alpha \in \mathfrak{g}$, define

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_i(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_i); \\ \bar{\theta}_{s_1}(X_\alpha) = \bar{\theta}_{s_2}(X_\alpha) = \dots = \bar{\theta}_{s_k}(X_\alpha) & \text{if } \alpha \in \cap_{j=1}^k \Phi(\lambda_{s_j}) \end{cases} \quad (8.14)$$

As with Equation 7.1, the linearity of $\bar{\theta}$ handles the cases where root vectors do not fall in one particular restricted system or the intersections of any systems.

We refer to the definition of $\bar{\theta}$ in terms of $\bar{\theta}_i$ as the *restricted rank one decomposition* of $\bar{\theta}$. This construction gives us a very powerful computational tool. In particular, computation of Groebner bases can be very time consuming. Equation 8.14 allows us to perform several smaller computations, then “glue” the results together.

To establish the validity of this construction, we need to make a few claims. First, we need to ensure that “intersecting” structure constants retain the same value.

Lemma 8.5.1. *If $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$, and $\bar{\theta}|_{\Phi(\lambda_1)} = \bar{\theta}_1$, $\bar{\theta}|_{\Phi(\lambda_2)} = \bar{\theta}_2$ then $c_{\alpha, \bar{\theta}_1} = c_{\alpha, \bar{\theta}_2}$.*

Proof. Pick some root $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$. Then

$$\bar{\theta}_1(X_\alpha) = c_{\alpha, \bar{\theta}_1} X_{\theta_1(\alpha)}$$

$$\bar{\theta}_2(X_\alpha) = c_{\alpha, \bar{\theta}_2} X_{\theta_2(\alpha)}$$

Since $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$, then from Equation 7.1 we have

$$c_{\alpha, \bar{\theta}_1} X_{\theta_1(\alpha)} = c_{\alpha, \bar{\theta}_1} X_{\theta(\alpha)}$$

$$c_{\alpha, \bar{\theta}_2} X_{\theta_2(\alpha)} = c_{\alpha, \bar{\theta}_2} X_{\theta(\alpha)}$$

Via restriction of $\bar{\theta}$, we have $\bar{\theta}_1 = \bar{\theta}_2$ for our chosen root. Then

$$c_{\alpha, \bar{\theta}_1} X_{\theta(\alpha)} = c_{\alpha, \bar{\theta}} X_{\theta(\alpha)}$$

$$c_{\alpha, \bar{\theta}_2} X_{\theta(\alpha)} = c_{\alpha, \bar{\theta}} X_{\theta(\alpha)}$$

Hence, $c_{\alpha, \bar{\theta}_1} = c_{\alpha, \bar{\theta}_2}$. □

It then follows that if a root vector X_α resides in the intersection of two restricted systems $\Phi(\lambda_1) \cap \Phi(\lambda_2)$, then $\bar{\theta}_1$ and $\bar{\theta}_2$ act on it in the same way.

Lemma 8.5.2. $\bar{\theta}_1|_{\Phi(\lambda_1) \cap \Phi(\lambda_2)} = \bar{\theta}_2|_{\Phi(\lambda_1) \cap \Phi(\lambda_2)}$.

Proof. Pick some root $\alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2)$, and X_α the corresponding root vector. We have

$$\bar{\theta}_1(X_\alpha) = c_{\alpha, \bar{\theta}_1} X_{\theta_1(\alpha)}$$

$$\bar{\theta}_2(X_\alpha) = c_{\alpha, \bar{\theta}_2} X_{\theta_2(\alpha)}$$

From Equation 7.1, $\theta_1(\alpha) = \theta_2(\alpha)$, Hence,

$$c_{\alpha, \bar{\theta}_1} X_{\theta_1(\alpha)} = c_{\alpha, \bar{\theta}_1} X_{\theta(\alpha)}$$

$$c_{\alpha, \bar{\theta}_2} X_{\theta_2(\alpha)} = c_{\alpha, \bar{\theta}_2} X_{\theta(\alpha)}$$

And from Lemma 8.5.1, $c_{\alpha, \bar{\theta}_1} = c_{\alpha, \bar{\theta}_2}$. Then

$$c_{\alpha, \bar{\theta}_1} X_{\theta(\alpha)} = c_{\alpha, \bar{\theta}_2} X_{\theta(\alpha)}$$

So

$$\bar{\theta}_1(X_\alpha) = \bar{\theta}_2(X_\alpha)$$

□

Remaining is the task to ensure that $\bar{\theta}$, as constructed, is an involution.

Theorem 8.5.3. $\theta \in \text{Aut}(\Phi)$ lifts to an involution $\bar{\theta} \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ if and only if θ_{λ_i} lifts for all $i = 1 \dots n$.

Proof. The \implies direction is less complicated. Suppose θ lifts. Then $\bar{\theta}$ is an involution, and $\bar{\theta}|_{\mathfrak{t}} = \theta$. Note that $\Phi(\lambda) \subset \Phi$. Then since $\bar{\theta}^2(X_\alpha) = X_\alpha$ for all $\alpha \in \Delta$, we have that $\bar{\theta}^2(X_\alpha) = X_\alpha$ for all $\alpha \in \Phi(\lambda)$. Hence, $\bar{\theta}_{\lambda_i}^2(X_\alpha) = X_\alpha$ for all $\alpha \in \Phi(\lambda)$. Also, $\bar{\theta}|_{\mathfrak{t}} = \theta$. For roots in $\Phi(\lambda)$ we have $\theta(\alpha) = \theta_{\lambda_i}(\alpha)$. Hence, $\bar{\theta}_{\lambda_i}|_{\mathfrak{t}_{\lambda_i}} = \theta_{\lambda_i}$. Then θ_{λ_i} lifts.

To show the other direction, let us proceed with an induction argument. To begin, we will demonstrate how to construct $\bar{\theta}$ given two restricted roots λ_1 and λ_2 . Let θ_{λ_1} and θ_{λ_2} lift. Then consider $\bar{\theta}$ as defined

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_{\lambda_1}(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_1); \\ \bar{\theta}_{\lambda_2}(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_2) - \Phi(\lambda_1). \end{cases} \quad (8.15)$$

First,

$$\begin{aligned} \bar{\theta}|_{\mathfrak{t}_{\lambda_1} + \mathfrak{t}_{\lambda_2}} &= \bar{\theta}|_{\mathfrak{t}_{\lambda_1}} + \bar{\theta}|_{\mathfrak{t}_{\lambda_2}} \\ &= \theta_{\lambda_1} + \theta_{\lambda_2} \\ &= \theta \end{aligned}$$

For $\alpha \in \Phi(\lambda_1)$ we have $\bar{\theta}^2(X_\alpha) = \bar{\theta}_{\lambda_1}^2(X_\alpha) = X_\alpha$. We have a similar condition for $\alpha \in \Phi(\lambda_2)$.

Now suppose $\alpha \in \Phi(\lambda_1)$ and $\beta \in \Phi(\lambda_2)$. We have

$$\begin{aligned} \bar{\theta}^2(X_{\alpha+\beta}) &= \frac{1}{N_{\alpha,\beta}} \bar{\theta}^2[X_\alpha, X_\beta] \\ &= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}^2(X_\alpha), \bar{\theta}^2(X_\beta)] \\ &= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} \frac{1}{N_{\alpha,\beta}} [X_{\theta^2(\alpha)}, X_{\theta^2(\beta)}] \end{aligned}$$

Since θ is an involution, we have

$$\begin{aligned}
\bar{\theta}^2(X_{\alpha+\beta}) &= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} \frac{1}{N_{\alpha,\beta}} [X_\alpha, X_\beta] \\
&= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} \frac{N_{\alpha,\beta}}{N_{\alpha,\beta}} X_{\alpha+\beta} \\
&= c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} X_{\alpha+\beta}
\end{aligned}$$

Recall that $\bar{\theta}_{\lambda_1}^2(X_\alpha) = c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} X_\alpha$. Since $\bar{\theta}_{\lambda_1}$ is an involution, we have $c_{\alpha,\bar{\theta}} c_{\theta(\alpha),\bar{\theta}} = 1$. Similarly, we have $c_{\beta,\bar{\theta}} c_{\theta(\beta),\bar{\theta}} = 1$. Then $\bar{\theta}^2(X_{\alpha+\beta}) = X_{\alpha+\beta}$, and $\bar{\theta}$ is an involution.

Now let us assume $\bar{\theta}^2(X_{\alpha_1+\dots+\alpha_n}) = X_{\alpha_1+\dots+\alpha_n}$. Then we have

$$\begin{aligned}
\bar{\theta}^2(X_{\alpha_1+\dots+\alpha_n+\alpha_{n+1}}) &= \frac{1}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} \bar{\theta}^2[X_{\alpha_1+\dots+\alpha_n}, X_{\alpha_{n+1}}] \\
&= \frac{1}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} [\bar{\theta}^2(X_{\alpha_1+\dots+\alpha_n}), \bar{\theta}^2(X_{\alpha_{n+1}})] \\
&= c_{\alpha_1+\dots+\alpha_n,\bar{\theta}} c_{\theta(\alpha_1+\dots+\alpha_n),\bar{\theta}} c_{\alpha_{n+1},\bar{\theta}} c_{\theta(\alpha_{n+1}),\bar{\theta}} \\
&\quad \frac{1}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} [X_{\theta^2(\alpha_1+\dots+\alpha_n)}, X_{\theta^2(\alpha_{n+1})}] \\
&= c_{\alpha_1+\dots+\alpha_n,\bar{\theta}} c_{\theta(\alpha_1+\dots+\alpha_n),\bar{\theta}} c_{\alpha_{n+1},\bar{\theta}} c_{\theta(\alpha_{n+1}),\bar{\theta}} \\
&\quad \frac{1}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} [X_{\alpha_1+\dots+\alpha_n}, X_{\alpha_{n+1}}] \\
&= c_{\alpha_1+\dots+\alpha_n,\bar{\theta}} c_{\theta(\alpha_1+\dots+\alpha_n),\bar{\theta}} c_{\alpha_{n+1},\bar{\theta}} c_{\theta(\alpha_{n+1}),\bar{\theta}} \\
&\quad \frac{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}}{N_{\alpha_1+\dots+\alpha_n,\alpha_{n+1}}} X_{\alpha_1+\dots+\alpha_n+\alpha_{n+1}} \\
&= c_{\alpha_1+\dots+\alpha_n,\bar{\theta}} c_{\theta(\alpha_1+\dots+\alpha_n),\bar{\theta}} c_{\alpha_{n+1},\bar{\theta}} c_{\theta(\alpha_{n+1}),\bar{\theta}} X_{\alpha_1+\dots+\alpha_n+\alpha_{n+1}}
\end{aligned}$$

By assumption, $c_{\alpha_1+\dots+\alpha_n,\bar{\theta}} c_{\theta(\alpha_1+\dots+\alpha_n),\bar{\theta}}$ must equal 1. Since $\bar{\theta}_{\lambda_{n+1}}$ is an involution, we have $c_{\alpha_{n+1},\bar{\theta}} c_{\theta(\alpha_{n+1}),\bar{\theta}} = 1$.

□

Involutions of restricted rank zero can be lifted “by default.” (That is, θ_Δ as defined in Definition 4.2.1 is an involution). Since all involutions on Φ of restricted rank

one can be lifted, and all involutions on Φ of restricted rank greater than one can be broken down into restricted-rank one components, then all involutions on Φ can be lifted.

Corollary 8.5.4. *Let θ be an involutorial automorphism in $\text{Aut}(\Phi)$. Then θ can be lifted to an involutorial automorphism in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$.*

The proof immediately follows from Theorem 8.5.3 and the preceding remarks. As a second corollary, suppose we have two disjoint Helminck diagrams. Then their rank-one decompositions are disjoint. Then follows:

Corollary 8.5.5. *Let θ be an involution on the root system induced from a reducible Helminck diagram $\mathbb{H} = \mathbb{H}_1 + \mathbb{H}_2 + \dots + \mathbb{H}_n$. Let θ_i be an involution on the root system induced by \mathbb{H}_i . Then θ lifts if and only if θ_i lifts for all $i = 1 \dots n$.*

Proof. Suppose θ lifts. Pick $\lambda_i = \pi(\alpha_i)$ where $\alpha_i \in \mathbb{H}_i$. Reduction to rank one isolates θ into an involution on the roots of \mathbb{H}_i . Call this new map θ_i . Then by Lemma 8.5.3, θ_i lifts.

Now suppose every θ_i induced by \mathbb{H}_i lifts. Then every restricted rank one involution θ_{λ_i} lifts. Then θ must lift by Theorem 8.5.3 and Corollary 8.5.4. \square

8.6 The Gluing Mechanism: $\bar{\theta}$ Involution Construction

While Equation 8.14 defines enough to apply it to every root vector in \mathfrak{g} , it may not prove straight-forward for cases where the root of the vector is not in any of the restricted rank one root systems. Indeed, consider $\alpha \in \Phi(\lambda_1)$, and $\beta \in \Phi(\lambda_2)$. Then $X_{\alpha+\beta}$ is in neither $\Phi(\lambda_1)$ nor $\Phi(\lambda_2)$.

A similar issue arises with Equation 7.1, but the linearity of the map handles the case quite nicely. e.g. $\theta(\alpha + \beta) = \theta(\alpha) + \theta(\beta)$. In the case of the involution on the algebra, $\bar{\theta}$, it is not true that $\bar{\theta}(X_{\alpha+\beta}) = \bar{\theta}(X_\alpha) + \bar{\theta}(X_\beta)$.

As with the restricted rank one cases of any possible size, we will again appeal to Lemma 4.2.2 to “split” the root vectors. Recall this is

$$[X_\alpha, X_\beta] = N_{\alpha,\beta} X_{\alpha+\beta}$$

While this is suitable, it requires us to know the Chevalley constants $N_{\alpha,\beta}$. We would like to avoid computing these - especially because it is possible to get around needing them for computing the structure constants.

It is possible to remove the Chevalley constants. Unfortunately, this introduces structure constants the algorithm presented in Chapter 7 is not equipped to produce.

We have, with $\alpha, \beta, X_\alpha, X_\beta, \theta, \bar{\theta}$ as previously mentioned, and $\bar{\theta}_1, \bar{\theta}_2$ components of the restricted rank one decomposition of $\bar{\theta}$:

$$\begin{aligned}
\bar{\theta}(X_{\alpha+\beta}) &= \frac{1}{N_{\alpha,\beta}} \bar{\theta}[X_\alpha, X_\beta] \\
&= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}(X_\alpha), \bar{\theta}(X_\beta)] \\
&= \frac{1}{N_{\alpha,\beta}} [\bar{\theta}_1(X_\alpha), \bar{\theta}_2(X_\beta)] \\
&= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}_1} c_{\beta,\bar{\theta}_2} [X_{\theta_1(\alpha)}, X_{\theta_2(\beta)}] \\
&= \frac{1}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\beta,\bar{\theta}} [X_{\theta(\alpha)}, X_{\theta(\beta)}] \\
&= \frac{N_{\theta(\alpha),\theta(\beta)}}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\beta,\bar{\theta}} X_{\theta(\alpha+\beta)}
\end{aligned}$$

Now recall from Lemma 4.2.2-(2) we have

$$c_{\alpha+\beta,\bar{\theta}} = \frac{N_{\theta(\alpha),\theta(\beta)}}{N_{\alpha,\beta}} c_{\alpha,\bar{\theta}} c_{\beta,\bar{\theta}}$$

Then we have

$$\bar{\theta}(X_{\alpha+\beta}) = c_{\alpha+\beta,\bar{\theta}} X_{\theta(\alpha+\beta)} \tag{8.16}$$

Hence, we must know either the structure constants, or the Chevalley constants.

8.7 Involution Merge

In practice we will know the Chevalley constants. After lifting all restricted rank one involutions (or a single restricted rank one involution of “minimal size” as in Section 8.3) we will want to quickly compute the remaining structure constants corresponding to roots that are a sum of two roots - each residing in a separate restricted component.

An effective means to accomplish this computation is to modify De Graaf's root system algorithm [6] as we did for computing the structure constants in Chapter 4. Because this algorithm begins by constructing the roots of height two, we are guaranteed to be adding two roots which reside completely in their own restricted rank one component (and hence, the values of the structure constants known). We compute all height two roots before moving on to the height threes, and so on. Therefore we are always adding together two roots whose corresponding structure constants are known.

The modified algorithm is as follows.

Algorithm 8.7.1 (Involution Merge).

Input Δ , the basis roots, and $c_{\alpha, \bar{\theta}}$ for all $\alpha \in \Delta$.

Output $c_{\alpha, \theta}$ for all $\alpha \in \Phi$.

1. $n := 1$; $\Phi := \Delta$; $s := 1$.

2. **while** $|\Phi| > s$

$s := |\Phi|$

for every element $\alpha = \sum_{\beta_i \in \Delta} k_i \beta_i \in \Phi$ and every element of $\delta \in \Delta$ **do**

$h := \sum k_i$

if $h = n$ **then**

Compute r , the highest integer so that $\alpha - r\delta$ is in Φ

$q := r - \sum k_i M_{i,j}$, where $M_{i,j}$ is the (i, j) entry in the Cartan Matrix

if $q > 0$ **then**

$\Phi := \Phi \cup (\alpha + \delta)$

$c_{\alpha+\delta, \theta} := \frac{N_{\theta(\delta), \theta(\alpha)}}{N_{\delta, \alpha}} c_{\alpha, \bar{\theta}} c_{\delta, \bar{\theta}}$

$c_{-(\alpha+\delta), \bar{\theta}} := (c_{\alpha+\delta, \bar{\theta}})^{-1}$

3. **return** $c_{\alpha, \bar{\theta}}$ for all $\alpha \in \Phi$

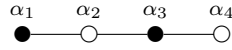
Theorem 8.7.2. Algorithm 8.7.1 computes all $c_{\alpha, \bar{\theta}}$ for all $\alpha \in \Phi$.

Proof. Proof that Algorithm 8.7.1 generates all the roots from the basis roots can be found in [6]. Every time a new root is found, it is found by adding two previously known roots. Hence, by computing the structure constant at the same time, for every new root found, we find its corresponding structure constant. The structure constants corresponding to the basis roots have been previously computed by the restricted rank one lifting algorithms. Because this algorithm computes all the roots, it computes all the structure constants. \square

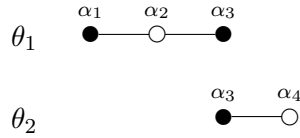
8.8 The Polarity of the Involution $\bar{\theta}$

A nasty little surprise one may encounter is that, after lifting (or applying θ_Δ in the case of 1-consistency) is that the structure constant $c_{\alpha, \bar{\theta}}$ for a shared root α does not turn out to be the same for each component. Take, for example, the following situation.

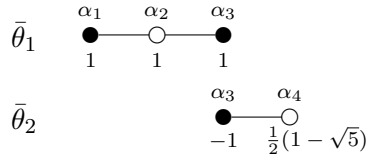
We wish to lift θ induced by the following diagram.



We have the following restricted rank one decomposition of θ into θ_1 and θ_2 :



Now the diagram corresponding to θ_1 is 1-consistent, hence we know immediately $\theta_1 = (\theta_\Delta)_1$. The second diagram requires work. Suppose we lift and obtain the following results (the value of the structure constant $c_{\alpha, \bar{\theta}}$ has been printed below each root α):



The tricky point is recovering the involution $\bar{\theta}$ described in the previous sections. The fix to our gluing scheme is to introduce the concept of the *polarity* of an involution over \mathfrak{g} . Let us begin with the following lemma.

Lemma 8.8.1. *Let $\bar{\theta} \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ be an involution. Let $\alpha \in \Delta$ be a fixed root. Then $c_{\alpha, \bar{\theta}} = \pm 1$.*

Proof. If $\bar{\theta}$ is an involution, then for all $\alpha \in \Delta$ we have $c_{\alpha, \bar{\theta}} c_{\theta(\alpha), \bar{\theta}} = 1$. Since α is fixed, $\theta(\alpha) = \alpha$, so $(c_{\alpha, \bar{\theta}})^2 = 1$. Then $c_{\alpha, \bar{\theta}} = \pm 1$. \square

Fortunately, due to the rank one construction, if a root is shared by two restricted rank one components, it must be fixed by θ . Also, that θ is an involution eliminates any root strings of length exceeding one (except at the very ends of the diagram). Hence, if two restricted rank one components share a common root, it will be a single fixed root at one of the two ends. We define the *polarity* of $\bar{\theta}$ as follows:

Definition 8.8.2. Let Φ be a root system with basis roots $\alpha_1, \dots, \alpha_n$. Then the polarity of $\bar{\theta}$ is the ordered pair (p_1, p_n) where

$$p_i = \begin{cases} 0 & \text{if } \theta(\alpha_i) \neq \alpha_i; \\ + & \text{if } c_{\alpha_i, \bar{\theta}} = 1; \\ - & \text{if } c_{\alpha_i, \bar{\theta}} = -1; \end{cases} \quad (8.17)$$

To join two restricted rank one components, the polarities must align. The polarities are aligned if any of the following conditions are met:

1. The polarities of the shared root are the same.
2. The polarity of the end of one of the components is zero.
3. The restricted rank one components reside in separate irreducible root systems.

In the third case gluing is not an issue at all (this is merely a consideration when implementing an algorithm dealing with polarity). In the second case, the components cannot share a root. This is due to the fact that if the polarity of a root is zero, it cannot be fixed (hence, cannot possibly collide with another component).

In the case of our previous example, we have the following polarities.

$$\begin{array}{ccc} \bar{\theta}_1 & \begin{array}{c} \alpha_1 \quad \alpha_2 \quad \alpha_3 \\ \bullet \text{---} \circ \text{---} \bullet \\ 1 \quad 1 \quad 1 \end{array} & (+, +) \\ \bar{\theta}_2 & \begin{array}{c} \alpha_3 \quad \alpha_4 \\ \bullet \text{---} \circ \\ -1 \quad \frac{1}{2}(1 - \sqrt{5}) \end{array} & (-, 0) \end{array}$$

The solution to our bind is to realize we can “switch” the polarity of an involution. This is done by switching the signs on the structure constants corresponding to the basis roots, then recomputing the subsequent roots. First we need to show that such a construction preserves an involution on the Lie algebra.

Lemma 8.8.3. *Let θ be an involution in $\text{Aut}(\Phi)$. Let $\bar{\theta}_1$ be an involution in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$ so that $\bar{\theta}_1|_{\Phi} = \theta$. Define $\bar{\theta}_2$ so that*

$$\bar{\theta}_2(X_{\alpha}) = -c_{\alpha, \bar{\theta}_1} X_{\theta(\alpha)} \quad (8.18)$$

for all fixed roots $\alpha \in \Delta$, the basis roots of Φ . Then

$$1. \bar{\theta}_2|_{\Phi} = \theta.$$

$$2. \bar{\theta}_2^2 = \text{id}.$$

Proof. Let α be a fixed root. Then we have

$$\begin{aligned} \bar{\theta}_2^2(X_{\alpha}) &= c_{\alpha, \bar{\theta}_2} c_{\theta(\alpha), \bar{\theta}_2} X_{\theta^2(\alpha)} \\ &= c_{\alpha, \bar{\theta}_2}^2 X_{\alpha} \\ &= (-c_{\alpha, \bar{\theta}_1})^2 X_{\alpha} \\ &= X_{\alpha} \end{aligned}$$

Next, suppose α is not fixed. Then

$$\begin{aligned} \bar{\theta}_2^2(X_{\alpha}) &= c_{\alpha, \bar{\theta}_2} c_{\theta(\alpha), \bar{\theta}_2} X_{\theta^2(\alpha)} \\ &= c_{\alpha, \bar{\theta}_1} c_{\theta(\alpha), \bar{\theta}_1} X_{\alpha} \\ &= X_{\alpha} \end{aligned}$$

□

Related to this claim is that if H is a correction vector, then $-H$ must also be one. In addition, using the correction vector $-H$ in place of H accomplishes a polarity switch.

Lemma 8.8.4. *Let $H \in \mathfrak{t}$ be a vector satisfying for all $\alpha \in \Delta$*

$$\alpha(H) \theta(\alpha(H)) c_{\alpha, \theta_\Delta} c_{\theta(\alpha), \theta_\Delta} = 1$$

then $-H$ also satisfies the above condition.

Proof. Write in place of H the vector $-H$. Then we have

$$\alpha(-H) \theta(\alpha(-H)) c_{\alpha, \theta_\Delta} c_{\theta(\alpha), \theta_\Delta}$$

Since both α and θ are linear, we then have

$$(-1)^2 \alpha(H) \theta(\alpha(H)) c_{\alpha, \theta_\Delta} c_{\theta(\alpha), \theta_\Delta}$$

which is 1 because H is a valid correction vector. \square

The natural follow-up question is how to actually compute $\bar{\theta}_2$ given $\bar{\theta}_1$ as described. We would like to avoid as much computation as possible. Thus, we certainly do not want to do any re-lifting. A quick means to accomplish a polarity change is to iterate along the root height. For all the *fixed* basis roots (height 1), we have $c_{\alpha, \bar{\theta}_2} = -c_{\alpha, \bar{\theta}_1}$. Now let us take two height one roots (α and β) and compute the structure constant for a height two root.

$$\begin{aligned} \bar{\theta}_2(X_{\alpha+\beta}) &= \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \bar{\theta}_2} c_{\beta, \bar{\theta}_2} X_{\theta(\alpha+\beta)} \\ &= (-1)^2 \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \bar{\theta}_1} c_{\beta, \bar{\theta}_1} X_{\theta(\alpha+\beta)} \\ &= c_{\alpha+\beta, \bar{\theta}_1} X_{\theta(\alpha+\beta)} \end{aligned}$$

Hence, for α height two, $c_{\alpha, \bar{\theta}_2} = c_{\alpha, \bar{\theta}_1}$. Now let us add a third basis root (γ):

$$\begin{aligned} \bar{\theta}_2(X_{\alpha+\beta+\gamma}) &= \frac{N_{\theta(\alpha+\beta), \theta(\gamma)}}{N_{\alpha+\beta, \gamma}} c_{\alpha+\beta, \bar{\theta}_2} c_{\gamma, \bar{\theta}_2} X_{\theta(\alpha+\beta+\gamma)} \\ &= (-1) \frac{N_{\theta(\alpha+\beta), \theta(\gamma)}}{N_{\alpha+\beta, \gamma}} c_{\alpha+\beta, \bar{\theta}_1} c_{\gamma, \bar{\theta}_1} X_{\theta(\alpha+\beta+\gamma)} \\ &= -c_{\alpha+\beta+\gamma, \bar{\theta}_1} X_{\theta(\alpha+\beta+\gamma)} \end{aligned}$$

Hence, for α height three, $c_{\alpha, \bar{\theta}_2} = -c_{\alpha, \bar{\theta}_1}$. Indeed, the heights alternate.

Lemma 8.8.5. *Let $\bar{\theta}_1$ and $\bar{\theta}_2$ be as in Lemma 8.8.3 and Equation 8.18. Let α be a fixed root. Then all structure constants of $\bar{\theta}_2$ are defined as follows:*

$$c_{\alpha, \bar{\theta}_2} = \begin{cases} c_{\alpha, \bar{\theta}_2} & \text{if the height of } \alpha \text{ is even;} \\ -c_{\alpha, \bar{\theta}_2} & \text{if the height of } \alpha \text{ is odd;} \end{cases} \quad (8.19)$$

Proof. We've demonstrated already that the sign changes as we move from height one to height two roots. Suppose we have a root of height n . We compute the structure constants for roots of height $n+1$. Let α be a fixed root of height n and β be a fixed basis root.

If n is even, then

$$\begin{aligned} \bar{\theta}_2(X_{\alpha+\beta}) &= \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \bar{\theta}_2} c_{\beta, \bar{\theta}_2} X_{\theta(\alpha+\beta)} \\ &= (-1)^{\frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}}} c_{\alpha, \bar{\theta}_1} c_{\beta, \bar{\theta}_1} X_{\theta(\alpha+\beta)} \\ &= c_{\alpha+\beta, \bar{\theta}_1} X_{\theta(\alpha+\beta)} \end{aligned}$$

because $c_{\alpha, \bar{\theta}_2} = c_{\alpha, \bar{\theta}_1}$. If n is odd, then

$$\begin{aligned} \bar{\theta}_2(X_{\alpha+\beta}) &= \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \bar{\theta}_2} c_{\beta, \bar{\theta}_2} X_{\theta(\alpha+\beta)} \\ &= (-1)^2 \frac{N_{\theta(\alpha), \theta(\beta)}}{N_{\alpha, \beta}} c_{\alpha, \bar{\theta}_1} c_{\beta, \bar{\theta}_1} X_{\theta(\alpha+\beta)} \\ &= c_{\alpha+\beta, \bar{\theta}_1} X_{\theta(\alpha+\beta)} \end{aligned}$$

because $c_{\alpha, \bar{\theta}_2} = -c_{\alpha, \bar{\theta}_1}$. □

If α were not a fixed basis root, then there was no sign change. Hence, we have the following claim.

Lemma 8.8.6. *Let α be a root not fixed by θ whose sum decomposition contains no fixed roots. Let $\bar{\theta}_1$ and $\bar{\theta}_2$ be as in Lemma 8.8.3 and Equation 8.18. Then $c_{\alpha, \bar{\theta}_2} = c_{\alpha, \bar{\theta}_1}$.*

Proof. Let $\alpha = \alpha_1 + \alpha_2$ and neither α_1 nor α_2 are fixed. Then

$$\begin{aligned}
c_{\alpha, \bar{\theta}_2} &= \frac{N_{\theta(\alpha_1), \theta(\alpha_2)}}{N_{\alpha_1, \alpha_2}} c_{\alpha_1, \bar{\theta}_2} c_{\alpha_2, \bar{\theta}_2} \\
&= \frac{N_{\theta(\alpha_1), \theta(\alpha_2)}}{N_{\alpha_1, \alpha_2}} c_{\alpha_1, \bar{\theta}_1} c_{\alpha_2, \bar{\theta}_1} \\
&= c_{\alpha, \bar{\theta}_1}
\end{aligned}$$

□

Remains are the roots which are not fixed, and are the sum of a fixed root and non-fixed root. Then the sign of the new structure constant depends on the sign of the fixed root in its sum decomposition.

Lemma 8.8.7. *Let $\alpha = \alpha_1 + \alpha_2$ where α_1 is a fixed root and α_2 is not fixed. Then*

$$c_{\alpha, \bar{\theta}_2} = \begin{cases} c_{\alpha, \bar{\theta}_2} & \text{if the height of } \alpha_1 \text{ is even;} \\ -c_{\alpha, \bar{\theta}_2} & \text{if the height of } \alpha_1 \text{ is odd;} \end{cases} \quad (8.20)$$

Proof. We have, if the height of α_1 is even

$$\begin{aligned}
c_{\alpha, \bar{\theta}_2} &= \frac{N_{\theta(\alpha_1), \theta(\alpha_2)}}{N_{\alpha_1, \alpha_2}} c_{\alpha_1, \bar{\theta}_2} c_{\alpha_2, \bar{\theta}_2} \\
&= \frac{N_{\theta(\alpha_1), \theta(\alpha_2)}}{N_{\alpha_1, \alpha_2}} c_{\alpha_1, \bar{\theta}_1} c_{\alpha_2, \bar{\theta}_1} \\
&= c_{\alpha, \bar{\theta}_1}
\end{aligned}$$

If the height of α_1 is odd, then

$$\begin{aligned}
c_{\alpha, \bar{\theta}_2} &= \frac{N_{\theta(\alpha_1), \theta(\alpha_2)}}{N_{\alpha_1, \alpha_2}} c_{\alpha_1, \bar{\theta}_2} c_{\alpha_2, \bar{\theta}_2} \\
&= -\frac{N_{\theta(\alpha_1), \theta(\alpha_2)}}{N_{\alpha_1, \alpha_2}} c_{\alpha_1, \bar{\theta}_1} c_{\alpha_2, \bar{\theta}_1} \\
&= -c_{\alpha, \bar{\theta}_1}
\end{aligned}$$

□

The following algorithm handles the switch.

Algorithm 8.8.8 (Switching the Polarity of an Involution over the Lie Algebra).

Input $\bar{\theta}$, an involution over the Lie algebra with roots Φ .

Output $\bar{\theta}_2$, an involution with the opposite polarity.

1. **for** every $\alpha \in \Phi$.

if α is fixed,

 Compute H , the height of α .

 Compute $c_{\alpha, \bar{\theta}_2}$ as per Equation 8.20.

if α is not fixed, $\alpha = \alpha_1 + \alpha_2$ where α_1 is fixed

 Compute H , the height of α_1 .

 Compute $c_{\alpha, \bar{\theta}_2}$ as per Equation 8.20.

if α is not fixed, $\alpha = \alpha_1 + \alpha_2$ where neither α_1 nor α_2 is fixed

 Compute $c_{\alpha, \bar{\theta}_2} := c_{\alpha, \bar{\theta}_1}$

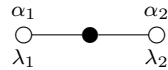
2. **return** $\{c_{\alpha, \bar{\theta}_2}\}$ for all $\alpha \in \Phi$.

8.9 An Illustration of $\bar{\theta}$ Involution Construction

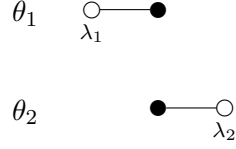
To illustrate the theorems, let us expand Example 7.4.6.

Example 8.9.1. *Reconstruction of an involution on the Lie algebra $\mathfrak{sl}_4(\mathbb{C})$ by its restricted rank one components*

Let $\mathfrak{g} = \mathfrak{sl}_4(\mathbb{C})$. Recall we presented the involution θ on Φ induced by the Helminck diagram below.



which had the following restricted rank one decomposition. Let $\lambda_i = \pi(\alpha_i)$ and $\Phi(\lambda_1)$, $\Phi(\lambda_2)$ be represented by the restricted rank-one diagrams below, inducing, respectively, involutions θ_1 and θ_2 .



We have the involution on the roots given by

$$\theta(\alpha) = \begin{cases} \theta_1(\alpha) & \text{if } \alpha \in \Phi(\lambda_1); \\ \theta_2(\alpha) & \text{if } \alpha \in \Phi(\lambda_2); \\ \theta_1(\alpha) = \theta_2(\alpha) & \text{if } \alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2) \end{cases} \quad (8.21)$$

Similarly, via Equation 8.14, we will define $\bar{\theta}$ as follows.

$$\bar{\theta}(X_\alpha) = \begin{cases} \bar{\theta}_1(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_1); \\ \bar{\theta}_2(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_2); \\ \bar{\theta}_1(X_\alpha) = \bar{\theta}_2(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2) \end{cases} \quad (8.22)$$

Let θ_Δ be as defined in Definition 4.2.1, and $\Delta = \{\alpha_1, \alpha_2, \alpha_3\}$ be a basis for Φ . A basis for $\Phi(\lambda_1)$ is $\{\alpha_1, \alpha_2\}$, and a basis for $\Phi(\lambda_2)$ is $\{\alpha_2, \alpha_3\}$.

Neither Helminck diagram for θ_1 or θ_2 is 1-consistent. So we will need to find correction vectors. Due to the symmetry, we can perform this computation only once, then “flip” the results for the second map.

Consider θ_1 . We have $\bar{\theta}_1 = \theta_\Delta^{(1)} \text{ad}(H_1)$, where $\theta_\Delta^{(1)}$ is the automorphism defined by Definition 4.2.1 with respect to the roots α_1 and α_2 . Via Algorithm 4.5.5 we compute

$$H_1 = \frac{\sqrt{5}}{3}H_{\alpha_1} + \frac{1}{6}(3 + \sqrt{5})H_{\alpha_2}$$

Similarly, let $\theta_\Delta^{(2)}$ be the automorphism defined by Definition 4.2.1 with respect to the roots α_2 and α_3 . We have $\bar{\theta}_2 = \theta_\Delta^{(2)} \text{ad}(H_2)$, where

$$H_2 = \frac{1}{6}(3 + \sqrt{5})H_{\alpha_2} + \frac{\sqrt{5}}{3}H_{\alpha_3}$$

From Equation 8.22, we now have $\bar{\theta}$ defined as follows.

$$\bar{\theta}(X_\alpha) = \begin{cases} \theta_\Delta^{(1)} \operatorname{ad}(\frac{\sqrt{5}}{3}H_{\alpha_1} + \frac{1}{6}(3 + \sqrt{5})H_{\alpha_2})(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_1); \\ \theta_\Delta^{(2)} \operatorname{ad}(\frac{1}{6}(3 + \sqrt{5})H_{\alpha_2} + \frac{\sqrt{5}}{3}H_{\alpha_3})(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_2); \\ \bar{\theta}_1(X_\alpha) = \bar{\theta}_2(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_1) \cap \Phi(\lambda_2) \end{cases} \quad (8.23)$$

Now let us consider the root vectors X_α and torus vectors H_α as given in the example in Section 2.9. Namely, we have

$$H_{\alpha_i} = E_{i,i} - E_{i+1,i+1}$$

and X_α as defined in the matrix M (Equation 2.22).

Let $\{E_{2,3}, E_{1,2}, E_{1,3}, E_{3,2}, E_{2,1}, E_{3,1}\}$ be an ordered basis for $\mathfrak{g}|_{\Phi(\lambda_1)}$. The matrix for $\bar{\theta}_1$ with respect to this basis is

$$[\bar{\theta}_1] = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1 + \sqrt{5}) \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1 - \sqrt{5}) & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{2}(-1 - \sqrt{5}) & 0 & 0 & 0 \\ 0 & \frac{1}{2}(-1 + \sqrt{5}) & 0 & 0 & 0 & 0 \end{pmatrix} \quad (8.24)$$

Let $\{E_{3,4}, E_{2,3}, E_{2,4}, E_{4,3}, E_{3,2}, E_{4,2}\}$ be an ordered basis for $\mathfrak{g}|_{\Phi(\lambda_2)}$. The matrix for $\bar{\theta}_2$ with respect to this basis is

$$[\bar{\theta}_2] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1 + \sqrt{5}) \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1 - \sqrt{5}) & 0 & 0 \\ 0 & 0 & \frac{1}{2}(-1 - \sqrt{5}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ \frac{1}{2}(-1 + \sqrt{5}) & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (8.25)$$

We see the expected behavior at the intersections. Namely, the action of $\bar{\theta}_1$ and $\bar{\theta}_2$ on $E_{2,3}$ and $E_{3,2}$ is the same via both maps. Let an ordered basis for \mathfrak{g} be given by $\{E_{3,4}, E_{2,3}, E_{2,4}, E_{1,2}, E_{1,3}, E_{1,4}, E_{4,3}, E_{3,2}, E_{4,2}, E_{2,1}, E_{3,1}, E_{4,1}\}$.

The action of $\bar{\theta}$ on $E_{i,j}$ where $E_{i,j}$ is in the basis for $\mathfrak{g}|_{\Phi(\lambda_1)}$ or $\mathfrak{g}|_{\Phi(\lambda_2)}$ is clear. For the other vectors we must be more clever. Every root vector corresponding to a basis root is in either $\Phi(\lambda_1)$ or $\Phi(\lambda_2)$, so for any root vector not in $\mathfrak{g}|_{\Phi(\lambda_1)}$ or $\mathfrak{g}|_{\Phi(\lambda_2)}$, its corresponding root is non-simple and can be described as $\alpha + \beta$, where $\alpha \in \Phi(\lambda_1)$ and $\beta \in \Phi(\lambda_2)$.

These root vectors are $E_{1,4}$ and $E_{4,1}$. To describe the roots of these vectors, let $\Delta = \{\alpha_1, \alpha_2, \alpha_3\}$ be a basis for the roots of \mathfrak{g} , where $\alpha_1, \alpha_2 \in \Phi(\lambda_1)$ and $\alpha_2, \alpha_3 \in \Phi(\lambda_2)$. Then

$$X_{\alpha_1+\alpha_2+\alpha_3} = E_{1,4}$$

and

$$X_{-\alpha_1-\alpha_2-\alpha_3} = E_{4,1}$$

We can write $X_{\alpha_1+\alpha_2+\alpha_3} = X_{(\alpha_1+\alpha_2)+(\alpha_3)}$ where $(\alpha_1+\alpha_2) \in \Phi(\lambda_1)$ and $\alpha_3 \in \Phi(\lambda_2)$. Then following from Equation we have

$$\begin{aligned} \bar{\theta}(X_{(\alpha_1+\alpha_2)+(\alpha_3)}) &= c_{\alpha_1+\alpha_2+\alpha_3, \bar{\theta}} X_{\bar{\theta}(\alpha_1+\alpha_2+\alpha_3)} \\ &= (1)X_{-\alpha_1-\alpha_2-\alpha_3} \\ &= E_{4,1} \end{aligned}$$

and similarly

$$\begin{aligned} \bar{\theta}(X_{(-\alpha_1-\alpha_2)+(-\alpha_3)}) &= c_{-\alpha_1-\alpha_2-\alpha_3, \bar{\theta}} X_{\bar{\theta}(-\alpha_1-\alpha_2-\alpha_3)} \\ &= (1)X_{\alpha_1+\alpha_2+\alpha_3} \\ &= E_{1,4} \end{aligned}$$

The matrix for $\bar{\theta}$ with respect to the basis

$\{E_{3,4}, E_{2,3}, E_{2,4}, E_{1,2}, E_{1,3}, E_{1,4}, E_{4,3}, E_{3,2}, E_{4,2}, E_{2,1}, E_{3,1}, E_{4,1}\}$ is

$$[\bar{\theta}] = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & B & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & A & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (8.26)$$

where

$$\begin{aligned} A &= \frac{1}{2} (1 + \sqrt{5}) \\ B &= \frac{1}{2} (1 - \sqrt{5}) \\ C &= \frac{1}{2} (-1 + \sqrt{5}) \\ D &= \frac{1}{2} (-1 - \sqrt{5}) \end{aligned}$$

Then $[\bar{\theta}]^2 = 1$ and $\bar{\theta}$ is an involution.

8.10 An Improved Lifting Algorithm

As previously remarked, Equation 8.14 enables us to build involutions $\bar{\theta}$ from its restricted rank one components. Computation of Groebner bases tends to exponential time, so we stand to make significant improvements over the algorithm described in Chapter 4.

It should be noted that the original algorithm has its place in computing the small cases we'll need to use as "building blocks" for the larger ones. It should also be noted that Algorithm 4.5.3 (Check Lifting Algorithm, Order 2) is also not without merit. It should be able to be easily modified to handle larger orders, where, currently, lifting is not known to be guaranteed. It was also crucial to the research leading up to Corollary 8.5.4. However,

in the spirit of computational efficiency, an improved check lifting algorithm for order 2 should be provided.

Algorithm 8.10.1 (Check Lifting Algorithm, Order 2 (Improved)).

Input θ , an involutorial automorphism in $\text{Aut}(\Phi)$; \mathfrak{g} , a Lie Algebra with roots Φ , having basis roots Δ

Output **TRUE** .

Design of an even faster algorithm is left as an exercise for the reader.

To design an improved lifting algorithm, we will consider $\bar{\theta}$ as decomposed via Equation 8.14. We will compute every $\bar{\theta}_i$ for $\Phi(\lambda_i)$, and “glue” the pieces together. Computation of $\bar{\theta}_i$ will rely on our original lifting algorithm.

An interesting problem does arise, however. In most cases the original lifting algorithm provides multiple solutions. Implementation should keep this in mind. While the gluing aspect can only be done in one way, we may have multiple “building blocks” to pick from.

As a second consideration, recall that in some cases θ_Δ as defined in Definition 4.2.1 is an involution. It will be useful to use θ_Δ whenever possible - and hence, we will want to keep our eyes focused on 1-consistency.

Algorithm 8.10.2 (Lifting Algorithm, Order 2 (Improved)).

Input θ_Δ as in definition 4.2.1 θ , an involutorial automorphism in $\text{Aut}(\Phi)$; \mathfrak{g} , a Lie Algebra with roots Φ , having basis roots Δ

Output $\bar{\theta}$, an involutorial automorphism in $\text{Aut}(\mathfrak{g}, \mathfrak{t})$.

1. Call Algorithm 7.8.1 to compute the structure constants $c_{\alpha, \theta}$ for every $\alpha \in \Phi$
2. **if** $c_{\theta(\alpha), \theta_\Delta} = 1$ for all $\alpha \in \Delta$ **then return** θ_Δ
3. Compute all $\Phi(\lambda_i), i = 1 \dots s$
4. For every $i = 1 \dots s$, call Algorithm 8.3.1 to compute $\bar{\theta}_i$.
5.
$$\bar{\theta}(X_\alpha) := \begin{cases} \bar{\theta}_i(X_\alpha) & \text{if } \alpha \in \Phi(\lambda_i); \\ \bar{\theta}_{s_1}(X_\alpha) = \bar{\theta}_{s_2}(X_\alpha) = \dots = \bar{\theta}_{s_k}(X_\alpha) & \text{if } \alpha \in \cap_{j=1}^k \Phi(\lambda_{s_j}) \end{cases}$$
6. **return** $\bar{\theta}$

Chapter 9

Supporting Algorithms and Notes on Implementation

This section arose during the development of a Mathematica package for computations in local symmetric spaces. Our aim will be to describe and discuss several algorithms which, while not related to the primary theory, prove helpful. The algorithms we will describe are not explicitly called by any previously mentioned scheme, but should be implemented for a “complete system”. The context of the discussion will be in terms of problems that need to be solved during the course of implementation, and their solution.

9.1 Retrieving the Action of an Involution on the Root System From the Helminck Diagram

We shall first discuss issues in retrieving the action of an involution $\theta \in \text{Aut}(\Phi)$ from its Helminck diagram. The theoretical aspect of this problem has already been given by Helminck in [5]. Recall from Equation 4.1 we have

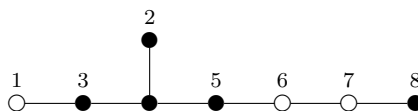
$$\theta = -\text{id} \circ \theta^* \circ w_0(\theta)$$

Computing the action of θ involves four primary steps.

1. Identifying the embedded root systems formed by the fixed roots
2. Computing $w_0(\theta)$

3. Computing the action of $w_0(\theta)$
4. Computing the action of θ^* .

Once the root system is known, a good resource for implementation of step 2 can be found in J. Stembridge's *Coxeter and Weyl Package* [7]. Implementation of steps 3 and 4 are fairly straightforward. On the contrary, implementation of the first step is surprisingly involved computationally. The goal is to describe the involutions with minimal input on the part of the user. Given only the root system θ lives in, the set of fixed roots, and the diagram automorphism, we want to construct the embedded root system. Consider, for example, the Helminck diagram



While we can quickly observe we have the embedded root system $D_4 \times A_1$, the computer, by nature, is not so clever. One means of identification is to look at the basis formed by the fixed roots. By the basis alone we cannot necessarily judge which root system it forms. However, we can construct from the basis the Cartan matrix.

From the Cartan matrix we can determine the root system and type. Reducible root systems form blocks along the diagonal. We can analyze each block to determine the irreducible component. First, though, we must build the matrix from the root basis. The following algorithm can be found in Stembridge's *Coxeter and Weyl Package* [7].

Algorithm 9.1.1 (Construction of the Cartan Matrix from Root System Basis).

Input $\Delta = \{\delta_1, \dots, \delta_n\}$, basis for root system Φ

Output $[a_{i,j}]_{i,j=1\dots n}$, the Cartan matrix.

1. **for** $i, j = 1 \dots n$

$$[a_{i,j}]_{i,j} := \langle \delta_i, \delta_j^\vee \rangle$$

A straightforward way to analyze the Cartan matrix is by elimination. We can count the number of single, double and triple bonds by counting in each row the number of -1 , -2 , or -3 entries. If we have a row with a -3 entry, immediately we have G_2 .

If we have any rows with a -2 entry, we must have a root system of type B, C, or F. We can then determine the exact position of the double bond, and in which direction it is pointing. If the -2 entry is in position $(2, 1)$ or $(n - 1, n)$ then we have a root system of type B. If the -2 entry is in the position $(1, 2)$ or $(n, n - 1)$ then we have a root system of type C. Finally, if the -2 entry is in the position $(2, 3)$, then we have the root system F_4 .

At this point, if we have not determined the root system, then we have eliminated types B, C, F, and G. To distinguish between the remaining cases, we count the number of single bonds (-1 entries) per row. If no row has more than two -1 entries, then we must have a root system of type A. If there are three -1 entries in row 4 (Recall that we're using Humphrey's numbering. Hence, row 4 corresponds to root 4), then we have one of D_6, E_6, E_7, E_8 . If $n = 7$ or 8 then we have E_7 or E_8 respectively. If $n = 6$ then we can use the determinant of the matrix. The determinant of the Cartan matrix for D_6 is 4. The determinant of the Cartan matrix for E_6 is 3. In all other cases, we have a root system of type D.

To summarize these statements, we provide the following algorithm.

Algorithm 9.1.2 (Identification of the Root System By Way of the Cartan Matrix).

Input M , the Cartan matrix for an irreducible root system.

Output Root system name R_n .

1. $n := \#$ of entries in each row of M .
2. **if** M contains a -3 entry **then return** G_2 .
3. **if** M contains a -2 entry **then**
 - if** the position of the -2 entry is $(2, 1)$ or $(n - 1, n)$ **then return** B_n .
 - if** the position of the -2 entry is $(1, 2)$ or $(n, n - 1)$ **then return** C_n .
 - return** F_4 .
4. **if** M contains a row with three -1 entries **then**
 - if** the row is the fourth **and** $n = 6$ **and** $\det(M) = 3$ **then return** E_6 .
 - if** the row is the fourth **and** $n = 7$ **then return** E_7 .
 - if** the row is the fourth **and** $n = 8$ **then return** E_8 .
 - return** D_n
5. **return** A_n .

We call the algorithm for each block in the Cartan matrix. We can then retrieve the action of θ in the following way.

Algorithm 9.1.3 (Constructing the Action of an Involution on $\text{Aut}(\Phi)$ from its Helminck Diagram).

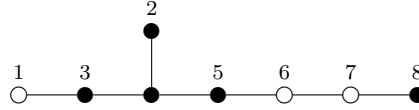
Input \mathbb{H} , Helminck Diagram describing an involution θ over root system Φ with basis Δ

Output θ .

1. Identify by $\beta_1, \beta_2, \dots, \beta_n$ the roots denoted by black dots.
2. Call Algorithm 9.1.1 to compute the Cartan matrix of the root system with basis $\{\beta_1, \dots, \beta_n\}$.
3. For each block M_i in the Cartan matrix, $i = 1 \dots m$, call Algorithm 9.1.2 to determine R_i .
4. $w_0(\theta) := \langle \text{longest element of } R_1 \times R_2 \times \dots \times R_m \rangle$
5. $\theta := -\text{id } \theta^* w_0(\theta)$

Example 9.1.4. Recovering the Involution From its Helminck Diagram

Consider the Helminck diagram below. We wish to recover the action of θ .



The diagram is based on the Dynkin diagram for E_8 . A basis for E_8 , corresponding to the numbering of the diagram, is

$$\Delta = \left\{ \frac{1}{2}(e_1 - e_2 - e_3 - e_4 - e_5 - e_6 - e_7 + e_8), e_1 + e_2, -e_1 + e_2, -e_2 + e_3, -e_3 + e_4, -e_4 + e_5, -e_5 + e_6, -e_6 + e_7 \right\}$$

We then want to look at only the roots denoted by black dots. We remove the basis elements corresponding to the white dots. A basis for the fixed roots is given by

$$\Delta^* = \{e_1 + e_2, -e_1 + e_2, -e_2 + e_3, -e_3 + e_4, -e_6 + e_7\}$$

Now we want to identify the root system formed by this basis. To begin, we call Algorithm 9.1.1 and obtain the Cartan matrix M .

$$M = \begin{pmatrix} 2 & 0 & -1 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 \\ -1 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \quad (9.1)$$

Observe we have two blocks.

$$M_1 = \begin{pmatrix} 2 & 0 & -1 & 0 \\ 0 & 2 & -1 & 0 \\ -1 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \quad (9.2)$$

and

$$M_2 = \begin{pmatrix} 2 \end{pmatrix} \quad (9.3)$$

The matrix M_1 has no -2 entries, hence, no double bonds. However, row 3 has three -1 entries. Recall that row 3 corresponds to root 4 in the Helminck diagram. Hence, we see that the fourth root has three bonds. We must have a system of type D or E. We have four rows, and hence, the embedded root system is either D_4 or E_4 . However, E_4 is not a root system. So we must have D_4 .

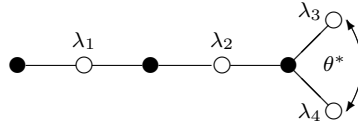
The matrix M_2 has no -2 entries or rows with three -1 entries. So it must give the root system A_1 .

The longest element of D_4 with respect to the roots $\{2, 3, 4, 5\}$ is $s_2 s_3 s_2 s_4 s_3 s_2 s_5 s_3 s_2 s_4 s_3 s_5$. The longest element of A_1 with respect to the root $\{8\}$ is s_8 . Since there is no diagram automorphism, we have

$$\theta = -s_2 s_3 s_2 s_4 s_3 s_2 s_5 s_3 s_2 s_4 s_3 s_5 s_8$$

9.2 Identifying the Type of the Restricted Root System

Another issue that will need attention concerns identifying the root system formed by the restricted roots. Consider, for example, the Helminck diagram for DIIIb, $n = 7$:



We label by white dots the roots which project down to some root λ_i in the local symmetric space. We would like to identify the root system formed by the basis $\Delta^\dagger = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$.

Considering the algorithms we discussed in Section 9.1, this is now a fairly straightforward task. Given a basis for the root system described by the Helminck diagram, we can project the roots denoted by white dots. This gives us a basis for the restricted root system. Algorithms 9.1.1 and 9.1.2 help us do the rest.

To summarize the procedure, we have the following algorithm:

Algorithm 9.2.1 (Identify the Type of the Restricted Root System).

Input Δ , basis for the root system Φ described by the Helminck diagram, and B , the set of fixed roots $\{\beta_1, \dots, \beta_k\}$.

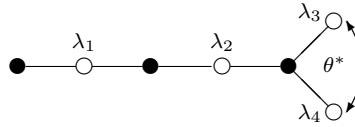
Output Root system name R_n .

1. Compute $\lambda_i := \pi(\beta_i)$ for all $i = 1 \dots k$.
2. $\Delta^\dagger := \{\lambda_1, \dots, \lambda_k\}$.
3. Call Algorithm 9.1.1 with Δ^\dagger to compute the Cartan matrix M .
4. Call Algorithm 9.1.2 with matrix M to compute the root system name R_n .
5. **return** R_n

To illustrate this procedure, we shall provide an example for the aforementioned DIIIb case.

Example 9.2.2. Identifying the Root System Formed by the Restricted Roots of DIIIb, $n = 7$

Recall that we are working with the Helminck diagram



A basis for D_7 is given by

$$\Delta = \{e_1 - e_2, e_2 - e_3, e_3 - e_4, e_4 - e_5, e_5 - e_6, e_6 - e_7, e_6 + e_7\}$$

The roots denoted by black dots are

$$B = \{e_1 - e_2, e_3 - e_4, e_6 - e_7, e_6 + e_7\}$$

Applying the projection π to each element in $\beta_i \in B$ we obtain

$$\Delta^\dagger = \{\frac{1}{2}(e_1 + e_2 - e_3 - e_4), \frac{1}{2}(e_3 + e_4 - e_5 - e_6), \frac{1}{2}(e_6 + e_7)\}$$

Note that the restricted rank of the involution is 3, not 4 as the number of white dots suggests. This is due to the diagram automorphism. Both β_3 and β_4 project to the same root $\frac{1}{2}(e_6 + e_7)$.

We then call Algorithm 9.1.1 and obtain the Cartan matrix M . We obtain

$$M = \begin{pmatrix} 2 & -1 & 0 \\ -2 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \quad (9.4)$$

which Algorithm 9.1.2 identifies as the Cartan matrix for B_3 .

