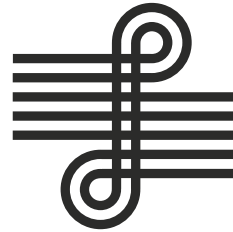


University of Music
Hochschule
für Musik
Karlsruhe



Music Processing Suite: A Software System for Context-based Symbolic Music Representation, Visualization, Transformation, Analysis and Generation

Dissertation

submitted by

David M. Hofmann

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Institute for Musicology and Music Informatics
University of Music Karlsruhe

First Supervisor:	Prof. Dr. Thomas A. Troge
Second Supervisor:	Prof. Dr. Marlon Schumacher
External Supervisor:	Prof. Dr. Meinard Müller
Date of Submission:	August 9, 2018

Abstract

This dissertation documents the design and development of a software system for symbolic music processing. *Music Processing Suite* is built on the foundation of new music representation models, which allow specifying, analyzing and processing compositions in a variety of musical dimensions. Instead of modeling music in terms of sequences of notes and rests, the proposed models provide individual layers for various musical aspects including instruments, meter, tempo, rhythms, multiple pitch specification systems, hierarchical harmonic contexts and lyrics. Composers can extend the model with arbitrary custom musical dimensions, resulting in music representation models offering more possibilities than traditional scores do. Two different model presentations, namely a time-based layer representation and a tree-based representation, are introduced and corresponding transformation algorithms are proposed. Furthermore, a corresponding language for music composition in multiple musical dimensions is presented. Apart from basic music notation, the language also offers possibilities to develop and form musical material and to utilize control structures for algorithmic composition. The language is designed with the objective of minimizing redundancy by means of reusing already specified musical information. Further applications of the music model presented in this work are music format transformations, context-dependent musical search functionality and a framework for music analysis. Finally, an evolutionary algorithm approach for generating music based on statistical criteria is proposed.

Acknowledgements

This dissertation project has been a life-enhancing experience for me and would not have been possible without the support of the following people, whose contributions I greatly appreciate.

First and foremost, I would like to acknowledge my gratefulness to Prof. Dr. Thomas Troge for continuously supporting me throughout my studies at the University of Music Karlsruhe and for numerous inspiring conversations on music, informatics and artificial intelligence.

I would sincerely like to thank Prof. Dr. Marlon Schumacher for the excellent collaboration during my time as research assistant at the University of Music and for the useful suggestions relating to this dissertation.

I am very grateful to Prof. Dr. Meinard Müller from the Audio Laboratories Erlangen for agreeing to review this dissertation in the role of an external advisor and for providing valuable feedback. I would also like to thank his colleague Frank Zalkow for reviewing selected chapters of this work.

Thank you very much to my colleague Johannes Utzig, who provided valuable insights on software development topics and supported me to get the automated software build process running. I would also like to thank my friend Dr. Thomas Damian for proofreading this dissertation.

I gratefully acknowledge a LGFG scholarship granted by the state of Baden-Württemberg, with which the majority of this research project was funded.

Finally, I would like to express my deepest gratitude to my family and my friends, who always supported me along the way. I would like to extend my gratefulness to my loving and supportive girlfriend Jennifer, who also proofread this dissertation, and her parents Petra and Mark for their constant efforts to support us whenever possible.

Contents

List of Figures	vi
List of Tables	xii
Acronyms	xv
1 Introduction	1
I Theory and Models	5
2 State of the Art	6
2.1 Music Representation	6
2.1.1 Musical Instrument Digital Interface	6
2.1.2 MuseData	10
2.1.3 Humdrum	14
2.1.4 abc	17
2.1.5 GUIDO	18
2.1.6 LilyPond	18
2.1.7 MusicXML	19
2.1.8 music21	24
2.1.9 SARAH	24
2.2 Computers and Creativity: An Overview	25
2.2.1 Computers as Tools in Creative Processes	26
2.2.2 Computer Models of Creativity	26
2.2.3 Constraints and Creativity	28
2.3 Algorithmic Composition	29
2.3.1 Historical Context	29
2.3.2 Stochastic Processes	31
2.3.3 Markov Models	32
2.3.4 Generative Grammars	36

2.3.5	Transition Networks	40
2.3.6	Artificial Neural Networks	43
2.3.7	Evolutionary Algorithms	47
2.4	Summary	57
3	Context Layer Composition Model	59
3.1	Motivation	60
3.2	Introductory Example	62
3.3	Model Structure	62
3.3.1	Instrumentation Context	64
3.3.2	Metric Contexts	64
3.3.3	Harmonic Contexts	64
3.3.4	Rhythmic Contexts	64
3.3.5	Pitch Contexts	65
3.3.6	Loudness Contexts	65
3.3.7	Lyrics	65
3.3.8	Musical Labels	65
3.3.9	Custom Contexts	65
3.4	Time Model	66
3.5	Parallel Streams	67
3.6	Stream Sequencers and Stream Events	67
3.7	Key Benefits and Versatility of the Model	71
3.8	Summary	72
4	Context Tree Model and Composition Language	73
4.1	Motivation	73
4.2	Overview	74
4.3	Introductory Example	74
4.4	Key Concepts	77
4.4.1	Hierarchical Structures	77
4.4.2	Inheritance	78
4.4.3	Polymorphism	82
4.4.4	Auto Expansion	82
4.4.5	Modularization using Fragments	84
4.5	Contexts	86
4.5.1	Rhythms	86
4.5.2	Meter	89
4.5.3	Tempo	91
4.5.4	Instruments	91

4.5.5	Pitches	94
4.5.6	Scales	97
4.5.7	Loudness	100
4.5.8	Harmonic Contexts	101
4.5.9	Lyrics	107
4.5.10	Custom Contexts	108
4.6	Context Modifiers	109
4.6.1	Rhythmic Modifiers	109
4.6.2	Pitch Modifiers	115
4.6.3	Harmonic Modifiers	119
4.7	Context Generators	120
4.7.1	Chord Generators	120
4.7.2	Arpeggio Generators	123
4.8	Control Structures	126
4.8.1	Parallelizations	126
4.8.2	Repetitions	127
4.8.3	Conditions	129
4.8.4	Iterations	131
4.8.5	Sequences	133
4.8.6	While-Loops	135
4.8.7	Switches	136
4.9	Expressions	137
4.9.1	Literals	138
4.9.2	Operators	138
4.9.3	Type Conversions	139
4.9.4	Function Calls	140
4.10	Implementation Details	141
4.10.1	Composition Domain Model	141
4.10.2	Domain-Specific Composition Language	144
4.11	Summary	144

II System Applications 146

5 Model Transformations 147

5.1	Transformation Infrastructure Overview	147
5.2	Transforming Context Tree Models to Context Layer Models	148
5.3	Transforming Context Layer Models to Score Representations	153
5.3.1	LilyPond Compiler	153

5.4	Transforming Context Layer Models to SuperCollider	155
5.4.1	Immediate Compilation and Execution	156
5.5	Transforming MIDI Files to Context Layer Models	158
5.6	Transforming MusicXML Files to Context Layer Models	158
5.7	Deriving and Compressing Context Tree Composition Models	159
5.7.1	Related Work	160
5.7.2	Compression Algorithm	162
5.7.3	Future Work	167
5.8	Graphical User Interface	170
5.9	Summary	170
6	Context-based Corpus Search	172
6.1	Motivation	172
6.2	Formulating Musical Search Queries	173
6.3	Search Methodology	174
6.4	Search Query Context Layer Models	175
6.5	Search Algorithm	175
6.6	Search Result Presentation	177
6.7	Results	177
6.8	Conclusion	184
7	Music Analysis	185
7.1	Motivation	185
7.2	Analysis Scopes	186
7.3	Rhythmic Analysis	187
7.3.1	Note Duration Analysis	187
7.3.2	Note Density Analysis	188
7.3.3	Beat Analysis	189
7.3.4	Combined Note Duration and Beat Analysis	190
7.4	Pitch Analysis	191
7.4.1	Piano Roll Representations	191
7.4.2	Pitch Distributions	192
7.4.3	Interval Analysis	193
7.4.4	Dissonance Analysis	196
7.4.5	Harmonic Analysis	198
7.5	Progression Analysis	202
7.5.1	Harmonic Progression Graphs	202
7.5.2	Lyric Progression Graphs	206
7.6	Comparative Analysis of Large Corpora	206

7.6.1	Comparing Composition Collections	206
7.6.2	Analyzing Large Corpora	208
7.7	Conclusion	212
III	Automated Composition	213
8	Evolutionary Composition Algorithm	214
8.1	Motivation	214
8.2	Composition vs. Improvisation	215
8.3	Composition Algorithm	216
8.3.1	Overview	216
8.3.2	Fitness Function	217
8.3.3	Multi-objective Optimization	220
8.3.4	Crossover Operators	222
8.3.5	Mutation Operators	223
8.3.6	Parameters	225
8.3.7	Genetic Programming Specifics	226
8.3.8	Example Evolutionary Algorithm Run	227
8.4	Applications	230
8.4.1	Composition Crossover and Variations	230
8.4.2	Style Imitations	236
8.4.3	Generating Compositions with Predefined Structures	239
8.4.4	Generating Compositions with Multiple Sections	242
8.5	Summary	254
IV	Conclusions and Appendices	255
9	Conclusions	256
	Bibliography	258
A	Contents of the Accompanying CD	273
B	Code Examples	274
B.1	Composition Language Grammar	274
B.2	MusicXML Code Example	284

List of Figures

1.1	Overview of Music Processing Suite components	2
1.2	Screenshot of the MPS Application	3
2.1	Score and piano roll representations of MIDI events in J. S. Bach, <i>Sinfonia 1, BWV 787</i> , mm. 1–2	8
2.2	W. A. Mozart, <i>Piano Sonata No. 11 in A major, K. 331/300i</i> , Mv. 3 ("Alla Turca"), mm. 98–102	12
2.3	J. S. Bach, <i>The Art of Fugue, BWV 1080</i> , subject	15
2.4	Score illustrating the harmonic progression specified in Listing 2.4	16
2.5	The Beatles, <i>Yesterday</i> , m. 1, vocal part	20
2.6	Logical structure of the MusicXML score representation shown in Listing 2.9	23
2.7	Harmonic progression graph and corresponding Markov model of <i>Yes- terday</i> by the Beatles	34
2.8	Transition network for simple natural language expressions	40
2.9	Illustrations of various artificial neural network topologies	45
2.10	Flow chart illustrating the basic structure of an evolutionary algorithm	49
2.11	Roulette wheel selection in evolutionary algorithms	49
2.12	Genetic algorithm one point crossover	50
2.13	Genetic algorithm bit flip mutation	50
2.14	Illustration of subtree crossover in genetic programming	55
3.1	Ludwig van Beethoven, <i>Symphony No. 5 in C Minor, Op. 67</i> , Mv. I, motif	60
3.2	Legend illustrating the color scheme of model elements presented in this dissertation	62
3.3	Score and context layer model of <i>Hey Jude</i> by the Beatles, mm. 1–4	63
3.4	Score and context layer model of Beethoven's <i>Piano Sonata No. 14 in C# minor, Op. 27, No. 2</i> , Mv. I, mm. 1–4	68
3.5	Segmentation of a context layer model into stream events	70

4.1	Score and context tree model of Beethoven's <i>Symphony No. 5 in C Minor</i> , <i>Op. 67</i> , Mv. I, mm. 1–4, violin part	75
4.2	Simple class hierarchy	78
4.3	Queen, <i>Bohemian Rhapsody</i> , mm. 1–4	79
4.4	Context tree model of Queen's <i>Bohemian Rhapsody</i> , mm. 1–2	80
4.5	Context tree model of Queen's <i>Bohemian Rhapsody</i> , mm. 1–4	81
4.6	Score and context tree model of Beethoven's <i>Symphony No. 9</i> , Mv. IV, mm. 543–550, soprano part	83
4.7	Score and context tree model of the English horn theme from Antonin Dvorak's <i>Symphony No. 9 in E minor</i> ("From the New World"), <i>Op. 95</i> , <i>B. 178</i> , Mv. II	84
4.8	Redundancy-optimized context tree model of the English horn theme from Antonin Dvorak's <i>Symphony No. 9 in E minor</i> ("From the New World"), <i>Op. 95</i> , <i>B. 178</i> , Mv. II	85
4.9	Score and context tree model of the the first measures of Vivaldi's <i>Concerto No. 1 in E major</i> , <i>Op. 8</i> , <i>RV 269</i>	89
4.10	Score and context tree model of a rhythm in two different metric contexts	90
4.11	Score and context tree model of an excerpt from <i>Boléro</i> by Maurice Ravel	92
4.12	Score and context tree model of W. A. Mozart's <i>Piano Sonata No. 16 in C major</i> , <i>K. 545</i> , mm. 1–2, right hand part	96
4.13	Score and context tree model of Bedřich Smetana's <i>Moldau</i> theme from <i>Vltava</i> , <i>JB 1:112/2</i>	98
4.14	Score and context tree model of the opening oboe theme from W. A. Mozart's <i>Flute and Harp Concerto in C major</i> , <i>K. 299/297c</i>	102
4.15	Schematic context tree model of W. A. Mozart's <i>Symphony No. 40 in G minor</i> , <i>K. 550</i> , Mv. I demonstrating hierarchically arranged global and local keys	103
4.16	Score and context tree model demonstrating a harmonic progression in the context of a global key	104
4.17	Context tree model demonstrating the definition of a harmonic progression and a corresponding harmonic rhythm	106
4.18	Score and context tree model of the first measures of <i>Hey Jude</i> by the Beatles demonstrating the specification of lyrics	108
4.19	Context tree model introducing a new custom context type to describe moods of individual sections	109

4.20	Context tree model of an excerpt from J. S. Bach's <i>The Art of Fugue</i> , <i>BWV 1080, Contrapunctus VII</i> , in which diminution and inversion is applied	111
4.21	Score of an excerpt from J. S. Bach's <i>The Art of Fugue, BWV 1080</i> , <i>Contrapunctus VII</i>	112
4.22	Context tree model of Steve Reich's <i>Clapping Music</i> , in which itera- tive rhythmic displacements are utilized	114
4.23	Score of Steve Reich's <i>Clapping Music</i>	116
4.24	Score and context tree model of the guitar introduction of Deep Pur- ple's <i>Smoke on the Water</i>	118
4.25	G minor blues scale	119
4.26	Score and context tree model demonstrating various harmony modi- fications	120
4.27	Context tree model demonstrating a chord generator and the resulting score	121
4.28	Context tree model using an arpeggio generator and the resulting score	123
4.29	Score and context tree model of J. S. Bach, <i>Prelude in C Major</i> , <i>BWV 846</i> , mm. 1–4	125
4.30	Score and context tree model of a simultaneously played excerpt from <i>Boléro</i> by Maurice Ravel	126
4.31	Score and context tree model of a simple drum groove containing nested repetitions	128
4.32	Context tree model and resulting drum introduction of Coldplay's <i>In</i> <i>My Place</i>	130
4.33	Context tree model using an iteration to produce a G minor blues scale.	132
4.34	Score and context tree model of a sequence from J. S. Bach's <i>Invention</i> <i>No. 4 in D minor, BWV 775</i> , mm. 7–10	133
4.35	Context tree model and resulting score demonstrating a while loop to generate randomly pitched notes	135
4.36	Context tree model and resulting score demonstrating a switch control structure	136
4.37	Screenshot of the editor for the musical context composition language	145
5.1	Music Processing Suite transformation infrastructure	148
5.2	Context tree model of W. A. Mozart, <i>Piano Sonata No. 16 in C</i> <i>major, K. 545</i> , mm. 1–4 and the resulting context and parallelization stacks in the compiler during the transformation to a context layer model	150

5.3	Intermediate context layer model of W. A. Mozart, <i>Piano Sonata No. 16 in C major</i> , K. 545, mm. 1–4	152
5.4	Context layer model of W. A. Mozart, <i>Piano Sonata No. 16 in C major</i> , K. 545, mm. 1–4	152
5.5	Abstract score model structure	153
5.6	Score of W. A. Mozart, <i>Piano Sonata No. 16 in C major</i> , K. 545, mm. 1–4	154
5.7	Three representations of Frédéric Chopin, <i>Étude Op. 10, No. 12 in C minor (Revolutionary Étude)</i> , mm. 1–4	157
5.8	Context layer model of J. S. Bach, <i>Sinfonia 1</i> , BWV 787, mm. 1–2, resulting from the MusicXML code in Appendix B.2	160
5.9	Huffman coding tree illustrating the construction of a possible code for the input sequence <code>abcbcbadae</code>	161
5.10	Ludwig van Beethoven, <i>Piano Sonata No. 21 in C major</i> , Op. 53 (“Waldstein”), mm. 1–8	163
5.11	Redundant context tree model of Ludwig van Beethoven, <i>Piano Sonata No. 21 in C major</i> , Op. 53 (“Waldstein”), mm. 1–8	164
5.12	Node matrix used for computing the hierarchical tree arrangement of the compressed model	165
5.13	Context tree model of Ludwig van Beethoven, <i>Piano Sonata No. 21 in C major</i> , Op. 53 (“Waldstein”), mm. 1–8, after applying <i>auto expansion</i> optimizations	166
5.14	Context tree model of Ludwig van Beethoven, <i>Piano Sonata No. 21 in C major</i> , Op. 53 (“Waldstein”), mm. 1–8, after optimizing the tree structure utilizing inheritance	168
5.15	Context tree model of Ludwig van Beethoven, <i>Piano Sonata No. 21 in C major</i> , Op. 53 (“Waldstein”), mm. 1–8, after extracting fragments	169
5.16	Toolbar of the graphical Music Processing Suite user interface	170
6.1	Ludwig van Beethoven, <i>Symphony No. 5 in C Minor</i> , Op. 67, Mv. I, motif	173
6.2	Overview of the context-based search infrastructure	174
6.3	Search result presentation in the graphical user interface	177
7.1	Analysis scopes	187
7.2	Note duration distribution of Ludwig van Beethoven’s <i>Piano Sonata No. 14 in C# minor</i>	188

7.3	Beat distribution visualizing the relative frequencies of note onset times relative to the beginning of the respective measures in Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	189
7.4	Note duration distribution dependent on beats of Ludwig van Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	190
7.5	Voice-specific note duration distribution dependent on beats of Ludwig van Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	191
7.6	Piano roll representation of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	192
7.7	Pitch distribution of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	193
7.8	Interval Leap Analysis	194
7.9	Interval leap distribution of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	194
7.10	Beat-dependent interval leap distribution of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	195
7.11	Dissonance values of simultaneously audible intervals within two octaves	197
7.12	Rhythm representing the ratio 3 : 2	198
7.13	Dissonance plot of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	199
7.14	Harmony distribution of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	200
7.15	Beat-dependent harmony distribution of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	201
7.16	Chord progression graph of Beethoven's <i>Piano Sonata No. 14 in C# minor</i> , Mv. 1	203
7.17	Harmonic progression graph of Paul Desmond's <i>Take Five</i> projected onto the circle of fifths	204
7.18	Graph visualizing lyric progressions of the first two verses in <i>Yesterday</i> by the Beatles	205
7.19	Aggregated note duration distribution of the 24 preludes in Johann Sebastian Bach's <i>The Well-Tempered Clavier, Book I</i>	207
7.20	J. S. Bach, <i>Prelude No. 15 in G Major, BWV 860</i> , m. 1	207
7.21	Aggregated key distribution of the 24 preludes in Johann Sebastian Bach's <i>The Well-Tempered Clavier, Book I</i>	208
7.22	Aggregated note duration distributions analyzed from a large corpus containing compositions of various composers	209

7.23 Aggregated beat distributions analyzed from a large corpus contain-	
ing compositions of various composers	210
7.24 Aggregated interval leap distributions analyzed from a large corpus	
containing compositions of various composers	211
8.1 Flowchart depicting the basic structure of the evolutionary composi-	
tion algorithm	216
8.2 Comparison between two beat distribution histograms	219
8.3 Node crossover between Beethoven's Symphony 5 motif and Steve	
Reich's <i>Clapping Music</i>	223
8.4 Graph illustrating a small-scale evolutionary process	229
8.5 Manually compiled model recombining musical fragments of eleven	
different compositions	232
8.6 User interface to configure the evolutionary algorithm for composition	
crossover	233
8.7 Context tree model of a composition generated by the evolutionary	
algorithm by recombining existing compositions	234
8.8 Score of a composition generated by the evolutionary algorithm by	
recombining existing compositions	235
8.9 Methodology applied to generate style imitations	238
8.10 Style imitation of W. A. Mozart, <i>Piano Sonata No. 16 in C major</i> ,	
<i>K. 545</i> , mm. 1–4, generated by the evolutionary algorithm	238
8.11 Context tree model of a template composition for a blues	240
8.12 Context tree model of a blues composition generated by the evolu-	
tionary algorithm	241
8.13 Blues composition generated by the evolutionary algorithm	242
8.14 Sections with individual statistical attributes in W. A. Mozart's <i>Piano</i>	
<i>Sonata No. 16 in C major, K. 545</i>	243
8.15 Fitness function defining statistical target features on three different	
hierarchy levels	243
8.16 Graphical user interface for section-wise composition generation . . .	245
8.17 Folder structure containing CSV files for target features and distri-	
butions	246
8.18 Circle of fifths distance distribution used to generate harmonic pro-	
gressions	248
8.19 Target note duration distributions for the first section	249
8.20 Target beat distributions for the first section	250
8.21 Target note duration distributions for the second section	251
8.22 Target beat distributions for the second section	252

8.23 Composition generated by the evolutionary algorithm containing two	
sections with different statistical and musical properties	253

List of Tables

2.1	Encoding of common MIDI events	7
2.2	MIDI meta events	9
2.3	Selection of MuseData codes	13
2.4	Markov transition matrix of harmonic progressions in <i>Yesterday</i> by the Beatles	35
2.5	Chomsky hierarchy	37
2.6	Transition network types and corresponding grammars	41
3.1	Stream events produced by a stream sequencer segmentation shown in Figure 3.5	69
4.1	Note and rest duration syntax	86
4.2	Rhythm syntax examples	88
4.3	Instrument definition parameters	94
4.4	MIDI note numbers and octave numbers according to scientific pitch notation	95
4.5	Pitch syntax	96
4.6	Pitch sequence parameters	97
4.7	List of scales provided by the Music Processing Suite library	99
4.8	Loudness specification syntax	101
4.9	Harmony additions	105
4.10	Rhythmic adjustment modifier parameters	112
4.11	Rhythmic insertion modifier parameters	113
4.12	Rhythmic displacement modifier parameters	113
4.13	Transposition modifier parameters	116
4.14	Parallel interval modifier parameters	119
4.15	Chord generator parameters	122
4.16	Sequence control structure parameters	134
4.17	Expression language literals	138
4.18	Expression language operators ordered by priority	138

4.19	Implicit type conversion rules	140
4.20	Table of available functions	140
4.21	Type cast specifications	142
5.1	Default context values	149
6.1	Search queries and corresponding query stream model patterns	176
6.2	Selected search results for the rhythm of Beethoven's 5 th <i>Symphony</i> motif	178
6.3	Search results for the absolute pitch sequence of Beethoven's 5 th <i>Symphony</i> motif	180
6.4	Search results for the rhythm of Beethoven's 5 th <i>Symphony</i> motif in the key C minor	182
6.5	Search results for the rhythm of Beethoven's 5 th <i>Symphony</i> motif in the key E \flat major	183
8.1	Fitness functions for statistical feature values	220
8.2	Fitness functions for statistical distributions	221
8.3	Mutation operators	224
8.4	Parameters for the evolutionary composition process	226
8.5	Parameters for the evolutionary composition process used for the run depicted in Figure 8.4	228
8.6	Fitness functions for the evolutionary crossover process	235
8.7	Ratios of origin composition elements in the crossover composition	236
8.8	Target values and actual values of statistical features of the generated composition	247
A.1	CD Contents	273

Acronyms

AI Artificial Intelligence [28](#), [30](#)

ANN Artificial Neural Network [44](#), [46](#)–[48](#), [53](#), [54](#), [57](#), [58](#)

ATN Augmented Transition Network [40](#)

BNF Backus–Naur form [37](#)

BPM Beats per Minute [94](#), [159](#)

CSV Comma Separated Values [199](#), [261](#), [262](#)

DFA Deterministic Finite Automaton [38](#)

EA Evolutionary Algorithm [48](#)–[51](#), [230](#)–[232](#), [235](#), [242](#), [253](#), [262](#), [264](#)

EBNF Extended Backus–Naur form [37](#), [154](#)

EC Evolutionary Computation [48](#)

EMF Eclipse Modeling Framework [153](#)

EMI Experiments in Musical Intelligence [42](#)–[44](#)

GA Genetic Algorithm [51](#)–[55](#), [239](#), [242](#)

GE Grammatical Evolution [53](#), [54](#)

GP Genetic Programming [51](#), [55](#), [57](#), [230](#), [231](#), [237](#), [239](#), [242](#)

GUI Graphical User Interface [25](#), [53](#), [271](#)

HMM Hidden Markov Model [33](#), [36](#)

HTML Hypertext Markup Language [290](#)

IDE Integrated Development Environment [2](#)

LSTM Long Short-Term Memory [46](#), [48](#)

MC²L Musical Context Composition Language [2](#), [88](#), [157](#), [290](#)

MIDI Musical Instrument Digital Interface [6](#), [7](#), [9](#), [10](#), [19](#), [23](#), [58](#), [98](#), [157](#), [164](#), [167](#), [169](#), [182](#), [199](#), [290](#)

MIR Music Information Retrieval [18](#)

MP3 MPEG Audio Layer III [290](#)

MPS Music Processing Suite [1](#), [2](#), [4](#), [33](#), [60](#), [63](#), [67](#), [68](#), [70](#), [79](#), [80](#), [91](#), [94](#), [98](#), [99](#), [104](#), [106](#), [108](#), [110](#), [111](#), [113](#), [133](#), [139](#), [144](#), [153](#), [157](#), [158](#), [167](#), [169](#), [182](#), [183](#), [187](#), [196](#), [199](#), [201](#), [213](#), [217](#), [218](#), [223](#), [226](#), [235](#), [248](#), [273](#), [274](#), [290](#)

NFA Non-deterministic Finite Automaton [38](#)

OSC Open Sound Control [167](#)

PDF Portable Document Format [19](#), [164](#), [182](#), [199](#), [226](#), [290](#)

PPQ Pulses per Quarter Note [7](#)

RLE Run-Length Encoding [175](#)

RTN Recursive Transition Network [40](#)

SMF Standard MIDI File [7](#)

SOM Self-organizing Map [46](#)

SPN Scientific Pitch Notation [66](#), [98](#)

TCP Transmission Control Protocol [167](#)

TN Transition Network [40](#), [42](#)

UDP User Datagram Protocol [167](#)

XML Extensible Markup Language [19](#), [20](#)

Chapter 1

Introduction

I can't understand why people are frightened of new ideas. I'm frightened of the old ones.

— John Cage (Kostelanetz 2003)

The advent of computers has drastically changed our possibilities of creating, processing and distributing information. Since the early beginnings of the computing era, computers have been considered for musical applications, which was first proposed by Lady Ada Lovelace in 1843. The first approaches for applying formalizable rules to produce music even go back to Guido of Arezzo around 1000 AD. Today, a broad range of computer-aided applications is available for creating, recording, editing, enhancing, searching, analyzing and evaluating music. Historic and current developments in music information processing are presented in Chapter 2 of this dissertation.

This dissertation focuses on symbolic music processing, which is concerned with the logical structure of elements in musical compositions and the relationships between these elements, as opposed to concrete sounds represented by means of sampled sound waves (Bellini et al. 2006, p. 73). This work contributes to the fields of music representation and algorithms used for symbolic music processing. All proposed models and algorithms are implemented in a versatile software system named Music Processing Suite (MPS), the development of which is documented in this work. MPS is designed to be a multifunctional tool for symbolic music processing applications such as music notation for scores and lead sheets, format transformations, music analysis, music visualization and algorithmic composition, as will be demonstrated in the course of this dissertation. An overview of the system's components is shown in Figure 1.1. Figure 1.2 provides a visual impression of the application.

¹<http://www.musicprocessing.net/>

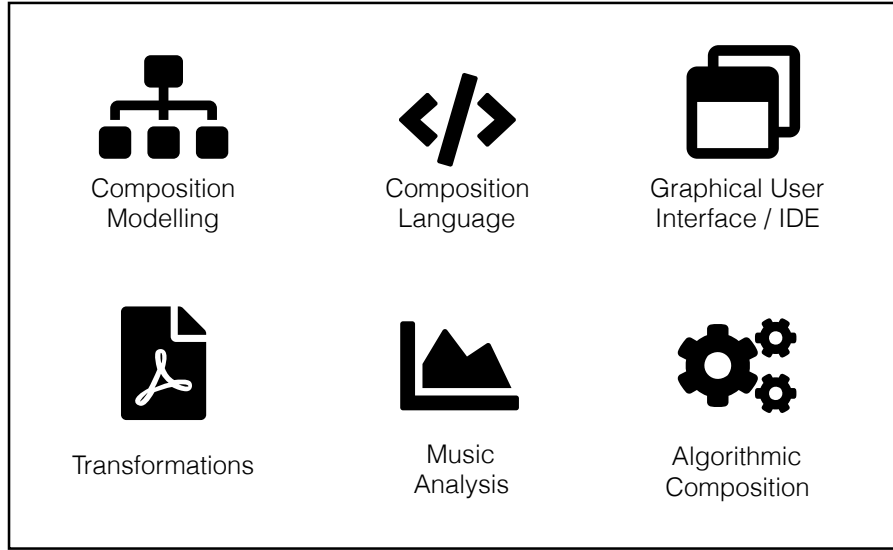


Figure 1.1: Overview of Music Processing Suite components. Icons from fontawesome.com used without modification and in accordance with the [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

New music representation models provide a basis for the proposed system. Instead of simply representing music in terms of note and rest sequences, **MPS** is based on an advanced model introducing multiple musical layers for individual musical aspects. For instance, information relating to instruments, meter, tempo, rhythms, pitches, keys, harmonies and lyrics are represented in separate context layers for each voice. A crucial advantage of the model is that custom context layers can be added. Two different model presentations are proposed: a time-based layer representation, explained in Chapter 3, and an advanced concise tree representation, which is introduced in Chapter 4.

A domain-specific composition language named **Musical Context Composition Language (MC²L)** was developed in conjunction with the composition model in order to make the capabilities of the model accessible to musicians, composers and researchers. The language has a simple intuitive syntax and also allows the specification of musical modifications and control structures. An integral part of the **MPS** application is an **Integrated Development Environment (IDE)** for the composition language featuring syntax highlighting, automatic code completion, code validation and a graphical outline view (see Figure 4.37). The language is introduced in conjunction with the tree-based composition model in Chapter 4.

MPS also features an infrastructure for music format transformations. This includes conversion between the proposed model and language formats, as well as functionality to import compositions in standard formats, namely MIDI and MusicXML. Possible transformations are specified and illustrated in Chapter 5.