9.3 Identifying the Table Entry in Table C.1

Another problem that may be encountered when working with any algorithms that rely on the table of restricted rank one involutions (Table C.1) is determining a systematic way to identify which of the 18 categories a restricted rank one root system falls into. Because these are the only 18 possibilities, a quick approach is to narrow down the possibilities by identifying the root system.

Types B, C, and D are relatively easy to handle. We can simply look at the position of the white dot. If it is not on the edge of the diagram, there is only one remaining possibility.

Types F and G have a similar solution. We simply determine whether or not the white dot is being “pointed to” by the double or triple bond arrow. Remember, we don’t want to rely on left versus right, as the diagram being mirrored horizontally is technically the same diagram.

There are a couple of tricks to determine if the white dot is being “pointed to.” Recall the arrow in the Dynkin diagram points to the shorter of the two roots. For the G case, we simply determine if the white dot is the longer or shorter of the two roots. For the F case, we can look at the root system formed by the black dots. If we have C_3 , then we have type 15. If we have B_3 , then this give type 16.

There is only one type E case for each of the sizes $n = 6, 7$, or 8 . So determining which E case we have is simply a matter of counting the number of roots in the basis.

Remaining is the A case. If we have a reducible system, we must have Type 1. The only other case with the presence of a diagram automorphism is Type 5. If there is no diagram automorphism, we can count the number of roots in the basis to distinguish between types 2, 3, and 4.

We provide the following algorithm.

Algorithm 9.3.1 (Identifying the Table Entry in Table C.1).

Input α , a root in the system Φ , θ^* , diagram automorphism

Output N , the entry number in C.1

1. Compute $\Phi(\pi(\alpha))$. Denote by $\Delta' = \{\alpha_1, \dots, \alpha_n\}$ the basis of $\Phi(\pi(\alpha))$.
2. Call Algorithms 9.1.1 and 9.1.2 to identify the root system type R_n .
3. **if** $R = A$ **then**
 - if** $\theta^* \neq \text{id}$ **and** Δ' has two members with no bond **then return** Type 1
 - if** $\theta^* = \text{id}$ **and** $n = 1$ **then return** Type 2
 - if** $\theta^* = \text{id}$ **and** $n = 2$ **then return** Type 3
 - if** $\theta^* = \text{id}$ **and** $n = 3$ **then return** Type 4
 - if** $\theta^* \neq \text{id}$ **and** Δ' has at least two members, all bonded **then return** Type 5
4. **if** $R = B$ **then**
 - if** α_1 is not a fixed root, **then return** Type 6
 - else return** Type 7
5. **if** $R = C$ **then**
 - if** α_1 is not a fixed root, **then return** Type 8
 - else return** Type 9
6. **if** $R = D$ **then**

if α_1 *is not a fixed root*, **then return** *Type 10*

else return *Type 11*

7. if $R = E$ **then**

if $n = 6$ **then return** *Type 12*

if $n = 7$ **then return** *Type 13*

if $n = 8$ **then return** *Type 14*

8. if $R = F$ **then**

Let $B = \{\beta_1, \beta_2, \beta_3\}$ *be the basis formed by the fixed roots.*

Call Algorithms 9.1.1 and 9.1.2 to identify the root system type S_3 .

if $S = C$ **then return** *Type 15*

if $S = B$ **then return** *Type 16*

9. if $R = G$ **then**

Let α *denote the shorter of the two roots.*

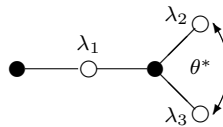
if α *is a fixed root* **then return** *Type 17*

else return *Type 18*

To illustrate the usefulness of this algorithm, we will provide an example which employs the algorithm to compute the structure constant of the root corresponding to the second white dot in the Helminck diagram for DIIIb.

Example 9.3.2. *Computing a Structure Constant Using Table C.1 and Algorithm 9.3.1*

The Helminck diagram for DIIIb, $n = 5$ is given by



We wish to compute the structure constant $c_{\theta(\alpha_2), \bar{\theta}}$ where $\pi(\alpha_2) = \lambda_1$. A basis for D_5 is

$$\Delta = \{e_1 - e_2, e_2 - e_3, e_3 - e_4, e_4 - e_5, e_4 + e_5\}$$

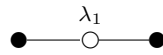
First we have

$$\lambda_1 = \pi(\alpha_2) = \frac{1}{2}(e_1 + e_2 - e_3 - e_4)$$

We compute all the roots in Δ which project to an integral multiple of λ_1 . This gives us the basis elements of $\Phi(\lambda_1)$. These are

$$\Delta' = \{e_1 - e_2, e_2 - e_3, e_3 - e_4\}$$

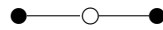
These elements correspond to the component of the Helminck diagram below:



Calling Algorithm 9.1.1 with Δ' , we get the Cartan matrix M .

$$M = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \quad (9.5)$$

which Algorithm 9.1.2 identifies as the Cartan matrix for A_3 . Since we have a root system of type A, and the number of basis elements is 3, then Algorithm 9.3.1 identifies this as Type 4:



Comparing the two diagrams, we see Algorithm 9.3.1 returns the correct entry. Now this diagram is 1-consistent. Recall this means $c_{\theta(\alpha), \bar{\theta}} = 1$ for all roots α in the diagram. Hence, since the white dot corresponds to α_2 , we have $c_{\theta(\alpha_2), \bar{\theta}} = 1$.

9.4 Determining if a Helminck Diagram Describes an Involution on the Roots

Just about every aspect of our work assumes that θ induced by a Helminck diagram is an involution. Unfortunately, our algorithms do not “fail gracefully” if an input θ is not an involution. In fact, our lifting algorithm (in particular, the Groebner-based Algorithm 4.5.5) will go so far as to return a solution. Of course, this correction vector is meaningless, as Proposition 4.3.2 does not even apply if θ is not an involution.

In the name of avoiding any mishaps, it is a meaningful endeavor to devote some attention to checking if θ is an involution to begin with. The straight-forward way is simply to check that each basis root gets mapped back to itself by θ^2 . Depending on the particular computer package in question, this may also be the easiest. This algorithm is as follows.

Algorithm 9.4.1 (Verification That a Helminck Diagram Induces an Involution (Version 1)).

Input \mathbb{H} , a Helminck diagram with respect to root system Φ , inducing $\theta \in \text{Aut}(\Phi)$.

Output **TRUE** if θ is an involution. Otherwise, **FALSE** .

1. Let $\Delta = \{\alpha_1, \dots, \alpha_n\}$ be a basis for Φ
2. **for** $i = 1 \dots n$
 - if** $\theta^2(\alpha_i) \neq \alpha_i$ **then return FALSE**
3. **return TRUE**

The downside to this algorithm is that it involves generating and implementing θ . This is a particular nuisance if one wants to generate a large number of possible involution diagrams - especially over very large root systems. For the purpose of checking conjectures involving a large number of diagrams (amongst other motives), we shall provide a second algorithm relying only upon the diagram itself.

In [5] Helminck gives us several points to consider. First, we have an involution diagram if each fixed root is mapped onto itself. It is true that, for B representing those basis roots fixed by θ , that we have $\theta(B) = B$. However, this does not immediately imply that for every $\beta \in B$ we have $\theta^2(\beta) = \beta$. We must have, for W being the Weyl group of the embedded root system, that $-\text{id} \in W$.

Recall that we have two cases for θ^* , the diagram automorphism:

$$\theta^* = \begin{cases} \text{id}; \\ \langle \text{Dynkin Diagram automorphism of order 2} \rangle \end{cases}$$

In the case of θ^* being the identity, we have a simple set of rules.

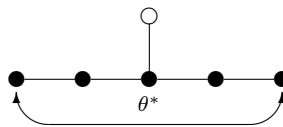
Lemma 9.4.2. *Should a Helminck diagram not violate any of the conditions given, it will induce an involution on the roots.*

1. *There should not be any embedded A-strings of length exceeding one.*
2. *There should not be an embedded D-string of odd length.*

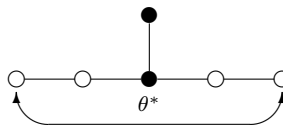
3. *There should not be an embedded E_6 system.*

The conditions should be fairly self-explanatory, and easily implemented on a computer. Unfortunately, in the case of θ^* being a diagram automorphism of order 2, the rules become a bit more complicated. θ^* of order 2 serves to “fix” the diagram should it violate one of the above conditions.

For instance, if θ^* is of order 2, then it is possible to have an A-string of length exceeding one. An example is given below:

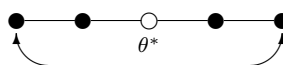


Even though we have an A-string of length 5, θ^* serves to ensure that $\theta^2(\beta) = \beta$ for all fixed roots β . However, one must be cautious checking this condition. θ^* must actually act on the embedded roots in a non-trivial manner. For instance, consider E_6 , with root positions 2 and 4 fixed.



This diagram still does not induce an involution. The problem goes like this: Suppose for now we have θ^* as the identity. Then $\theta^2(\beta) \neq \beta$ for fixed roots β . θ^* being of order 2 does not remedy this, because θ^* acts trivially on the second and fourth roots. With the diagram automorphism represented as a mostly horizontal line (as it is depicted in our two diagrams), the condition to check is that the A-string is “parallel” to θ^* .

Another word of caution is that in a Helminck diagram of type A, If we have θ^* of order 2 and an A-string of length exceeding one, the A-string must be the only one. For instance, the following is not allowed:



The task becomes more involved as we also need to ensure that θ^* is actually a diagram automorphism. The quick check here is to make sure that θ^* maps black dots to black dots, and white to white. We must also ensure the diagram is still drawn the same way. (That is, we must have a diagram that is symmetric with respect to the action of θ^*).

Fortunately, this means we can immediately eliminate all Helminck diagrams of types B, C, F, G, and E (for 7 or 8 basis roots). Since we've already discussed the A types, we need to clear up any issues with D and E_6 .

For the E_6 case we must verify that

1. there are not more than one A-string of length exceeding one
2. the diagram is symmetric

The second rule ensures we can only have an embedded A-string. An embedded D-string (D_4) is ruled out. We can then refer to the rules for the A case. For the D cases, the presence of $\theta^* \neq \text{id}$ means we must have only odd length.

We now introduce the following algorithm.

Algorithm 9.4.3 (Verification That a Helminck Diagram Induces an Involution (Version 2)).

Input \mathbb{H} , a Helminck diagram with respect to root system Φ , inducing $\theta \in \text{Aut}(\Phi)$.

Output **TRUE** if θ is an involution. Otherwise, **FALSE** .

1. **if** $\theta^* = \text{id}$
 - if** there is an embedded A -string of length exceeding one, **return** **FALSE** .
 - if** there is an embedded D -string of odd length, **return** **FALSE** .
 - if** there is an embedded E_6 system, **return** **FALSE** .
2. **else**
 - if** the diagram is not symmetric, **return** **FALSE** .
 - if** there are two A -strings, **return** **FALSE** .
 - if** θ^* acts trivially on the embedded roots (the arrow denoting θ^* is “perpendicular” to the string of black dots), **return** **FALSE** .
 - if** there is an embedded D -string of even length, **return** **FALSE** .
3. **return** **TRUE**

9.5 Representation of Involutions on the Lie Algebra

A result found in Helminck’s paper [5] is that the structure constants $c_{\alpha, \bar{\theta}}$ completely determine the action of $\bar{\theta}$ on the root vectors of a Lie algebra \mathfrak{g} . Thanks to the involution merge algorithm (Algorithm 8.7.1), if the Chevalley constants are known, then we can build from the basis structure constants $\{c_{\alpha, \bar{\theta}} \mid \alpha \in \Delta\}$ the structure constants pertaining to the other roots.

Hence, the minimal representation of $\bar{\theta}$ is

$$\bar{\theta} \cong \{c_{\alpha, \bar{\theta}} \mid \alpha \in \Delta\}$$

where Δ is the basis of the root system Φ .

In practice, however, the involution merge algorithm may be too slow to call whenever we need to know *all* the structure constants. It may very well be more practical to store all the structure constants. However, if we wish to manipulate $\bar{\theta}$ in any way, it is sufficient to manipulate only the structure constants pertaining to the roots of Δ , then re-compute the remaining roots when our computation has finished.

A good application is the following. Suppose we start with θ_Δ , the automorphism according to Definition 4.2.1. Recall that

$$\theta_\Delta \cong \{c_{\alpha, \theta_\Delta} = 1 \mid \alpha \in \Delta\}$$

and we can compute the structure constants for the remaining roots using the involution merge algorithm. Now we wish to modify θ_Δ with our correction vector H so that it is an involution. We have for all basis roots α

$$\bar{\theta}(X_\alpha) = \theta_\Delta \operatorname{ad}(H)(X_\alpha)$$

$$= \alpha(H)\theta_\Delta(X_\alpha)$$

Then we compute $c_{\alpha, \bar{\theta}} = \alpha(H)c_{\alpha, \theta_\Delta}$ for all $\alpha \in \Delta$. The remaining structure constants are computed via the involution merge algorithm, and stored in memory for future use.

Chapter 10

Summary of Conclusions and Future Work

We have now completed our discussion. The chapters that follow gives a manual and source code for an implementation of all the algorithms described, as well as supporting code. To date it is not a complete package for computation in local symmetric spaces, but provides a foundation. The appendix gives referenced tables (in particular, tables of important Helminck diagrams we referred to).

We'll conclude with a brief summary of conclusions, and a description of future work to be done.

10.1 Conclusions

Our major results are as follows. Given an involution on the roots of a Lie algebra, this involution can always be lifted to one of the algebra itself. Our first algorithm, Algorithm 4.5.5, accomplishes two important tasks:

1. Determine if lifting is necessary. (That is, if θ_Δ is already an involution).
2. If not, find all correction vectors by which we can modify θ_Δ so that it is an involution.

The drawback to this algorithm is that finding the correction vector(s) is a slow process. We can speed up the process via a “divide-and-conquer” approach using restricted

rank one decomposition. This is the technique of Algorithm 8.3.1. Helminck ?? gives us 18 restricted rank one involutions. Some of these are 1-consistent (so θ_Δ with respect to the set of roots in the restricted rank one system is an involution). Some of these are not, but they are much smaller in size (so Algorithm 4.5.5 runs relatively quickly). Hence, we can construct a modified involution $\theta_{\Delta \text{ ad}(H)}$ (again, with respect to the set of roots in the restricted rank one system). We then “glue” all these involutions together to give an involution defined over the original algebra.

10.2 Some Questions to Address

In the future, this work can be extended in multiple ways. Immediate questions that arise are as follows.

First, it may be worth investigating if a particular choice of monomial order speeds up computation. While our two algorithms provide a reasonably fast approach, restricted rank one decomposition may not be feasible or applicable to generalizations of this work.

Second, we only addressed involutions. We can address higher order involutions as well. Two questions immediately follow.

1. Can higher order involutions on the roots always be lifted?
2. If not, can these involutions be classified with respect to whether or not they lift?
3. Can higher order involutions on the roots be classified with respect to whether or not θ_Δ represents an automorphism of the same order as θ (e.g. 1-consistency schemes)

Third, in the spirit of Helminck’s work in [?], we can investigate commuting pairs of involutions. The same three questions listed above apply. We can then generalize to any group action.

Finally, we can recover an involution on the *local* symmetric space. However, we would also like to lift even higher, to the symmetric space itself.

Chapter 11

Programming Interface for Symbolic Computation in LiE Groups and Symmetric Spaces (Version 1.1 Manual)

The two chapters that follow describe the construction of a Local Symmetric Spaces package for Mathematica. The package implements the algorithms we discussed. We shall first present the full manual for installing and operating the package. Included is a discussion of the data structures and algorithm design choices. Immediately following the manual is a record of the source code used for the research presented.

11.1 Introduction

The *Programming Interface for Symbolic Computation in LiE Groups and Symmetric Spaces* package suite (PISCES) provides about 160 Mathematica programs designed to assist in the study of Lie groups and local symmetric spaces. While these procedures may not be equipped to answer any question one may have about Lie groups or local symmetric spaces, they should, at the very least, provide a foundation which can be built upon. With this in mind, the goal is to provide a good quantity of small, generalized, flexible procedures

which make as few assumptions as possible. Alternatives include the LiE package by van Leeuwen, Cohen, and Lisser [9], the Atlas of Lie Groups and Representations [10], and the Coxeter and Weyl packages by John Stembridge [7].

Some of the major features include:

- Functions for manipulating roots and root systems. These include procedures for drawing Dynkin diagrams, constructing Cartan matrices, and identifying the type of root system from the Cartan matrix.
- Functions for computing reflections between roots, inner products between roots, and the longest element of the Weyl group.
- Functions for describing and computing with involutions on the root system. Included are procedures for drawing the Helminck diagrams as in [5]. PISCES can also compute projections onto the roots of a local symmetric space and print diagrams to help visualize the action of the projections.
- PISCES can lift an involution on the root system to one on the Lie algebra. Hence we produce an involution suitable for computations in a local symmetric space.
- For most of the diagram capabilities, a related command exists which gives LaTeX source to draw the same diagram.

11.2 System Requirements

Much of the code for PISCES was developed on Mathematica 7.0, but has been tested 6.0. Most of the code relies on the lists, matrices, and graphics primitives - all of which have been available since the first versions. Hence, PISCES should work fine on earlier versions. However, this has not been tested.

PISCES is OS-independent and conservative with CPU and RAM resources. However, some of the lifting functions rely on computationally expensive Groebner Bases procedures. Much effort has been made to keep this reliance to a minimum.

PISCES's visualization capabilities depend upon the standard **GraphUtilities** package. This package is automatically loaded upon loading PISCES.

There are two flavors from which to choose. A *package* edition (collection of .m files) will allow easy integration into the Mathematica system. Installation of the package edition allows the entire suite (or specific components) to be loaded via Mathematica's *Needs* command. The *notebook* edition provides several Mathematica notebook (.nb) files.

11.3 Organization

The suite of five packages is arranged in a dependency tree. The packages are:

- Root System and Lie Algebra Package (PI-ROOT)
- Chevalley Structure Package (PI-CHEVY)
- Weyl Package (PI-WEYL)
- Group Action Package (PI-GAP)
- Local Symmetric Spaces Package (PI-LOSS)

With the *chevalley* package removed, the four remaining packages form a dependency chain, with PI-ROOT being the root-level package. PI-CHEVY depends only on PI-ROOT. Loading a specific package will automatically load any dependencies. Hence, to load PI-LOSS, it is not necessary to specifically ask for PI-ROOT, PI-WEYL, and PI-GAP.

11.4 Descriptions of Packages

The following guide will describe each package.

11.4.1 Root System and Lie Algebra Package (PI-ROOT)

PI-ROOT provides all root-level functionality to PISCES. This includes all utilities (e.g. message generation, graphics primitives, utility macros). Also included are many basic linear algebra routines which either provide functionality Mathematica does not have, or functionality which has been optimized for the specific uses PISCES will need.

PI-ROOT also includes a vast array of programs designed to work with root systems and Lie algebras. These features include

- Constructing root systems from bases, Cartan matrices, and names
- Identification of root system type
- Computing reflections of roots
- Constructing and utilizing automorphisms on root systems
- Constructing and utilizing automorphisms on semisimple Lie algebras
- Drawing and labeling Dynkin diagrams

11.4.2 Chevalley Structure Package (PI-CHEVY)

While PI-ROOT contains an algorithm to compute Chevalley structure constants [11], PI-CHEVY adds additional programs to perform further analysis. Currently the set of features is limited to working with and identifying special (and extra-special) pairs.

PI-CHEVY requires PI-ROOT.

11.4.3 Weyl Package (PI-WEYL)

PI-WEYL provides additional functionality geared toward working with and analyzing the Weyl group. PI-WEYL can

- Perform more complex Weyl group reflection operations
- Perform computations directly on Weyl group elements (e.g. compare, reduce)
- Analyze the Weyl group (e.g. compute the longest element)

Most of the functionality of PI-WEYL is based off of the functionality of the Weyl package by J. Stembridge [7].

PI-WEYL requires PI-ROOT.

11.4.4 Group Action Package (PI-GAP)

PI-GAP extends both PI-ROOT and PI-WEYL with additional functionality for working with group actions on the root system. PI-GAP can

- Work with roots fixed by root-automorphisms
- Work with projections into eigenspaces of root-automorphisms (including bases and root systems formed by projections)
- Extend the Dynkin diagram with additional information concerning fixed and projecting roots (e.g. *Helminck diagram*)

PI-GAP requires PI-WEYL and its dependency, PI-ROOT.

11.4.5 Local Symmetric Spaces Package (PI-LOSS)

PI-LOSS adds functionality for working with local symmetric spaces. PI-GAP can be seen as a generalization of PI-LOSS, where PI-LOSS works specifically with projections into the -1 eigenspace. PI-LOSS can

- Perform additional analysis on automorphisms on the roots which are of order 2 (involutional)
- Perform additional analysis on the roots which project into the local symmetric space
- Lift involutions on the root system to the local symmetric space

PI-LOSS requires PI-GAP and all of its dependencies (PI-WEYL, PI-ROOT).

11.5 Definitions and Data Structures

Here follows a brief discussion of the important data structures. If one is not careful, the choice of representations for the various elements (vectors, roots, etc.) can be quite limiting. The aim in PISCES is to be as general as possible. The trade-off is a weakened ability to check the types of arguments passed to a function. The user is cautioned to make careful note of the arguments to the common functions, listed in the following section.

The common data structures and representations are as follows:

- **Vectors.** Vectors are the most basic element, and are easily represented in Mathematica as lists.

- **Matrix.** Matrices are represented as sets of row vectors, e.g. $\{\{1,2\},\{3,4\}\}$ gives

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Vectors and matrices are the primary components of most of the data structures in PISCES .

- **Root Systems.** Root systems are named in three ways, and all of PISCES 's functions are equipped to handle all three as input. The human-readable *string form* gives a root system as “Rn+Rn+...” where R is the root system and n is the dimension. For example, the user can write the root system $A_4 \times E_6 \times F_4$ as “A4+E6+F4”.

The *list form* denotes a root system as $\{\{“R”,n\},\{“R”,n\},...\}$ where R and n are as before. The root system from our previous example, $A_4 \times E_6 \times F_4$, would then be $\{\{“A”,4\},\{“E”,6\},\{“F”,4\}\}$. Note that the root system names are enclosed in quotations. This form is more friendly for the programmer. Procedures to easily convert between the two types are in place, as well as a procedure which takes both types and converts to the single list form.

For any function which requires the input of a root system, its basis can be given. Indeed, for higher-level packages (PI-GAP, PI-LOSS) this is often the preferred calling method.

- **Roots.** Roots are denoted in two ways. As with root systems, the motivation is to balance readability and usability. Unfortunately, both representations must use Mathematica's *list* data structure. Hence, one representation is used as the standard, and the other for optional output. If a root is represented in *Euclidean form*, or *e-form*, it is expressed as a vector residing in the span of a chosen basis for the root system. For example, it is well known that a basis for a root system of type A_4 is given by

$$\alpha_i = e_i - e_{i+1}$$

where i ranges from 1 to 4. We say the root α_3 is in Euclidean form if it is represented as $e_3 - e_4$. Mathematica writes this vector as

$$\alpha_3 = \{0, 0, 1, -1, 0\}$$

This form is more suitable for computation, and is the standard in PISCES . The alternative form is *alpha form*. Alpha form gives the coordinates of the root with respect to the basis. In our A_4 example, α_3 is represented as

$$\alpha_3 = \{0, 0, 1, 0\}$$

which are the coordinates of α_3 with respect to the basis we gave.

The use of Euclidean form is primarily motivated by two factors. First, many of the low level root and Weyl computations are much more straight-forward. Second, projections into local symmetric spaces are much more easily handled. It is easy to convert between the two formats. Because Mathematics is somewhat “blind” as to which we are using, we will use Euclidean form as the standard.

- **Weyl Group Elements.** Weyl group elements are represented as a list of integers which index the order of the reflections produced. An example may provide the most straight-forward explanation. Suppose the user wishes to represent the element

$$s_{\alpha_1} s_{\alpha_2} s_{\alpha_5} s_{\alpha_9}$$

Then the reflection is represented as $\{1, 2, 5, 9\}$.

- **Basis.** A basis is represented as either a set of vectors (typically seen for roots, in Euclidean form), or a set of matrices (typically seen for Lie algebras). Note that a set of vectors is itself a matrix. This speeds up many of the computations. When a basis is given as a matrix, the individual elements form the *rows* of the matrix.

The RootBase command will compute the basis as described in Humphreys [?].

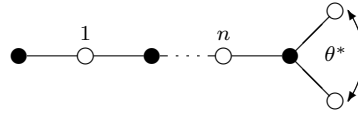
- **Lie Algebras.** A Lie algebra is represented as a collection of matrices which form the basis of the algebra. The command printMatrixArray is useful for listing the elements of the basis of a Lie algebra.

- **Automorphisms on the Root System.** *PI-ROOT* requires that automorphisms be designated by the matrices representing their action on the basis roots. Hence, upon the basis is almost always required alongside the automorphism.

Please note a common error when working with non-standard bases (those which do not match the basis given in Humphreys [1]): If a non-standard basis is used, that basis must be used for every subsequent computation. This scenario is typical when working with projections. For example, suppose a non-standard basis for A_4 is produced by a projection. Next we compute the matrix for an automorphism on this basis. If the next function call uses the name A_4 in place of the specific basis, then PISCES will compute the standard “Humphreys” basis *which does not match the matrix we use for our automorphism!*

When working with *involutions*, additional functionality is offered.

The representation of an involution on the root system is per the information encoded in the Helminck diagram [5]. A Helminck diagram encodes the pertinent information concerning an involution θ over the roots (enough to recover the action on the roots) in the following way. We extend the usual Dynkin diagram by coloring black dots corresponding to the roots fixed by θ . We show the action of the diagram automorphism (θ^*) by drawing arrows showing which roots are exchanged. An example follows:



In [5] Helmcink showed

$$\theta^* = -\text{id} \circ \theta \circ w_0(\theta)$$

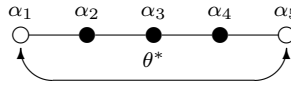
where $w_0(\theta)$ is the longest element of the Weyl group over the subsystem (“embedded” root system) formed by the fixed roots, and

$$\theta^* = \begin{cases} \text{id}; \\ \langle \text{Dynkin Diagram automorphism of order 2} \rangle \end{cases}$$

Because θ^* is therefore an involution, and it is known $w_0(\theta)$ is an involution, then we can write

$$\theta = -\text{id} \circ \theta^* \circ w_0(\theta)$$

PISCES represents an involution over the roots by utilizing two sets. The set usually named **disks** is a list of the simple roots fixed by θ . For example, if we say $\text{disks} = \{1, 3, 4\}$ then the roots fixed by theta are $\alpha_1, \alpha_3, \alpha_4$. The set usually named **arches** describes the diagram automorphism. This automorphism is encoded as a list of integer pairs, where each pair denotes which two roots are exchanged by θ^* . An example should prove helpful. Suppose we have the diagram below:



Then we encode θ^* by setting $\text{arches} = \{ \{1,5\}, \{2,4\} \}$. It should be noted that in [5] the arrows giving the action of θ^* on the fixed roots is typically omitted. However, the exchange of these pairs must be represented in the set arches . Hence, we have the pair $\{2,4\}$ listed.

The action of θ on the simple roots is represented as a matrix M , where $M\alpha = \theta(\alpha)$.

PI-LOSS adds an additional option for specifying root system involutions. For any argument which must be an *involution* on the roots, two arguments can be given in place

- *disks*, a list of indices of fixed roots
- *arches*, the diagram automorphism

Upon loading PI-LOSS, this option is available system-wide. Hence, programs in PI-ROOT will accept *disks* and *arches* in place of *theta*.

- **Involutions on the Lie Algebra.** A result found in [5] is as follows:

Lemma 11.5.1. *Let Δ be a basis of a root system Φ , $\theta \in \text{Aut}(\Phi)$ an involution, and $\phi \in \text{Aut}(\mathfrak{g}, \mathfrak{t})$ such that $\phi|_{\mathfrak{t}} = \theta$. Then ϕ is uniquely determined by the tuple $\{c_{\alpha, \phi}\}_{\alpha \in \Delta}$, where*

$$\phi(X_{\alpha}) = c_{\alpha, \phi} X_{\theta(\alpha)}$$

We call $c_{\alpha, \phi}$ the *structure constants* of the involution ϕ . Due to the lemma, recording these constants is a sufficient representation of the involution. Hence, PISCES stores the involution as a list of these values. However, PISCES must also be given Δ in order to recover the involution.

11.6 List of Procedures

This is a brief introduction to the procedures provided in PISCES . For online help, the command `?<command name>` will provide the brief descriptions below.

Each description is preceded by the command name and a list of the type of each argument (e.g. string, integer, matrix, etc.). Optional arguments are *italicized*. Each argument is separated by a pipe. (e.g. [string|integer|vector]).

In the event a command accepts multiple variations of arguments and types, each variant is listed independently, along with its specific behavior. If the behavior is consistent regardless of the type or number of arguments, there is only a single listing. In this case, the acceptable types are separated by a comma. As a quick example, [string|vector,matrix] denotes argument 1 must be a string, while argument 2 may be a vector or a matrix.

Commands for each package are separated into three categories. *Primary* commands are geared toward end-users. *Diagram* commands are primarily for information purposes only. Their outputs may not be used for further input. *Internal* commands may be of use to developers, but likely not end-users.

11.6.1 Root Systems and Lie Algebra (PI-ROOT) Primary

- `adMatrix [vector list|vector]`
`adMatrix[L,h]` writes the matrix for $\text{ad}(L)H$ where H is a vector in the Lie algebra L .

- `adMatrixDiagonal` [vector list|vector]
`adMatrixDiagonal[L,h]` writes the matrix for $\text{ad}(L)H$ where H is a vector in the Lie algebra L . It is a specialized variant of `adMatrix[]` which is faster, but is only applicable if the matrix for $\text{ad}(L)H$ is known to be diagonal.
- `AllRootBasisConnected` [basis|root]
`AllRootBasisConnected[basis,root]` returns the set of all basis roots which are members of the supplied basis and lie in the same irreducible root subsystem as the supplied root.
- `ApplyRootMap` [root system|matrix|root]
`ApplyRootMap[r,theta,root]` applies root (denoting a single root or set of roots) to an automorphism θ defined over a root system r . The root must lie in the system r , but may be mapped out of r by θ .
- `BasisCoefficients` [basis|vector,matrix]
`BasisCoefficients[b,x]` gives the coordinates of x with respect to basis b . x is understood to be either a vector or a matrix.
- `CartanMatrix` [root system]
`CartanMatrix[r]` gives the Cartan Matrix for root system r .
- `CartanToRootSystem` [matrix]
`CartanToRootSystem[m]` identifies the root system represented by Cartan Matrix m .
- `ChevalleyLookup` [list|root|root]
`ChevalleyLookup[nconsts,a,b]` looks up the Chevalley constant with respect to the pair of roots (a,b) . $nconsts$ is a table of the form returned by `kleinChevalley`.
- `ChevalleyLookup` [root system|root|root]
`ChevalleyLookup[r,a,b]` looks up the Chevalley constant with respect to the pair of roots (a,b) and the Lie algebra with root system r . WARNING: This variant of the procedure may be slow. Recommended for multiple calls is computing the table of Chevalley constants first, and using the “ $nconsts$ ” variation.
- `CoRoot` [root]
`CoRoot[r]` gives the co-root of root r .

- `cvMinimalPolynomialList` [root system|list]
`cvMinimalPolynomialList[basis,cvals]` writes $\{k,c\}$ for every basis coordinate in the basis structure constants list given by `cvals` so that the minimal polynomial of the coordinate is $x^2 + kx + c$. If the minimal polynomial is degree 2, only `c` is returned (`k` must be 1 due to the minimal polynomial being monic).
- `e` [integer|integer|integer]
`e[n,r,c]` forms an $n * n$ matrix with a 1 in the (r,c) position, and zeroes elsewhere.
- `e` [integer|integer]
`e[n,r]` forms an $1 \times n$ vector with a 1 in the r position.
- `FundamentalDominantWeights` [root system]
`FundamentalDominantWeights[r]` gives the fundamental dominant weights for a root system r .
- `gInvolutionListFormToMatrix` [root system|roots|matrix|list]
`gInvolutionListFormToMatrix[r,roots,theta,list]` takes a list of the form $\{\text{ROOT}, \text{CCONST}\}$ denoting an involution on the Lie algebra, the list of all roots, and the root system (r) involution θ and returns a matrix for the involution over the Lie algebra with respect to the ordered basis of root vectors (arranged with respect to the roots). The argument `roots` is optional. If omitted, the matrix will be set with respect to the ordering of the roots resulting from the procedure `RootSystem` with r as calling argument.
- `gInvolutionListFormToMatrix` [root system|roots|integer list|list|list]
(Pi-LOSS) `gInvolutionListFormToMatrix[r,roots,disks,arches,list]` takes a list of the form $\text{ROOT}, \text{CCONST}$ denoting an involution on the Lie algebra, the list of all roots, and the root system (r) involution θ (defined via fixed roots `disks` and diagram automorphism `arches`) and returns a matrix for the involution over the Lie algebra with respect to the ordered basis of root vectors (arranged with respect to the roots). The argument `roots` is optional. If omitted, the matrix will be set with respect to the ordering of the roots resulting from the procedure `RootSystem` with r as calling argument.

- `gMergeInvolutions` [basis|matrix|list|list]
`gMergeInvolutions[cbasis,theta,invol,nvals]` merges multiple involutions over the Lie algebra g (denoted by `invol` in list form). `invol` is a list formed by joining the separate involutions' lists. This function then fills in the missing structure constants corresponding to the roots in basis `cbasis` $a+b$, where a and b are roots residing in the separate restricted root systems. `nvals` is a list of Chevalley constants. `theta` is the involution over the root system.
- `gMergeInvolutions` [basis|integer list|list|list|list]
(Pi-LOSS) `gMergeInvolutions[cbasis,disks,arches,invol,nvals]` merges multiple involutions over the Lie algebra g (denoted by `invol` in list form). `invol` is a list formed by joining the separate involutions' lists. This function then fills in the missing structure constants corresponding to the roots in basis `cbasis` $a+b$, where a and b are roots residing in the separate restricted root systems. `nvals` is a list of Chevalley constants. `theta` is the involution over the root system, described via fixed roots disks and diagram automorphism arches.
- `HighestRoot` [root system|root]
`HighestRoot[d,roots]` gives the highest root amongst the set `roots` in the root system with basis `d`.
- `HighestRoot` [root system]
`HighestRoot[r]` gives the highest root in the root system `r`.
- `IdentifyRootSystem` [root system]
`IdentifyRootSystem[r]` identifies the type of root system where `r` is any form of a root system data type (string, list form, basis, set of roots).
- `InnerProduct` [vector|vector]
`InnerProduct[a,b]` gives the inner product of vectors `a` and `b`.
- `InnerProduct` [vector]
`InnerProduct[a]` gives the inner product of vector `a` with itself.
- `KleinChevalley` [root system]
`KleinChevalley[basis]` takes the basis of a root system and returns a Chevalley basis

for the relevant Lie algebra. The table returned is of the form TABLE A, TABLE B. Table B lists the Chevalley constants $N[i,j]$ where the (i,j) entry is the Chevalley constant with respect to root i , root j in the list of all roots of the root system. Table A lists all the roots in the root system, organized to match the order of roots for Table B.

- `LieBracket [vector|vector]`
`LieBracket[a,b]` gives the Lie bracket defined as $[a,b] = ab-ba$.
- `LinearOperatorMatrix [vector...]`
`LinearOperatorMatrix[l , l_1 ,..., l_n]` creates an $n * n$ matrix for a linear operator with respect to a basis B . The argument l_i denotes the coordinates of the vector the i basis element is mapped to.
- `LinearOperatorOrder [matrix]`
`LinearOperatorOrder[theta]` determines the order of some linear operator θ represented by a matrix. If θ is not an automorphism of any order n , then -1 is returned.
- `MakeBasis [vector list]`
`MakeBasis[v]` will return a basis for the collection of vectors v .
- `MakeRootBasis [root list]`
`MakeRootBasis[r]` will return a basis for the collection of root vectors r . `MakeRootBasis` preserves the condition that one root is double (or hence, half) that of another.
- `OperatorMatrixFromFunction [basis|function]`
`OperatorMatrixFromFunction[d,fn]` applies each vector in a basis d to a function fn and returns the matrix for the corresponding linear operator.
- `PositiveRootSystem [root system|root list]`
`PositiveRootSystem[r,roots]` returns the positive roots in the set 'roots' under root system r .
- `PositiveRootSystem [root system]`
`PositiveRootSystem[r]` returns all positive roots in the root system r .

- `PrintMatrixArray [matrix list]`
`PrintMatrixArray[l]` creates a human-readable table of a set of matrices `l`.
- `Reflect [root|root]`
`Reflect[a,b]` computes the reflection of the vector `b` across the hyperplane formed by `a`.
- `Reflect [root]`
`Reflect[a]` computes the reflection of vector `a` across its own hyperplane.
- `Reflect [root list|root|boolean]`
`Reflect[aList,b,rev]` computes the reflection of `b` across every vector listed in the ordered list `aList`. The optional argument `rev`, if `True`, will iterate through the list `aList` backward.
- `Reflect [root|root list]`
`Reflect[a,bList]` computes the set of vectors formed by reflecting each vector in `bList` across `a`.
- `Reflect [root list|root list|boolean]`
`Reflect[aList,bList,rev]` computes the set of vectors formed by reflecting each vector in `bList` across every vector in the ordered list `aList`. The optional argument `rev`, if `True`, will iterate through the list `aList` backward.
- `RestrictedRootAut [root system|matrix|basis]`
`RestrictedRootAut[r,theta,sub]` computes the matrix for an automorphism `theta` on the root system `r`, restricted to the given sub-basis.
- `RootAlphaForm [basis|root list]`
`RootAlphaForm[d,r]` gives the coordinates of a root or set of roots `r` with respect to basis `d`. (Writes root `r` in alpha form).
- `RootBase [root system]`
`RootBase[r]` gives a basis for a root system `r`.
- `RootBasisConnectedSet [root system]`
`RootBasisConnectedSet[basis]` returns a set of lists of basis vectors. The lists group

together basis roots which are members of the same irreducible root subsystem.

- `RootBasisConnectedQ` [root system|root|root]
`RootBasisConnectedQ[basis,a,b]` returns True if two basis vectors a and b in the supplied basis reside in the same irreducible root system.
- `RootBasisQ` [root system]
`RootBasisQ[r]` returns True if r is the basis of a root system.
- `RootCoBase` [root system]
`RootCoBase[r]` gives a co-basis for a root system r.
- `RootDecomposition` [root system]
`RootDecomposition[basis]` returns two tables TABLE A, TABLE B. The first table is a list of all roots in a root system with the given basis. The second table is a list pairs of roots. The i entry in table B is a set of two roots of a lower level such that they sum to the root given in the i entry of table A. The table is not unique.
- `RootFunctional` [root system|root|vector]
`RootFunctional[r,a,tv]` computes $a(tv)$ where a is a root and tv is a vector in the Cartan subalgebra (given as coordinates with respect to basis of the root system r).
- `RootHeight` [root system|root]
`RootHeight[d,r]` gives the height of root r in the root system with basis d.
- `RootLessQ` [root system|root|root]
`RootLessQ[r,a,b]` returns True if root a \leq root b with respect to root ordering and root system r.
- `RootSplit` [root system|root|boolean]
`RootSplit[d,r,forceb]` returns for a root r in a system with basis d the pair a,b so that $r = a + b$, where b is a basis root (if r is ≥ 0), or the negative of a basis root (if r ≤ 0), and a is a shorter root. If r is height 1 or -1, then returned is 0,r. The optional argument forceb ensures that b is always a basis element (for the case that the root is negative), or that 0,r is returned.

- `RootString` [root system|root|root]
`RootString[roots,a,b]`, where roots is the set of all roots in a root system, a and b are two roots, will compute the a-string through b
- `RootStringBounds` [root system|root|root]
`RootStringBounds[roots,a,b]`, where roots is the set of all roots in a root system, a and b are two roots, will compute positive integers p,q where $b + ka = -p$ $i=k$ $j=q$ is the a-string through b
- `RootSumPath` [root system|special|root]
`RootSumPath[basis,addtable,root]` computes a sequence of additions to root space vectors such that: the start of the sequence is a simple root, each intermediate step is a root, and the end result is the supplied root. The root system has the supplied basis. The argument addtable is "Table B" returned by `RootDecomposition`. The sequence will correlate to that used by `RootSystem`.
- `RootSumPath` [root system|root]
`RootSumPath[basis,root]` computes a sequence of additions to root space vectors such that: the start of the sequence is a simple root, each intermediate step is a root, and the end result is the supplied root. The root system has the supplied basis. The sequence may not correlate to the unique sequence produced by `RootDecomposition`, but does not require computing the addition table.
- `RootSystem` [root system]
`RootSystem[r]` gives the roots for root system r.
- `RootToString` [root]
`RootToString[r]` converts from r, the list form representation of a root system, to the human-readable string form. r is of the form A,n,B,n,C,n,... where A, B, C are the root system type (A-G), n is an integer, representing the root system $A_n+B_n+C_n+\dots$
- `StructureConstantsFromBasis` [root system|special|list|matrix]
`StructureConstantsFromBasis[r,sc,nconsts,theta]` returns a list of structure constants for an automorphism over the Lie algebra with root system r and root involution theta. nconsts supplies the Chevalley constants. sc supplies the structure constants for the

basis roots. The table returned is a list of elements of the form ROOT, CONSTANT. If nconsts is omitted, the procedure KleinChevalley will be called. However, for repeated usage it is recommended to compute once and store the Chevalley constants in memory.

- `StructureConstantsLookup [list|root]`
`StructureConstantsLookup[cconsts,a]` looks up the structure constant with respect to the root a. cconsts is a table of the form returned by `StructureConstants`.

11.6.2 Root Systems and Lie Algebra (PI-ROOT) Diagram

- `DrawRootSystem [root system|boolean]`
`DrawRootSystem[r,force]` draws a 2D or 3D plot of all roots of a root system r that is of dimension at most 3. Basis lines are represented in bold. The optional argument force, if True, forces `DrawRootSystem` to establish the diagram in a 3D plane. Because a Graphics object is returned, this is useful for combining 2D and 3D plots.
- `DynkinDiagram [root system]`
`DynkinDiagram[r]` gives the Dynkin Diagram of root system r.
- `gInvolutionDiagram [root system|matrix|list]`
`gInvolutionDiagram[r,theta,cvals]` extends the Helminck diagram with the values c and k necessary to recover the structure constants of an involution on the Lie algebra. Each basis root is labelled with $\{c,k\}$. The minimal polynomial of the corresponding structure constant is $1x^2 + kx + c$. r is the root system, theta the root system automorphism, and cvals is the list of structure constants.
- `gInvolutionDiagram [root system|integer list|list|list]`
(S-LOSS) `gInvolutionDiagram[r,disks,arches,cvals]` extends the Helminck diagram with the values c and k necessary to recover the structure constants of an involution on the Lie algebra. Each basis root is labelled with $\{c,k\}$. The minimal polynomial of the corresponding structure constant is $1x^2 + kx + c$. r is the root system, disks labels the fixed roots, arches represents the diagram automorphism, and cvals is the list of structure constants.

11.6.3 Root Systems and Lie Algebra (PI-ROOT) Internal

- `BasisCoefficientsMatrix` [basis|matrix]
`BasisCoefficientsMatrix[b,x]` gives the coordinates of x with respect to basis b . x must be a matrix.
- `BasisCoefficientsVector` [basis|vector]
`BasisCoefficientsVector[b,x]` gives the coordinates of x with respect to basis b . x must be a vector.
- `BasisToRootSystem` [basis]
`BasisToRootSystem[d]` identifies the type of root system formed by a basis d .
- `BlockAssemble` [matrix list]
`BlockAssemble[mlist]` assembles square matrices listed in m list into one large matrix with each list element along the diagonal.
- `BlockAssemble` [matrix...]
`BlockAssemble[a,b,...]` assembles square matrices a, b, \dots into one large matrix with each element along the diagonal.
- `BlockList` [matrix]
`BlockList[matr]` creates a list of each of the block matrices found in matrix $matr$.
- `ByBasisSort` [basis|list|*sort algorithm*]
`ByBasisSort[basis,list,p]` sorts a list of elements in the span of a given basis according to their coordinate vectors.
- `CartanMatrixFromBasis` [basis|*boolean*]
`CartanMatrixFromBasis[d,force]` gives the Cartan Matrix formed by a basis of roots d . The optional argument *force*, if `True`, will force a matrix to be generated even if the given set of vectors does not form a basis.
- `CartanNorm` [matrix|basis|root|root]
`CartanNorm[m,d,a,b]` computes $\langle a, b \rangle$ where a is a root, b is a simple root, and m is the Cartan Matrix for a root system with basis d .

- `CartanToRootSystemSimple [matrix]`
`CartanToRootSystemSimple[m]` identifies the root system represented by Cartan Matrix `m` if it is known to be an irreducible root system.
- `Diagonalize [matrix]`
`Diagonalize[m]` returns a matrix `p` and diagonal matrix `d` such that $d = p^{-1}mp$.
- `DynkinData [root system|integer list]`
`DynkinData[r,disks]` returns relevant data for a Dynkin diagram with root system `r`. The output is a list of elements of the form `neighbor 1, edge type , neighbor 2, edge type , ...` describing the neighbors a simple root shares, and the connecting bond type. The optional argument `disks` will throw out any data which points to a root not in the set of disks, and return `for` for any roots which are not in the set of disks. i.e. Return data for only a sub-diagram defined by the roots of disks.
- `DynkinDataNaturalOrdering [root system|integer list]`
`DynkinDataNaturalOrdering[r,disks]` returns relevant data for a Dynkin diagram with root system `r`. The output is a list of elements of the form `neighbor 1, edge type , neighbor 2, edge type , ...` describing the neighbors a simple root shares, and the connecting bond type. The optional argument `disks` will throw out any data which points to a root not in the set of disks, and return `for` for any roots which are not in the set of disks. i.e. Return data for only a sub-diagram defined by the roots of disks. This variant of `dynkinData` re-orders the labeling of the simple roots of type E. Roots 2 and 3 are swapped, then 3 and 4.
- `DynkinEdgeCodes [root system type|integer]`
`DynkinEdgeCodes[type,dim]` returns a list of codes where the `i` entry denotes the type of edge between the `i` and `(i+1)` simple roots. The edge codes are: 10 = single line, 21 = double left line, 22 = double right line, 31 = triple left line. `type` denotes the root system type, and `dim` denotes the dimension.
- `DynkinEdgeCons [root system type|integer]`
`DynkinEdgeCons[type,dim]` returns a list of pairs of simple roots which are joined together. `type` denotes the root system type, and `dim` denotes the dimension.

- `DynkinEdgeConsNestedForm` [root system type|integer]
`DynkinEdgeCons[type,dim]` returns a list of pairs of simple roots which are joined together. `type` denotes the root system type, and `dim` denotes the dimension. If a root shares multiple neighbors, then the entries are merged. e.g. 1,2,3 indicates root 1 is connected to both roots 2 and 3.
- `DynkinHeight` [root system]
`DynkinHeight[r]` gives the height of a Dynkin diagram of type `r`, where the height is the number of simple roots along the tallest vertical line.
- `DynkinPoints` [root system]
`DynkinPoints[r]` provides a list of relative x,y positions of the dots of a Dynkin diagram for a root system `r`.
- `DynkinPoints` [root system type|integer|integer|integer]
`DynkinPoints[type,dim,xOff,yOff]` provides a list of relative x,y positions of the dots of a Dynkin diagram for a root system of type `(type,dim)`. The points are offset along the x and y axes by `xOff` and `yOff` respectively.
- `DynkinOrientation` [root system name|basis]
`DynkinOrientation[r,basis]` returns True if the Dynkin diagram corresponding to the root system with the supplied basis is oriented in the direction standard to Pisces (See Humphreys, Introduction to Lie Algebra and Representation Theory).
- `DynkinOrientation` [root system]
`DynkinOrientation[r]` returns 1 if `r` is oriented 'correctly' (as per Humphreys), -1 if oriented backward, or 0 if basis elements are not ordered. If `r` is the name of a root system (not a basis), then 1 is always returned.
- `DynkinToCartan` [dynkin data]
`DynkinToCartan[D]` recovers the Cartan matrix from the Dynkin diagram information `D` returned by `dynkinData` and `dynkinDataNaturalOrdering`.
- `DynkinWidth` [root system]
`DynkinWidth[r]` gives the width of a Dynkin diagram of type `r`, where the width is the number of simple roots along the longest horizontal line.

- `eForm [root system|root]`
`eForm[d,r]` translates a root or set of roots `r` in alpha form to Euclidean form, where `d` is the basis of the root system.
- `FindPivots [matrix]`
`FindPivots[m]` indexes the columns containing pivots in `RREF[m]`.
- `GroebnerBackSolver [equations|symbol list]`
`GroebnerBackSolver[eqns,vars]` finds all solutions to the multivariate polynomial system given by `eqns` with set of variables `vars`.
- `GroebnerOneSolution [equations|symbol list]`
`GroebnerOneSolution[eqns,vars]` finds one solution to the multivariate polynomial system given by `eqns` with set of variables `vars`.
- `IrreducibleRootInput [root system]`
`IrreducibleRootInput[r]` converts an irreducible root system `r` into a variant of list form `TYPE,DIM`. An error message and `is` is returned if `r` is not irreducible.
- `LieMultTable [matrix list]`
`LieMultTable[L]` gives the Lie Multiplication Table for a set of vectors `L` forming the basis for a Lie algebra. The (i,j) entry of the table is the basis coordinate of the vector $v = [L_i, L_j]$.
- `LittleDynk [root system|integer|integer|integer|string list]`
`LittleDynk[r,xOff,yOff,nOff,labels]` draws the Dynkin diagram for an irreducible root system `r`. `xOff` and `yOff` are, respectively, the x and y coordinate offsets of the diagram. `nOff` provides the offset for automatic root numbering (starting value). The optional argument `labels` allows custom labels for the simple roots.
- `MatrixMinimalPolynomial [matrix|symbol]`
`MatrixMinimalPolynomial[a,x]` gives the minimal polynomial for a square matrix `a` with variable `x`.

Rowland,Todd and Weisstein,Eric W.”Matrix Minimal Polynomial.” From MathWorld—A Wolfram Web Resource.

<http://mathworld.wolfram.com/MatrixMinimalPolynomial.html>

- `MatrixNorm` [matrix]
`MatrixNorm[x]` normalizes a matrix `x`.
- `RootInput` [root system]
`RootInput[r]` converts `r` into the list form representation for a root system if `r` is in string form. If `r` is in list form, then `r` itself is returned.
- `RootInputQ` [root system]
`RootInputQ[r]` returns `True` if `r` is the name of a root system in list or string form.
- `RootListFormQ` [root system]
`RootListFormQ[r]` returns `True` if `r` is the name of a root system in list form.
- `RootStringFormQ` [root system]
`RootStringFormQ[r]` returns `True` if `r` is the name of a root system in string form.
- `RootSystemFromBasis` [root system]
`RootSystemFromBasis[d]` gives the roots in the root system formed by basis roots `d`.
- `RowSwap` [matrix|integer|integer]
`RowSwap[m,a,b]` swaps rows `a` and `b` in matrix `m`.
- `SimpleRootBase` [root system]
`SimpleRootBase[r]` gives a basis for an irreducible root system `r`.
- `StringToRoot` [string]
`StringToRoot[str]` converts a string `str` of the form `An+Bn+Cn...` to the list form representation for a root system, where `A`, `B`, `C` are the root system type (A-G), and `n` is an integer.
- `TakeElements` [list|integer list]
`TakeElements[l,in]` forms a sub-list of list `l` which contains the entries indexed in integer list `in`.
- `TakeRows` [matrix|integer list]
`TakeRows[m,r]` creates from a given matrix `m` a second matrix consisting of rows listed in a list `r`.

- `VectorPad [vector|integer|integer]`
`VectorPad[v,p,t]` takes a vector `v` and pads it on the left with `p` number of zeroes and enough zeroes on the right so that its total length is `t`.
- `ZeroesAbove [matrix|integer]`
`ZeroesAbove[matr,i]` returns `True` if each entry above the `(i,i)` entry in matrix `matr` is zero.
- `ZeroesBelow [matrix|integer]`
`ZeroesBelow[matr,i]` returns `True` if each entry below the `(i,i)` entry in matrix `matr` is zero.
- `ZeroesLeft [matrix|integer]`
`ZeroesLeft[matr,i]` returns `True` if each entry to the left of the `(i,i)` entry in matrix `matr` is zero.
- `ZeroesRight [matrix|integer]`
`ZeroesRight[matr,i]` returns `True` if each entry to the right of the `(i,i)` entry in matrix `matr` is zero.

11.6.4 Chevalley Structure Package (PI-CHEVY) Primary

- `ExtraSpecialPairs [root system]`
`ExtraSpecialPairs[r]` lists all pairs of roots which form extra special pairs.
- `ExtraSpecialPairs [root system|root list]`
`ExtraSpecialPairs[r,rlist]` lists all pairs of roots in the list `rlist` which form extra special pairs.
- `ExtraSpecialPairQ [root system|root|root]`
`ExtraSpecialPairQ[r,a,b]` returns `True` if positive roots `a` and `b` (under root system `r`) form an extra special pair.
- `ExtraSpecialPairQ [root system|root list|root|root]`
`ExtraSpecialPairQ[r,rlist,a,b]` returns `True` if positive roots `a` and `b` (under root system `r` and members of the set `rlist`) form an extra special pair. For repeated uses of this

procedure with the same root system, it is suggested to pre-compute a list of positive roots and use this variation. WARNING: rlist is intended to be either the set of all positive roots, or the entire root system. False will be returned if the sum $a + b$ is not a member of rlist. If rlist is not at least the set of all positive roots, False can be returned in the incorrect circumstances.

- `ExtraSpecialPairQ [root system|root list|root list|root|root]`
`ExtraSpecialPairQ[r,rlist,specs,a,b]` returns True if positive roots a and b (under root system r and members of the set $rlist$ with list of special pairs 'specs') form an extra special pair. For repeated uses of this procedure with the same root system, it is suggested to pre-compute a list of positive roots and special pairs, and use this variation. The same warning concerning $rlist$ as in the previous variation applies. There is no such warning for the list 'specs'. If $specs$ does not contain all special pairs in a root system, then only the extra special pairs within the given list will be marked.
- `SpecialPairs [root system]`
`SpecialPairs[r]` lists all pairs of roots which form special pairs.
- `SpecialPairs [root system|root list]`
`SpecialPairs[r,rlist]` lists all pairs of roots in the list $rlist$ which form special pairs.
- `SpecialPairQ [root system|root|root]`
`SpecialPairQ[r,a,b]` returns True if positive roots a and b (under root system r) form a special pair.
- `SpecialPairQ [root system|root list|root|root]`
`SpecialPairQ[r,rlist,a,b]` returns True if positive roots a and b (under root system r and members of the set $rlist$) form a special pair. For repeated uses of this procedure with the same root system, it is suggested to pre-compute a list of positive roots and use this variation. WARNING: $rlist$ is intended to be either the set of all positive roots, or the entire root system. False will be returned if the sum $a + b$ is not a member of $rlist$. If $rlist$ is not at least the set of all positive roots, False can be returned in the incorrect circumstances.

11.6.5 Weyl Package (PI-WEYL) Primary

- `LongestElement` [root system]
`LongestElement[r]` computes the longest element of the Weyl group of root system r .
- `LongestElement` [root system|integer list]
`LongestElement[r,disks]` computes the longest element of the Weyl group of a subsystem of root r which is formed by the basis roots indexed in `disks`.
- `ReflectWeyl` [root system|integer list|root]
`ReflectWeyl[d,a,b]` calculates $S_{\alpha_1} S_{\alpha_2} \dots S_{\alpha_s}(b)$ for root b and Weyl group element $a = \{a_1, a_2, \dots, a_s\}$, where all roots live in the set d .
- `WeylCompare` [root system|integer list|integer list]
`WeylCompare[d,w1,w2]` returns `True` if $w1 = w2$. $w1, w2$ are elements of $\text{Weyl}(d)$.
- `WeylLength` [root system|integer list]
`WeylLength[d,w]` computes the length of Weyl group element w . w is in the Weyl group for a root system with basis d .
- `WeylReduce` [root system|integer list]
`WeylReduce[d,w]` uses the Deletion Property to write a Weyl group element w reduced. w is in the Weyl group for a root system with basis d .

11.6.6 Weyl Package (PI-WEYL) Internal

- `InteriorPoint` [root system]
`InteriorPoint[r]` computes an interior point in the fundamental chamber with respect to root system r .
- `FundamentalChamber` [vector|root system]
`FundamentalChamber[v,r]` maps a vector v to the fundamental chamber with respect to a root system r .

11.6.7 Group Action Package (PI-GAP) Primary

- `ArchesListInvolution` [root system|integer list|matrix]

`ArchesListInvolution[r,disks,theta]` recovers the diagram automorphism from an involution `theta` defined over a root system `r` with fixed roots: `disks`.

- `DiskList [root system|matrix]`
`DiskList[r,theta]` is a variant of `fixedBasis` which lists the indices of the roots fixed by an involution `theta` over a root system `r`.
- `EigenspaceProject [root system|matrix|root|integer]`
`EigenspaceProject[r,theta,root,E]` projects `root` into some root in the `E`-eigenspace of the root automorphism `theta` over root system `r`.
- `EigenspaceProject [root system|integer list|list|root|integer]`
(S-LOSS) `EigenspaceProject[r,disks,arches,root,E]` projects `root` into some root in the `E`-eigenspace of the root automorphism described by fixed roots `disks` and diagram automorphism `arches` over root system `r`.
- `EmbeddedRootGroups [root system|matrix]`
`EmbeddedRootGroups[r,theta]` gives a list of the roots fixed by an automorphism `theta` over root system `r`. The list groups together roots by membership in an embedded root system.
- `EmbeddedRootGroups [root system|integer list]`
`EmbeddedRootGroups[r,disks]` gives a list of the roots fixed by an involution `theta` defined with fixed roots `disks` and diagram involution `arches` over root system `r`. The list groups together roots by membership in an embedded root system.
- `EmbeddedRootIndices [root system|matrix]`
`EmbeddedRootIndices[r,theta]` gives an index of the roots fixed by an involution `theta` over root system `r`. The list groups together roots by membership in an embedded root system.
- `EmbeddedRootIndices [root system|integer list]`
`EmbeddedRootIndices[r,disks]` gives an index of the roots fixed by an involution `theta` defined with fixed roots `disks` and diagram involution `arches` over root system `r`. The list groups together roots by membership in an embedded root system.

- `EmbeddedRootSystems [root system|matrix]`
`EmbeddedRootSystems[r,theta]` gives the embedded root systems formed by the root involution `theta` over a root system `r`.
- `EmbeddedRootSystems [root system|integer list]`
`EmbeddedRootSystems[r,disks]` gives the embedded root systems formed by the fixed roots disks of a root involution over a root system `r`.
- `FixedBasis [root system|matrix]`
`FixedBasis[r,theta]` computes the set of all basis roots fixed by an involution `theta` defined over root system `r`.
- `FixedBasis [root system|integer list|list]`
(S-LOSS) `FixedBasis[r,disks,arches]` computes the set of all basis roots fixed by an involution defined over root system `r` with fixed roots disks and diagram automorphism `arches`.
- `FixedRootQ [root system|matrix|root]`
`FixedRootQ[r,theta,root]` returns `True` if `root` is fixed by the involution `theta` defined over the root system `r`.
- `FixedRootQ [root system|integer list|list|root]`
(S-LOSS) `FixedRootQ[r,disks,arches,root]` returns `True` if `root` is fixed by the involution defined over the root system `r` with fixed roots disks and diagram automorphism `arches`.
- `FixedRoots [root system|matrix]`
`FixedRoots[r,theta]` returns the set of all roots fixed by the involution `theta` defined over the root system `r`.
- `FixedRoots [root system|integer list|list]`
(S-LOSS) `FixedRoots[r,disks,arches]` returns the set of all roots fixed by the involution defined over the root system `r` with fixed roots disks and diagram automorphism `arches`.
- `IsRootAutOrder [matrix|integer]`

`IsRootAutOrder[theta,n]` determines if an automorphism `theta` is of order `n`, or an order that divides `n`.

- `wInvolution [root system|matrix]`
`wInvolution[r,theta]` computes the longest element of the root system formed by the roots fixed by `theta`, a root system automorphism.
- `wInvolution [root system|integer list]`
`wInvolution[r,disks]` computes the longest element of the root system formed by the embedded roots (disks).
- `wInvolutionAction [root system|matrix]`
`wInvolutionAction[r,theta]` computes the matrix representing the action of the longest element of the Weyl group formed by the root system consisting of the roots fixed by `theta`, the root system automorphism over root system `r`.
- `wInvolutionAction [root system|integer list]`
`wInvolutionAction[r,disks]` computes the matrix representing the action of the longest element of the Weyl group formed by the embedded root systems (disks), where `r` is the root system.

11.6.8 Group Action Package (PI-GAP) Diagram

- `HelminckDiagram [root system|matrix|string list]`
`HelminckDiagram[r,theta,labels]` gives the Helminck Diagram of an involution `theta` on the roots acting on root system `r`. The optional argument `labels` allows custom labels for the simple roots.
- `HelminckDiagram [root system|integer list|list|string list]`
`HelminckDiagram[r,disks,arches,labels]` gives the Helminck Diagram of an involution on the roots acting on root system `r`. The involution is described by the fixed simple roots (disks) and the roots swapped by the diagram automorphism (arches). The optional argument `labels` allows custom labels for the simple roots.
- `HelminckDiagramTeX [root system|matrix|string list]`
`HelminckDiagramTeX[r,theta,labels]` gives LaTeX code to draw the Helminck Diagram

of an involution θ on the roots acting on root system r . The optional argument `labels` allows custom labels for the simple roots.

- `HelminckDiagramTeX` [root system|integer list|list|*string list*]
`HelminckDiagramTeX[r,disks,arches,labels]` gives LaTeX code to draw the Helminck Diagram of an involution on the roots acting on root system r . The involution is described by the fixed simple roots (disks) and the roots swapped by the diagram automorphism (arches). The optional argument `labels` allows custom labels for the simple roots.

11.6.9 Group Action Package (PI-GAP) Internal

- `DynkinPointsTeX` [root system type|integer|integer|integer]
`DynkinPointsTeX[type,dim,xOff,yOff]` provides a LaTeX-formatted list of relative x, y positions of the dots of a Dynkin diagram for a root system of type $(type, dim)$. The points are offset along the x and y axes by `xOff` and `yOff` respectively.
- `HelminckDiagramSTeX` [root system type|integer list|integer|integer|integer|*string list*]
`HelminckDiagramSTeX[r,disks,xOff,yOff,nOff,labels]` gives LaTeX code for the Helminck diagram for an irreducible root system r with fixed roots listed in `disks`. `xOff` and `yOff` are, respectively, the x and y coordinate offsets of the diagram. `nOff` provides the offset for automatic root numbering (starting value). The optional argument `labels` allows custom labels for the simple roots.
- `ThetaComponents` [root system|matrix]
`ThetaComponents[r,theta]` returns the components for a Helminck diagram to be merged with the Dynkin diagram. r denotes the root system, and θ the involution over r .
- `ThetaComponents` [root system|integer list|list]
`ThetaComponents[r,disks,arches]` returns the components for a Helminck diagram to be merged with the Dynkin diagram. r denotes the root system, `disks` denotes the fixed roots, and `arches` denotes the diagram automorphism.

11.6.10 Local Symmetric Spaces Package (PI-LOSS) Primary

- **AlignPolarities** [root system|matrix|list|root system|matrix|list|*integer*]
 AlignPolarities[d1,theta1,cvals1,d2,theta2,cvals2,w] aligns the polarities of two involutions over the Lie algebra cvals1 and cvals2. Each is defined with respect to root system involutions theta1 and theta2 over root systems d1 and d2. If the optional argument w is omitted, or is set to 1, the involution cvals1 is returned, modified so that its polarity aligns with cvals2. If w is 2, then cvals2 is returned, modified to align with cvals1. Each instance of theta (1 or 2) can be replaced with two arguments: disks, an integer index of the fixed roots, and arches, the diagram automorphism.
- **ApplyRootInvolution** [root system|integer list|list|root]
 ApplyRootInvolution[r,disks,arches,root] applies root (denoting a single root or set of roots) to an involution defined over a root system r, with fixed roots disks and diagram automorphism arches.
- **ApplyRootInvolution** [root system|matrix|root]
 ApplyRootInvolution[r,theta,root] applies root (denoting a single root or set of roots) to an involution theta defined over a root system r.
- **ApplyRootInvolutionBasis** [root system|integer list|list|root]
 ApplyRootInvolutionBasis[r,disks,arches,root] applies root (denoting a single root or set of roots) to an involution defined over a root system with basis d, with fixed roots disks and diagram automorphism arches.
- **ApplyRootInvolutionBasis** [root system|matrix|root]
 ApplyRootInvolutionBasis[d,theta,root] applies root (denoting a single root or set of roots) to an involution theta defined over a root system with basis d.
- **ComplementRoot** [root]
 ComplementRoot[root] complements the given root.
- **CorrectionVector** [root|matrix|list|symbol]
 CorrectionVector[r,theta,cvals,x] computes all involution correction vectors for an involution theta over the root system r with structure constants cvals. x supplies the name of the variable set for the toral vector coordinates.

- `DiagramInvolution` [root system]
`DiagramInvolution[r]` gives a list of root pairs “arches” in root system `r` which are swapped by the diagram automorphism of order 2.
- `InvBasisTable` [root system|matrix|list]
`InvBasisTable[basis,theta,rrdl]` gives a table of the structure constants necessary to determine if the Lie algebra homomorphism `rrdl` lifted from root system involution `theta` over the given root basis is an involution.
- `InvBasisTableAlphaForm` [root system|matrix|list]
`InvBasisTable[basis,theta,rrdl]` gives a table of the structure constants necessary to determine if the Lie algebra homomorphism `rrdl` lifted from root system involution `theta` over the given root basis is an involution. Roots are printed in alpha form.
- `InvolutionPolarity` [root system|matrix|list]
`InvolutionPolarity[r,theta,cconsts]` returns the polarity of an involution over the Lie algebra with structure constants `cconsts`. The involution is defined with respect to an involution `theta` over the root system `r`.
- `InvolutionPolarity` [root system|integer list|list|list]
`InvolutionPolarity[r,disks,arches,cconsts]` returns the polarity of an involution over the Lie algebra with structure constants `cconsts`. The involution is defined with respect to an involution `theta` (given as fixed roots `disks` and diagram automorphism `arches`) over the root system `r`.
- `LocalBasis` [root system|matrix]
`LocalBasis[r,theta]` computes the set of basis roots which project down to some root on the local symmetric space. `r` denotes the root system on which an involution `theta` is defined.
- `LocalBasis` [root system|integer list|list]
(S-LOSS) `LocalBasis[r,disks,arches]` computes the set of basis roots which project down to some root on the local symmetric space. `r` denotes the root system on which an involution with fixed roots `disks` and diagram automorphism `arches` is defined.

- `LocalProject` [root system|integer list|list|root]
`LocalProject[r,disks,arches,root]` projects root into some root in the local symmetric space whose root system is determined over an involution over root system r with fixed roots disks and diagram automorphism arches.
- `LocalProject` [root system|matrix|root]
`LocalProject[r,theta,root]` projects root into some root in the local symmetric space whose root system is determined by an involution theta over root system r.
- `OneCorrectionVector` [root system|matrix|list|symbol|integer]
`OneCorrectionVector[r,theta,cvals,x,sn]` computes one involution correction vector for an involution theta over the root system r with structure constants cvals. x supplies the name of the variable set for the toral vector coordinates. The optional argument sn allows the user to specify solution, number sn, in the case multiple solutions are present.
- `OrthoComplement` [root list|root]
`OrthoComplement[roots,com]` finds all roots in the set roots which are orthogonal to the root or set of roots given in com.
- `RankOneComponentBasis` [root system|matrix|root]
`RankOneComponentBasis[r,theta,root]` is a variant of `RankOneLocalBasis`. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta over the root system r. Roots which are fixed by theta are removed unless they are part of the same irreducible component of the rank one restricted root system with respect to root. In effect, this procedure generates the class of the rank one restricted root system found in Table I of "Algebraic Groups with a Commuting Pair of Involutions and Semisimple Symmetric Spaces" by A.G. Helminck.
- `RankOneComponentBasis` [root system|integer list|list|root]
`RankOneComponentBasis[r,disks,arches,root]` is a variant of `RankOneLocalBasis`. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta (described by fixed roots disks and diagram automorphism arches) over the root system r. Roots which are fixed by theta are removed

unless they are part of the same irreducible component of the rank one restricted root system with respect to root. In effect, this procedure generates the class of the rank one restricted root system found in Table I of "Algebraic Groups with a Commuting Pair of Involutions and Semisimple Symmetric Spaces" by A.G. Helminck.

- **RankOneLocalBasis** [root system|matrix|root]
RankOneLocalBasis[r,theta,root] is a variant of **RankOneBasis**. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta over the root system r. Roots which are fixed by theta are removed, leaving present only the roots which project down to some root in the local symmetric space.
- **RankOneLocalBasis** [root system|integer list|list|root]
RankOneLocalBasis[r,disks,arches,root] is a variant of **RankOneBasis**. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta (described by fixed roots disks and diagram automorphism arches) over the root system r. Roots which are fixed by theta are removed, leaving present only the roots which project down to some root in the local symmetric space.
- **RankOneRootLift** [root system|matrix|list|list]
RankOneRootLift[r,theta,cconsts,nvals] lifts a restricted rank one involution theta, with respect to root system r, to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. nvals supplies the Chevalley constants.
- **RankOneRootLift** [root system|integer list|list|list|list]
RankOneRootLift[r,disks,arches,cconsts,nvals] lifts a restricted rank one involution theta (described via fixed roots disks and diagram automorphism arches), with respect to root system r, to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. nvals supplies the Chevalley constants.
- **ReduceRestrictedRank** [root system|integer list|list|root]
ReduceRestrictedRank[r,disks,arches,root] reduces the restricted rank of an involution defined over root system r with fixed roots disks and diagram automorphism arches by eliminating root.

- `ReduceRestrictedRank` [root system|matrix|root]
`ReduceRestrictedRank[r,theta,root]` reduces the restricted rank of an involution θ defined over root system r by eliminating root .
- `RestrictedRankOneBasis` [root system|integer list|list|root]
`RestrictedRankOneBasis[r,disks,arches,root]` computes the basis of the restricted rank one root system with respect to root , defined by an involution over root system r with fixed roots disks and diagram automorphism arches .
- `RestrictedRankOneBasis` [root system|matrix|root]
`RestrictedRankOneBasis[r,theta,root]` computes the basis of the restricted rank one root system with respect to root , defined by an involution θ over root system r .
- `RestrictedRankOneDecomp` [root system|matrix]
`RestrictedRankOneDecomp[r,theta]` computes the restricted rank one decomposition of an involution θ over a root system r .
- `RestrictedRankOneSystem` [root system|integer list|list|root]
`RestrictedRankOneSystem[r,disks,arches,root]` computes the restricted rank one root system with respect to root , defined by an involution over root system r with fixed roots disks and diagram automorphism arches .
- `RestrictedRankOneSystem` [root system|matrix|root]
`RestrictedRankOneSystem[r,theta,root]` computes the restricted rank one root system with respect to root , defined by an involution θ over root system r .
- `RestrictedRootBasis` [root system|integer list|list]
`RestrictedRootBasis[r,disks,arches]` gives a basis for the restricted root system determined by an involution over root system r with fixed roots disks and diagram automorphism arches .
- `RestrictedRootBasis` [root system|matrix]
`RestrictedRootBasis[r,theta]` gives a basis for the restricted root system determined by an involution θ over root system r .
- `RestrictedRootRank` [root system|integer list|list]
`RestrictedRootRank[r,disks,arches]` gives the rank of the restricted root system de-

terminated over an involution over root system r with fixed roots disks and diagram automorphism arches.

- `RestrictedRootRank` [root system|matrix]
`RestrictedRootRank[r,theta]` gives the rank of the restricted root system determined by an involution θ over root system r .
- `RestrictedRootSystem` [root system|integer list|list]
`RestrictedRootSystem[r,disks,arches]` computes the set of all roots composing the restricted root system determined over an involution over root system r with fixed roots disks and diagram automorphism arches.
- `RestrictedRootSystem` [root system|matrix]
`RestrictedRootSystem[r,theta]` computes the set of all roots composing the restricted root system determined by an involution θ over root system r .
- `RestrictedRootSystemType` [root system|integer list|list]
`RestrictedRootSystemType[r,disks,arches]` identifies the type of the restricted root system determined by an involution over root system r with fixed roots disks and diagram automorphism arches.
- `RestrictedRootSystemType` [root system|matrix]
`RestrictedRootSystemType[r,theta]` identifies the type of the restricted root system determined by an involution θ over root system r .
- `RootCriticalValues` [root system|matrix|list]
`RootCriticalValues[d,theta,cconsts]` returns a table of $\theta(d)$ where θ is an automorphism of the root system r , and $cconsts$ is the table of all structure constants returned by `structureConstants`.
- `RootCriticalValues` [root system|integer list|list|list]
`RootCriticalValues[d,disks,arches,cconsts]` returns a table of $\theta(d)$ where θ is an automorphism of the root system r described by fixed roots disks and diagram automorphism arches, and $cconsts$ is the table of all structure constants returned by `structureConstants`.

- `RootInvolution` [root system|integer list|list]
`RootInvolution[r,disks,arches]` or `RootInvolution[r,disks,arches]` gives the matrix of an involution acting on root system `r`, where disks are those points represented by solid black disk, and arches is a list of elements `a,b` denoting the diagram automorphism maps `a` to `b`.
- `RRDLift` [root system|matrix|list|list]
`RRDLift[r,theta,cconsts,nvals]` lifts an involution `theta`, with respect to root system `r`, to an involution on its corresponding Lie algebra. `cconsts` supplies the structure constants. Output is in list format. `nvals` supplies the Chevalley constants.
- `RRDLift` [root system|integer list|list|list|list]
`RRDLift[r,disks,arches,cconsts,nvals]` lifts an involution `theta` (described via fixed roots disks and diagram automorphism arches), with respect to root system `r`, to an involution on its corresponding Lie algebra. `cconsts` supplies the structure constants. Output is in list format. `nvals` supplies the Chevalley constants.
- `RROITable` [integer]
`RROITable[n]` generates the table of Helminck diagrams for involutions (over the root systems) of restricted rank one and of minimal size. The optional argument `n` returns the `n` entry (1 - 18), while the absence of an argument generates the entire table.
- `RROITableEntry` [root system|matrix]
`RROITableEntry[r,theta]` identifies the Restricted Rank One Involutions table entry corresponding to an involution `theta` over root system `r`.
- `RROITableEntry` [root system|integer list|list]
`RROITableEntry[r,disks,arches]` identifies the Restricted Rank One Involutions table entry corresponding to an involution `theta` (given by fixed roots disks and diagram automorphism arches) over root system `r`.
- `SimpleRootLift` [root system|matrix|list|list|boolean]
`SimpleRootLift[d,theta,cconsts,nvals,donly]` lifts an involution `theta` with respect to root system `r` to an involution on its corresponding Lie algebra. `cconsts` supplies the structure constants. `nvals` supplies the Chevalley constants. The technique used is

the simpler Groebner basis technique, which is slow. For a quicker calculation, use `RRDLift`. `simpleLift` does not check for 1-consistency. Output is in list format. If the optional argument `only` is `True`, only the basis structure constants are returned.

- `SimpleRootLift` [root system|integer list|list|list|list|*boolean*]
`SimpleRootLift[d,disks,arches,cconsts,nvals,only]` lifts an involution θ (defined via fixed roots `disks` and diagram automorphism `arches`) with respect to root system r to an involution on its corresponding Lie algebra. `cconsts` supplies the structure constants. `nvals` supplies the Chevalley constants. The technique used is the simpler Groebner basis technique, which is slow. For a quicker calculation, use `RRDLift`. `simpleLift` does not check for 1-consistency. Output is in list format. If the optional argument `only` is `True`, only the basis structure constants are returned.
- `SteinbergThetaDelta` [root system|list|matrix]
`SteinbergThetaDelta[r,nconsts,theta]` returns a list of structure constants for the automorphism over the Lie algebra `TD` with root system r and root involution θ . `TD` is the unique automorphism such that each basis root structure constant is 1. `nconsts` supplies the Chevalley constants. The table returned is a list of elements of the form `ROOT, CONSTANT`. If `nconsts` is omitted, the procedure `KleinChevalley` will be called. However, for repeated usage it is recommended to compute once and store the Chevalley constants in memory.
- `SwitchPolarity` [root system|matrix|list]
`SwitchPolarity[r,theta,cconsts]` reverses the polarity of an involution over the Lie algebra defined by `cconsts`. θ is the involution on the root system r .
- `SwitchPolarity` [root system|integer list|list|list]
`SwitchPolarity[r,disks,arches,cconsts]` reverses the polarity of an involution over the Lie algebra defined by `cconsts`. θ is the involution, described by fixed roots `disks` and diagram automorphism `arches`, on the root system r .
- `ThetaStable` [vector list|root system|matrix]
`ThetaStable[h,d,theta]` returns `True` if $\theta(x)$ is in the set h for all x in h , where d supplies the basis θ is defined over.

11.6.11 Local Symmetric Spaces Package (PI-LOSS) Diagram

- `RankOneDecompDiagram` [root system|matrix]
`RankOneDecompDiagram[r,theta]` draws the restricted rank one decomposition diagram for an involution `theta` over the root system `r`.
- `ReduceRestrictedRankDiagram` [root system|integer list|list|root]
`ReduceRestrictedRankDiagram[r,disks,arches,root]` gives a Helminck diagram which illustrates the reduction of the restricted rank of an involution defined over root system `r` with fixed roots `disks` and diagram automorphism `arches` by eliminating `root`.
- `RestrictedRankOneDiagram` [root system|integer list|list|root]
`RestrictedRankOneDiagram[r,disks,arches,root]` computes a Helminck diagram illustrating the restricted rank one root system with respect to `root`, defined by an involution over root system `r` with fixed roots `disks` and diagram automorphism `arches`.
- `RestrictedRankOneDiagram` [root system|matrix|root]
`RestrictedRankOneDiagram[r,theta,root]` computes a Helminck diagram illustrating the restricted rank one root system with respect to `root`, defined by an involution `theta` over root system `r`.
- `RestrictedRootDiagram` [root system|integer list|list]
`RestrictedRootDiagram[r,disks,arches]` draws a Helminck diagram and Dynkin diagram for the restricted root system determined over an involution over root system `r` with fixed roots `disks` and diagram automorphism `arches`.
- `RestrictedRootDiagram` [root system|matrix]
`RestrictedRootDiagram[r,theta]` draws a Helminck diagram and Dynkin diagram for the restricted root system determined by an involution `theta` over root system `r`.

11.6.12 Local Symmetric Spaces Package (PI-LOSS) Internal

- `DiagramInvolutionSimple` [root system|integer]
`DiagramInvolution[r,offset]` gives a list of root pairs “arches” in an irreducible root system `r` which are swapped by the diagram automorphism of order 2. The optional argument `offset` shifts the indices by the specified number of units.

11.7 Tutorial and Examples

Here we present a few examples which should also suffice as a “quick start guide” to PISCES . Some prerequisite knowledge of Mathematica is assumed. At the very least, the user should be familiar with the way Mathematica represents lists, vectors, and matrices.

A quick 20 minute introduction to Mathematica can be found on Wolfram’s website at <http://www.wolfram.com/broadcast/screencasts/handsonstart/>. A one-page written guide can also be found at www.gpc.edu/~dunmol/PDFHandouts/mathematica_quick_start.pdf.

Example 11.7.1. *Roots and Root Systems*

As a first example, let us compute the basis of the root system A_5 . This is easily accomplished with one command.

```
basis = RootBase["A5"]
```

which returns

$$\{\{1, -1, 0, 0, 0, 0\}, \{0, 1, -1, 0, 0, 0\}, \{0, 0, 1, -1, 0, 0\}, \{0, 0, 0, 1, -1, 0\}, \{0, 0, 0, 0, 1, -1\}\}$$

we can then compute *all* the roots in A_5 in one of two ways. If we know the basis above forms A_5 we can call

```
roots = RootSystem["A5"]
```

However, if we had only the basis roots and did not know what root system they formed, we can also call

```
roots = RootSystem[basis]
```

Both return the following set.

$\{ \{0, 0, 0, 0, 1, -1\}, \{0, 0, 0, 1, -1, 0\}, \{0, 0, 0, 1, 0, -1\}, \{0, 0, 1, -1, 0, 0\},$
 $\{0, 0, 1, 0, -1, 0\}, \{0, 0, 1, 0, 0, -1\}, \{0, 1, -1, 0, 0, 0\}, \{0, 1, 0, -1, 0, 0\},$
 $\{0, 1, 0, 0, -1, 0\}, \{0, 1, 0, 0, 0, -1\}, \{1, -1, 0, 0, 0, 0\}, \{1, 0, -1, 0, 0, 0\},$
 $\{1, 0, 0, -1, 0, 0\}, \{1, 0, 0, 0, -1, 0\}, \{1, 0, 0, 0, 0, -1\}, \{0, 0, 0, 0, -1, 1\},$
 $\{0, 0, 0, -1, 1, 0\}, \{0, 0, 0, -1, 0, 1\}, \{0, 0, -1, 1, 0, 0\}, \{0, 0, -1, 0, 1, 0\},$
 $\{0, 0, -1, 0, 0, 1\}, \{0, -1, 1, 0, 0, 0\}, \{0, -1, 0, 1, 0, 0\}, \{0, -1, 0, 0, 1, 0\},$
 $\{0, -1, 0, 0, 0, 1\}, \{-1, 1, 0, 0, 0, 0\}, \{-1, 0, 1, 0, 0, 0\}, \{-1, 0, 0, 1, 0, 0\},$
 $\{-1, 0, 0, 0, 1, 0\}, \{-1, 0, 0, 0, 0, 1\} \}$

One command writes all the roots in alpha form.

`RootAlphaForm[basis]`

returns the following set of roots.

$\{ \{0, 0, 0, 0, 1\}, \{0, 0, 0, 1, 0\}, \{0, 0, 0, 1, 1\}, \{0, 0, 1, 0, 0\},$
 $\{0, 0, 1, 1, 0\}, \{0, 0, 1, 1, 1\}, \{0, 1, 0, 0, 0\}, \{0, 1, 1, 0, 0\},$
 $\{0, 1, 1, 1, 0\}, \{0, 1, 1, 1, 1\}, \{1, 0, 0, 0, 0\}, \{1, 1, 0, 0, 0\},$
 $\{1, 1, 1, 0, 0\}, \{1, 1, 1, 1, 0\}, \{1, 1, 1, 1, 1\}, \{0, 0, 0, 0, -1\},$
 $\{0, 0, 0, -1, 0\}, \{0, 0, 0, -1, -1\}, \{0, 0, -1, 0, 0\}, \{0, 0, -1, -1, 0\},$
 $\{0, 0, -1, -1, -1\}, \{0, -1, 0, 0, 0\}, \{0, -1, -1, 0, 0\}, \{0, -1, -1, -1, 0\},$
 $\{0, -1, -1, -1, -1\}, \{-1, 0, 0, 0, 0\}, \{-1, -1, 0, 0, 0\}, \{-1, -1, -1, 0, 0\},$
 $\{-1, -1, -1, -1, 0\}, \{-1, -1, -1, -1, -1\} \}$

We can easily construct the Cartan matrix. The command:

`matrix = CartanMatrix[basis]`

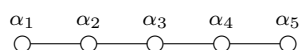
gives us the matrix:

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

Note that we did not need to know the basis formed the root system A_5 . Of course, we can also draw the Dynkin diagram.

```
DynkinDiagram["A5"]
```

produces the diagram



As a closing note, suppose we did not know the basis formed the system A_5 , and we wished to identify this. We use the command

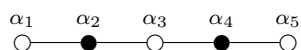
```
CartanToRootSystem[matrix]
```

and PISCES tells us the root system is

$$\{ \{A,5\} \}$$

Example 11.7.2. *Involutions on the Root System*

Suppose we have θ as induced by the Helminck diagram given below



The fixed roots are α_2 and α_4 . These roots form a basis for a subsystem. We'll call this basis "basis2":

```
basis2 = { basis[[2]], basis[[4]] }
```

This is the set of vectors that form basis2.

$$\{ \{0, 1, -1, 0, 0, 0\}, \{0, 0, 0, 1, -1, 0\} \}$$

We construct the Cartan matrix from this set:

```
matrix2 = CartanMatrix[basis2]
```

and obtain

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

The reader may immediately recognize this as the Cartan matrix for $A_1 \times A_1$. PISCES confirms this. The command

```
rootssystem2 = CartanToRootSystem[matrix2]
```

gives

$$\{\{A,1\},\{A,1\}\}$$

The longest element of $A_1 \times A_1$ with respect to α_2 and α_4 is given by the command

```
le = LongestElement[rootssystem2, {2, 4}]
```

and we learn

$$le = \{ 2, 4 \}$$

hence

$$w_0(\theta) = s_{\alpha_2} s_{\alpha_4}$$

What PISCES needs to know are the fixed roots. We say

```
disks = { 2, 4 };
```

to denote that α_2 and α_4 are fixed by θ . There is no diagram automorphism, so we denote the identity as

```
arches = { };
```

and PISCES can compute the action of θ :

```
theta = RootInvolution["A5",disks,arches];
```

We receive as output the matrix giving the action of θ with respect to our basis.

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

Example 11.7.3. *Root Projections*

The roots which are not fixed by θ project down to roots in some local symmetric space. Recall the projection π is defined as

$$\pi(\alpha) = \frac{1}{2}(\alpha - \theta(\alpha))$$

PISCES can quickly compute the basis of this projected root system, given by

$$\bar{\Delta} = \{\lambda_i = \pi(\alpha) \mid \alpha \in \Delta, \theta(\alpha) \neq \alpha\}$$

The command

```
rbasis = RestrictedRootBasis["A5",disks,arches]
```

gives this set as

$$\left\{ \left\{ 0, 0, 0, \frac{1}{2}, \frac{1}{2}, -1 \right\}, \left\{ 0, \frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, 0 \right\}, \left\{ 1, -\frac{1}{2}, -\frac{1}{2}, 0, 0, 0 \right\} \right\}$$

The Cartan matrix formed by this basis is given by:

```
rmatrix = CartanMatrix[rbasis]
```

and is

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

PISCES identifies the restricted root system.

```
CartanToRootSystem[rmatrix]
```

gives

$$\{\{A, 3\}\}$$

We can compute the entire root system.

```
RestrictedRootSystem["A5",disks,arches]
```

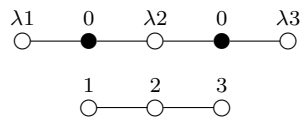
gives

$$\begin{aligned} &\left\{0, 0, 0, \frac{1}{2}, \frac{1}{2}, -1\right\}, \quad \left\{0, \frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, 0\right\}, \quad \left\{0, \frac{1}{2}, \frac{1}{2}, 0, 0, -1\right\}, \\ &\left\{1, -\frac{1}{2}, -\frac{1}{2}, 0, 0, 0\right\}, \quad \left\{1, 0, 0, -\frac{1}{2}, -\frac{1}{2}, 0\right\}, \quad \left\{1, 0, 0, 0, 0, -1\right\}, \\ &\left\{0, 0, 0, -\frac{1}{2}, -\frac{1}{2}, 1\right\}, \quad \left\{0, -\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0\right\}, \quad \left\{0, -\frac{1}{2}, -\frac{1}{2}, 0, 0, 1\right\}, \\ &\left\{-1, \frac{1}{2}, \frac{1}{2}, 0, 0, 0\right\}, \quad \left\{-1, 0, 0, \frac{1}{2}, \frac{1}{2}, 0\right\}, \quad \left\{-1, 0, 0, 0, 0, 1\right\} \end{aligned}$$

Finally, PISCES provides a nice diagram mechanism to represent both the root and restricted system. The command

```
RestrictedRootDiagram["A5",disks,arches]
```

produces the following figure.



Next we can quickly verify that θ is, indeed, an involution.

```
LinearOperatorOrder[theta]
```

produces the result 2.

Let us define a map on the Lie algebra in the following way. Let T be our automorphism on the Lie algebra, and X_a be the root vector corresponding to root a . Then let

$$T(X_a) = X(\theta(a)) \text{ for every basis root } a.$$

A result due to Steinberg [4] is that T is unique and a Lie algebra homomorphism. It is not, however, always an involution.

PISCES stores Lie algebra homomorphisms by storing their structure constants. Our first goal is to build the homomorphism T and call its set of structure constants "cvals". First, however, we need to build the Lie algebra itself. We shall construct a Chevalley basis using Klein's algorithm. [11]

The following command stores the Chevalley constants in the table "nvals".

```
nvals = KleinChevalley[basis];
```

Next we construct the structure constants for T , and store them in the table "cvals".

```
cvals = SteinbergThetaDelta[basis, nvals, theta];
```

cvals describes completely the homomorphism T . It is either order 2 (involution), or order 4. If it is an involution we are done. Let us verify by constructing the matrix with respect to the root vectors for the Lie algebra.

```
tMatrix = gInvolutionListFormToMatrix[r, roots, theta, cvals];
```

tMatrix is a very large matrix. However, we can quickly check the order of the map it represents.

```
LinearOperatorOrder[tMatrix]
```

produces the value 4.

This should be expected, as the command below verifies that not every diagonal entry of $(tMatrix)^2$ is one.

That the diagonal entries are all one implies rrdl is an involution. The order of the matrix (and hence, rrdl itself) is easily computed.

LinearOperatorOrder[rMatrix]

produces the result 2.

PISCES - A system for symbolic computation in Lie Groups and Symmetric Spaces
Copyright (C) 2010 Robert L. Watson

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Chapter 12

Programming Interface for Symbolic Computation in LiE Groups and Symmetric Spaces (Version 1.1.0 Source)

In this chapter we'll provide Mathematica source code for actual implementation of our discussed routines. Most of this code was written in 2009 - 2010 for use on Mathematica 7. There are three primary sections.

First we discuss some of the coding strategy (data structures, etc.), the justification, and any words of caution. We follow this discussion with basic routines in Lie algebra which we will require. Finally we provide implementation of the algorithms discussed in this text.

12.1 Notes on Construction of The Mathematica Lie Algebra and Local Symmetric Spaces Package

The code provides a complete system for implementation of the previously introduced algorithms. However, it does have several limitations that should be noted.

1. A few of the routines are only equipped to work with irreducible root systems.

2. Lie algebra elements must be matrices.
3. Chevalley constants must be provided by other software. One good procedure is given in Klein [11]. An implementation is provided.

As discussed in the manual, five packages are provided. With the exception of the Chevalley constants package, the packages form a dependency chain. At the top level of the chain is the local symmetric spaces package. Each previous package is intended to be a generalization of the one above it. The organization was chosen with future work in mind. In particular, the group action package generalizes the local symmetric spaces package so that projections onto other eigenspaces can be explored.

Data structures have been chosen to be efficient in speed. In particular, this translates to working with Mathematica primitives (e.g. lists) as much as possible. In some instances this approach hinders type-checking. This is particularly true when working with different sorts of bases. To a limited degree the user will need to take responsibility for ensuring the correct types are being worked with. Partially for this reason the diagram capabilities are provided.

12.2 Root System and Lie Algebra Package (Primary)

12.2.1 adMatrix

adMatrix::usage=

"adMatrix[L,h] writes the matrix for $\text{ad}(L)H$ where H is a vector in the Lie algebra L ."

Write the matrix for $\text{ad}_L(H)$.

```
(* REVISION:
Use LU Decomposition! We're solving systems with the same coefficient \
matrix, with different RHS vectors *)

adMatrix[L_?ListQ, h_?MatrixQ] := Module[
  {ad, bvec, v, i, bstd, bm, f, vf},

  (* STAGE I -
  Essentially copy the basisCoeffsM code to solve a linear system *)
  \
  (* 1. Write all the basis vectors in standard basis *)

  For[i = 1, i <= Length[L], i++,
    bstd[i] = L[[i]] // Flatten;
  ];
```

```

(* 2. Form a matrix with bstd as rows *)

bm = Table[bstd[i], {i, 1, Length[L]}];

(* 3. Take the transpose *)
bm = Transpose[bm];

(* 4. Solve the Linear System generically *)

Quiet[f = LinearSolve[bm]];

(* STAGE II - Solve the system with varying RHS *)

For[i = 1, i <= Length[L], i++,
  v = LieBracket[h, L[[i]]];

  (* Write v in standard basis *)
  vf = Flatten[v];

  bvec[i] = f[vf];
];

(* Form the matrix *)
ad = Table[bvec[i], {i, 1, Length[L]}];

ad
];

adMatrix[a___] := InvalidArg["adMatrix", a];

```

12.2.2 adMatrixDiagonal

adMatrixDiagonal::usage=

”adMatrixDiagonal[L,h] writes the matrix for $\text{ad}(L)H$ where H is a vector in the Lie algebra L . It is a specialized variant of `adMatrix[]` which is faster, but is only applicable if the matrix for $\text{ad}(L)H$ is known to be diagonal.”;

```
(* Here we know ahead of time the matrix for ad will be diagonal *)

adMatrixDiagonal[L_?ListQ, h_?MatrixQ] := Module[
  {ad, v, b, soln, k, i},

  ad = ConstantArray[0, {Length[L], Length[L]}];

  For[i = 1, i <= Length[L], i++,
    (* Diagonal ad matrix => [h,L[[i]]] = k L[[i]] *)

    v = L[[i]] // Flatten;

    b = LieBracket[h, L[[i]]] // Flatten;

    soln = Solve[b == k*v, k] // Flatten;

    ad[[i, i]] = k /. soln;
  ];

  ad
];

adMatrixDiagonal[a_...] := InvalidArg["adMatrixDiagonal", a];
```

12.2.3 AllRootBasisConnected

AllRootBasisConnected::usage=

”AllRootBasisConnected[basis,root] returns the set of all basis roots which are members of the supplied basis and lie in the same irreducible root subsystem as the supplied root.”;

```
AllRootBasisConnected[d_?ListQ, root_?VectorQ] := Module[
  {bcs, index},

  bcs = RootBasisConnectedSet[d];

  If[! MemberQ[d, root], Return[Fail];];

  For[index = 1, index <= Length[bcs], index++,
    If[MemberQ[bcs[[index]], root],
      Break[];
    ];
  ];

  Return[bcs[[index]]];
];

AllRootBasisConnected[a___] :=
InvalidArg["AllRootBasisConnectedSet", a];
```

12.2.4 ApplyRootMap

ApplyRootMap::usage=

"ApplyRootMap[r,theta,root] applies root (denoting a single root or set of roots) to an automorphism theta defined over a root system r. The root must lie in the system r, but may be mapped out of r by theta.";

```
ApplyRootMap[r_?RootInputQ, theta_?MatrixQ, root_?ListQ] := Module[
  {basis, kroot, i},

  If[SameQ[Depth[root], 2],
    basis = RootBase[r];
    kroot = BasisCoefficients[basis, root];

    Return[kroot.Transpose[theta].basis];
  ,
  Return[
    DeleteDuplicates[
      Table[ApplyRootInvolution[r, theta, root[[i]]], {i, 1,
        Length[root]}]]];
];

ApplyRootMap[d_?RootBasisQ, theta_?MatrixQ, root_?ListQ] := Module[
  {kroot, i},

  If[SameQ[Depth[root], 2],
    kroot = BasisCoefficients[d, root];

    Return[kroot.Transpose[theta].d];
  ,
  Return[
    DeleteDuplicates[
      Table[ApplyRootInvolutionBasis[d, theta, root[[i]]], {i, 1,
        Length[root]}]]];
];

(*
ApplyRootMap[r_?RootInputQ, theta_?MatrixQ, root_?ListQ] := \
ApplyRootInvolution[r, theta, root];
*)

(*
ApplyRootMap[d_?RootBasisQ, theta_?MatrixQ, root_?ListQ] := \
ApplyRootInvolution[d, theta, root];
*)

ApplyRootMap[a_...] := InvalidArg["ApplyRootMap", a];
```

12.2.5 BasisCoefficients

BasisCoefficients::usage=

”BasisCoefficients[b,x] gives the coordinates of x with respect to basis b. x is understood to be either a vector or a matrix.”;

```
BasisCoefficients::nmember = "'1' can not be expressed in the basis";
```

```
BasisCoefficients::arg2 = "'1' is not a vector or a matrix.";
```

```
BasisCoefficients[b_?ListQ, x_?ListQ] := Module[
  {},
```

```
  Which[
    VectorQ[x], Return[BasisCoefficientsVector[b, x]];,
    MatrixQ[x], Return[BasisCoefficientsMatrix[b, x]];,
    _, Message[BasisCoefficients::arg2, x];
  ];
```

```
  Return[{}];
];
```

```
BasisCoefficients[a___] := InvalidArg["BasisCoefficients", a];
```


12.2.6 CartanMatrix

CartanMatrix::usage=

"CartanMatrix[r] gives the Cartan Matrix for root system r."

```
CartanMatrix[R_?RootInputQ] := Module[
  {basis, i, j, rin},

  rin = RootInput[R];
  basis = RootBase[R];

  (*Table[Table[innerProduct[basis[[i]],cobasis[[j]]],{j,1,Length[
basis]}],{i,1,Length[basis]}]*)

  Return[
    Table[Table[
      2*InnerProduct[basis[[i]],
        basis[[j]]/InnerProduct[basis[[j]], basis[[j]]], {j, 1,
        Length[basis]}], {i, 1, Length[basis]}]];
  ];

  CartanMatrix[R_?RootBasisQ] := CartanMatrixFromBasis[R];

  CartanMatrix[a_...] := InvalidArg["CartanMatrix", a];
```

12.2.7 CartanToRootSystem

CartanToRootSystem::usage=

”CartanToRootSystem[m] identifies the root system represented by Cartan Matrix m.”;

```
CartanToRootSystem[matr_?MatrixQ] := Module[
  {blocks, str, i},

  str = "";
  blocks = BlockList[matr];

  For[i = 1, i <= Length[blocks], i++,
    str = str <> CartanToRootSystemSimple[blocks[[i]]];

    If[! SameQ[i, Length[blocks]],
      str = str <> "+";
    ];
  ];

  Return[str];
];

CartanToRootSystem[a_...] := InvalidArg["CartanToRootSystem", a];
```

12.2.8 ChevalleyLookup

ChevalleyLookup::usage=

”ChevalleyLookup[nconsts,a,b] looks up the Chevalley constant with respect to the pair of roots (a,b). nconsts is a table of the form returned by kleinChevalley.

ChevalleyLookup[r,a,b] looks up the Chevalley constant with respect to the pair of roots (a,b) and the Lie algebra with root system r. WARNING: This variant of the procedure may be slow. Recommended for multiple calls is computing the table of Chevalley constants first, and using the “nconsts” variation.”;

```

ChevalleyLookup::noroot =
  "Root '1' or '2' not found in supplied table.";

ChevalleyLookup[d_?RootBasisQuietQ, a_?VectorQ, b_?VectorQ] :=
  ChevalleyLookup[KleinChevalley[d], a, b];

ChevalleyLookup[nconsts_?ListQ, a_?VectorQ, b_?VectorQ] := Module[
  {i, j},
  For[i = 1, i <= Length[nconsts[[1]]], i++,
    For[j = 1, j <= Length[nconsts[[1]]], j++,
      If[SameQ[nconsts[[1, i]], a] && SameQ[nconsts[[1, j]], b],
        Return[nconsts[[2, i, j]]];
      ];
    ];
  ];

Message[ChevalleyLookup::noroot, a, b];

Return[{}];
];

ChevalleyLookup[a_...] := InvalidArg["ChevalleyLookup", a];

```

12.2.9 CoRoot

CoRoot::usage=

”CoRoot[r] gives the co-root of root r.”;

```
CoRoot[x_?VectorQ] := 2*x/InnerProduct[x, x];
```

```
CoRoot[a___] := InvalidArg["CoRoot", a];
```

12.2.10 cvMinimalPolynomialList

cvMinimalPolynomialList::usage=

”cvMinimalPolynomialList[basis,cvals] writes k,c for every basis coordinate in the basis structure constants list given by cvals so that the minimal polynomial of the coordinate is $x^2 + kx + c$. If the minimal polynomial is degree 2, only c is returned (k must be 1 due to the minimal polynomial being monic).”;

```
cvMinimalPolynomialList[basis_?RootBasisQ, cvals_?ListQ] := Module[
  {i, mlist, x, mpoly, clist, cval},

  mlist = {};

  For[i = 1, i <= Length[basis], i++,
    cval = StructureConstantsLookup[cvals, basis[[i]]];

    mpoly = MinimalPolynomial[cval, x];
    clist = CoefficientList[mpoly, x];

    If[SameQ[Length[clist], 3],
      mlist = Join[mlist, {{clist[[1]], clist[[2]]}}];
    ,
      mlist = Join[mlist, {clist[[1]]}];
    ];
  ];

  Return[mlist];
];

cvMinimalPolynomialList[a___] :=
  InvalidArg["cvMinimalPolynomialList", a];
```

12.2.11 e

e::usage=

"e[n,r,c] forms an n * n matrix with a 1 in the (r,c) position, and zeroes elsewhere.

e[n,r] forms an 1 x n vector with a 1 in the r position.";

```
e[n_?IntegerQ, r_?IntegerQ, c_?IntegerQ] :=
  Transpose[{UnitVector[n, r]}.{UnitVector[n, c]}];
```

```
e[n_?IntegerQ, r_?IntegerQ] := UnitVector[n, r];
```

```
e[a___] := InvalidArg["e", a];
```

12.2.12 eForm

eForm::usage=

"eForm[d,r] translates a root or set of roots r in alpha form to Euclidean form, where d is the basis of the root system.";

```
eForm[basis_?RootBasisQ, roots_?ListQ] := Module[
  {i},
  If[SameQ[Depth[roots], 3],
    Return[Table[basis.roots[[i]], {i, 1, Length[roots]}]];
  ,
  Return[basis.roots];
];
eForm[a_...] := InvalidArg["eForm", a];
```

12.2.13 FundamentalDominantWeights

FundamentalDominantWeights::usage=

”FundamentalDominantWeights[r] gives the fundamental dominant weights for a root system r.”;

```
FundamentalDominantWeights[r_?RootInputQ] :=
  FundamentalDominantWeights[RootBase[r]];

FundamentalDominantWeights[d_?RootBasisQ] := Module[
  {matr},

  matr = CartanMatrix[d];

  Return[Inverse[matr]];
];

FundamentalDominantWeights[a_...] :=
  InvalidArg["FundamentalDominantWeights", a];
```


12.2.14 gInvolutionListFormToMatrix

gInvolutionListFormToMatrix::usage=

”gInvolutionListFormToMatrix[r,roots,theta,list] takes a list of the form ROOT,CCONST denoting an involution on the Lie algebra, the list of all roots, and the root system (r) involution theta and returns a matrix for the involution over the Lie algebra with respect to the ordered basis of root vectors (arranged with respect to the roots). The argument roots is optional. If omitted, the matrix will be set with respect to the ordering of the roots resulting from the procedure RootSystem with r as calling argument.

(PI-LOSS) gInvolutionListFormToMatrix[r,roots,disks,arches,list] takes a list of the form ROOT,CCONST denoting an involution on the Lie algebra, the list of all roots, and the root system (r) involution theta (defined via fixed roots disks and diagram automorphism arches) and returns a matrix for the involution over the Lie algebra with respect to the ordered basis of root vectors (arranged with respect to the roots). The argument roots is optional. If omitted, the matrix will be set with respect to the ordering of the roots resulting from the procedure RootSystem with r as calling argument.”;

```
gInvolutionListFormToMatrix[r_?RootInputQ, inroots_: {},
  theta_?MatrixQ, rlist_?ListQ] :=
  gInvolutionListFormToMatrix[RootBase[r], inroots, theta, rlist];

gInvolutionListFormToMatrix[d_?RootBasisQ, inroots_: {},
  theta_?MatrixQ, rlist_?ListQ] := Module[
  {n, ret, i, s, t, root, troot, k, roots},

  If[SameQ[inroots, {}],
    roots = RootSystem[d];
  ,
    roots = inroots;
  ];

  n = Length[roots];

  ret = ConstantArray[0, {n, n}];

  For[i = 1, i <= Length[roots], i++,
    root = roots[[i]];
    k = StructureConstantsLookup[rlist, root];

    troot = ApplyRootInvolution[d, theta, root];

    For[s = 1, s <= n, s++,
      If[roots[[s]] == troot, Break[]];
    ];

    For[t = 1, t <= n, t++,
      If[roots[[t]] == root, Break[]];
    ];
```

```

    ret[[s, t]] = k;
  ];

  Return[ret];
];

gInvolutionListFormToMatrix[a_++] :=
  InvalidArg["gInvolutionListFormToMatrix", a];

(*
gInvolutionListFormToMatrixOLD[r_, roots_, theta_, rlist_] := Module[
{n, ret, i, s, t, root, troot, k},

n=Length[roots];

ret=ConstantArray[0,{n,n}];

For[i=1,i<=Length[rlist],i++,
root=rlist[[i,1]];
k=rlist[[i,2]];

troot=applyInvolution[r,theta,root];

For[s=1,s<=n,s++,
If[roots[[s]]==troot,Break[]];
];

For[t=1,t<=n,t++,
If[roots[[t]]==root,Break[]];
];

ret[[s,t]]=k;
];

Return[ret];
];
*)

```

12.2.15 gMergeInvolutions

gMergeInvolutions::usage=

”gMergeInvolutions[cbasis,theta,invol,nvals] merges multiple involutions over the Lie algebra g (denoted by $invol$ in list form). $invol$ is a list formed by joining the separate involutions’ lists. This function then fills in the missing structure constants corresponding to the roots in basis $cbasis$ $a+b$, where a and b are roots residing in the separate restricted root systems. $nvals$ is a list of Chevalley constants. $theta$ is the involution over the root system.

(PI-LOSS) gMergeInvolutions[cbasis,disks,arches,invol,nvals] merges multiple involutions over the Lie algebra g (denoted by $invol$ in list form). $invol$ is a list formed by joining the separate involutions’ lists. This function then fills in the missing structure constants corresponding to the roots in basis $cbasis$ $a+b$, where a and b are roots residing in the separate restricted root systems. $nvals$ is a list of Chevalley constants. $theta$ is the involution over the root system, described via fixed roots disks and diagram automorphism $arches$.”;

```
gMergeInvolutions[cbasis_?ListQ, theta_?MatrixQ, invol_?ListQ,
  nvals_?ListQ] := Module[
  {k, p, n, h, i, j, r, t, q, s, size, m,
   ret, c1, c2, n1, n2},

  (* 1. Init *)
  p = cbasis;
  ret = invol;
  n = 1;

  m = CartanMatrixFromBasis[cbasis];

  size = 1;

  (* 2. Construct *)
  While[Length[p] > size,
    size = Length[p];

    For[i = 1, i <= Length[p], i++,
      For[j = 1, j <= Length[cbasis], j++,
        t = p[[i]];

        (* 2.1.
        Get the basis coefficients and check height of the root *)

        k = BasisCoefficients[cbasis, t];
        h = Sum[k[[s]], {s, 1, Length[k]}];

        If[h == n,
          (* 2.2. Determine the integer r *)
          r = 0;
          While[MemberQ[p, t - r*cbasis[[j]]], r++];
          r = r - 1;
```

```

(* 2.3. Define q *)

q = r - Sum[k[[s]]*m[[s, j]], {s, 1, Length[cbasis]}];

(* 2.4. Ammend p? *)
If[q > 0,
  p = Union[p, {t + cbasis[[j]]}];

(* MOD to De Graaf: merge involution lists here *)

Off[StructureConstantsLookup::noroot];

(* POSITIVE ROOTS *)

If[SameQ[
  StructureConstantsLookup[ret, t + cbasis[[j]]], {}],
  c1 = StructureConstantsLookup[ret, t];
  c2 = StructureConstantsLookup[ret, cbasis[[j]]];

  n1 = ChevalleyLookup[nvals,
    ApplyRootInvolutionBasis[cbasis, theta, t],
    ApplyRootInvolutionBasis[cbasis, theta, cbasis[[j]]]];
  n2 = ChevalleyLookup[nvals, t, cbasis[[j]]];

  ret = Union[ret, {{t + cbasis[[j]], c1*c2*n1/n2}}];
];

(* NEGATIVE ROOTS *)

If[SameQ[StructureConstantsLookup[ret, -t - cbasis[[j]]], {}],
  c1 = StructureConstantsLookup[ret, -t];
  c2 = StructureConstantsLookup[ret, -cbasis[[j]]];

  n1 = ChevalleyLookup[nvals,
    ApplyRootInvolutionBasis[cbasis, theta, -t],
    ApplyRootInvolutionBasis[cbasis, theta, -cbasis[[j]]]];
  n2 = ChevalleyLookup[nvals, -t, -cbasis[[j]]];

  ret = Union[ret, {{-t - cbasis[[j]], c1*c2*n1/n2}}];
];

On[StructureConstantsLookup::noroot];
];
];
];

n = n + 1;
];

(*
Print[Length[ret]];
*)

Return[ret];
];

gMergeInvolutions[a___] := InvalidArg["gMergeInvolutions", a];

```

12.2.16 HighestRoot

HighestRoot::usage=

"HighestRoot[d,roots] gives the highest root amongst the set roots in the root system with basis d.

HighestRoot[r] gives the highest root in the root system r.

HighestRoot[d] gives the highest root in the root system with basis d."

```
HighestRoot[r_?RootInputQ] := HighestRoot[RootBase[r], RootSystem[r]];
```

```
HighestRoot[d_?RootBasisQ] := HighestRoot[d, RootSystemFromBasis[d]];
```

```
HighestRoot[d_?RootBasisQ, roots_?ListQ] := Module[
  {high, i},
  high = roots[[1]];
  For[i = 1, i <= Length[roots], i++,
    If[RootHeight[d, roots[[i]]] > RootHeight[d, high],
      high = roots[[i]];
    ];
  ];
  Return[high];
];
```

```
HighestRoot[a_...] := InvalidArg["HighestRoot", a];
```

12.2.17 IdentifyRootSystem

IdentifyRootSystem::usage=

”IdentifyRootSystem[r] identifies the type of root system where r is any form of a root system data type (string, list form, basis, set of roots).”;

```

IdentifyRootSystem[r_?RootStringFormQ] := r;

IdentifyRootSystem[r_?RootListFormQ] := RootToString[r];

IdentifyRootSystem[r_?RootBasisQuietQ] := Module[
  {rb},

  Return[BasisToRootSystem[rb]];
];

IdentifyRootSystem[r_?MatrixQ] := Module[
  {rb},

  rb = MakeRootBasis[r];
  Return[BasisToRootSystem[rb]];
];

IdentifyRootSystem[a_...] := InvalidArg["IdentifyRootSystem", a];

```

12.2.18 InnerProduct

InnerProduct::usage=

”InnerProduct[a,b] gives the inner product of vectors a and b.

InnerProduct[a] gives the inner product of vector a with itself.”;

```

InnerProduct::length =
  "vectors '1' and '2' do not have the same length.";

InnerProduct[a_?VectorQ] := InnerProduct[a, a];

InnerProduct[a_?VectorQ, b_?VectorQ] := Module[{},
  If[Length[a] != Length[b],
    Message[InnerProduct::length, a, b];
    Return[Fail];
  ];

  Sum[a[[i]]*b[[i]], {i, 1, Length[a]}]
];

```

12.2.19 KleinChevalley

KleinChevalley::usage=

”KleinChevalley[basis] takes the basis of a root system and returns a Chevalley basis for the relevant Lie algebra. The table returned is of the form TABLE A, TABLE B. Table B lists the Chevalley constants $N[i,j]$ where the (i,j) entry is the Chevalley constant with respect to root i , root j in the list of all roots of the root system. Table A lists all the roots in the root system, organized to match the order of roots for Table B.