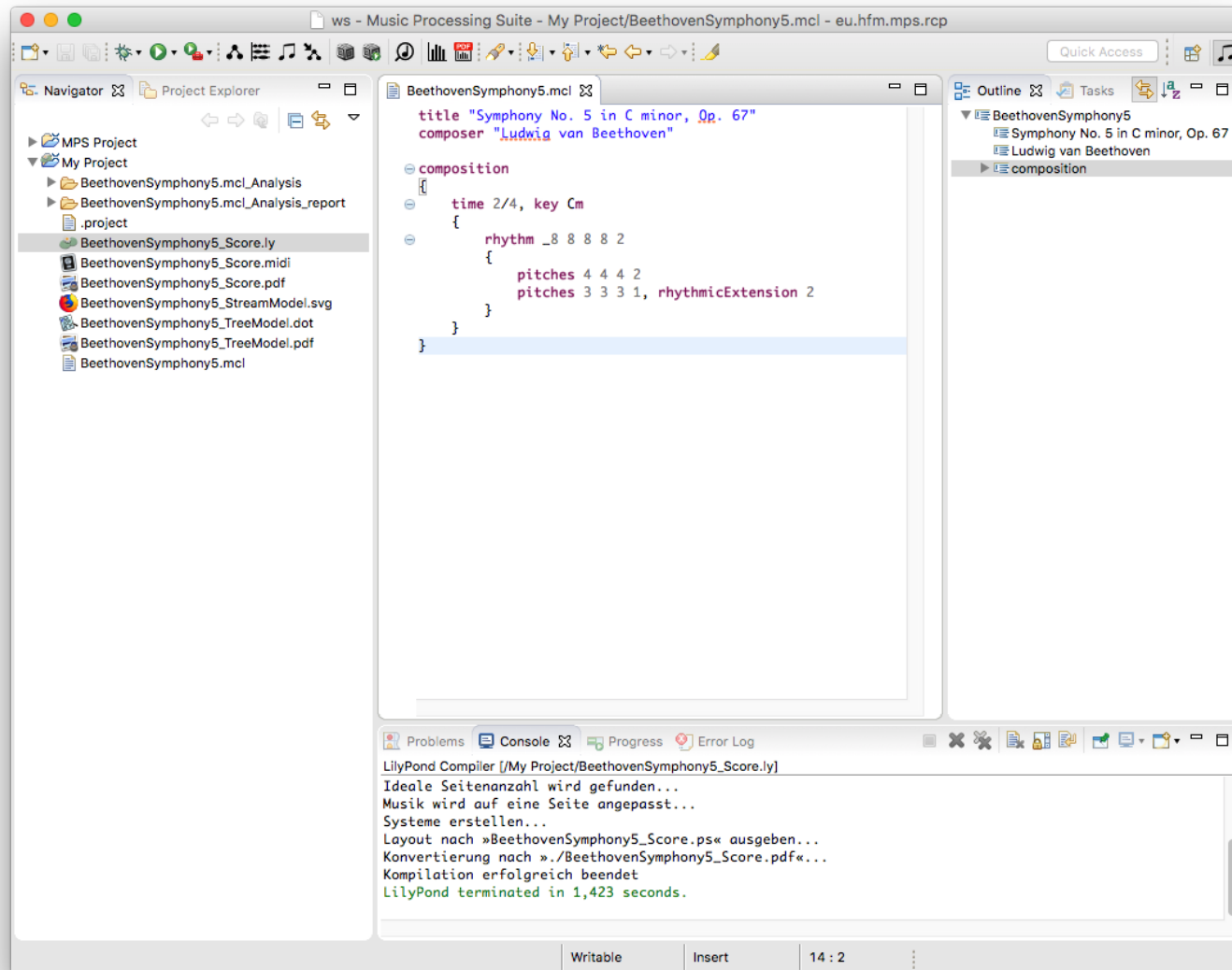


Figure 1.2: Screenshot of the MPS Application. The following views are visible: a toolbar (top, see Chapter 5.8 and Figure 5.16), project explorer (left), composition language editor (center), outline view (right) and console (bottom).

Chapter [6](#) introduces approaches to search musical fragments in arbitrary sized corpora. The advantage of the system is that due to the fine-grained composition model, arbitrary combinations of musical aspects can be specified as search criteria. The composition model design also facilitates fine-grained music analysis. [MPS](#) is capable of generating detailed PDF reports visualizing statistical music analysis data in various scopes from single pieces up to large corpora. The functionality of this subsystem is demonstrated in Chapter [7](#). This chapter has a preparatory character in view of the final chapter, in which statistical music analysis methods play a decisive role.

Finally, Chapter [8](#) is concerned with automated composition. Based on the proposed models and analysis functionality, an evolutionary algorithm is presented, which is capable of recombining musical compositions, creating style imitations and generating original musical material with varying musical properties.

A compact description of the system and its functionality can be found in the related 2016 conference paper published in the proceedings of the International Computer Music Conference (Hofmann [2016](#)).

Part I

Theory and Models

Chapter 2

State of the Art

Know the rules well, so you
can break them effectively.

— Dalai Lama XIV

This chapter summarizes the state of the art regarding essential research topics, models and systems. Relevant research fields include computer science, mathematics, linguistics, musicology, creativity research and biology. The chapter is structured as follows: first, music representation formats are introduced and compared. Second, several aspects of creativity and their potential of being simulated by means of computer programs are discussed. Finally, essential models and algorithms suitable for analyzing, processing and generating music are introduced and discussed.

2.1 Music Representation

This section provides an overview of existing data models to represent symbolic musical information.

2.1.1 Musical Instrument Digital Interface

Musical Instrument Digital Interface (MIDI) is a standard for exchanging symbolic music-related control messages, which was originally designed for exchanging data between keyboards, synthesizers and other devices for musical applications in the early 1980s (Collins 2010, p. 47). The MIDI specification defines the encoding of certain event types and corresponding parameters, the most important of which are summarized in Table 2.1 (Weihs et al. 2016). Although originally designed to interconnect musical equipment for real time applications, event sequences can also

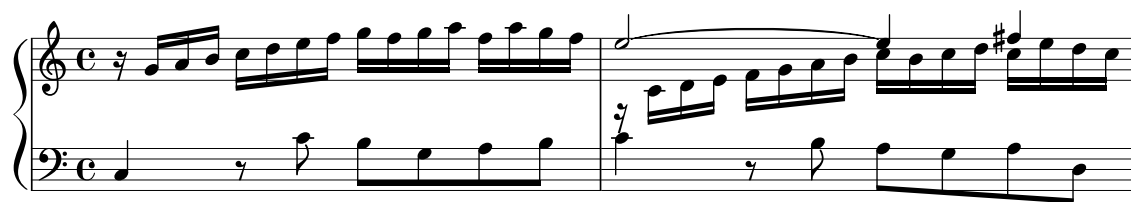
Table 2.1: Encoding of common MIDI events. Status bytes are shown in hexadecimal representation, where n is one of 16 possible MIDI channels.

Status Byte	Data Byte 1	Data Byte 2	Description
8n	note (0-127)	release velocity (0-127)	Note off event: stops the given note in MIDI channel n
9n	note (0-127)	attack velocity (0-127)	Note on event: starts sounding the given note in channel n
An	note (0-127)	pressure (0-127)	Polyphonic pressure: applies aftertouch pressure with the given value to the given note in MIDI channel n
Bn	controller (0-127)	value (0-127)	Controller: sets a given value to a given numbered controller on MIDI channel n
Cn	program (0-127)		Program Change: selects the given program on MIDI channel n , e.g. an instrument
Dn	pressure (0-127)		Channel Pressure: applies aftertouch pressure to all sounding notes in channel n
En	lsb (0-127)	msb (0-127)	Pitch Bend: applies pitch bending to all sounding notes in channel n . The seven bits of both data bytes are interpreted as one large 14 bit number, providing accurate pitch bend resolution.

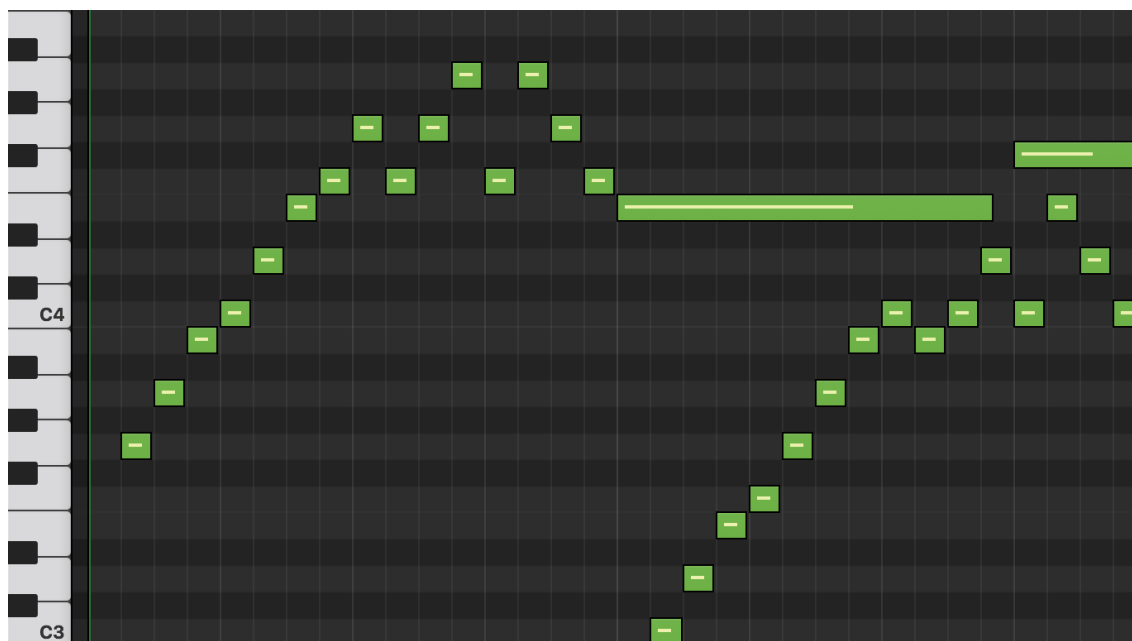
be represented in files. A common encoding is defined for **Standard MIDI Files (SMFs)**. The events are associated with time stamps, which are given in the unit *MIDI ticks*. Typically, the number of MIDI ticks per quarter note, or **Pulses per Quarter Note (PPQ)**, are specified in a corresponding **MIDI** file header.

MIDI files contain one or more tracks, each of which contain MIDI events (see section 2.1.1). A common visualization of *note on* and corresponding *note off* events is a piano roll representation, which is illustrated using J. S. Bach's *Sinfonia 1, BWV 787* as an example. The score and corresponding piano roll representations of the two **MIDI** tracks are shown in Figure 2.1. MIDI uses the following strategy to encode pitches: each key on the piano, i.e. each semitone step, is assigned a unique number, middle C being encoded as the number 60, C \sharp above as 61, D as 62 and so on.

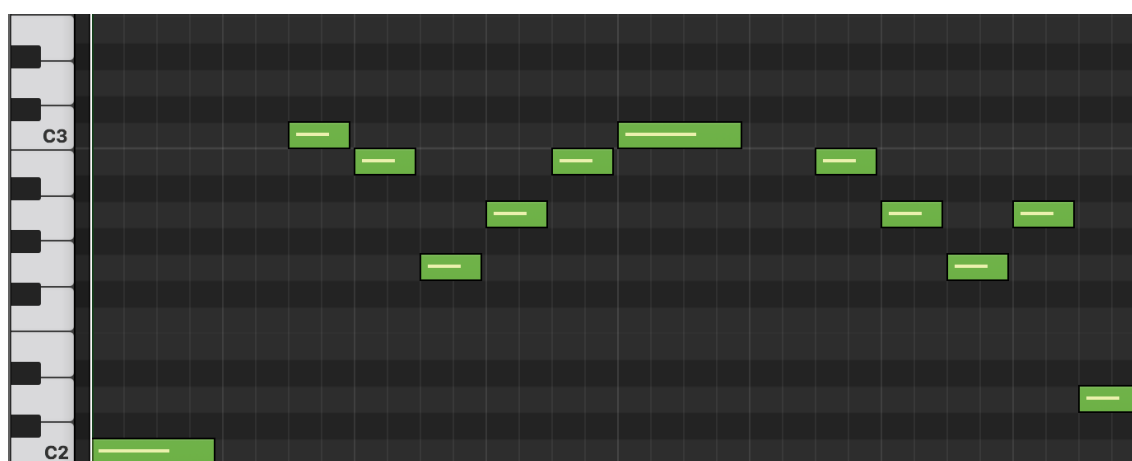
The duration of notes is not explicitly given in the **MIDI** format. Instead, individual note durations have to be derived from the time stamps of *note on* and *note off* events with corresponding pitches. The note event matching is not trivial, because events with matching pitches do not necessarily occur consecutively. For example, the *note off* event for the half note beginning in measure 2 (see Figure 2.1a), which



(a) Score of J. S. Bach, *Sinfonia 1*, BWV 787, mm. 1–2



(b) Piano roll representation of the upper part



(c) Piano roll representation of the lower part

Figure 2.1: Score and piano roll representations of MIDI events in J. S. Bach, *Sinfonia 1*, BWV 787, mm. 1–2, visualized in Logic Pro X. Corresponding audio and MIDI files are available on the accompanying CD under `Audio/Examples/Bach/BWV787_Sinfonia1` (see Appendix [A](#)).

Table 2.2: MIDI meta events

Data Sequence	Description
FF 01 length text	Arbitrary ASCII text for comments or descriptions
FF 02 length text	Copyright text
FF 03 length text	Track name
FF 04 length text	Instrument name
FF 05 length text	Lyrics
FF 06 length text	Marker text, e.g. ‘Verse’ or ‘Chorus’
FF 2F 00	End of track
FF 51 03 tt tt tt	Tempo specification. The three bytes tt tt tt are interpreted as 24-bit value containing the tempo in microseconds per quarter note.
FF 58 04 nn dd cc bb	Time signature with numerator nn and denominator 2^{dd} . The number cc specifies the number of MIDI ticks between metronome clicks. bb specifies the number of notated 32 nd notes per quarter note (normally 8).
FF 59 02 sf mi	Indicates a key signature where the byte sf encodes the number of flats or sharps: 0 equals C major (or A minor), -1 represents 1 flat, 1 is equivalent to 1 sharp, etc. The second byte mi indicates a major (0) or minor (1) key.

is tied to the following quarter note, occurs after note events for the simultaneously played series of sixteenth notes.

MIDI also specifies so called *meta events*, which encode additional information musically relevant for a piece, such as tempo specifications, time signatures, key signatures or lyrics. Correspondent MIDI messages have the format **FF type length data**. A selection of meta events is listed in Table 2.2.

MIDI has proven to be a versatile symbolic music format and is still used today in many recording, production and live performance scenarios. Disadvantages commonly associated with MIDI are: the encoding of pitches, which makes enharmonic decisions impossible (e.g. F \sharp and G \flat are encoded with the same MIDI note number) and difficulties encoding complex rhythms, which are not in all cases accurately representable in MIDI ticks, leading to quantization issues. The latter issue sometimes becomes apparent when importing MIDI files into score editing software. Furthermore, MIDI data is not human-readable due to its organization in raw byte sequences and is therefore only interpretable by computers.

2.1.2 MuseData

MuseData is a music description format used to encode the logical content of musical scores. The *MuseData* specification includes guidelines on how to store files representing a corpus of compositions in a hierarchical folder structure. Files are categorized into *Stage-1 files*, which contain pure pitch and duration information, and *Stage-2 files* which are used for a variety of applications such as generating MIDI files, analysis and printing scores or parts (Hewlett 1997, p. 402).

```

1 measure 1
2 E4 2
3 D4 2
4 C4 2
5 rest 2
6 measure 2
7 E4 2
8 D4 2
9 C4 2
10 rest 2
11 measure 3
12 G4 2
13 F4 1
14 F4 1
15 E4 4
16 measure 4
17 G4 2
18 F4 1
19 F4 1
20 E4 4

```

Listing 2.1: *MuseData* records representing the first four measures of the children’s folk song *Three Blind Mice* (Hewlett 1997, p. 412)

MuseData files are organized as “a set of time-ordered, variable length ASCII records” (Hewlett 1997, p. 406) Header information include work and movement numbers and titles, key, time division metadata, clefs, transposing part metadata, number of staves and the number of instruments represented. Pitch information is encoded with the three parameters *pitch name* (A...G or r for rests), *chromatic inflection* (sharps, flats, natural) and *octave number*. Durations are specified based on divisions. The number of divisions per quarter note is given as header information. A simple example is shown in Listing 2.1.

A more complex example is shown in Listing 2.2, in which four measures of a Mozart sonata are encoded. The corresponding score is shown in Figure 2.2.

```

1
2
3
4 06/23/94 W. Hewlett
5 WK#:331 MV#:3
6 Breitkopf & H\3artel
7 Piano Sonata
8 1 0
9 Group memberships: sound, score

```

CHAPTER 2. STATE OF THE ART

```

10 sound: part 1 of 1
11 score: part 1 of 1
12 &
13 Initial conversion from stage 1 to stage 2
14 &
15 $ K:3 Q:4 T:2/4 C1:4 C2:22
16 C#6 8 h u1 S
17 back 8 1
18 A5 4 q d1
19 E5 4 q d1
20 C#5 4 q d1
21 back 4
22 gA2 4 t u2 [[[ [
23 P C33:o
24 gC#3 4 t u2 ===
25 gE3 4 t u2 ]]]
26 A3 2 e d2 [ ]
27 A3 2 e d2 =
28 A3 2 e d2 =
29 A3 2 e d2 ]
30 measure 2
31 D6 1 s d1 [[ (
32 C#6 1 s d1 == )
33 B5 1 s d1 == .
34 C#6 1 s d1 ]] .
35 D6 1 s d1 [[ (
36 C#6 1 s d1 == )
37 B5 1 s d1 == .
38 C#6 1 s d1 ]] .
39 back 8
40 gA2 4 t u2 [[[ [
41 P C33:o
42 gC#3 4 t u2 ===
43 gE3 4 t u2 ]]]
44 A3 2 e d2 [ ]
45 A3 2 e d2 =
46 A3 2 e d2 =
47 A3 2 e d2 ]
48 measure 3
49 D6 8 h d1
50 A5 8 h d1
51 F#5 8 h d1
52 back 8
53 gD2 4 t u2 [[[ [
54 P C33:o
55 gF#2 4 t u2 ===
56 gA2 4 t u2 ]]]
57 D3 2 e d2 [ ]
58 D3 2 e d2 =
59 D3 2 e d2 =
60 D3 2 e d2 ]
61 measure 4
62 gD6 4 t u1 (
63 C#6 2 e d1 [ )
64 A5 2 e d1
65 E5 2 e d1
66 gD6 4 t u1 (
67 C#6 2 e d1 = )

```

```

68  A5      2      e      d1
69  E5      2      e      d1
70  gD6     4      t      u1      (
71  C#6     2      e      d1 =    )
72  A5      2      e      d1
73  E5      2      e      d1
74  gD6     4      t      u1      (
75  C#6     2      e      d1 ]    )
76  A5      2      e      d1
77  E5      2      e      d1
78  back    8
79  gA2     4      t      u2 [[[  [
80  P      C33:o
81  gC#3    4      t      u2 ===
82  gE3     4      t      u2 ]]]
83  A3      2      e      d2 [    ]
84  A3      2      e      d2 =
85  A3      2      e      d2 =
86  A3      2      e      d2 ]
87  measure 5
88  B5      6      q.    u1      (
89  E6      2      e      u1      )
90  back    8
91  G#5     8      h      d1
92  E5      8      h      d1
93  back    8
94  gE2     4      t      u2 [[[  [
95  P      C33:o
96  gG#2    4      t      u2 ===
97  gB2     4      t      u2 ]]]
98  E3      2      e      d2 [    ]
99  E3      2      e      d2 =
100 E3      2      e      d2 =
101 E3      2      e      d2 ]
102 measure 6
103 /END

```

Listing 2.2: MuseData code representing W. A. Mozart, *Piano Sonata No. 11 in A major, K. 331/300i*, Mv. 3 (“Alla Turca”), mm. 98–102 (Hewlett [1997], p. 412)



Figure 2.2: W. A. Mozart, *Piano Sonata No. 11 in A major, K. 331/300i*, Mv. 3 (“Alla Turca”), mm. 98–102. Edited by Drew Weymouth and made available at imslp.org under the [Creative Commons BY-NC 3.0 License](https://creativecommons.org/licenses/by-nc/3.0/). Corresponding audio and MIDI files are available on the accompanying CD under `Audio/Examples/Mozart/RondoAllaTurca` (see Appendix [A](#)).

In Listing [2.2](#), chords are specified in two different ways. The first variant is to use spaces as prefix for pitch specifications, which causes notes to be appended to the

previous one (e.g. lines 19 and 20). Another strategy is to use *backup* commands (encoded as **back**) to go back in time and add simultaneously played notes with a different duration (e.g. in line 17).

The table-like structure of MuseData records is intuitive regarding the chronological sequence and the granularity of the represented instructions. However, the encoding of the conveyed instructions is not always clear, since in part cryptic codes are used. A selection of MuseData codes is shown in Table 2.3 (Hewlett 1997, pp. 410ff.).

Table 2.3: Selection of MuseData codes

Code	Description	Code	Description
L	graphic note type long	g	grace note
b	graphic note type breve	c	cue note
w	graphic note type whole	.	dot for rhythmic prolongation
h	graphic note type half	:	double dot for rhythmic prolongation
q	graphic note type quarter	*	start tuplet
e	graphic note type eighth	!	stop tuplet
s	graphic note type sixteenth	A	vertical accent up
t	graphic note type 32 nd	V	vertical accent down
x	graphic note type 64 th	>	horizontal accent
y	graphic note type 128 th	.	staccato
z	graphic note type 256 th	_	tenuto or marcato
#	sharp	s	sharp on ornament
n	natural	h	natural on ornament
f	flat	b	flat on ornament
x	double sharp	ss	double sharp on ornament
X	sharp-sharp	+	cautionary/written out accidental
&	flat-flat	bb	double flat on ornament
d	stem down	S	arpeggiate (chords)
u	stem up	F	upright fermata
[start beam	E	inverted fermata
=	continue beam	p	piano (<i>p</i> , <i>pp</i> , etc.)

Continued on next page

Table 2.3 – *Continued from previous page*

Code	Description	Code	Description
]	end beam	m	mezzo (<i>mp</i> , <i>mf</i>)
-	tie	f	forte (<i>f</i> , <i>ff</i> , <i>fp</i> , etc.)
(open slur 1	Z	<i>sfz</i>
)	close slur 1	Zp	<i>sfp</i>
[open slur 2	R	<i>rfz</i>
]	close slur 2	t	<i>tr</i> (trill)

Analyzing the encodings in Table 2.3, another issue becomes apparent: depending on the context, different encodings are used for equivalent circumstances. For instance, accidentals on regular notes are encoded differently than on ornaments: sharps are denoted with the code `#` in conjunction with regular notes and with `s` for ornaments.

2.1.3 Humdrum

Humdrum is a music information processing software primarily targeted at music researchers (Huron 2002). The system consists of two components: the Humdrum Syntax and the Humdrum Toolkit. The syntax is designed to accommodate any type of sequential symbolic data, and can also be used for non-musical purposes. The Humdrum Toolkit provides a broad variety of tools to process data, namely:

- Visual display (rendering scores or score parts)
- Aural display (playing score or score parts)
- Searching (looking for specific information)
- Counting (counting occurrences of specific structures)
- Editing (changing scores according to certain rules or instructions)
- Editorializing (adding editorial comments)
- Transformations (e.g. applying transpositions, replacing chords according to specified rules)
- Arithmetic analysis (e.g. analyzing intervals)
- Extraction and selection of data (e.g. extracting score sections, excluding specified parts)

- Linking and joining data (e.g. assembling scores, correlate musical data with additional external data)
- Generating inventories (creating lists of certain symbols or structures)
- Classifying (e.g. identifying certain chords or intervals)
- Labelling (e.g. marking sections or words)
- Comparison (e.g. comparing manuscripts and edited scores)
- Trouble-shooting (e.g. checking if notational criteria are met)

Similar to MuseData, Humdrum uses a table-oriented description syntax, which is evocative of spreadsheets. The data representation is very flexible, since the columns, each representing a block of arbitrary data, can be arranged as required by the specific musical task at hand. For instance, these could contain pitches, durations, harmonies or fingering instructions. Generally, entries in the same row represent concurrent events and rows are ordered chronologically, i.e. rows are interpreted as consecutive events. Refer to Listing 2.3, in which the subject of J. S. Bach's *The Art of Fugue, BWV 1080* is encoded in the so called *kern*¹ representation, which accommodates musical core information for western musical scores². An equivalent score is presented in Figure 2.3.



Figure 2.3: J. S. Bach, *The Art of Fugue, BWV 1080*, subject. Corresponding files are available on the accompanying CD under `Examples/Compositions/Bach/BWV1080_TheArtOfFugue` (see Appendix A).

A more complex example containing multiple columns (so called *spines*) is shown in Listing 2.4. The code was generated by a program developed specifically to interpret a custom Humdrum representation of harmonic progressions in jazz music (Broze and Shanahan 2012). The input is repeated in the first column or spine with the identifier `**jazz`. The `**kern` representation of the root note is visible in the second spine. The `**exten` column contains extended harmonic information only, without the root note. The following column represents the root note of the corresponding harmony as defined by solfège solmization. `**mint` stands for *melodic interval representation* and encodes the musical intervals between successive

¹*Kern* is German for *core*

²An excellent graphical explanation of the file contents is available at <http://www.humdrum.org/guide/ch02/>

```

1  **kern
2  *clefG2
3  *k[b-]
4  *d:
5  *M2/2
6  *met(c)
7  =1-
8  2d/
9  2a/
10 =2
11 2f/
12 2d/
13 =3
14 2c#/
15 4d/
16 4e/
17 =4
18 [2f/
19 8f]/L
20 8g/
21 8f/
22 8e/J
23 =5
24 4d/
25 *-

```

Listing 2.3: Humdrum representation of J. S. Bach, *The Art of Fugue*, BWV 1080, subject (Huron 1995)

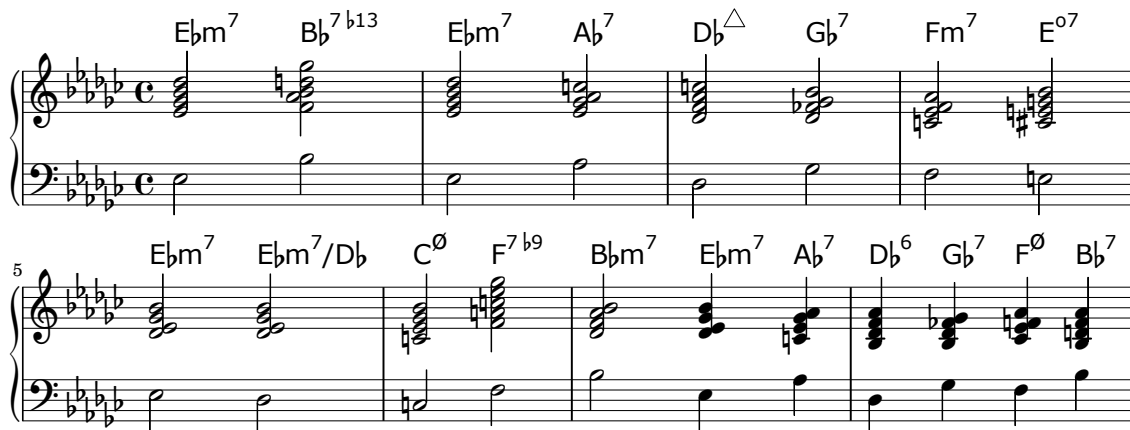


Figure 2.4: Score illustrating the harmonic progression specified in Listing 2.4. Note that the Humdrum code in the listing only specifies abstract harmonies, not the specific chord voicings. This score was primarily generated using chord generators, which were developed as a component of Music Processing Suite (see Chapter 4.7.1). Corresponding files are available on the accompanying CD under `Examples/Model/ChordGenerator/JazzHarmonicProgression` (see Appendix A).

harmony root notes. Interval numbers are accompanied by the following signifiers: P for perfect, M for major, m for minor and d for diminished intervals, among others³. The `**quals` spine contains the chord quality only, and the final column (`**dur`) contains harmony durations. A corresponding score is shown in Figure 2.4.

³<http://www.humdrum.org/rep/mint/>

```

1  **jazz      **kern  **exten **solfa **mint  **quals **dur
2  *thru      *thru  *thru  *thru  *thru  *thru  *thru
3  *M4/4      *M4/4  *M4/4  *M4/4  *M4/4  *M4  *M4/4
4  *D-:      *D-:   *D-:   *D-:   *D-:   *D-:   *D-:
5  2E-:min7   E-    min7   re    [E-]   min7   2.0000
6  2B-7b13    B-    7b13   la     P5     dom   2.0000
7  =          =      =      =      =      =      =
8  2E-:min7   E-    min7   re     P4     min7  2.0000
9  2A-7       A-     7      so     P4     dom   2.0000
10 =          =      =      =      =      =      =
11 2D-:maj7    D-    maj7   do     P4     maj   2.0000
12 2G-7       G-     7      fa     P4     dom   2.0000
13 =          =      =      =      =      =      =
14 2F:min7     F      min7   mi     M7     min7  2.0000
15 2Eo7        E      o7     ri     M7     dim   2.0000
16 =          =      =      =      =      =      =
17 2E-:min7    E-    min7   re     d1     min7  2.0000
18 2E-:min7/D- E-    min7   re     P1     min7  2.0000
19 =          =      =      =      =      =      =
20 2Ch7        C      h7     ti     M6     half  2.0000
21 2F7b9       F      7b9    mi     P4     dom   2.0000
22 =          =      =      =      =      =      =
23 2B-:min7    B-    min7   la     P4     min7  2.0000
24 4E-:min7    E-    min7   re     P4     min7  1.0000
25 4A-7        A-     7      so     P4     dom   1.0000
26 =          =      =      =      =      =      =
27 4D-6        D-     6      do     P4     maj   1.0000
28 4G-7        G-     7      fa     P4     dom   1.0000
29 4Fh         F      h      mi     M7     half  1.0000
30 4B-7        B-     7      la     P4     dom   1.0000
31 ==          ==     ==     ==     ==     ==     ==
32 *-          *-     *-     *-     *-     *-     *-

```

Listing 2.4: Humdrum representation of a harmonic progression in a jazz piece (Broze and Shanahan [2012](#))

2.1.4 abc

abc is a text-based system for music notation. By design, it is readable and comprehensible by both humans and computers (Walshaw [2011](#)). The *abc* representation of the Bach subject in Figure [2.3](#) is presented in Listing [2.5](#).

```

1 T:Art of Fugue Subject
2 C:J. S. Bach
3 M:4/4
4 L:1/2
5 K:Dm
6 D A F D | ^C D/2 E/2 | F- F/4G/4F/4E/4 | D/2

```

Listing 2.5: J. S. Bach, *The Art of Fugue*, BWV 1080, subject, notated in the *abc* language

Listing [2.5](#) starts with the specification of metadata, namely the title and the composer of the piece followed by musical context information such as the time signature (M:4/4), a reference note duration (L:1/2) and the key of the composition (K:Dm). The score contents are given as a sequence of symbols. Pitches are specified using note names, the suffixes , and ' may be used for octave translations. The prefixes ^ and _ are used for sharp and flat accidentals, respectively. Durations are specified as

a multiple of the reference duration given in the header. In the example, the reference duration is set to half notes, therefore quarter notes are written with the suffix /2, e.g. the third note D/2. The *abc* system also supports advanced musical notations such as beams, repeats, ties, slurs, grace notes, triplets, chords, chord symbols and lyrics (Walshaw [2011]). A number of software implementations for interpreting and rendering *abc* are available⁴.