KleinChevalley[r] computes the table above, with r being the name of a root system.”;

```
KleinChevalley[r_?RootInputQ] := KleinChevalley[RootBase[r]];

KleinChevalley[d_?RootBasisQ] := Module[
  {i, j, a, b, c, l, p, q, rs, gm, de, gmIndex, deIndex, c2,
   roots, at, posroots, negroots, max, root, index, at1, at2, gamma,
   delta},

  rs = RootDecomposition[d];

  roots = rs[[1]];
  at = rs[[2]];

  posroots = Take[roots, Length[roots]/2];
  negroots = Take[roots, -Length[roots]/2];

  (* From RECONSTRUCTING THE GEOMETRIC STRUCTURE OF A RIEMANNIAN \
  SYMMETRIC SPACE FROM ITS SATAKE DIAGRAM - Sebastian Klein *)

  (* All positive roots *)

  For[i = 1, i <= Length[posroots], i++,
    For[j = 1, j <= Length[posroots], j++,
      a = posroots[[i]];
      b = posroots[[j]];

      If[! MemberQ[posroots, a + b],
        c[a, b] = 0;
      ];
    ];

  max = RootHeight[d, HighestRoot[d, roots]];

  For[l = 2, l <= max, l++,
    For[i = 1, i <= Length[posroots], i++,
      root = posroots[[i]];

      If[RootHeight[d, root] != 1, Continue[]];

      (* Get the index of root *)

      index = Position[posroots, root][[1, 1]];

      (* Root_index = Sum of roots at1 and at2 *)
```



```

(*
Note for cartanNorm, it is only linear in first variable.
So the second must be a simple root.
rootSystemAT is designed to build root sums this way.
The algorithm is designed to call at2 as the first variable.
So switch. *)
at1 = at[[index, 1]];
at2 = at[[index, 2]];

(* Find the smallest integer so that at2 - ( p +
1 ) at1 NOT IN roots holds
Start with p = -1 and keep adding until the above expression is \
not a root *)
p = 0;

(*While[MemberQ[roots,at2-p*at1],
p=p+1;
];
*)
While[MemberQ[roots, at2 - (p + 1)*at1],
p = p + 1;
];

(*
p=0;
While[MemberQ[roots,at2-p*at1],p++];
p=p-1;
*)
(*
p=max;

While[!MemberQ[roots,at2-(p+1)*at1],
p=p-1;
];

p++;
*)

q = p - 2*InnerProduct[at2, at1]/(InnerProduct[at1, at1]);

c[at1, at2] = Sqrt[(q*(1 + p))/2]*Sqrt[InnerProduct[at1, at1]];

(*
If[c[at1,at2]==0,Print["ERROR. (q*(1+p))=", (q*(1+p)),
". innerProduct[at1,at1]=", InnerProduct[at1,at1]"."];];
*)

c[at2, at1] = -c[at1, at2];

(* For all pairs (gm,de) of positive roots with gm + de = root,
where neither gm nor de are in {at1,at2} *)

For[gmIndex = 1, gmIndex <= Length[roots]/2, gmIndex++,
For[deIndex = 1, deIndex <= Length[roots]/2, deIndex++,
gm = roots[[gmIndex]];
de = roots[[deIndex]];

(*
If[MemberQ[{at1,at2},gm],Continue[]];
If[MemberQ[{at2,at2},gm],Continue[]];
*)
If[MemberQ[{at1, at2}, gm], Continue[]];

```

```

If[MemberQ[{at1, at2}, de], Continue[]];
If[gm + de != root, Continue[]];

c2[1, 1] = Which[
  MemberQ[posroots, gm - at2], c[at2, gm - at2],
  MemberQ[-posroots, gm - at2], c[gm, at2 - gm],
  True, 0];

c2[1, 2] = Which[
  MemberQ[posroots, at1 - de], c[de, at1 - de],
  MemberQ[-posroots, at1 - de], c[at1, de - at1],
  True, 0];

c2[2, 1] = Which[
  MemberQ[posroots, at1 - gm], c[at1 - gm, gm],
  MemberQ[-posroots, at1 - gm], c[gm - at1, at1],
  True, 0];

c2[2, 2] = Which[
  MemberQ[posroots, de - at2], c[at2, de - at2],
  MemberQ[-posroots, de - at2], c[de, at2 - de],
  True, 0];

c[gm,
  de] = (1/c[at1, at2])*(c2[1, 1]*c2[1, 2] +
    c2[2, 1]*c2[2, 2]);
]; (* End deIndex loop *)
]; (*
End gmIndex loop *)
]; (* End loop on i *)
]; (*
End loop on l *)

(* Now get the other roots *)

For[i = 1, i <= Length[posroots], i++,
  For[j = 1, j <= Length[posroots], j++,
    c[posroots[[i]], -posroots[[j]]] = Which[
      MemberQ[posroots, posroots[[j]] - posroots[[i]],
      c[posroots[[j]] - posroots[[i]], posroots[[i]]],
      MemberQ[-posroots, posroots[[j]] - posroots[[i]],
      c[posroots[[i]] - posroots[[j]], posroots[[j]]],
      True, 0];

    c[-posroots[[i]],
      posroots[[j]]] = -c[posroots[[i]], -posroots[[j]]];

    c[-posroots[[i]], -posroots[[j]]] = -c[posroots[[i]],
      posroots[[j]]];
  ];
];

Return[{roots,
  Table[Table[
    c[roots[[i]], roots[[j]]], {j, 1, Length[roots]}], {i, 1,
    Length[roots]}]};
];

KleinChevalley[a___] := InvalidArg["KleinChevalley", a];

```

12.2.20 LieBracket

LieBracket::usage=

”LieBracket[a,b] gives the Lie bracket defined as $[a,b] = ab-ba$.”;

```
LieBracket[a_?MatrixQ, b_?MatrixQ] := a.b - b.a;
```

```
LieBracket[a___] := InvalidArg["LieBracket", a];
```

12.2.21 LinearOperatorMatrix

LinearOperatorMatrix::usage=

"LinearOperatorMatrix[l,l1,...,ln] creates an $n \times n$ matrix for a linear operator with respect to a basis B. The argument li denotes the coordinates of the vector the i basis element is mapped to.

```
LinearOperatorMatrix[l_?VectorQ] := Module[
  {n, vlist},

  vlist = List[l];
  n = Length[vlist];

  Return[Transpose[vlist]];
];

LinearOperatorMatrix[a_...] := InvalidArg["LinearOperatorMatrix", a];
```

12.2.22 LinearOperatorOrder

LinearOperatorOrder::usage=

"LinearOperatorOrder[theta] determines the order of some linear operator theta represented by a matrix. If theta is not an automorphism of any order n, then -1 is returned."

```
LinearOperatorOrder[theta_?MatrixQ] := Module[
  {polyn, x, clist, degree, i},

  (* Add one... explanation later *)

  polyn = 1 + MatrixMinimalPolynomial[theta, x];

  clist = CoefficientList[polyn, x];

  (* Degree is one less the length of the coefficient list *)

  degree = Length[clist] - 1;

  (* Hacque *)
  If[SameQ[degree, 1] && SameQ[theta[[1, 1]]^2, 1],
    Return[1];
  ];

  (* Now we want to see if the minimal polynomial is of the form -1+
  x^n. If this is so,
  our own polynomial should only have a 1 in the last entry in the \
  Coefficient List. (because we added one to the polynomial) *)

  For[i = 1, i <= Length[clist] - 1, i++,
    If[! SameQ[clist[[i]], 0],
      Return[-1];
    ];
  ];

  Return[degree];
];

LinearOperatorOrder[a_...] := InvalidArg["LinearOperatorOrder", a];
```

12.2.23 MakeBasis

MakeBasis::usage=

"MakeBasis[v] will return a basis for the collection of vectors v."

```
MakeBasis[v_?MatrixQ] := Module[
  {pb, plist, i, d},

  (* Row reduce echelon form *)
  pb = RowReduce[Transpose[v]];

  (* Find the pivots *)
  plist = FindPivots[pb];

  (* Pivot locations correspond to basis vectors *)
  d = {};

  For[i = 1, i <= Length[plist], i++,
    d = Join[d, {v[[plist[[i]]]]}];
  ];

  Return[d];

  (* Remove the zero vectors *)
  (*
  b={};

  For[i=1,i<=Length[pb],i++,
    If[!SameQ[Norm[pb[[i]]],0],
      b=Join[b,{pb[[i]]}];
    ];
  ];

  Return[b];
  *)
];

MakeBasis[a___] := InvalidArg["MakeBasis", a];
```

12.2.24 MakeRootBasis

MakeRootBasis::usage=

"MakeRootBasis[r] will return a basis for the collection of root vectors r. MakeRootBasis preserves the condition that one root is double (or hence, half) that of another."

```
MakeRootBasis[v_?MatrixQ] := Module[
  {basis, rbasis, i, j, bc},

  (* First make a 'preliminary' basis *)
  rbasis = MakeBasis[v];
  basis = rbasis;
  bc = False;

  (* Check if any roots are 2x multiples of each other. *)
  For[i = 1, i <= Length[v], i++,
    For[j = 1, j <= Length[v], j++,
      If[SameQ[v[[i]], 2*v[[j]]] || SameQ[2*v[[i]], v[[j]]],
        bc = True;
      ];
    ];

  (* Preserve BC: Next,
  if any roots in the given system are twice that of a basis vector,
  add it in. *)
  If[bc,
    For[i = 1, i <= Length[rbasis], i++, (* For all basis vectors *)

      For[j = 1, j <= Length[v], j++, (* For all rots *)

        If[SameQ[rbasis[[i]], 2*v[[j]]] ||
          SameQ[2*rbasis[[i]], v[[j]]],
          basis = Join[basis, {v[[j]]}];
        ];
      ];
    ];

  Return[basis];
];

MakeRootBasis[a___] := InvalidArg["MakeRootBasis", a];
```

12.2.25 OperatorMatrixFromFunction

OperatorMatrixFromFunction::usage=

”OperatorMatrixFromFunction[d,fn] applies each vector in a basis d to a function fn and returns the matrix for the corresponding linear operator.”;

```
OperatorMatrixFromFunction[d_?ListQ, fn_] := Module[
  {i, v},

  For[i = 1, i <= Length[d], i++,
    (*v[i]=alphaForm[d,fn[d[[i]]]];*)

    v[i] = BasisCoefficients[d, fn[d[[i]]]];
  ];

  Return[Transpose[Table[v[i], {i, 1, Length[d]}]]];
];

OperatorMatrixFromFunction[a_...] :=
  InvalidArg["OperatorMatrixFromFunction", a];
```


12.2.26 PositiveRootSystem

PositiveRootSystem::usage=

"PositiveRootSystem[r,roots] returns the positive roots in the set 'roots' under root system r.

PositiveRootSystem[r] returns all positive roots in the root system r."

```
PositiveRootSystem[r_?RootInputQ, {argroots_?VectorQ}] :=
  PositiveRootSystem[RootBase[r], List[argroots]];

PositiveRootSystem[d_?RootBasisQ, {argroots_?VectorQ}] := Module[
  {roots, posroots, ht, i},

  roots = List[argroots];
  posroots = {};

  For[i = 1, i <= Length[roots], i++,
    ht = RootHeight[d, roots[[i]]];

    If[ht > 0,
      posroots = Join[posroots, {roots[[i]]}];
    ];
  ];

  Return[posroots];
];

PositiveRootSystem[r_?RootInputQ] := PositiveRootSystem[RootBase[r]];

PositiveRootSystem[d_?RootBasisQ] := Module[
  {k, p, n, h, i, j, r, t, q, s, size, time, cntr, m},

  (* 1. Init *)
  p = d;
  n = 1;

  m = CartanMatrixFromBasis[d];

  size = 1;

  time = TimeUsed[];
  cntr = time;

  (* 2. Construct *)
  While[Length[p] > size,
    size = Length[p];

    For[i = 1, i <= Length[p], i++,
      For[j = 1, j <= Length[d], j++,
        t = p[[i]];

        (* 2.1.
        Get the basis coefficients and check height of the root *)

        k = BasisCoefficients[d, t];
        h = Sum[k[[s]], {s, 1, Length[k]}];

        If[h == n,
          (* 2.2. Determine the integer r *)
```

```

r = 0;
While[MemberQ[p, t - r*d[[j]]], r++];
r = r - 1;

(* 2.3. Define q *)
q = r - Sum[k[[s]]*m[[s, j]], {s, 1, Length[d]}];

(* 2.4. Ammend p? *)
If[q > 0,
  p = Union[p, {t + d[[j]]}];
];

(*
If[TimeUsed[]-cntr>10,
Print["(rootSystem) CPU Time: [", TimeUsed[]-time, ""] Roots: [",
2*Length[p], ""];
cntr=TimeUsed[];
];
*)
];
];

n = n + 1;
];

Return[p];
];

PositiveRootSystem[a_...] := InvalidArg["PositiveRootSystem", a];

```

12.2.27 PrintMatrixArray

PrintMatrixArray::usage=

"PrintMatrixArray[l] creates a human-readable table of a set of matrices l."

```
PrintMatrixArray[{larg_?MatrixQ}] := Module[{i, l},
  l = List[larg];

  Table[l[[i]] // MatrixForm, {i, 1, Length[l]}]
];

PrintMatrixArray[a_...] := InvalidArg["PrintMatrixArray", a];
```

12.2.28 Reflect

Reflect::usage=

”Reflect[a,b] computes the reflection of the vector b across the hyperplane formed by a.

Reflect[a] computes the reflection of vector a across its own hyperplane.

Reflect[aList,b,rev] computes the reflection of b across every vector listed in the ordered list aList. The optional argument rev, if True, will iterate through the list aList backward.

Reflect[a,bList] computes the set of vectors formed by reflecting each vector in bList across a.

Reflect[aList,bList,rev] computes the set of vectors formed by reflecting each vector in bList across every vector in the ordered list aList. The optional argument rev, if True, will iterate through the list aList backward.”;

```

Reflect[a_?VectorQ] := Reflect[a, a];

Reflect[a_?VectorQ, b_?VectorQ] := Module[
  {},

  b - 2*(InnerProduct[a, b]/InnerProduct[a, a])*a
];

Reflect[{arga_?VectorQ}, b_?VectorQ, reverse_: False] := Module[
  {i, v, a},

  a = List[arga];

  If[reverse,
    For[i = Length[a], i >= 1, i--,
      v = Reflect[a[[i]], b];
    ],
    For[i = 1, i <= Length[a], i++,
      v = Reflect[a[[i]], b];
    ];

  Return[v];
];

Reflect[a_?VectorQ, {argb_?VectorQ}] := Module[
  {i, b},

  b = List[argb];

  Return[Table[Reflect[a, b[[i]]], {i, 1, Length[b]}]];
];

Reflect[{arga_?VectorQ}, {argb_?VectorQ}, reverse_: False] := Module[
  {i, a, b},

  a = List[arga];

```

```
b = List[argv];  
  
Return[Table[Reflect[a, b[[i]], reverse], {i, 1, Length[b]}]];  
];  
  
Reflect[a_...] := InvalidArg["Reflect", a];
```

12.2.29 RestrictedRootAut

RestrictedRootAut::usage=

”RestrictedRootAut[r,theta,sub] computes the matrix for an automorphism theta on the root system r, restricted to the given sub-basis.”;

```
RestrictedRootAut[r_?RootInputQ, theta_?MatrixQ, sub_?RootBasisQ] :=
  RestrictedRootAut[RootBase[r], theta, sub];

RestrictedRootAut[basis_?RootBasisQ, theta_?MatrixQ,
  sub_?RootBasisQ] := Module[
  {rMatr, i, keeplist},

  keeplist = {};

  For[i = 1, i <= Length[basis], i++,
    If[MemberQ[sub, basis[[i]]],
      keeplist = Join[keeplist, {UnitVector[Length[basis], i]}];
    ];

  (*{{0,1,0,0},{0,0,0,1}}.m.Transpose[{{1,0,0,0},{0,0,1,0}}]//
  MatrixForm
  keeps rows 2 and 4 of m, and columns 1 and 3. *)

  rMatr = keeplist.theta.Transpose[keeplist];

  Return[rMatr];
  ];

RestrictedRootAut[a___] := InvalidArg["RestrictedRootAut", a];
```

12.2.30 RootAlphaForm

RootAlphaForm::usage=

"RootAlphaForm[d,r] gives the coordinates of a root or set of roots r with respect to basis d. (Writes root r in alpha form).";

```
RootAlphaForm[basis_?RootBasisQ, roots_?ListQ] := Module[
  {i},
  If[SameQ[Depth[roots], 3],
    Return[
      Table[BasisCoefficients[basis, roots[[i]]], {i, 1,
        Length[roots]}]];
  ,
  Return[BasisCoefficients[basis, roots]];
];

RootAlphaForm[a___] := InvalidArg["RootAlphaForm", a];
```

12.2.31 RootBase

RootBase::usage=

"RootBase[r] gives a basis for a root system r."

```
RootBase[r_?RootInputQ] := Module[
  {i, j, basis, sbasis, nbasis, totalLen, vLen, rin},

  rin = RootInput[r];
  basis = {};

  (* Build all the simple bases *)

  For[i = 1, i <= Length[rin], i++,
    sbasis[i] = SimpleRootBase[rin[[i]]];
  ];

  (* Find the total vector length needed *)
  totalLen = 0;
  vLen = 0;

  For[i = 1, i <= Length[rin], i++,
    totalLen = totalLen + Length[sbasis[i][[1]]];
  ];

  (* Pad. For each simple basis, make a new basis with padding *)

  For[i = 1, i <= Length[rin], i++,
    nbasis =
      Table[VectorPad[sbasis[i][[j]], vLen, totalLen], {j, 1,
        Length[sbasis[i]]}];
    basis = Join[basis, nbasis];

    vLen = vLen + Length[sbasis[i][[1]]];
  ];

  Return[basis];
];

RootBase[a___] := InvalidArg["RootBase", a];
```


12.2.32 RootBasisConnectedSet

RootBasisConnectedSet::usage=

"RootBasisConnectedSet[basis] returns a set of lists of basis vectors. The lists group together basis roots which are members of the same irreducible root subsystem.";

```
RootBasisConnectedSet[r_?RootInputQ] :=
  RootBasisConnectedSet[RootBase[r]];

RootBasisConnectedSet[d_?ListQ] := Module[
  {group, gnum, i, todo, didchange, j},

  If[SameQ[d, {}],
    Return[{}];
  ];

  todo = d; (* Every root needs to be processed *)

  gnum = 0; (* Start with group 1 *)

  While[! SameQ[todo, {}], (* Process every root *)
    (*
    Start a new group *)
    gnum++;
    group[gnum] = {todo[[1]]};
    todo = Complement[todo, {todo[[1]]}];

    didchange = True;

    While[didchange,
      didchange = False; (* If we add a new root to the group,
        flip this to indicate another run needed *)

      For[i = 1, i <= Length[todo], i++,
        (* If (root i, root j) is not zero for some j,
        then add it to the group *)

        For[j = 1, j <= Length[group[gnum]], j++,
          If[InnerProduct[todo[[i]], group[gnum][[j]]] != 0,
            didchange = True;
            group[gnum] = Join[group[gnum], {todo[[i]]}];
            todo = Complement[todo, {todo[[i]]}];
            Break[];
          ]; (* END IF *)
        ]; (* END FOR j *)

        If[didchange, Break[]]; (*
        Need to reset the iteration due to todo changing *)
      ]; (*
      END FOR i *)
    ];

    Return[Table[group[i], {i, 1, gnum}]];
  ];

RootBasisConnectedSet[a_...] := InvalidArg["RootBasisConnectedSet", a];

(* Bad strategy. Use inner products.;
RootBasisConnectedSet[d_?ListQ]:=Module[
```

```

{blist,cMatr,set,lens,i,j,iset,cntr},

If[SameQ[d,{}],Return[{}]];

cMatr=CartanMatrix[d];
blist=BlockList[cMatr];

(* size of each irred component *)
lens={};
For[i=1,i<=Length[blist],i++,
lens=Join[lens,{Length[blist[[i]]]}];
];

(* list of indices *)
cntr = 1;
For[i=1,i<=Length[lens],i++, (* FOR every irred component *)
set[i]={};

For[j=1,j<=lens[[i]],j++, (* FOR every root in irred component *)
\
set[i]=Join[set[i],{cntr}];
cntr++;
];
];

Return[Table[Table[d[[set[i]][[j]]]],{j,1,Length[set[i]]},{i,1,Length[\
lens]}]];
];
*)

RootBasisConnectedSet[a___] := InvalidArg["RootBasisConnectedSet", a];

```

12.2.33 RootBasisConnectedQ

RootBasisConnectedQ::usage=

”RootBasisConnectedQ[basis,a,b] returns True if two basis vectors a and b in the supplied basis reside in the same irreducible root system.”;

```
RootBasisConnectedQ[d_?ListQ, rlw_?VectorQ, eaw_?VectorQ] := Module[
  {bcs, index},

  bcs = RootBasisConnectedSet[d];

  If[! MemberQ[d, rlw] || ! MemberQ[d, eaw], Return[Fail]];

  For[index = 1, index <= Length[bcs], index++,
    If[MemberQ[bcs[[index]], rlw],
      Break[];
    ];
  ];

  Return[MemberQ[bcs[[index]], eaw]];
];

RootBasisConnectedQ[a___] := InvalidArg["IsRootBasisConnected", a];
```

12.2.34 RootBasisQ

```

RootBasisQ::usage=
"RootBasisQ[r] returns True if r is the basis of a root system.";

RootBasisQ::basis = "Supplied set is not a root system basis:\n'1'";

RootBasisQ[d_?StringQ] := False;

RootBasisQ[d_?MatrixQ] := Module[
  {},

  If[RootInputQ[d],
    Return[False];
  ];

  If[SameQ[Length[d], Length[MakeRootBasis[d]]],
    Return[True];
  ,
    Message[RootBasisQ::basis, d];
    Return[False];
  ];
];

RootBasisQ[a___] := False;

RootBasisQuietQ[d_?MatrixQ] := Module[
  {},
  If[SameQ[Length[d], Length[MakeRootBasis[d]]],
    Return[True];
  ,
    Return[False];
  ];
];

RootBasisQuietQ[a___] := False;

```

12.2.35 RootCoBase

RootCoBase::usage=

"RootCoBase[r] gives a co-basis for a root system r."

```
RootCoBase[r_?RootInputQ] := Module[{basis, i},
  basis = RootBase[r];

  Table[2*basis[[i]]/InnerProduct[basis[[i]], basis[[i]]], {i, 1,
    Length[basis]}]
];

RootCoBase[a_...] := InvalidArg["RootCoBase", a];
```

12.2.36 RootDecomposition

RootDecomposition::usage=

"RootDecomposition[basis] returns two tables TABLE A, TABLE B. The first table is a list of all roots in a root system with the given basis. The second table is a list pairs of roots. The i entry in table B is a set of two roots of a lower level such that they sum to the root given in the i entry of table A. The table is not unique.";

```
RootDecomposition[d_?RootBasisQ] := Module[
  {k, p, n, h, i, j, r, t, q, s, m, size, time, cntr, at, attable,
   roots},

  (* 1. Init *)
  p = d;
  n = 1;

  size = 1;

  time = TimeUsed[];
  cntr = time;

  m = CartanMatrixFromBasis[d];

  (* Initialize AT *)
  For[i = 1, i <= Length[d], i++,
    at[d[[i]]] = {d[[i]], 0};
    at[-d[[i]]] = {-d[[i]], 0};
  ];

  (* 2. Construct *)
  While[Length[p] > size,
    size = Length[p];

    For[i = 1, i <= Length[p], i++,
      For[j = 1, j <= Length[d], j++,
        t = p[[i]];

        (* 2.1.
         Get the baiss coefficients and check height of the root *)

        k = BasisCoefficients[d, t];
        h = Sum[k[[s]], {s, 1, Length[k]}];

        If[h == n,
          (* 2.2. Determine the integer r *)
          r = 0;
          While[MemberQ[p, t - r*d[[j]]], r++];
          r = r - 1;

          (* 2.3. Define q *)

          q = r - Sum[k[[s]]*m[[s, j]], {s, 1, Length[d]}];

          (* 2.4. Ammend p? *)
          If[q > 0,
            p = Union[p, {t + d[[j]]}];

            (* AT version: data for addition table *)
```

```

    at[t + d[[j]]] = {d[[j]], t};
    at[-t - d[[j]]] = {-d[[j]], -t};

    (*Print["root=",t+d[[j]],"  table entry=",at[t+d[[j]]],
      "  d[[j]]=",d[[j]],"  t=",t];*)
  ];
  (*
  If[TimeUsed[]-cntr>10,
  Print["(rootSystemAT) CPU Time:",TimeUsed[]-time,
  "  Roots:",2*Length[p],""];
  cntr=TimeUsed[];
  ];
  *)
];
];

n = n + 1;
];

(* If[TimeUsed[]-time>10,
  Print["(rootSystemAT) CPU Time:",TimeUsed[]-time,"  Roots:",2*
  Length[p],"  (done)"];
]; *)

roots = Join[p, -p];

(* AT version: construct addition table *)

atable = Table[at[roots[[i]]], {i, 1, Length[roots]}];

(*
Print["(rootSystemAT) Used ",TimeUsed[]-time,
"s CPU Time to find ",2*Length[p],
" roots and construct the addition table."];
*)

(* RETURN *)
{roots, atable}
];

RootDecomposition[a___] := InvalidArg["RootDecomposition", a];

```

12.2.37 RootFunctional

RootFunctional::usage=

”RootFunctional[r,a,tv] computes $a(tv)$ where a is a root and tv is a vector in the Cartan subalgebra (given as coordinates with respect to basis of the root system r).”;

```
RootFunctional[r_?RootInputQ, root_?VectorQ, tv_?VectorQ] :=
  RootFunctional[RootBase[r], root, tv];

RootFunctional[d_?RootBasisQ, root_?VectorQ, tv_?VectorQ] := Module[
  {kf, i},

  (* oops! Backward...
  kf=2*innerProduct[tv.d,root]/innerProduct[
  root,root];
  *)
  kf =
    Sum[tv[[i]]*2*
      InnerProduct[root, d[[i]]]/InnerProduct[d[[i]], d[[i]]], {i, 1,
      Length[d]};

  Return[kf];
];

RootFunctional[a_...] := InvalidArg["RootFunctional", a];
```


12.2.38 RootHeight

RootHeight::usage=

”RootHeight[d,r] gives the height of root r in the root system with basis d.”;

```
RootHeight[d_?RootBasisQ, root_?VectorQ] := Module[{k, i},
  k = BasisCoefficients[d, root];

  Sum[k[[i]], {i, 1, Length[k]}]
];

RootHeight[a___] := InvalidArg["RootHeight", a];
```

12.2.39 RootLessQ

RootLessQ::usage=

"RootLessQ[r,a,b] returns True if root a \preceq root b with respect to root ordering and root system r.";

```
RootLessQ[r_?RootInputQ, a_?VectorQ, b_?VectorQ] :=
  RootLessQ[RootBase[r], a, b];

RootLessQ[d_?RootBasisQ, a_?VectorQ, b_?VectorQ] := Module[
  {hta, htb},

  hta = RootHeight[d, a];
  htb = RootHeight[d, b];

  If[hta < htb,
    Return[True];
  ];

  If[hta > htb,
    Return[False];
  ];

  Return[BasisOrder[a, b]];
];

RootLessQ[a_...] := InvalidArg["RootLessQ", a];
```

12.2.40 RootSplit

RootSplit::usage=

"RootSplit[d,r,forceb] returns for a root r in a system with basis d the pair a,b so that $r = a + b$, where b is a basis root (if r is $\neq 0$), or the negative of a basis root (if r ≤ 0), and a is a shorter root. If r is height 1 or -1, then returned is 0,r. The optional argument forceb ensures that b is always a basis element (for the case that the root is negative), or that 0,r is returned.";

```
RootSplit[d_?RootBasisQ, r_?VectorQ, forceb_: False] := Module[
  {i},

  If[RootHeight[d, r] > 0 || forceb, (* POSITIVE ROOT *)
    (*
      Iterate backward so that we can easily decompose a root into sums \
      of basis elements in lexicographic order. *)

    For[i = Length[d], i >= 1, i--,

      (* Lemma 9.4 in Humphreys *)

      If[InnerProduct[r, d[[i]]] > 0,
        Return[{r - d[[i]], d[[i]]}];
      ];
    ], (* NEGATIVE ROOT *)
    (*
      Iterate backward so that we can easily decompose a root into sums \
      of basis elements in lexicographic order. *)

    For[i = Length[d], i >= 1, i--,

      (* Lemma 9.4 in Humphreys *)

      If[InnerProduct[r, -d[[i]]] > 0,
        Return[{r + d[[i]], -d[[i]]}];
      ];
    ];

  Return[{0, r}];
];

RootSplit[a___] := InvalidArg["RootSplit", a];
```

12.2.41 RootString

RootString::usage=

”RootString[roots,a,b], where roots is the set of all roots in a root system, a and b are two roots, will compute the a-string through b”;

```
RootString[{argroots_?VectorQ}, a_?VectorQ, b_?VectorQ] := Module[
  {roots, bounds, rstr, i},

  roots = List[argroots];

  rstr = {};
  bounds = RootStringBounds[roots, a, b];

  For[i = -bounds[[1]], i <= bounds[[2]], i++,
    rstr = Join[rstr, {b + i*a}];
  ];

  Return[rstr];
];

RootString[a_...] := InvalidArg["RootString", a];
```

12.2.42 RootStringBounds

RootStringBounds::usage=

"RootStringBounds[roots,a,b], where roots is the set of all roots in a root system, a and b are two roots, will compute positive integers p,q where $b + ka - p$ is the a-string through b";

```
RootStringBounds[{argroots_?VectorQ}, a_?VectorQ, b_?VectorQ] :=
Module[
  {p, q, roots, i, k},
  roots = List[argroots];

  (* Find -p *)
  k = 0;

  While[MemberQ[roots, b + k*a],
    k--;
  ];

  (* b + ka is no longer a root. Add one back to k so that it is *)

  k++;

  (* Found -p *)
  p = k;

  (* Find q *)
  k = 0;

  While[MemberQ[roots, b + k*a],
    k++;
  ];

  (* b + ka is no longer a root. Add one back to k so that it is *)

  k--;

  (* Found q *)
  q = k;

  (* Return -
  p because we want to return the postive integer for the lower \
  bound *)
  Return[{-p, q}];
];

RootStringBounds[a_...] := InvalidArg["RootStringBounds", a];
```

12.2.43 RootSumPath

RootSumPath::usage=

"RootSumPath[basis,addtable,root] computes a sequence of additions to root space vectors such that: the start of the sequence is a simple root, each intermediate step is a root, and the end result is the supplied root. The root system has the supplied basis. The argument addtable is "Table B" returned by RootDecomposition. The sequence will correlate to that used by RootSystem.

RootSumPath[basis,root] computes a sequence of additions to root space vectors such that: the start of the sequence is a simple root, each intermediate step is a root, and the end result is the supplied root. The root system has the supplied basis. The sequence may not correlate to the unique sequence produced by RootDecomposition, but does not require computing the addition table.";

```
RootSumPath[d_?RootBasisQ, at_?ListQ, root_?VectorQ] :=
  RootSumPath[d, RootSystemFromBasis[d], at, root];

RootSumPath[d_?RootBasisQ, roots_?ListQ, at_?ListQ, root_?VectorQ] :=
  Module[{ret},
    If[MemberQ[Union[d, -d], root],
      ret = root;
    ,
      (*ret={atSplit[d,roots,at,at[[i]][[1]]],atSplit[d,roots,at,at[[
      i]][[2]]];*)

      ret = {RootSumPath[d, roots, at,
        at[[Position[roots, root][[1, 1]], 1]]],
        RootSumPath[d, roots, at,
        at[[Position[roots, root][[1, 1]], 2]]]};
    ];

    ret
  ];

RootSumPath[d_?RootBasisQ, root_?VectorQ] :=
  CreateDialog[{TextCell[
    "TO DO: Implement root sum path using Humphreys Lemma (ch9)",
    DefaultButton[]]};

RootSumPath[a_...] := InvalidArg["RootSumPath", a];
```

12.2.44 RootSystem

RootSystem::usage=

”RootSystem[r] gives the roots for root system r.”;

```
RootSystem[r_?RootInputQ] := RootSystemFromBasis[RootBase[r]];
```

```
RootSystem[r_?RootBasisQ] := RootSystemFromBasis[r];
```

```
RootSystem[a___] := InvalidArg["RootSystem", a];
```

12.2.45 RootToString

RootToString::usage=

”RootToString[r] converts from r, the list form representation of a root system, to the human-readable string form. r is of the form A,n,B,n,C,n,... where A, B, C are the root system type (A-G), n is an integer, representing the root system $A_n+B_n+C_n+\dots$ ”;

```
RootToString[r_?ListQ] := Module[
  {i, str},

  (* If user forgets double brace on an irreducible system... e.g.
  passes {A,4} instead of {{A,4}} *)

  If[SameQ[Depth[r], 2], Return[r[[1]] <> ToString[r[[2]]]];];

  str = "";

  For[i = 1, i <= Length[r], i++,
    If[i > 1, str = str <> "+"];];

  str = str <> r[[i, 1]] <> ToString[r[[i, 2]]];
  ];

  Return[str];
];

RootToString[a___] := InvalidArg["RootToString", a];
```


12.2.46 StructureConstantsFromBasis

StructureConstantsFromBasis::usage=

”StructureConstantsFromBasis[r,sc,nconsts,theta] returns a list of structure constants for an automorphism over the Lie algebra with root system r and root involution theta. nconsts supplies the Chevalley constants. sc supplies the structure constants for the basis roots. The table returned is a list of elements of the form ROOT, CONSTANT. If nconsts is omitted, the procedure KleinChevalley will be called. However, for repeated usage it is recommended to compute once and store the Chevalley constants in memory.”;

```
StructureConstantsFromBasis::badroot = "N_‘1‘,‘2‘ is zero.";

StructureConstantsFromBasis[d_?RootBasisQ, sc_?VectorQ,
  theta_?MatrixQ] :=
  StructureConstantsFromBasis[d, sc, KleinChevalley[d], theta];

StructureConstantsFromBasis[r_?RootInputQ, sc_?VectorQ,
  theta_?MatrixQ] := Module[
  {d},

  d = RootBase[r];
  Return[
    StructureConstantsFromBasis[d, sc, KleinChevalley[d], theta]];
];

StructureConstantsFromBasis[r_?RootInputQ, sc_?VectorQ,
  nconsts_?ListQ, theta_?MatrixQ] :=
  StructureConstantsFromBasis[RootBase[r], sc, nconsts, theta];

(* remove the variable c_ and return a list described in the TODO \
section *)
StructureConstantsFromBasis[d_?RootBasisQ, sc_?VectorQ,
  nconsts_?ListQ, theta_?MatrixQ] := Module[
  {k, p, n, h, i, j, r, t, q, s, size, time, cntr, m,
   root1, root2, r1, r2, r3, r4, num, b, nn, ao, ap, c, roots, cTbl,
   nc12, nc34},

  (* 0. Init *)
  p = d;
  n = 1;

  size = 1;

  time = TimeUsed[];
  cntr = time;

  m = CartanMatrixFromBasis[d];

  (* 1. Basis *)
  For[i = 1, i <= Length[d], i++,
    c[d[[i]]] = sc[[i]];
    c[-d[[i]]] = Power[sc[[i]], -1];
  ];

  (*Print["OK"];*)
  (* 2. Construct *)
```

```

While[Length[p] > size,
  size = Length[p];

For[i = 1, i <= Length[p], i++,
  For[j = 1, j <= Length[d], j++,
    t = p[[i]];

    (* 2.1.
    Get the basis coefficients and check height of the root *)

    k = BasisCoefficients[d, t];
    h = Sum[k[[s]], {s, 1, Length[k]}];

    If[h == n,
      (* 2.2. Determine the integer r *)
      r = 0;
      While[MemberQ[p, t - r*d[[j]]], r++];
      r = r - 1;

      (* 2.3. Define q *)

      q = r - Sum[k[[s]]*m[[s, j]], {s, 1, Length[d]}];

      (* 2.4. Amend p? *)
      If[q > 0,
        p = Union[p, {t + d[[j]]}];

        (* Write the root in "[Alpha] notation" *)

        root1 = BasisCoefficients[d, d[[j]]];
        root2 = BasisCoefficients[d, t];

        (* Apply theta and convert back to "[Epsilon], i]
        notation" *)
        r1 = d[[j]];
        r2 = t;
        r3 = root1.Transpose[theta].d;
        r4 = root2.Transpose[theta].d;

        nc12 = ChevalleyLookup[nconsts, r1, r2];
        nc34 = ChevalleyLookup[nconsts, r3, r4];

        If[nc12 == 0,
          Message[StructureConstantsFromBasis::badroot, r1, r2]];
        If[nc34 == 0,
          Message[StructureConstantsFromBasis::badroot, r3, r4]];

        c[t + d[[j]]] = (nc34/nc12)*c[t]*c[d[[j]]];

        c[-(t + d[[j]])] = 1/c[t + d[[j]]];
      ];
    ];
  ];

  n = n + 1;
];

(* p is the set of all positive roots. Return p and -
p along with the structure constants *)
roots = Join[p, -p];

```

```

cTbl = {};
For[i = 1, i <= Length[roots], i++,
  cTbl = Join[cTbl, {{roots[[i]], c[roots[[i]]]}},
];

Return[cTbl];
];

StructureConstantsFromBasis[a___] :=
  InvalidArg["StructureConstantsFromBasis", a];

```

12.2.47 StructureConstantsLookup

StructureConstantsLookup::usage=

”StructureConstantsLookup[cconsts,a] looks up the structure constant with respect to the root a. cconsts is a table of the form returned by StructureConstants.”;

```
StructureConstantsLookup::noroot =
  "Root '1' not found in supplied table.";

StructureConstantsLookup[cconsts_?ListQ, a_?VectorQ] := Module[
  {i},

  For[i = 1, i <= Length[cconsts], i++,
    If[SameQ[cconsts[[i, 1]], a],
      Return[cconsts[[i, 2]]];
    ];
  ];

  Message[StructureConstantsLookup::noroot, a];

  Return[{}];
];

StructureConstantsLookup[a_...] :=
  InvalidArg["StructureConstantsLookup", a];
```

12.3 Root System and Lie Algebra Package (Diagram)

12.3.1 DrawRootSystem

DrawRootSystem::usage=

”DrawRootSystem[r,force] draws a 2D or 3D plot of all roots of a root system r that is of dimension at most 3. Basis lines are represented in bold. The optional argument force, if True, forces drawRootSystem to establish the diagram in a 3D plane. Because a Graphics object is returned, this is useful for combining 2D and 3D plots.”;

```
DrawRootSystem::dim = "dimension of '1' is too high. Max is 3.";

DrawRootSystem[basis_?RootBasisQ, force_: False] := Module[
  {rin, roots, dim, rootlines, basislines, g1, g2},

  dim = Length[basis[[1]]];

  If[dim > 3,
    Message[DrawRootSystem::dim, dim];
  ];

  roots = RootSystem[basis];

  Switch[dim,
    1,
```

```

If[force,
  basislines =
    Table[Line[{{0, 0, 0}, {basis[[i, 1]], 0, 0}}, {i, 1,
      Length[basis]}];
  rootlines =
    Table[Line[{{0, 0, 0}, {roots[[i, 1]], 0, 0}}, {i, 1,
      Length[roots]}];
  g1 = Graphics3D[{Cyan, rootlines}];
  g2 = Graphics3D[{Thick, Cyan, basislines}];
  ,
  basislines =
    Table[Line[{{0, 0}, {basis[[i, 1]], 0}}, {i, 1,
      Length[basis]}];
  rootlines =
    Table[Line[{{0, 0}, {roots[[i, 1]], 0}}, {i, 1,
      Length[roots]}];
  g1 = Graphics[{Cyan, rootlines}];
  g2 = Graphics[{Thick, Cyan, basislines}];
  ];
,
2,
If[force,
  basislines =
    Table[Line[{{0, 0, 0}, {basis[[i, 1]], basis[[i, 2]], 0}}, {i,
      1, Length[basis]}];
  rootlines =
    Table[Line[{{0, 0, 0}, {roots[[i, 1]], roots[[i, 2]], 0}}, {i,
      1, Length[roots]}];
  g1 = Graphics3D[{Cyan, rootlines}];
  g2 = Graphics3D[{Thick, Cyan, basislines}];
  ,
  basislines =
    Table[Line[{{0, 0}, {basis[[i, 1]], basis[[i, 2]]}}, {i, 1,
      Length[basis]}];
  rootlines =
    Table[Line[{{0, 0}, {roots[[i, 1]], roots[[i, 2]]}}, {i, 1,
      Length[roots]}];
  g1 = Graphics[{Cyan, rootlines}];
  g2 = Graphics[{Thick, Cyan, basislines}];
  ];
,
3,
basislines =
  Table[Line[{{0, 0, 0}, {basis[[i, 1]], basis[[i, 2]],
    basis[[i, 3]]}}, {i, 1, Length[basis]}];
rootlines =
  Table[Line[{{0, 0, 0}, {roots[[i, 1]], roots[[i, 2]],
    roots[[i, 3]]}}, {i, 1, Length[roots]}];
g1 = Graphics3D[{Cyan, rootlines}];
g2 = Graphics3D[{Thick, Red, basislines}];
];

Show[g1, g2, Background -> Black]
];

DrawRootSystem[r_?RootInputQ, force_: False] := Module[
  {rin, basis, roots, dim, rootlines, basislines, g1, g2},

  rin = RootInput[r];
  basis = RootBase[r];
  dim = Length[basis[[1]]];

```

```

If[dim > 3,
  Message[DrawRootSystem::dim, dim];
];

roots = RootSystem[r];

Switch[dim,
  1,
  If[force,
    basislines =
      Table[Line[{{0, 0, 0}, {basis[[i, 1]], 0, 0}}, {i, 1,
        Length[basis]}];
    rootlines =
      Table[Line[{{0, 0, 0}, {roots[[i, 1]], 0, 0}}, {i, 1,
        Length[roots]}];
    g1 = Graphics3D[{Cyan, rootlines}];
    g2 = Graphics3D[{Thick, Cyan, basislines}];
    ,
    basislines =
      Table[Line[{{0, 0}, {basis[[i, 1]], 0}}, {i, 1,
        Length[basis]}];
    rootlines =
      Table[Line[{{0, 0}, {roots[[i, 1]], 0}}, {i, 1,
        Length[roots]}];
    g1 = Graphics[{Cyan, rootlines}];
    g2 = Graphics[{Thick, Cyan, basislines}];
    ];
  ,
  2,
  If[force,
    basislines =
      Table[Line[{{0, 0, 0}, {basis[[i, 1]], basis[[i, 2]], 0}}, {i,
        1, Length[basis]}];
    rootlines =
      Table[Line[{{0, 0, 0}, {roots[[i, 1]], roots[[i, 2]], 0}}, {i,
        1, Length[roots]}];
    g1 = Graphics3D[{Cyan, rootlines}];
    g2 = Graphics3D[{Thick, Cyan, basislines}];
    ,
    basislines =
      Table[Line[{{0, 0}, {basis[[i, 1]], basis[[i, 2]]}}, {i, 1,
        Length[basis]}];
    rootlines =
      Table[Line[{{0, 0}, {roots[[i, 1]], roots[[i, 2]]}}, {i, 1,
        Length[roots]}];
    g1 = Graphics[{Cyan, rootlines}];
    g2 = Graphics[{Thick, Cyan, basislines}];
    ];
  ,
  3,
  basislines =
    Table[Line[{{0, 0, 0}, {basis[[i, 1]], basis[[i, 2]],
      basis[[i, 3]]}}, {i, 1, Length[basis]}];
  rootlines =
    Table[Line[{{0, 0, 0}, {roots[[i, 1]], roots[[i, 2]],
      roots[[i, 3]]}}, {i, 1, Length[roots]}];
  g1 = Graphics3D[{Cyan, rootlines}];
  g2 = Graphics3D[{Thick, Red, basislines}];
  ];

Show[g1, g2, Background -> Black]
];

```

```
DrawRootSystem[a___] := InvalidArg["DrawRootSystem", a];
```

12.3.2 DynkinDiagram

DynkinDiagram::usage=

”DynkinDiagram[r] gives the Dynkin Diagram of root system r.”;

```

DynkinDiagram[d_?RootBasisQ, labels_: {}] :=
  DynkinDiagram[BasisToRootSystem[d], labels];

DynkinDiagram[r_?RootInputQ, labels_: {}] := Module[
  {rin, i, gr, toff, tht, biggest, mywidth, g},

  rin = RootInput[r];

  gr = {};
  toff = 0;
  tht = 0;
  biggest = 0;

  For[i = 1, i <= Length[rin], i++,
    mywidth = DynkinWidth[rin[[i]]];

    If[mywidth > biggest,
      biggest = mywidth;
    ];
  ];

  For[i = 1, i <= Length[rin], i++,
    mywidth = DynkinWidth[rin[[i]]];

    If[i > 1,
      tht = tht - DynkinHeight[rin[[i]]];
    ];

    gr = Join[gr,
      LittleDynk[rin[[i]], (biggest - mywidth)/2, tht, toff,
        labels]];
    toff = toff + rin[[i, 2]];
  ];

  g = Graphics[gr];

  Return[g];
];

DynkinDiagram[a___] := InvalidArg["DynkinDiagram", a];

```


12.3.3 gInvolutionDiagram

gInvolutionDiagram::usage=

”gInvolutionDiagram[r,theta,cvals] extends the Helminck diagram with the values c and k necessary to recover the structure constants of an involution on the Lie algebra. Each basis root is labelled with c,k. The minimal polynomial of the corresponding structure constant is $1x^2 + kx + c$. r is the root system, theta the root system automorphism, and cvals is the list of structure constants.

(S-LOSS) gInvolutionDiagram[r,disks,arches,cvals] extends the Helminck diagram with the values c and k necessary to recover the structure constants of an involution on the Lie algebra. Each basis root is labelled with c,k. The minimal polynomial of the corresponding structure constant is $1x^2 + kx + c$. r is the root system, disks labels the fixed roots, arches represents the diagram automorphism, and cvals is the list of structure constants.”;

```
gInvolutionDiagram[d_?RootBasisQ, theta_?MatrixQ, cvals_?ListQ] :=
Module[
{cv},

cv = cvMinimalPolynomialList[d, cvals];

HelminckDiagram[d, theta, cv]
];

gInvolutionDiagram[d_?RootInputQ, theta_?MatrixQ, cvals_?ListQ] :=
Module[
{cv},

cv = cvMinimalPolynomialList[d, cvals];

HelminckDiagram[d, theta, cv]
];

gInvolutionDiagram[a___] := InvalidArg["gInvolutionDiagram", a];
```

12.4 Root System and Lie Algebra Package (Internal)

12.4.1 BasisCoefficientsMatrix

BasisCoefficientsMatrix::usage=

”BasisCoefficientsMatrix[b,x] gives the coordinates of x with respect to basis b. x must be a matrix.”;

```
BasisCoefficientsMatrix[b_?ListQ, x_?MatrixQ] := Module[
```

```

{k, i, bstd, bm, c},

(* 1. Write x in standard basis *)
k = x // Flatten;

(* 2. Write all the basis vectors in standard basis *)

For[i = 1, i <= Length[b], i++,
  bstd[i] = b[[i]] // Flatten;
];

(* 3. Form a matrix with bstd as rows *)

bm = Table[bstd[i], {i, 1, Length[b]}];

(* 4. Take the transpose *)
bm = Transpose[bm];

(* 5. Find the vector, which multiplied by bm, gives k *)

c = LinearSolve[bm, k];

If[c[[0]] === LinearSolve,
  Message[BasisCoefficients::nmember, c // MatrixForm];
];

c
];

BasisCoefficientsMatrix[a____] :=
  InvalidArg["BasisCoefficientsMatrix", a];

```

12.4.2 BasisCoefficientsVector

BasisCoefficientsVector::usage=

"BasisCoefficientsVector[b,x] gives the coordinates of x with respect to basis b. x must be a vector.";

```
BasisCoefficientsVector[b_?ListQ, x_?VectorQ] := Module[
  {k},
  k = LinearSolve[Transpose[b], x];
  k
];

BasisCoefficientsVector[a_...] :=
  InvalidArg["BasisCoefficientsVector", a];
```

12.4.3 BasisToRootSystem

BasisToRootSystem::usage=

"BasisToRootSystem[d] identifies the type of root system formed by a basis d.";

```

BasisToRootSystem[d_?RootBasisQ] := Module[
  {cMatr, bMatr, i, sbasis, start, str},

  (*
  Print["BasisToRootSystem."];
  *)

  str = "";

  (* Due to the possibility of type BC,
  this is slightly more complicated than what it may seem to be.
  First create the Cartan Matrix for the basis.
  Force the creation regardless of the possible presence of BC. *)

  cMatr = CartanMatrixFromBasis[d, True];

  (* Form the blocks *)
  bMatr = BlockList[cMatr];

  start = 1;
  For[i = 1, i <= Length[bMatr], i++, (* FOR each block *)
    (*
    Extract the "sub basis" consisting of elements of the basis \
    corresponding to the current block *)

    sbasis = Take[d, {start, start + Length[bMatr[[i]]] - 1}];

    (* Check if this corresponds to BC *)

    If[! SameQ[MatrixRank[sbasis], Length[sbasis]],
      (* YES *)
      str = str <> "BC";
      str = str <> ToString[MatrixRank[sbasis]];
    ,
    (* NO *)
    str = str <> CartanToRootSystem[bMatr[[i]]];
    ];

    If[! SameQ[i, Length[bMatr]],
      str = str <> "+";
    ];

    start = start + Length[bMatr[[i]]]; (*
    shift the index over to the next sub system *)
    ];

  Return[str];
];

BasisToRootSystem[a_...] := InvalidArg["BasisToRootSystem", a];

```

12.4.4 BlockAssemble

BlockAssemble::usage=

"BlockAssemble[mlist] assembles square matrices listed in mlist into one large matrix with each list element along the diagonal.