2.1.5 GUIDO

GUIDO is a formal language for the representation of musical scores in human-readable text format (Hoos, Hamel, et al. [1998]). The system consists of three layers with support for increasing notational complexity:

1. *Basic GUIDO* for basic music representation
2. *Advanced GUIDO* for exact score formatting options and more complex musical structures
3. *Extended GUIDO* supporting non-conventional music notation

The Bach subject shown in Figure 2.3 can be represented in *GUIDO* as illustrated in Listing 2.6:

```
1 [ \meter <"4/4"> \key <-1>
2 d/2 a f d c# d/4 e \tieBegin f/2 \beam( f/8 \tieEnd g f e) d/4 ]
```

Listing 2.6: J. S. Bach, *The Art of Fugue*, BWV 1080, subject, in *GUIDO* notation

Advanced applications, such as more complex notation and Music Information Retrieval (MIR) based on *GUIDO* are documented in (Hoos, Hamel, et al. [1998]; Hoos, Renz, et al. [2001]).

2.1.6 LilyPond

LilyPond is a music engraving system based on text files and a compiler which is capable of rendering high quality Portable Document Format (PDF) files and MIDI files. The system provides automatic layout and formatting capabilities, i.e. even if no layout specifications are given, the system is designed to output well-organized and easily readable scores. Raw music information (e.g. pitches and durations) can be enhanced with instructions regarding the layout of the score (Nienhuys and Nieuwenhuizen [2003]).

⁴<http://abcnotation.com/software>

The subject of Bach's *The Art of Fugue*, BWV 1080 (see Figure 2.3) is represented in LilyPond as shown in Listing 2.7.

```

1 \version "2.12.0"
2 #(set-default-paper-size "a4")
3
4 \header {
5   title = "Art of Fugue Subject"
6   composer = "J. S. Bach"
7 }
8
9 \score {
10   \new Staff {
11     \key d \minor
12
13     \relative c' {
14       d2 a' f d cis d4 e f2~ f8 g f e d4
15     }
16   }
17
18   \layout {}
19 }

```

Listing 2.7: J. S. Bach, *The Art of Fugue*, BWV 1080, subject, represented in LilyPond

LilyPond is capable of representing a great variety of musical elements such as microtonal pitches, basic and complex rhythms (e.g. nested tuplets), expressive marks, repeats with multiple alternatives and chord symbols. Special notation constructs for vocal music, multi-staff instruments, unfretted and fretted string instruments, percussion and wind instruments are available. The system also supports advanced facilities for contemporary music, ancient notation and world music⁵.

2.1.7 MusicXML

MusicXML is a music notation interchange format based on the generic **Extensible Markup Language (XML)**. MusicXML was explicitly designed to circumvent issues of existing proprietary music interchange formats, and is based on the foundations of MuseData and Humdrum (Good 2001, p. 2). **XML** is a generic data interchange format which is both readable by humans and computers. It allows the hierarchical arrangement of arbitrarily named elements represented in the form of so called *tags*, which can optionally be associated with attributes and can have textual content (Marchal 2002, p. 42ff.). For example, a whole note with middle C pitch is represented in MusicXML as shown in Listing 2.8.

Each note is represented as an **XML** element named **note**, which contains three child elements: **pitch**, in turn containing the child elements **step** and **octave**, **duration** and **type**. Each element is represented by an opening tag and a corresponding closing tag. Closing tags are denoted by the corresponding opening tag name prefixed with

⁵<http://lilypond.org/doc/v2.18/Documentation/notation/>

```

1 <note>
2   <pitch>
3     <step>C</step>
4     <octave>4</octave>
5   </pitch>
6   <duration>4</duration>
7   <type>whole</type>
8 </note>

```

Listing 2.8: MusicXML representation of a measure containing a whole note with middle C pitch

a slash (/). Elements can in turn contain nested elements or text content (for example, the element `<type>` contains the text `whole`, designating a whole note printed in the score). Note durations are specified in terms of so called *divisions*. In the previous example, a whole note has the length of four divisions, implying that a quarter note has a duration of one division. Refer to the next example for another division scenario. The reason why there is a supposedly redundant element named `type` with the value `whole` is that the actual duration of the note can be different from the note printed in the score. For example, long sequences of triplets are often not explicitly marked as such, but as notes with a “simpler” duration. Examples for this can be found in Beethoven’s *Piano Sonata No. 14 in C# minor* (see Figure 3.4a, mm. 2ff.) and Bach’s *Prelude No. 15 in G Major, BWV 860* (see Figure 7.20). A more complex example containing the first measure of *Yesterday* by the Beatles is demonstrated in Listing 2.9. The corresponding score representation is shown in Figure 2.5.

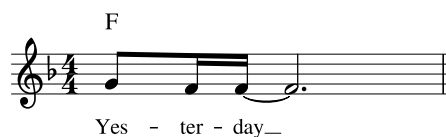


Figure 2.5: The Beatles, *Yesterday*, m. 1, vocal part

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
3   "http://www.musicxml.org/dtds/partwise.dtd">
4 <score-partwise>
5   <work>
6     <work-title>Yesterday</work-title>
7   </work>
8   <identification>
9     <creator type="composer">Paul McCartney</creator>
10  </identification>
11  <part-list>
12    <score-part id="P1">
13      <part-name>Voice</part-name>
14      <score-instrument id="P1-I1">
15        <instrument-name>Voice</instrument-name>
16      </score-instrument>

```

```

17     <midi-device id="P1-I1" port="1"></midi-device>
18     <midi-instrument id="P1-I1">
19         <midi-channel>1</midi-channel>
20         <midi-program>53</midi-program>
21         <volume>78.7402</volume>
22         <pan>0</pan>
23     </midi-instrument>
24 </score-part>
25 </part-list>
26 <part id="P1">
27     <measure number="1" width="425.98">
28         <attributes>
29             <divisions>4</divisions>
30             <key>
31                 <fifths>-1</fifths>
32             </key>
33             <time>
34                 <beats>4</beats>
35                 <beat-type>4</beat-type>
36             </time>
37             <clef>
38                 <sign>G</sign>
39                 <line>2</line>
40             </clef>
41         </attributes>
42         <harmony print-frame="no">
43             <root>
44                 <root-step>F</root-step>
45             </root>
46             <kind>major</kind>
47         </harmony>
48         <note default-x="85.20" default-y="-30.00">
49             <pitch>
50                 <step>G</step>
51                 <octave>4</octave>
52             </pitch>
53             <duration>2</duration>
54             <voice>1</voice>
55             <type>eighth</type>
56             <stem>up</stem>
57             <beam number="1">begin</beam>
58             <lyric number="1">
59                 <syllabic>begin</syllabic>
60                 <text>Yes</text>
61             </lyric>
62         </note>
63         <note default-x="163.45" default-y="-35.00">
64             <pitch>
65                 <step>F</step>
66                 <octave>4</octave>
67             </pitch>
68             <duration>1</duration>
69             <voice>1</voice>
70             <type>16th</type>
71             <stem>up</stem>
72             <beam number="1">continue</beam>
73             <beam number="2">begin</beam>
74             <lyric number="1">

```

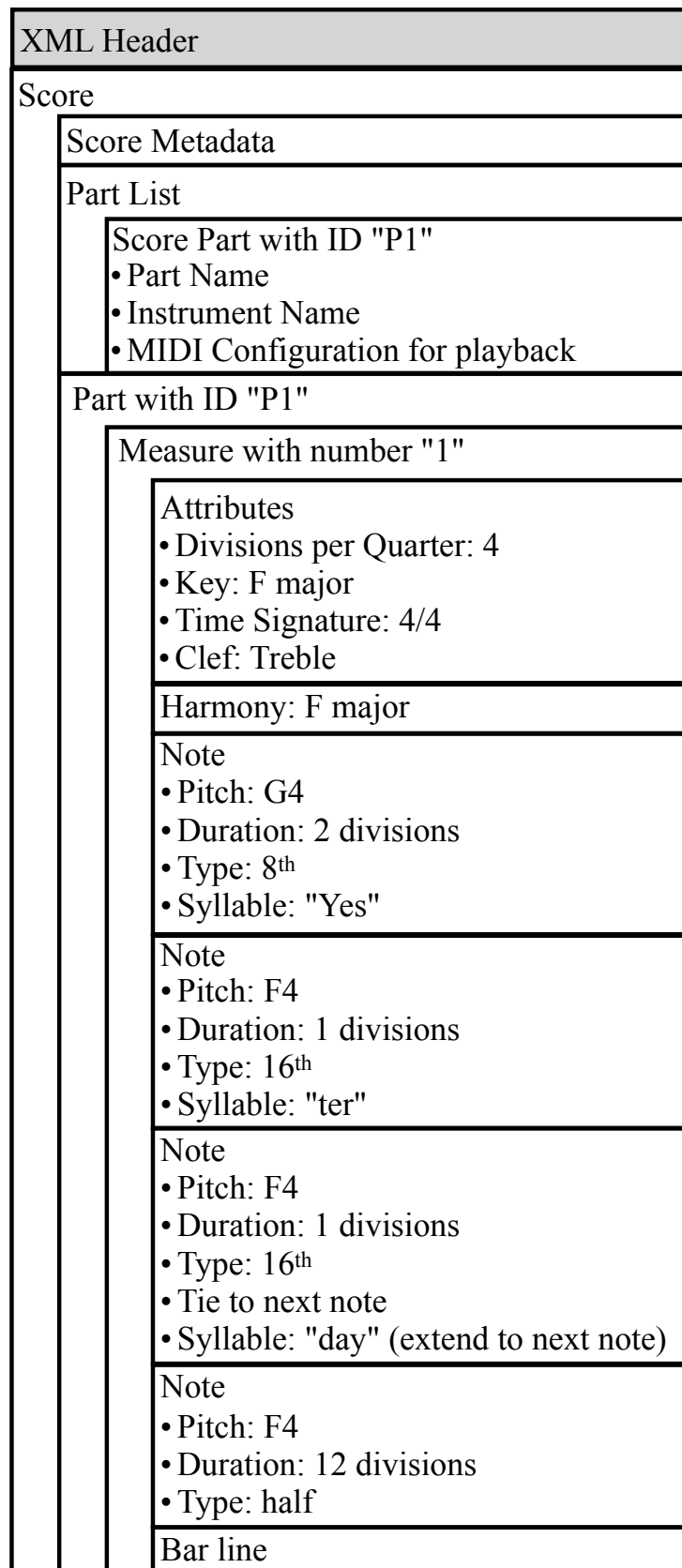
```

75     <syllabic>middle</syllabic>
76     <text>ter</text>
77 </lyric>
78 </note>
79 <note default-x="212.36" default-y="-35.00">
80   <pitch>
81     <step>F</step>
82     <octave>4</octave>
83   </pitch>
84   <duration>1</duration>
85   <tie type="start"/>
86   <voice>1</voice>
87   <type>16th</type>
88   <stem>up</stem>
89   <beam number="1">end</beam>
90   <beam number="2">end</beam>
91   <notations>
92     <tied type="start"/>
93   </notations>
94   <lyric number="1">
95     <syllabic>end</syllabic>
96     <text>day</text>
97     <extend/>
98   </lyric>
99 </note>
100 <note default-x="260.91" default-y="-35.00">
101   <pitch>
102     <step>F</step>
103     <octave>4</octave>
104   </pitch>
105   <duration>12</duration>
106   <tie type="stop"/>
107   <voice>1</voice>
108   <type>half</type>
109   <dot/>
110   <stem>up</stem>
111   <notations>
112     <tied type="stop"/>
113   </notations>
114 </note>
115 <barline location="right">
116   <bar-style>light-heavy</bar-style>
117 </barline>
118 </measure>
119 </part>
120 </score-partwise>

```

Listing 2.9: MusicXML representation the first measure of *Yesterday* by the Beatles. Refer to Figure 2.6 for a visualization of the logical structure.

Listing 2.9 contains not only the notes, but also lots of metadata regarding the printed representation of the score. The logical structure of the document is visualized in Figure 2.6. Note that MusicXML representation does not only address the encoding of musical entities such as notes, rests and chords, but also encodes information relevant for the corresponding score notation. While MIDI only en-

Figure 2.6: Logical structure of the MusicXML score representation shown in Listing 2.9

codes events (such as note on and note off), MusicXML is designed to encode score elements and additional graphical notation details. For example, it is explicitly specified that the note on the syllable “day”, which lasts for 9 sixteenth notes, is notated as a sixteenth note tied to a half note. Another example is the explicit specification of beams. In line 57, a beam (with the number 1) starts at the first note. At the second note in line 72, this beam is continued and a second beam (with the number 2) starts in line 73. Both beams end at the third note, in lines 89 and 90, respectively. Refer to Figure 2.5 for the visualization of these beams.

2.1.8 music21

Designed as a toolkit for computer-aided musicology, *music21*⁶ provides a variety of features concerning music data processing. The system is based on the Python programming language and uses object-oriented data structures for music representation (Ariza and Cuthbert 2010). It provides an import functionality for MIDI, MuseData, Humdrum’s ****kern** format and MusicXML files (Cuthbert and Ariza 2010).

The toolkit offers lots of flexibility due to the fact that users are able to load, traverse and manipulate musical object structures by writing custom Python code. *music21* also provides an extensible syntax for easy music notation called *TinyNotation*⁷. For example, the subject from Bach’s *The Art of Fugue*, which was already introduced in Figure 2.3, can be notated and displayed with the Python code in Listing 2.10.

```
1 from music21 import *
2 converter.parse("tinynotation: d2 a f d c# d4 e f2~ f8 g f e d4").show()
```

Listing 2.10: Python code to notate and display the subject of Bach’s *The Art of Fugue*, BWV 1080 using *music21*

Other advantages of *music21* include support for musical analysis (Cuthbert and Ariza 2010), feature extraction (Cuthbert, Ariza, and Friedland 2011) and the enrichment of scores with derived data (Ariza and Cuthbert 2011).

2.1.9 SARAH

SARAH is a structured composition language based on term rewriting (Fox 2006). Conceptually, the language is comparable to transformational music grammars proposed by Holtzman (Holtzman 1980). Using *SARAH*, hierarchical musical structures can be defined in terms of a set of grammar rules, as illustrated in Listing 2.11.

⁶<http://web.mit.edu/music21/>

⁷http://web.mit.edu/music21/doc/usersGuide/usersGuide_16_tinyNotation.html

```

1 POP-SONG => {INTRO, VERSE, CHORUS, VERSE, CHORUS, BRIDGE, CHORUS, OUTRO}
2 VERSE => {RIFF-A, RIFF-B, RIFF-A, RIFF-B}
3 RIFF => [VOCALS, BASS, GUITAR]

```

Listing 2.11: Set of grammar rules in the *SARAH* language describing different hierarchical levels of a pop song (Fox [2006], p. 1). Rules in curly brackets indicate sequential parts, whereas rules in square brackets represent simultaneously played parts.

Based on the concept of bracketed Lindenmayer systems (Prusinkiewicz and Lindenmayer [1990]; Ochoa [1998]), rules can be parameterized in order to specify musical transformations. In Listing 2.12, brackets prepended to rule names contain musical transformation instructions, namely pitch transpositions and a rhythmic augmentation.

```

1 TWINKLE => {NOTE, NOTE, (~5) NOTE, (~5) NOTE, (~6) NOTE, (~6) NOTE, (~2) (~5) NOTE}

```

Listing 2.12: *SARAH* representation of *Twinkle, Twinkle, Little Star*, mm. 1–2, containing musical transposition and augmentation transformations specified in brackets. The terminal NOTE represents a middle C (Fox [2006], p. 1).

Musical transformations assume an important role in compositional processes, as will be taken on in Chapter 4. See section 2.3.4 for more detailed discussions on grammar-based music representations. A Graphical User Interface (GUI) for editing, compiling and viewing compositions in the *SARAH* language named *Music Genie* was developed (Fox [2006]). The environment also features automated music generation capabilities, which are covered in Chapter 2.3.7.

2.2 Computers and Creativity: An Overview

The following sections are concerned with the question whether and how creativity can be imitated by means of computer models. Human creativity is both a fascinating and complex research field which has occupied scientists of various disciplines such as neuroscience, psychology, computer science, fine arts, architecture and musicology for centuries. Margaret A. Boden defines creativity as follows:

Creativity [...] is the generation of novel, surprising, and valuable ideas. ‘Ideas’, here, is a catch-all term covering not only concepts and theories but also (for example) music and literature, and artefacts such as architecture, sculpture, and paintings. (Boden [2010], p. 1)

Not only is creativity a crucial human ability to solve problems and generate new artifacts and techniques, but is also “critical for our ability to function and change as a society” (McCormack and d’Inverno [2012], p. viii).

On a scientific level, creativity is hard to grasp, formalize and analyze. Consequently, transferring aspects of creativity to the world of computation is an even more challenging task. However, computers have proven to be effective both as tools in the artistic context and for the algorithmic generation of musical compositions, as will be discussed in the course of this chapter. Depending on the applications, computers play rather passive roles, i.e. merely perform instructions given by users, or can effectively make contributions which might be considered ‘creative’, whereby distinctions between the two paradigms are often blurred.

2.2.1 Computers as Tools in Creative Processes

According to Iannis Xenakis, undreamt-of possibilities open up for composers utilizing computer technology:

With the aid of electronic computers, the composer becomes a sort of pilot: he presses buttons, introduces coordinates, and supervises the controls of a cosmic vessel sailing in the space of sound, across sonic constellations and galaxies that he could formerly glimpse only in a distant dream. (Xenakis [1992](#), p. 144)

In the described scenario, computers are used as a vehicle to explore new spaces of possibilities. However, the composer is still in control of the creative processes and decisions. Another perspective is given in the following quote, in which computers are considered a means of overcoming technical difficulties:

It is our belief that we now need to embrace and support the new forms of creativity made possible by technology across all forms of human endeavour. This creativity is important because it provides opportunities that have not been previously available, and are necessary if we are to address the complex challenges we face in our increasingly technology-dependent world (McCormack and d’Inverno [2012](#), p. viii).

When fulfilling the function of a tool, computers play a rather passive role, which is not considered very creative. In the following, strategies to increase the autonomy of computer systems with regard to the generation of music are discussed.

2.2.2 Computer Models of Creativity

While some ‘magic’ aspects of creativity can only hardly be modeled, other facets of creativity can be simulated well using computers:

There are two distinct types of creativity: the flash out of the blue (inspiration? genius?), and the process of incremental revisions (hard work). Not only are we years away from modelling the former, we do not even begin to understand it. The latter is algorithmic in nature and has been modelled in many systems both musical and non-musical. (Jacob [1996](#))

Even though we are far away from a complete understanding of human creativity, scientists aim to gain further insight into creative processes by iteratively developing models and algorithms to evaluate their possibilities. This methodology yields advanced and complex models, as described in the following:

One of the great challenges for computing is to achieve a fuller understanding of process [sic] and representations which are beyond those that are easily computable or even fully comprehensible by humans. Necessarily, human design of software requires reducing difficult and complex concepts to far simpler abstractions that can be practically implemented, in some cases even ignoring those aspects of a phenomena [sic] that are too complex to express directly in a program. One way to overcome this limitation is to design programs that are capable of initiating their own creativity — to increase their complexity and discover ways of interacting independently of human design. Yet people don't naturally think of creative expression in terms of formal algorithms, leading to a perceived gap between natural creative human expression and computation. (McCormack and d'Inverno [2012](#), p. viii)

Using appropriate models and algorithms, it is in fact possible to construct computer programs which even surprise their creators. In the words of Douglas Hofstadter: “It may not constitute creativity, but when programs cease to be transparent to their creators, then the approach to creativity has begun” (Hofstadter [1979](#), p. 673).

There are a number of reasons why formalizing creativity is complex, especially in the domain of music composition (Nilsson [1971](#); Simon and Newell [1971](#); Ramalho and Ganascia [1994](#)):

- Non-expressible goals: in most cases, it is difficult to exactly define and formalize goals and termination criteria for automated music composition algorithms
- Non-determinism: many musical solutions (compositions) are acceptable for one specific problem

- Objectivity of evaluation: often no universal criteria for the evaluation of generated music can be defined because aesthetic judgements are dependent on individual taste, personal experiences and cultural influences

Especially modelling the aspect of creativity which is **not** to follow rules poses a great challenge, as it contradicts the deterministic mode of operation implemented in computer systems. However, astonishing applications are possible if higher-level models and algorithms are used. Research has shown that some aspects of musical creativity can at least partially be simulated by means of computational models (Boden [1994]; Ramalho and Ganascia [1994]; Jacob [1996]; Cope [2005]; McCormack and d’Inverno [2012]). Furthermore, computer models and algorithms in the field of Artificial Intelligence (AI) have been developed which allow advanced functionality with regard to abstraction, classification, prediction, recombination and efficient heuristics for optimization and search problems. These are discussed in detail in Chapter 2.3.

2.2.3 Constraints and Creativity

A valuable perspective is Boden’s notion of the relation between constraints and creativity:

Constraints, far from being opposed to creativity, make creativity possible. To throw away all constraints would be to destroy the capacity for creative thinking. (Boden [1994], p. 6)

As soon as such constraints exist, creative processes can emerge by *adhering* to the constraints or explicitly *ignoring* constraints. In the domain of music, a specific style can be defined by a set of constraints: “A style is a (culturally favoured) space of structural possibilities” (Boden [2010], p. 2).

Boden identifies three creative strategies which are applicable in constrained scenarios (Boden [2004]):

- **Exploration** of the space of possibilities resulting from the constraints
- **Combination** of ideas within the space, possibly leading to innovative *combinational creativity*
- **Transformation**, which involves pushing the boundaries of the constraint space itself by amending or breaking constraints

In the course of musical compositions, certain constraints defining a musical space of possibilities (such as keys, metric structures, tempo, harmonic progressions, rhythms and melodies) are typically introduced. Within this musical space, composers are able to explore and combine musical ideas. Transformations of the constraint space are possible as follows: If musical structures are constantly present or repeated in regular patterns, listeners raise musical expectations (Huron [2006](#)). If these expectations are not met by suddenly ignoring one or more of the constraints (e.g. by presenting sudden time signature or key changes or new sections introducing significantly new musical ideas), listeners usually consider such musical twists to be creative. Some aspects of these creative strategies can be simulated by means of computer algorithms in combination with suitable music representation models, as will be shown in the course of this dissertation.

2.3 Algorithmic Composition

Algorithmic composition involves using formalizable methods in order to generate music. In this chapter, the historic development of algorithmic composition is summarized. Afterwards, common algorithmic composition approaches are presented, explained and respective advantages and disadvantages are discussed.

Algorithmic composition is perhaps a little bit like religion or politics; one must find one's own path. (Jacob [1996](#), p. 164)

2.3.1 Historical Context

Algorithmic music composition approaches have been employed by human composers since around 1000 AD. It was Guido of Arezzo, who not only played a central role in the development of music notation, but also proposed a system for the “automatic generation of melodies out of text material” (Nierhaus [2009](#), p. 21).

Raimundus Lullus (1232-1315) was the first one to propose a system using components which closely resemble the ones of a modern computer. In his *Ars Magna*, he proposes a system with a structure defined by diagrams (comparable to hardware), a set of definitions forming a knowledge base (comparable to data) and a set of application instructions (comparable to software). Although the system was not explicitly designed for musical purposes, it could have been utilized for algorithmic composition if the generated statements were interpreted in a suitable way (Nierhaus [2009](#), p. 24).

Athanasius Kircher (1602-1680) was a universal scholar concerned with astronomy, mathematics, medicine, music, mineralogy, physics and linguistics. By applying

combinatorial methods in an effort to decipher Egyptian hieroglyphs, he laid the foundations of cryptology (Nierhaus [2009], pp. 24f.). He also developed a system named *Musurgia Universalis* for algorithmic composition utilizing wooden sticks (syntagmas) assigned to three categories. Numbers and rhythmic proportions were engraved into the sticks, resulting in “contrapuntal compositions in the style of the contrapunctus simplex and floridus”, which can be applied to other musical styles (namely church style, madrigal, motet, fugue and monody) in an advanced version (Nierhaus [2009], p. 25). Kircher also proposed the concept of *pitch classes* and abstracting pitches from concrete modes (Nierhaus [2009], pp. 25f.).

Efforts made by Gottfried Leibniz pursued the goal of constructing an all-encompassing language for the representation of all human knowledge and to find solutions for new scientific problems (Glashoff [2003]). He planned on providing a complete encyclopedia of scientific terms, a numeric encoding of all terms in his *lingua universalis* and a logical system (*calculus ratiocinator*) for connecting and processing terms (Nierhaus [2009], pp. 26f.). The algebraic formal logic proposed by Leibniz constitutes an elementary foundation of AI (Glashoff [2003]). According to Leibniz, music relies on mathematical principles as well:

Musica est exercitium arithmeticae occultum, nescientis se numerare animi (Music is a hidden arithmetic exercise of the soul, which does not know that it is counting). (Geiringer [1969]; Nierhaus [2009], p. 28)

In 1679, Leibniz described a mechanical calculating machine using a binary number encoding system. According to the concept, containers with holes for cubes or marbles should be provided, configurable to be closed (representing 0) or opened (corresponding to 1). Depending on the container configurations, marbles either fall into tracks or are blocked by closed containers. Although never actually built, the machine represents a major breakthrough towards binary-based computation (Nierhaus [2009], p. 33).

The first mechanical calculating machine to be built on this basis was the *difference engine* developed by Charles Babbage from 1822 to 1832 (Nierhaus [2009], p. 40). Babbage subsequently designed an advanced machine named *analytical engine* to solve more complex mathematical problems. In his concept, punch cards should be used for programming the machine. Punch cards were at that time used in the textile industry to control looms. Babbage transferred the concept of this binary representation to use punch cards as data carriers for operations (instructions) and numbers (data). Although the analytical engine was never completed before his death, his concepts were later validated and used as a basis for later computer architectures.

Ada Countess of Lovelace, considered to be the first female programmer in the history of computer science, was an assistant of Babbage and was the first to mention the idea of generating music algorithmically in 1843: “the engine might compose elaborate and scientific pieces of music of any degree of complexity” (Collins [2010], p. 300). However, algorithmic composition programs were not actualized at the time (Miranda [2001]).

In 1941, Konrad Zuse accomplished the development of the first programmable electronic computer to process binary numbers named *Z3*, in which mechanical switches were replaced with relays (Nierhaus [2009], p. 50). With his invention, Zuse paved the way for the advent of modern digital computers.

2.3.2 Stochastic Processes

A simple approach to algorithmically generate a musical piece is to concatenate randomly chosen musical symbols (such as notes and rests). Solely random compositions are usually perceived as chaotic and not musically appealing. However, randomness can be utilized as an interesting element in musical decision making.

It is a common notion that randomness is an indispensable ingredient of creative acts. This may be true, but it does not have any bearing on the mechanizability – or rather, programmability! – of creativity. (Hofstadter [1979], p. 673)

A prominent example for aleatory algorithmic composition is a type of musical game which gained popularity in the eighteenth century known as *Musikalisches Würfelspiel* (*Dice Music*), in which pre-composed measures are combined according to numbers produced by rolling dice. Games of the described nature were created by well-known composers such as Carl Philipp Emanuel Bach, Johann Philipp Kirnberger, Franz Joseph Haydn, Wolfgang Amadeus Mozart and Maximilian Stadler (Cope [1996]). An unimaginably large number of unique works can be generated with a small basis of musical material: in a typical dice game with 11 pre-composed measures (corresponding to the numbers 2–12 producible with two dice) and a piece duration of 16 measures, $11^{16} = 45,949,729,863,572,161$ (nearly forty-six quadrillion) works can be produced (Cope [1996], p. 2).

Random combinations are also central elements in Stockhausen’s piano piece *XI* (1956), in which 19 score fragments are arbitrarily combined by the performer (Troche [2018]). Stochastic processes for music composition have also been explored by composers such as Lejaren Hiller (Hiller and Isaacson [1958]; Hiller and Isaacson [1959]) and Gottfried Koenig (Koenig [1971]). A separate musical genre named

stochastic music was created by Iannis Xenakis (Xenakis [1966]; Butchers [1968]). A number of other applications for stochastic procedures are feasible in algorithmic composition (Jones [1981]). In fact, many essential algorithmic composition models are based on stochastic analyses and procedures, as will be shown in the following sections.

An important fact pertaining to randomness in the context of algorithmic composition is that computers are not capable of generating genuine random numbers due to their deterministic architecture. In order to provide ‘real’ random numbers, an external source of randomness has to be connected to the computer (Gentle [2006], p. 2). Typically, external sources are not available and pseudo-random number generators are used, which provide a seemingly arbitrary sequence of numbers. However, this sequence repeats after a certain number of steps, referred to as cycle length (Stewart [2009], p. 626).

2.3.3 Markov Models

Markov models, named after the Russian mathematician Andrey Andreyevich Markov (1856–1922), allow to predict future states of a system depending on a current state (and optionally previous states). Markov models are used to stochastically predict a chain of events, which is why Markov models are often referred to as *Markov chains*. Formally, Markov models include a set of N states (S_1, S_2, \dots, S_N) and a number of discrete points of time $t \in \mathbb{N}_{>0}$. The random variable q_t represents a state S_i at point of time t . In other words, $q_t = S_i$ indicates that the system is in state S_i at time t . The probability of the next state depending on the previous states can be expressed as $P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \dots)$. In *first-order Markov models*, the next state is only dependent on the previous state, as mathematically defined in Equation [2.1] (Alpaydin [2014], p. 418).

$$P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, \dots) = P(q_{t+1} = S_j | q_t = S_i) \quad (2.1)$$

If the *transition probability* from state S_i to S_j is independent of time, the Markov chain is said to be *homogeneous* with *stationary transition probabilities* as shown in Equation [2.2] (Veerarajan [2002], p. 447; Alpaydin [2014], p. 418).

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i) \quad (2.2)$$

The probabilities a_{ij} must be greater than or equal to zero and the sum of all transition probabilities must be 1, as shown in Equation [2.3].

$$\sum_{j=1}^N a_{ij} = 1 \quad (2.3)$$

Furthermore, probabilities for the initial state of the system must be provided in the form of a vector $\Pi = [\pi_i]$, where $\pi_i = P(q_1 = S_i)$ is the probability that the system initially is in state S_i . Likewise, the sum of the initial probabilities must be 1 as illustrated in Equation 2.4 (Alpaydin 2014, pp. 418f.).

$$\sum_{i=1}^N \pi_i = 1 \quad (2.4)$$

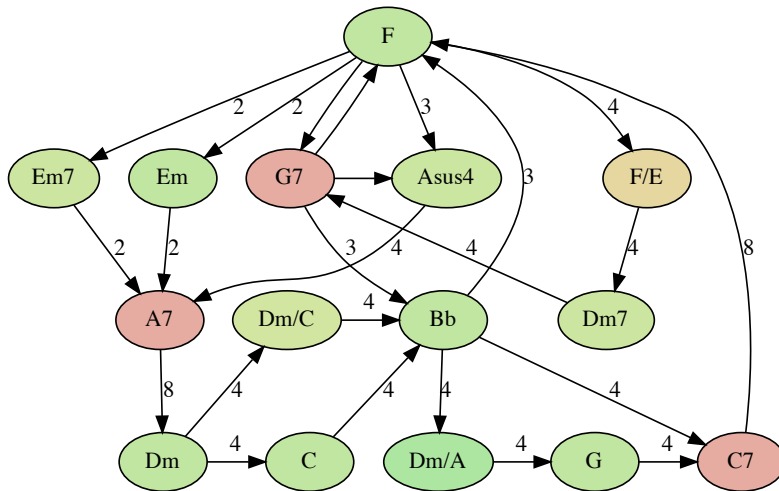
Markov models can be used for the algorithmic generation of arbitrary symbols, which can be musical symbols such as notes, rests, pitches, chords or harmonies. In order to model a Markov chain, a sequence of symbols is analyzed. In MPS, these analyses can be performed with the developed analysis tools described in Chapter 7. The following graphs were generated using these tools.

As an example, the harmonic progressions in *Yesterday* by the Beatles are analyzed. Figure 2.7a depicts a graph visualizing how often specific harmonic progressions occur in the piece. A corresponding Markov model is presented in Figure 2.7b. The probabilities a_{ij} for the corresponding harmonic progressions are computed by dividing the number of absolute progressions depicted in Figure 2.7a by the total number of outgoing connections for each harmony state.

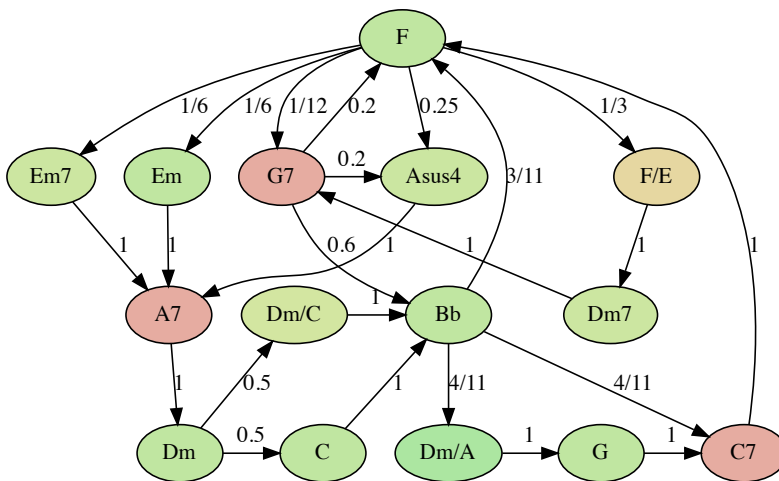
The Markov model presented as a graph in Figure 2.7b can also be represented as a matrix $A = [a_{ij}]$ in which each transition probability is given as a number between 0 and 1. A Markov transition matrix for the harmonic progressions in *Yesterday* is shown in Table 2.4. The probabilities in each row sum up to 1 by definition.

Higher-order Markov chains are used to consider more than one previous state to determine the transition probabilities for the next state (Han and Kobayashi 2007, p. 76f.). Another extension of Markov models are **Hidden Markov Models (HMMs)**, in which the models produce sequences of symbols (such as notes) based on internal states and transitions which are not visible to the observer (Nierhaus 2009, p. 69ff.). One of the first works concerned with the production of musical sequences using Markov chains was contributed by Harry F. Olson, in which he proposed an “Electronic Music Composing Machine Employing a Random Probability System” (Olson 1967). The analog system contained a component designated as *probability matrix decoder*, which was supplied with first and second order Markov chains of pitch sequences analyzed from eleven Stephen Foster songs (Olson 1967, p. 431ff.).

The first known composition to be generated by a digital computer was the *Illiac Suite*, which utilized Markov chains of different orders in the fourth movement titled



(a) Harmonic progression graph of *Yesterday* by the Beatles. Edge labels indicate the number of times a harmonic progression occurs in the piece.



(b) Markov model of harmonic progressions in *Yesterday* by the Beatles. Edge labels indicate the probabilities for the corresponding transition.

Figure 2.7: Harmonic progression graph and corresponding Markov model of *Yesterday* by the Beatles. The colors of the chords represent the consonance (green) or dissonance (red) of the corresponding chord (see Chapter [7.4.4](#)).

Table 2.4: Markov transition matrix of harmonic progressions in *Yesterday* by the Beatles. Each column represents a transition probability from the harmony at the left in the corresponding row to the harmony at the top in the corresponding column.

[illegible]

Experiment Four (Hiller and Isaacson [1958]). Markov models were also employed by Iannis Xenakis in his works *Analogique A*, *Analogique B* and *Syrmos* (Xenakis [1992; Nierhaus 2009, p. 72]).

Brooks et al. conducted Markov analyses up to the 8th order in a corpus of 37 chorales (Brooks et al. [1957]). A work by Ponsford et al. engages in the generation of chord progressions in the style of 17th-century dance music with Markov chains, utilizing improvements of the applied techniques such as the segmentation of the corpus into phrases and bars (Ponsford et al. [1999]). A hierarchical approach is proposed by Allan, in which multiple HMMs are used to generate chorale harmonisations in the style of J. S. Bach (Allan [2002]).

A number of works based on HMMs have been proposed for musical classification tasks (Batlle and Cano [2000; Chai and Vercoe 2001; Pollastri and Simoncelli 2001; Shao et al. 2004; Xu et al. 2005; Scaringella et al. 2006; Cheng et al. 2008; Fu et al. 2011]).

2.3.4 Generative Grammars

The linguistic research foundations of generative grammars were proposed by Noam Chomsky (Chomsky [1957; Chomsky 1959]). A central aspect of both natural and computer languages is *syntax*, which is concerned with the structure of possible expressions in a certain language. The decision whether given sequences of input symbols (which might be words in a natural language) are syntactically correct involves building hierarchical structures called *syntax trees* in order to produce a tree structure conforming to the given sequence of words.

Grammar Representations

Syntax trees are representations of possible choices in a set of so called *rewriting rules*, which make up the grammar of a language. Rewriting rules are typically specified in Backus–Naur form (BNF) introduced by John Backus and Peter Naur in 1959 and contain *non-terminal symbols* on the left hand side, which may be rewritten as a sequence of *non-terminal* or *terminal* symbols, which are given on the right hand side, respectively. An example of BNF rules for simple mathematical expressions is demonstrated in Listing 2.13.

An advancement of BNF is the Extended Backus–Naur form (EBNF) which allows the definition of concatenations (\cdot), alternatives ($|$), optional groups ($[]$) and repetitions ($\{\}$), among other extensions. A grammar equivalent to 2.13 in EBNF is specified in Listing 2.14. Note that non-terminals are not enclosed in triangular brackets anymore. Instead, terminals are surrounded by quotes.