BlockAssemble[a,b,...] assembles square matrices a, b,... into one large matrix with each element along the diagonal.";

```
BlockAssemble::nsquare = "Matrix in position '1' is not square.";

BlockAssemble[{argmlist__?MatrixQ}] := Module[
  {mat, dim, i, row, newrow, len, mlist},

  mlist = List[argmlist];
  mat = {};

  (* Get the total dimension *)
  dim = 0;
  For[i = 1, i <= Length[mlist], i++,
    dim = dim + Length[mlist[[i]]];

  (* Check that each element is square *)

  If[! SameQ[Length[mlist[[i]]], Length[mlist[[i, 1]]]],
    Message[BlockAssemble::nsquare, i];
    Return[Fail];
  ];

  len = 0;

  For[i = 1, i <= Length[mlist], i++, (* FOR each block *)

    For[row = 1, row <= Length[mlist[[i]]], row++, (*
      FOR each row in block i *)

      newrow = VectorPad[mlist[[i, row]], len, dim];
      mat = Join[mat, {newrow}];
    ];

    len += Length[mlist[[i]]];
  ];

  Return[mat];
];

BlockAssemble[mlist__?MatrixQ] := BlockAssemble[{mlist}];

BlockAssemble[a___] := InvalidArg["BlockAssemble", a];
```

12.4.5 BlockList

BlockList::usage=

”BlockList[matr] creates a list of each of the block matrices found in matrix matr.”;

```
BlockList[matr_?MatrixQ] := Module[
  {i, splitlist, matrixlist, startpos},

  splitlist = {};
  matrixlist = {};

  (* 1. Create a list of the lower right most entry of each block *)

  For[i = 1, i <= Length[matr] - 1, i++,
    If[ZeroesBelow[matr, i] && ZeroesRight[matr, i] &&
      ZeroesAbove[matr, i + 1] && ZeroesLeft[matr, i + 1],
      splitlist = Join[splitlist, {i}];
    ];
  ];

  splitlist = Join[splitlist, {i}];

  (* 2. List all the blocks *)
  startpos = 1;

  For[i = 1, i <= Length[splitlist], i++,
    matrixlist =
      Join[matrixlist, {Take[
        matr, {startpos, splitlist[[i]]}, {startpos,
          splitlist[[i]]}]}];
    startpos = splitlist[[i]] + 1;
  ];

  Return[matrixlist];
];

BlockList[a_...] := InvalidArg["BlockList", a];
```

12.4.6 ByBasisSort

ByBasisSort::usage=

”ByBasisSort[basis,list,p] sorts a list of elements in the span of a given basis according to their coordinate vectors.”;

```
ByBasisSort[basis_?ListQ, list_?ListQ] := Module[
  {kTable, i},

  (* Grab the coordinate vectors *)

  kTable =
    Table[BasisCoefficients[basis, list[[i]]], {i, 1, Length[list]};

  (* Sort the coordinate vectors *)

  kTable = Sort[kTable, BasisOrder];

  (* Return the original basis vectors *)

  Return[Table[kTable[[i]].basis, {i, 1, Length[list]}]];
];

ByBasisSort[a_...] := InvalidArg["ByBasisSort", a];

BasisOrder[a_, b_] := Module[
  {i},

  For[i = 1, i <= Length[a], i++,
    If[a[[i]] < b[[i]],
      Return[False];
    ];

    If[a[[i]] > b[[i]],
      Return[True];
    ];

  ];

  Return[False];
];

SyLexiOrder[a_, b_] := Module[
  {i},

  For[i = 1, i <= Length[a], i++,
    If[a[[i]] < b[[i]],
      Return[True];
    ];

    If[a[[i]] > b[[i]],
      Return[False];
    ];

  ];

  Return[False];
];
```

12.4.7 CartanMatrixFromBasis

CartanMatrixFromBasis::usage=

"CartanMatrixFromBasis[d,force] gives the Cartan Matrix formed by a basis of roots d. The optional argument force, if True, will force a matrix to be generated even if the given set of vectors does not form a basis."

```
CartanMatrixFromBasis::rank =
  "argument is not a basis. Number of the elements is '1'. Rank is \
  '2'.";

CartanMatrixFromBasis[d_?MatrixQ, force_: False] := Module[{i, j},
  If[! force && ! SameQ[MatrixRank[d], Length[d]],
    Message[CartanMatrixFromBasis::rank, Length[d], MatrixRank[d]];
    Return[{}];];

  Return[
    Table[Table[
      2*InnerProduct[d[[i]], d[[j]]]/InnerProduct[d[[j]], d[[j]]], {j,
        1, Length[d]}, {i, 1, Length[d]}];
    (*Table[Table[innerProduct[d[[i]],coRoot[d[[j]]]],{j,1,Length[
      d]}],{i,1,Length[d]}*)
  ];

CartanMatrixFromBasis[a___] := InvalidArg["CartanMatrixFromBasis", a];
```


12.4.8 CartanNorm

CartanNorm::usage=

”CartanNorm[m,d,a,b] computes $\langle a, b \rangle$ where a is a root, b is a simple root, and m is the Cartan Matrix for a root system with basis d.”;

```
CartanNorm::root = "'1' is not a simple root";

CartanNorm[d_?RootBasisQ, a_?VectorQ, b_?VectorQ] := Module[
  {k, i, pos, cMatr},

  k = BasisCoefficients[d, a];

  cMatr = CartanMatrixFromBasis[d];

  If[! MemberQ[d, b],
    Message[CartanNorm::root, b];
    Return["X"];
  ];

  pos = Position[d, b][[1, 1]];

  Sum[k[[i]]*cMatr[[i, pos]], {i, 1, Length[k]}]
];

CartanNorm[a_...] := InvalidArg["CartanNorm", a];
```

12.4.9 CartanToRootSystemSimple

CartanToRootSystemSimple::usage=

"CartanToRootSystemSimple[m] identifies the root system represented by Cartan Matrix m if it is known to be an irreducible root system.";

```
CartanToRootSystemSimple::fail =
  "Could not identify matrix as a root system.";

(* bugfix: was confusing B and F
credit: C. Buell *)

CartanToRootSystemSimple[matr_?MatrixQ] := Module[
  {sg1, dbl, tp1, n, i, pos, sg2, db2, tp2, rowSum, colSum, a, b},

  n = Length[matr];

  (* Count the number of single, double,
  and triple bonds represented *)

  sg1 = Table[Count[matr[[i]], -1], {i, 1, Length[matr]}];
  dbl = Table[Count[matr[[i]], -2], {i, 1, Length[matr]}];
  tp1 = Table[Count[matr[[i]], -3], {i, 1, Length[matr]}];
  sg2 = Table[Count[matr[[All, i]], -1], {i, 1, Length[matr]}];
  db2 = Table[Count[matr[[All, i]], -2], {i, 1, Length[matr]}];
  tp2 = Table[Count[matr[[All, i]], -3], {i, 1, Length[matr]}];

  (* Rule out G2... any row have a triple bond? *)

  If[Norm[tp1] > 0, Return[RootToString[{"G", n}]]];

  (* B, C, and F case: any row have a double bond? *)

  If[Norm[dbl] > 0,
    pos = Position[matr, -2];
    {{a, b}} = Position[matr, -2];
    rowSum = Sum[matr[[a, i]], {i, n}];
    colSum = Sum[matr[[i, b]], {i, n}];

    If[SameQ[pos, {{2, 1}}] && n == 2,
      Return[RootToString[{"B", n}]]];
    If[SameQ[{rowSum, colSum}, {-1, 0}],
      Return[RootToString[{"B", n}]]];

    If[SameQ[pos, {{1, 2}}] && n == 2,
      Return[RootToString[{"C", n}]]];
    If[SameQ[{rowSum, colSum}, {0, -1}],
      Return[RootToString[{"C", n}]]];

    If[SameQ[{rowSum, colSum}, {-1, -1}],
      Return[RootToString[{"F", 4}]]];

    Message[CartanToRootSystemSimple::fail];
    Return[Fail];
  ];

  (* D, E case: Any node have three single bonds? *)

  If[MemberQ[sg1, 3],
```

```

If[6 <= n <= 8,
  If[Det[matr] == 3 || Det[matr] == 2 || Det[matr] == 1,
    Return[RootToString[{"E", n}]]];];

Return[RootToString[{"D", n}]]];
];

Return[RootToString[{"D", n}]]];
];

(* A is only remaining case *)

Return[RootToString[{"A", n}]]];
];

(*
CartanToRootSystemSimple[matr_?MatrixQ]:=Module[
{sgl,dbl,tpl,n,i,pos},

n=Length[matr];

(* Count the number of single, double, and triple bonds represented *)
\
sgl=Table[Count[matr[[i]],-1],{i,1,Length[matr]}];
dbl=Table[Count[matr[[i]],-2],{i,1,Length[matr]}];
tpl=Table[Count[matr[[i]],-3],{i,1,Length[matr]}];

(* Rule out G2... any row have a triple bond? *)
\
If[Norm[tpl]>0,Return[RootToString[{"G",n}]]];];

(* B, C, and F case: any row have a double bond? *)
If[Norm[dbl]>0,
pos=Position[matr,-2];

If[SameQ[pos,{2,1}],Return[RootToString[{"B",n}]]];];
If[SameQ[pos,{n-1,n}],Return[RootToString[{"B",n}]]];];

If[SameQ[pos,{1,2}],Return[RootToString[{"C",n}]]];];
If[SameQ[pos,{n,n-1}],Return[RootToString[{"C",n}]]];];

If[SameQ[pos,{2,3}],Return[RootToString[{"F",4}]]];];

Message[CartanToRootSystemSimple::fail];
Return[RootToString[{"X",n}]]];
];

(* D, E case: Any node have three single bonds? *)
If[MemberQ[sgl,3],
pos=Position[sgl,3][[1,1]];

If[pos==4,
If[n>=7,Return[RootToString[{"E",n}]]];];

If[n==6,
If[Det[matr]==3,Return[RootToString[{"E",n}]]];];

Return[RootToString[{"D",n}]]];
];
];

Return[RootToString[{"D",n}]]];

```

```

];

(* A is only remaining case *)
Return[RootToString[{"A",n}]];
];
*)

CartanToRootSystemSimple[a_...] :=
  InvalidArg["CartanToRootSystemSimple", a];

```

12.4.10 Diagonalize

Diagonalize::usage=

"Diagonalize[m] returns a matrix p and diagonal matrix d such that $d = p^{-1} m p$."

```
Diagonalize[m_?MatrixQ] := Module[
  {evs, p, pinv},

  evs = Eigenvectors[m];

  p = Transpose[evs];
  pinv = Inverse[p];

  {p, pinv.m.p}
];

Diagonalize[a_...] := InvalidArg["Diagonalize", a];
```

12.4.11 DynkinData

DynkinData::usage=

"DynkinData[r,disks] returns relevant data for a Dynkin diagram with root system r. The output is a list of elements of the form neighbor 1, edge type , neighbor 2, edge type , ... describing the neighbors a simple root shares, and the connecting bond type. The optional argument disks will throw out any data which points to a root not in the set of disks, and return for any roots which are not in the set of disks. i.e. Return data for only a sub-diagram defined by the roots of disks.";

```

DynkinData[d_?RootBasisQ, disks_: {}] :=
  DynkinData[BasisToRootSystem[d], disks];

DynkinData[r_?RootInputQ, disks_: {}] := Module[
  {rs, rootsystem, i, j, out, type, dim, of, dellist, w},

  rs = RootInput[r];

  out = {};
  of = 0;

  For[i = 1, i <= Length[rs], i++,
    rootsystem = rs[[i]];
    type = rootsystem[[1]];
    dim = rootsystem[[2]];

    Switch[type,
      "A",
      out = Join[out, {{2 + of, 10}}];

      For[j = 2, j <= dim - 1, j++,
        out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
      ];

      out = Join[out, {{dim - 1 + of, 10}}];,
      "B",
      out = Join[out, {{2 + of, 10}}];

      For[j = 2, j <= dim - 2, j++,
        out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
      ];

      out =
        Join[out, {{dim - 2 + of, 10}, {dim + of, 22}, {dim - 1 + of,
          22}}];,
      "C",
      out = Join[out, {{2 + of, 10}}];

      For[j = 2, j <= dim - 2, j++,
        out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
      ];

      out =
        Join[out, {{dim - 2 + of, 10}, {dim + of, 21}, {dim - 1 + of,
          21}}];,
      "D",

```

```

out = Join[out, {{2 + of, 10}}];

For[j = 2, j <= dim - 3, j++,
  out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
];

out =
  Join[out, {{dim - 3 + of, 10}, {dim - 1 + of, 10}, {dim + of,
    10}}, {{dim - 2 + of, 10}}, {{dim - 2 + of, 10}}];,
"E",
Switch[dim,
  6,
  out =
    Join[out, {{3 + of, 10}}, {4 + of, 10}}, {{1 + of,
      10}, {4 + of, 10}}, {{2 + of, 10}, {3 + of, 10}, {5 + of,
        10}}, {{4 + of, 10}, {6 + of, 10}}, {{5 + of, 10}}];,
  7,
  out =
    Join[out, {{3 + of, 10}}, {4 + of, 10}}, {{1 + of,
      10}, {4 + of, 10}}, {{2 + of, 10}, {3 + of, 10}, {5 + of,
        10}}, {{4 + of, 10}, {6 + of, 10}}, {{5 + of,
          10}, {7 + of, 10}}, {{6 + of, 10}}];,
  8,
  out =
    Join[out, {{3 + of, 10}}, {4 + of, 10}}, {{1 + of,
      10}, {4 + of, 10}}, {{2 + of, 10}, {3 + of, 10}, {5 + of,
        10}}, {{4 + of, 10}, {6 + of, 10}}, {{5 + of,
          10}, {7 + of, 10}}, {{6 + of, 10}, {8 + of,
            10}}, {{7 + of, 10}}];,
  ],,
"F",
out =
  Join[out, {{2 + of, 10}}, {{1 + of, 10}, {3 + of,
    22}}, {{2 + of, 22}, {4 + of, 10}}, {{3 + of, 10}}];,
"G",
out = Join[out, {{2 + of, 31}}, {{1 + of, 31}}];
];

of = of + dim;
];

(* Disks? *)
If[Length[disk] > 0,
  For[i = 1, i <= Length[out], i++,
    If[MemberQ[disk, i],
      dellist = {};
      For[j = 1, j <= Length[out[[i]]], j++,
        If[! MemberQ[disk, out[[i, j, 1]]],
          dellist = Join[dellist, {j}];
        ];
      ];
      dellist = Table[{j}, {j, 1, Length[out[[i]]]}];
    ];
    w = Delete[out[[i]], dellist];
    out[[i]] = w;
  ];
];

out
];

```

```
DynkinData[a_]:=InvalidArg["DynkinData", a];
```


12.4.12 DynkinDataNaturalOrdering

DynkinDataNaturalOrdering::usage=

”DynkinDataNaturalOrdering[r,disks] returns relevant data for a Dynkin diagram with root system r. The output is a list of elements of the form neighbor 1, edge type , neighbor 2, edge type , ... describing the neighbors a simple root shares, and the connecting bond type. The optional argument disks will throw out any data which points to a root not in the set of disks, and return for any roots which are not in the set of disks. i.e. Return data for only a sub-diagram defined by the roots of disks. This variant of dynkinData re-orders the labeling of the simple roots of type E. Roots 2 and 3 are swapped, then 3 and 4.”;

```

DynkinDataNaturalOrdering[d_?RootBasisQ, disks_: {}] :=
  DynkinDataNaturalOrdering[BasisToRootSystem[d], disks];

DynkinDataNaturalOrdering[r_?RootInputQ, disks_: {}] := Module[
  {rs, rootsystem, i, j, out, type, dim, of, dellist, w},

  rs = RootInput[r];

  out = {};
  of = 0;

  For[i = 1, i <= Length[rs], i++,
    rootsystem = rs[[i]];
    type = rootsystem[[1]];
    dim = rootsystem[[2]];

    Switch[type,
      "A",
      out = Join[out, {{2 + of, 10}}];

      For[j = 2, j <= dim - 1, j++,
        out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
      ];

      out = Join[out, {{dim - 1 + of, 10}}];
      "B",
      out = Join[out, {{2 + of, 10}}];

      For[j = 2, j <= dim - 2, j++,
        out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
      ];

      out =
        Join[out, {{dim - 2 + of, 10}, {dim + of, 22}, {dim - 1 + of,
          22}}];,
      "C",
      out = Join[out, {{2 + of, 10}}];

      For[j = 2, j <= dim - 2, j++,
        out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
      ];

      out =
        Join[out, {{dim - 2 + of, 10}, {dim + of, 21}, {dim - 1 + of,
```

```

        21}}}],
"D",
out = Join[out, {{2 + of, 10}}];

For[j = 2, j <= dim - 3, j++,
  out = Join[out, {{j - 1 + of, 10}, {j + 1 + of, 10}}];
];

out =
  Join[out, {{dim - 3 + of, 10}, {dim - 1 + of, 10}, {dim + of,
    10}}, {{dim - 2 + of, 10}, {dim - 2 + of, 10}}];,
"E",
Switch[dim,
  6,
    out =
      Join[out, {{2 + of, 10}}, {{1 + of, 10}, {3 + of,
        10}}, {{2 + of, 10}, {4 + of, 10}, {5 + of,
        10}}, {{3 + of, 10}}, {{3 + of, 10}, {6 + of,
        10}}, {{5 + of, 10}}];,
  7,
    out =
      Join[out, {{2 + of, 10}}, {{1 + of, 10}, {3 + of,
        10}}, {{2 + of, 10}, {4 + of, 10}, {5 + of,
        10}}, {{3 + of, 10}}, {{3 + of, 10}, {6 + of,
        10}}, {{5 + of, 10}, {7 + of, 10}}, {{6 + of, 10}}];,
  8,
    out =
      Join[out, {{2 + of, 10}}, {{1 + of, 10}, {3 + of,
        10}}, {{2 + of, 10}, {4 + of, 10}, {5 + of,
        10}}, {{3 + of, 10}}, {{3 + of, 10}, {6 + of,
        10}}, {{5 + of, 10}, {7 + of, 10}}, {{6 + of,
        10}, {8 + of, 10}}, {{7 + of, 10}}];
];,
"F",
out =
  Join[out, {{2 + of, 10}}, {{1 + of, 10}, {3 + of,
    22}}, {{2 + of, 22}, {4 + of, 10}}, {{3 + of, 10}}];,
"G",
out = Join[out, {{2 + of, 31}}, {{1 + of, 31}}];
];

of = of + dim;
];

(* Disks? *)
If[Length[disks] > 0,
  For[i = 1, i <= Length[out], i++,
    If[MemberQ[disks, i],
      dellist = {};
      For[j = 1, j <= Length[out[[i]]], j++,
        If[! MemberQ[disks, out[[i, j, 1]]],
          dellist = Join[dellist, {j}];
        ];
      ];
    ,
    dellist = Table[{j}, {j, 1, Length[out[[i]]]};
  ];

  w = Delete[out[[i]], dellist];
  out[[i]] = w;
];
];

```

```
out
];

DynkinDataNaturalOrdering[a_...] :=
  InvalidArg["DynkinDataNaturalOrdering", a];
```

12.4.13 DynkinEdgeCodes

DynkinEdgeCodes::usage=

” th th th DynkinEdgeCodes[type,dim] returns a list of codes where the i entry denotes the type of edge between the i and (i+1) simple roots. The edge codes are: 10 = single line, 21 = double left line, 22 = double right line, 31 = triple left line. type denotes the root system type, and dim denotes the dimension.”;

```

DynkinEdgeCodes[r_?RootInputQ] := DynkinEdgeCodes[RootBase[r]];

DynkinEdgeCodes[d_?RootBasisQ] := Module[
  {i, ret, thislen, nextlen},

  ret = {};

  For[i = 1, i <= Length[d] - 1, i++,
    thislen = Norm[d[[i]]];
    nextlen = Norm[d[[i + 1]]];

    Switch[thislen^2/nextlen^2,
      1, (* single bond *)
      ret = Join[ret, {11}];
    ,
      2, (* double bond *)

      If[thislen < nextlen, (* the left element is the shorter root *)

        ret = Join[ret, {21}];
      ,
        ret = Join[ret, {22}]; (*
        the right element is the shorter root *)
      ];
    ,
      1/2, (* double bond *)

      If[thislen < nextlen, (* the left element is the shorter root *)

        ret = Join[ret, {21}];
      ,
        ret = Join[ret, {22}]; (*
        the right element is the shorter root *)
      ];
    ,
      3, (* triple bond *)

      If[thislen < nextlen, (* the left element is the shorter root *)

        ret = Join[ret, {31}];
      ,
        ret = Join[ret, {32}]; (*
        the right element is the shorter root *)
      ];
    ,
      1/3, (* triple bond *)

      If[thislen < nextlen, (* the left element is the shorter root *)

        ret = Join[ret, {31}];

```

```

    ,
    ret = Join[ret, {32}]; (*
    the right element is the shorter root *)
  ];
];

Return[ret];
];

DynkinEdgeCodes[type_?StringQ, dim_?IntegerQ] := Module[
{i, t},

Switch[type,
  "A",
  t = Table[10, {i, 1, dim - 1}];,
  "B",
  t = Table[10, {i, 1, dim - 2}];
  t = Join[t, {22}];,
  "C",
  t = Table[10, {i, 1, dim - 2}];
  t = Join[t, {21}];,
  "D",
  t = Table[10, {i, 1, dim - 1}];,
  "E",
  t = Table[10, {i, 1, dim - 1}];,
  "F",
  t = {10, 22, 10};,
  "G",
  t = {31};,
  -,
  Message[lie::type, type];
];

t
];

DynkinEdgeCodes[a___] := InvalidArg["DynkinEdgeCodes", a];

```

12.4.14 DynkinEdgeCons

DynkinEdgeCons::usage=

"DynkinEdgeCons[type,dim] returns a list of pairs of simple roots which are joined together.
type denotes the root system type, and dim denotes the dimension.";

```
DynkinEdgeCons[type_?StringQ, dim_?IntegerQ] := Module[{i, t},
  Switch[type,
    "A",
    t = Table[{i, i + 1}, {i, 1, dim - 1}];,
    "B",
    t = Table[{i, i + 1}, {i, 1, dim - 1}];,
    "C",
    t = Table[{i, i + 1}, {i, 1, dim - 1}];,
    "D",
    t = Table[{i, i + 1}, {i, 1, dim - 3}];
    t = Join[t, {{dim - 2, dim - 1}, {dim - 2, dim}}];,
    "E",
    Switch[dim,
      6,
      t = {{1, 3}, {3, 4}, {4, 2}, {4, 5}, {5, 6}};
      ,
      7,
      t = {{1, 3}, {3, 4}, {4, 2}, {4, 5}, {5, 6}, {6, 7}};
      ,
      8,
      t = {{1, 3}, {3, 4}, {4, 2}, {4, 5}, {5, 6}, {6, 7}, {7, 8}};
    ];
    ,
    "F",
    t = Table[{i, i + 1}, {i, 1, dim - 1}];,
    "G",
    t = Table[{i, i + 1}, {i, 1, dim - 1}];,
    -,
    Message[lie::type, type];
  ];

  Return[t];
];

DynkinEdgeCons[a___] := InvalidArg["DynkinEdgeCons", a];
```

12.4.15 DynkinEdgeConsNestedForm

DynkinEdgeConsNestedForm::usage=

"DynkinEdgeCons[type,dim] returns a list of pairs of simple roots which are joined together. type denotes the root system type, and dim denotes the dimension. If a root shares multiple neighbors, then the entries are merged. e.g. 1,2,3 indicates root 1 is connected to both roots 2 and 3."

```
DynkinEdgeConsNestedForm[type_?StringQ, dim_?IntegerQ] :=
Module[{i, t},
Switch[type,
"A",
t = Table[{i, i + 1}, {i, 1, dim - 1}];,
"B",
t = Table[{i, i + 1}, {i, 1, dim - 1}];,
"C",
t = Table[{i, i + 1}, {i, 1, dim - 1}];,
"D",
t = Table[{i, i + 1}, {i, 1, dim - 3}];
t = Join[t, {{dim - 2, {dim - 1, dim}}}],
"E",
Switch[dim,
6,
t = {{1, 3}, {3, 4}, {4, {2, 5}}, {5, 6}};
,
7,
t = {{1, 3}, {3, 4}, {4, {2, 5}}, {5, 6}, {6, 7}};
,
8,
t = {{1, 3}, {3, 4}, {4, {2, 5}}, {5, 6}, {6, 7}, {7, 8}};
];
,
"F",
t = Table[{i, i + 1}, {i, 1, dim - 1}];,
"G",
t = Table[{i, i + 1}, {i, 1, dim - 1}];,
-,
Message[lie::type, type];
];

Return[t];
];

DynkinEdgeConsNestedForm[a___] :=
InvalidArg["DynkinEdgeConsNestedForm", a];
```

12.4.16 DynkinHeight

DynkinHeight::usage=

”DynkinHeight[r] gives the height of a Dynkin diagram of type r, where the height is the number of simple roots along the tallest vertical line.”;

```
DynkinHeight[r_?RootBasisQ] := DynkinHeight[BasisToRootSystem[r]];

DynkinHeight[r_?RootInputQ] := Module[
  {w, rin},

  rin = IrreducibleRootInput[r];

  Switch[rin[[1]],
    "A", w = 1;,
    "B", w = 1;,
    "C", w = 1;,
    "D", w = 2;,
    "E", w = 2;,
    "F", w = 1;,
    "G", w = 1;
  ];

  Return[w];
];

DynkinHeight[a_...] := InvalidArg["DynkinHeight", a];
```


12.4.17 DynkinPoints

DynkinPoints::usage=

"DynkinPoints[r] provides a list of relative x,y positions of the dots of a Dynkin diagram for a root system r.

DynkinPoints[type,dim,xOff,yOff] provides a list of relative x,y positions of the dots of a Dynkin diagram for a root system of type (type,dim). The points are offset along the x and y axes by xOff and yOff respectively."

```
DynkinPoints[type_?StringQ, dim_?IntegerQ, xOffset_?NumberQ,
yOffset_?NumberQ] := Module[
{i, points},

Switch[type,
"A",
points = Table[{i + xOffset, 0 + yOffset}, {i, 0, dim - 1}];,
"B",
points = Table[{i + xOffset, 0 + yOffset}, {i, 0, dim - 1}];,
"C",
points = Table[{i + xOffset, 0 + yOffset}, {i, 0, dim - 1}];,
"D",
points = Table[{i + xOffset, 0 + yOffset}, {i, 0, dim - 3}];
points =
Join[points, {{dim - 2 + xOffset,
0.5 + yOffset}, {dim - 2 + xOffset, -0.5 + yOffset}}];,
"E",
Switch[dim,
6,
points = {{0 + xOffset, 0 + yOffset}, {2 + xOffset,
1 + yOffset}, {1 + xOffset, 0 + yOffset}, {2 + xOffset,
0 + yOffset}, {3 + xOffset, 0 + yOffset}, {4 + xOffset,
0 + yOffset}}];,
7,
points = {{0 + xOffset, 0 + yOffset}, {2 + xOffset,
1 + yOffset}, {1 + xOffset, 0 + yOffset}, {2 + xOffset,
0 + yOffset}, {3 + xOffset, 0 + yOffset}, {4 + xOffset,
0 + yOffset}, {5 + xOffset, 0 + yOffset}}];,
8,
points = {{0 + xOffset, 0 + yOffset}, {2 + xOffset,
1 + yOffset}, {1 + xOffset, 0 + yOffset}, {2 + xOffset,
0 + yOffset}, {3 + xOffset, 0 + yOffset}, {4 + xOffset,
0 + yOffset}, {5 + xOffset, 0 + yOffset}, {6 + xOffset,
0 + yOffset}}];,
],
"F",
points = {{0 + xOffset, 0 + yOffset}, {1 + xOffset,
0 + yOffset}, {2 + xOffset, 0 + yOffset}, {3 + xOffset,
0 + yOffset}}];,
"G",
points = {{0 + xOffset, 0 + yOffset}, {1 + xOffset,
0 + yOffset}}];,
-,
Message[lie::type, type];
];
```

```

Return[points];
];

DynkinPoints[d_?RootBasisQ] := DynkinPoints[BasisToRootSystem[d]];

DynkinPoints[r_?RootInputQ] := Module[
  {i, points, toff, tht, biggest, mywidth, rin},

  rin = RootInput[r];
  points = {};
  toff = 0;
  tht = 0;
  biggest = 0;

  For[i = 1, i <= Length[rin], i++,
    mywidth = DynkinWidth[rin[[i]]];

    If[mywidth > biggest,
      biggest = mywidth;
    ];
  ];

  For[i = 1, i <= Length[rin], i++,
    mywidth = DynkinWidth[rin[[i]]];

    If[i > 1,
      tht = tht - DynkinHeight[rin[[i]]];
    ];

    points =
      Join[points,
        DynkinPoints[rin[[i, 1]], rin[[i, 2]], (biggest - mywidth)/2,
          tht]];
    toff = toff + rin[[i, 2]];
  ];

  Return[points];
];

DynkinPoints[a___] := InvalidArg["DynkinPoints", a];

```

12.4.18 DynkinOrientation

DynkinOrientation::usage=

"DynkinOrientation[r,basis] returns True if the Dynkin diagram corresponding to the root system with the supplied basis is oriented in the direction standard to Pisces (See Humphreys, Introduction to Lie Algebra and Representation Theory).

DynkinOrientation[r] returns 1 if r is oriented 'correctly' (as per Humphreys), -1 if oriented backward, or 0 if basis elements are not ordered. If r is the name of a root system (not a basis), then 1 is always returned."

```
DynkinOrientation[r_?RootInputQ] := 1;

DynkinOrientation[d_?RootBasisQ] := Module[
  {rname, rin, type, dim},

  (* Get the name *)
  rname = BasisToRootSystem[d];
  rin = IrreducibleRootInput[rname];
  type = rin[[1]];
  dim = rin[[2]];

  Switch[type,
    "A",
    Return[1];
  ,
    "B",
    (* Give n is at least 3,
    If roots 1 and 2 have the same length then oriented correctly.
    Else incorrectly *)
    If[dim >= 3,
      If[SameQ[Norm[d[[1]]], Norm[d[[2]]]],
        Return[1];
      ,
        Return[-1];
      ];
    ,
    (* If n is 2,
    then oriented correctly if length of first root is greater *)
    If[Norm[d[[1]]] > Norm[d[[2]]],
      Return[1];
    ,
      Return[-1];
    ];
  ];

  "C",
  (* If roots 1 and 2 have the same length then oriented correctly.
  Else incorrectly *)
  If[SameQ[Norm[d[[1]]], Norm[d[[2]]]],
    Return[1];
  ,
    Return[-1];
  ];

  "F",
```

```

(* Length of roots on left should be greater *)

If[Norm[d[[2]]] > Norm[d[[3]]],
  Return[1];
,
  Return[-1];
];

"G",
(* Length of root on right should be greater *)

If[Norm[d[[1]]] < Norm[d[[2]]],
  Return[1];
,
  Return[-1];
];

Return[0];
];

DynkinOrientation[r_, basis_?ListQ] := Module[
  {},

  Return[SameQ[CartanMatrix[basis], CartanMatrix[r]]];
];

DynkinOrientation[a___] := InvalidArg["DynkinOrientation", a];

```

12.4.19 DynkinToCartan

DynkinToCartan::usage=

"DynkinToCartan[D] recovers the Cartan matrix from the Dynkin diagram information D returned by dynkinData and dynkinDataNaturalOrdering.";

```

DynkinToCartan[d_?ListQ] := Module[
  {i, j, m, dim, rowdata, rootdata},

  dim = Length[d];

  m = ConstantArray[0, {dim, dim}];

  (* For every row *)
  For[i = 1, i <= dim, i++,
    rowdata = d[[i]];

    If[Length[rowdata] > 0,
      m[[i, i]] = 2;
    ];

    (* For every root connected to root i *)

    For[j = 1, j <= Length[rowdata], j++,
      rootdata = rowdata[[j]];

      (* What is the type of connection? *)
      Switch[rootdata[[2]],
        10, (* Single Edge *)
          m[[i, rootdata[[1]]]] = -1;,
        21,
          If[rootdata[[1]] > i,
            m[[i, rootdata[[1]]]] = -1;
          ,
            m[[i, rootdata[[1]]]] = -2;
          ];,
        22,
          If[rootdata[[1]] > i,
            m[[i, rootdata[[1]]]] = -2;
          ,
            m[[i, rootdata[[1]]]] = -1;
          ];,
        31,
          If[rootdata[[1]] > i,
            m[[i, rootdata[[1]]]] = -1;
          ,
            m[[i, rootdata[[1]]]] = -3;
          ];
      ];
    ];
  ];

  Return[m];
];

DynkinToCartan[a___] := InvalidArg["DynkinToCartan", a];

```

12.4.20 DynkinWidth

DynkinWidth::usage=

"DynkinWidth[r] gives the width of a Dynkin diagram of type r, where the width is the number of simple roots along the longest horizontal line.";

```

DynkinWidth[r_?RootBasisQ] := DynkinWidth[BasisToRootSystem[r]];

DynkinWidth[r_?RootInputQ] := Module[
  {w, rin},

  rin = IrreducibleRootInput[r];

  Switch[rin[[1]],
    "A", w = rin[[2]];;,
    "B", w = rin[[2]];;,
    "C", w = rin[[2]];;,
    "D", w = rin[[2]] - 1;;,
    "E", w = rin[[2]] - 1;;,
    "F", w = rin[[2]];;,
    "G", w = rin[[2]];;
  ];

  Return[w];
];

DynkinWidth[a_...] := InvalidArg["DynkinWidth", a];

```

12.4.21 FindPivots

FindPivots::usage=

"FindPivots[m] indexes the columns containing pivots in RREF[m].";

```
FindPivots[m_?MatrixQ] := Module[
  {mt, pvlist, i, erows},

  (* Look for columns that have pivots *)
  pvlist = {};

  (*mt=RowReduce[m];*)

  (* Take the transpose so mt[[
  i]] corresponds to column i and not row i of m *)

  mt = Transpose[m];

  (* Make a table of unit rows *)

  erows = Table[
    UnitVector[Length[mt[[1]]], i], {i, 1, Length[mt[[1]]]};

  For[i = 1, i <= Length[mt], i++,
    If[MemberQ[erows, mt[[i]]],
      pvlist = Join[pvlist, {i}];
    ];
  ];

  Return[pvlist];
];

FindPivots[a___] := InvalidArg["FindPivots", a];
```

12.4.22 GroebnerBackSolver

GroebnerBackSolver::usage=

"GroebnerBackSolver[eqns,vars] finds all solutions to the multivariate polynomial system given by eqns with set of variables vars."

```
GroebnerBackSolver[eqns_?ListQ, vars_?ListQ] :=
  GroebnerBackSolver[eqns, vars, Length[vars], {}, {}];

GroebnerBackSolver[eqns_?ListQ, vars_?ListQ, m_?IntegerQ, rules_,
  solnset_] := Module[{gb, subeqns, soln, i, nrules, nsolnset},
  If[m == 0,
    Return[Union[solnset, {rules}]];
  ];

  gb = GroebnerBasis[eqns, vars, Drop[vars, -(Length[vars] + 1 - m)]];

  If[gb == {},
    Return[GroebnerBackSolver[eqns, vars, m - 1, rules, solnset]];
  ];

  subeqns =
    Simplify[Table[gb[[i]] == 0, {i, 1, Length[gb]}] /. rules];

  soln = Solve[subeqns, vars[[m]]] // Flatten;

  (*Print["At m=",m," ",Length[soln]," solutions found."];*)

  nsolnset = solnset;

  For[i = 1, i <= Length[soln], i++,
    (*Print["Level ",m," Solution ",i];*)

    nrules = Join[rules, {soln[[i]]}];

    nsolnset = GroebnerBackSolver[eqns, vars, m - 1, nrules, nsolnset];
  ];

  Return[nsolnset];
];

GroebnerBackSolver[a___] := InvalidArg["GroebnerBackSolver", a];

(*
groebnerBackSolver2[solnset_,gb_,vars_,x_]:=groebnerBackSolver[
solnset,gb,vars,x,1,{}];
*)

(*
groebnerBackSolver2[solnset_,gb_,vars_,x_,cureqn_,rules_]:=Module[{
eqn,soln,j,nrules,nsolnset},
nsolnset={};
eqn=(gb[[cureqn]]==0)/.rules;

soln=Flatten[Solve[eqn,vars[[-cureqn]]]];

For[j=1,j<=Length[soln],j++,
nrules=Join[rules,{soln[[j]]}];

If[cureqn<Length[vars],
```



```

nsolnset=Union[nsolnset,groebnerBackSolver[nsolnset,gb,vars,x,cureqn+\
1,nrules]];
,
nsolnset=Union[nsolnset,{nrules}];
];
];

nsolnset
];
*)

```

12.4.23 GroebnerOneSolution

GroebnerOneSolution::usage=

"GroebnerOneSolution[eqns,vars] finds one solution to the multivariate polynomial system given by eqns with set of variables vars."

```
GroebnerOneSolution[eqns_?ListQ, vars_?ListQ] :=
  GroebnerOneSolution[eqns, vars, Length[vars], {}, {}];

(* Return just 1 solution *)

GroebnerOneSolution[eqns_?ListQ, vars_?ListQ, m_?IntegerQ, rules_,
  solnset_] := Module[{gb, subeqns, soln, i, nrules, nsolnset},
  If[m == 0,
    Return[rules];
  ];

  gb = GroebnerBasis[eqns, vars, Drop[vars, -(Length[vars] + 1 - m)]];

  If[gb == {},
    Return[GroebnerOneSolution[eqns, vars, m - 1, rules, solnset]];
  ];

  subeqns =
    Simplify[Table[gb[[i]] == 0, {i, 1, Length[gb]}] /. rules];

  soln = Solve[subeqns, vars[[m]]] // Flatten;

  If[Length[soln] < 1, Return[{}]];

  (*Print["At m=",m," ",Length[soln]," solutions found."];*)

  nsolnset = solnset;

  nrules = Join[rules, {soln[[1]]}];

  nsolnset = GroebnerOneSolution[eqns, vars, m - 1, nrules, nsolnset];

  Return[nsolnset];
];

GroebnerOneSolution[a___] := InvalidArg["GroebnerOneSolution", a];
```

12.4.24 IrreducibleRootInput

IrreducibleRootInput::usage=

”IrreducibleRootInput[r] converts an irreducible root system r into a variant of list form TYPE,DIM. An error message and is returned if r is not irreducible.”;

```
IrreducibleRootInput::arg = "'1' is not an irreducible root system.";

IrreducibleRootInput[r_] := Module[
  {rin},

  rin = RootInput[r];

  If[Length[rin] > 1,
    Message[IrreducibleRootInput::arg, r];
    Return[Fail];
  ];

  Return[Flatten[rin]];
];

IrreducibleRootInput[a___] := InvalidArg["IrreducibleRootInput", a];
```

12.4.25 LieMultTable

LieMultTable::usage=

"LieMultTable[L] gives the Lie Multiplication Table for a set of vectors L forming the basis for a Lie algebra. The (i,j) entry of the table is the basis coordinate of the vector $v = [L_i, L_j]$.

```
LieMultTable[{argL_?MatrixQ}] := Module[
  {L, i, j, v, mt, time, cntr},

  L = List[argL];
  time = TimeUsed[];
  cntr = time;

  mt = ConstantArray[0, {Length[L], Length[L]}];

  For[i = 1, i <= Length[L], i++,
    For[j = 1, j <= Length[L], j++,
      v = LieBracket[L[[i]], L[[j]]];

      mt[[i, j]] = Position[L, v];
      (*
      If[TimeUsed[]-cntr>10,
      Print["(rootSpaceDecomp) CPU Time:[",TimeUsed[]-time,
      "]" Table entries:[",i*(Length[L]-1)+j," out of ",(Length[
      L])^2,""]];
      cntr=TimeUsed[];
      ];
      *)
    ];

  (*
  Print["(lieMultTable) Used ",TimeUsed[]-time,
  "s CPU Time to compute the multiplication table."];
  *)

  mt
];

LieMultTable[a___] := InvalidArg["LieMultTable", a];
```

12.4.26 LittleDynk

LittleDynk::usage=

"LittleDynk[r,xOff,yOff,nOff,labels] draws the Dynkin diagram for an irreducible root system r. xOff and yOff are, respectively, the x and y coordinate offsets of the diagram. nOff provides the offset for automatic root numbering (starting value). The optional argument labels allows custom labels for the simple roots.";

```

LittleDynk[d_?RootBasisQ, xOffset_?NumberQ, yOffset_?NumberQ,
  nOffSet_?IntegerQ, labels_: {}] :=
  LittleDynk[BasisToRootSystem[d], xOffset, yOffset, nOffSet, labels];

LittleDynk[r_?RootInputQ, xOffset_?NumberQ, yOffset_?NumberQ,
  nOffSet_?IntegerQ, labels_: {}] := Module[
  {rin, type, dim, vertexTypes, dotRadius, edgeThickness,
   points, edgeCodes, edgeCons, grPoints, grEdges, grLabels, grArgs,
   i, x1, y1, x2, y2},

  rin = IrreducibleRootInput[r];

  type = rin[[1]];
  dim = rin[[2]];

  (* Defaults *)
  vertexTypes = Table[Circle, {i, 1, dim}];
  dotRadius = .065;
  edgeThickness = .004;

  (* Apply options *)
  vertexTypes=vertexTypes//.options;
  dotRadius=dotRadius//.options;
  edgeThickness=edgeThickness//.options;
  *)
  points = DynkinPoints[type, dim, xOffset, yOffset];
  grPoints =
    Table[vertexTypes[[i]][points[[i]], dotRadius], {i, 1,
      Length[points]};
  edgeCodes = DynkinEdgeCodes[type, dim];
  edgeCons = DynkinEdgeCons[type, dim];

  If[SameQ[labels, {}],
    grLabels =
      Table[Text[
        i + nOffSet, {points[[i, 1]] + .1, points[[i, 2]] - .2}], {i,
        1, Length[points]};
    ,
    grLabels =
      Table[Text[
        labels[[i + nOffSet]], {points[[i, 1]] + .1,
        points[[i, 2]] - .2}], {i, 1, Length[points]};
  ];

  grEdges = {};

  For[i = 1, i <= Length[edgeCons], i++,
    x1 = points[[edgeCons[[i, 1]], 1]];
    y1 = points[[edgeCons[[i, 1]], 2]];
    x2 = points[[edgeCons[[i, 2]], 1]];

```

```

y2 = points[[edgeCons[[i, 2]], 2]];

Switch[edgeCodes[[i]],
  10,
  Which[
    y1 == y2,
    (* Horizontal Line *)

    grEdges = Join[grEdges, {Line[{x1 + .1, y1}, {x2 - .1, y2}]}];
    ,
    x1 == x2,
    (* Vertical Line *)

    grEdges = Join[grEdges, {Line[{x1, y1 + .1}, {x2, y2 - .1}]}];
    ,
    y1 < y2,
    (* Diagonal Up *)

    grEdges =
      Join[grEdges, {Line[{x1 + .1, y1 + .05}, {x2 - .1,
        y2 - .05}]}];
    ,
    y1 > y2,
    (* Diagonal Down *)

    grEdges =
      Join[grEdges, {Line[{x1 + .1, y1 - .05}, {x2 - .1,
        y2 + .05}]}];
    ];
  ,
  21,
  grEdges =
    Join[grEdges, {Line[{x1 + .1, y2 - .025}, {x2 - .1,
      y2 - .025}]}];
  grEdges =
    Join[grEdges, {Line[{x1 + .1, y2 + .025}, {x2 - .1,
      y2 + .025}]}];
  grEdges =
    Join[grEdges, {Polygon[{(x1 + x2)/2 + .15,
      y1 - .13}, {(x1 + x2)/2 + .15,
        y1 + .13}, {(x1 + x2)/2 - .15, y1}]}];
  ,
  22,
  grEdges =
    Join[grEdges, {Line[{x1 + .1, y2 - .025}, {x2 - .1,
      y2 - .025}]}];
  grEdges =
    Join[grEdges, {Line[{x1 + .1, y2 + .025}, {x2 - .1,
      y2 + .025}]}];
  grEdges =
    Join[grEdges, {Polygon[{(x1 + x2)/2 - .15,
      y1 - .13}, {(x1 + x2)/2 - .15,
        y1 + .13}, {(x1 + x2)/2 + .15, y1}]}];
  ,
  31,
  grEdges = Join[grEdges, {Line[{x1 + .1, y2}, {x2 - .1, y2}]}];
  grEdges =
    Join[grEdges, {Line[{x1 + .1, y2 - .025}, {x2 - .1,
      y2 - .025}]}];
  grEdges =
    Join[grEdges, {Line[{x1 + .1, y2 + .025}, {x2 - .1,
      y2 + .025}]}];

```

```

grEdges =
  Join[grEdges, {Polygon[{(x1 + x2)/2 + .15,
    y1 - .13}, {(x1 + x2)/2 + .15,
    y1 + .13}, {(x1 + x2)/2 - .15, y1}]}]];
];

grArgs =
  Join[{Thickness[edgeThickness]], grPoints, grEdges, grLabels];

grArgs
];

LittleDynk[a_...] := InvalidArg["LittleDynk", a];

```

12.4.27 MatrixMinimalPolynomial

MatrixMinimalPolynomial::usage=

"MatrixMinimalPolynomial[a,x] gives the minimal polynomial for a square matrix a with variable x.

Rowland,Todd and Weisstein,Eric W."Matrix Minimal Polynomial."

From MathWorld--A Wolfram Web Resource.

<http://mathworld.wolfram.com/MatrixMinimalPolynomial.html>";

```
(* source:
Rowland,Todd and Weisstein,Eric W."Matrix Minimal Polynomial." From \
MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/MatrixMinimalPolynomial.html
*)
MatrixMinimalPolynomial[a_List?MatrixQ, x_] :=
Module[{i, n = 1, qu = {}},
  mnm = {Flatten[IdentityMatrix[Length[a]]]};
  While[Length[qu] == 0, AppendTo[mnm, Flatten[MatrixPower[a, n]]];
    qu = NullSpace[Transpose[mnm]];
    n++;
  First[qu].Table[x^i, {i, 0, n - 1}]]

MatrixMinimalPolynomial[a_...] :=
  InvalidArg["MatrixMinimalPolynomial", a];
```


12.4.28 MatrixNorm

MatrixNorm::usage=

"MatrixNorm[x] normalizes a matrix x."

```
MatrixNorm[x_?MatrixQ] := Module[{},
  (*
  If[Norm[x]==0,
  Print["mNorm: Zero Norm Encountered. x=",MatrixForm[x]];
  ];
  *)
  x/Norm[x]
];

MatrixNorm[a___] := InvalidArg["MatrixNorm", a];
```

12.4.29 RootInput

RootInput::usage=

”RootInput[r] converts r into the list form representation for a root system if r is in string form. If r is in list form, then r itself is returned.”;

```
RootInput::arg = "Unknown argument.";

RootInput[r_?StringQ] := StringToRoot[r];

RootInput[r_?ListQ] := Module[
  {},

  If[SameQ[Depth[r], 2],
    Return[{r}];
  ,
    Return[r];
  ];

  Message[RootInput::arg];

  Return[Fail];
];

RootInput[a___] := InvalidArg["RootInput", a];
```

12.4.30 RootInputQ

RootInputQ::usage=

”RootInputQ[r] returns True if r is the name of a root system in list or string form.”;

```
RootInputQ[
  r_] := (!
    SameQ[r, {}] && (RootStringFormQ[r] || RootListFormQ[r]));

RootInputQ[a___] := False;
```

12.4.31 RootListFormQ

RootListFormQ::usage=

”RootListFormQ[r] returns True if r is the name of a root system in list form.”;

```
RootListFormQ[r_?ListQ] :=
  StringQ[r[[1]]] || (Depth[r] > 2 && StringQ[r[[1, 1]]])

RootListFormQ[a_...] := False;
```

12.4.32 RootStringFormQ

RootStringFormQ::usage=

”RootStringFormQ[r] returns True if r is the name of a root system in string form.”;

```
RootStringFormQ[r_] := StringQ[r];
```

```
RootStringFormQ[a___] := False;
```

12.4.33 RootSystemFromBasis

RootSystemFromBasis::usage=

"RootSystemFromBasis[d] gives the roots in the root system formed by basis roots d.";

```
RootSystemFromBasis[d_?RootBasisQ] := Module[
  {k, p, n, h, i, j, r, t, q, s, size, time, cntr, m},

  (* 1. Init *)
  p = d;
  n = 1;

  m = CartanMatrixFromBasis[d];

  size = 1;

  time = TimeUsed[];
  cntr = time;

  (* 2. Construct *)
  While[Length[p] > size,
    size = Length[p];

    For[i = 1, i <= Length[p], i++,
      For[j = 1, j <= Length[d], j++,
        t = p[[i]];

        (* 2.1.
        Get the basis coefficients and check height of the root *)

        k = BasisCoefficients[d, t];
        h = Sum[k[[s]], {s, 1, Length[k]}];

        If[h == n,
          (* 2.2. Determine the integer r *)
          r = 0;
          While[MemberQ[p, t - r*d[[j]]], r++];
          r = r - 1;

          (* 2.3. Define q *)

          q = r - Sum[k[[s]]*m[[s, j]], {s, 1, Length[d]}];

          (* 2.4. Ammend p? *)
          If[q > 0,
            p = Union[p, {t + d[[j]]}];
            ];

          (*
          If[TimeUsed[]-cntr>10,
            Print["(rootSystem) CPU Time:[",TimeUsed[]-time,"]   Roots:[",
              2*Length[p],"]"];
            cntr=TimeUsed[];
            ];
          *)
        ];
      ];
    ];

  n = n + 1;
```

```

];

(* If[TimeUsed[]-time>10,
  Print["(rootSystem) CPU Time:[",TimeUsed[]-time,"]   Roots:[",2*
Length[p],"   (done)"];
]; *)
(*
Print["(rootSystem) Used ",TimeUsed[]-time,
"s CPU Time to find ",2*Length[p]," roots."];
*)

Join[p, -p]
];

RootSystemFromBasis[a___] := InvalidArg["RootSystemFromBasis", a];

```

12.4.34 RowSwap

```

RowSwap::usage=
"RowSwap[m,a,b] swaps rows a and b in matrix m.";

RowSwap::invarg = "Invalid argument type. Input was '1', '2', '3'.";

RowSwap[m_?MatrixQ, a_?IntegerQ, b_?IntegerQ] := Module[
  {matr, rowa, rowb},

  matr = m;

  rowa = m[[a]];
  rowb = m[[b]];

  matr = Delete[matr, {{a}, {b}}];

  matr = Insert[matr, rowb, a];
  matr = Insert[matr, rowa, b];

  matr
];

RowSwap[a___] := InvalidArg["RowSwap", a];

```


12.4.35 SimpleRootBase

SimpleRootBase::usage=

"SimpleRootBase[r] gives a basis for an irreducible root system r."

```
SimpleRootBase[r_?RootInputQ] := Module[
  {s, type, dim, base, rin, i},

  rin = Flatten[RootInput[r]];
  type = rin[[1]];
  dim = rin[[2]];
  (*If[dim<1,Message[simplebase::type,r];{}];*)

  base = {};

  Switch[type,
    "A",
    If[dim == 1, base = {{1}}];,
    For[i = 1, i <= dim, i++,
      base =
        Join[base, {UnitVector[dim + 1, i] -
          UnitVector[dim + 1, i + 1]}];
    ];,
    "B",
    For[i = 1, i <= dim - 1, i++,
      base =
        Join[base, {UnitVector[dim, i] - UnitVector[dim, i + 1]}];
    ];
    base = Join[base, {UnitVector[dim, dim]}];,
    "C",
    For[i = 1, i <= dim - 1, i++,
      base =
        Join[base, {UnitVector[dim, i] - UnitVector[dim, i + 1]}];
    ];
    base = Join[base, {2 UnitVector[dim, dim]}];,
    "D",
    For[i = 1, i <= dim - 1, i++,
      base =
        Join[base, {UnitVector[dim, i] - UnitVector[dim, i + 1]}];
    ];
    base =
      Join[base, {UnitVector[dim, dim - 1] + UnitVector[dim, dim]}];,
    "E",
    Switch[dim,
      6,
      base = {{(1/
        2), -(1/2), -(1/2), -(1/2), -(1/2), -(1/2), -(1/2), (1/
        2)}, {1, 1, 0, 0, 0, 0, 0, 0}, {-1, 1, 0, 0, 0, 0, 0, 0},
        {0, -1, 1, 0, 0, 0, 0, 0}, {0, 0, -1, 1, 0, 0, 0, 0},
        {0, 0, 0, -1, 1, 0, 0, 0}};,
      7,
      base = {{(1/
        2), -(1/2), -(1/2), -(1/2), -(1/2), -(1/2), -(1/2), (1/
        2)}, {1, 1, 0, 0, 0, 0, 0, 0}, {-1, 1, 0, 0, 0, 0, 0, 0},
        {0, -1, 1, 0, 0, 0, 0, 0}, {0, 0, -1, 1, 0, 0, 0, 0},
        {0, 0, 0, -1, 1, 0, 0, 0}, {0, 0, 0, 0, -1, 1, 0, 0}};,
      8,
      base = {{(1/
        2), -(1/2), -(1/2), -(1/2), -(1/2), -(1/2), -(1/2), (1/
```

```

      2)}, {1, 1, 0, 0, 0, 0, 0, 0}, {-1, 1, 0, 0, 0, 0, 0, 0},
      {0, -1, 1, 0, 0, 0, 0, 0}, {0, 0, -1, 1, 0, 0, 0, 0},
      {0, 0, 0, -1, 1, 0, 0, 0}, {0, 0, 0, 0, -1, 1, 0, 0},
      {0, 0, 0, 0, 0, -1, 1, 0}};
    _, Message[lie::type, type]; base = {};
  ];
  "F",
  base = {{0, 1, -1, 0}, {0, 0, 1, -1}, {0, 0, 0,
    1}, {(1/2), -(1/2), -(1/2), -(1/2)}};
  "G",
  base = {{1, -1, 0}, {-2, 1, 1}};
  _, Message[lie::type, type]; {}];
base
];

SimpleRootBase[a___] := InvalidArg["SimpleRootBase", a];

```

12.4.36 StringToRoot

StringToRoot::usage=

"StringToRoot[str] converts a string str of the form $A_n+B_n+C_n...$ to the list form representation for a root system, where A, B, C are the root system type (A-G), and n is an integer."

```
StringToRoot::rootsystem = "'1' is not a valid root system.";
StringToRoot::parse =
  "'1' cannot be parsed. Valid syntax is <TYPE><DIM>+<TYPE><DIM>+...";

StringToRoot[str_?StringQ] := Module[
  {chars, i, r, cur, type, n, c},
  r = {};
  cur = {}; (* current irreducible component *)
  type = "X"; (*
  default type in case nothing was given *)
  n = 0; (*
  default dimension *)

  chars = Characters[str];

  For[i = 1, i <= Length[chars], i++,
    c = chars[[i]];

    Which[
      SameQ[c, "A"] || SameQ[c, "a"], type = "A";,
      SameQ[c, "B"] || SameQ[c, "b"], type = "B";,
      SameQ[c, "C"] || SameQ[c, "c"],
      If[SameQ[type, "B"], (* If last character was B *)

        type = "BC";,
        type = "C";
      ];

    ,
      SameQ[c, "D"] || SameQ[c, "d"], type = "D";,
      SameQ[c, "E"] || SameQ[c, "e"], type = "E";,
      SameQ[c, "F"] || SameQ[c, "f"], type = "F";,
      SameQ[c, "G"] || SameQ[c, "g"], type = "G";,
      SameQ[c, "+"],
      (* Do some error checking *)

      If[SameQ[type, "X"], Message[StringToRoot::rootsystem, str];
        Return[Fail];];

      cur = {type, n};
      r = Join[r, {cur}];
      type = "X";
      n = 0;

    ,
      (* If the input string was not one of the types,
      then it must be syntatically correct input *)
      ! SyntaxQ[c],
      Message[StringToRoot::arg, str]; Return[Fail];

    ,
      IntegerQ[ToExpression[c]],
      If[n == 0,
        n = ToExpression[c];
      ],
    ],
  ];
```

```

        n *= 10;
        n += ToExpression[c];
    ];
    ,
    _, Message[StringToRoot::arg, str]; Return[Fail];
    ];
];

(* Close the current irreducible component *)
cur = {type, n};
r = Join[r, {cur}];

(* Do some error checking *)

If[SameQ[type, "X"], Message[StringToRoot::rootssystem, str];
    Return[Fail];];

Return[r];
];

StringToRoot[a___] := InvalidArg["StringToRoot", a];

```

12.4.37 TakeElements

TakeElements::usage=

”TakeElements[l,in] forms a sub-list of list l which contains the entries indexed in integer list in.”;

```
TakeElements[l_?ListQ, {argin___?IntegerQ}] := Module[
  {i, in, out},

  in = List[argin];
  out = {};

  For[i = 1, i <= Length[l], i++,
    If[MemberQ[in, i],
      out = Join[out, {l[[i]]}];
    ];
  ];

  Return[out];
];

TakeElements[a___] := InvalidArg["TakeElements", a];
```

12.4.38 TakeRows

TakeRows::usage=

"TakeRows[m,r] creates from a given matrix m a second matrix consisting of rows listed in a list r."

```
TakeRows::invarg = "Invalid argument type. Input was '1', '2'."
```

```
TakeRows[matr_?MatrixQ, {argrows___?IntegerQ}] := Module[
  {i, mat, rows},
```

```
  rows = List[argrows];
  mat = {};
```

```
  For[i = 1, i <= Length[rows], i++,
    mat = Join[mat, Take[matr, {rows[[i]], rows[[i]]}]];
  ];
```

```
  mat
];
```

```
TakeRows[a___] := InvalidArg["TakeRows", a];
```

12.4.39 VectorPad

VectorPad::usage=

"VectorPad[v,p,t] takes a vector v and pads it on the left with p number of zeroes and enough zeroes on the right so that its total length is t.";

```
VectorPad[vector_?VectorQ, prelen_?IntegerQ, totallen_?IntegerQ] :=
Module[{v},
  v = vector;
  v = Join[ConstantArray[0, prelen], v];
  v = Join[v, ConstantArray[0, totallen - prelen - Length[v]]];
  v
];

VectorPad[a_...] := InvalidArg["VectorPad", a];
```

12.4.40 ZeroesAbove

ZeroesAbove::usage=

"ZeroesAbove[matr,i] returns True if each entry above the (i,i) entry in matrix matr is zero.";

```
ZeroesAbove[matr_?MatrixQ, i_?IntegerQ] := Module[
  {k},
  For[k = i - 1, k >= 1, k--,
    If[! SameQ[matr[[k, i]], 0], Return[False];];
  ];
  Return[True];
];
ZeroesAbove[a_...] := InvalidArg["ZeroesAbove", a];
```


12.4.41 ZeroesBelow

ZeroesBelow::usage=

”ZeroesBelow[matr,i] returns True if each entry below the (i,i) entry in matrix matr is zero.”;

```
ZeroesBelow[matr_?MatrixQ, i_?IntegerQ] := Module[
  {k},

  For[k = i + 1, k <= Length[matr], k++,
    If[! SameQ[matr[[k, i]], 0], Return[False];];
  ];

  Return[True];
];

ZeroesBelow[a___] := InvalidArg["ZeroesBelow", a];
```

12.4.42 ZeroesLeft

ZeroesLeft::usage=

”ZeroesLeft[matr,i] returns True if each entry to the left of the (i,i) entry in matrix matr is zero.”;

```
ZeroesLeft[matr_?MatrixQ, i_?IntegerQ] := Module[
  {k},
  For[k = i - 1, k >= 1, k--,
    If[! SameQ[matr[[i, k]], 0], Return[False];];
  ];
  Return[True];
];
ZeroesLeft[a___] := InvalidArg["ZeroesLeft", a];
```

12.4.43 ZeroesRight

ZeroesRight::usage=

”ZeroesRight[matr,i] returns True if each entry to the right of the (i,i) entry in matrix matr is zero.”;

```
ZeroesRight[matr_?MatrixQ, i_?IntegerQ] := Module[
  {k},

  For[k = i + 1, k <= Length[matr], k++,
    If[! SameQ[matr[[i, k]], 0], Return[False];];
  ];

  Return[True];
];

ZeroesRight[a_...] := InvalidArg["ZeroesRight", a];
```

12.5 Chevalley Structure Package (Primary)

12.5.1 ExtraSpecialPairs

ExtraSpecialPairs::usage=

”ExtraSpecialPairs[r] lists all pairs of roots which form extra special pairs.

ExtraSpecialPairs[r,rlist] lists all pairs of roots in the list rlist which form extra special pairs.”;

```
ExtraSpecialPairs[r_?RootInputQ] :=
  ExtraSpecialPairs[RootBase[r], PositiveRootSystem[r]];

ExtraSpecialPairs[d_?RootBasisQ] :=
  ExtraSpecialPairs[d, PositiveRootSystem[d]];

ExtraSpecialPairs[r_?RootInputQ, {argroots_?VectorQ}] :=
  ExtraSpecialPairs[RootBase[r], List[argroots]];

ExtraSpecialPairs[d_?RootBasisQ, {argroots_?VectorQ}] := Module[
  {pairs, roots, i, specs},

  pairs = {};
  roots = List[argroots];
  specs = SpecialPairs[d, roots];

  For[i = 1, i <= Length[specs], i++,
    If[ExtraSpecialPairQ[d, roots, specs, specs[[i, 1]],
      specs[[i, 2]]],
      pairs = Join[pairs, {specs[[i]]}];
    ];
  ];

  Return[pairs];
];
```

```
ExtraSpecialPairs[a___] := InvalidArg["ExtraSpecialPairs", a];
```

12.5.2 ExtraSpecialPairQ

ExtraSpecialPairQ::usage=

”ExtraSpecialPairQ[r,a,b] returns True if positive roots a and b (under root system r) form an extra special pair.