```

1 <expression> ::= <term>
2 <expression> ::= <term> + <expression>
3 <term> ::= <term> * <factor>
4 <term> ::= <factor>
5 <factor> ::= (<expression>)
6 <factor> ::= <literal>
7 <literal> ::= <integer>
8 <integer> ::= <digit>
9 <integer> ::= <digit> <integer>
10 <digit> ::= 0
11 <digit> ::= 1
12 <digit> ::= 2
13 <digit> ::= 3
14 <digit> ::= 4
15 <digit> ::= 5
16 <digit> ::= 6
17 <digit> ::= 7
18 <digit> ::= 8
19 <digit> ::= 9

```

Listing 2.13: Grammar for simple mathematical expressions in Backus-Naur form

```

1 expression ::= term | term, "+", expression
2 term ::= factor | term, "*", factor
3 factor ::= literal | "(", expression, ")"
4 literal ::= integer
5 integer ::= digit | digit integer
6 digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

```

Listing 2.14: Grammar for simple mathematical expressions in Backus-Naur form

Chomsky Hierarchy

Chomsky differentiated between four types of grammars which constitute the *Chomsky Hierarchy*, which is summarized in Table 2.5. With increasing complexity of grammars, more complex mechanisms are required to decide whether a specific input is producible with a given grammar. In terms of the complex hierarchy, these decision processes can be implemented in the form of specific automaton (Kumar 2010, p. 124ff.).

Table 2.5: Chomsky hierarchy

Type	Description	Complexity	Corresponding Automaton
Type-0	Unrestricted Grammar	undecidable (up to infinite)	Non-deterministic Turing Machine
Type-1	Context-sensitive Grammar	exponential	Linear-bounded Automaton
Type-2	Context-free Grammar	polynomial	Pushdown Automaton
Type-3	Regular Grammar	linear	DFA or NFA

Related Work

Systems using generative grammars can either be constructed in a *knowledge-based* fashion, where authors explicitly subdivide given inputs in a hierarchical fashion and formalize these divisions in terms of grammar rules, or by applying *grammatical interference*, meaning that systems derive grammars for given inputs automatically (Nierhaus 2009, p. 91).

Curtis Roads was one of the pioneers pursuing the idea of representing music in terms of grammatical structures (Roads 1979). Inspired by the works of Heinrich Schenker (Schenker 1935), Fred Lerdahl and Ray Jackendoff introduced an essential theory of tonal music based on generative mechanisms (Lerdahl and Jackendoff 1983). The hierarchical structure proposed in their works includes: *Grouping structure*, dividing pieces into sections and phrases, and *metrical structure*, representing a hierarchy of accented and unaccented beats. The strategies of *time-span reduction* and *prolongation reduction* assign individual importance levels to pitches in the above-mentioned structures. In contrast to solely linguistic generative theories, the music theory of Lerdahl and Jackendoff provides mechanisms to differentiate between the *well-formedness* of structural descriptions and the “musical coherence” of the structures, which is examined with so called *preference rules*.

In a linguistic grammar, perhaps the most important distinction is grammaticality: whether or not a given string of words is a sentence in the language in question. A subsidiary distinction is ambiguity: whether a given string is assigned two or more structures with different meanings. In music, on the other hand, grammaticality per se always plays a far less important role, since almost any passage of music is potentially vastly ambiguous. [...] The reason for this is that music is not tied down to specific meanings and functions, as language is. In a sense, music is pure structure, to be “played with” within certain bounds. The interesting musical issues usually concern what is the most coherent or ‘preferred’ way to hear a passage. Musical grammar must be able to express these preferences among interpretations, a function that is largely absent from generative linguistic theory. (Lerdahl and Jackendoff 1983, p. 9)

Acting on the suggestion of Lerdahl and Jackendoff, Temperley and Sleator published a theory for analyzing meter and harmony based on preference rules (Temperley and Sleator 1999). Generative grammars were also used for creating circle rounds, which are perpetually repeatable canons (Rader 1974), and for generating children’s songs and folk songs (Sundberg and Lindblom 1976). Generative grammars have also proved efficient to a certain extent in generating music in the style

of French 18th century dance music and Lutheran chorales (Baroni et al. [1982]). A similar approach was used for the generation of musical phrases in the style of Franz Schubert's *Lieder* cycles (Camilleri [1982]).

Grammar formalisms have also been applied for the representation of Javanese Gamelan music (Hughes [1988]; Hughes [1991]; J. Becker and A. Becker [1979]), the music of a South African ethnic group (Blacking [1970]) and Inuit music (Pelinski [1982]). The studies of North Indian tabla drum music conducted by Bernard Bel and Jim Kippen led to the development of a software system named *Bol Processor*⁸ for analysis and music generation based on generative grammars (Bel and Kippen [1992]).

Grammatical mechanisms have also been used for the description of jazz chord progressions (Steedman [1984]) and as a combined method for rhythms, chord sequences and bass lines (Johnson-Laird [1991]; Johnson-Laird [2002]). Another approach is taken by François Pachet analyzing whether the extent of expectation and surprise is algorithmically detectable and reproducible (Pachet [1999]). For detecting surprising progressions, Pachet applies a compression algorithm (Ziv and Lempel [1978]), which indicates how often specific sequences occur. Rarely occurring harmonic progressions are considered surprising. The system implements a method for grammar inference for which a corpus of compositions is given as input. Several other systems incorporating grammatical interference were proposed (Nevill-Manning and Witten [1997]; Kohonen [1989]). In advanced approaches, the generative capabilities of grammars are combined with other models such as artificial neural networks (see Chapter 2.3.6) and evolutionary algorithms (see Chapter 2.3.7).

Summary

In summary, generative grammars form a basic foundation for analyzing and generating symbol sequences using underlying hierarchical structures. Depending on the complexity of the grammar (see Table 2.5), systems of considerable expressiveness can be constructed. As grammars are a concept rooted in linguistics, several issues become apparent when applying grammars in a musical context, which are related to different semantics and interpretations of music, which are not in all cases matchable (Patel [2008]). An essential aspect which is not covered by generative grammars is vertical structure in music, i.e. processes taking place at multiple simultaneously occurring levels, which is indispensable in the context of polyphonic music (Nierhaus [2009], p. 117).

⁸<http://bolprocessor.sourceforge.net/>

2.3.5 Transition Networks

Transition Networks (TNs) can be described by means of finite automata with individual states (visualized as graph nodes) and transitions (*edges* or *arcs* between states). In Recursive Transition Networks (RTNs), transitions may refer to other networks and furthermore allow direct or indirect references to themselves. A RTN for simple natural language expressions is shown in Figure 2.8.

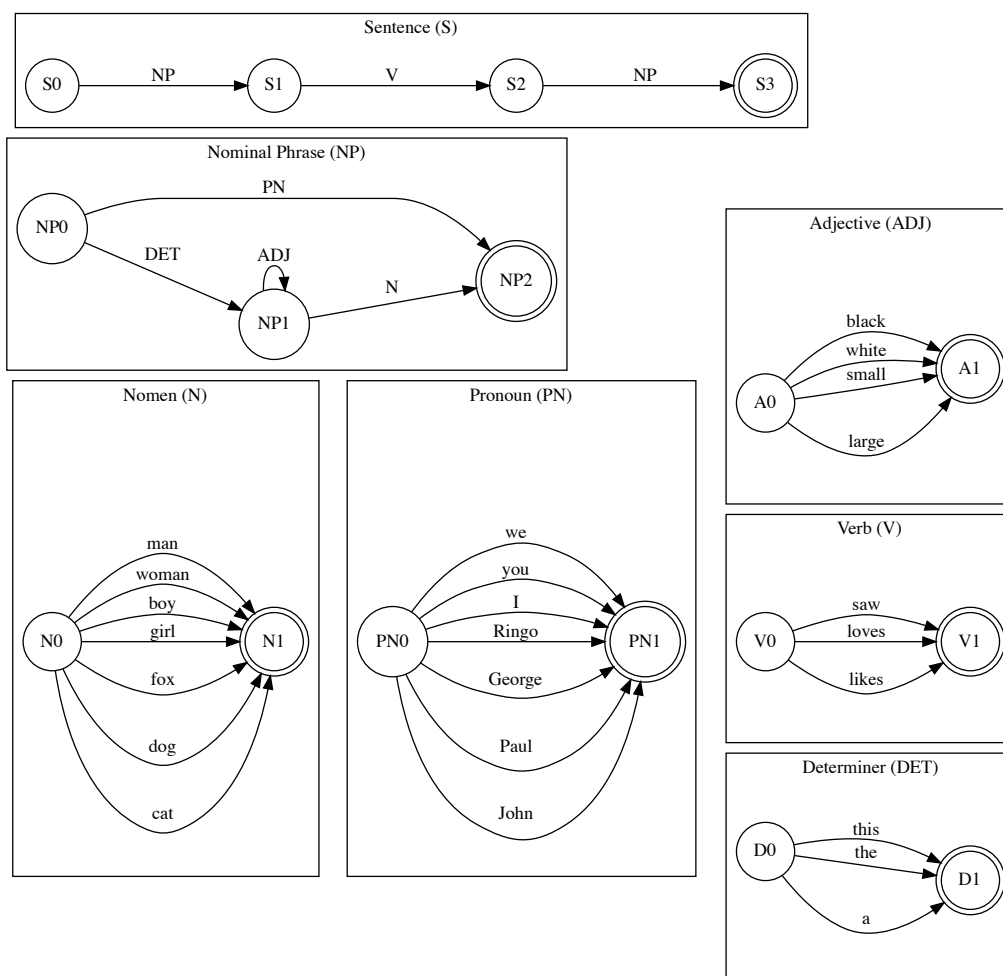


Figure 2.8: Transition network for simple natural language expressions. Adapted from Horáček et al. 2011.

To check if a given sequence of symbols is producible by a specific TN, the network is traversed via corresponding arcs starting at the first state of the top-level network (i.e. sentence in the example in Figure 2.8). This procedure is comparable to the process of deciding whether a given sequence of tokens adheres to a given grammar (see Chapter 2.3.4). The close relation between transition networks and generative

Table 2.6: Transition network types and corresponding grammars

Transition Network Type	Grammar Type	Grammar Name	Corresponding Automaton
Recursive Transition Network	Type-2	Context-free Grammar	Pushdown Automaton
Augmented Transition Network	Type-0	Unrestricted Grammar	Turing Machine

grammars is illustrated in Table 2.6, in which equivalent transition network and grammar types are listed.

An extension to the presented model are Augmented Transition Networks (ATNs), in which additional conditions, actions and jumps can be specified for each transition (Nierhaus 2009, p. 122). This allows the detection and storage of additional context information. For example, correct verb forms could be chosen according to the previously chosen nominal phrase in respect to singular/plural matching or appending an s-suffix if the nominal phrase refers to the third singular person (Horáček et al. 2011).

A prominent music-centered system based on TNS is named Experiments in Musical Intelligence (EMI). It was proposed by David Cope in 1981 and has been continuously enhanced since (Cope 1991; Cope 1996; Cope 2000; Cope 2004; Cope 2009). In EMI, advanced analysis methods are applied to a corpus of compositions in a certain musical style in order to recombine the material to new pieces credibly reproducing the analyzed style (thus referred to as *style imitations*).

EMI performs extensive analyses of the given corpus before transforming and recombining musical fragments similar to the musical dice game described in Chapter 2.3.2. The analysis model developed by Cope formally subdivides a given musical piece in a hierarchical manner, resulting in assignments of formal musical meanings for each musical event. The model is named *SPEAC*, abbreviating the following elements (Cope 2000, p. 58):

- Statement
- Preparation
- Extension
- Antecedent
- Consequent

SPEAC structures for musical pieces can be built based on the following rules specifying possible SPEAC unit successors (Nierhaus 2009, p. 125):

```

1 S => P, E, A
2 P => S, A, C
3 E => S, P, A, C
4 A => E, C
5 C => S, P, E, A

```

Listing 2.15: Rules for producing SPEAC structures

When recombining new compositions, material is not arbitrarily selected, but according to the assigned musical “function” obtained through SPEAC analysis, resulting in musically coherent yet original style imitations.

Further strategies are implemented in [EMI](#) for enhancing style compliance. One of these is to identify musical *signatures*, which are specified as follows:

Signatures are contiguous patterns which recur in two or more works of a single composer and therefore indicate aspects of that composer’s musical style. Signatures are typically two to five beats (four to ten melodic notes) in length and usually consist of composites of melody, harmony, and rhythm. Signatures typically occur between four and ten times in any given work. (Cope [2004](#), p. 109)

In the process of identifying signatures, [EMI](#) searches for musical matches with a certain tolerance. For example, pitch transpositions, interval alterations and rhythmic variations are considered. The level of abstraction is increased until a specified number of signatures is detected (Nierhaus [2009](#), p. 126). While the system generally aims to divide musical compositions into small reusable parts, the structure of signatures must be retained in order to achieve credible style imitations. Therefore, [EMI](#) does not subdivide signatures for recombination (Cope [2004](#), p. 109).

[EMI](#) also detects so called *earmarks*, which are characteristic musical hints appearing “just before or just after important events” (Cope [2004](#), p. 114). With the help of these markers, an example of which are cadential trills (Cope [2004](#), p. 112), the structuring of generated pieces can be further improved.

Unifications are patterns relating to harmonic, thematic and rhythmic structures (Cope [2004](#), pp. 122f.). In contrast to signatures, which capture styles of specific composers in the scope of multiple pieces, unifications capture piece-specific patterns. Initially, [EMI](#) analyzes the number of occurrences, locations and involved variations of the detected unifications. During music generation, the program attempts to recreate similar patterns of unifications in a similar or modified frequency of occurrence (Cope [2004](#), p. 125).

The compositions generated by [EMI](#) are so convincing that Douglas Hofstadter, author of the renowned book *Gödel, Escher Bach: an Eternal Golden Braid*, even withdrew one of his ten prognoses stated in the book in 1979:

Question: Will a computer program ever write beautiful music?

Speculation: Yes, but not soon. Music is a language of emotions, and until programs have emotions as complex as ours, there is no way a program will write anything beautiful. There can be ‘forgeries’ — shallow imitations of the syntax of earlier music — but despite what one might think at first, there is much more to musical expression than can be captured in syntactical rules. There will be no new kinds of beauty turned up for a long time by computer music-composing programs. (Hofstadter 1979, p. 676)

In 1995, Hofstadter heard about EMI and was moved by a mazurka generated by the program in the style of Chopin: “It was *new*, it was unmistakably *Chopin-like* in spirit, and it was not *emotionally empty*. I was truly shaken. How could emotional music be coming out of a program that had never heard a note, never lived a moment of life, never had any emotions whatsoever? The more I grappled with this, the more disturbed I became – but also fascinated.” (Cope 2004, p. 38)

2.3.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are biologically inspired models of interconnected nerve cells, in which propagation and processing of neuronal signals is simulated. The networks can be considered as a mathematical function, in which an input space is mapped to an output space (Priddy and Keller 2005, p. 1). ANNs have proven to be effective for a variety of applications including pattern recognition, computer vision, prediction, optimization and automatic classification (Goodfellow et al. 2016; Nierhaus 2009, p. 205).

Biological Foundations

Artificial neurons are modeled on their biological archetypes, which receive electrical impulses through nerve fibers named *dendrites*. Depending on the inputs, the electric potential of the neuron changes, which is regulated by changing the proportions of sodium and potassium in the cell body (Priddy and Keller 2005, p. 2). If the electric potential exceeds a certain threshold, the neuron “fires”, which results in an *action potential* being transmitted over the so called *axons*, through which the signal reaches *synapses* which in turn stimulate adjacent dendrites of other neurons. According to current research, the human brain contains approximately 86 billion interconnected neurons (Azevedo et al. 2009), enabling the parallel processing of stimuli such as complex visual and acoustic information in time frames of about 0.1 seconds (Nierhaus 2009, p. 205). This is a major advantage of human brains

compared to digital computers, in which each processor (core) executes instructions sequentially (Krawczak [2013], p. 1).

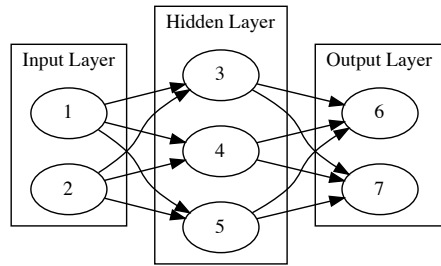
The firing rate of neurons can vary between frequencies close to zero up to about 300 times per second, which is influenced by the neurotransmitter *adrenaline* in biological neurons. Individual firing rates are modeled in ANNs by introducing weighted connections between neurons, where higher weight coefficients correspond to higher firing rates and vice versa (Priddy and Keller [2005], p. 2).

Network Topologies

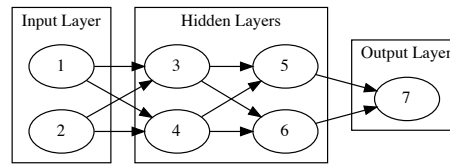
Figure 2.9 shows basic artificial neural network topologies. An elementary type of ANN are so called *feedforward* networks, in which the connections only point in one direction. These networks, which are shown in various complexities in Figures 2.9a, 2.9b and 2.9c and are sometimes referred to as *multi-layer perceptrons*, are typically used for tasks requiring abstraction, e.g. recognition tasks. An outstanding feature of these networks is that they can be trained to perform certain tasks, which simulates an artificial learning process (thus referred to as *machine learning*). Learning capabilities are usually not attributed to computers, but can be imitated by means of ANNs, among other models, to a certain degree and complexity. The most frequently applied learning procedure is *gradient descent*, which minimizes the differences between expected and actual outputs of neurons. A popular method to compute gradients is *back-propagation*, which is a supervised training method adjusting the connection weights of the network after propagating specific input values through the network, comparing expected values with actual values in order to compute differences, and propagating the errors back through the network while adjusting corresponding connection weights (Rumelhart et al. [1986]).

Advanced network topologies were developed for context-dependent applications, which contain so called *context neurons* which are connected recursively, i.e. have an output connection that directly or indirectly connects to the neuron itself as input. This topology facilitates the consideration of previous network states. Well-known examples for recurrent neural networks are *Jordan Nets* (Figure 2.9d) and *Elman Nets* (Figure 2.9e). Other types of neural networks contain one layer of completely interconnected neurons. Examples for such networks are *Self-organizing Maps (SOMs)* (Kohonen [1982]), Hopfield nets (Hopfield [1982]) and Boltzmann machines (Hinton and Sejnowski [1986]), which are suitable for automatic classification and categorization tasks. A Hopfield net is presented in Figure 2.9f.

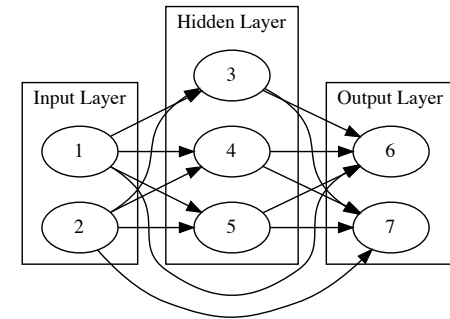
An advanced model suitable for both short-term and long-term dependencies employs elaborated building blocks named *Long Short-Term Memory (LSTM)* units (Hochreiter and Schmidhuber [1997]). These contain so called *memory cells*, *input*



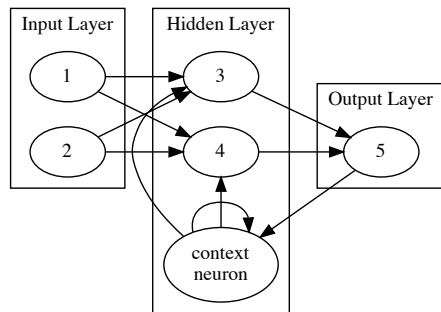
(a) Feedforward neural network with one hidden layer



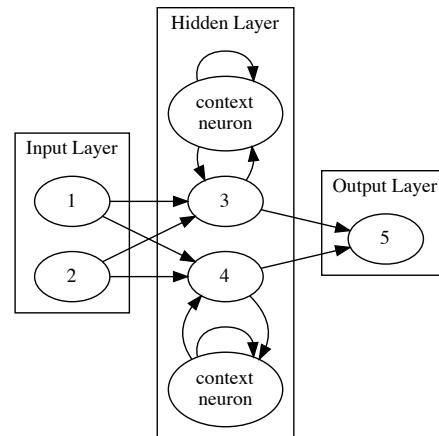
(b) Feedforward neural network with two hidden layers



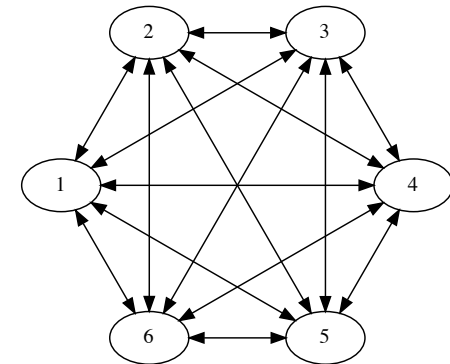
(c) Feedforward neural network in which some connections skip the hidden layer



(d) Jordan network containing one context neuron



(e) Elman network containing context neurons for each neuron in the hidden layer



(f) Hopfield network in which all neurons are interconnected. Neurons are considered both as input and output neurons. Boltzmann Machines have a similar topology with the difference that the contained neurons are designated as *hidden* or *visible* units.

Figure 2.9: Illustrations of various artificial neural network topologies. This figure shows three feedforward network architectures (a-c) and three recursive network types (d-f).

gates, *output gates* and *forget gates*. Neural nets with **LSTM** units were successfully used for natural language processing, handwriting recognition, speech recognition, grammar learning and image tagging (Bianchi et al. [2017](#), p. 25).

Related Work

Several approaches to utilizing **ANNs** for musical applications have been proposed. Peter M. Todd discusses two approaches to represent time-dependent sequences in neural networks: either successive points in time can be represented by means of multiple neurons (meaning time is unfolded in terms of space), or the architecture produces successive notes, whereupon previously generated notes are taken into consideration. Todd uses the latter approach to develop a recurrent Jordan net for generating melodies (Todd [1989](#)). Todd also compares the functionality of **ANNs** to Markov models (see Chapter [2.3.3](#)). He points out that context-dependent neural nets can in principle generate symbol sequences which do not occur in the training corpus in exactly the same succession: “This lets [the network] produce new melodies quite different from those it was originally trained on” (Todd [1989](#), p. 40).

HARMONET A system named *HARMONET* harmonizes melodies in the style of J. S. Bach using recurrent neural networks and a rule-based system (Hild et al. [1992](#)). The system generates a harmony sequence for each quarter beat by taking previous harmonies, melody notes, the position within the current musical phrase and beat stressing into account. The basic model consists of 106 input units, 20 output units and a hidden layer with 70 units. An advanced version was implemented using three neural nets for harmony generation in parallel, which were trained with different temporal window sizes. The actual harmony used is determined by a majority decision of these three nets, and is subsequently processed by a neural net for chord inversions and one concerned with “characteristic dissonances” (Hild et al. [1992](#), p. 271).

CONCERT Another recurrent neural network architecture named *CONCERT* was proposed for predicting melody notes. Both the input and output layer of the network represent the pitch, the duration and the harmonic accompaniment of a note (Mozer [1994](#)). The network successfully generates simple folk melodies, soprano voices in the style of J. S. Bach and waltz melodies. In a listening experiment, the neural net was considered to be superior to a third-order Markov system. A drawback of the *CONCERT* network, however, is the lack of capabilities of structuring music over longer periods of time, which is a common issue in algorithmic composition (Mozer [1994](#); Nierhaus [2009](#), p. 219).

Chorale Harmonization Bellgard and Tsang developed a system for chorale harmonization based on Boltzmann Machines (Bellgard and Tsang 1994), which are variants of Hopfield nets (see section 2.3.6). The system is trained with chorales transposed into a common key. Additional constraints in the form of rules are introduced to ensure a certain number of voices and to avoid voice crossings (Bellgard and Tsang 1994, pp. 291f. Nierhaus 2009, pp. 219f.).

Music Classification Music classification has successfully be implemented using ANNs in the fields of style classification (Dannenberg et al. 1997; Kiernan 2000) and detecting the tonality of given musical sections (Tillmann et al. 2000).

Chord Prediction A neural network for predicting chord progressions was proposed (Cunha and Ramalho 1999). A specific characteristic of the network is the capability to adapt to new chord sequences captured in real time. This is implemented by combining an ANN with a rule-based system, which detects recurrent chord sequences during the performance.

LSTM Networks An LSTM-based system was proposed, which leverages the functionality of memory units to build both short-term and long-term relations for generated compositions (Eck and Schmidhuber 2002a; Eck and Schmidhuber 2002b). The system is capable of generating blues chord progressions and pentatonic melodies. Although LSTMs are capable of considering a larger context than conventional recurrent neural networks, the system “reveals certain limitations in terms of the generation of larger musical sections” (Nierhaus 2009, p. 221).

More recent systems incorporating LSTMs units successfully generated melodies (Coca et al. 2013), melodies in conjunction with chords (Eck and Lapalme 2008), drum patterns (Choi et al. 2016) and piano pieces (Svegliato 2017). LSTM networks are also utilized in Google’s *Magenta* research project which focuses on machine learning in the context of music and art. Successful applications of LSTM networks were reported for piano transcriptions (Hawthorne et al. 2018) and for learning and producing long-term music structures (Roberts et al. 2018). Furthermore, a network was proposed producing not only a score, but also human-like expressive variations in tempo and loudness (Oore et al. 2017).

2.3.7 Evolutionary Algorithms

Inspired by the principles of biological evolution (Darwin 1859), Evolutionary Computation (EC) has become a widely used technique in computer science for solving optimization problems. The employed Evolutionary Algorithms (EAs) apply the

mechanisms of evolution such as *selection*, *crossover* and *mutation* to populations of artificial individuals, which represent solutions to a previously modeled domain-specific problem. **EAs** are applied in a great variety of fields such as logistics, electrical engineering, industrial engineering, chemical and environmental engineering, image processing, mathematics, medicine, physics and arts (Coello Coello and Lamont [2004](#); Romero and Machado [2008](#)).

EAs are commonly used if the number of possible solutions for a specific problem is so large that it is not feasible to enumerate all solutions. This is due to the combinatorial plurality of search spaces, which is often not even manageable using modern computers (Sarker and Coello Coello [2003](#), p. 170). **EAs** can be used to find feasible solutions without checking every possible solution:

A popular way to solve a problem, answer a question, or in general derive a suitable structure to fit a set of requirements, is to cast the problem or question as a *search problem*, a technique central to artificial intelligence. The goal is to look through the entire set of possible solutions to find one that satisfies the original criteria; the trick is to structure the set of all possible solutions so that one does not have to check every solution, allowing the search to complete in a finite amount of time. (Jacob [1995](#))

The basic structure of **EAs** is outlined in Figure [2.10](#). Initially, a first generation containing randomly generated solutions is created. The solutions are evaluated by means of *fitness functions*, which assign comparable ratings to each solution. For some applications, the evaluation process is straight-forward, e.g. if a specific variable is to be minimized or maximized. However, this is not the case for all problems, especially when aesthetic judgement of art or music is required, as will be elaborated in Chapter [8.3.2](#).

As long as the optimization criteria are not specified, evolutionary algorithms generate new generations of individuals by selecting two individuals from the current generation, recombining the chromosomes of these (*crossover*) and performing random modifications in the chromosomes (*mutation*).

For the selection process, various methods are applied. The simplest alternative is *random selection*. However, random selection is not optimal, since individuals with relatively bad fitness are selected with the same probability as individuals with relatively good fitness ratings. Hence, the probability of selection should proportionally increase with higher fitness ratings. This is accomplished with *fitness-proportionate selection*, also known as *roulette wheel selection*. Figuratively, each individual is represented on a roulette wheel, whereupon the area on the roulette wheel is proportional to the fitness of the respective individuals. Accordingly, individuals with

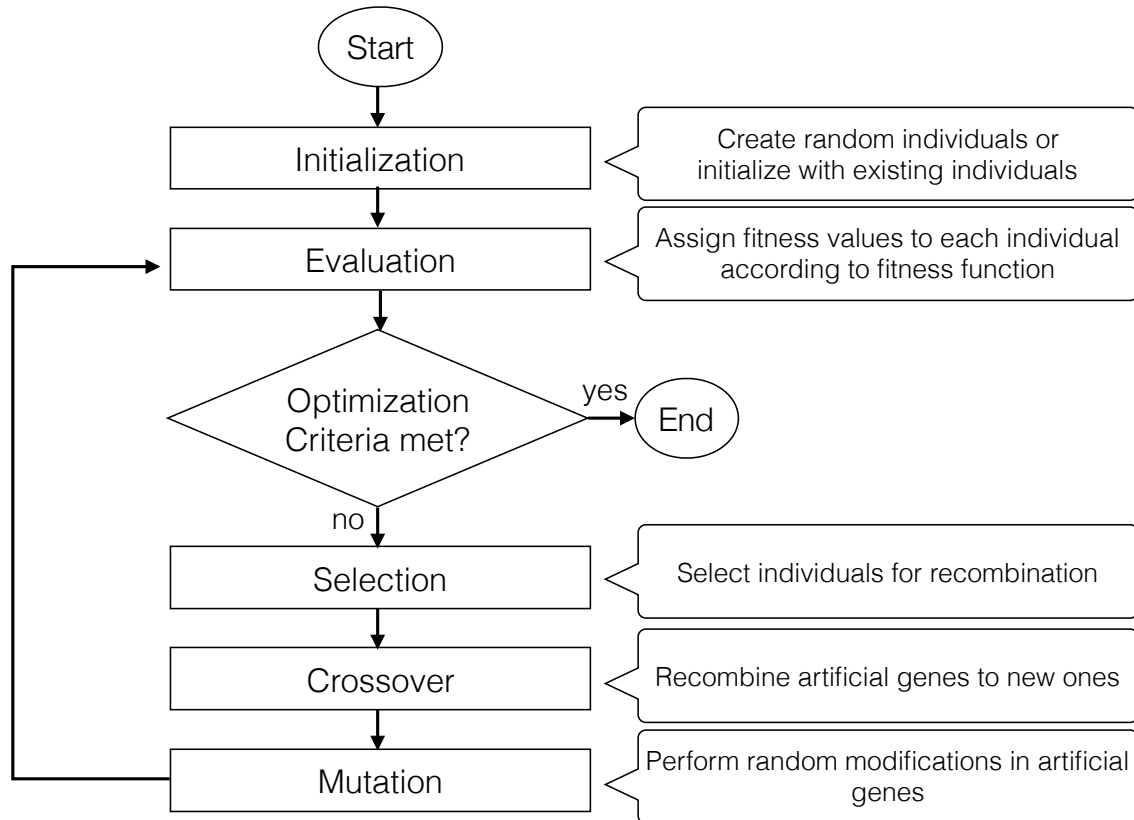


Figure 2.10: Flow chart illustrating the basic structure of an evolutionary algorithm

higher fitness are more likely to be selected and vice versa, as illustrated in Figure 2.11 (Luke 2013, p. 43).

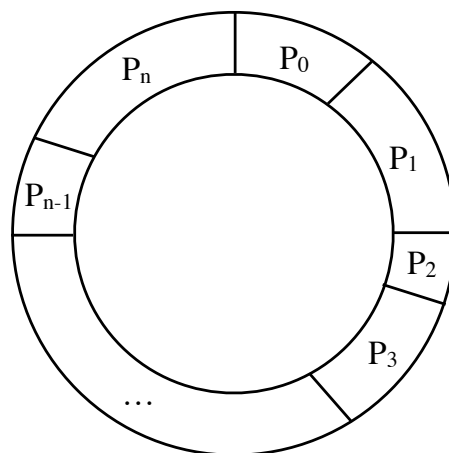


Figure 2.11: Roulette wheel selection in evolutionary algorithms. The selection probability P_i is proportional to the fitness of the corresponding n individuals.

Another frequently used selection strategy is *tournament selection*, where k random individuals are chosen from the current population and the best-rated individual of these is selected. The parameter k is called *tournament size*. Crossover and mutation techniques are introduced in the following sections.

The first ideas for **EAs** were published by Friedberg et al. (Friedberg 1958; Friedberg et al. 1959). Later, four essential approaches were proposed: *evolutionary programming* (Fogel et al. 1966), *evolutionary strategies* (Rechenberg 1973), *genetic algorithms* (Holland 1975) and *genetic programming* (Koza 1992). The differences in the mentioned techniques lie in the details of genetic representations, selection criteria and the implementation of crossover and mutation operations (Sivanandam and Deepa 2007, p. 2). In the following, the differences are illustrated on the basis of the most common paradigms **Genetic Algorithms (GAs)** and **Genetic Programming (GP)**.

Genetic Algorithms

GAs remain one of the most commonly implemented form of **EAs** (Ghanea-Hercock 2003). In **GAs**, chromosomes are usually represented as fixed-length bit strings, i.e. sequences of zeros and ones, which are interpretable as one or more binary numbers (Sivanandam and Deepa 2007, p. 2).

A commonly used crossover operator is *bit-string crossover*, which crosses two chromosomes by swapping a sub-sequence of bits between the bit strings, as shown in Figure 2.12. The beginning of the sub-sequence is referred to as *crossover point* (Sivanandam and Deepa 2007, p. 3).

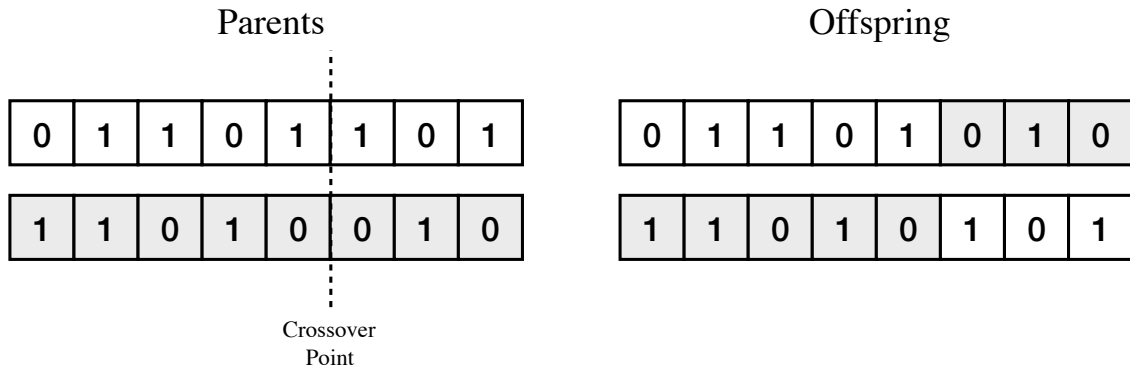


Figure 2.12: Genetic algorithm one point crossover

In the simplest case, mutation can be implemented by inverting a bit in the chromosome, which is illustrated in Figure 2.13. A variety of alternative operations are possible for both crossover and mutation (Sivanandam and Deepa 2007).

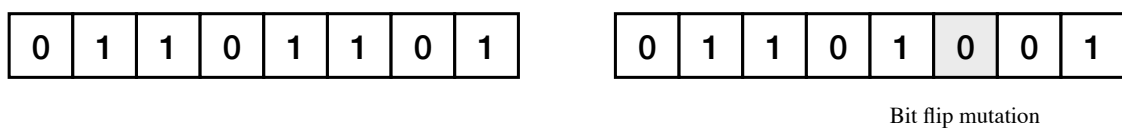


Figure 2.13: Genetic algorithm bit flip mutation

Related Work In the following, essential contributions in the field of music generation using **GAs** are introduced and discussed.

Thematic Bridging Horner and Goldberg proposed a GA-based approach to thematic bridging, which aims at transforming “an initial musical pattern to some final pattern over a specified duration” (Horner and Goldberg 1991). This is achieved by performing modifications on the initial pattern such as adding, deleting, changing or reordering elements. A transition of the initial pattern is constructed by concatenating all intermediate results of the transformations. The chromosomes encode a sequence of operations and operands (e.g. indices of elements to be modified) belonging to the operations.

GenJam John Al Biles developed an interactive system capable of evolving jazz solos using a genetic algorithm. The system maintains two separate populations of bit strings representing phrases and measures. Measures contain strings of 16 different musical instructions, which are *rest*, *hold* or one of 14 possible pitches. Events of the type *hold* can also cross measure boundaries. Phrases contain references to four measures of the measure population, respectively. The first version of the system uses a human mentor for evaluating results (Biles 1994).

The system uses crossover and mutation operators for both populations which are designed in such a way that appealing phrases and measures are not modified in a musically inappropriate way, for example by assuring that crossover will not produce large melodic gaps. The pitches in the measure chromosomes represent degrees on scales, which are in turn chosen in order to match a chord in a predefined chord progression. In this way, Biles assures that all output notes are “correct” in a musical sense (Biles 2007b; Biles 2007a). Regarding disadvantages of the system, the author states:

Two obvious disadvantages to GenJam’s scheme are that notes occur only in eighth note multiples, and there are only 14 pitches to choose from at any one time. These certainly would be a noticeable limitation on most human improvisers, but allowing greater rhythmic and chromatic diversity would increase the string lengths needed for measures, thereby exploding the size of the space searched by the measure population. (Biles 1994, p. 134).

Variations *Variations* is a GA-based composition system developed by Bruce Jacob (Jacob 1995). It consists of three modules: *composer*, *ear* and *arranger*. While the purpose of the composer is exclusively to produce musical material, the

ear is responsible for filtering out unsatisfactory material and the arranger module assembles the remaining phrases to form an orchestrated piece.

Given a set of initial motives, the system creates variations of the motives using a number of defined transformations and combines them to form phrases. These are evaluated by the *ear* module, which is evolved to fit the taste of a human user regarding harmonic progressions.

Melodic Development David Ralley developed a system for melodic development of phrases. The chromosomes contain a start pitch in a scale and an offset-based list of pitches. It produces “interesting melodic variations of initial material” (Ralley [1995]). However, the author mentions that “the acceptability of the final melodic material is entirely up to the user” (Ralley [1995]).

Jazz Music Various works have been proposed concerned with Jazz music generation based on **GAs**. An evolutionary algorithm generating responses to call phrases in jazz pieces by evolutionary means was proposed (Spector and Alpern [1995]). The fitness function was implemented by means of an **ANN** (see Chapter 2.3.6). Yet, the accuracy of the ratings was not reliable due to the fact that too few negative (poorly rated) samples were used during training (Johanson and Poli [1998]).

GAs were also employed for generating jazz melodies (Papadopoulos and Wiggins [1998]). In the proposed algorithm, domain-specific operators are implemented. The proposed fitness function takes intervals, pitch patterns, suspensions, beat positions, note and rest durations, contour and speed into account.

GeNotator *GeNotator* is a generative music system based on **GAs** (Thywissen [1996]; Thywissen [1999]). The genotype structure contains information about choice sequences in a musical grammar. By means of the grammar, musical elements such as “scales, keys, rhythms, phrases and larger compositional structures” can be specified hierarchically. The grammar also supports transformational rules incorporating modifications, for example transpositions, retrogrades and inversions (Thywissen [1999]). The system also features a **GUI** for editing musical grammars, configuring the algorithm and providing feedback.

Grammatical Evolution **Grammatical Evolution (GE)** is a technique in which programs or expressions in a specified language are evolved. To this end, grammar rule decisions are encoded in the chromosomes using integer numbers. A first approach was presented in 1996 (Putnam [1996]). Later, a system which maps geno-

types to AP440⁹ programs was reported (Ortega et al. 2002). GE was also utilized to create novel piano pieces including runs, turns and arpeggios using tonal statistics as fitness criteria (Loughran et al. 2015).

Statistical Approaches Alfonseca et al. propose a generic fitness function for evolutionary music generation based on the *normalized information distance* metric (Li, Chen, et al. 2004; Alfonseca et al. 2006; Alfonseca et al. 2007). It is a universal similarity metric computable by means of *Kolgomorov complexity*, which evaluates whether a given string or string pattern can be generated by means of a computer program which is shorter than the string itself (Li and P. Vitányi 2013). If this is the case, the string is considered *compressible*. Unfortunately, Kolgomorov complexities are not directly computable. However, an estimation function, which is provided in Equation 2.5, was reported (Cilibrasi and P. M. Vitányi 2005), where xy is the concatenation of x and y and $C(x)$ is the resulting length after compressing the string x with compressor C . Based on previous research results, a *LZ77* compressor was chosen (Alfonseca et al. 2006).

$$\hat{d}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}} \quad (2.5)$$

This generic distance function is suitable for computing the similarity of two arbitrary inputs, which was successfully utilized for music classification and clustering algorithms without providing further domain-specific knowledge (Cilibrasi, P. M. Vitányi, and Wolf 2004). When comparing evolved compositions to a given set of “guide-compositions” using this metric, the evolutionary algorithm will automatically produce results with similar features if the distance is minimized (Alfonseca et al. 2006; Alfonseca et al. 2007).

GenDash In *GenDash*, a GA-based program proposed by Rodney Waschka II, the evolutionary process itself is reflected in the generated compositions. In particular, the algorithm does not produce a final result at the end, but rather interprets the developments throughout the whole algorithm run as a musical composition (Waschka II 2007).

Hybrid Approaches *NEUROGEN* is a genetic algorithm capable of producing western four-part harmony works in the style of traditional hymns. Compositions are generated in three stages. In the first stage, only rhythms are evolved, followed by additional pitches in the second stage. Finally, the previously produced phrases are combined in the last stage. A hybrid approach is chosen in which an ANN

⁹AP440 is an auxiliary sound language for IBM’s APL2

trained with the help of existing pieces fulfills the role of a musical judge, yielding promising results (Gibson and Byrne [1991]).

Eigenfeldt reported a system for generating polyphonic rhythms using GAs (Eigenfeldt [2009]). In his system named *Kinetic Engine*, existing compositions are analyzed by means of Markov chains (see Chapter 2.3.3). The goal of the system is not to evolve a single final solution, but to interpret the evolutionary process itself as a composition, which was also proposed in the *GenDash* system. The chromosomes consist of integer numbers corresponding to indices in a database of short rhythmic sequences, each representing a beat. The kinetic engine analyzes rhythms in regard to density, complexity and similarity in order to evolve new rhythms. In subsequent publications, Eigenfeldt reported extensions of the system which also enabled the generation of pitches (Eigenfeldt [2012]; Eigenfeldt and Pasquier [2012]).

Genetic Programming

GP is a variant of an evolutionary algorithm proposed by John R. Koza (Koza [1992]). In contrast to the evolutionary techniques introduced so far, genetic programming uses tree structures for chromosome representation instead of bit strings. Trees are versatile data structures in which a variety of information can be encoded (Storer [2002], pp. 127ff.). In GP, trees are typically used to represent programs. However, these can be abstracted to arbitrary structures in a variety of domains (Yu [1999]).

Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. At the most abstract level GP is a *systematic, domain-independent* method for getting computers to solve problems *automatically* starting from a *high-level statement* of what needs to be done. (Poli et al. [2008], p. 1)

For the illustration of typical GP data structures and operations, a subtree crossover is demonstrated in Figure 2.14. Mutation is typically implemented in the form of *subtree mutation*, in which a random node is replaced with a randomly generated subtree (Poli et al. [2008], pp. 16f).

Related Work GP-based systems related to music processing and generation are introduced in the following sections.

GPmuse *GPmuse* is a GP-based system attempting to generate 16th-century counterpoint music (Polito et al. [1997]). Given a main melody (*cantus firmus*), GPmuse adds new voices consistent with counterpoint-specific rules. Phrases are

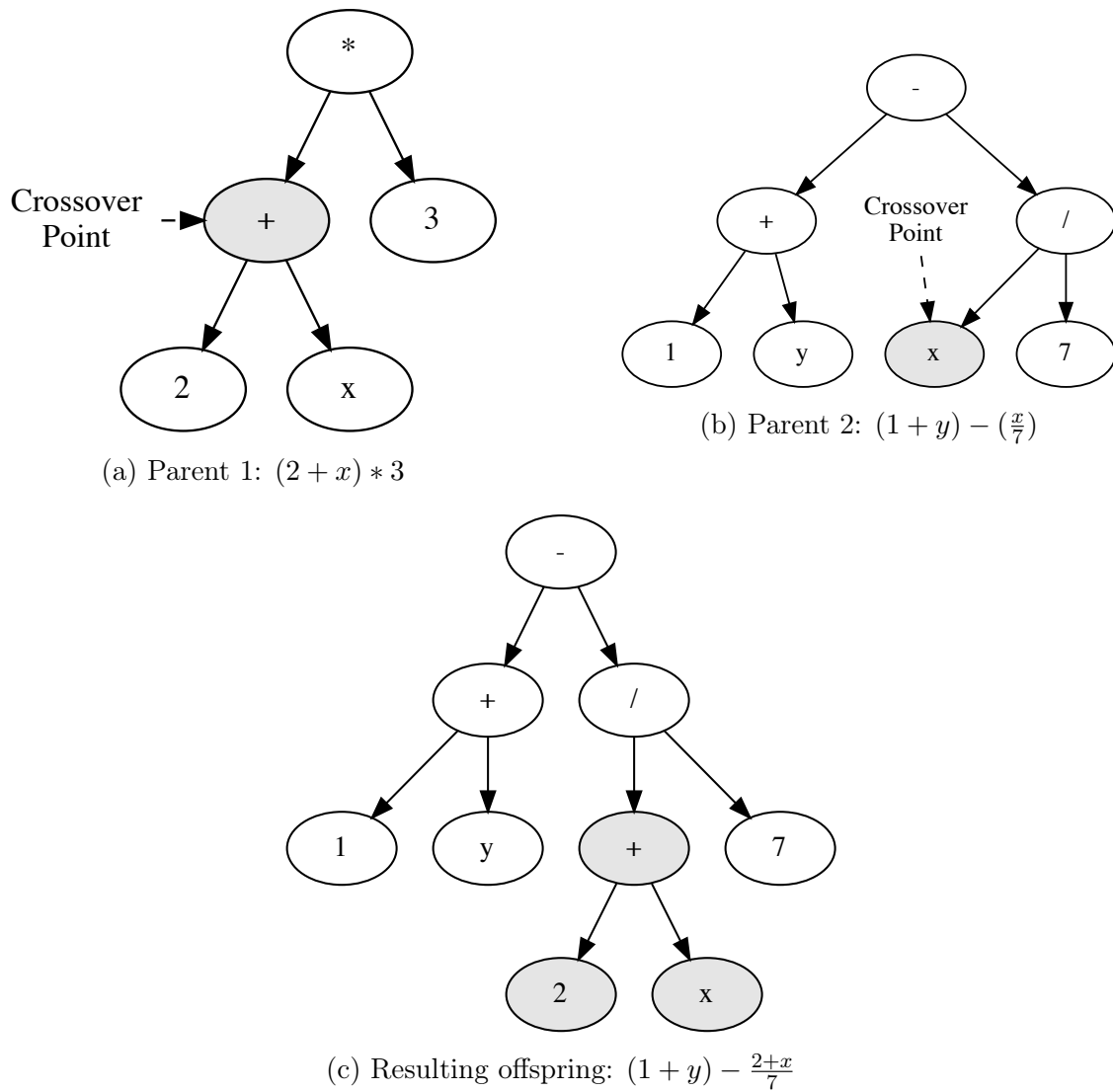


Figure 2.14: Illustration of subtree crossover in genetic programming. Random nodes were selected as crossover points in the two parent individuals, which represent the mathematical expressions $(2 + x) * 3$ and $(1 + y) - \frac{x}{7}$. The offspring results from replacing the subtree defined by the crossover point in parent 2 with the subtree defined by the crossover point in parent 1.

also imitated, i.e. repeated asynchronously and in transposed versions. According to the authors, a surprising outcome of the research project was that the generation of music according to 16th century counterpoint rules partly yielded compositions that sounded rather like 20th century music.

Music Genie *Music Genie* is an environment to process music represented in the *SARAH* language, which was already introduced in Chapter 2.1.9. A GP process for “crossbreeding” compositions is available, in which users effectively take the part of the fitness function by manually selecting favourite compositions. Mutation operators include: inserting random transforms, changing numeric transform parameters, inserting and removing part references, changing brace types, adding new random parts and removing parts. Crossover recombines individual genes (parts) and considers preserving the consistency of referenced parts in the parents, if the referenced parts were also selected to be copied to the new individual. Otherwise, a simple heuristic based on gene positions is used to re-associate references with other existing parts (Fox 2006, p. 3). *Music Genie* produced variations of an extract of J. S. Bach’s *Well-Tempered Clavier*, which was additionally crossbred with the pop song *Wannabe* by the Spice Girls (Fox 2006, p. 4).

Hybrid Approaches Spector and Alpern reported a GP-based approach for generating Bebop jazz melodies. In their work, a so called *case-base* of existing musical works can be supplied by the user, as well as additional “critical criteria” (Spector and Alpern 1994). By creating programs which operate on the input compositions, the system creates new compositions in a “trading four” mode known from jazz improvisation, meaning that short sequences with durations of four measures are given as input, and the system responds with an evolved sequence also comprising four measures. Evaluation criteria are: *tonal novelty balance*, *rhythmic novelty balance*, *tonal response balance* and *rhythmic coherence* (Spector and Alpern 1994). In a proposed system extension, ANNs are used to “automatically induce structural principles underlying a corpus of jazz melodies” to “distinguish reasonable from unreasonable melodies” (Spector and Alpern 1995). The authors report that evaluation results with a single neural network were unsatisfactory. Therefore, a hybrid symbolic/neural “critics community”, each critic having a “complementary type of expertise”, was used for composition evaluation, leading to better results (Spector and Alpern 1995).

GP-Music is an interactive system which combines a GP approach with ANNs (Johanson and Poli 1998). The system is capable of generating small musical sequences, however it is not possible to evolve polyphonic music. Notes are of constant dura-

tion and the pitch space is limited to two octaves. In its basic version, the evolved phrases are evaluated by the user. An extension was developed in which a neural network was trained according to the user’s preferences and therefore could replace the user, making the system autonomous. Automatically rated runs sometimes produced pleasant results. However, the performance was not consistent and the results were not as good as when evaluated by humans (Johanson and Poli [1998](#)).