ExtraSpecialPairQ[r,rlist,a,b] returns True if positive roots a and b (under root system r and members of the set rlist) form an extra special pair. For repeated uses of this procedure with the same root system, it is suggested to pre-compute a list of positive roots and use this variation. WARNING: rlist is intended to be either the set of all positive roots, or the entire root system. False will be returned if the sum $a + b$ is not a member of rlist. If rlist is not at least the set of all positive roots, False can be returned in the incorrect circumstances.

ExtraSpecialPairQ[r,rlist,specs,a,b] returns True if positive roots a and b (under root system r and members of the set rlist with list of special pairs 'specs') form an extra special pair. For repeated uses of this procedure with the same root system, it is suggested to pre-compute a list of positive roots and special pairs, and use this variation. The same warning concerning rlist as in the previous variation applies. There is no such warning for the list 'specs'. If specs does not contain all special pairs in a root system, then only the extra special pairs within the given list will be marked.”;

```
ExtraSpecialPairQ[r_?RootInputQ, a_?VectorQ, b_?VectorQ] := Module[
  {basis, posroots, specs},

  basis = RootBase[r];
  posroots = PositiveRootSystem[basis];
  specs = SpecialPairs[basis, posroots];

  Return[ExtraSpecialPairQ[basis, posroots, specs, a, b]];
];

ExtraSpecialPairQ[d_?RootBasisQ, a_?VectorQ, b_?VectorQ] := Module[
  {basis, posroots, specs},

  posroots = PositiveRootSystem[d];
  specs = SpecialPairs[d, posroots];

  Return[ExtraSpecialPairQ[d, posroots, specs, a, b]];
];

ExtraSpecialPairQ[r_?RootInputQ, {argroots__?VectorQ}, a_?VectorQ,
  b_?VectorQ] := Module[
  {basis, posroots, specs},

  basis = RootBase[r];
  posroots = List[argroots];
  specs = SpecialPairs[basis, posroots];
```

```

Return[ExtraSpecialPairQ[basis, posroots, specs, a, b]];
];

ExtraSpecialPairQ[d_?RootBasisQ, {argroots__?VectorQ}, a_?VectorQ,
b_?VectorQ] := Module[
{basis, posroots, specs},

posroots = List[argroots];
specs = SpecialPairs[d, posroots];

Return[ExtraSpecialPairQ[d, posroots, specs, a, b]];
];

ExtraSpecialPairQ[r_?RootInputQ, {argroots__?VectorQ}, specs_?ListQ,
a_?VectorQ, b_?VectorQ] :=
ExtraSpecialPairQ[RootBase[r], List[argroots], specs, a, b];

ExtraSpecialPairQ[d_?RootBasisQ, {argroots__?VectorQ}, specs_?ListQ,
a_?VectorQ, b_?VectorQ] := Module[
{roots, ht, i},

roots = List[argroots];

If[! MemberQ[specs, {a, b}],
Return[False];
];

For[i = 1, i <= Length[specs], i++,
If[! SameQ[a + b, specs[[i, 1]] + specs[[i, 2]]],
Continue[];
];

If[SameQ[{a, b}, specs[[i]]],
Continue[];
];

If[! RootLessQ[d, a, specs[[i, 1]]],
Return[False];
];
];

Return[True];
];

ExtraSpecialPairQ[a___] := InvalidArg["ExtraSpecialPairQ", a];

```

12.5.3 SpecialPairs

SpecialPairs::usage=

”SpecialPairs[r] lists all pairs of roots which form special pairs.

SpecialPairs[r,rlist] lists all pairs of roots in the list rlist which form special pairs.”;

```
SpecialPairs[r_?RootInputQ] :=
  SpecialPairs[RootBase[r], PositiveRootSystem[r]];

SpecialPairs[d_?RootBasisQ] := SpecialPairs[d, PositiveRootSystem[d]];

SpecialPairs[r_?RootInputQ, {argroots_?VectorQ}] :=
  SpecialPairs[RootBase[r], List[argroots]];

SpecialPairs[d_?RootBasisQ, {argroots_?VectorQ}] := Module[
  {pairs, roots, i, j},

  pairs = {};
  roots = List[argroots];

  For[i = 1, i <= Length[roots], i++,
    For[j = 1, j <= Length[roots], j++,
      If[SpecialPairQ[d, roots, roots[[i]], roots[[j]]],
        pairs = Join[pairs, {{roots[[i]], roots[[j]]}}];
      ];
    ];

  Return[pairs];
];

SpecialPairs[a___] := InvalidArg["SpecialPairs", a];
```

12.5.4 SpecialPairQ

SpecialPairQ::usage=

”SpecialPairQ[r,a,b] returns True if positive roots a and b (under root system r) form a special pair.

SpecialPairQ[r,rlist,a,b] returns True if positive roots a and b (under root system r and members of the set rlist) form a special pair. For repeated uses of this procedure with the same root system, it is suggested to pre-compute a list of positive roots and use this variation. WARNING: rlist is intended to be either the set of all positive roots, or the entire root system. False will be returned if the sum $a + b$ is not a member of rlist. If rlist is not at least the set of all positive roots, False can be returned in the incorrect circumstances.”;

```
SpecialPairQ[r_?RootInputQ, a_?VectorQ, b_?VectorQ] :=
  SpecialPairQ[RootBase[r], a, b];

SpecialPairQ[d_?RootBasisQ, a_?VectorQ, b_?VectorQ] := Module[
  {posroots},

  posroots = PositiveRootSystem[d];

  If[! MemberQ[posroots, a],
    Return[False];
  ];

  If[! MemberQ[posroots, b],
    Return[False];
  ];

  If[! MemberQ[posroots, a + b],
    Return[False];
  ];

  Return[RootLessQ[d, a, b]];
];

SpecialPairQ[r_?RootInputQ, {argroots__?VectorQ}, a_?VectorQ,
  b_?VectorQ] := SpecialPairQ[RootBase[r], List[argroots], a, b];

SpecialPairQ[d_?RootBasisQ, {argroots__?VectorQ}, a_?VectorQ,
  b_?VectorQ] := Module[
  {roots, ht},

  roots = List[argroots];

  (* Caution:
  the user may have included negative roots in the passed list. *)

  ht = RootHeight[d, a];
  If[ht < 0,
    Return[False];
  ];

  ht = RootHeight[d, b];
  If[ht < 0,
```



```

Return[False];
];

If[! MemberQ[roots, a + b],
Return[False];
];

Return[RootLessQ[d, a, b]];
];

SpecialPairQ[a___] := InvalidArg["SpecialPairQ", a];

```

12.6 Weyl Package (Primary)

12.6.1 LongestElement

LongestElement::usage=

”LongestElement[r] computes the longest element of the Weyl group of root system r.

LongestElement[r,disks] computes the longest element of the Weyl group of a subsystem of root r which is formed by the basis roots indexed in disks.”;

```

LongestElement::irred = "'1' is not irreducible.";

LongestElement[r_?RootInputQ] :=
FundamentalChamber[-1*InteriorPoint[r], RootBase[r]];

LongestElement[r_?RootBasisQ] :=
FundamentalChamber[-1*InteriorPoint[r], r];

LongestElement[r_?RootInputQ, {argdisks___?IntegerQ}] :=
LongestElement[RootBase[r], List[argdisks]];

LongestElement[r_?RootBasisQ, {argdisks___?IntegerQ}] := Module[
{le, sr, disks},

disks = List[argdisks];
le = LongestElement[r];

sr = Table[i -> disks[[i]], {i, 1, Length[r]}];

Return[le /. sr];
];

LongestElement[a___] := InvalidArg["LongestElement", a];

```

12.6.2 ReflectWeyl

ReflectWeyl::usage=

"ReflectWeyl[d,a,b] calculates $S S \dots S$ (b) for root b and Weyl group element $a=a_1, a_2, \dots, a_n$, where all roots live in the set d.

```
ReflectWeyl[{argd_?VectorQ}, {arga_?IntegerQ}, b_?VectorQ] :=
Module[
  {a, d, i, root, dIndex},

  d = List[argd];
  a = List[arga];
  root = b;

  For[i = Length[a], i >= 1, i--,
    dIndex = a[[i]];
    root = Reflect[d[[dIndex]], root];
  ];

  Return[root]
];

ReflectWeyl[a_?IntegerQ] := InvalidArg["ReflectWeyl", a];
```

12.6.3 WeylCompare

WeylCompare::usage=

"WeylCompare[d,w1,w2] returns True if $w1 = w2$. w1, w2 are elements of Weyl(d).";

```
WeylCompare[{argd_?VectorQ}, {argw1___?IntegerQ}, {argw2___?
  IntegerQ}] := Module[
  {w1, w2, d, i, r1, r2},

  w1 = List[argw1];
  w2 = List[argw2];
  d = List[argd];

  For[i = 1, i <= Length[d], i++,
    r1 = ReflectWeyl[d, w1, d[[i]]];
    r2 = ReflectWeyl[d, w2, d[[i]]];
    If[r1 != r2,
      Break[];
    ];
  ];

  i > Length[d]
];

WeylCompare[a___] := InvalidArg["WeylCompare", a];
```

12.6.4 WeylLength

WeylLength::usage=

"WeylLength[d,w] computes the length of Weyl group element w. w is in the Weyl group for a root system with basis d.";

```
WeylLength[d_?ListQ, {argw___?IntegerQ}] := Module[
  {w, wr},

  w = List[argw];
  wr = WeylReduce[d, w];

  Length[wr]
];

WeylLength[a___] := InvalidArg["WeylLength", a];
```

12.6.5 WeylReduce

WeylReduce::usage=

"WeylReduce[d,w] uses the Deletion Property to write a Weyl group element w reduced. w is in the Weyl group for a root system with basis d.";

```
WeylReduce[d_?ListQ, {argw___?IntegerQ}] := Module[
  {w, i, j, wp, test},

  w = List[argw];
  test = 0;

  (* Look for pairs i,j which can be deleted *)

  For[i = 1, i <= Length[w], i++,
    For[j = 1, j <= Length[w], j++,
      wp = Delete[w, {{i}, {j}}];

      If[WeylCompare[d, w, wp],
        test = 1;
        Break[];
      ];
    ];

  If[test == 1,
    Break[];
  ];

  (* i,j found. Recurse. Else return w *)
  If[i <= Length[w],
    WeylReduce[d, wp]
  ,
    w
  ]
];

WeylReduce[a___] := InvalidArg["WeylReduce", a];
```

12.7 Weyl Package (Internal)

12.7.1 InteriorPoint

InteriorPoint::usage=

"InteriorPoint[r] computes an interior point in the fundamental chamber with respect to root system r."

```

InteriorPoint[r_?RootInputQ] := Module[
  {basis, v, x, i, eqns, soln},
  (* REF: Maple / Stembridge *)

  basis = RootBase[r];

  v = Sum[x[i]*basis[[i]], {i, 1, Length[basis]}];
  v = Collect[v, Table[x[i], {i, 1, Length[basis]}]];
  eqns =
    Table[InnerProduct[basis[[i]], v] == 1, {i, 1, Length[basis]}];
  soln = Flatten[Solve[eqns, Table[x[i], {i, 1, Length[basis]}]]];
  Sum[basis[[i]]*x[i], {i, 1, Length[basis]}] /. soln
];

InteriorPoint[r_?RootBasisQ] := Module[
  {basis, v, x, i, eqns, soln},
  (* REF: Maple / Stembridge *)

  basis = r;

  v = Sum[x[i]*basis[[i]], {i, 1, Length[basis]}];
  v = Collect[v, Table[x[i], {i, 1, Length[basis]}]];
  eqns =
    Table[InnerProduct[basis[[i]], v] == 1, {i, 1, Length[basis]}];
  soln = Flatten[Solve[eqns, Table[x[i], {i, 1, Length[basis]}]]];
  Sum[basis[[i]]*x[i], {i, 1, Length[basis]}] /. soln
];

InteriorPoint[a___] := InvalidArg["InteriorPoint", a];

```

12.7.2 FundamentalChamber

FundamentalChamber::usage=

”FundamentalChamber[v,r] maps a vector v to the fundamental chamber with respect to a root system r.”;

```

FundamentalChamber[pt_?VectorQ, r_?RootInputQ] :=
  FundamentalChamber[pt, RootBase[r]];

FundamentalChamber[pt_?VectorQ, basis_?RootBasisQ] := Module[
  {v, w, i},

  (* REF: Maple / Stembridge *)
  v = pt;
  w = {};

  While[True,
    For[i = 1, i <= Length[basis], i++,
      If[InnerProduct[basis[[i]], v] < 0,
        w = Join[w, {i}];
        v = Reflect[basis[[i]], v];
        Break[];
      ];
    ];

    If[i > Length[basis], Break[]];
  ];

  Return[w];
];

FundamentalChamber[a_...] := InvalidArg["FundamentalChamber", a];

```

12.8 Group Action Package (Primary)

12.8.1 ArchesListInvolution

ArchesListInvolution::usage=

”ArchesListInvolution[r,disks,theta] recovers the diagram automorphism from an involution theta defined over a root system r with fixed roots: disks.”;

```

ArchesListInvolution::order =
  "supplied root system automorphism over '1' is not an involution: \
  '2' order '3'.";

(* Backward compatibility *)

ArchesListInvolution[r_?RootInputQ, {argdisks___?IntegerQ},
  theta_?MatrixQ] := ArchesListInvolution[RootBase[r], theta];

(* Backward compatibility *)

ArchesListInvolution[d_?RootBasisQ, {argdisks___?IntegerQ},

```

```

theta_?MatrixQ] := ArchesListInvolution[d, theta];

ArchesListInvolution[r_?RootInputQ, theta_?MatrixQ] :=
  ArchesListInvolution[RootBase[r], theta];

ArchesListInvolution[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {thetastar, w0, tm, arlist, i, plist, p},

  arlist = {};
  plist = {}; (*
  A list of roots processed so that we don't include duplicates e.g.
  both {1,2} and {2,1} *)

  (* Get the matrix for the diagram automorphisms *)

  w0 = wInvolutionAction[d, theta];
  thetastar = -theta.w0;

  (* Write the list of translations. *)

  tm = Transpose[thetastar]; (* Now tm[[
  i]] is the column i of thetastar *)

  For[i = 1, i <= Length[d], i++,
    If[! SameQ[tm[[i, i]], 1] && ! MemberQ[plist, i],
      p = Position[tm[[i]], 1][[1, 1]];
      plist = Join[plist, {p}];
      plist = Join[plist, {i}];

      arlist = Join[arlist, {{i, p}}];
    ];
  ];

  Return[arlist];
];

(*
ArchesListInvolution[d_?RootBasisQ,{argdisks___?IntegerQ},theta_?
MatrixQ]:=Module[
{i,j,autom,blockaut,btheta,blocklist,offset,disks,ntheta,esystems,\
esystemsflat,arlist},

disks=List[argdisks];

If[!IsRootAutOrder[theta,2],
Message[ArchesListInvolution::order,BasisToRootSystem[d],theta,\
MatrixForm[LinearOperatorOrder[theta]]];
Return[{}];
];

(* Identify any roots mapped to a different irreducible system. \
Ammend to arches, then reverse the maps to create ntheta. to perform \
the reversal of the arch, multiply theta by the permutation matrix \
(i,j) where {i,j} is the detected arch. *)
\
esystems=RootBasisConnectedSet[d];
esystemsflat=Flatten[esystems,1]; (* all roots in one list *)
\
arlist={};

For[i=1,i<=Length[esystems],i++,
For[j=1,j<=Length[esystems[[i]]],j++,

```



```

];
];

ntheta=theta;
Print[esystems];

(* Now pick apart arches within a single system *)
\
blocklist=BlockList[ntheta];
Print["bl=",blocklist];
autom={};
offset=0;
Print["ali1."];
(* For all irred components *)
For[i=1,i<=Length[esystems],i++,
(* Due to a result in [Hel88], the diagram automorphism is either the \
identity, or of order 2. *)
(* blocklist[[i]] is theta restricted to \
irred. root system i. *)

(* Create the map with identity diagram automorphism. *)
\
btheta=RootInvolution[esystems[[i]],disks,{}];
Print[blocklist[[2]]];
Print["ali2.",BasisToRootSystem[d]];
(* If it matches blocklist[[i]], then we've found the right diagram \
automorphism. Otherwise, return the autom. of order 2. *)
\
If[!SameQ[btheta,blocklist[[i]]],
Print["es=",esystems[[i]]];
autom=Join[autom,DiagramInvolution[BasisToRootSystem[esystems[[i]]]]+\
offset];
];

offset+=Length[esystems[[i]]];
];

Print[autom];

Return[Join[arlist,autom]];
];
*)

ArchesListInvolution[{argdisks___?IntegerQ}, theta_?MatrixQ] :=
Module[
{disks, arches, i, j, ind},

arches = {};
disks = List[argdisks];

If[! IsRootAutOrder[theta, 2],
Message[ArchesListInvolution::order, "root system", theta,
MatrixForm[LinearOperatorOrder[theta]]];
Return[{}];
];

(* For every entry in disks i, if the entry (i,
i) in theta matrix is not 1,
find the proper row to j to swap with row i so that (i,
i) entry is 1. *)

```

```

For[i = 1, i <= Length[disks], i++,
  ind = disks[[i]];

  If[SameQ[theta[[ind, ind]], 1],
    Continue[];
  ];

  (*
  Print["x"];
  *)

  (* Only look forward so that we don't write the same pair twice. *)

  For[j = i + 1, j <= Length[disks], j++,
    If[! SameQ[theta[[disks[[j]], ind]], 1],
      Continue[];
    ];

    arches = Join[arches, {{ind, disks[[j]]}}];
  ];
];

Return[arches];
];

ArchesListInvolution[a___] := InvalidArg["ArchesListInvolution", a];

(* PRE 0.0.95.... cvals not updated, uses old memory structure
liftThetaEqns[basis_, theta_, systemMatrix_, cvals_, x_] := Module[
{n, polynomials, vars, i},

n=Length[theta];

polynomials=Table[(utilLTsidea[n,i,systemMatrix,x])*(utilLTsideb[n,i,\
theta,systemMatrix,x])*cvals[UnitVector[n,i].basis]*cvals[UnitVector[\
n,i].Transpose[theta].basis]-1,{i,1,n}];
vars=Table[x[i],{i,1,n}];

Return[{polynomials,vars}];
];
*)

```

12.8.2 DiskList

DiskList::usage=

”DiskList[r,theta] is a variant of fixedBasis which lists the indices of the roots fixed by an involution theta over a root system r.”;

```

DiskList[r_?RootInputQ, theta_?MatrixQ] :=
  DiskList[RootBase[r], theta];

DiskList[r_?RootBasisQ, theta_?MatrixQ] := Module[
  {flist, basis, i},

  basis = r;
  flist = {};

  For[i = 1, i <= Length[theta], i++,
    If[SameQ[basis[[i]], ApplyRootInvolution[r, theta, basis[[i]]]],
      flist = Join[flist, {i}];
    ];
  ];

  Return[flist];
];

DiskList[a___] := InvalidArg["DiskList", a];

```

12.8.3 EigenspaceProject

EigenspaceProject::usage=

”EigenspaceProject[r,theta,root,E] projects root into some root in the E-eigenspace of the root automorphism theta over root system r.

(S-LOSS) EigenspaceProject[r,disks,arches,root,E] projects root into some root in the E-eigenspace of the root automorphism described by fixed roots disks and diagram automorphism arches over root system r.”;

```
EigenspaceProject::evaluate =
  "‘1’ is not an Eigenvalue of the automorphism provided.";

EigenspaceProject[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ,
  espace_] := EigenspaceProject[RootBase[r], theta, root, espace];

EigenspaceProject[d_?RootBasisQ, theta_?MatrixQ, root_?VectorQ,
  espace_] := Module[
  {nroot, pr, n, e, i},

  If[! MemberQ[Eigenvalues[theta], espace],
    Message[EigenspaceProject::evaluate, espace];
    Return[Fail];
  ];

  nroot = ApplyRootMap[d, theta, root];

  n = LinearOperatorOrder[theta];
  e = espace;

  pr = (1/n)*
    Sum[Power[e, n - i]*
      ApplyRootMap[d, MatrixPower[theta, i], root], {i, 0, n - 1}];
  pr = FullSimplify[pr];

  (*
  pr=(1/2)*(root+espace*nroot);
  *)

  Return[pr];
];

EigenspaceProject[a___] := InvalidArg["EigenspaceProject", a];
```

12.8.4 EmbeddedRootGroups

EmbeddedRootGroups::usage=

”EmbeddedRootGroups[r,theta] gives a list of the roots fixed by an automorphism theta over root system r. The list groups together roots by membership in an embedded root system.

EmbeddedRootGroups[r,disks] gives a list of the roots fixed by an involution theta defined with fixed roots disks and diagram involution arches over root system r. The list groups together roots by membership in an embedded root system.”;

```
EmbeddedRootGroups[r_?RootInputQ, theta_?MatrixQ] :=
  EmbeddedRootGroups[RootBase[r], theta];

EmbeddedRootGroups[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {dlist, sbasis},

  dlist = DiskList[d, theta];
  sbasis = TakeElements[d, dlist];

  Return[RootBasisConnectedSet[sbasis]];
];

EmbeddedRootGroups[r_?RootInputQ, {argdisks___?IntegerQ}] :=
  EmbeddedRootGroups[RootBase[r], List[argdisks]];

EmbeddedRootGroups[d_?RootBasisQ, {argdisks___?IntegerQ}] := Module[
  {disks, sbasis},

  disks = List[argdisks];
  sbasis = TakeElements[d, disks];

  Return[RootBasisConnectedSet[sbasis]];
];

EmbeddedRootGroups[a___] := InvalidArg["EmbeddedRootGroups", a];
```

12.8.5 EmbeddedRootIndices

EmbeddedRootIndices::usage=

” EmbeddedRootIndices[r,theta] gives an index of the roots fixed by an involution theta over root system r. The list groups together roots by membership in an embedded root system.

EmbeddedRootIndices[r,disks] gives an index of the roots fixed by an involution theta defined with fixed roots disks and diagram involution arches over root system r. The list groups together roots by membership in an embedded root system.”;

```
EmbeddedRootIndices[r_?RootInputQ, theta_?MatrixQ] :=
  EmbeddedRootIndices[RootBase[r], theta];

EmbeddedRootIndices[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {esystems, i, j, subind, ind},

  ind = {};
  esystems = EmbeddedRootGroups[d, theta];

  For[i = 1, i <= Length[esystems], i++,
    subind = {};

    For[j = 1, j <= Length[esystems[[i]]], j++,
      subind = Join[subind, Flatten[Position[d, esystems[[i, j]]]]];
    ];

    ind = Join[ind, {subind}];
  ];

  Return[ind];
];

EmbeddedRootIndices[r_?RootInputQ, {argdisks___?IntegerQ}] :=
  EmbeddedRootIndices[RootBase[r], List[argdisks]];

EmbeddedRootIndices[d_?RootBasisQ, {argdisks___?IntegerQ}] := Module[
  {disks, esystems, i, j, subind, ind},

  ind = {};
  disks = List[argdisks];
  esystems = EmbeddedRootGroups[d, disks];

  For[i = 1, i <= Length[esystems], i++,
    subind = {};

    For[j = 1, j <= Length[esystems[[i]]], j++,
      subind = Join[subind, Flatten[Position[d, esystems[[i, j]]]]];
    ];

    ind = Join[ind, {subind}];
  ];

  Return[ind];
];

EmbeddedRootIndices[a___] := InvalidArg["EmbeddedRootIndices", a];
```

12.8.6 EmbeddedRootSystems

EmbeddedRootSystems::usage=

"EmbeddedRootSystems[r,theta] gives the embedded root systems formed by the root involution theta over a root system r.

EmbeddedRootSystems[r,disks] gives the embedded root systems formed by the fixed roots disks of a root involution over a root system r."

```
EmbeddedRootSystems[r_?RootInputQ, {argdisks___?IntegerQ}] :=
  EmbeddedRootSystems[RootBase[r], List[argdisks]];

EmbeddedRootSystems[r_?RootBasisQ, {argdisks___?IntegerQ}] := Module[
  {disks, esystems, out, i},

  out = "";
  disks = List[argdisks];
  esystems = EmbeddedRootGroups[r, disks];

  For[i = 1, i <= Length[esystems], i++,
    out = out <> BasisToRootSystem[esystems[[i]]];
    (*Print[esystems[[i]]];*)
  ];

  Return[out];
];

EmbeddedRootSystems[r_?RootInputQ, theta_?MatrixQ] :=
  EmbeddedRootSystems[RootBase[r], theta];

EmbeddedRootSystems[r_?RootBasisQ, theta_?MatrixQ] := Module[
  {disks, esystems, out, i},

  out = "";
  disks = DiskList[r, theta];
  esystems = EmbeddedRootGroups[r, disks];

  For[i = 1, i <= Length[esystems], i++,
    out = out <> BasisToRootSystem[esystems[[i]]];
    (*Print[esystems[[i]]];*)
  ];

  Return[out];
];

EmbeddedRootSystems[a___] := InvalidArg["EmbeddedRootSystems", a];
```

12.8.7 FixedBasis

FixedBasis::usage=

”FixedBasis[r,theta] computes the set of all basis roots fixed by an involution theta defined over root system r.

(S-LOSS) FixedBasis[r,disks,arches] computes the set of all basis roots fixed by an involution defined over root system r with fixed roots disks and diagram automorphism arches.”;

```
FixedBasis[r_?RootInputQ, theta_?MatrixQ] :=
  Intersection[RootBase[r], FixedRoots[r, theta]];

FixedBasis[r_?RootBasisQ, theta_?MatrixQ] :=
  Intersection[r, FixedRoots[r, theta]];

FixedBasis[a___] := InvalidArg["FixedBasis", a];

(*
stronglyOrthoRoots[r_,disks_,arches_]:=Module[
{fr,sor,i,j,osize},

fr=fixedRoots[r,disks,arches];
sor=fr;
osize=0;

While[Length[sor]!=osize,
osize=Length[sor];

For[i=1,i<=Length[sor],i++,
For[j=1,j<=Length[sor],j++,
If[i==j,Continue[]];];

If[MemberQ[sor,sor[[i]]+sor[[j]]]||MemberQ[sor,sor[[i]]-sor[[j]]],
sor=Delete[sor,Position[sor,sor[[i]]][[1,1]]];
Break[]];];
];
];

Return[sor];
];
*)

(*
stronglyOrthoRoots[r_,theta_]:=Module[
{fr,sor,i,j,osize},

fr=fixedRoots[r,theta];
sor=fr;
osize=0;

While[Length[sor]!=osize,
osize=Length[sor];

For[i=1,i<=Length[sor],i++,
For[j=1,j<=Length[sor],j++,
If[i==j,Continue[]];];
```



```

If[MemberQ[sor,sor[[i]]+sor[[j]]]||MemberQ[sor,sor[[i]]-sor[[j]]],
sor=Delete[sor,Position[sor,sor[[i]]][[1,1]]];
Break[]];
];
];
];

Return[sor];
];
*)

(*
stronglyOrthoRoots2[r_,disks_,arches_]:=Module[
{fr,sor,i,j},

fr=fixedRoots[r,disks,arches];
sor=Intersection[fr,base[r]];

(*For[i=1,i<=Length[fr],i++,*)
For[i=Length[fr],i>=1,i--,
For[j=1,j<=Length[sor],j++,
If[i==j,Continue[]];

If[MemberQ[sor,fr[[i]]+sor[[j]]]||MemberQ[sor,fr[[i]]-sor[[j]]],Break[
];];
];

If[j>Length[sor],
sor=Union[sor,{fr[[i]]}];
];
];

Return[sor];
];
*)

```

12.8.8 FixedRootQ

FixedRootQ::usage=

”FixedRootQ[r,theta,root] returns True if root is fixed by the involution theta defined over the root system r.

(S-LOSS) FixedRootQ[r,disks,arches,root] returns True if root is fixed by the involution defined over the root system r with fixed roots disks and diagram automorphism arches.”;

```
FixedRootQ[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ] := Module[
  {k, basis, rk},

  basis = RootBase[r];
  k = BasisCoefficients[basis, root];

  rk = k.Transpose[theta].basis;

  Return[SameQ[root, rk]];
];

FixedRootQ[basis_?RootBasisQ, theta_?MatrixQ, root_?VectorQ] :=
Module[
  {k, rk},

  k = BasisCoefficients[basis, root];

  rk = k.Transpose[theta].basis;

  Return[SameQ[root, rk]];
];

FixedRootQ[a___] := InvalidArg["FixedRootQ", a];
```

12.8.9 FixedRoots

FixedRoots::usage=

”FixedRoots[r,theta] returns the set of all roots fixed by the involution theta defined over the root system r.

(S-LOSS) FixedRoots[r,disks,arches] returns the set of all roots fixed by the involution defined over the root system r with fixed roots disks and diagram automorphism arches.”;

```
FixedRoots[r_?RootInputQ, theta_?MatrixQ] := Module[
  {i, roots, froots},

  roots = RootSystemFromBasis[RootBase[r]];
  froots = {};

  For[i = 1, i <= Length[roots], i++,
    If[FixedRootQ[r, theta, roots[[i]]],
      froots = Join[fruits, {roots[[i]]}];
    ];
  ];

  Return[fruits];
];

FixedRoots[basis_?RootBasisQ, theta_?MatrixQ] := Module[
  {i, roots, froots},

  roots = RootSystemFromBasis[basis];
  froots = {};

  For[i = 1, i <= Length[roots], i++,
    If[FixedRootQ[basis, theta, roots[[i]]],
      froots = Join[fruits, {roots[[i]]}];
    ];
  ];

  froots = ByBasisSort[basis, froots];

  Return[fruits];
];

FixedRoots[a___] := InvalidArg["FixedRoots", a];
```

12.8.10 IsRootAutOrder

IsRootAutOrder::usage=

”IsRootAutOrder[theta,n] determines if an automorphism theta is of order n, or an order that divides n.”;

```
IsRootAutOrder[theta_?MatrixQ, n_?IntegerQ] := Module[
  {order},

  order = LinearOperatorOrder[theta];

  Return[SameQ[Mod[n, order], 0]];
];

IsRootAutOrder[a___] := InvalidArg["IsRootAutOrder", a];
```

12.8.11 wInvolution

wInvolution::usage=

"wInvolution[r,theta] computes the longest element of the root system formed by the roots fixed by theta, a root system automorphism.

wInvolution[r,disks] computes the longest element of the root system formed by the embedded roots (disks).";

```
wInvolution[r_?RootBasisQ, {argdisks___?IntegerQ}] :=
  wInvolution[BasisToRootSystem[r], List[argdisks]];

wInvolution[r_?RootBasisQ, theta_?MatrixQ] :=
  wInvolution[BasisToRootSystem[r], DiskList[r, theta]];

wInvolution[r_?RootInputQ, theta_?MatrixQ] :=
  wInvolution[r, DiskList[r, theta]];

wInvolution[r_?RootInputQ, {argdisks___?IntegerQ}] := Module[
  {rin, i, w, offset, disks},

  disks = List[argdisks];
  rin = RootInput[r];
  offset = 0;
  w = {};

  For[i = 1, i <= Length[rin], i++,
    w = Join[w, wInvolutionSimple[rin[[i]], disks, offset]];

    offset = offset + rin[[i, 2]];
  ];

  Return[w];
];

wInvolutionSimple[r_, disks_, offSet_: 0] := Module[
  {rin, basis, dim, esystems, w0, i, emsys, ds, w},

  rin = Flatten[RootInput[r]]; (* r is simple *)

  basis = RootBase[rin];
  dim = Length[basis];

  (* 1. Determine the embedded root systems *)

  esystems = EmbeddedRootSystems[rin, disks - offSet];

  (* For every root system *)
  w0 = {};

  For[i = 1, i <= Length[esystems], i++,
    (* Ugh. Have to do some "E-
    hacking" because of the numbering of the roots of E
    The 1,2,3,... disks labels here assumes r is irred.
    Otherwise we'd need to account for offset. *)

    emsys = Flatten[RootInput[esystems[[i, 1]]]]; (*
    embedded system *)
```

```

If[SameQ[rin[[1]], "E"] && ! SameQ[emsys[[1]], "E"] &&
  MemberQ[esystems[[i, 2]], 2] && MemberQ[esystems[[i, 2]], 3],
  (* The problem is if we have a root system of Type E then roots \
2 and 3 are not joined.
  This results in "weird" Cartan Matrices for types A, D. *)

  ds = esystems[[i, 2]];

  (* If we have 1 and 3 included AND the embedded root system is \
not E, then 6 is not included.
  So if 5 is included, we have type D5.
  If 5 is not included, we have type A4 *)

  If[MemberQ[esystems[[i, 2]], 1],
    (* 5 included? *)
    If[MemberQ[esystems[[i, 2]], 5],
      ds = {1, 3, 4, 2, 5};
      , (* else *)
      ds = {1, 3, 4, 2};
    ];
    , (* else 1 not included *)
    (*
    Without 1 then we have type D for sure.
    Simple thing to do is just run through all cases for disk sets *)

    Switch[Max[esystems[[i, 2]]],
      8, ds = {8, 7, 6, 5, 4, 2, 3};,
      7, ds = {7, 6, 5, 4, 2, 3};,
      6, ds = {6, 5, 4, 2, 3};,
      5, ds = {3, 4, 2, 5};,
      4, ds = {3, 4, 2};
    ];
  ];
  w = LongestElement[esystems[[i, 1]], ds];
  ,
  w = LongestElement[esystems[[i, 1]], esystems[[i, 2]]];
];

w0 = Join[w0, w];
];

Return[w0 + offSet];
];

(*
wInvolution[r_?RootBasisQ, theta_?MatrixQ] := Module[
{disks, basis, sbasis, esystems, i, le},

disks = DiskList[r, theta];
basis = r;

esystems = EmbeddedRootIndices[r, disks];

le = {};
For[i = 1, i <= Length[esystems], i++,
sbasis = TakeElements[basis, esystems[[i]]];
le = Join[le, LongestElement[sbasis, esystems[[i]]]];
];

Return[le];
];
*)

```

```

(*)
wInvolution[r_?RootInputQ,theta_?MatrixQ]:=Module[
{disks,basis,sbasis,rin,esystems,le,i},

rin=RootInput[r];
disks=DiskList[r,theta];
basis=RootBase[rin];

esystems=EmbeddedRootIndices[r,disks];

le={};
For[i=1,i<=Length[esystems],i++,
sbasis=TakeElements[basis,esystems[[i]]];
le=Join[le,LongestElement[sbasis,esystems[[i]]]];
];

Return[le];
];
*)

(*)
wInvolution[r_?RootBasisQ,{argdisks___?IntegerQ}]:=Module[
{disks,basis,sbasis,esystems,i,le},

disks=List[argdisks];
basis=r;

esystems=EmbeddedRootIndices[r,disks];

le={};
For[i=1,i<=Length[esystems],i++,
sbasis=TakeElements[basis,esystems[[i]]];
le=Join[le,LongestElement[sbasis,esystems[[i]]]];
];

Return[le];
];
*)

(*)
wInvolution[r_?RootInputQ,{argdisks___?IntegerQ}]:=Module[
{disks,basis,sbasis,rin,esystems,le,i},

rin=RootInput[r];
disks=List[argdisks];
basis=RootBase[rin];

esystems=EmbeddedRootIndices[r,disks];

le={};
For[i=1,i<=Length[esystems],i++,
sbasis=TakeElements[basis,esystems[[i]]];
le=Join[le,LongestElement[sbasis,esystems[[i]]]];
];

Return[le];
];
*)

wInvolution[a___] := InvalidArg["wInvolution", a];

```

12.8.12 wInvolutionAction

wInvolutionAction::usage=

"wInvolutionAction[r,theta] computes the matrix representing the action of the longest element of the Weyl group formed by the root system consisting of the roots fixed by theta, the root system automorphism over root system r.

wInvolutionAction[r,disks] computes the matrix representing the action of the longest element of the Weyl group formed by the embedded root systems (disks), where r is the root system."

```
wInvolutionAction[r_?RootInputQ, theta_?MatrixQ] := Module[
  {w0, basis, row, w0theta, i},

  basis = RootBase[r];
  w0 = wInvolution[r, theta];

  For[i = 1, i <= Length[basis], i++,
    row[i] =
      BasisCoefficients[basis, ReflectWeyl[basis, w0, basis[[i]]]];
  ];

  w0theta = Table[row[i], {i, 1, Length[basis]}];

  w0theta = Transpose[w0theta];

  Return[w0theta];
];

wInvolutionAction[r_?RootBasisQ, theta_?MatrixQ] := Module[
  {w0, basis, row, w0theta, i},

  basis = r;
  w0 = wInvolution[r, theta];

  For[i = 1, i <= Length[basis], i++,
    row[i] =
      BasisCoefficients[basis, ReflectWeyl[basis, w0, basis[[i]]]];
  ];

  w0theta = Table[row[i], {i, 1, Length[basis]}];

  w0theta = Transpose[w0theta];

  Return[w0theta];
];

wInvolutionAction[r_?RootInputQ, {argdisks___?IntegerQ}] := Module[
  {disks, w0, basis, row, w0theta, i},

  disks = List[argdisks];
  basis = RootBase[r];
  w0 = wInvolution[r, disks];

  For[i = 1, i <= Length[basis], i++,
    row[i] =
      BasisCoefficients[basis, ReflectWeyl[basis, w0, basis[[i]]]];
  ];
```



```

];

w0theta = Table[row[i], {i, 1, Length[basis]}];

w0theta = Transpose[w0theta];

Return[w0theta];
];

wInvolutionAction[r_?RootBasisQ, {argdisks___?IntegerQ}] := Module[
{disks, w0, basis, row, w0theta, i},

disks = List[argdisks];
basis = r;

w0 = wInvolution[r, disks];

For[i = 1, i <= Length[basis], i++,
row[i] =
BasisCoefficients[basis, ReflectWeyl[basis, w0, basis[[i]]]];
];

w0theta = Table[row[i], {i, 1, Length[basis]}];

w0theta = Transpose[w0theta];

Return[w0theta];
];

wInvolutionAction[a___] := InvalidArg["wInvolutionAction", a];

```

12.9 Group Action Package (Diagram)

12.9.1 DynkinPointsTeX

DynkinPointsTeX::usage=

”DynkinPointsTeX[type,dim,xOff,yOff] provides a LaTeX-formatted list of relative x,y positions of the dots of a Dynkin diagram for a root system of type (type,dim). The points are offset along the x and y axes by xOff and yOff respectively.”;

```

DynkinPointsTeX[type_?StringQ, dim_?IntegerQ, xOffset_?IntegerQ,
yOffset_?IntegerQ] := Module[
{i, points, xSize, ySize},

xSize = 25; (* Space between points on X axis *)
ySize = 24; (*
Space between points on y axis *)

Switch[type,
"A",
points =
Table[{xSize*i + xOffset, 0 + yOffset}, {i, 0, dim - 1}];,
"B",
points =
Table[{xSize*i + xOffset, 0 + yOffset}, {i, 0, dim - 1}];,

```

```

"C",
points =
  Table[{xSize*i + xOffset, 0 + yOffset}, {i, 0, dim - 1}],
"D",
points =
  Table[{xSize*i + xOffset, 0 + yOffset}, {i, 0, dim - 3}],
points =
  Join[points, {{xSize*(dim - 2) + xOffset, (ySize/2) +
    yOffset}, {xSize*(dim - 2) + xOffset, -(ySize/2) +
    yOffset}}];
"E",
Switch[dim,
  6,
    points = {{0 + xOffset, 0 + yOffset}, {2*xSize + xOffset,
      ySize + yOffset}, {xSize + xOffset,
        0 + yOffset}, {2*xSize + xOffset,
          0 + yOffset}, {3*xSize + xOffset,
            0 + yOffset}, {4*xSize + xOffset, 0 + yOffset}};,
  7,
    points = {{0 + xOffset, 0 + yOffset}, {2*xSize + xOffset,
      ySize + yOffset}, {xSize + xOffset,
        0 + yOffset}, {2*xSize + xOffset,
          0 + yOffset}, {3*xSize + xOffset,
            0 + yOffset}, {4*xSize + xOffset,
              0 + yOffset}, {5*xSize + xOffset, 0 + yOffset}};,
  8,
    points = {{0 + xOffset, 0 + yOffset}, {2*xSize + xOffset,
      ySize + yOffset}, {xSize + xOffset,
        0 + yOffset}, {2*xSize + xOffset,
          0 + yOffset}, {3*xSize + xOffset,
            0 + yOffset}, {4*xSize + xOffset,
              0 + yOffset}, {5*xSize + xOffset,
                0 + yOffset}, {6*xSize + xOffset, 0 + yOffset}};
];
,
"F",
points = {{0 + xOffset, 0 + yOffset}, {xSize + xOffset,
  0 + yOffset}, {2*xSize + xOffset,
    0 + yOffset}, {3*xSize + xOffset, 0 + yOffset}};,
"G",
points = {{0 + xOffset, 0 + yOffset}, {xSize + xOffset,
  0 + yOffset}};
,
-,
Message[lie::type, type];
];

points
];

DynkinPointsTeX[a_...] := InvalidArg["DynkinPointsTeX", a];

```

12.9.2 HelminckDiagram

HelminckDiagram::usage=

”HelminckDiagram[r,theta,labels] gives the Helminck Diagram of an involution theta on the roots acting on root system r. The optional argument labels allows custom labels for the simple roots.

HelminckDiagram[r,disks,arches,labels] gives the Helminck Diagram of an involution on the roots acting on root system r. The involution is described by the fixed simple roots (disks) and the roots swapped by the diagram automorphism (arches). The optional argument labels allows custom labels for the simple roots.”;

```
HelminckDiagram[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  labels_: {}] := Module[
  {g1, g2},

  g1 = DynkinDiagram[r, labels];
  g2 = Graphics[ThetaComponents[r, List[argdisks], arches]];

  Quiet[Show[g1, g2]]
];

HelminckDiagram[r_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  labels_: {}] := Module[
  {g1, g2},

  g1 = DynkinDiagram[r, labels];
  g2 = Graphics[ThetaComponents[r, List[argdisks], arches]];

  Quiet[Show[g1, g2]]
];

HelminckDiagram[r_?RootInputQ, theta_?MatrixQ, labels_: {}] :=
Module[
  {g1, g2, disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  g1 = DynkinDiagram[r, labels];
  g2 = Graphics[ThetaComponents[r, disks, arches]];

  Quiet[Show[g1, g2]]
];

HelminckDiagram[r_?RootBasisQ, theta_?MatrixQ, labels_: {}] :=
Module[
  {g1, g2, disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  g1 = DynkinDiagram[r, labels];
  g2 = Graphics[ThetaComponents[r, disks, arches]];

  Quiet[Show[g1, g2]]
];
```

```
];  
HelminckDiagram[a___] := InvalidArg["HelminckDiagram", a];
```

12.9.3 HelminckDiagramTeX

HelminckDiagramTeX::usage=

”HelminckDiagramTeX[r,theta,labels] gives LaTeX code to draw the Helminck Diagram of an involution theta on the roots acting on root system r. The optional argument labels allows custom labels for the simple roots.

HelminckDiagramTeX[r,disks,arches,labels] gives LaTeX code to draw the Helminck Diagram of an involution on the roots acting on root system r. The involution is described by the fixed simple roots (disks) and the roots swapped by the diagram automorphism (arches). The optional argument labels allows custom labels for the simple roots.”;

```
HelminckDiagramTeX[r_?RootBasisQ, theta_?MatrixQ, custom_: {}] :=
Module[
  {disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  Return[HelminckDiagramTeX[r, disks, arches, theta]];
];

HelminckDiagramTeX[r_?RootInputQ, theta_?MatrixQ, custom_: {}] :=
Module[
  {disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  Return[HelminckDiagramTeX[r, disks, arches, theta]];
];

HelminckDiagramTeX[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, custom_: {}] :=
HelminckDiagramSTeX[BasisToRootSystem[r], List[argdisks], arches,
  custom];

HelminckDiagramTeX[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, custom_: {}] := Module[
  {out, totalHeight, tWidth, tHeight, i, j, xSize, ySize, ptSize,
    nOff, pointMap, points, x1, x2, y1, y2, rin, disks},

  disks = List[argdisks];

  ptSize = 6; (* Size of a point *)
  xSize = 25; (*
  Space between points on X axis *)
  ySize = 24; (*
  Space between points on y axis *)

  tHeight = 0;
  totalHeight = 0;

  out = "";
  points = {};
```

```

rin = RootInput[r];

(* pointMap will determine which points correspond to which \
diagram *)
pointMap = {};

For[i = 1, i <= Length[rin], i++,
  pointMap = Join[pointMap, Table[i, {j, 1, rin[[i, 2]]}]];
];

If[Length[rin] <= 1,
  out = HelminckDiagramTeX[rin, disks, 0, 0, 0, custom];
  tWidth = xSize*DynkinWidth[rin];
  tHeight = ySize*DynkinHeight[rin];
  totalHeight = tHeight;
  points = DynkinPointsTeX[rin[[1, 1]], rin[[1, 2]], 0, 0];
  ,
  tWidth = 0;
  tHeight = 0;
  nOff = 0;

  For[i = 1, i <= Length[rin], i++,
    tWidth = Max[tWidth, xSize*DynkinWidth[rin[[i]]]];
    totalHeight =
      totalHeight + Round[(ySize*3/2)]*DynkinHeight[rin[[i]]];
  ];

  tHeight = totalHeight;

  For[i = 1, i <= Length[rin], i++,
    tHeight = tHeight - Round[(ySize*3/2)]*DynkinHeight[rin[[i]]];
    out =
      out <> HelminckDiagramTeX[rin[[i]], disks,
        Round[(tWidth - 25*DynkinWidth[rin[[i]]])/2], tHeight, nOff,
        custom];
    points =
      Join[points,
        DynkinPointsTeX[rin[[i, 1]], rin[[i, 2]],
          Round[(tWidth - 25*DynkinWidth[rin[[i]]])/2], tHeight]];
    nOff = nOff + rin[[i, 2]];
  ];
];

For[i = 1, i <= Length[arches], i++,
  x1 = points[[arches[[i, 1]], 1]];
  y1 = points[[arches[[i, 1]], 2]];
  x2 = points[[arches[[i, 2]], 1]];
  y2 = points[[arches[[i, 2]], 2]];

  Which[
    pointMap[[arches[[i, 1]]]] != pointMap[[arches[[i, 2]]]],
    (* If both points correspond to a different diagram,
    draw a simple connecting line *)

    out = out <> "\\put(" <> ToString[x1] <> "," <>
      ToString[y1 + (ptSize/2)] <> ")\\line(0,-1){" <>
      ToString[Abs[y2 - y1] - ptSize] <> "}}\\n";
    ,
    y1 == y2,
    (* If both points have same x, draw a horizontal arch *)

    out = out <> "\\put(" <> ToString[Round[(x1 + x2)/2]] <> "," <>

```

```

ToString[y1 - 5] <> "){\oval(" <>
ToString[Round[Abs[x2 - x1]]] <> ", " <> ToString[ySize] <>
") [b]}\n";
out =
out <> "\\put(" <> ToString[x1] <> ", " <> ToString[y1 - 8] <>
"){\vector(0,1){3}}\n";
out =
out <> "\\put(" <> ToString[x2] <> ", " <> ToString[y1 - 8] <>
"){\vector(0,1){3}}\n";
,
x1 == x2,
(* If both points have same y, draw a vertical arch *)
(*)

out=out<>"\\put(" <> ToString[x1+6] <> ", " <> ToString[Round[(y1+
y2)/2]] <> "){\oval(" <> ToString[xSize] <> ", " <> ToString[Round[Abs[
y2-y1]]] <> ") [r]}\n";
*)

out = out <> "\\put(" <> ToString[x1 + 5] <> ", " <>
ToString[Round[(y1 + y2)/2]] <>
"){\bezier{150}(0,17)(11,0)(0,-17)}\n";
out =
out <> "\\put(" <> ToString[x1 + 5.5] <> ", " <>
ToString[Round[(y1 + y2)/2] - 15.9] <>
"){\vector(-2,-3){1}}\n";
out =
out <> "\\put(" <> ToString[x1 + 5.5] <> ", " <>
ToString[Round[(y1 + y2)/2] + 15.9] <>
"){\vector(-2,3){1}}\n";
];
];

(*out="\\begin{picture}(" <> ToString[tWidth+20] <> ", " <> ToString[
totalHeight-ySize] <> ")(-10,-10)\n" <> out;*)

out = "\\begin{picture}(" <> ToString[tWidth + 20] <> ", " <>
ToString[totalHeight] <> ")(-10,-10)\n" <> out;

out = out <> "\\end{picture}\n";

out
];

HelminckDiagramTeX[a_++] := InvalidArg["HelminckDiagramTeX", a];

```

12.10 Group Action Package (Internal)

12.10.1 HelminckDiagramSTeX

HelminckDiagramSTeX::usage=

"HelminckDiagramSTeX[r,disks,xOff,yOff,nOff,labels] gives LaTeX code for the Helminck diagram for an irreducible root system r with fixed roots listed in disks. xOff and yOff are, respectively, the x and y coordinate offsets of the diagram. nOff provides the offset for

automatic root numbering (starting value). The optional argument labels allows custom labels for the simple roots.”;

```

HelminckDiagramSTeX[r_?RootBasisQ, {argdisks___?IntegerQ},
  xOff_?IntegerQ, yOff_?IntegerQ, nOff_?IntegerQ, custom_: {}] :=
  HelminckDiagramSTeX[BasisToRootSystem[r], List[argdisks], xOff,
    yOff, nOff, custom];

HelminckDiagramSTeX[r_?RootInputQ, {argdisks___?IntegerQ},
  xOff_?IntegerQ, yOff_?IntegerQ, nOff_?IntegerQ, custom_: {}] :=
  Module[
    {type, dim, points, out, labels, edgeCodes, edgeCons, x1, x2, y1,
      y2, i, ptSize, xSize, ySize, ceList, rin, disks},

    disks = List[argdisks];
    rin = IrreducibleRootInput[r];

    ptSize = 6; (* Size of a point *)
    xSize = 25; (*
      Space between points on X axis *)
    ySize = 24; (*
      Space between points on y axis *)

    out = "";

    type = rin[[1]];
    dim = rin[[2]];

    (* All Points, labels, edges *)

    points = DynkinPointsTeX[type, dim, xOff, yOff];
    labels =
      Table["\\alpha_{\" <> ToString[i] <> \"}, {i, 1 + nOff,
        dim + nOff}];
    edgeCodes = DynkinEdgeCodes[type, dim];
    edgeCons = DynkinEdgeCons[type, dim];
    ceList = {};

    (* Customizations *)
    If[! SameQ[custom, {}],
      If[! SameQ[custom[[1]], {}],
        labels = Take[custom[[1]], {1 + nOff, dim + nOff}];];
    If[Length[custom] >= 2 && ! SameQ[custom[[2]], {}],
      ceList = custom[[2]];];
    ];

    (* Construct points *)

    For[i = 1, i <= dim, i++,
      If[MemberQ[disks, i + nOff],
        (* I'm a Bold, Filled in Circle *)

        out = out <> "\\put(\" <> ToString[points[[i, 1]]] <> \",\" <>
          ToString[points[[i, 2]]] <> \"){\circle*{\" <>
            ToString[ptSize] <> \"}}\n";

        (* I'm hollow inside. I have roots, but they're not fixed. *)

        out =
          out <> "\\put(\" <> ToString[points[[i, 1]]] <> \",\" <>
            ToString[points[[i, 2]]] <> \"){\circle{\" <>

```



```

        ToString[ptSize] <> "}}\\n";
    ];
];

(* Construct labels *)

For[i = 1, i <= dim, i++,
  If[StringLength[ToString[labels[[i]]]] < 1, Continue[]];
  out =
    out <> "\\put(" <> ToString[points[[i, 1]]] <> "," <>
      ToString[points[[i, 2]] + ptSize] <>
      "\\makebox(0,0)[b]{\\scriptsize $" <> ToString[labels[[i]]] <>
      "$}\\n";
];

(* Construct edges *)

For[i = 1, i <= Length[edgeCons], i++,
  x1 = points[[edgeCons[[i, 1]], 1]];
  y1 = points[[edgeCons[[i, 1]], 2]];
  x2 = points[[edgeCons[[i, 2]], 1]];
  y2 = points[[edgeCons[[i, 2]], 2]];

  Switch[edgeCodes[[i]],
    10,
    Which[
      y1 == y2,
      (* Horizontal Line line(1,0) - 1 unit in x,
        0 in y diredtions {line size} *)

      If[MemberQ[ceList, i + nOff],

        out = out <> "\\put(" <> ToString[Round[(x1 + x2)/2]] <>
          "," <> ToString[y1] <>
          "\\makebox(0,0)[b]{\\ldots $}\\n";
        ,

        out = out <> "\\put(" <> ToString[x1 + (ptSize/2)] <> "," <>
          ToString[y1] <> "\\line(1,0){" <>
          ToString[xSize - ptSize] <> "}}\\n";
      ];
    ,
    x1 == x2,
    (* Vertical Line *)

    out = out <> "\\put(" <> ToString[x1] <> "," <>
      ToString[y1 + (ptSize/2)] <> "\\line(0,1){" <>
      ToString[ySize - ptSize] <> "}}\\n";
    ,
    y1 < y2,
    (* Diagonal Up *)

    out = out <> "\\put(" <> ToString[x1 + (ptSize/2)] <> "," <>
      ToString[y1] <> "\\line(2,1){" <>
      ToString[xSize - ptSize] <> "}}\\n";
    ,
    y1 > y2,
    (* Diagonal Down *)

    out = out <> "\\put(" <> ToString[x1 + (ptSize/2)] <> "," <>
      ToString[y1] <> "\\line(2,-1){" <>
      ToString[xSize - ptSize] <> "}}\\n";
  ];
];

```

```

];
,
21,
out =
out <> "\\put(" <> ToString[x1 + (ptSize*3/2)] <> "," <>
ToString[y1 - 1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";
out =
out <> "\\put(" <> ToString[x1 + (ptSize*3/2)] <> "," <>
ToString[y1 + 1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";

out =
out <> "\\put(" <> ToString[x1 + (ptSize)] <> "," <>
ToString[y1] <> ")\\vector(-1,0){" <> ToString[1] <> "}}\\n";
,
22,
out =
out <> "\\put(" <> ToString[x1 + (ptSize/2)] <> "," <>
ToString[y1 - 1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";
out =
out <> "\\put(" <> ToString[x1 + (ptSize/2)] <> "," <>
ToString[y1 + 1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";

out =
out <> "\\put(" <> ToString[x2 - (ptSize)] <> "," <>
ToString[y1] <> ")\\vector(1,0){" <> ToString[1] <> "}}\\n";
,
31,
out =
out <> "\\put(" <> ToString[x1 + (ptSize*3/2)] <> "," <>
ToString[y1 - 1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";
out =
out <> "\\put(" <> ToString[x1 + (ptSize*3/2)] <> "," <>
ToString[y1 + 1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";
out =
out <> "\\put(" <> ToString[x1 + (ptSize*3/2)] <> "," <>
ToString[y1] <> ")\\line(1,0){" <>
ToString[xSize - 2 ptSize] <> "}}\\n";

out =
out <> "\\put(" <> ToString[x1 + (ptSize)] <> "," <>
ToString[y1] <> ")\\vector(-1,0){" <> ToString[1] <> "}}\\n";
];
];

out
];

HelminckDiagramSTeX[a_++] := InvalidArg["HelminckDiagramSTeX", a];

```

12.10.2 ThetaComponents

ThetaComponents::usage=

”ThetaComponents[r,theta] returns the components for a Helminck diagram to be merged with the Dynkin diagram. r denotes the root system, and theta the involution over r.