Manaris et al. proposed an evolutionary framework in which the fitness function relies on *Zipf’s law*. It states that the frequencies of symbols (e.g. words in a book or pitches and note durations in music) are inversely proportional to the ranks of the corresponding symbol according to the formula $p(n) \sim n^{-a}$, where a is close to -1 (Manaris, Vaughan, et al. [2003](#); Saichev et al. [2009](#)). In other words, the most frequent symbol appears about twice as often as the second most frequent symbol, and so on. This correlation was later extended by Benoît Mandelbrot, who observed natural phenomena with distributions ranging from $a = 0$ (random phenomena) to $a = -\infty$ (monotonous phenomena), referred to as *power law distributions* (Mandelbrot [2003](#)). The validity of the Zipf-Mandelbrot law was tested using a corpus of 220 pieces in [MIDI](#) format. The occurrences of the following symbol combinations were analyzed: *pitch*, *pitch class*, *duration*, *pitch & duration*, *melodic intervals*, *harmonic intervals*, *melodic bigrams*, *harmonic bigrams*, *melodic trigrams* and *higher-order melodic intervals* (changes of changes in melodic intervals). The results suggest that the Zipf-Mandelbrot law is a “necessary, but not sufficient condition” for aesthetically pleasing music which can be used as a fitness function in evolutionary algorithms (Manaris, Vaughan, et al. [2003](#); Manaris, Machado, et al. [2005](#)). The system was later extended with an [ANN](#) predicting the “popularity” of musical pieces (Manaris, Roos, et al. [2007](#)).

2.4 Summary

In this chapter, theoretical foundations of music representation, creativity research and algorithmic composition were presented. By combining adequate data models and algorithms, several aspects of musical creativity can de facto be imitated by computers, including:

1. Recognition, categorization and reproduction of musical styles (see Chapters [2.3.3](#), [2.3.4](#), [2.3.5](#) and [2.3.6](#))
2. The ability to learn musical structures and styles automatically (see Chapter [2.3.6](#))
3. Generation of musical sequences based on models of varying complexity (see Chapters [2.3.2](#), [2.3.3](#) and [2.3.6](#))
4. Generation of musical structures with underlying hierarchical models (see Chapters [2.3.4](#), [2.3.5](#) and [2.3.7](#))
5. Exhaustive combination of musical elements and aspects, including heuristics to effectively search large possibility spaces [2.3.7](#)

Virtually all of the proposed systems output music in terms of note and rest sequences. The primary concern of this work is to evaluate whether better outcomes can be achieved by combining more complex music models with the introduced algorithms. In the following chapters [3](#) and [4](#), new music representation models with certain advantages compared to the presented models are introduced, on the basis of which musical applications are developed in chapters [5](#), [6](#) and [7](#). Thereafter, an algorithm imitating selected aspects of creativity is designed in Chapter [8](#).

Chapter 3

Context Layer Composition Model

I do not perform any miracles.
I am merely exact.

— Bohuslav Martinů (Safranek
2013)

Computer programs with the purpose of creating, manipulating or processing musical data require an internal representation of music. Computers utilize abstract data models for representing circumstances of the real world. Internally (i.e. in computer memory and other digital storage media), all instructions and data are represented in terms of numbers (Patterson and Hennessy 2008). In digital computers, a binary number encoding system is employed which uses only two digits, namely 0 and 1. These two digits can physically be represented using electrical signals carrying low voltage or high voltage, respectively (Warford 2017). These digits can be interpreted as integer or floating point numbers, characters, symbols, pixels of images, sound samples and complex object structures (Fenwick 2014; Steinberg et al. 2008). For instance, in music applications these objects could be notes, chords, rests or loudness instructions.

Data models define which types of symbols or elements are needed for an application, how the contained data is represented and how these elements are associated with each other. Nowadays, advanced software tools are available to construct computer data models. Refer to section 4.10.1 for detailed information about the development of the MPS data model.

Most computer systems processing musical data operate on sequential, time-based models that describe music as (one or multiple) sequences of notes and rests. This seems to be an adequate representation when considering that the common form of music notation are scores, which primarily contain sequences of notes, rests and

instructions how these are to be played. While sequential representations are suitable for many computer-aided music processing scenarios and also convenient for performing musicians, these are not sufficient for describing all aspects and relations of musical compositions from a composer’s point of view.

There are myriad time-based models that could be used for generating music, and many composers have explored their possibilities. The fact that a process is formally defined as a function of time does not in any way ensure that the musical outcome will be engaging, nor even that the temporal dynamics can be appreciated by the listener. (Husbands et al. 2007, p. 21)

In this dissertation, two advanced models are proposed which are designed to accommodate contextual musical information of various musical dimensions and relations among these.

3.1 Motivation

Imagine a single sound event or note being isolated from the rest of a composition. This note or sound would not have the same significance anymore. This is because the meaning of each sound only becomes apparent when considering its musical context.

To illustrate this, consider the following thought experiment: imagine a database containing a large corpus of compositions stored in a sequence-oriented digital format. The goal is to find occurrences of a musical fragment within the corpus. For instance, we could search for the motif of Ludwig van Beethoven’s *Symphony No. 5 in C minor*, *Op. 67* shown in Figure 3.1.



Figure 3.1: Ludwig van Beethoven, *Symphony No. 5 in C Minor*, *Op. 67*, Mv. I, motif

Our search query, however, still needs additional clarification. How should the motif that we search for be matched exactly with the material in the corpus? We could think of various alternatives, for example:

1. Look for a perfect match, considering both the rhythm ♩ ♩ ♩ ♩ and the pitch sequence *G G G Eb*.
2. Search only for matches of the rhythm ♩ ♩ ♩ ♩

3. Consider only pitches and look for the pitch sequence $G\ G\ G\ Eb$.
4. Consider the relative distance of the pitches and look for three equal pitches in a row followed by a pitch which is a perfect third lower.
5. Focus on the harmonic context C minor, allowing to interpret the pitches as fifths and a minor third relative to the tonic. Find these intervals either in C *minor* or in an arbitrary other minor key.
6. Interpret the pitches as degrees on a scale $(5\ 5\ 5\ 3)$. This abstraction enables the search of the identified degree pattern in arbitrary keys (including major keys).

A variety of other search query variants and combinations are possible. Depending on the query used, various matches will be found in the corpus. The important question now is: *are the matches found in our corpus musically equivalent with Beethoven's motif?* This can only be answered adequately if the **musical context** of each match is analyzed.

An important musical context, for example, is the metric context. The metric context of the original rhythm is a $\frac{2}{4}$ time signature. By analyzing the rhythmic relation to the time signature, it becomes apparent that the first eighth note is part of a syncopation due to the left out strong pulse on the very first beat in the measure. When positioning the rhythm in a measure at a different point of time or in a measure with a different time signature, the musical meaning of the rhythm changes as well. In conclusion, even if a perfect rhythmic match would be found in the database, this would not automatically imply a musical equivalence. The latter could only be ascertained by analyzing its relation to the metric context.

Similarly, if the exact pitch sequence $G\ G\ G\ Eb$ would be found in another piece, no musical correspondence would directly be implied. Pitches, especially in tonal music, have to be considered in relation to the respective harmonic context. For example, in the key Eb *major* the pitches $G\ G\ G\ Eb$ would have another musical function than in the parallel key C *minor*. In Eb *major*, G has the function of a major third and Eb is considered the tonic. On the contrary, in the key C *minor* G is considered a perfect fifth and Eb represents a minor third relative to the tonic. Harmonic contexts can be nested hierarchically, why in some cases multiple harmonic contexts have to be taken into account. Refer to sections [4.5.6](#) and [4.5.8](#) for a more detailed discussion on scales and harmonic contexts.

In summary, comparing note sequences is not sufficient to find musical analogies between compositions. In order to conduct musically adequate comparisons, other musical context dimensions have to be taken into account. Simple note-based models

are not capable of explicitly encoding all relevant contexts. For this reason, models capable of representing musical context information more explicitly are proposed in this dissertation.

MPS provides the functionality to perform context-dependent search operations in musical corpora, providing musically adequate search facilities as would be required in the proposed thought experiment. This topic is covered in Chapter **6**.

3.2 Introductory Example

The first model proposed in this dissertation is the so called *context layer model*. As an example, a context layer model of the first four measures of the well-known Beatles song *Hey Jude* is presented along with the corresponding score in Figure **3.3**. All model representations in this dissertation follow a consistent color scheme, which is explained in Figure **3.2**.

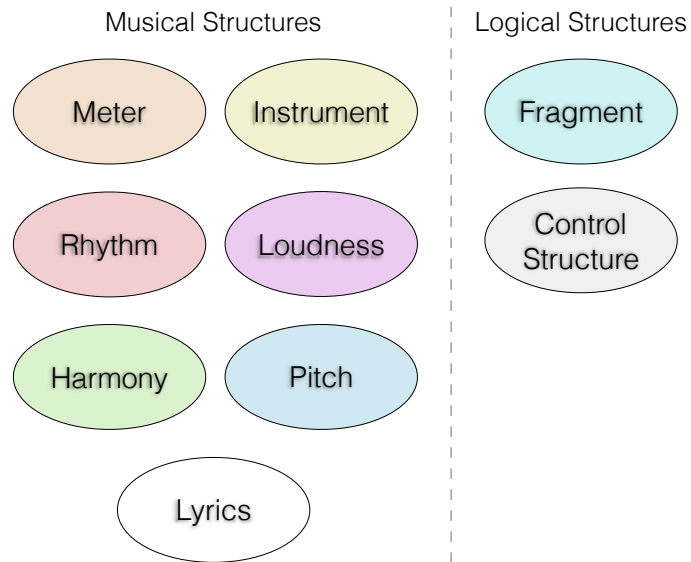


Figure 3.2: Legend illustrating the color scheme of model elements presented in this dissertation

3.3 Model Structure

Context layer models contain one or multiple *streams* which are comparable to voices or parts in scores. Each staff in a score is represented by at least one stream. A single part is divided into multiple streams if it in turn contains multiple voices (e.g. a fugue in four voices might be notated in two piano staves, however the context layer model representation will contain four streams).



(a) Vocal part of *Hey Jude* by the Beatles, mm. 1–4

Stream 1	Instrument (1)	vocals																								
	Meter (5)	4/4 time	4/4 time					4/4 time					4/4 time					4/4 time								
	Tempo (1)	tempo 72																								
	Key (1)	F																								
	Harmony (5)		F					C					C7					F								
	Harmonic Rhythm (5)		1					1					1					1								
	Scale (1)	major																								
	Rhythm (21)	4	4	_4.		8	8	8	4	_2		8	8	4	4	_8		8	8	8	8	16	16	4	_2	
	Degrees (18)	4	2		2	4	5	1		1	2	3	7		7	6	4	5	4	3	2					
	Pitches (18)	C5	A		A	C5	D5	G		G	A	Bb	F5		F5	E5	C5	D5	C5	Bb	A					
	Loudness (1)	loudness mf																								
	Lyrics (18)	Hey	Jude	don't		make	it	bad		take	a	sad	song		and	make	it	bet-	te-	-	r					
	Score Label (1)	Verse																								
		Time (Measures)	1	2					3					4					5							
		Time (Absolute)	-1 / 4	0					1					2					3							

(b) Context layer model of *Hey Jude* by the Beatles, mm. 1–4, vocal part. It represents the composition in terms of individual context layers, including an instrumental context, metric contexts, harmonic contexts, rhythmic contexts, tonal contexts, lyrics and logical contexts. At the bottom of the model, timelines for measures and absolute time are shown.

Figure 3.3: Score and context layer model of *Hey Jude* by the Beatles, mm. 1–4. Corresponding files are available on the accompanying CD under Examples/Compositions/Beatles/HeyJude/Leadsheet (see Appendix [A](#)).

Streams contain individual layers for various musical context dimensions, which are explained in the following sections. The layers in turn contain time-dependent context elements, each of which have a *start time* and a *duration*. Technical details of time representation are set out in section [3.4](#).

3.3.1 Instrumentation Context

The instrument layer specifies which instruments are used at which point of time in a stream. In the context layer model in Figure [3.3b](#), this context never changes and indicates a vocal part.

3.3.2 Metric Contexts

The metric layer contains time signatures, the start times and durations of which represent measures. Note that pieces may commence with an *anacrusis* (also known as *pickup* or *upbeat*) which implies a shortened initial measure (see Chapter [4.5.1](#)). The measure numbers are shown in a timeline at the bottom of the layer model visualization.

The current tempo is determined by an individual context layer. It usually contains elements specifying constant tempi, as shown in the example in Figure [3.3b](#). Multiple constant tempo specifications can be used to model sudden tempo changes. The tempo layer also supports gradual tempo changes to model *accelerando* and *decelerando*.

3.3.3 Harmonic Contexts

The current key context of each stream is given by a correspondent context layer. Note that each stream can have individual keys, enabling multi-tonal music. Another harmonic context is given by *context harmonies*, which usually change more frequently than keys. In the *Hey Jude* example in Figure [3.3b](#), the key remains constant while the context harmonies change in each measure.

3.3.4 Rhythmic Contexts

Rhythm is one of the most essential dimensions in music. This is also reflected in context layer models: rhythmic context layers are obligatory for each stream. The rhythmic dimension defines the proportional durations of the notes or sound events produced by the stream. Another rhythmic dimension is the *harmonic rhythm*, which specifies the durations of context harmonies.

3.3.5 Pitch Contexts

The context layer model in Figure 3.3b also contains context layers regarding pitches, namely **Scale**, **Degrees** and **Pitches**. Often pitches are derived from a contextually suitable scale, on which pitches are addressed using scale degrees. In the example in Figure 3.3b, pitches are derived from the F major scale (which in turn matches the key context) using zero-based scale degrees. For example, the degree 0 will resolve to F, 1 to G, 2 to A, 3 to B \flat and so on. The resulting absolute note names (including the octave) are visible in the pitch context layer. If no octave is specified, the middle octave, which is encoded as octave with number 4 according to **Scientific Pitch Notation (SPN)**, is implied (see section 4.5.5 for more details).

3.3.6 Loudness Contexts

Another musical context layer represents the progress of loudness throughout the musical stream. It contains elements with static loudness instructions such as *forte* or *piano*. Gradual loudness progressions are also supported to model *crescendo* and *decrescendo*.

Another loudness-related context layer, which is not covered in the previous example, accommodates dynamic accents such as *sforzando* (notated as *sfz*, *sf* or *fz*), *sforzando* immediately followed by *piano* (*sfp*), *rinforzando* (*rfz*) or *fortepiano*, forte immediately followed by piano (*fp*).

3.3.7 Lyrics

Vocal streams can contain lyrics as an individual context. By using this layer, syllables can be assigned to individual notes as shown in Figure 3.3b.

3.3.8 Musical Labels

Another context can be supplied in the form of labels for individual parts of a composition. These could be, for example: *Verse*, *Chorus*, *Bridge*, *Solo* for popular music (as demonstrated in Figure 3.3b), or *Exposition*, *Development* and *Recapitulation* for a piece based on a sonata form.

3.3.9 Custom Contexts

MPS provides a number of default context layer types, most of which have been discussed in the previous sections. Yet, the number of layers is not fixed and the model was designed to be extensible in order to accommodate arbitrary new context

layers. For example, new context dimensions for positions in space or the emotional character of certain sections could be added. Refer to section [4.5.10](#) for detailed instructions on how to add custom context layers.

3.4 Time Model

Each context layer model has an internal timeline which by definition starts at $t = 0$ at the beginning of the first full measure. The earliest point in time can become negative in the case of anacruses at the beginning of the piece, as shown in the example in Figure [3.3b](#), which starts at $t = -\frac{1}{4}$ due to the upbeat.

In order to synchronize all individual context layer elements throughout compositions, the start times and durations of each model element must be stored accurately. A first approach was to use floating point numbers. However, the internal representation of floating point numbers can be problematic under certain conditions. Even simple arithmetic operations with seemingly trivial floating point numbers can result in erroneous outputs. As an example, consider the expression $2.0 - 1.1$. When printing the result of this expression using the Java programming language and a current runtime environment, the output reveals inaccuracies regarding floating point encoding: the result is not 0.9 as expected, but 0.8999999999999999. This is due to the commonly used encoding for floating point numbers as defined by the standard IEEE 754 (IEEE [1985](#)).

In Java and many other common programming languages, two types of floating point data types are supported: *float*, corresponding to 32-bit IEEE-754 binary floating point representation, and *double*, using a 64-bit representation offering higher precision. According to the specification, decimal fractions are represented in terms of negative powers of two. This leads to a situation in which negative powers of two, such as $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and so on can be precisely represented. Other decimal fractions, however, may not be representable as a sum of these fractions. In particular, commonly used negative exponent fractions of 10 (such as $\frac{1}{10}$) can not be represented as a finite-length binary fraction (Bloch [2018](#), pp. 270ff.).

While in the musical domain many durations are based on fractions of two, this is by far not true for all rhythmical and durational constructions. For example, when using tuplets (such as triplets or quintuplets) in rhythms, the individual note durations can not be expressed as powers of two anymore. An adequate data type capable of encoding all possible durations in music are fractions. In fractions, two integer numbers (namely *numerator* and *denominator*) are stored separately. Fractions are used as representations for all start times and durations in context layer models. All common arithmetic operations (such as additions, subtractions, multiplications and

divisions) of fractions in turn result in fractions which precisely represent each point of time and each duration of context layer model events. `MPS` uses the Apache Commons Mathematics Library for fraction-related operations¹. Additional code was implemented for fraction comparisons and modulo operations². Refer to section 3.6, in particular Table 3.1, for examples demonstrating the time model.

3.5 Parallel Streams

To demonstrate a model combining multiple parts, a context layer model of the first four measures of Ludwig van Beethoven's *Piano Sonata No. 14 in C# minor, Op. 27, No. 2* is presented in Figure 3.4. The model presented in Figure 3.4b contains two streams, namely one for the arpeggios in the right hand and an individual stream for the accompaniment in the left hand. Note that both streams contain the same information on instrument, meter, tempo key, harmony, harmonic rhythm, scale and loudness layers. However, the streams contain individual rhythm, degree and pitch layers.

3.6 Stream Sequencers and Stream Events

When processing context layer models, mechanisms for navigating through the contents of the model are required. So called *stream sequencers* were developed for this purpose. The sequencers iteratively walk through context layer models while dividing the available context information into logical units, so called *stream events*. The duration of the stream events is determined by a configurable layer in the model, which is set to the rhythm layer by default. By this means, streams can be processed in logically grouped units. This is demonstrated in Figure 3.5, in which the context layer model already presented in Figure 3.3b is segmented into individual stream events. The resulting stream events are listed in Table 3.1.

Stream sequencers also keep track of event start times relative to the beginning of the current measure. The time elapsed in the current measure is referred to as *beat*, starting with 0 on the very first beat in each measure. The beat values for all stream events are listed in the respective column in Table 3.1.

Stream sequencers were implemented for processing single streams³ and multiple

¹<http://commons.apache.org/proper/commons-math/>

²Source code in class `eu.hfm.mps.util.math.FractionUtils` (see Appendix A)

³Implementation in class `eu.hfm.mps.core.streams.sequencer.StreamSequencer` (see Appendix A)

Adagio sostenuto
Si deve suonare tutto questo pezzo delicatissimamente e senza sordini

*sempre **pp** e senza sordini*

(a) Score representation. Edited by RSB and made available at imslp.org under the [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by/3.0/).




















Stream 1	Instrument (1)	instrument piano																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	Meter (4)	2/2 time												2/2 time												2/2 time												2/2 time																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Tempo (1)	tempo 54																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	Key (1)	C#m																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
	Harmony (8)	C#m												C#m/B												A								D/F#								G#7-D#				C#m/G#				G#sus4				G#7-G#3																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Harmonic Rhythm (8)	1												1												2								2								4				4				4				4																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Rhythm (48)	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12

(b) Context layer model representation

Figure 3.4: Score and context layer model of Beethoven's *Piano Sonata No. 14 in C# minor*, Op. 27, No. 2, Mv. I, mm. 1–4. Corresponding files are available on the accompanying CD under `Examples/Compositions/Beethoven/Op27No2_PianoSonataInCSharpMinor` (see Appendix [A](#)).

parallel streams⁴. Processing context layer models by means of segmented stream events is a key functionality of `MPS`, which is utilized for functionality such as model and format transformations (see Chapter 5), context-based music search (introduced in Chapter 6) and music analysis (see Chapter 7).

Table 3.1: Stream events produced by a stream sequencer segmentation shown in Figure 3.5. Note that some context layers which have constant values (namely instrument, meter, tempo, key, harmonic rhythm, scale, loudness and score label) were omitted in this table for more clarity.

#	Start Time	Duration	Beat	Harmony	Rhythm	Degree	Pitch	Lyrics
1	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$			4	C5	Hey
2	0	$\frac{1}{4}$	0	F		2	A	Jude
3	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{4}$	F				
4	$\frac{5}{8}$	$\frac{1}{8}$	$\frac{5}{8}$	F		2	A	don't
5	$\frac{6}{8}$	$\frac{1}{8}$	$\frac{6}{8}$	F		4	C5	make
6	$\frac{7}{8}$	$\frac{1}{8}$	$\frac{7}{8}$	F		5	D5	it
7	1	$\frac{1}{4}$	0	C		1	G	bad
8	$\frac{5}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	C	-			
9	$\frac{7}{4}$	$\frac{1}{8}$	$\frac{3}{4}$	C		1	G	take
10	$\frac{15}{8}$	$\frac{1}{8}$	$\frac{7}{8}$	C		2	A	a
11	2	$\frac{1}{4}$	0	C ⁷		3	B \flat	sad
12	$\frac{9}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	C ⁷		7	F5	song
13	$\frac{10}{4}$	$\frac{1}{8}$	$\frac{2}{4}$	C ⁷				
14	$\frac{21}{8}$	$\frac{1}{8}$	$\frac{5}{8}$	C ⁷		7	F5	and
15	$\frac{22}{8}$	$\frac{1}{8}$	$\frac{6}{8}$	C ⁷		6	E5	make
16	$\frac{23}{8}$	$\frac{1}{8}$	$\frac{7}{8}$	C ⁷		4	C5	it
17	3	$\frac{1}{8}$	0	F		5	D5	bet-
18	$\frac{25}{8}$	$\frac{1}{16}$	$\frac{1}{8}$	F		4	C5	te-
19	$\frac{51}{16}$	$\frac{1}{16}$	$\frac{3}{16}$	F		3	B \flat	-
20	$\frac{13}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	F		2	A	r
21	$\frac{14}{4}$	$\frac{1}{2}$	$\frac{2}{4}$	F	-			

⁴Implementation in class `eu.hfm.mps.core.streams.sequencer.MultiStreamSequencer` (see Appendix A)

[illegible]

Figure 3.5: Segmentation of the context layer model of *Hey Jude* by the Beatles, mm. 1–4, vocal part, into individual stream events. Refer to Table 3.1 for a detailed listing of individual stream events.

3.7 Key Benefits and Versatility of the Model

A key benefit of the model is the clear separation of musical aspects and the time-dependent visualization of how these aspects are combined in the course of musical compositions. In a sense, context layer models provide fine-grained views of musical structures and the interrelations between these, all of which are relevant when composing, interpreting, performing, perceiving and analyzing music. In musical scores, information is presented in a concise, performance-oriented way, which sometimes requires combining information specified in different places, which at times are a long way from each other. This limitation is overcome in context layer stream models since all available information at certain points of time is easily visible by scanning the model visualization **vertically**. Focusing on individual musical aspects is facilitated by the possibility to read relevant layers **horizontally**. While scores are preferable for performing musicians, this thesis demonstrates advantages of the proposed models for musical analysis, format transformations and algorithmic composition.

Furthermore, certain information can not be represented explicitly in traditional scores, such as information about utilized scales, how pitches can be interpreted in terms of scale degrees and specifications of harmonic rhythm. New musical styles and forms, such as electroacoustic music, pose even more difficult challenges with regard to music notation (Burnson et al. 2010).

Streams can either share common information or contain stream-specific information. The model is not limited to the representation of tonal music. For example, for multi-tonal compositions concurrent streams could contain different harmonic and tonal contexts at the same point of time. For compositions which do not rely on tonality, all context layers relating to tonality can be removed. Arbitrary new layers can be added if additional musical dimensions are required. Consequently, the proposed model is suitable for the representation of a number of musical concepts and ideas, which can not in all cases be made visible in musical scores. Note that the model is by design not intrinsically tied to the representation of western music, although the examples in this thesis are limited to this type of music since other systems would go beyond the scope of this dissertation. However, the layer-based approach supports the representation of arbitrary musical systems and dimensions.

3.8 Summary

Context layer models capture various dimensions of music individually and represent these in terms of separate context layers. The approach to describe individual musical aspects separately is a key aspect of the models proposed in this dissertation. This explicitly reveals information which is not easily visible (or even not visible at all) in musical scores. Moreover, the model provides a generic approach to represent music which can not be notated in traditional scores. Context layer models can be processed using stream sequencers, which provide stream segmentations in the form of stream events. In the following chapter, the model is further enhanced and a more concise representation is proposed.

Chapter 4

Context Tree Model and Composition Language

Our language and our song are like
an old tree, continually putting out
new leaves.

— Ralph Vaughan Williams

The layer-based model proposed in the previous chapter is further refined and extended in this chapter. *Context tree models* are concise, redundancy-optimized representations of context layer models, which additionally allow the specification of musical modification processes and algorithmic structures. The model is presented in conjunction with a corresponding description language which enables the textual representation of the models. It facilitates the specification of musical compositions in terms of comprehensible, computer- and human-readable text files.

4.1 Motivation

Musical scores and context layer models (introduced in Chapter 3) usually contain redundant information. Musical information occurring repeatedly throughout the course of a composition can be identified in two variations:

Horizontal Redundancy: Individual context layers potentially contain large amounts of redundant information. These redundancies can easily be identified when analyzing the progression of a single context layer horizontally.

Vertical Redundancy: In context layer models containing more than one stream, individual streams may share information in certain context layers, which can be discovered by analyzing the context layer model vertically.

The main motivation to design an alternative representation of context layer models is to eliminate the necessity of specifying context information more than once. Further motivation is to represent inherent hierarchical structures used in musical compositions.

4.2 Overview

The composition model proposed in this work is capable of representing musical pieces by means of a tree-based structure containing musical context information. Besides musical contexts, the model comprises so called *context modifiers*, *context generators* and *control structures*. All aforementioned elements are explained in the following sections.

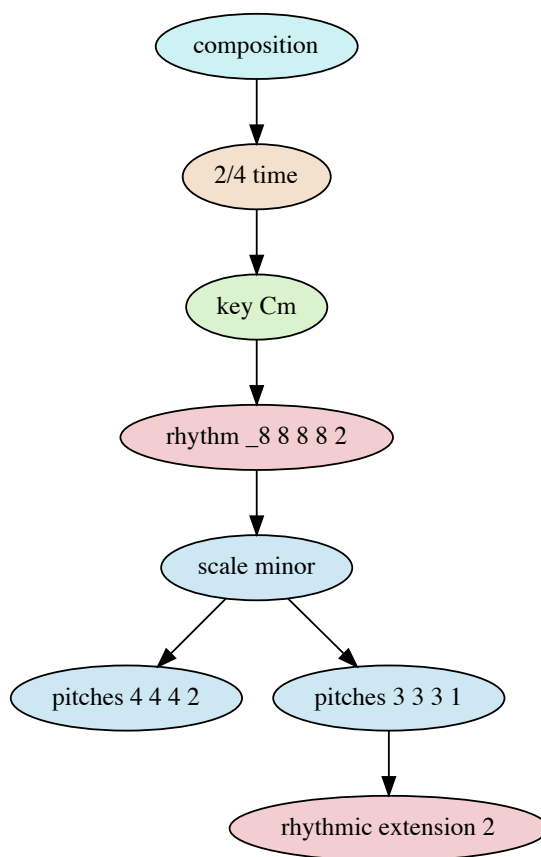
The model was developed in conjunction with a domain-specific composition language. It serves as a medium between the computer and human composers. To humans, model representations in the memory of computers are ultimately confusing conglomerates of numbers. However, humans are very good at comprehending languages. Therefore, a language is a suitable interface between computers and humans. With the help of the developed language, composition models can be transformed to simple textual representations. In this form they become comprehensible and also persistable in text files. The process is also reversible: Human composers can take advantage of the language to compose music by writing simple text files representing musical composition models. These techniques and processes are illustrated in the following sections.

4.3 Introductory Example

As a first example, a context tree model of Ludwig van Beethoven’s world-famous *Symphony No. 5 in C Minor, Op. 67* opening motif is presented along with the corresponding score in Figure 4.1.

Each context tree model defines a root node labeled **composition**. It contains a tree of context elements. In the model shown in Figure 4.1a, a metric context (a $\frac{2}{4}$ time signature) is defined. Below this, the key of the composition (namely *C minor*) is defined by means of a **key** context. On the next tree level, a **rhythm** is specified representing the famous rhythm of the motif, namely $\text{♩} \text{♩} \text{♩} \text{♩}$. The syntax used to describe this rhythm is part of the composition language, which is also introduced in this chapter. Refer to section 4.5.1 for more details.

The pitches to be played are specified in terms of zero-based degrees on the minor scale, meaning that the number 0 represents the tonic C, 1 the note D, 2 the note



(a) Context tree model representation



(b) Score representation

Figure 4.1: Score and context tree model of Beethoven's *Symphony No. 5 in C Minor*, *Op. 67*, Mv. I, mm. 1–4, violin part. Corresponding files are available on the accompanying CD under `Examples/Compositions/Beethoven/Op67_Symphony5/Motif` (see Appendix [A](#)).

E \flat and so on. Note that the context tree diverges below the **scale** node into two separate branches. This is interpreted as follows: First, all contexts between the **composition** and the **scale** node are combined with the left branch (namely the node **pitches 4 4 4 2**), and sequentially combined with the right branch (namely the node **pitches 3 3 3 1** and the **rhythmic extension**). Note that both combined context sets contain the same rhythm, but different pitches. This is a pattern which is very frequently used in musical compositions: the same musical context (in this case a rhythm) is combined with a set of other musical contexts of another type (in this case pitches).

The left branch effectively represents the first two measures of the composition. In this case, the pitches evaluate three times to G and once to E \flat . In the right branch, which represents the rest of the motif, the pitches evaluate three times to F and once to D. The D, however, is rhythmically different from the E \flat in the second measure, as its duration is two half notes instead of only one. This is only a minor modification compared to the original rhythm. In the proposed model, it is not required to define a new rhythm. Instead, only the modification of the current rhythm can be specified, which is done with a so called *context modifier* named **rhythmic extension**, which doubles the duration of the last half note.

The context model in Figure 4.1a can also be represented in terms of a simple text file in the corresponding domain-specific language which was developed in the scope of this dissertation. Compare the syntactical representation in Listing 4.1 with the graphical model in Figure 4.1a.