ThetaComponents[r,disks,arches] returns the components for a Helminck diagram to be merged with the Dynkin diagram. r denotes the root system, disks denotes the fixed roots, and arches denotes the diagram automorphism.”;

```

ThetaComponents[r_?RootInputQ, theta_?MatrixQ] := Module[
  {disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  Return[ThetaComponents[r, disks, arches]];
];

ThetaComponents[r_?RootBasisQ, theta_?MatrixQ] := Module[
  {disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  Return[ThetaComponents[r, disks, arches]];
];

ThetaComponents[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  ThetaComponents[BasisToRootSystem[r], List[argdisks], arches];

ThetaComponents[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] := Module[
  {rin, pointMap, points, dotRadius, i, j, x1, x2, y1, y2, disks,
   cth, xc, radius, yc,
   grDisks, grArches, grArgs},

  dotRadius = .065;

  disks = List[argdisks];
  rin = RootInput[r];

  (* pointMap will determine which points correspond to which \
  diagram *)
  pointMap = {};

  For[i = 1, i <= Length[rin], i++,
    pointMap = Join[pointMap, Table[i, {j, 1, rin[[i, 2]]}]];
  ];

  points = DynkinPoints[rin];

  (* Draw the solid black dots *)

  grDisks =
    Table[Disk[points[[disks[[i]]]], dotRadius], {i, 1,
      Length[disks]};

```

```

(* Arches *)
grArches = {};

For[i = 1, i <= Length[arches], i++,
  x1 = points[[arches[[i, 1]], 1]];
  y1 = points[[arches[[i, 1]], 2]];
  x2 = points[[arches[[i, 2]], 1]];
  y2 = points[[arches[[i, 2]], 2]];

  Which[
    pointMap[[arches[[i, 1]]]] != pointMap[[arches[[i, 2]]]],
    (* If both points correspond to a different diagram,
      draw a simple connecting line *)

    grArches = Join[grArches, {Line[{{x1, y1}, {x2, y2}}]}];
    ,
    y1 == y2,
    (* If both points have same x, draw a horizontal arch *)

    cth = Pi/4;
    xc = (x2 + x1)/2;
    radius = (x2 - x1)*Cos[cth];
    yc = y1 + radius*Sin[cth];
    grArches =
      Join[grArches, {Circle[{xc, yc},
        radius, {3 Pi/2 - cth, 3 Pi/2 + cth}]}];
    ,
    x1 == x2,
    (* If both points have same y, draw a vertical arch *)

    cth = Pi/4;
    yc = (y2 + y1)/2;
    radius = Abs[(y2 - y1)]*Sin[cth];
    xc = x1 - radius*Cos[cth];
    grArches =
      Join[grArches, {Circle[{xc, yc}, radius, {-cth, cth}]}];
    ,
    True,
    (* Diagonal connection? *)

    grArches = Join[grArches, {Line[{{x1, y1}, {x2, y2}}]}];
  ];
];

grArgs = Join[grDisks, grArches];

grArgs
];

ThetaComponents[a___] := InvalidArg["ThetaComponents", a];

```

12.11 Local Symmetric Spaces Package (Primary)

12.11.1 AlignPolarities

AlignPolarities::usage=

”AlignPolarities[d1,theta1,cvals1,d2,theta2,cvals2,w] aligns the polarities of two involutions over the Lie algebra cvals1 and cvals2. Each is defined with respect to root system involutions theta1 and theta2 over root systems d1 and d2. If the optional argument w is omitted, or is set to 1, the involution cvals1 is returned, modified so that its polarity aligns with cvals2. If w is 2, then cvals2 is returned, modified to align with cvals1. Each instance of theta (1 or 2) can be replaced with two arguments: disks, an integer index of the fixed roots, and arches, the diagram automorphism.”;

```
AlignPolarities[d1_?RootBasisQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ,
  d2_?RootBasisQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, RootInvolution[d1, List[argdisks1], arches1],
  cvals1, d2, RootInvolution[d2, List[argdisks2], arches2], cvals2,
  w];

AlignPolarities[d1_?RootInputQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ,
  d2_?RootBasisQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1],
  RootInvolution[RootBase[d1], List[argdisks1], arches1], cvals1, d2,
  RootInvolution[d2, List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootBasisQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ,
  d2_?RootInputQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, RootInvolution[d1, List[argdisks1], arches1],
  cvals1, RootBase[d2],
  RootInvolution[RootBase[d2], List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootInputQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ,
  d2_?RootInputQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1],
  RootInvolution[RootBase[d1], List[argdisks1], arches1], cvals1,
  RootBase[d2],
  RootInvolution[RootBase[d2], List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootBasisQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ, d2_?RootBasisQ, theta2_?MatrixQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, RootInvolution[d1, List[argdisks1], arches1],
  cvals1, d2, theta2, cvals2, w];
```

```

AlignPolarities[d1_?RootInputQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ, d2_?RootBasisQ, theta2_?MatrixQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1],
  RootInvolution[RootBase[d1], List[argdisks1], arches1], cvals1, d2,
  theta2, cvals2, w];

AlignPolarities[d1_?RootBasisQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ, d2_?RootInputQ, theta2_?MatrixQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, RootInvolution[d1, List[argdisks1], arches1],
  cvals1, RootBase[d2], theta2, cvals2, w];

AlignPolarities[d1_?RootInputQ, {argdisks1___?IntegerQ},
  arches1_?ListQ, cvals1_?ListQ, d2_?RootInputQ, theta2_?MatrixQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1],
  RootInvolution[RootBase[d1], List[argdisks1], arches1], cvals1,
  RootBase[d2], theta2, cvals2, w];

AlignPolarities[d1_?RootBasisQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootBasisQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, theta1, cvals1, d2,
  RootInvolution[d2, List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootInputQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootBasisQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1], theta1, cvals1, d2,
  RootInvolution[d2, List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootBasisQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootInputQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, theta1, cvals1, RootBase[d2],
  RootInvolution[RootBase[d2], List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootInputQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootInputQ, {argdisks2___?IntegerQ}, arches2_?ListQ,
  cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1], theta1, cvals1, RootBase[d2],
  RootInvolution[RootBase[d2], List[argdisks2], arches2], cvals2, w];

AlignPolarities[d1_?RootInputQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootBasisQ, theta2_?MatrixQ, cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1], theta1, cvals1, d2, theta2, cvals2,
  w];

AlignPolarities[d1_?RootBasisQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootInputQ, theta2_?MatrixQ, cvals2_?ListQ, w_: 1] :=
AlignPolarities[d1, theta1, cvals1, RootBase[d2], theta2, cvals2,
  w];

AlignPolarities[d1_?RootInputQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootInputQ, theta2_?MatrixQ, cvals2_?ListQ, w_: 1] :=
AlignPolarities[RootBase[d1], theta1, cvals1, RootBase[d2], theta2,
  cvals2, w];

```

```

AlignPolarities[d1_?RootBasisQ, theta1_?MatrixQ, cvals1_?ListQ,
  d2_?RootBasisQ, theta2_?MatrixQ, cvals2_?ListQ, w_: 1] := Module[
  {p1, p2},

  p1 = InvolutionPolarity[d1, theta1, cvals1];
  p2 = InvolutionPolarity[d2, theta2, cvals2];

  (* If either end is zero then the two involutions do not meet (by \
  construction of rank one decomposition) *)

  If[SameQ[p1[[2]], 0] || SameQ[p2[[1]], 0],
    If[SameQ[w, 1],
      Return[cvals1];
    ,
      Return[cvals2];
    ];
  ];

  (* If the joining ends have the same polarity, no worries *)

  If[SameQ[p1[[2]], p2[[1]]],
    If[SameQ[w, 1],
      Return[cvals1];
    ,
      Return[cvals2];
    ];
  ];

  (* Change the polarity of cvals2 so it aligns with cvals1 *)

  If[SameQ[w, 1],
    Return[SwitchPolarity[d1, theta1, cvals1]];
  ,
    Return[SwitchPolarity[d2, theta2, cvals2]];
  ];
];

AlignPolarities[a___] := InvalidArg["AlignPolarities", a];

```

12.11.2 ApplyRootInvolution

ApplyRootInvolution::usage=

"ApplyRootInvolution[r,disks,arches,root] applies root (denoting a single root or set of roots) to an involution defined over a root system r, with fixed roots disks and diagram automorphism arches.

ApplyRootInvolution[r,theta,root] applies root (denoting a single root or set of roots) to an involution theta defined over a root system r."

```

ApplyRootInvolution[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?ListQ] :=
  ApplyRootInvolutionBasis[r, List[argdisks], arches, root];

ApplyRootInvolution[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?ListQ] := Module[
  {theta, i, disks},

  disks = List[argdisks];
  theta = RootInvolution[r, disks, arches];

  If[SameQ[Depth[root], 2],
    Return[ApplyRootInvolution[r, theta, root]];
  ,
  Return[
    DeleteDuplicates[
      Table[ApplyRootInvolution[r, theta, root[[i]]], {i, 1,
        Length[root]}]]];
  ];
];

ApplyRootInvolution[r_?RootBasisQ, theta_?MatrixQ, root_?ListQ] :=
  ApplyRootInvolutionBasis[r, theta, root];

ApplyRootInvolution[r_?RootInputQ, theta_?MatrixQ, root_?ListQ] :=
  Module[
  {basis, kroot, i},

  If[SameQ[Depth[root], 2],
    basis = RootBase[r];
    kroot = BasisCoefficients[basis, root];

    Return[kroot.Transpose[theta].basis];
  ,
  Return[
    DeleteDuplicates[
      Table[ApplyRootInvolution[r, theta, root[[i]]], {i, 1,
        Length[root]}]]];
  ];
];

ApplyRootInvolution[a___] := InvalidArg["ApplyRootInvolution", a];

```


12.11.3 ApplyRootInvolutionBasis

ApplyRootInvolutionBasis::usage=

”ApplyRootInvolutionBasis[r,disks,arches,root] applies root (denoting a single root or set of roots) to an involution defined over a root system with basis d, with fixed roots disks and diagram automorphism arches.

ApplyRootInvolutionBasis[d,theta,root] applies root (denoting a single root or set of roots) to an involution theta defined over a root system with basis d.”;

```
ApplyRootInvolutionBasis[d_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?ListQ] := Module[
  {kroot, i, disks, theta},

  disks = List[argdisks];
  theta = RootInvolution[d, disks, arches];

  If[SameQ[Depth[root], 2],
    kroot = BasisCoefficients[d, root];

    Return[kroot.Transpose[theta].d];
  ,
  Return[
    DeleteDuplicates[
      Table[ApplyRootInvolutionBasis[d, theta, root[[i]]], {i, 1,
        Length[root]}]]];
  ];

ApplyRootInvolutionBasis[d_?RootBasisQ, theta_?MatrixQ, root_?ListQ] :=
Module[
  {kroot, i},

  If[SameQ[Depth[root], 2],
    kroot = BasisCoefficients[d, root];

    Return[kroot.Transpose[theta].d];
  ,
  Return[
    DeleteDuplicates[
      Table[ApplyRootInvolutionBasis[d, theta, root[[i]]], {i, 1,
        Length[root]}]]];
  ];

ApplyRootInvolutionBasis[a___] :=
  InvalidArg["ApplyRootInvolutionBasis", a];
```

12.11.4 ComplementRoot

```

ComplementRoot::usage=
"ComplementRoot[root] complements the given root.";

ComplementRoot[root_?VectorQ] := Module[{},
  Print["The root ", root, " is looking mighty fine today."];

  Return[root];
];

ComplementRoot[a___] := InvalidArg["ComplementRoot", a];

```

12.11.5 CorrectionVector

CorrectionVector::usage=

”CorrectionVector[r,theta,cvals,x] computes all involution correction vectors for an involution theta over the root system r with structure constants cvals. x supplies the name of the variable set for the toral vector coordinates.”;

```

CorrectionVector[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cvals_?ListQ, x_] :=
  CorrectionVector[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], cvals, x];

CorrectionVector[d_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cvals_?ListQ, x_] :=
  CorrectionVector[d, RootInvolution[d, List[argdisks], arches],
    cvals, x];

CorrectionVector[r_?RootInputQ, theta_?MatrixQ, cvals_?ListQ, x_] :=
  CorrectionVector[RootBase[r], theta, cvals, x];

CorrectionVector[d_?RootBasisQ, theta_?MatrixQ, cvals_?ListQ, x_] :=
  Module[
    {cntr, polynomials, vars, eqns, i, solnset, n},

    solnset = {};
    cntr = TimeUsed[];
    n = Length[theta];

    vars = Table[x[i], {i, 1, n}];
    polynomials = Table[(RootFunctional[d, d[[i]], vars])*
      (RootFunctional[d,
        BasisCoefficients[d, d[[i]]].Transpose[theta].d, vars])*
      StructureConstantsLookup[cvals, d[[i]]]*
      StructureConstantsLookup[cvals,
        BasisCoefficients[d, d[[i]]].Transpose[theta].d] - 1, {i, 1,
        n}];
    eqns = Table[polynomials[[i]] == 0, {i, 1, Length[polynomials]}];

    solnset = GroebnerBackSolver[eqns, vars];

    Return[solnset];
  ];

CorrectionVector[a___] := InvalidArg["CorrectionVector", a];

```

12.11.6 DiagramInvolution

DiagramInvolution::usage=

”DiagramInvolution[r] gives a list of root pairs “arches” in root system r which are swapped by the diagram automorphism of order 2.”;

```
DiagramInvolution[r_?RootInputQ] := Module[
  {rin, i, n, tdim, out, pout},

  rin = RootInput[r];
  n = Length[rin];
  out = {};
  tdim = 0;

  For[i = 1, i <= n, i++,
    pout = DiagramInvolutionSimple[rin[[i]], tdim];

    If[! SameQ[pout, {}],
      out = Join[out, pout];
    ];

    tdim += rin[[i, 2]];
  ];

  Return[out];
];

DiagramInvolution[a___] := InvalidArg["DiagramInvolution", a];
```

12.11.7 InvBasisTable

InvBasisTable::usage=

"InvBasisTable[basis,theta,rrdl] gives a table of the structure constants necessary to determine if the Lie algebra homomorphism rrdl lifted from root system involution theta over the given root basis is an involution.";

```
InvBasisTable[basis_?RootBasisQ, theta_?MatrixQ, rrdl_?ListQ] :=
TableForm[
FullSimplify[
Table[{basis[[i]]}, {ApplyRootInvolutionBasis[basis, theta,
basis[[i]]}],
StructureConstantsLookup[rrdl, basis[[i]]]*
StructureConstantsLookup[rrdl,
ApplyRootInvolutionBasis[basis, theta, basis[[i]]]}, {i, 1,
Length[basis]}]],
TableHeadings -> {None, {"[Alpha]", "[Theta]([Alpha])",
"!(*SubscriptBox["c",
RowBox[{"[Alpha]", ",", "[Theta]"}]) !(*SubscriptBox["c",
",
RowBox[{
RowBox[{"[Theta]",
RowBox[{"(", "[Alpha]", ")"}]}, ",", "[Theta]"}])"}]}]
InvBasisTable[a___] := InvalidArg["InvBasisTable", a];
```

12.11.8 InvBasisTableAlphaForm

InvBasisTableAlphaForm::usage=

”InvBasisTable[basis,theta,rrdl] gives a table of the structure constants necessary to determine if the Lie algebra homomorphism rrdl lifted from root system involution theta over the given root basis is an involution. Roots are printed in alpha form.”;

```
InvBasisTableAlphaForm[basis_?RootBasisQ, theta_?MatrixQ,
  rrdl_?ListQ] :=
TableForm[
  FullSimplify[
    Table[{RootAlphaForm[basis, basis[[i]]], {RootAlphaForm[basis,
      ApplyRootInvolutionBasis[basis, theta, basis[[i]]]},
      StructureConstantsLookup[rrdl, basis[[i]]]*
      StructureConstantsLookup[rrdl,
        ApplyRootInvolutionBasis[basis, theta, basis[[i]]]}, {i, 1,
        Length[basis]}}],
    TableHeadings -> {None, {"[Alpha]", "[Theta]([Alpha])",
      "!(SubscriptBox["c",
RowBox[{"[Alpha]", "", "[Theta]"}]) !(SubscriptBox["c",
",
RowBox[{
RowBox[{"[Theta]",
RowBox[{"[Theta]",
RowBox[{"(", "[Alpha]", ")"}]}], "", "[Theta]"}]}]}]}

InvBasisTableAlphaForm[a___] :=
  InvalidArg["InvBasisTableAlphaForm", a];
```

12.11.9 InvBTRaw

```

InvBTRaw::usage=
"InvBTRaw::usage";

InvBTRaw[basis_?RootBasisQ, theta_?MatrixQ, rrdl_?ListQ] :=
Length[Position[
FullSimplify[
Table[{StructureConstantsLookup[rrdl, basis[[i]]]*
StructureConstantsLookup[rrdl,
ApplyRootInvolutionBasis[basis, theta, basis[[i]]]}], {i, 1,
Length[basis]}}], {-1}]];

InvBTRaw[a___] := InvalidArg["InvBTRaw", a];

```

12.11.10 InvolutionPolarity

InvolutionPolarity::usage=

”InvolutionPolarity[r,theta,cconsts] returns the polarity of an involution over the Lie algebra with structure constants cconsts. The involution is defined with respect to an involution theta over the root system r.

InvolutionPolarity[r,disks,arches,cconsts] returns the polarity of an involution over the Lie algebra with structure constants cconsts. The involution is defined with respect to an involution theta (given as fixed roots disks and diagram automorphism arches) over the root system r.”;

```

InvolutionPolarity[r_?RootInputQ, theta_?MatrixQ, cv_?ListQ] :=
  InvolutionPolarity[RootBase[r], theta, cv];

InvolutionPolarity[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, cv_?ListQ] :=
  InvolutionPolarity[r, RootInvolution[r, List[argdisks], arches],
  cv];

InvolutionPolarity[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, cv_?ListQ] :=
  InvolutionPolarity[RootBase[r],
  RootInvolution[RootBase[r], List[argdisks], arches], cv];

InvolutionPolarity[d_?RootBasisQ, theta_?MatrixQ, cv_?ListQ] := {
  If[! FixedRootQ[d, theta, d[[1]]], 0,
  StructureConstantsLookup[cv, d[[1]]]],
  If[! FixedRootQ[d, theta, d[[Length[d]]]], 0,
  StructureConstantsLookup[cv, d[[Length[d]]]]]
};

InvolutionPolarity[a___] := InvalidArg["InvolutionPolarity", a];

```


12.11.11 LocalBasis

LocalBasis::usage=

”LocalBasis[r,theta] computes the set of basis roots which project down to some root on the local symmetric space. r denotes the root system on which an involution theta is defined.

(S-LOSS) LocalBasis[r,disks,arches] computes the set of basis roots which project down to some root on the local symmetric space. r denotes the root system on which an involution with fixed roots disks and diagram automorphism arches is defined.”;

```
(* Find the white dots on the Helminck Diagram *)

LocalBasis[r_?RootInputQ, theta_?MatrixQ] :=
  Complement[RootBase[r], FixedBasis[r, theta]];

LocalBasis[r_?RootBasisQ, theta_?MatrixQ] :=
  Complement[RootBase[r], FixedBasis[r, theta]];

(* Find the white dots on the Helminck Diagram *)

LocalBasis[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  Complement[RootBase[r], FixedBasis[r, List[argdisks], arches]];

LocalBasis[r_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  Complement[RootBase[r], FixedBasis[r, List[argdisks], arches]];

LocalBasis[a___] := InvalidArg["LocalBasis", a];
```

12.11.12 LocalProject

LocalProject::usage=

”LocalProject[r,disks,arches,root] projects root into some root in the local symmetric space whose root system is determined over an involution over root system r with fixed roots disks and diagram automorphism arches.

LocalProject[r,theta,root] projects root into some root in the local symmetric space whose root system is determined by an involution theta over root system r.”;

```
LocalProject[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ] :=
  EigenspaceProject[r, theta, root, -1];

LocalProject[r_?RootBasisQ, theta_?MatrixQ, root_?VectorQ] :=
  EigenspaceProject[r, theta, root, -1];

LocalProject[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  root_?VectorQ] :=
  EigenspaceProject[r, List[argdisks], arches, root, -1];

LocalProject[r_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  root_?VectorQ] :=
  EigenspaceProject[r, List[argdisks], arches, root, -1];

LocalProject[a___] := InvalidArg["LocalProject", a];
```

12.11.13 OneCorrectionVector

OneCorrectionVector::usage=

"OneCorrectionVector[r,theta,cvals,x,sn] computes one involution correction vector for an involution theta over the root system r with structure constants cvals. x supplies the name of the variable set for the toral vector coordinates. The optional argument sn allows the user to specify solution, number sn, in the case multiple solutions are present."

```
OneCorrectionVector[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, cvals_?ListQ, x_, sn_: 1] :=
  OneCorrectionVector[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], cvals, x, sn];

OneCorrectionVector[d_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, cvals_?ListQ, x_, sn_: 1] :=
  OneCorrectionVector[d, RootInvolution[d, List[argdisks], arches],
    cvals, x, sn];

OneCorrectionVector[r_?RootInputQ, theta_?MatrixQ, cvals_?ListQ, x_,
  sn_: 1] := OneCorrectionVector[RootBase[r], theta, cvals, x, sn];

OneCorrectionVector[d_?RootBasisQ, theta_?MatrixQ, cvals_?ListQ, x_,
  sn_: 1] := Module[
  {cntr, polynomials, vars, i, solnset, eqns, n, roots, troots},

  cntr = TimeUsed[];
  n = Length[d];

  vars = Table[x[i], {i, 1, n}];
  roots = Table[BasisCoefficients[d, d[[i]]].d, {i, 1, n}];

  (* In case theta corresponds to a larger basis containing the one \
  passed *)

  troots =
    Table[ApplyRootInvolutionBasis[d, theta, roots[[i]]], {i, 1, n}];

  polynomials = Table[
    (RootFunctional[d, roots[[i]], vars])*
    (RootFunctional[d, troots[[i]], vars])*
    StructureConstantsLookup[cvals, roots[[i]]]*
    StructureConstantsLookup[cvals, troots[[i]] - 1, {i, 1, n}];
  eqns = Table[polynomials[[i]] == 0, {i, 1, Length[polynomials]}];

  (*solnset=groebnerOneSolution[eqns,vars];*)

  solnset = Solve[eqns, vars][[sn]];

  Return[solnset];
];

OneCorrectionVector[a___] := InvalidArg["OneCorrectionVector", a];

(* wait... thetadelta is not an involution...
\
oneCorrectionVector[basis_,theta_,systemMatrix_,cvals_,x_] := Module[
{cntr,polynomials,vars,tvars1,tvars2,i,j,solnset,eqns,n,roots,troots}\
```

```

,

cntr=TimeUsed[];
n=Length[basis];

vars=Table[x[i],{i,1,n}];
roots=Table[basisCoeffs[basis,basis[[i]]].basis,{i,1,n}];
roots=Table[applyInvolutionBasis[basis,theta,roots[[i]]],{i,1,n}];(* \
In case theta corresponds to a larger basis containing the one passed \
*)

tvars1={};
tvars2={};
For[i=1,i<=n,i++,
If[isFixedRootBasis[basis,theta,roots[[i]]],
tvars1=Join[{x[i]},tvars1];
tvars2=Join[{0},tvars2];
,
tvars1=Join[{0},tvars1];
tvars2=Join[{x[i]},tvars2];
];
];

polynomials=Table[
(rootFunctional[basis,roots[[i]],vars])*
(rootFunctional[basis,roots[[i]],tvars1] - \
rootFunctional[basis,roots[[i]],tvars2])*
structureConstantsLookup[cvals,roots[[i]]]*
structureConstantsLookup[cvals,roots[[i]]-1,{i,1,n}];
eqns=Table[polynomials[[i]]==0,{i,1,Length[polynomials]}];

(*solnset=groebnerOneSolution[eqns,vars];*)
\
solnset=Solve[eqns,vars][[1]];

Print["(liftThetaOneSolution) Used ",TimeUsed[]-cntr,"s CPU Time to \
look for one solution."];

solnset
];
*)

```

12.11.14 OrthoComplement

OrthoComplement::usage=

"OrthoComplement[roots,com] finds all roots in the set roots which are orthogonal to the root or set of roots given in com.";

```
(* Find roots orthogonal to complement *)

OrthoComplement[roots_?ListQ, complement_?ListQ] := Module[
  {i, j, oset, compl},

  oset = {};

  (* given one root or a set of roots? *)

  If[SameQ[Depth[complement], 2],
    compl = {complement};
    ,
    compl = complement;
  ];

  For[i = 1, i <= Length[roots], i++,
    For[j = 1, j <= Length[compl], j++,
      If[InnerProduct[roots[[i]], compl[[j]]] != 0,
        Break[];
      ];
    ];

    If[j > Length[compl],
      oset = Union[oset, {roots[[i]]}];
    ];
  ];

  Return[oset];
];

OrthoComplement[a_...] := InvalidArg["OrthoComplement", a];

(* Find roots *)
(*
\
XXorthoRestrictedRoots[r_,disks_,arches_,rootset_] := Module[
{roots,oset,i,j},

oset={};
roots=restrictedRootSystem[r,disks,arches];

For[i=1,i<=Length[roots],i++,
For[j=1,j<=Length[rootset],j++,

];
];
];
*)
```

12.11.15 RankOneComponentBasis

RankOneComponentBasis::usage=

"RankOneComponentBasis[r,theta,root] if a variant of RankOneLocalBasis. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta over the root system r. Roots which are fixed by theta are removed unless they are part of the same irreducible component of the rank one restricted root system with respect to root. In effect, this procedure generates the class of the rank one restricted root system found in Table I of "Algebraic Groups with a Commuting Pair of Involutions and Semisimple Symmetric Spaces" by A.G. Helminck.

RankOneComponentBasis[r,disks,arches,root] if a variant of RankOneLocalBasis. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta (described by fixed roots disks and diagram automorphism arches) over the root system r. Roots which are fixed by theta are removed unless they are part of the same irreducible component of the rank one restricted root system with respect to root. In effect, this procedure generates the class of the rank one restricted root system found in Table I of "Algebraic Groups with a Commuting Pair of Involutions and Semisimple Symmetric Spaces" by A.G. Helminck.";

```
RankOneComponentBasis[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ] :=
  RankOneComponentBasis[RootBase[r], theta, root];

RankOneComponentBasis[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RankOneComponentBasis[RootBase[r],
    RootInvolution[List[argdisks], arches], root];

RankOneComponentBasis[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RankOneComponentBasis[r, RootInvolution[List[argdisks], arches],
    root];

RankOneComponentBasis[r_?RootBasisQ, theta_?MatrixQ, root_?VectorQ] :=
  Module[
    {i, j, rob, rwp, rset, nset, cursize, newsize, fixed},

    rob = RestrictedRankOneBasis[r, theta, root];
    rwp = RankOneLocalBasis[r, theta, root];
    fixed = FixedBasis[r, theta];
    rset = rwp;

    cursize = Length[rset];
    newsize = 0;
    (* Add back the fixed roots which are connected *)

    While[! SameQ[cursize, newsize],
      cursize = Length[rset];
```

```

nset = {};
For[i = 1, i <= Length[fixed], i++, (* FOR every fixed root *)

  For[j = 1, j <= Length[rset], j++, (* FOR every rset root *)

    If[RootBasisConnectedQ[rob, fixed[[i]], rset[[j]]],
      nset = Union[nset, {fixed[[i]]}];
    ];
  ];

rset = Union[rset, nset];
newsize = Length[rset];
];

(* Sort each element of rset *)
rset = ByBasisSort[r, rset];

Return[rset];
];

RankOneComponentBasis[a_...] := InvalidArg["RankOneComponentBasis", a];

```

12.11.16 RankOneLocalBasis

RankOneLocalBasis::usage=

"RankOneLocalBasis[r,theta,root] if a variant of RankOneBasis. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta over the root system r. Roots which are fixed by theta are removed, leaving present only the roots which project down to some root in the local symmetric space.

RankOneLocalBasis[r,disks,arches,root] if a variant of RankOneBasis. This procedure computes the basis of the restricted rank one root system with respect to root, defined by an involution theta (described by fixed roots disks and diagram automorphism arches) over the root system r. Roots which are fixed by theta are removed, leaving present only the roots which project down to some root in the local symmetric space.";

```
RankOneLocalBasis[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ] :=
  RankOneLocalBasis[RootBase[r], theta, root];

RankOneLocalBasis[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RankOneLocalBasis[RootBase[r],
    RootInvolution[r, List[argdisks], arches], root];

RankOneLocalBasis[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RankOneLocalBasis[r, RootInvolution[r, List[argdisks], arches],
    root];

RankOneLocalBasis[r_?RootBasisQ, theta_?MatrixQ, root_?VectorQ] :=
  Module[
    {i, rr, ret, rob},

    rob = RestrictedRankOneBasis[r, theta, root];

    ret = {};
    For[i = 1, i <= Length[rob], i++,
      rr = LocalProject[r, theta, rob[[i]]];
      If[Norm[rr] > 0,
        ret = Join[{rob[[i]]}, ret];
      ];
    ];

    Return[ret];
  ];

RankOneLocalBasis[a___] := InvalidArg["RankOneLocalBasis", a];
```


12.11.17 RankOneRootLift

RankOneRootLift::usage=

”RankOneRootLift[r,theta,cconsts,nvals] lifts a restricted rank one involution theta, with respect to root system r, to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. nvals supplies the Chevalley constants.

RankOneRootLift[r,disks,arches,cconsts,nvals] lifts a restricted rank one involution theta (described via fixed roots disks and diagram automorphism arches), with respect to root system r, to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. nvals supplies the Chevalley constants.”;

```
RankOneRootLift::rank =
  "supplied involution is not restricted rank one.";

RankOneRootLift[r_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cconsts_?ListQ, nvals_?ListQ] :=
  RankOneRootLift[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], cconsts,
    nvals];

RankOneRootLift[basis_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, cconsts_?ListQ, nvals_?ListQ] :=
  RankOneRootLift[basis,
    RootInvolution[basis, List[argdisks], arches], cconsts, nvals];

RankOneRootLift[r_?RootInputQ, theta_?MatrixQ, cconsts_?ListQ,
  nvals_?ListQ] :=
  RankOneRootLift[RootBase[r], theta, cconsts, nvals];

RankOneRootLift[basis_?RootBasisQ, theta_?MatrixQ, cconsts_?ListQ,
  nvals_?ListQ] := Module[
  {roots, entryn, tableentry, rettbl,
   minbasis, mintheta, mincconsts, mininvals,
   i, n, mininvol, minroots},

  (* Initialize *)
  (*r=BasisToRootSystem[basis];*)

  roots = RootSystemFromBasis[basis];
  n = Length[basis];

  (*Print["rorl1."];*)

  (* Rank zero? This is all fixed roots *)

  If[SameQ[RestrictedRootRank[basis, theta], 0],
    rettbl = {}];

  For[i = 1, i <= Length[basis], i++,
    rettbl = Join[{{basis[[i]], 1}}, rettbl];
    rettbl = Join[{{-basis[[i]], 1}}, rettbl];
  ];

  rettbl = gMergeInvolutions[basis, theta, rettbl, nvals];
  Return[rettbl];
```

```

];

(* Rank one?
This isn't typically a user-called function.
If this message is returned, there's probably some bigger bug. *)

  If[! SameQ[RestrictedRootRank[basis, theta], 1],
    Message[RankOneRootLift::rank];
    Return[{}];
  ];

(* Identify the RRO entry number *)

entryn = RROTableEntry[basis, theta];
tableentry = RROTable[entryn];

(*Print["rorl2."];*)

(* It is 1-consistent *)
If[tableentry[[4]],
  (* Theta-Delta is an involution. Return list of {ROOT, coeff=
  STRUCTURECONST} *)

  rettbl =
    Table[{roots[[i]],
      StructureConstantsLookup[cconsts, roots[[i]]], {i, 1,
        Length[roots]}}];
  Return[rettbl];
];

(* Not 1-
consistent AND is a classical type of length exceeding minimum *)

  If[(SameQ[entryn, 7] || SameQ[entryn, 8] || SameQ[entryn, 11]) &&
    Length[basis] > tableentry[[5]],
    (* Rank one entry has infinite possible number of sizes *)
    (*
    Lift the min size involution *)
    (* simpleLift[minbasis,
    mintheta, minsystemMatrix, cconsts]; *)

    (* STRATEGY:
      1. First lift the minimum size involution.
      Do this with a new basis.
      2. The map the structure constants for our "bigger" involution to \
the lifted smaller one...
      e.g. for type B,
      the first two and last two basis roots map to the first and last \
two consts of our mini lifted invol.
      3. Fill in the other basis roots with 1. (They're fixed, so 1-
consistent)
      4. gMergeInvolutions to fill in the other roots
    *)

    Switch[entryn,
      7, (* B *)
      (*
      trim all but the first and last two roots *)
      (*minr="B4";*)

      minbasis = {};
      For[i = 1, i <= Length[basis], i++,

```

```

    If[
      SameQ[i, 1] || SameQ[i, 2] || SameQ[i, n - 1] || SameQ[i, n],
      minbasis = Join[minbasis, {basis[[i]]}];
    ];
  ];
,
8, (* C *)
(*
trim all but the first root and last two roots *)
(*minr=
"C3";*)
minbasis = {};
For[i = 1, i <= Length[basis], i++,
  If[SameQ[i, 1] || SameQ[i, n - 1] || SameQ[i, n],
    minbasis = Join[minbasis, {basis[[i]]}];
  ];
];
,
11, (* D *)
(*
trim all but the first two and last four roots *)
(*minr=
"D6";*)
minbasis = {};
For[i = 1, i <= Length[basis], i++,
  If[
    SameQ[i, 1] || SameQ[i, 2] || SameQ[i, n - 3] ||
    SameQ[i, n - 2] || SameQ[i, n - 1] || SameQ[i, n],
    minbasis = Join[minbasis, {basis[[i]]}];
  ];
];
]; (* END switch *)

(* trick:
can use restRootAut for any matrix we wish to reduce accordingly *)
\
(*minibasis=RootBase[minr];*)

mintheta = RestrictedRootAut[basis, theta, minbasis];
minroots = RootSystemFromBasis[minbasis];
mininvals = KleinChevalley[minbasis];
minconsts = SteinbergThetaDelta[minbasis, mininvals, mintheta];

(* Lift the mini involution *)

mininvol =
SimpleRootLift[minbasis, mintheta, minconsts, nvals, True];

(* Map the structure constants *)
(*
rettbl is our "main" involution. *)
rettbl = {};

(* minbasis and minibasis should be "aligned" *)
(* WARNING:
relies on the proper ordering of the bases. If fails,
can try a case-by-case similar to previous switch statement. *)

For[i = 1, i <= Length[minbasis], i++,
  rettbl =
  Join[{{minbasis[[i]]},
    StructureConstantsLookup[mininvol, minbasis[[i]]]}],

```

```

    rettbl];
    rettbl =
    Join[{{-minbasis[[i]],
      StructureConstantsLookup[mininvol, -minbasis[[i]]]}},
      rettbl];
  ];

  (* Fill in the holes *)
  For[i = 1, i <= Length[basis], i++,
    If[! MemberQ[minbasis, basis[[i]]],
      rettbl = Join[{{basis[[i]], 1}}, rettbl];
      rettbl = Join[{{-basis[[i]], 1}}, rettbl];
    ];
  ];

  (* Now we have to construct the bigger involution *)

  rettbl = gMergeInvolutions[basis, theta, rettbl, nvals];

  Return[rettbl];
];

(* The remaining types are finite cases *)

Return[SimpleRootLift[basis, theta, cconsts, nvals]];
];

RankOneRootLift[a___] := InvalidArg["RankOneRootLift", a];

```

12.11.18 ReduceRestrictedRank

ReduceRestrictedRank::usage=

"ReduceRestrictedRank[r,disks,arches,root] reduces the restricted rank of an involution defined over root system r with fixed roots disks and diagram automorphism arches by eliminating root.

ReduceRestrictedRank[r,theta,root] reduces the restricted rank of an involution theta defined over root system r by eliminating root.";

```

ReduceRestrictedRank[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  ReduceRestrictedRank[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], root];

ReduceRestrictedRank[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  ReduceRestrictedRank[r, RootInvolution[r, List[argdisks], arches],
    root];

ReduceRestrictedRank[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ] :=
  ReduceRestrictedRank[RootBase[r], theta, root];

ReduceRestrictedRank[r_?RootBasisQ, theta_?MatrixQ, root_?VectorQ] :=
  Module[
    {rbasis, wbasis, fbasis, wset, oset, i, j, wroot},

    oset = {};

    (* The black dots *)
    fbasis = FixedBasis[r, theta];

    (* rank one basis (basis roots that project to integral multiples \
of ROOT) *)
    rbasis = RestrictedRankOneBasis[r, theta, root];

    (* all the white dots (roots that will project to non-zero roots) *)

    wbasis = LocalBasis[r, theta];

    (* These are the white dots which project to integral multiples of \
ROOT *)
    wset = Intersection[rbasis, wbasis];

    (* Identify basis roots orthogonal to projection of wset onto -1 \
eigenspace. We already know the black dots are *)

    For[i = 1, i <= Length[r], i++,
      wroot = r[[i]];

      For[j = 1, j <= Length[wset], j++,
        If[InnerProduct[wroot, LocalProject[r, theta, wset[[j]]]] != 0,
          Break[]];
      ];

    If[j > Length[wset],
      oset = Union[oset, {wroot}];
    ];
  ];
```

```

];

(*
Print["fbasis ",fbasis];
Print["rbasis ",rbasis];
Print["wbasis ",wbasis];
Print["wset   ",wset];
*)

Return[Union[oset, fbasis]];
];

ReduceRestrictedRank[a___] := InvalidArg["ReduceRestrictedRank", a];

```

12.11.19 RestrictedRankOneBasis

RestrictedRankOneBasis::usage=

”RestrictedRankOneBasis[r,disks,arches,root] computes the basis of the restricted rank one root system with respect to root, defined by an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRankOneBasis[r,theta,root] computes the basis of the restricted rank one root system with respect to root, defined by an involution theta over root system r.”;

```
RestrictedRankOneBasis[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RestrictedRankOneBasis[RootBase[r], List[argdisks], arches, root];

RestrictedRankOneBasis[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] := Module[
  {basis, rset, disks, ret},

  basis = r;
  disks = List[argdisks];
  rset = RestrictedRankOneSystem[r, disks, arches, root];

  ret = Intersection[basis, rset];
  ret = ByBasisSort[r, ret];

  Return[ret];
];

RestrictedRankOneBasis[r_?RootInputQ, theta_?MatrixQ, root_?VectorQ] :=
  RestrictedRankOneBasis[RootBase[r], theta, root];

RestrictedRankOneBasis[r_?RootBasisQ, theta_?MatrixQ, root_?VectorQ] :=
  Module[
  {basis, rset, ret},

  basis = r;
  rset = RestrictedRankOneSystem[r, theta, root];

  ret = Intersection[basis, rset];
  ret = ByBasisSort[r, ret];

  Return[ret];
];

RestrictedRankOneBasis[a___] :=
  InvalidArg["RestrictedRankOneBasis", a];
```

12.11.20 RestrictedRankOneDecomp

RestrictedRankOneDecomp::usage=

”RestrictedRankOneDecomp[r,theta] computes the restricted rank one decomposition of an involution theta over a root system r.”;

```

RestrictedRankOneDecomp[r_?RootInputQ, theta_?MatrixQ] :=
  RestrictedRankOneDecomp[RootBase[r], theta];

RestrictedRankOneDecomp[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RestrictedRankOneDecomp[r,
    RootInvolution[r, List[argdisks], arches]];

RestrictedRankOneDecomp[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RestrictedRankOneDecomp[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches]];

RestrictedRankOneDecomp[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {i, j, k, rsets, cset, foundlist},

  (* Loop through all the basis roots that are not fixed *)
  (*
  Skip any roots which are members of an already constructed \
rankOneHelBasis set *)
  rsets = {};
  foundlist = {};

  For[i = 1, i <= Length[d], i++,
    If[FixedRootQ[d, theta, d[[i]]],
      Continue[];
    ];

    cset = RankOneComponentBasis[d, theta, d[[i]]];

    (*rsets=Union[rsets,{cset}];*)
    rsets = Join[rsets, {cset}];

    (* Keep track of all the roots we found *)

    For[j = 1, j <= Length[cset], j++,
      foundlist = Union[{cset[[j]]}, foundlist];
    ];
  ];

  (*Print["rsets=",rsets];*)

  (* There is a chance that some irreducible component consists \
entirely of fixed roots. *)
  While[Length[foundlist] < Length[d],
    (* Find a fixed root that isn't in the foundlist and link it with \
all the other fixed roots it is connected to *)

    For[i = 1, i <= Length[d], i++,
      If[MemberQ[foundlist, d[[i]]], Continue[]];

    cset = {d[[i]]};

    For[j = 1, j <= Length[cset], j++,

```



```

For[k = 1, k <= Length[d], k++,
  If[RootBasisConnectedQ[d, cset[[j]], d[[k]]],
    cset = Union[cset, {d[[k]]}];
    foundlist = Union[{d[[k]]}, foundlist];
  ];
]; (* END FOR k *)
]; (* END FOR j *)

(* Sort each element of cset *)

cset = ByBasisSort[d, cset];

(* Put cset in the proper position *)

For[j = 1, j <= Length[rsets], j++,
  If[BasisOrder[cset[[1]], rsets[[j, 1]]],
    rsets = Insert[rsets, cset, j];
    Break[];
  ];
]; (* END FOR i *)
]; (* END WHILE *)

Return[rsets];
];

RestrictedRankOneDecomp[a___] :=
  InvalidArg["RestrictedRankOneDecomp", a];

```

12.11.21 RestrictedRankOneSystem

RestrictedRankOneSystem::usage=

”RestrictedRankOneSystem[r,disks,arches,root] computes the restricted rank one root system with respect to root, defined by an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRankOneSystem[r,theta,root] computes the restricted rank one root system with respect to root, defined by an involution theta over root system r.”;

```
RestrictedRankOneSystem[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RestrictedRankOneSystem[r, List[argdisks], arches, root];

RestrictedRankOneSystem[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] := Module[
  {restroot, theta, roots, i, rset, v, k, soln, x, disks},

  disks = List[argdisks];
  rset = {};
  roots = RootSystemFromBasis[r];
  theta = RootInvolution[r, disks, arches];

  (* Compute the restricted root *)

  restroot = LocalProject[r, theta, root];

  (* Want to find all roots B so that RESTRICT (B) is a integral \
multiple of RESTROOT *)
  For[i = 1, i <= Length[roots], i++,
    v = LocalProject[r, theta, roots[[i]]];

    soln = Flatten[Solve[v == k*restroot, k]];
    x = k /. soln;

    If[IntegerQ[x],
      rset = Join[{roots[[i]]}, rset];
    ];
  ];

  Return[rset];
];

RestrictedRankOneSystem[r_?RootInputQ, theta_?MatrixQ,
  root_?VectorQ] :=
  RestrictedRankOneSystem[RootBase[r], theta, root];

RestrictedRankOneSystem[r_?RootBasisQ, theta_?MatrixQ,
  root_?VectorQ] := Module[
  {restroot, roots, i, rset, v, k, soln, x},

  rset = {};
  r;
  roots = RootSystemFromBasis[r];

  (* Compute the restricted root *)

  restroot = LocalProject[r, theta, root];
```

```

(*Print[restroot];*)

(* Want to find all roots B so that RESTRICT (B) is a integral \
multiple of RESTROOT *)
For[i = 1, i <= Length[roots], i++,
  v = LocalProject[r, theta, roots[[i]]];

  soln = Flatten[Solve[v == k*restroot, k]];
  x = k /. soln;

  If[IntegerQ[x],
    rset = Join[{roots[[i]]}, rset];
  ];
];

Return[rset];
];

RestrictedRankOneSystem[a___] :=
  InvalidArg["RestrictedRankOneSystem", a];

```

12.11.22 RestrictedRootBasis

RestrictedRootBasis::usage=

”RestrictedRootBasis[r,disks,arches] gives a basis for the restricted root system determined by an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRootBasis[r,theta] gives a basis for the restricted root system determined by an involution theta over root system r.”;

```

RestrictedRootBasis[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RestrictedRootBasis[RootBase[r], List[argdisks], arches];

RestrictedRootBasis[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] := Module[
  {theta, basis, wd, i, n, disks},

  disks = List[argdisks];

  theta = RootInvolution[r, disks, arches];
  n = Length[theta];

  (* White dots... calculate (1/2) (a - THETA a) for roots a *)
  wd = {};

  For[i = 1, i <= n, i++,
    If[MemberQ[disks, i], Continue[]];

    wd = Union[
      wd, {(1/2)*UnitVector[n, i].r - (1/2)*
        UnitVector[n, i].Transpose[theta].r}];
  ];

  Return[wd];
];

RestrictedRootBasis[r_?RootInputQ, theta_?MatrixQ] :=
  RestrictedRootBasis[RootBase[r], theta];

RestrictedRootBasis[r_?RootBasisQ, theta_?MatrixQ] := Module[
  {wd, i, n},

  n = Length[theta];

  (* White dots... calculate (1/2) (a - THETA a) for roots a *)
  wd = {};

  For[i = 1, i <= n, i++,
    (* if it is a fixed root, continue *)

    If[SameQ[ApplyRootInvolution[r, theta, r[[i]]], r[[i]]],
      Continue[]];

    wd = Union[
      wd, {(1/2)*UnitVector[n, i].r - (1/2)*
        UnitVector[n, i].Transpose[theta].r}];
  ];
];

```

```
Return[wd];  
];  
  
RestrictedRootBasis[a_...] := InvalidArg["RestrictedRootBasis", a];
```

12.11.23 RestrictedRootRank

RestrictedRootRank::usage=

”RestrictedRootRank[r,disks,arches] gives the rank of the restricted root system determined over an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRootRank[r,theta] gives the rank of the restricted root system determined by an involution theta over root system r.”;

```
RestrictedRootRank[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  Length[RestrictedRootBasis[r, List[argdisks], arches]];

RestrictedRootRank[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  Length[RestrictedRootBasis[r, List[argdisks], arches]];

RestrictedRootRank[r_?RootInputQ, theta_?MatrixQ] :=
  Length[RestrictedRootBasis[r, theta]];

RestrictedRootRank[r_?RootBasisQ, theta_?MatrixQ] :=
  Length[RestrictedRootBasis[r, theta]];

RestrictedRootRank[a___] := InvalidArg["RestrictedRootRank", a];
```

12.11.24 RestrictedRootSystem

RestrictedRootSystem::usage=

”RestrictedRootSystem[r,disks,arches] computes the set of all roots composing the restricted root system determined over an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRootSystem[r,theta] computes the set of all roots composing the restricted root system determined by an involution theta over root system r.”;

```

RestrictedRootSystem[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] := Module[
  {rs, basis, disks},

  disks = List[argdisks];
  basis = RestrictedRootBasis[r, disks, arches];
  rs = RootSystemFromBasis[basis];

  Return[rs];
];

RestrictedRootSystem[r_?RootInputQ, theta_?MatrixQ] := Module[
  {rs, basis},

  basis = RestrictedRootBasis[r, theta];
  rs = RootSystemFromBasis[basis];

  Return[rs];
];

RestrictedRootSystem[basis_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RestrictedRootSystem[basis, RootInvolution[List[argdisks], arches]];

RestrictedRootSystem[basis_?RootBasisQ, theta_?MatrixQ] := Module[
  {rs, rbasis},

  rbasis = RestrictedRootBasis[basis, theta];
  rs = RootSystemFromBasis[rbasis];

  Return[rs];
];

RestrictedRootSystem[a___] := InvalidArg["RestrictedRootSystem", a];

```

12.11.25 RestrictedRootSystemType

RestrictedRootSystemType::usage=

”RestrictedRootSystemType[r,disks,arches] identifies the type of the restricted root system determined by an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRootSystemType[r,theta] identifies the type of the restricted root system determined by an involution theta over root system r.”;

```

RestrictedRootSystemType[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  CartanToRootSystem[
    CartanMatrixFromBasis[
      RestrictedRootBasis[r, List[argdisks], arches]]];

RestrictedRootSystemType[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  CartanToRootSystem[
    CartanMatrixFromBasis[
      RestrictedRootBasis[r, List[argdisks], arches]]];

RestrictedRootSystemType[r_?RootInputQ, theta_?MatrixQ] :=
  CartanToRootSystem[
    CartanMatrixFromBasis[RestrictedRootBasis[r, theta]]];

RestrictedRootSystemType[r_?RootBasisQ, theta_?MatrixQ] :=
  CartanToRootSystem[
    CartanMatrixFromBasis[RestrictedRootBasis[r, theta]]];

RestrictedRootSystemType[a___] :=
  InvalidArg["RestrictedRootSystemType", a];

```


12.11.26 RootCriticalValues

RootCriticalValues::usage=

"RootCriticalValues[d,theta,cconsts] returns a table of theta(d) where theta is an automorphism of the root system r, and cconsts is the table of all structure constants returned by structureConstants.d) where theta is an automorphism of the root system r described by fixed roots disks and diagram automorphism arches, and cconsts is the table of all structure constants returned by structureConstants.

RootCriticalValues[d,disks,arches,cconsts] returns a table of theta(";

```
RootCriticalValues[r_?RootInputQ, theta_?MatrixQ, cvals_?ListQ] :=
  RootCriticalValues[RootBase[r], theta, cvals];

RootCriticalValues[basis_?RootBasisQ, theta_?MatrixQ, cvals_?ListQ] :=
  Module[
    {},
    Return[
      Table[StructureConstantsLookup[cvals,
        ApplyRootInvolution[basis, theta, basis[[i]]], {i, 1,
          Length[basis]}]];
    ];

RootCriticalValues[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, cvals_?ListQ] :=
  RootCriticalValues[RootBase[r],
    RootInvolution[r, List[argdisks], arches], cvals];

RootCriticalValues[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, cvals_?ListQ] :=
  RootCriticalValues[r, RootInvolution[r, List[argdisks], arches],
    cvals];

RootCriticalValues[a___] := InvalidArg["RootCriticalValues", a];

(*
rootCvals[basis_,theta_,cvals_] := Table[cvals[UnitVector[Length[\
basis],i].Transpose[theta].basis],{i,1,Length[basis]}];
*)
```

12.11.27 RootInvolution

RootInvolution::usage=

"RootInvolution[r,disks,arches] or RootInvolution[r,disks,arches] gives the matrix of an involution acting on root system r, where disks are those points represented by solid black disk, and arches is a list of elements a,b denoting the diagram automorphism maps a to b."

```
RootInvolution::fixedroots =
  "Warning. Did not produce an involution with fixed indices '1'. \
  Check definition of diagram involution.";

RootInvolution[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  RootInvolution[RootBase[r], List[argdisks], arches];

RootInvolution[d_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  Module[
    {disks, w0, thetastar, i, ri},

    disks = List[argdisks];

    (* 1. Compute the action of w0 *)

    w0 = wInvolutionAction[d, disks];

    (* 2. Compute the matrix for thetastar *)

    thetastar = IdentityMatrix[Length[d]];

    For[i = 1, i <= Length[arches], i++,
      thetastar = RowSwap[thetastar, arches[[i, 1]], arches[[i, 2]]];
    ];

    thetastar = Transpose[thetastar];

    ri = -thetastar.w0;
    (*
    If[!SameQ[disks,DiskList[d,ri]],
    Print[DiskList[d,ri]];
    Message[RootInvolution::fixedroots,disks];
    ];
    *)
    Return[ri];
  ];

RootInvolution[a___] := InvalidArg["RootInvolution", a];

(*
rootInvolutionSimple::usage=\\
"rootInvolutionSimple[r,disks,arches,\\
offSet] or rootInvolutionSimple[{r,disks,arches}] gives the matrix of \\
an involution acting on an irreducible root system r, where disks are \\
those points represented by solid black disk, and arches is a list of \\
elements {a,b} denoting the diagram automorphism maps a to b. The \\
optional argument offSet shifts the indices by the specified number \\
of units.";
*)

(*
rootInvolution[td_]:=rootInvolution[td[[1]],td[[2]],td[[3]]];
*)
```

```

(*
rootInvolutionSimple[td_,offset_]:=rootInvolutionSimple[td[[1]],td[[2]],td[[3]],offset];
*)

(*
rootInvolution[r_,disks_,arches_]:=Module[
{diskset,archset,marches,multiarches,block,i,j,k,rin,set,offset,tMatr,\
thetastar,totaldim},

offset=0;
marches=arches;
rin=rootInput[r];

(* 1. First split disks and arches into little groups according to \
their individual irreducible root systems. This shouldn't be \
difficult, because we know the dimensions of each component.

Any pairs in arches which cross multiple irred components will go in \
their own list to be evaluated later *)
For[i=1,i<=Length[rin],i++,
(* Create a set of all disks in this irred component *)
\
set=Table[j,{j,offset+1,offset+rin[[i,2]]}];

(* Intersect this set with the real set of disks to get just the ones \
in this component *)
diskset[i]=Intersection[set,disks];

(* Create a set of all possible ordered pairs in this component *)
\
set=Table[Table[{j,k},{k,offset+1,offset+rin[[i,2]]},{j,offset+1,\
offset+rin[[i,2]]}];
set=Flatten[set,1];

(* Intersect with "real" set to get this component *)
\
archset[i]=Intersection[set,marches];

(* Modify offset *)
offset+=rin[[i,2]];

(* Delete archset from marches so marches contains only multiple \
component spanning elements *)
\
marches=Complement[marches,archset[i]];
];

(* At the end of this loop, offset becomes the value of the total \
dimension (sum of dim's of irred components) *)
totaldim=offset;
offset=0;

(* 2. We now have sets for each irred component. Pass them to \
rootInvolutionSimple. *)
For[i=1,i<=Length[rin],i++,
block[i]=rootInvolutionSimple[rin[[i]],diskset[i],archset[i],offset];
offset+=rin[[i,2]];
];

(* 3. Build a big block matrix *)

```

```

\
tMatr=blockAssemble[Table[block[i],{i,1,Length[rin]}]];

(* 4. Swap rows according to arches:multiple irred components *)
(* \
Build matrix for thetastar *)

thetastar=IdentityMatrix[totaldim];

For[i=1,i<=Length[marches],i++,
thetastar=rowSwap[thetastar,marches[[i,1]],marches[[i,2]]];
];

thetastar=Transpose[thetastar];

Return[thetastar.tMatr];
];
*)

(* Assumes r is irreducible. *)
(*
\
rootInvolutionSimple[r_,disks_,arches_,offset_:0]:=Module[
{rin,basis,dim,w,w0,w0theta,theta,thetastar,
esystems,i,row,ds,emsys},

rin=Flatten[rootInput[r]]; (* r is simple *)

basis=base[rin];
dim=Length[basis];

(* 1. Determine the embedded root systems *)
\
esystems=embeddedRootSystems[rin,disks-offset,arches];

(* For every root system *)
w0={};

For[i=1,i<=Length[esystems],i++,
(* Ugh. Have to do some "E-hacking" because of the numbering of the \
roots of E
The 1,2,3,... disks labels here assumes r is irred. Otherwise we'd \
need to account for offset. *)
\
emsys=Flatten[rootInput[esystems[[i,1]]]]; (* embedded system *)

If[SameQ[rin[[1]],"E"]&&!SameQ[emsys[[1]],"E"]&&MemberQ[esystems[[i,2]\
],2]&&MemberQ[esystems[[i,2]],3],
(* The problem is if we have a root system of Type E then roots 2 and \
3 are not joined. This results in "weird" Cartan Matrices for types \
A, D. *)
ds=esystems[[i,2]];

(* If we have 1 and 3 included AND the embedded root system is not E, \
then 6 is not included.
So if 5 is included, we have type D5.
If 5 is not included, we have type A4 *)
If[MemberQ[esystems[[i,2]],1],
(* 5 included? *)
If[MemberQ[esystems[[i,2]],5],
ds={1,3,4,2,5};
, (* else *)

```

```

ds={1,3,4,2};
];
, (* else 1 not included *)
(* Without 1 then we have type D for \
sure. Simple thing to do is just run through all cases for disk sets *)
\
Switch[Max[esystems[[i,2]]],
8,ds={8,7,6,5,4,2,3};,
7,ds={7,6,5,4,2,3};,
6,ds={6,5,4,2,3};,
5,ds={3,4,2,5};,
4,ds={3,4,2};
];
];
w=longestElement[esystems[[i,1]],ds];
,
w=longestElement[esystems[[i,1]],esystems[[i,2]]];
];

w0=Join[w0,w];
];

(* Build matrix for w0 *)
For[i=1,i<=Length[basis],i++,
row[i]=basisCoeffs[basis,reflectWeyl[basis,w0,basis[[i]]]];
];

w0theta=Table[row[i],{i,1,Length[basis]}];

w0theta=Transpose[w0theta];

(* Build matrix for thetastar *)
thetastar=IdentityMatrix[dim];

For[i=1,i<=Length[arches],i++,
thetastar=rowSwap[thetastar,arches[[i,1]]-offSet,arches[[i,2]]-offSet];
];

thetastar=Transpose[thetastar];

(* FORMULA is THETA = -ID Subscript[W, 0](THETA) \
THETA-STAR    XXX see below *)
\
(*theta=-IdentityMatrix[dim].w0theta.thetastar;*)

(* FORMULA is THETA = -ID THETA-STAR Subscript[W, 0](THETA)      p34 \
HEL88 *)
theta=-IdentityMatrix[dim].thetastar.w0theta;

Return[theta];
];
*)

```

12.11.28 RRD Lift

RRDLift::usage=

"RRDLift[r,theta,cconsts,nvals] lifts an involution theta, with respect to root system r, to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. Output is in list format. nvals supplies the Chevalley constants.