```

1 composition
2 {
3     time 2/4, key Cm
4     {
5         rhythm _8 8 8 8 2
6         {
7             scale minor
8             {
9                 pitches 4 4 4 2
10                pitches 3 3 3 1
11                {
12                    rhythmicExtension 2
13                }
14            }
15        }
16    }
17 }
```

Listing 4.1: Syntactical representation of the context tree model of Beethoven’s 5th Symphony motif shown in Figure 4.1a

4.4 Key Concepts

The preceding example demonstrates a few key aspects of the model:

- Compositions can be expressed as combinations of musical aspects or contexts in various constellations.
- Contexts can be represented in the form of a tree. When combining different tree branches sequentially, the tree can be interpreted as a musical composition.
- The tree structure is suitable for representing music in a compact form avoiding redundant information. In the example, the `rhythm` is used twice but must be specified only once.
- If a musical structure (such as a rhythm) is modified in the course of a composition, the modification process can be specified rather than defining a new rhythm instance.

These concepts, among other mechanisms, are further elaborated in the following sections.

4.4.1 Hierarchical Structures

Musical compositions are usually to some extent organized and perceived in hierarchically arranged units.

Regardless of how planned or unplanned a compositional form is, musical structures tend to be hierarchical. (Biles [2007a](#), p. 30)

Compositions can generally have multiple hierarchical levels of organization.

Music is not a mere linear sequence of notes. Our minds perceive pieces of music on a level far higher than that. We chunk notes into phrases, phrases into melodies, melodies into movements, and movements into full pieces. (Hofstadter [1979](#), p. 525)

Hierarchical structures for music representation have been explored by Lerdahl and Jackendoff in their *generative theory of tonal music* (Lerdahl and Jackendoff [1983](#)) and *tonal pitch space theory* (Lerdahl [2001](#)). Another related contribution is the *generative syntax model* by Rohrmeier (Rohrmeier [2011](#)). These works suggest that listeners unconsciously build hierarchically organized structures when experiencing

music, which can be represented in the form of tree structures. However, “to date, no neurophysiological investigation has tested whether individuals perceive music cognitively according to tree structures” (Arbib [2013](#), p. 144).

The hierarchical nature of context tree models is used for multiple purposes:

- Utilizing hierarchical structures is useful for **logical and graphical grouping** and for improving the clarity and readability of musical context tree models. The number of hierarchy levels in MPS context models is not limited, which allows modelling musical context trees with arbitrary complexity.
- The hierarchy level of contexts in the tree models is also used to express the **scope** of the corresponding contexts. Generally, the higher a context is located in the tree, the more global its impact on the musical composition is.
- Furthermore, **hierarchical relations between individual contexts** can be modeled. For example, pitches can be put in a local harmonic context such as a chord or harmony, which in turn can be related to a local and/or global key.

4.4.2 Inheritance

A very effective way of avoiding redundant information is to harness a technique commonly used in object-oriented software development called *inheritance*. It involves defining hierarchical dependencies between object types in order to utilize already existing properties and/or functionality from another object type. To illustrate, consider the diagram in Figure [4.2](#).

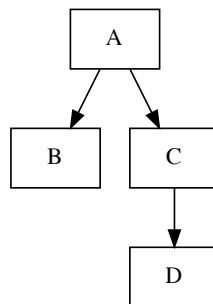


Figure 4.2: Simple class hierarchy

In this example, a type (also denoted as *class*) named *B* inherits from a class named *A*, which means *B* will automatically have the same functionality as *A* and will usually define individual additional functionality. The same holds true for type *C*,

which also inherits from A , but adds different functionality than B . Class D in turn inherits the functionality from C , which in sum yields the functionality of A , C and individual additional functionality defined by D .

In [MPS](#), the principle of inheritance is applied to musical context tree models. This is illustrated using a musical example. Consider the score of the beginning of Queen’s *Bohemian Rhapsody* shown in [Figure 4.3](#).



Figure 4.3: Queen, *Bohemian Rhapsody*, mm. 1–4. Corresponding files are available on the accompanying CD under `Examples/Compositions/Queen/Bohemian Rhapsody` (see [Appendix A](#)).

The score in [Figure 4.3](#) contains redundant information. For example, the parts are arranged in a homorhythmic manner, i.e. the rhythms of all four parts are exactly identical except for the end of the third measure. Also, the lyrics for all parts are exactly identical. In traditional scores, the composer or arranger has no other choice but to write the same rhythms and syllables all over again. In [MPS](#) context tree models, however, the rhythm and the lyrics have to be specified only once and can be reused using various techniques. One of these techniques is inheritance, which is demonstrated by means of the context tree model in [Figure 4.4](#) representing the first two measures of the piece.

The inheritance hierarchy is made visible by arrows and by the positions of the context nodes. Arrows are interpreted as ‘all inherited contexts are passed on to

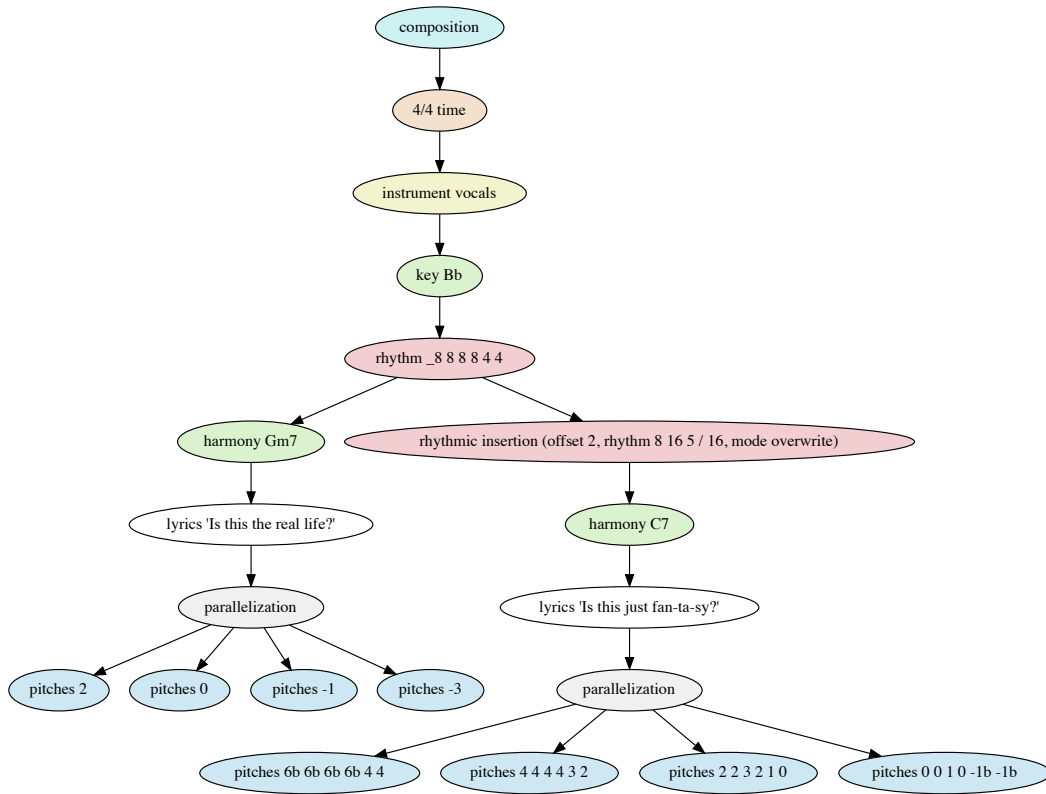


Figure 4.4: Context tree model of Queen’s *Bohemian Rhapsody*, mm. 1–2. Common contexts for both measures are a $\frac{4}{4}$ time signature, vocal instrumentation, the key B \flat major, and the rhythm $\gamma \text{ ♩ } \text{♩} \text{ ♩} \text{ ♩}$. The harmony context in the first measure (which is modeled in the left subtree) is defined as G minor with an added seventh. Furthermore, the lyrics “Is this the real life?” and individual pitches for the four voices are defined. The rhythm of the second measure is derived from the globally defined rhythm by means of a rhythmic insertion. Below, the harmonic context (C 7), the lyrics and individual pitches are specified for the second measure.

the node in direction of the arrow’. Inheriting nodes will normally be drawn on the next hierarchy level, which implies a lower position in the graph visualization. This way, the instrument (vocals), the key (B \flat major), the rhythm, the context harmony (Gm 7) and the lyrics (“Is this the real life?”) are combined and passed on to the left **parallelization** node. It has four child nodes, which produce the four individual vocal parts of the first measure. They have different pitches, but have all the previously enumerated contexts in common. Using inheritance, all common contexts have to be specified only once, which is a major advantage of context tree models.

The same technique is used in the second measure, which inherits common instrument, key, base rhythm, context harmony and lyrics contexts. Note that further optimization methods are available, which are explained in the following sections.

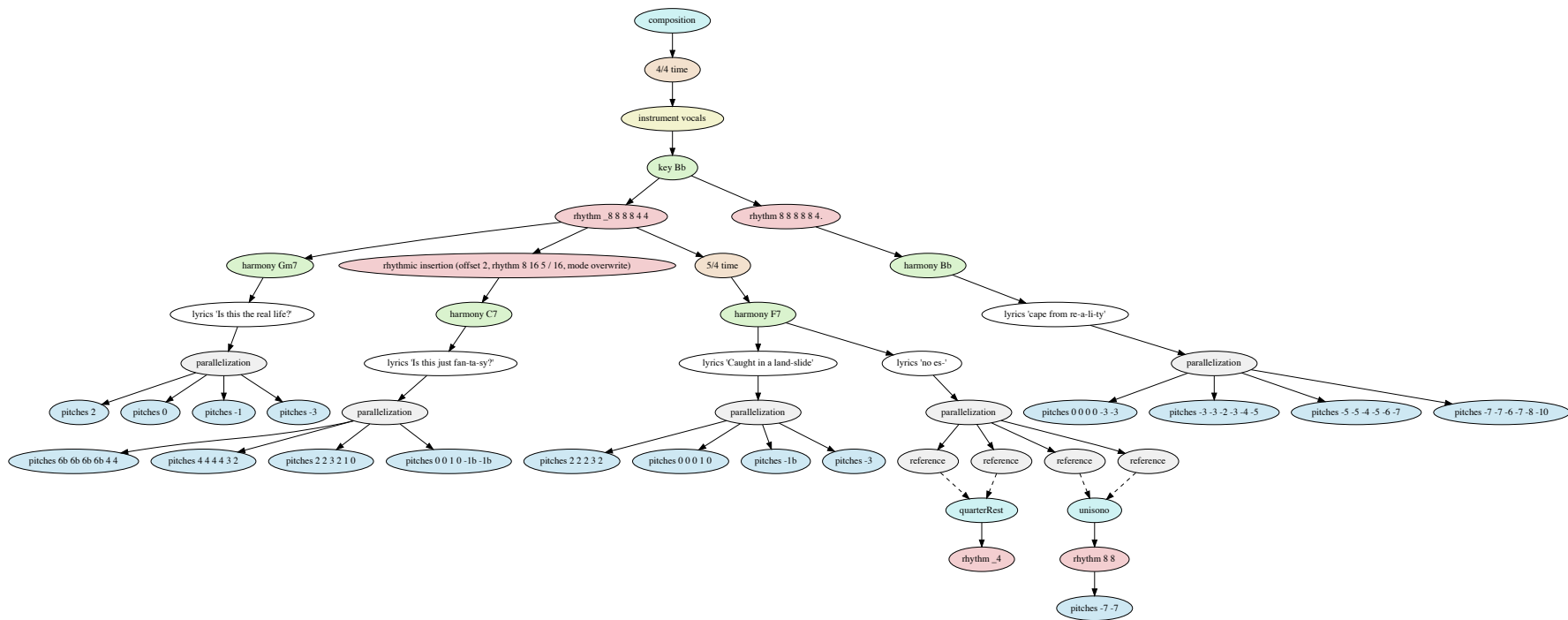


Figure 4.5: Context tree model of Queen's *Bohemian Rhapsody*, mm. 1–4. The two leftmost subtrees are identical to the context tree model in Figure 4.4. The $\frac{5}{4}$ time signature overwrites the globally defined $\frac{4}{4}$ time signature at the top of the tree temporarily, namely in the third measure. In the fourth measure, which corresponds to the rightmost subtree, the global $\frac{4}{4}$ time becomes operative again. Compare with the score in Figure 4.3

4.4.3 Polymorphism

Another model concept inspired by object-oriented programming is *polymorphism*, which allows overriding (and also extending) particular parts of inherited functionality. In context tree models, this concept can be used to overwrite contextual information. To elaborate, another context tree model of *Bohemian Rhapsody* is shown in Figure 4.5, this time containing context information of the first four measures of the piece.

The time signature change in the third measure (see score in Figure 4.3) is modeled using a polymorphic construction. In the model in Figure 4.5, the $\frac{5}{4}$ time signature context is positioned on a lower hierarchy level than the $\frac{4}{4}$ time context near the root of the tree. The metric context $\frac{5}{4}$ effectively overrides the $\frac{4}{4}$ context temporarily (namely for one measure). After the subtree of the $\frac{5}{4}$ measure is processed, the main $\frac{4}{4}$ time signature becomes operative again. This technique can be applied to any musical context. For instance, temporary changes regarding meter, tempo, instruments, rhythms, pitches and harmonic contexts can be modeled.

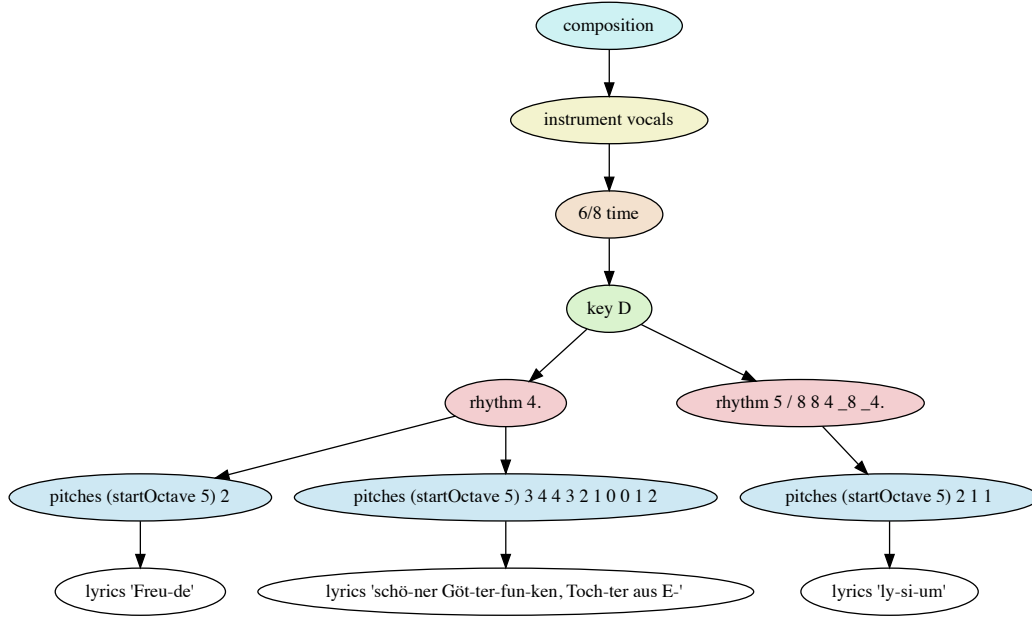
Note that this representation has additional value compared to a purely sequential representation. In the context tree model, it is directly visible that the $\frac{4}{4}$ meter is of higher importance in the composition than the $\frac{5}{4}$ meter. In fact, it becomes apparent that the $\frac{5}{4}$ meter is only used in terms of a temporary ‘excursus’ from the standard meter of the piece and is used in a subsidiary manner.

4.4.4 Auto Expansion

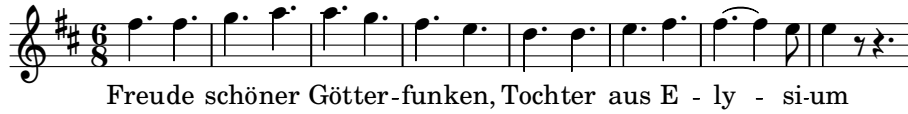
When combining contexts defining musical sequences (e.g. rhythms, pitches or lyric syllables), these sequences do not necessarily need to have the same length. If the number of available rhythm notes, pitches and syllables does not match, the system will automatically apply a so called *auto expansion*. The consequence is that shorter sequences are automatically repeated until the longest sequence is consumed completely.

An example model and the corresponding score of Beethoven’s *Ode to Joy* chorus from the *Symphony No. 9* is shown in Figure 4.6. Syntactically, this model can be represented in a language form as demonstrated in Listing 4.2.

In Figure 4.6a, musical sequences of different lengths are combined; in particular, the leftmost subtree combines the rhythm 4. (i.e. a dotted quarter note) with a pitch on the third scale degree (zero-based, i.e. 2) and the two lyric syllables **Freu-de**. While the rhythm and the pitch sequence only contain one element, the lyric sequence contains two syllables. The system automatically wraps and repeats the rhythm and the pitches until both syllables are processed. In sum, this results in two dotted



(a) Context tree model representation



(b) Score representation

Figure 4.6: Score and context tree model of Beethoven’s *Symphony No. 9*, Mv. IV, mm. 543–550, soprano part. Corresponding files are available on the accompanying CD under `Examples/Compositions/Beethoven/Op125_Symphony9` (see Appendix [A](#)).

quarter notes, both with the same pitch, but with different syllables.

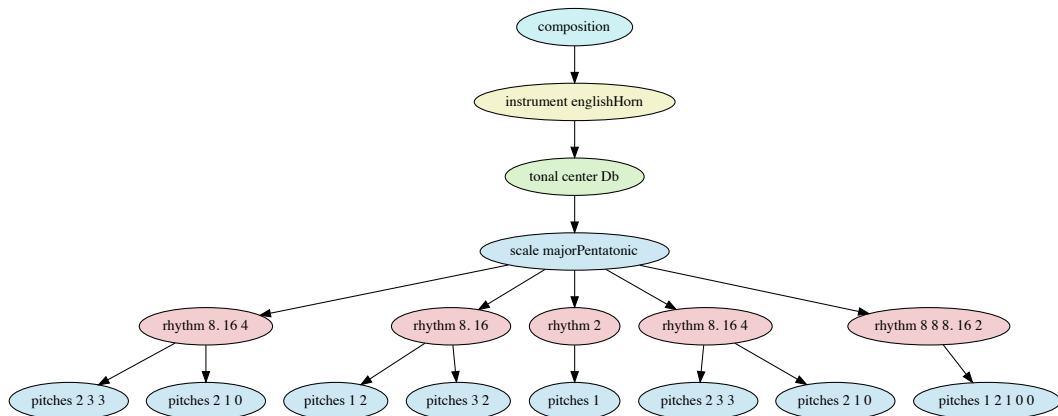
Auto expansion was also used in the previous *Bohemian Rhapsody* example in Figure [4.4](#), in which the rhythm `_8 8 8 8 4 4` is combined with the lyrics “Is this the real life?” and multiple pitch contexts for individual parts, namely `pitches 2`, `pitches 0`, `pitches -1` and `pitches -3`. The rhythm contains six rhythmic notes, the first of which is a rest, leaving five assignable notes for syllables and pitches. The lyrics contain exactly five matching syllables. The pitch contexts, however, contain only one pitch each. Therefore, the pitches are repeated until the rhythm and the lyrics are consumed completely. Using auto expansion, redundant musical sequences can be represented in an effective way, providing yet another useful compression method for context tree models. An automatic algorithm for context tree model compression is proposed in Chapter [5.7](#).

```

1 composition
2 {
3     instrument vocals, time 6/8, key D
4     {
5         rhythm 4.
6         {
7             pitches(startOctave 5) 2
8             {
9                 lyrics "Freu-de"
10            }
11            pitches(startOctave 5) 3 4 4 3 2 1 0 0 1 2
12            {
13                lyrics "schö-ner Göt-ter-fun-ken, Toch-ter aus E-"
14            }
15        }
16        rhythm 5/8 8 4 _8 _4.
17        {
18            pitches(startOctave 5) 2 1 1
19            {
20                lyrics "ly-si-um"
21            }
22        }
23    }
24 }
    
```

Listing 4.2: Syntactical representation of the context tree model of Beethoven’s *Ode to Joy* chorus from *Symphony No. 9*.

4.4.5 Modularization using Fragments



(a) Context tree model representation containing two redundant subtrees originating from the rhythms 8. 16 4



(b) Score representation

Figure 4.7: Score and context tree model of the English horn theme from Antonin Dvorak’s *Symphony No. 9 in E minor* (“From the New World”), *Op. 95, B. 178*, Mv. II. Corresponding files are available on the accompanying CD under `Examples/Compositions/Dvorak/Op95_SymphonyNo9_Mv2` (see Appendix [A](#)).

Another technique to avoid redundant information in context tree models is modularization. To this means, arbitrary subtrees can be extracted into so called *fragments*. These are named subtrees which can be referenced from other places in the model. If subtrees occur multiple times in a model, they only have to be defined once in a fragment in order to be referenced at any place they are required.

As an example, consider the score and model of the English horn theme of Antonin Dvorak's *Symphony No. 9 in E minor*, "From the New World", Op. 95, B. 178 in Figure 4.7. This model can be further optimized, as it contains redundant information. Compare measures 1 and 3 in the score in Figure 4.7b, which are exactly identical. The corresponding subtrees, i.e. the subtrees originating at the rhythms 8. 16 4, can be extracted to a fragment and referenced twice, as shown in Figure 4.8.

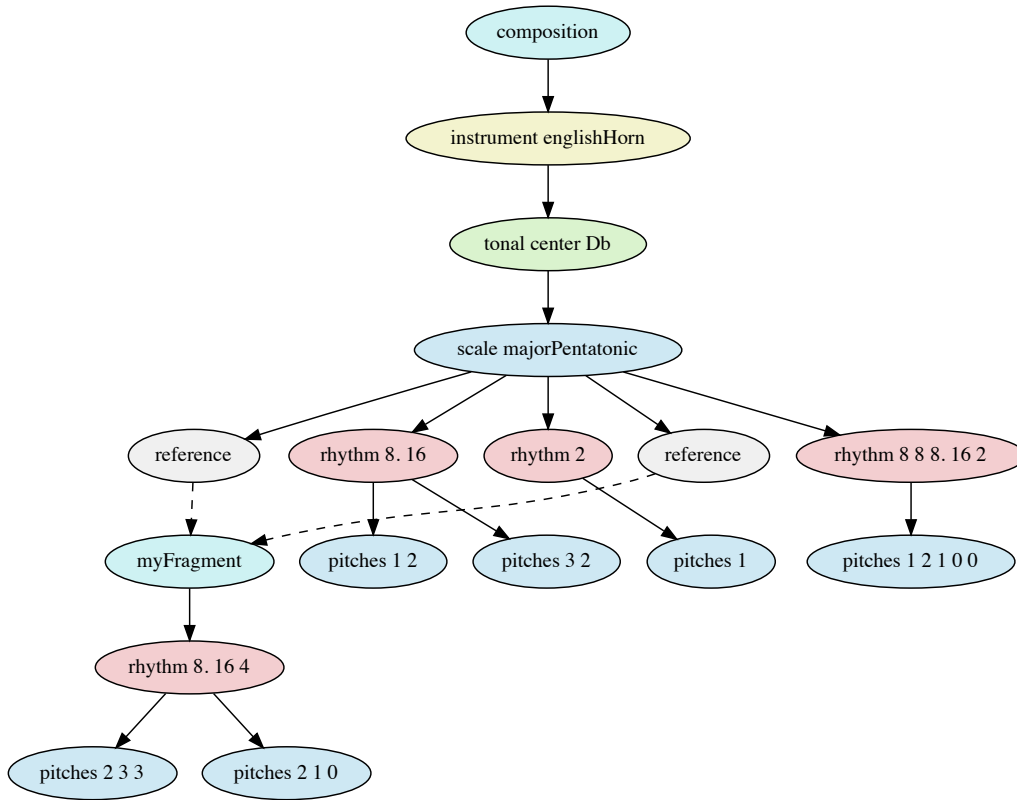


Figure 4.8: Redundancy-optimized context tree model of the English horn theme from Antonin Dvorak's *Symphony No. 9 in E minor* ("From the New World"), Op. 95, B. 178, Mv. II

4.5 Contexts




The basic unit of information in the model are musical contexts. All available contexts are introduced and explained in greater detail in this section. For each context, an equivalent textual representation in the corresponding composition language is presented.

4.5.1 Rhythms

Rhythm is one of the most central aspects in music. While at first glance its purpose is only to define “durational proportions” of sound events (Cooper and Meyer 1960, p. 1), the impact and the presence of rhythmic proportions inherently exist in multiple dimensions of music. Not only are rhythms present defining the duration of sound events or notes, but also for other musical aspects: the *harmonic rhythm*, for example, concerns the duration and length of contextual harmonies, and can be different from the rhythm of the notes being played.

Sequences of rhythmic notes (i.e. durations of notes and rests) are specified using a simple syntax which was developed as part of MC²L. Refer to Table 4.1 for detailed explanations of basic note and rest duration syntax variants.

Table 4.1: Note and rest duration syntax

Syntax	Example	Description
n (integer number)	2 4 8 16 	Integer literals are interpreted as reciprocal duration, e.g. 4 represents a quarter note, 8 an eighth note etc.
_ (prefix)	_2 _4 _8 _16 	Prefix to indicate that the following duration is to be interpreted as a rest duration.
! (suffix)	2! 	Integer literals followed by an exclamation mark are not interpreted as reciprocal duration but as literal duration, e.g. 2! specifies a duration of two whole notes.

Continued on next page

Table 4.1 – *Continued from previous page*








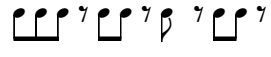


Syntax	Example	Description
. (suffix)	8. 16 8.. 32 	Dots are used as suffixes to extend the preceding note or rest duration with a factor of 1.5. Multiple dots can be used in a row. The total duration of a note with original duration d followed by n dots is computed as follows (Wright 2009, p. 20): $ \begin{aligned} d_n &= d \left(1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^n} \right) \\ &= d \sum_{i=0}^n \left(\frac{1}{2} \right)^i \\ &= d \left(2 - \left(\frac{1}{2} \right)^n \right) = d \left(2 - \frac{1}{2^n} \right) \end{aligned} \tag{4.1} $
n/m (fraction with integer numerator and denominator)	5/4 	Fractional note or rest duration, normally used if the duration can not be expressed as a canonical duration using simple fractions of two and dots.
~ (suffix)	1~4 	Suffix used to indicate that the current note is rhythmically tied to the following note.
(n/m: <durations>) (Tuplet)	(3/2: 8 8 8) 	Specifies a tuplet in which n notes are played in the original duration of m notes. The adjacent example produces an eighth triplet. To compute the resulting durations, the original durations have to be multiplied with the fraction $\frac{m}{n}$. For example, in the case of the triplet, the note durations are multiplied with $\frac{2}{3}$, yielding durations of $\frac{1}{8} \cdot \frac{2}{3} = \frac{1}{12}$ for each of the eighth triplet notes.

Table 4.2 demonstrates how rhythms of well-known compositions are expressed using the composition language syntax.

Table 4.2: Rhythm syntax examples

Syntax and Resulting Rhythm	Description
$_8 \ 8 \ 8 \ 8 \ 2$ 	Ludwig van Beethoven, <i>Symphony No. 5 in C Minor, Op. 67</i> , Motif Rhythm
$4. \ 8 \ 8 \ 8 \ _4$ 	George Frideric Handel, <i>Hallelujah Chorus from Messiah, HWV 56</i> , Motif Rhythm
$2 \ 4 \ 4 \ 4. \ 16 \ 16 \ 4 \ _4$ 	Wolfgang Amadeus Mozart, <i>Piano Sonata No. 16 in C major, K. 545</i> , Opening Theme Rhythm
$8 \ 8 \ 8 \ _8 \ 8 \ 8 \ _8 \ 8 \ _8 \ 8 \ 8 \ _8$ 	Steve Reich, <i>Clapping Music</i> , Rhythmic Motif
$8 \ 16 \ 16 \sim 2$ 	The Beatles, <i>Yesterday</i> , Opening Vocal Rhythm
$4 \ _8 \ (3/2: \ 16 \ 16 \ 16) \ 4$ $_8 \ (3/2: \ 16 \ 16 \ 16) \ 4$ 	Wolfgang Amadeus Mozart, <i>Symphony No. 41 in C major, K. 551</i> , Opening Theme Rhythm

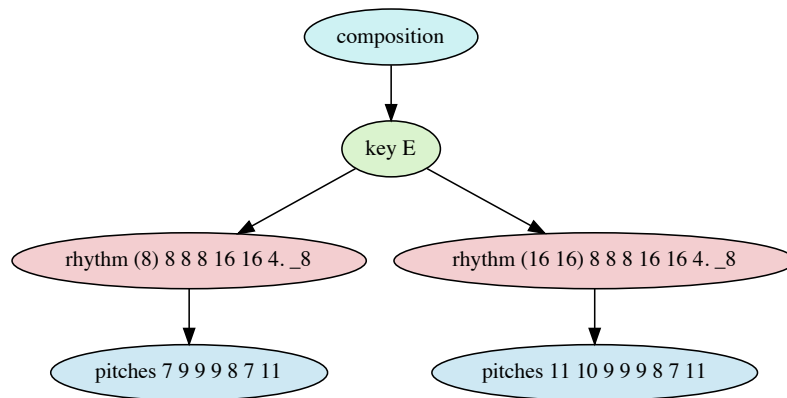
Anacruses

Some musical phrases do not start directly on a metrically strong beat, but are preceded by one or more notes, referred to as *anacrusis*, also known as *pickup* or *upbeat* (Randel 2003, p. 42). Often this happens at the very beginning of a piece, yet also phrases in the middle of compositions can be initiated using pickup beats. To indicate anacruses in **MPS**, the pickup beats are simply enclosed in parentheses. For example, Listing 4.3 specifies a rhythm with an eighth note pickup beat.

```
1 rhythm (8) 8 8 8 16 16 4. _8
```

Listing 4.3: Rhythm starting with an anacrusis

Consider the context model of the first measures of Vivaldi's *Concerto No. 1 in E major, Op. 8, RV 269*, which is shown in Figure 4.9a along with the resulting score in Figure 4.9b. Also compare with the syntactical representation in Listing 4.4. The example demonstrates the use of anacruses both at the beginning and in the course of a piece. Note that existing rhythmic information is overwritten when using pickup beats. Specifically, the model in Figure 4.9a defines an eighth rest at the end



(a) Context tree model representation



(b) Score representation with highlighted anacruses at the very beginning and before the second full measure

Figure 4.9: Score and context tree model of the the first measures of Vivaldi’s *Concerto No. 1 in E major, Op. 8, RV 269*. Corresponding files are available on the accompanying CD under `Examples/Compositions/Vivaldi/RV269_Concerto_No1_in_E_Major` (see Appendix [A](#)).

of the first rhythm, which is overwritten by the two sixteenth pickup notes of the second rhythm.

```

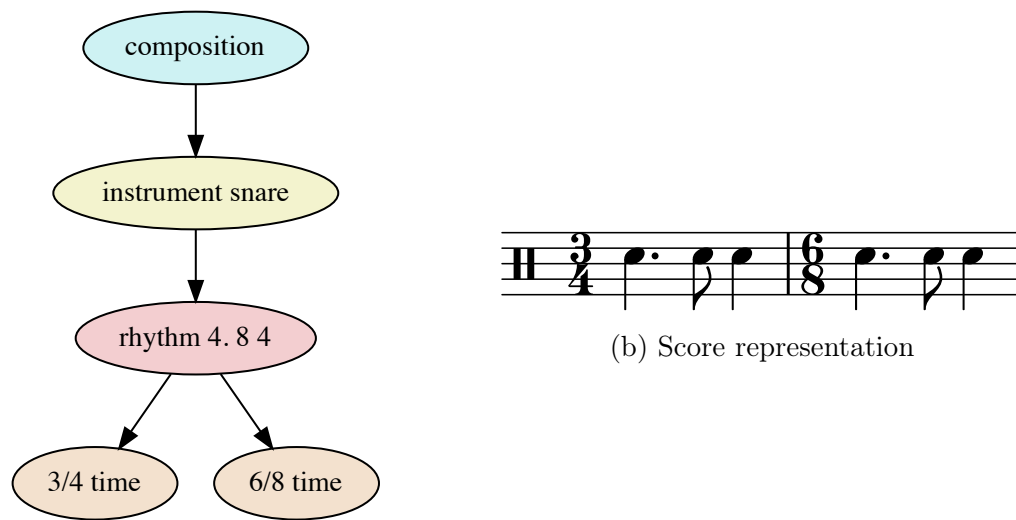
1 composition
2 {
3     key E
4     {
5         rhythm (8) 8 8 8 16 16 4. _8
6         {
7             pitches 7 9 9 9 8 7 11
8         }
9     }
10    rhythm (16 16) 8 8 8 16 16 4. _8
11    {
12        pitches 11 10 9 9 9 8 7 11
13    }
14 }
15
    
```

Listing 4.4: Syntactical representation of the context model shown in Figure [4.9a](#)

4.5.2 Meter

Rhythms are usually embedded in a metric context defining the importance of individual beats in a measure. Its time signature indicates how measures are divided.

For example, a $\frac{2}{4}$ time signature indicates the duration of a measure is divided into two quarter note pulses. Notes occurring on these two points of time in the measure are generally considered accented, whereas notes occurring between these two beats are considered weak. More specifically, each pulse in the measure has an individual importance. Clarence Barlow developed formulas for computing the importance of individual beats depending on the metric context (Barlow 2012, pp. 44ff.).



(a) Context tree model representation

Figure 4.10: Score and context tree model of a rhythm in two different metric contexts

```

1 composition
2 {
3   instrument snare
4   {
5     rhythm 4. 8 4
6     {
7       time 3/4
8       time 6/8
9     }
10  }
11 }
```

Listing 4.5: Syntactical representation of the model shown in Figure 4.10a

Depending on the metric context, the very same rhythm can have different musical meanings. This is illustrated in the model shown in Figure 4.10. An equivalent syntactical representation is given in Listing 4.5. The model puts the rhythm $\text{♩.} \text{♪♪}$ into two different metric contexts. In the first measure with a $\frac{3}{4}$ time signature, the dotted quarter note and the final quarter note are both accented, whereas in the

second measure with a $\frac{6}{8}$ time signature the dotted quarter note and the eighth note are accented. This example demonstrates that the same rhythm can have different musical meanings depending on the metric context.

4.5.3 Tempo

Tempo is an individual context dimension which can be changed independently from time signatures. The tempo is specified in **Beats per Minute (BPM)**. For example:

```
1 tempo 100
```

By default, the **BPM** specification defines the temporal distance of quarter notes. It is also possible to define other note durations to which the **BPM** specification relates. To specify the tempo for eighth notes, for example, the following syntax is used:

```
1 tempo 80 noteDuration 8
```

It is also possible to define gradual tempo changes:

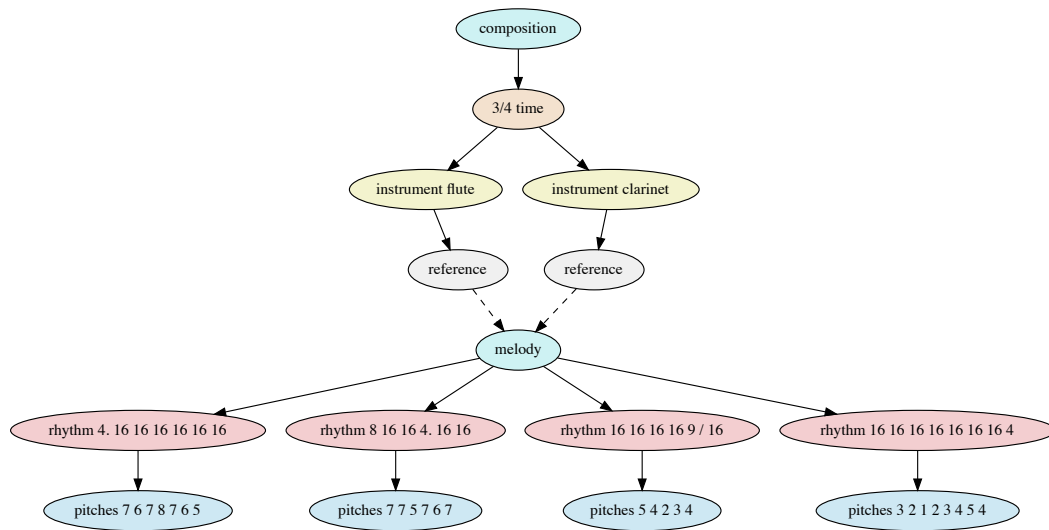
```
1 tempo 80 -> 110 noteDuration 8
```

4.5.4 Instruments

A common task for composers and arrangers is the instrumentation or orchestration of pieces, which involves the assignment of instruments or instrument combinations for individual parts of the piece.

The model shown in Figure 4.11 represents an excerpt from *Boléro* by Maurice Ravel, in which the melody is sequentially played by the flute and the clarinet. The equivalent syntactical representation is shown in Listing 4.6. The code in Listing 4.6 contains a fragment named *melody*, in which an abbreviatory syntax is used to express child node relations. If a context node contains only one child node, the child node can be specified after a preceding comma. In this case, no curly braces are required, which contributes to improved readability of the code.

For a version of this model in which the melody is played simultaneously, refer to Figure 4.30 in Chapter 4.8.1.



(a) Context tree model representation



(b) Score representation. Note that the clarinet is notated in Bb.

Figure 4.11: Score and context tree model of an excerpt from *Boléro* by Maurice Ravel, in which the melody is played sequentially. Corresponding audio and MIDI files are available on the accompanying CD under Examples/Compositions/Ravel/Bolero (see Appendix [A](#)).