RRDLift[r,disks,arches,cconsts,nvals] lifts an involution theta (described via fixed roots disks and diagram automorphism arches), with respect to root system r, to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. Output is in list format. nvals supplies the Chevalley constants."

```
RRDLift[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cvals_?ListQ, nvals_?ListQ] :=
  RRDLift[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], cvals, nvals];

RRDLift[d_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cvals_?ListQ, nvals_?ListQ] :=
  RRDLift[d, RootInvolution[d, List[argdisks], arches], cvals, nvals];

RRDLift[r_?RootInputQ, theta_?MatrixQ, cvals_?ListQ, nvals_?ListQ] :=
  RRDLift[RootBase[r], theta, cvals, nvals];

RRDLift[d_?RootBasisQ, theta_?MatrixQ, cvals_?ListQ, nvals_?ListQ] :=
Module[
  {rrd, rn, rbasis, rtheta, rcm, ret, sret, roots, i, j, delList,
    rname},

  (* Compute the restricted rank one decomposition *)

  roots = RootSystemFromBasis[d];

  (*Print["begin RROD."];*)

  rrd = RestrictedRankOneDecomp[d, theta];
  (*Print["end RROD."];*)

  ret = {};

  (* Lift all the rank one components *)

  For[rn = 1, rn <= Length[rrd], rn++,
    (* Compute the RRO data *)
    rbasis[rn] = rrd[[rn]];
    (*rname[rn]=BasisToRootSystem[rbasis[rn]];*)

    rtheta[rn] = RestrictedRootAut[d, theta, rbasis[rn]];

    (* Lift *)

    sret[rn] = RankOneRootLift[rbasis[rn], rtheta[rn], cvals, nvals];

    (* Align the polarity of the previous component to match this \
one. If we switch this one, have to check all previous! *)
    (*
```

```

WARNING! This code relies on rrd being nicely sorted!

TO DO: Remed that! *)
If[rn > 1,
  For[i = rn, i > 1, i--,
    sret[i - 1] =
      AlignPolarities[rbasis[i - 1], rtheta[i - 1], sret[i - 1],
        rbasis[i], rtheta[i], sret[i]];
  ];
];

(* Merge *)
For[rn = 1, rn <= Length[rrd], rn++,
  ret = Union[ret, sret[rn]];
];

(* Now we need to add all the roots which are not exclusively \
residing in one RRO component. *)
(* This is fairly tricky.
This root is the sum of two roots, each which:
  1. Reside in a restricted root system
  2. Were previously computed earlier in this loop.

The strategy:
modify De Graaf's algorithm to go through each level.
The lowest level (L2)
are sums of basis roots,
whose structure constants we know are already computed. Proceed
  building up from there. *)

ret = gMergeInvolutions[d, theta, ret, nvals];

Return[ret];
];

RRDLift[a_...] := InvalidArg["RRDLift", a];

```

12.11.29 RROITable

RROITable::usage=

"RROITable[n] generates the table of Helminck diagrams for involutions (over the root systems) of restricted rank one and of minimal size. The optional argument n returns the n entry (1 - 18), while the absence of an argument generates the entire table. For more details, please see the manual."

```
RROITable[n_: {0}] := Module[
  {entry, i},

  If[! IntegerQ[n] || n > 18 || n < 0,
    Message[RROITable::arg, n];
    Return[{}];
  ];

  (* Just list all the entries. This is a fixed table. *)
  (*
  Name, Disks, Arches, 1 Consistency *)

  entry[1] = {"A1+A1", {}, {{1, 2}}, True, 2};
  entry[2] = {"A1", {}, {}, True, 1};
  entry[3] = {"A2", {2}, {}, False, 2};
  entry[4] = {"A3", {1, 3}, {}, True, 3};
  entry[5] = {"A4", {2, 3}, {{1, 4}, {2, 3}}, True, 4};

  entry[6] = {"B3", {2, 3}, {}, True, 3};
  entry[7] = {"B4", {1, 3, 4}, {}, False, 4};
  entry[8] = {"C4", {2, 3, 4}, {}, False, 4};
  entry[9] = {"C4", {1, 3, 4}, {}, True, 4};
  entry[10] = {"D5", {2, 3, 4, 5}, {}, True, 5};

  entry[11] = {"D6", {1, 3, 4, 5, 6}, {}, False, 6};
  entry[12] = {"E6", {1, 3, 4, 5, 6}, {{1, 6}, {3, 5}}, False, 6};
  entry[13] = {"E7", {2, 3, 4, 5, 6, 7}, {}, False, 7};
  entry[14] = {"E8", {1, 2, 3, 4, 5, 6, 7}, {}, False, 8};
  entry[15] = {"F4", {2, 3, 4}, {}, False, 4};

  entry[16] = {"F4", {1, 2, 3}, {}, True, 4};
  entry[17] = {"G2", {1}, {}, False, 2};
  entry[18] = {"G2", {2}, {}, False, 2};

  If[SameQ[n, 0],
    Return[Table[entry[i], {i, 1, 18}]];
  ,
    Return[entry[n]];
  ];
];

RROITable[a_...] := InvalidArg["RROITable", a];
```


12.11.30 RROITableEntry

RROITableEntry::usage=

"RROITableEntry[r,theta] identifies the Restricted Rank One Involutions table entry corresponding to an involution theta over root system r.

RROITableEntry[r,disks,arches] identifies the Restricted Rank One Involutions table entry corresponding to an involution theta (given by fixed roots disks and diagram automorphism arches) over root system r."

```
RROITableEntry::arg =
  "supplied involution is not restricted rank one.";

RROITable::arg =
  "'1' is not an integer in the range of 1 through 18.";

RROITableEntry[r_?RootInputQ, theta_?MatrixQ] := Module[
  {disks, arches, d},

  d = RootBase[r];
  disks = DiskList[d, theta];
  arches = ArchesListInvolution[d, disks, theta];

  Return[RROITableEntry[d, disks, arches]];
];

RROITableEntry[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {disks, arches},

  disks = DiskList[d, theta];
  arches = ArchesListInvolution[d, disks, theta];
  (*Print["rroite."];*)

  Return[RROITableEntry[d, disks, arches]];
];

RROITableEntry[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  RROITableEntry[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches]];

(* Need the basis because the root system may be 'oriented' the \
other way... e.g. o <= o--o--o *)

RROITableEntry[basis_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] := Module[
  {r, i, rin, isleft, n, rin2, alpha, ret, disks},

  (* Q: Why do we identify a root system if the RootInput variants \
of the command convert to a basis?

  A related problem is that 'basis' may be different than the \
standard one provided by MLAP. If so,
  the matrix for theta may be with respect to this basis.
  We do not want to compute a new basis based off of the root system \
name, because it may not correlate to the matrix for theta. *)

  disks = List[argdisks];
  r = BasisToRootSystem[basis];
```

```

rin = RootInput[r];

(*isleft=dynkinOrientation[r,basis];*)
isleft = False;
n = Length[basis];

If[! SameQ[RestrictedRootRank[basis, disks, arches], 1],
  Message[RRUITableEntry::arg];
  Return[{}];
];

(* Handle type 1 *)
If[Length[rin] > 1,
  Return[1];
];

rin = Flatten[rin];

Switch[rin[[1]],
  "A",
  If[! SameQ[arches, {}],
    Return[5];
  ];

  Switch[n,
    1, ret = 2;
    2, ret = 3;
    3, ret = 4;
  ];

  Return[ret];
,
  "B",
  If[! isleft,
    If[! FixedRootQ[basis, disks, arches, basis[[1]]],
      Return[6];
    ,
      Return[7];
    ];
  ,
  If[! FixedRootQ[basis, disks, arches, basis[[n]]],
    Return[6];
  ,
    Return[7];
  ];
];
,
  "C",
  If[! isleft,
    If[! FixedRootQ[basis, disks, arches, basis[[1]]],
      Return[8];
    ,
      Return[9];
    ];
  ,
  If[! FixedRootQ[basis, disks, arches, basis[[n]]],
    Return[8];
  ,
    Return[9];
  ];
];

```

```

,
"D",
If[! isleft,
  If[! FixedRootQ[basis, disks, arches, basis[[1]]],
    Return[10];
  ,
  Return[11];
];
,
If[! FixedRootQ[basis, disks, arches, basis[[n]]],
  Return[10];
,
  Return[11];
];
];
,
"E",
Switch[n,
  6, Return[12];,
  7, Return[13];,
  8, Return[14];
];
,
"F",
rin2 =
  RootInput[BasisToRootSystem[FixedBasis[basis, disks, arches]]];
rin2 = Flatten[rin2];

Switch[rin2[[1]],
  "C", Return[15];
,
  "B", Return[16];
];
,
"G",
(* Let alpha be the shorter of the two roots *)

If[Norm[basis[[1]]] > Norm[basis[[2]]],
  alpha = basis[[2]];
,
  alpha = basis[[1]];
];

If[FixedRootQ[basis, disks, arches, alpha],
  Return[17];
,
  Return[18];
];
];
];

RRUITableEntry[a___] := InvalidArg["RRUITableEntry", a];

```

12.11.31 SimpleRootLift

SimpleRootLift::usage=

”SimpleRootLift[d,theta,cconsts,nvals,donly] lifts an involution theta with respect to root system r to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. nvals supplies the Chevalley constants. The technique used is the simpler Groebner basis technique, which is slow. For a quicker calculation, use RRD Lift. simpleLift does not check for 1-consistency. Output is in list format. If the optional argument donly is True, only the basis structure constants are returned.

SimpleRootLift[d,disks,arches,cconsts,nvals,donly] lifts an involution theta (defined via fixed roots disks and diagram automorphism arches) with respect to root system r to an involution on its corresponding Lie algebra. cconsts supplies the structure constants. nvals supplies the Chevalley constants. The technique used is the simpler Groebner basis technique, which is slow. For a quicker calculation, use RRD Lift. simpleLift does not check for 1-consistency. Output is in list format. If the optional argument donly is True, only the basis structure constants are returned.”;

```
SimpleRootLift[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cvals_?ListQ, nvals_?ListQ, donly_: False] :=
SimpleRootLift[RootBase[r],
  RootInvolution[RootBase[r], List[argdisks], arches], cvals, nvals,
  donly];

SimpleRootLift[d_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cvals_?ListQ, nvals_?ListQ, donly_: False] :=
SimpleRootLift[d, RootInvolution[d, List[argdisks], arches], cvals,
  nvals, donly];

SimpleRootLift[r_?RootInputQ, theta_?MatrixQ, cvals_?ListQ,
  nvals_?ListQ, donly_: False] :=
SimpleRootLift[RootBase[r], theta, cvals, nvals, donly];

SimpleRootLift[d_?RootBasisQ, theta_?MatrixQ, cvals_?ListQ,
  nvals_?ListQ, donly_: False] := Module[
  {cv, k, i, coeff, ret, root, kv, kf, rk, roots, trr, nega},

  trr = True;
  nega = False;

  While[trr,
    ret = {};
    (* Obtain the correction vector *)

    cv = OneCorrectionVector[d, theta, cvals, k];
    kv = Table[k[i], {i, 1, Length[d]}] /. cv;
    (*kv=Transpose[{kv}];*)
    (*kv=kv.Transpose[
    theta];*)
    (*kv=-kv;*)
```

```

If[nega, kv = -kv; trr = False;];

(*Print["kv=",kv];*)

(* Build the list of roots and coefficients *)

For[i = 1, i <= Length[d], i++,
  root = d[[i]];
  coeff = StructureConstantsLookup[cvals, root];
  kf =
    RootFunctional[d, ApplyRootInvolutionBasis[d, theta, root],
      kv];
  coeff = coeff*kf;
  (*
  If[isFixedRootBasis[d,theta,root],
    coeff*=-1;
  ];
  *)
  ret = Union[ret, {{root, coeff}}];
  ret = Union[ret, {{-root, 1/coeff}}];

  (* If root is fixed then it is in the +1 Eigenspace of theta... \
Hence, [Xa,X (-a)] = +1 Ha
Else, it is in -1... [Xa,X (-a)] = - Ha *)
  (*
  If[
    isFixedRootBasis[d,theta,root],
    ret=Union[ret,{{-root,1/coeff}}];
  ,
    ret=Union[ret,{{-root,1/coeff}}];
  ];
  *)
  (*ret=Union[ret,{{-root,1/coeff}}];*)
];

(* Build the list of roots and coefficients *)
(*
For[i=1,
i<=Length[d],i++,
root=-d[[i]];
coeff=structureConstantsLookup[cvals,root];
kf=rootFunctional[d,applyInvolutionBasis[d,theta,root],kv];
coeff=coeff*kf;
ret=Union[ret,{{root,coeff}}];
];
*)
If[! donly,
  ret = gMergeInvolutions[d, theta, ret, nvals];
];

If[InvBTRaw[d, theta, ret] != 0, nega = True;, Break[];];
];

Return[FullSimplify[ret]];
];

SimpleRootLift[a_...] := InvalidArg["SimpleRootLift", a];

(*
simpleLift2[d_,theta_,systemMatrix_,cvals_,nvals_,donly_:False]:=\\
Module[

```

```

{cv,k,i,coeff,ret,root,kv,kf,rk,roots},

ret={};

(* Obtain the correction vector *)
\
cv=oneCorrectionVector[d,theta,systemMatrix,cvals,k];
kv=Table[k[i],{i,1,Length[d]}]/.cv;
(*kv=Transpose[{kv}];*)
(*kv=kv.Transpose[theta];*)
(*kv=-kv;*)

(*Print["kv=",kv];*)

(* Build the list of roots and coefficients *)
\
For[i=1,i<=Length[d],i++,
root=d[[i]];
coeff=structureConstantsLookup[cvals,root];
kf=rootFunctional[d,applyInvolutionBasis[d,theta,root],kv];
coeff=coeff*kf;
(*
If[isFixedRootBasis[d,theta,root],
coeff*=-1;
];
*)
ret=Union[ret,{{root,coeff}}];
ret=Union[ret,{{-root,1/coeff}}];

(* If root is fixed then it is in the +1 Eigenspace of theta... \
Hence,  $[X_a, X(-a)] = +1 H_a$ 
Else, it is in -1...  $[X_a, X(-a)] = - H_a$  *)
(*
\
If[isFixedRootBasis[d,theta,root],
ret=Union[ret,{{-root,1/coeff}}];
,
ret=Union[ret,{{-root,1/coeff}}];
];
*)
(*ret=Union[ret,{{-root,1/coeff}}];*)
];

(* Build the list of roots and coefficients *)
(*
\
For[i=1,i<=Length[d],i++,
root=-d[[i]];
coeff=structureConstantsLookup[cvals,root];
kf=rootFunctional[d,applyInvolutionBasis[d,theta,root],kv];
coeff=coeff*kf;
ret=Union[ret,{{root,coeff}}];
];
*)
If[!donly,
ret=gMergeInvolutions[d,theta,ret,nvals];
];

Return[FullSimplify[ret]];
];
*)

```

```

(*
simpleLift3[d_,theta_,systemMatrix_,cvals_,nvals_,donly_:False]:=
Module[
{cv,k,i,coeff,ret,root,kv,kf,rk,roots},

ret={};

(* Obtain the correction vector *)
\
cv=oneCorrectionVector[d,theta,systemMatrix,cvals,k];
kv=Table[k[i],{i,1,Length[d]}]/.cv;
(*kv=Transpose[{kv}];*)

(* Build the list of roots and coefficients *)
\
roots=rootSystemFromBasis[d];
For[i=1,i<=Length[roots],i++,
root=roots[[i]];
coeff=structureConstantsLookup[cvals,root];
kf=rootFunctional[d,applyInvolutionBasis[d,theta,root],kv];
coeff=coeff*kf;
ret=Union[ret,{{root,coeff}}];
(*ret=Union[ret,{{-root,1/coeff}}];*)
];

If[!donly,
ret=gMergeInvolutions[d,theta,ret,nvals];
];

Return[FullSimplify[ret]];
];
*)

(*
simpleLiftSpecifySN[d_,theta_,systemMatrix_,cvals_,nvals_,sn_]:=
Module[
{cv,k,i,coeff,ret,root,kv,kf,rk,roots,trr,nega},

trr=True;
nega=False;

While[trr,
ret={};
(* Obtain the correction vector *)
\
cv=oneCorrectionVector[d,theta,systemMatrix,cvals,k,sn];
kv=Table[k[i],{i,1,Length[d]}]/.cv;
(*kv=Transpose[{kv}];*)
(*kv=kv.Transpose[theta];*)
(*kv=-kv;*)

If[nega,kv=-kv;trr=False;];

(*Print["kv=",kv];*)

(* Build the list of roots and coefficients *)
\
For[i=1,i<=Length[d],i++,
root=d[[i]];
coeff=structureConstantsLookup[cvals,root];
kf=rootFunctional[d,applyInvolutionBasis[d,theta,root],kv];
coeff=coeff*kf;

```

```

(*)
If[isFixedRootBasis[d,theta,root],
coeff*=-1;
];
*)
ret=Union[ret,{{root,coeff}}];
ret=Union[ret,{{-root,1/coeff}}];

(* If root is fixed then it is in the +1 Eigenspace of theta... \
Hence, [Xa,X (-a)] = +1 Ha
Else, it is in -1... [Xa,X (-a)] = - Ha *)
(*)
\
If[isFixedRootBasis[d,theta,root],
ret=Union[ret,{{-root,1/coeff}}];
,
ret=Union[ret,{{-root,1/coeff}}];
];
*)
(*ret=Union[ret,{{-root,1/coeff}}];*)
];

(* Build the list of roots and coefficients *)
(*)
\
For[i=1,i<=Length[d],i++,
root=-d[[i]];
coeff=structureConstantsLookup[cvals,root];
kf=rootFunctional[d,applyInvolutionBasis[d,theta,root],kv];
coeff=coeff*kf;
ret=Union[ret,{{root,coeff}}];
];
*)

ret=gMergeInvolutions[d,theta,ret,nvals];

If[invBTRaw[d,theta,ret]!=0,nega=True;,Break[]];
];

Return[FullSimplify[ret]];
];
*)

```


12.11.32 SteinbergThetaDelta

SteinbergThetaDelta::usage=

”SteinbergThetaDelta[r,nconsts,theta] returns a list of structure constants for the automorphism over the Lie algebra TD with root system r and root involution theta. TD is the unique automorphism such that each basis root structure constant is 1. nconsts supplies the Chevalley constants. The table returned is a list of elements of the form ROOT, CONSTANT. If nconsts is omitted, the procedure KleinChevalley will be called. However, for repeated usage it is recommended to compute once and store the Chevalley constants in memory.”;

```
SteinbergThetaDelta[d_?RootBasisQ, nconsts_?ListQ, theta_?MatrixQ] :=
  StructureConstantsFromBasis[d, ConstantArray[1, Length[d]], nconsts,
    theta];

SteinbergThetaDelta[r_?RootInputQ, nconsts_?ListQ, theta_?MatrixQ] :=
  Module[
    {d},

    d = RootBase[r];
    Return[
      StructureConstantsFromBasis[d, ConstantArray[1, Length[d]],
        nconsts, theta]];
  ];

SteinbergThetaDelta[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {nconsts},

  nconsts = KleinChevalley[d];
  Return[
    StructureConstantsFromBasis[d, ConstantArray[1, Length[d]],
      nconsts, theta]];
  ];

SteinbergThetaDelta[r_?RootInputQ, theta_?MatrixQ] := Module[
  {d, nconsts},

  d = RootBase[r];
  nconsts = KleinChevalley[d];
  Return[
    StructureConstantsFromBasis[d, ConstantArray[1, Length[d]],
      nconsts, theta]];
  ];

SteinbergThetaDelta[a___] := InvalidArg["SteinbergThetaDelta", a];
```

12.11.33 SwitchPolarity

SwitchPolarity::usage=

"SwitchPolarity[r,theta,cconsts] reverses the polarity of an involution over the Lie algebra defined by cconsts. theta is the involution on the root system r.

SwitchPolarity[r,disks,arches,cconsts] reverses the polarity of an involution over the Lie algebra defined by cconsts. theta is the involution, described by fixed roots disks and diagram automorphism arches, on the root system r."

```
SwitchPolarity[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cconsts_?ListQ] :=
  SwitchPolarity[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], cconsts];

SwitchPolarity[d_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  cconsts_?ListQ] :=
  SwitchPolarity[d, RootInvolution[d, List[argdisks], arches],
    cconsts];

SwitchPolarity[r_?RootInputQ, theta_?MatrixQ, cconsts_?ListQ] :=
  SwitchPolarity[RootBase[r], theta, cconsts];

SwitchPolarity[d_?RootBasisQ, theta_?MatrixQ, cconsts_?ListQ] :=
  Module[
    {i, cvals, h, root, rsplit, a, b},

    cvals = cconsts;

    For[i = 1, i <= Length[cconsts], i++,
      root = cvals[[i, 1]];
      rsplit = RootSplit[d, root];
      a = rsplit[[1]];
      b = rsplit[[2]];

      Which[
        FixedRootQ[d, theta, root],
        h = RootHeight[d, root];

        If[OddQ[h],
          cvals[[i, 2]] = -cvals[[i, 2]];
          ,
          cvals[[i, 2]] = cvals[[i, 2]];
          ];

        ,
        FixedRootQ[d, theta, a],
        h = RootHeight[d, a];

        If[OddQ[h],
          cvals[[i, 2]] = -cvals[[i, 2]];
          ,
          cvals[[i, 2]] = cvals[[i, 2]];
          ];

        ,
        True,
        cvals[[i, 2]] = cvals[[i, 2]];
        ];
    ];
  ];
```

```
    Return[cvals];  
  ];  
SwitchPolarity[a___] := InvalidArg["SwitchPolarity", a];
```

12.11.34 ThetaStable

ThetaStable::usage=

”ThetaStable[h,d,theta] returns True if theta(x) is in the set h for all x in h, where d supplies the basis theta is defined over.”;

```
ThetaStable[h_?ListQ, d_?RootBasisQ, theta_?MatrixQ] := Module[
  {k, i, j, root, v, hbasis},

  hbasis = Table[Diagonal[h[d[[i]]]], {i, 1, Length[d]}];

  For[i = 1, i <= Length[d], i++,
    root = UnitVector[Length[d], i].Transpose[theta].d;
    k = BasisCoefficients[d, root];

    v = Sum[k[[j]]*h[d[[j]]], {j, 1, Length[d]}];

    k = BasisCoefficients[hbasis, Diagonal[v]];
    (*
    Print[k];
    *)
  ];

  ThetaStable[a___] := InvalidArg["ThetaStable", a];
```

12.12 Local Symmetric Spaces Package (Diagram)

12.12.1 RankOneDecompDiagram

RankOneDecompDiagram::usage=

”RankOneDecompDiagram[r,theta] draws the restricted rank one decomposition diagram for an involution theta over the root system r.”;

```
RankOneDecompDiagram[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RankOneDecompDiagram[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches]];

RankOneDecompDiagram[d_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RankOneDecompDiagram[d, RootInvolution[d, List[argdisks], arches]];

RankOneDecompDiagram[r_?RootInputQ, theta_?MatrixQ] :=
  RankOneDecompDiagram[RootBase[r], theta];

RankOneDecompDiagram[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {rrd, ti, rbasis, rtheta, rdisk, rarches, rname, rlbl, i},

  rrd = RestrictedRankOneDecomp[d, theta];

  For[ti = 1, ti <= Length[rrd], ti++,
    rbasis[ti] = rrd[[ti]]];
```

```

(*rname[ti]=BasisToRootSystem[rbasis[ti]];*)

rtheta[ti] = RestrictedRootAut[d, theta, rbasis[ti]];

(*
rdisks[ti]=DiskList[rname[ti],rtheta[ti]];
rarches[ti]=ArchesListInvolution[rdisks[ti],rtheta[ti]];
*)

rlbl[ti] =
  Table[Position[BasisCoefficients[d, rbasis[ti][[i]]], 1][[1,
    1]], {i, 1, Length[rbasis[ti]]}];

If[DynkinOrientation[rbasis[ti]] == -1,
  lbl[ti] = Reverse[rlbl[ti]];
];

(*TableForm[Table[HelminckDiagram[rname[ti],rdisks[ti],rarches[ti],
  lbl[ti]],{ti,1,Length[rrd]}]];*)

TableForm[
  Table[HelminckDiagram[rbasis[ti], rtheta[ti], lbl[ti]], {ti, 1,
    Length[rrd]}]]
];

RankOneDecompDiagram[a___] := InvalidArg["RankOneDecompDiagram", a];

```

12.12.2 ReduceRestrictedRankDiagram

ReduceRestrictedRankDiagram::usage=

”ReduceRestrictedRankDiagram[r,disks,arches,root] gives a Helminck diagram which illustrates the reduction of the restricted rank of an involution defined over root system r with fixed roots disks and diagram automorphism arches by eliminating root.”;

```

ReduceRestrictedRankDiagram[r_?RootInputQ, theta_?MatrixQ,
  root_?VectorQ] :=
  ReduceRestrictedRankDiagram[RootBase[r], theta, root];

ReduceRestrictedRankDiagram[r_?RootBasisQ, theta_?MatrixQ,
  root_?VectorQ] := Module[
  {disks, arches},

  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  Return[ReduceRestrictedRankDiagram[r, disks, arches, root]];
];

ReduceRestrictedRankDiagram[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  ReduceRestrictedRankDiagram[RootBase[r], List[argdisks], arches,
  root];

(* Visualize the reduced rank root system *)

ReduceRestrictedRankDiagram[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] := Module[
  {disks, rbasis, i, labelset, lb, j},

  j = 1;
  labelset = {};
  disks = List[argdisks];
  rbasis = ReduceRestrictedRank[r, disks, arches, root];

  For[i = 1, i <= Length[r], i++,
    Which[
      MemberQ[disks, i], lb = "0";
      ,
      MemberQ[rbasis, r[[i]]], lb = "[Lambda]" <> ToString[j]; j++;
      ,
      True, lb = "X";
    ];

    labelset = Join[labelset, {lb}];
  ];

  HelminckDiagram[r, disks, arches, labelset]
];

ReduceRestrictedRankDiagram[a___] :=
  InvalidArg["ReduceRestrictedRankDiagram", a];

```

12.12.3 RestrictedRankOneDiagram

RestrictedRankOneDiagram::usage=

"RestrictedRankOneDiagram[r,disks,arches,root] computes a Helmicnk diagram illustrating the restricted rank one root system with respect to root, defined by an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRankOneDiagram[r,theta,root] computes a Helmicnk diagram illustrating the restricted rank one root system with respect to root, defined by an involution theta over root system r."

```
RestrictedRankOneDiagram[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] :=
  RestrictedRankOneDiagram[RootBase[r], List[argdisks], arches, root];

RestrictedRankOneDiagram[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ] := Module[
  {disks, rbasis, labelset, lb, i},

  disks = List[argdisks];
  labelset = {};
  rbasis = RestrictedRankOneBasis[r, disks, arches, root];

  For[i = 1, i <= Length[r], i++,
    Which[
      MemberQ[disks, i], lb = "[Alpha]" <> ToString[i];
      ,
      MemberQ[rbasis, r[[i]]], lb = "[Lambda]";
      ,
      True, lb = "X";
    ];

    labelset = Join[labelset, {lb}];
  ];

  HelminckDiagram[r, disks, arches, labelset]
  ];

RestrictedRankOneDiagram[r_?RootInputQ, theta_?MatrixQ,
  root_?VectorQ] :=
  RestrictedRankOneDiagram[RootBase[r], theta, root];

RestrictedRankOneDiagram[r_?RootBasisQ, theta_?MatrixQ,
  root_?VectorQ] := Module[
  {rbasis, labelset, lb, i, disks, arches},

  labelset = {};
  rbasis = RestrictedRankOneBasis[r, theta, root];
  disks = DiskList[r, theta];
  arches = ArchesListInvolution[r, disks, theta];

  For[i = 1, i <= Length[r], i++,
    Which[
      MemberQ[disks, i], lb = "[Alpha]" <> ToString[i];
      ,
      MemberQ[rbasis, r[[i]]], lb = "[Lambda]";
      ,
    ],
```

```

    True, lb = "X";
  ];

  labelset = Join[labelset, {lb}];
];

HelminckDiagram[r, disks, arches, labelset]
];

RestrictedRankOneDiagram[a_...] :=
  InvalidArg["RestrictedRankOneDiagram", a];

```


12.12.4 RestrictedRootDiagram

RestrictedRootDiagram::usage=

”RestrictedRootDiagram[r,disks,arches] draws a Helminck diagram and Dynkin diagram for the restricted root system determined over an involution over root system r with fixed roots disks and diagram automorphism arches.

RestrictedRootDiagram[r,theta] draws a Helminck diagram and Dynkin diagram for the restricted root system determined by an involution theta over root system r.”;

```
(* Visualize the reduced rank root system *)

RestrictedRootDiagram[d_?RootBasisQ, theta_?MatrixQ] := Module[
  {rbasis, i, labelset, lb, j, r2, h1, h2, disks, arches},

  j = 1;
  labelset = {};
  rbasis = RestrictedRootBasis[d, theta];
  disks = DiskList[d, theta];
  arches = ArchesListInvolution[d, disks, theta];

  For[i = 1, i <= Length[d], i++,
    Which[
      MemberQ[disks, i], lb = "0";
      ,
      True, lb = "[Lambda]" <> ToString[j]; j++;
    ];

    labelset = Join[labelset, {lb}];
  ];

  r2 = RestrictedRootSystemType[d, theta];

  h1 = HelminckDiagram[d, disks, arches, labelset];
  h2 = DynkinDiagram[r2];

  GraphicsGrid[{{h1}, {h2}}]
];

RestrictedRootDiagram[r_?RootInputQ, theta_?MatrixQ] :=
  RestrictedRootDiagram[RootBase[r], theta];

(* Visualize the reduced rank root system *)

RestrictedRootDiagram[d_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ] := Module[
  {rbasis, i, labelset, lb, j, r2, h1, h2, disks},

  j = 1;
  disks = List[argdisks];
  labelset = {};
  rbasis = RestrictedRootBasis[d, disks, arches];

  For[i = 1, i <= Length[d], i++,
    Which[
      MemberQ[disks, i], lb = "0";
      ,
      True, lb = "[Lambda]" <> ToString[j]; j++;
    ];
  ];
];
```

```

];

labelset = Join[labelset, {lb}];
];

r2 = RestrictedRootSystemType[d, disks, arches];

h1 = HelminckDiagram[d, disks, arches, labelset];
h2 = DynkinDiagram[r2];

GraphicsGrid[{{h1}, {h2}}]
];

RestrictedRootDiagram[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ] :=
  RestrictedRootDiagram[RootBase[r], List[argdisks], arches];

RestrictedRootDiagram[a___] := InvalidArg["RestrictedRootDiagram", a];

```

12.13 Local Symmetric Spaces Package (Internal)

12.13.1 EigenspaceProject (Additional Definitions)

EigenspaceProject::usage=

”EigenspaceProject[r,theta,root,E] projects root into some root in the E-eigenspace of the root automorphism theta over root system r.

(S-LOSS) EigenspaceProject[r,disks,arches,root,E] projects root into some root in the E-eigenspace of the root automorphism described by fixed roots disks and diagram automorphism arches over root system r.”;

```
EigenspaceProject[r_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ, espace_] :=
  EigenspaceProject[RootBase[r],
    RootInvolution[RootBase[r], List[argdisks], arches], root, espace];

EigenspaceProject[r_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, root_?VectorQ, espace_] :=
  EigenspaceProject[r, RootInvolution[r, List[argdisks], arches],
    root, espace];
```

12.13.2 FixedRootQ (Additional Definitions)

FixedRootQ::usage=

”FixedRootQ[r,theta,root] returns True if root is fixed by the involution theta defined over the root system r.

(S-LOSS) FixedRootQ[r,disks,arches,root] returns True if root is fixed by the involution defined over the root system r with fixed roots disks and diagram automorphism arches.”;

```
FixedRootQ[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ,
  root_?VectorQ] := Module[
  {k, basis, theta, rk, disks},

  disks = List[argdisks];
  basis = RootBase[r];
  k = BasisCoefficients[basis, root];
  theta = RootInvolution[r, disks, arches];

  rk = k.Transpose[theta].basis;

  Return[SameQ[root, rk]];
];

FixedRootQ[basis_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ,
  root_?VectorQ] := Module[
  {k, rk, theta, disks},

  disks = List[argdisks];
  theta = RootInvolution[basis, disks, arches];

  k = BasisCoefficients[basis, root];

  rk = k.Transpose[theta].basis;

  Return[SameQ[root, rk]];
];
```

12.13.3 FixedRoots (Additional Definitions)

FixedRoots::usage=

”FixedRoots[r,theta] returns the set of all roots fixed by the involution theta defined over the root system r.

(S-LOSS) FixedRoots[r,disks,arches] returns the set of all roots fixed by the involution defined over the root system r with fixed roots disks and diagram automorphism arches.”;

```
FixedRoots[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
Module[
  {i, roots, froots, theta, disks},

  disks = List[argdisks];
  theta = RootInvolution[r, disks, arches];

  roots = RootSystemFromBasis[RootBase[r]];
  froots = {};

  For[i = 1, i <= Length[roots], i++,
    If[FixedRootQ[r, theta, roots[[i]]],
      froots = Join[fruits, {roots[[i]]}];
    ];
  ];

  Return[fruits];
];

FixedRoots[basis_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
Module[
  {i, roots, froots, theta, disks},

  disks = List[argdisks];
  theta = RootInvolution[basis, disks, arches];

  roots = RootSystemFromBasis[basis];
  froots = {};

  For[i = 1, i <= Length[roots], i++,
    If[FixedRootQ[basis, theta, roots[[i]]],
      froots = Join[fruits, {roots[[i]]}];
    ];
  ];

  Return[fruits];
];
```

12.13.4 FixedBasis (Additional Definitions)

FixedBasis::usage=

”FixedBasis[r,theta] computes the set of all basis roots fixed by an involution theta defined over root system r.

(S-LOSS) FixedBasis[r,disks,arches] computes the set of all basis roots fixed by an involution defined over root system r with fixed roots disks and diagram automorphism arches.”;

```
FixedBasis[r_?RootInputQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  Intersection[RootBase[r], FixedRoots[r, List[argdisks], arches]];
```

```
FixedBasis[r_?RootBasisQ, {argdisks___?IntegerQ}, arches_?ListQ] :=
  Intersection[r, FixedRoots[r, List[argdisks], arches]];
```

12.13.5 gInvolutionDiagram (Additional Definitions)

gInvolutionDiagram::usage=

”gInvolutionDiagram[r,theta,cvals] extends the Helminck diagram with the values c and k necessary to recover the structure constants of an involution on the Lie algebra. Each basis root is labelled with c,k. The minimal polynomial of the corresponding structure constant is $1x^2 + kx + c$. r is the root system, theta the root system automorphism, and cvals is the list of structure constants.

(S-LOSS) gInvolutionDiagram[r,disks,arches,cvals] extends the Helminck diagram with the values c and k necessary to recover the structure constants of an involution on the Lie algebra. Each basis root is labelled with c,k. The minimal polynomial of the corresponding structure constant is $1x^2 + kx + c$. r is the root system, disks labels the fixed roots, arches represents the diagram automorphism, and cvals is the list of structure constants.”;

```
gInvolutionDiagram[d_?RootInputQ, {argdisks___?IntegerQ},
  arches_?ListQ, cvals_?ListQ] := Module[
  {cv, disks},

  disks = List[argdisks];
  cv = cvMinimalPolynomialList[d, cvals];

  HelminckDiagram[d, disks, arches, cv]
];

gInvolutionDiagram[d_?RootBasisQ, {argdisks___?IntegerQ},
  arches_?ListQ, cvals_?ListQ] := Module[
  {cv, disks},

  disks = List[argdisks];
  cv = cvMinimalPolynomialList[d, cvals];

  HelminckDiagram[d, disks, arches, cv]
];
```

12.13.6 gInvolutionListFormToMatrix (Additional Definitions)

gInvolutionListFormToMatrix::usage=

”gInvolutionListFormToMatrix[r,roots,theta,list] takes a list of the form ROOT,CCONST denoting an involution on the Lie algebra, the list of all roots, and the root system (r) involution theta and returns a matrix for the involution over the Lie algebra with respect to the ordered basis of root vectors (arranged with respect to the roots). The argument roots is optional. If omitted, the matrix will be set with respect to the ordering of the roots resulting from the procedure RootSystem with r as calling argument.

(PI-LOSS) gInvolutionListFormToMatrix[r,roots,disks,arches,list] takes a list of the form ROOT,CCONST denoting an involution on the Lie algebra, the list of all roots, and the root system (r) involution theta (defined via fixed roots disks and diagram automorphism arches) and returns a matrix for the involution over the Lie algebra with respect to the ordered basis of root vectors (arranged with respect to the roots). The argument roots is optional. If omitted, the matrix will be set with respect to the ordering of the roots resulting from the procedure RootSystem with r as calling argument.”;

```
gInvolutionListFormToMatrix[r_?RootInputQ,
  inroots_: {}, {argdisks___?IntegerQ}, arches_?ListQ,
  rlist_?ListQ] :=
  gInvolutionListFormToMatrix[RootBase[r], inroots,
    RootInvolution[RootBase[r], List[argdisks], arches], rlist];

gInvolutionListFormToMatrix[d_?RootBasisQ,
  inroots_: {}, {argdisks___?IntegerQ}, arches_?ListQ,
  rlist_?ListQ] :=
  gInvolutionListFormToMatrix[d, inroots,
    RootInvolution[d, List[argdisks], arches], rlist];

gMergeInvolutions[cbasis_?ListQ, {argdisks___?IntegerQ},
  arches_?ListQ, invol_?ListQ, nvals_?ListQ] :=
  gMergeInvolutions[cbasis,
    RootInvolution[cbasis, List[argdisks], arches], invol, nvals];
```


12.13.7 DiagramInvolutionSimple

DiagramInvolutionSimple::usage=

”DiagramInvolution[r,offset] gives a list of root pairs “arches” in an irreducible root system r which are swapped by the diagram automorphism of order 2. The optional argument offset shifts the indices by the specified number of units.”;

```
DiagramInvolutionSimple::noInv =
  "Warning. No diagram involution for a root system of type '1' '2'. \
  Returning identity map {}.";

DiagramInvolutionSimple[r_?RootInputQ, offset_: 0] := Module[
  {rin, type, dim, i, out},

  rin = Flatten[RootInput[r]]; (*
  Flatten because input may either be user input (which could have 1 \
  or 2 curly braces), or internal input (usually 1 brace) *)

  type = rin[[1]];
  dim = rin[[2]];

  out = "";

  Switch[type,
    "A",
    If[EvenQ[dim],
      out = Table[{i, dim + 1 - i}, {i, 1, (dim/2)}];
    ,
    out = Table[{i, dim + 1 - i}, {i, 1, (dim - 1)/2}];
    ];
  ,
  "D",
  out = {{dim - 1, dim}};
  ,
  "E",
  If[dim == 6,
    out = {{1, 6}, {3, 5}};
  ,
    (*Message[diagramInvolutionSimple::noInv,type,dim];*)

    out = {};
  ];
  ,
  -,
  (*Message[diagramInvolutionSimple::noInv,type,dim];*)
  out = {};
  ];

  Return[out + offset];
]

DiagramInvolutionSimple[a___] :=
  InvalidArg["DiagramInvolutionSimple", a];
```

Bibliography

- [1] Humphreys, James E. *Introduction to Lie Algebras and Representation Theory*. Springer, New York. 1972
- [2] Von Zur Gathen, Joachim and Jürgen Gerhard. *Modern Computer Algebra, Second Edition*. Cambridge, New York. 2003.
- [3] Cox, David, John Little and Donal O'Shea. *Ideals, Varieties, and Algorithms, Third Edition*. Springer, New York. 2007.
- [4] Steinberg, R. *Lectures on Chevalley Groups* Mimeographed lecture notes. Yale University Mathematics Department. New Haven. 1968.
- [5] Helminck, Aloysius G. *Algebraic Groups With a Commuting Pair of Involutions and Semisimple Symmetric Spaces* Academic Press 21 - 91. 1988.
- [6] DeGraaf, W.A. *Lie Algebras: Theory and Algorithms*. Elsevier Science B.V., Amsterdam, The Netherlands. 2000.
- [7] Stembridge, John. coxeter and weyl: <http://www.math.lsa.umich.edu/~jrs/maple.html>
- [8] Rowland, Todd and Weisstein, Eric W. "Matrix Minimal Polynomial." From MathWorld—A Wolfram Web Resource: <http://mathworld.wolfram.com/MatrixMinimalPolynomial.html>
- [9] LiE: <http://young.sp2mi.univ-poitiers.fr/~marc/LiE/>
- [10] Atlas of Lie Groups and Representations: <http://www.liegroups.org/>
- [11] Klein, Sebastian *Reconstructing the Geometric Structure of a Riemannian Symmetric Space From its Satake Diagram* Geom. Dedicata. 138, 25 - 50. 2009.
- [12] Knapp, A. *Lie Groups, Beyond an Introduction, Second Edition*. Boston, MA. 2002.
- [13] Vavilov, Nikolai and Eugene Plotkin *Chevalley Groups over Commutative Rings. Elementary Calculations* 1991.
- [14] Gilkey, P. and G. Seitz *Some Representations of Exceptional Lie Algebras* Geom. Dedic. 25 no. 1-3, 407 - 416. 1988.
- [15] Bourbaki, N. *Groupes et Algebres de Lie* Hermann, Paris. 1968

Appendices

APPENDIX A: Dynkin Diagrams and Cartan Matrices

Figure A.1: Cartan Matrix for Type A

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

Figure A.2: Cartan Matrix for Type B

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

Figure A.3: Cartan Matrix for Type C

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

Figure A.4: Cartan Matrix for Type D

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

Figure A.5: Cartan Matrix for Type E_6

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

Figure A.6: Cartan Matrix for Type E_7

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

Figure A.7: Cartan Matrix for Type E_8

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

Figure A.8: Cartan Matrix for Type F_4

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

Figure A.9: Cartan Matrix for Type G_2

$$\begin{pmatrix} 2 & -1 \\ -3 & 2 \end{pmatrix}$$

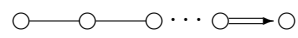
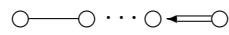
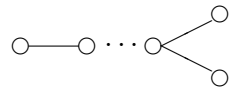
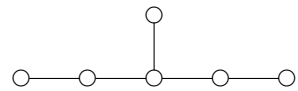
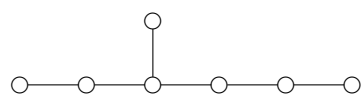
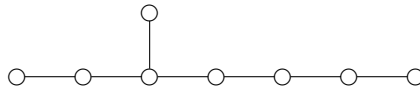
Figure A.10: Dynkin Diagram for A_n Figure A.11: Dynkin Diagram for B_n Figure A.12: Dynkin Diagram for C_n Figure A.13: Dynkin Diagram for D_n Figure A.14: Dynkin Diagram for E_6 Figure A.15: Dynkin Diagram for E_7 

Figure A.16: Dynkin Diagram for E_8 Figure A.17: Dynkin Diagram for F_4 Figure A.18: Dynkin Diagram for G_2 

APPENDIX B: Helminck Diagrams For Involutional Automorphisms with a Maximal (-1)-Eigenspace

Table B.1: Helminck Diagrams For Involutional Automorphisms with a Maximal (-1)-Eigenspace

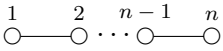
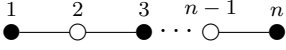
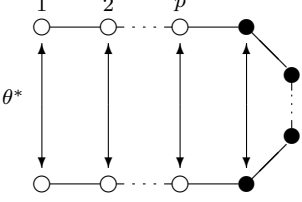
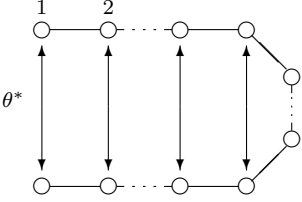
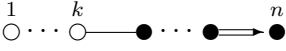
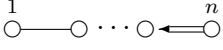
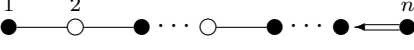
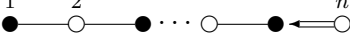
Name	θ	θ_Δ State
AI		Involution
AII		Involution
AIIIa		Involution
AIIIb		Involution
BI		Involution
CI		Involution
CIIa		Involution
CIIb		Involution
Continued on next page		

Table B.1 – continued from previous page

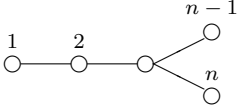
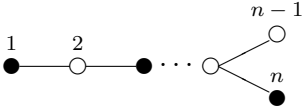
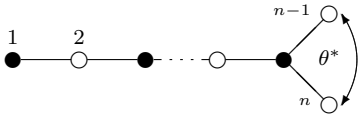
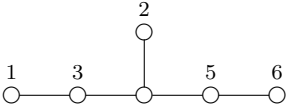
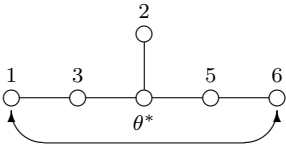
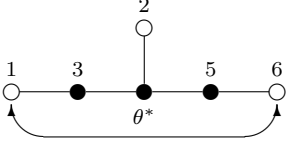
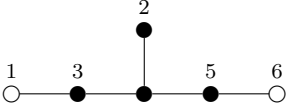
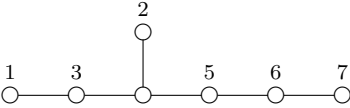
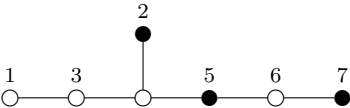
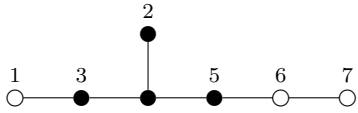
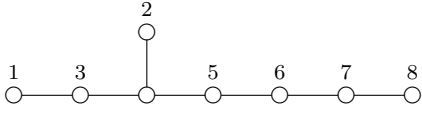
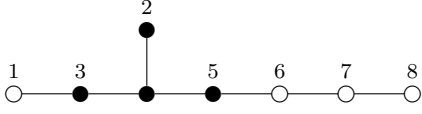
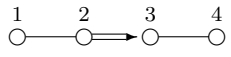
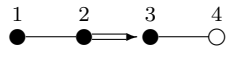
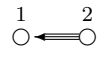
Name	θ	θ_Δ State
DI		Involution
DIIIa		Involution
DIIIb		36 Solutions * ₁
EI		Involution
EII		Involution
EIII		24 Solutions * ₂
EIV		Involution
EV		Involution
EVI		Involution
Continued on next page		

Table B.1 – continued from previous page

Name	θ	θ_Δ State
EVII		Involution
EVIII		Involution
EIX		Involution
FI		Involution
FII		Involution
G		Involution

*₁ In the DIIIb case, for $n = 7$ there were 36 possible correction vectors. The table on the following page gives the number of correction vectors for several values of n , and one “nice” vector.

*₂ In the EIII case there were 24 possible correction vectors. The coordinates of one such vector relative to the basis of \mathfrak{t} are

$$\begin{aligned}
 x_1 &= \frac{1}{3}(1 - 3\sqrt{5}) \\
 x_2 &= -\sqrt{5} \\
 x_3 &= \frac{1}{6}(1 - 9\sqrt{5}) \\
 x_4 &= -1 - 2\sqrt{5} \\
 x_5 &= \frac{1}{6}(-7 - 9\sqrt{5}) \\
 x_6 &= \frac{1}{3}(-1 - 3\sqrt{5})
 \end{aligned}$$

Table B.2: Correction Vectors in the DIIIb Case

n	No. Solutions	Coordinates of Correction Vector	CPU sec
5	12	$x(1) \rightarrow \frac{1}{2}(-1 - \sqrt{5})$ $x(2) \rightarrow -\sqrt{5}$ $x(3) \rightarrow \frac{1}{2}(-1 - 3\sqrt{5})$ $x(4) \rightarrow -\sqrt{5}$ $x(5) \rightarrow -\sqrt{5}$	1.2
7	36	$x(1) \rightarrow \frac{1}{2}(-1 - \sqrt{5})$ $x(2) \rightarrow -\sqrt{5}$ $x(3) \rightarrow \frac{1}{2}(-1 - 3\sqrt{5})$ $x(4) \rightarrow -2\sqrt{5}$ $x(5) \rightarrow \frac{1}{2}(-1 - 5\sqrt{5})$ $x(6) \rightarrow -\frac{3\sqrt{5}}{2}$ $x(7) \rightarrow -\frac{3\sqrt{5}}{2}$	135.42

APPENDIX C: Helminck Diagrams for Involutions of Restricted Rank One

Table C.1: Helminck Diagrams for Involutions of Restricted Rank One

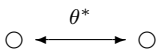


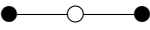





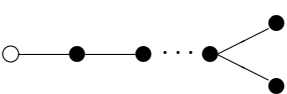
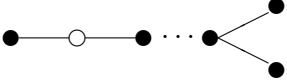
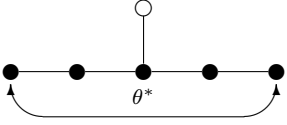
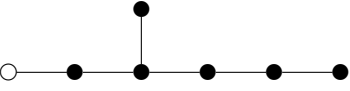
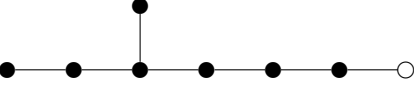
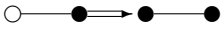
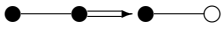


Type	Name	Diagram	1-Consistent
1	$A_1 \times A_1$		+
2	A_1		+
3	A_2		-
4	A_3		+
5	A_n		+
6	$B_{n \geq 2}$		+
7	$B_{n \geq 2}$		-
8	$C_{n \geq 3}$		-
9	$C_{n \geq 3}$		+
10	$D_{n \geq 4}$		+
Continued on next page			

Table C.1 – continued from previous page

Type	Name	Diagram	1-Consistent
11	$D_{n \geq 4}$		—
12	E_6		—
13	E_7		—
14	E_8		—
15	F_4		—
16	F_4		+
17	G_2		—
18	G_2		—

In the following tables one correction vector is provided for each of the Helminck diagrams which are not 1-consistent. Given are the coordinates for $H \in \mathfrak{t}$ with respect to the basis for \mathfrak{t} : $\{H_{\alpha_1}, H_{\alpha_2}, \dots, H_{\alpha_n}\}$.

$$H = x_1 H_{\alpha_1} + \dots + x_n H_{\alpha_n}$$

Table C.2: Correction Vector for Type 3

$$\begin{vmatrix} x_1 & \frac{\sqrt{5}}{3} \\ x_2 & \frac{1}{6}(3 + \sqrt{5}) \end{vmatrix}$$

Table C.3: Correction Vector for Type 7, n = 4

$$\begin{vmatrix} x_1 & \frac{1}{14} \left(-7 - (5\sqrt{2} + \sqrt{15}) \sqrt{13 - 2\sqrt{30}} \right) \\ x_2 & -\frac{1}{7} (5\sqrt{2} + \sqrt{15}) \sqrt{13 - 2\sqrt{30}} \\ x_3 & -\frac{1}{7} (5\sqrt{2} + \sqrt{15}) \sqrt{13 - 2\sqrt{30}} \\ x_4 & -\sqrt{\frac{1}{2} (13 - 2\sqrt{30})} \end{vmatrix}$$

Table C.4: Correction Vector for Type 8, n = 3

$$\begin{vmatrix} x_1 & -\frac{1}{4} \sqrt{\frac{5}{7} (9 - \sqrt{65})} \\ x_2 & -\frac{1}{2} (2\sqrt{5} + \sqrt{13}) \sqrt{\frac{1}{7} (9 - \sqrt{65})} \\ x_3 & -\frac{1}{4} \sqrt{\frac{5}{7} (9 - \sqrt{65})} \end{vmatrix}$$

Table C.5: Correction Vector for Type 11, n = 6

$$\begin{vmatrix} x_1 & \frac{1}{2}(1 - \sqrt{5}) \\ x_2 & -\sqrt{5} \\ x_3 & -1 - \sqrt{5} \\ x_4 & -1 - \sqrt{5} \\ x_5 & -\frac{\sqrt{5}}{2} \\ x_6 & -\frac{\sqrt{5}}{2} \end{vmatrix}$$

Table C.6: Correction Vector for Type 12

x_1	$\frac{1}{6} (1 - 3\sqrt{5})$
x_2	$-\sqrt{5}$
x_3	$-\frac{2}{3} - \sqrt{5}$
x_4	$\frac{1}{2} (-1 - 3\sqrt{5})$
x_5	$\frac{1}{3} (-4 - 3\sqrt{5})$
x_6	$\frac{1}{6} (-7 - 3\sqrt{5})$

Table C.7: Correction Vector for Type 13

x_1	$-\sqrt{5}$
x_2	$\frac{1}{2} (1 - 2\sqrt{5})$
x_3	$\frac{1}{2} (-1 - 3\sqrt{5})$
x_4	$-2\sqrt{5}$
x_5	$\frac{1}{2} (-2 - 3\sqrt{5})$
x_6	$-1 - \sqrt{5}$
x_7	$-\frac{\sqrt{5}}{2}$

Table C.8: Correction Vector for Type 14

x_1	$-7 - 5\sqrt{5}$
x_2	$-\frac{3}{2} (7 + 5\sqrt{5})$
x_3	$-5 (3 + 2\sqrt{5})$
x_4	$-22 - 15\sqrt{5}$
x_5	$-\frac{5}{2} (7 + 5\sqrt{5})$
x_6	$2 (-6 - 5\sqrt{5})$
x_7	$\frac{1}{2} (-11 - 15\sqrt{5})$
x_8	$-5\sqrt{5}$

Table C.9: Correction Vector for Type 15

$$\begin{array}{c|c} x_1 & \frac{1}{2}(1 + \sqrt{5}) \\ x_2 & 1 \\ x_3 & -1 \\ x_4 & 1 \end{array}$$

Table C.10: Correction Vector for Type 17

$$\begin{array}{c|c} x_1 & \frac{1}{22}(3i\sqrt{143} + i\sqrt{451}) \\ x_2 & i\sqrt{\frac{13}{11}} \end{array}$$

Table C.11: Correction Vector for Type 18

$$\begin{array}{c|c} x_1 & 1 \\ x_2 & \frac{1}{2}(1 + \sqrt{5}) \end{array}$$