```

1 composition
2 {
3   time 3/4
4   {
5     instrument flute
6     {
7       fragmentRef melody
8     }
9     instrument clarinet
10    {
11      fragmentRef melody
12    }
13  }
14 }
15 fragment melody
16 {
17   rhythm 4. 16 16 16 16 16 16, pitches 7 6 7 8 7 6 5
18   rhythm 8 16 16 4. 16 16, pitches 7 7 5 7 6 7
19   rhythm 16 16 16 16 9/16, pitches 5 4 2 3 4
20   rhythm 16 16 16 16 16 16 16 4, pitches 3 2 1 2 3 4 5 4
21 }

```

Listing 4.6: Syntactical representation of the *Boléro* excerpt shown in Figure 4.11a

Instrument Definitions

MPS provides a number of built-in instrument definitions. Refer to the documentation¹ for a complete list of available instruments. If additional instruments are required, users are able to define custom instruments by providing instrument definitions. Consider the code in Listing 4.7 for an instrument definition of an acoustic bass guitar.

```

1 instrumentDef acousticBass
2 {
3   pitchRange [23..67]
4   maxSimultaneousNotes 4
5   scoreLabel "Bass"
6   lilyPondInstrumentName "acoustic bass"
7   defaultClef bass
8   defaultOctave 2
9 }

```

Listing 4.7: Syntax for an instrument definition for an acoustic bass guitar

The `instrumentDef` keyword is followed by an instrument identifier, which is used to reference the instrument definition in instrument contexts. Enclosed in curly braces, optional instrument parameters follow. Refer to Table 4.3 for a list of available parameters.

¹The **MPS** documentation is available on the accompanying CD (see Appendix A) and online at <http://www.musicprocessing.net/doc/>.

Table 4.3: Instrument definition parameters

Parameter	Description
type	Either percussion for percussion instruments or synth for synthesizers used for electronic / electroacoustic music. Omit this parameter to create an instrument of default type which is playable in different pitches.
pitchRange	Specifies the compass of the instrument in terms of MIDI notes in the syntax <code>[lowest note..highest note]</code> .
maxSimultaneousNotes	Specifies the maximum number of notes which can be played simultaneously.
scoreLabel	Name of the instrument which is displayed at the beginning of staves in scores.
lilyPondInstrumentName	Instrument name used for assigning a MIDI instrument when exporting LilyPond scores ² .
defaultClef	Default clef to use in scores. Currently supported clef names are: treble , alto , tenor and bass .
defaultOctave	Default MIDI octave to use if none is specified in composition models.

4.5.5 Pitches

Pitch is an elementary dimension of music. Pitches are perceived by humans in terms of periodic oscillations of air pressure. The unit for the pitch of a single oscillating sound wave is measured in *Hertz* (abbreviated Hz), which is defined as cycles per second, or s^{-1} (H. E. White and D. H. White [2014], p. 11). Physically, musical instruments produce individual timbres which are characterized by certain ratios of partial frequencies, which are expressible as multiples of a so called *fundamental* frequency. Musicians refer only to the fundamental frequency when specifying the pitch of a note. Moreover, musicians have established a domain-specific terminology for frequencies. While a physicist might refer to the fundamental frequency of the concert pitch A as *440 Hz*, musicians refer to this frequency using a pitch name and an octave specification, such as *A above middle C*. Middle C is located approximately in the middle of a standard piano keyboard.

Another system for referring to pitches is specified by the **MIDI** protocol (see Chapter 2.1.1). Furthermore, **SPN** defines numbers for each octave, 4 being defined as the middle octave (Müller [2015], p. 4). Refer to Table 4.4 for an overview of **MIDI** note numbers and corresponding octaves.

MPS supports multiple types of pitch specifications. One possibility is to specify absolute pitches and octave numbers such as *Ab5*. **MIDI** note numbers were specif-

Table 4.4: MIDI note numbers and octave numbers according to scientific pitch notation

MIDI Note Numbers	Octave Number	Octave Name
0-11	-1	Double Contra
12-23	0	Sub Contra
24-35	1	Contra
36-47	2	Great
48-59	3	Small
60-71	4	One-line
72-83	5	Two-line
84-95	6	Three-line
96-107	7	Four-line
108-119	8	Five-line
120-127	9	Six-line

ically not chosen as pitch unit, since enharmonic differentiations are not possible. For instance, the pitch names $G\sharp$ and $A\flat$ correspond to the same key on a piano (assuming the same octave number is specified), but have different musical meanings relating to the harmonic context (see Chapter 4.5.8 for more details). For this reason, MPS uses harmonically significant pitch names. Alternatively, pitches may be given in terms of degrees on a scale, which is elaborated in section 4.5.6.

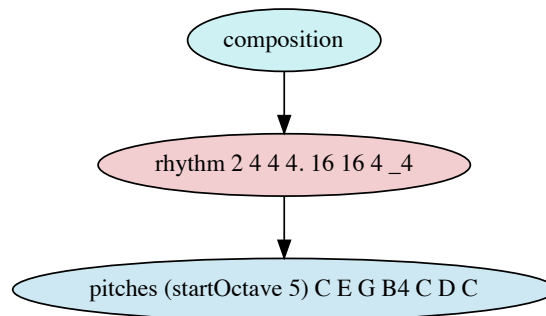
The first two measures of W. A. Mozart’s *Piano Sonata No. 16 in C major, K. 545*, also known as *Sonata Facile* are used as an example for pitch specifications using pitch names and octave numbers. Consider the representations shown in Figure 4.12 along with the correspondent syntactical representation in Listing 4.8. Various syntax alternatives for pitch specifications are listed in Table 4.5.

```

1 composition
2 {
3   rhythm 2 4 4 4. 16 16 4 _4
4   {
5     pitches (startOctave 5) C E G B_4 C D C
6   }
7 }
```

Listing 4.8: Syntactical representation of the model shown in Figure 4.12a.

Additional parameters may be used when specifying pitch sequences, which are explained in Table 4.6. If these parameters are used, they have to be syntactically enclosed in parentheses before pitches or scale degrees are specified, as demonstrated in Listing 4.8 with the `startOctave` parameter.



(a) Context tree model representation. Pitches are specified using absolute note names and the octave number.



(b) Score representation

Figure 4.12: Score and context tree model of W. A. Mozart's *Piano Sonata No. 16 in C major, K. 545*, mm. 1–2, right hand part. Corresponding files are available on the accompanying CD under `Examples/Compositions/Mozart/KV545_SonataFacile` (see Appendix [A](#)).

Table 4.5: Pitch syntax

Syntax	Description
<note name>	Used for specifying pitches explicitly, e.g. D, C# or Eb.
<integer number>	Used for pitch specifications based on scale degrees. Refer to section 4.5.6 for more details.
# (suffix)	Raises the previously specified pitch or scale degree by one semitone.
b (suffix)	Lowers the previously specified pitch or scale degree by one semitone.
[<pitches>]	Square brackets are used to specify chords. For example, a D major chord can be written as [D F# A].
@ (prefix)	Indicates the usage of an expression to dynamically compute a pitch or scale degree. For example, the expression @getRootNote() evaluates to the root note of the current context harmony. Refer to section 4.9 for more details.

Table 4.6: Pitch sequence parameters

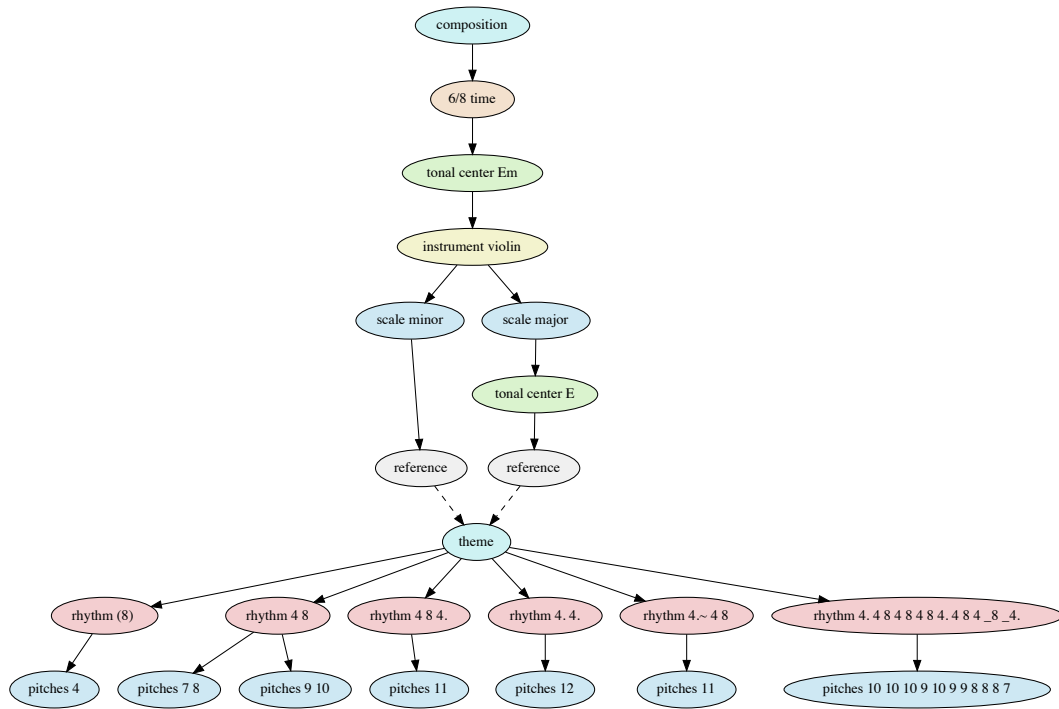
Parameter	Description
<code>startOctave</code>	Specifies the octave to use if no octaves are defined explicitly.
<code>findNearestOctave</code>	If set to <code>true</code> , the system will change the octave automatically if it implies a smaller semitone distance to the previous note. Example: in the pitch sequence <code>A C</code> the system would start in the default octave yielding <code>A4</code> . With <code>findNearestOctave</code> enabled, the next pitch would be <code>C5</code> because it has a smaller distance to <code>A4</code> than <code>C4</code> .
<code>relative to</code>	Specifies which harmonic context is to be used to determine the context scale and its tonic. Possible values are <code>key</code> and <code>harmony</code> . Refer to sections 4.5.6 and 4.5.8 for more details.

4.5.6 Scales

Frequently pitches in compositions are not randomly chosen, but selected from a specific set of pitches forming a scale. By referring to individual scale degrees, corresponding pitches are derived. Consider the model in Figure [4.13](#); the syntactical representation is shown in Listing [4.9](#).

Using scale degrees instead of absolute note names has several advantages: first, scale degrees are syntactically easier and shorter to write. Second, thinking in terms of scale degrees is often more adequate regarding music theory and reflects the way most composers and musicians think about pitches. Third, scale degrees can easily be projected onto another scales. In other words, the same degrees can be used in another scale context, which allows interesting musical variations. This is also the case for the *Vltava* model presented in Figure [4.13a](#), in which the theme is presented in two scale contexts, namely a minor and a major version.

Note that the scale contexts used in Figure [4.13a](#) are optional, because a default scale context is derived from the current harmonic context automatically. In the left branch, the current key context is E minor which results in a matching minor scale context by default. In the right branch, the harmonic context is E major and therefore the default scale is *major* accordingly. Refer to section [4.5.8](#) for more details on harmonic contexts.



(a) Context tree model representation. The theme, in which the pitches are expressed in terms of scale degrees, is referenced twice in a major and a minor scale and key context.



(b) Score representation

Figure 4.13: Score and context tree model of Bedřich Smetana's *Moldau* theme from *Vltava*, *JB 1:112/2*. The theme is referenced in a minor and a major scale context. Corresponding files are available on the accompanying CD under **Examples/Compositions/Smetana/Moldau** (see Appendix [A](#)).

```

1 composition
2 {
3     time 6/8, key Em, instrument violin
4     {
5         scale minor
6         {
7             fragmentRef theme
8         }
9         scale major, key E
10        {
11            fragmentRef theme
12        }
13    }
14 }
15
16 fragment theme
17 {
18     rhythm (8)
19     {
20         pitches 4
21     }
22     rhythm 4 8
23     {
24         pitches 7 8
25         pitches 9 10
26     }
27     rhythm 4 8 4.
28     {
29         pitches 11
30     }
31     rhythm 4. 4.
32     {
33         pitches 12
34     }
35     rhythm 4. ~ 4 8
36     {
37         pitches 11
38     }
39     rhythm 4. 4 8 4 8 4 8 4. 4 8 4 _8 _4.
40     {
41         pitches 10 10 10 9 10 9 9 8 8 8 7
42     }
43 }

```

Listing 4.9: Syntactical representation of the model shown in Figure 4.13a

Scale Definitions

MPS provides a number of built-in scales, which are listed in Table 4.7. If additional scales are required, users are able to define custom scales using scale definitions in the header section of composition files. An example definition for the dorian scale is shown in Listing 4.10.

Table 4.7: List of scales provided by the Music Processing Suite library

Name	Identifier	Degrees in Semitones
Major	major	0 2 4 5 7 9 11
Ionian	ionian	0 2 4 5 7 9 11

Continued on next page

Table 4.7 – *Continued from previous page*

Name	Identifier	Degrees in Semitones
Minor	minor	0 2 3 5 7 8 10
Aeolian	aeolian	0 2 3 5 7 8 10
Blues	blues	0 3 5 6 7 10
Chromatic	chromatic	0 1 2 3 4 5 6 7 8 9 10 11
Diminished	diminished	0 1 3 4 6 7 9 10
Dorian	dorian	0 2 3 5 7 9 10
Harmonic Major	harmonicMajor	0 2 4 5 7 8 11
Harmonic Minor	harmonicMinor	0 2 3 5 7 8 11
Locrian	locrian	0 1 3 5 6 8 10
Lydian	lydian	0 2 4 6 7 9 11
Major Pentatonic	majorPentatonic	0 2 4 7 9
Minor Pentatonic	minorPentatonic	0 3 5 7 10
Melodic Major	melodicMajor	0 2 4 5 7 8 10
Melodic Minor	melodicMinor	0 2 3 5 7 9 11
Mixolydian	mixolydian	0 2 4 5 7 9 10
Phrygian	phrygian	0 1 3 5 7 8 10
Whole-tone	whole	0 2 4 6 8 10

```

1 scaleDef dorian
2 {
3     degrees 0 2 3 5 7 9 10
4 }

```

Listing 4.10: Syntax of a scale definition for the dorian scale

4.5.7 Loudness

To account for the loudness dimension of music, **MPS** supports both static loudness contexts and gradual loudness contexts. The latter are used to model *crescendo* and *decrescendo*. Static loudness specifications are syntactically described with the **loudness** keyword followed by a single loudness instruction as shown in Listing 4.11. Refer to Table 4.8 for a enumeration of possible loudness specifications.

```

1 loudness ff

```

Listing 4.11: Syntactical representation of a static loudness instruction

Table 4.8: Loudness specification syntax

Syntax	Description
pppp	pianissississimo
ppp	pianississimo
pp	pianissimo
p	piano
mp	mezzo-piano
mf	mezzo-forte
f	forte
ff	fortissimo
fff	fortississimo
ffff	fortissississimo
current	Refers to the last loudness level specified. This can be used for specifying gradual dynamics (<i>crescendo</i> and <i>decrescendo</i>).

Gradual loudness specifications contain two loudness instructions delimited by the token `->` as illustrated in Listing 4.12.

```
1 loudness p -> f
```

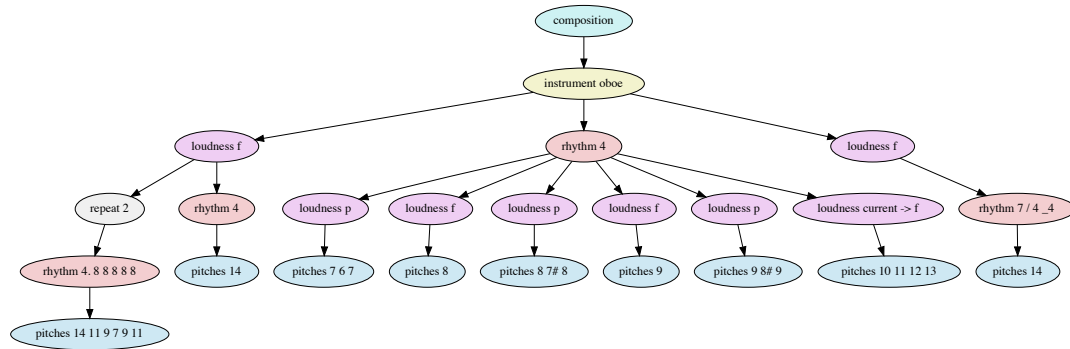
Listing 4.12: Syntactical representation of a gradual loudness instruction

For gradual loudness instructions, the special loudness instruction **current** may be used, which refers to the last loudness level specified in the composition. This is demonstrated in the following example in Listing 4.13, which contains the loudness context `loudness current -> f`. It results in a crescendo visible in the sixth measure of the score in Figure 4.14b.

In Figure 4.14, a model of W. A. Mozart’s *Flute and Harp Concerto in C major, K. 299/297c* is presented containing both static and gradual dynamic instructions along with the resulting score. The equivalent language representation is given in Listing 4.13.

4.5.8 Harmonic Contexts

Harmonic contexts are especially important in western tonal music, in which pitches in compositions are usually organized in reference to specific keys (Randel 2003, p. 898). Matching scales and functions of specific chords can be derived depending on the key context. MPS supports explicit specifications of harmonic contexts including hierarchically arranged keys and contextual harmonies.



(a) Context tree model representation



(b) Score representation

Figure 4.14: Score and context tree model of the opening oboe theme from W. A. Mozart's *Flute and Harp Concerto in C major, K. 299/297c*, containing static and gradual loudness contexts. Corresponding files are available on the accompanying CD under **Examples/Compositions/Mozart/KV299_297c_ConcertoInCMajor** (see Appendix **A**).

```

1 composition
2 {
3     instrument oboe
4     {
5         loudness f
6         {
7             repeat 2
8             {
9                 rhythm 4. 8 8 8 8 8, pitches 14 11 9 7 9 11
10            }
11            rhythm 4, pitches 14
12        }
13        rhythm 4
14        {
15            loudness p, pitches 7 6 7
16            loudness f, pitches 8
17            loudness p, pitches 8 7# 8
18            loudness f, pitches 9
19            loudness p, pitches 9 8# 9
20            loudness current -> f, pitches 10 11 12 13
21        }
22        loudness f, rhythm 7/4 _4, pitches 14
23    }
24 }
    
```

 Listing 4.13: Syntactical representation of the model shown in Figure **4.14a**

Keys

Keys serve as musical “landmarks” in tonal compositions. While simple pieces might only define one key, more complex compositions might incorporate temporary key changes (modulations) or even key changes for whole sections or parts of the piece, for instance compositions geared to the sonata form (Randel [2003](#), pp. 800, 898). Modulations and key changes can be modeled elegantly in [MPS](#) using hierarchical arrangements (as discussed in section [4.4.1](#)) and polymorphism (see section [4.4.3](#)). In this way, the scope of the specified keys can be controlled using an arbitrary number of logical levels.

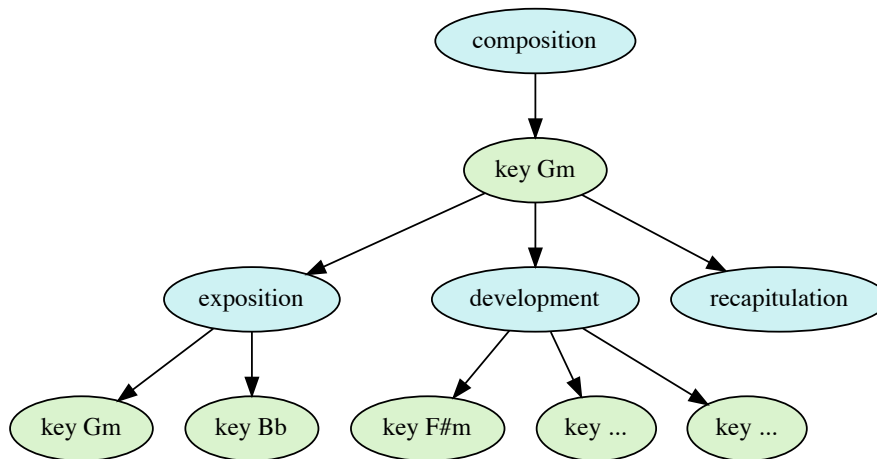


Figure 4.15: Schematic context tree model of W. A. Mozart’s *Symphony No. 40 in G minor*, K. 550, Mv. I demonstrating hierarchically arranged global and local keys

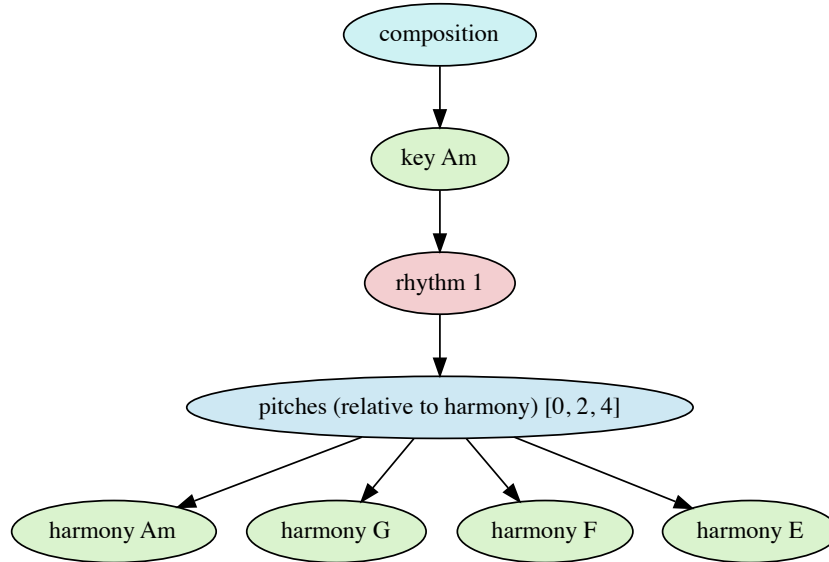
An example is provided in Figure [4.15](#), which contains a schematic hierarchical arrangement of keys used in the first movement of Mozart’s *Symphony No. 40 in G minor*, K. 550. The global key of this movement is *G minor*. Themes are presented in the exposition in *G minor* and its relative major key *Bb major*. In the development, Mozart modulates through a number of keys starting with *F# minor*. The recapitulation concludes in the global key *G minor*.

Syntactically, keys are defined by referring to the root note name (for instance *G* or *D#*) and the optional suffix *m* indicating a minor harmony (e.g. *Am* or *Bbm*).

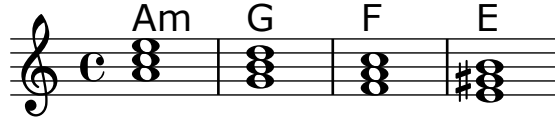
Harmonic Progressions

While keys provide a global harmonic context in tonal compositions, harmonic progressions provide local harmonic transitions. These can be expressed implicitly by

specifying simultaneously sounding notes or in an explicit way, for example in the style of lead sheets (as shown in Figure 3.3a, for instance).



(a) Context tree model representation



(b) Score representation

Figure 4.16: Score and context tree model demonstrating a harmonic progression in the context of a global key. Corresponding files are available on the accompanying CD under `Examples/Model/HarmonicProgression` (see Appendix A).

The context tree model in Figure 4.16 defines a harmonic progression consisting of four local harmonies. These are hierarchically embedded in the global key context *A minor*. The corresponding language representation is shown in Listing 4.14. The complexity of harmonies is not limited to major and minor chords. MPS supports additional notes and harmony specifications as specified in Table 4.9. Note that these additions can be combined, for instance `A7sus4` defines a harmony with the notes A, D, E and G. Refer to Chapter 4.6.3 for more examples demonstrating harmony additions.

Another feature of MPS is the support for Roman numeral harmony notation. The numerals represent harmonies based on specific scale degrees on the underlying scale, which is derived from the current key context. For example, in the key of C major, the C major chord can be represented by *I*, F major by *IV* and G major by *V*.

```

1 composition
2 {
3     key Am
4     {
5         rhythm 1
6         {
7             pitches (relative to harmony) [0 2 4]
8             {
9                 harmony Am
10                harmony G
11                harmony F
12                harmony E
13            }
14        }
15    }
16 }

```

Listing 4.14: Syntactical representation of the context model in Figure 4.16a demonstrating a harmonic progression in the context of a global key.

Table 4.9: Harmony additions

Syntax	Description
<integer number>	Additional harmony note relative to the root note, expressed in terms of scale degrees. For example, F7 translates to a F major chord with added minor seventh.
# or b prefix	Optional prefix for additional harmony notes to indicate a semitone correction upwards or downwards, respectively.
maj7	Adds a major seventh relative to the root note.
m7	Indicates a minor chord with a minor seventh.
sus2	Suspended second chord in which a perfect second is added and the third is omitted.
sus4	Suspended second chord containing a perfect fourth but no third.
◦	Diminished chord
+	Augmented chord
power	Power chord containing only the root and the fifth. Frequently used in rock and metal genres (McDonald 2000).

Generally, upper case Roman numbers are used to represent major chords and lower case letters are used for minor chords. For example, D minor in the key of C major can be expressed with *ii*. The harmonic additions from Table 4.9 can be used as well. For instance, a dominant seventh chord can be written as V7.

In certain cases it is convenient to specify a harmonic sequence as a whole. In MPS, this is possible using `harmonicProgression` contexts in combination with a *harmonicRhythm* instruction defining the duration of each harmony in the progression. This is demonstrated in the context tree model depicted in Figure 4.17. It results in an equivalent score as the previous example (see Figure 4.16b). Listing 4.15 shows the corresponding language representation.

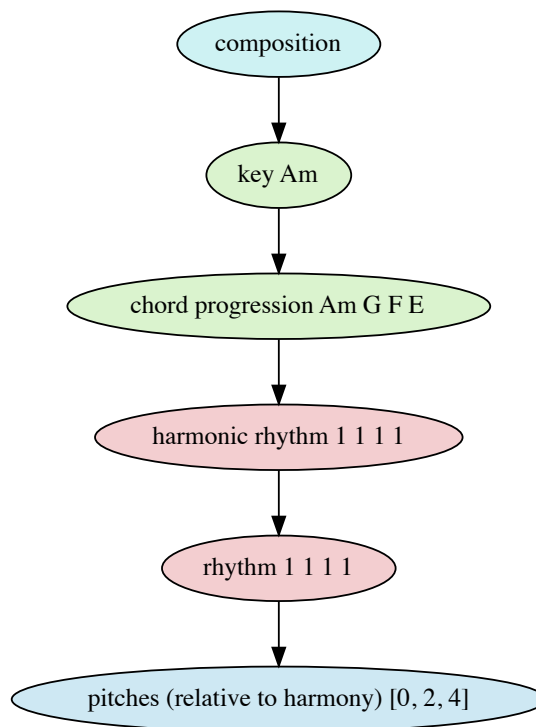


Figure 4.17: Context tree model demonstrating the definition of a harmonic progression and a corresponding harmonic rhythm

```

1 composition
2 {
3   key Am
4   {
5     harmonicProgression Am G F E, harmonicRhythm 1 1 1 1
6     {
7       rhythm 1 1 1 1
8       {
9         pitches(relative to harmony) [0 2 4]
10      }
11    }
12  }
13 }

```

Listing 4.15: Syntactical representation of the context model in Figure 4.17 demonstrating the definition of a harmonic progression and a corresponding harmonic rhythm.

4.5.9 Lyrics

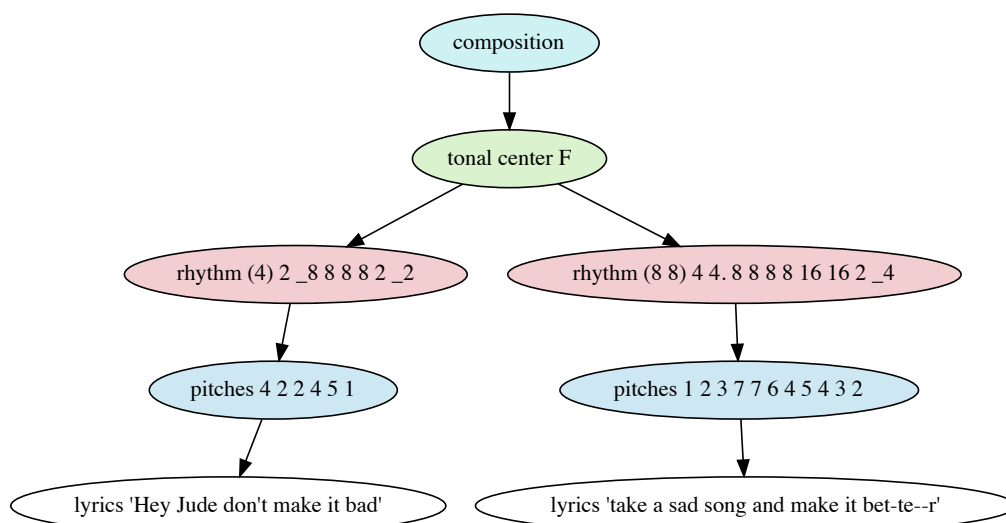
In vocal music, sung notes are usually associated with syllables, which is considered as a separate context dimension in the MPS model. Syllables are specified using a simple syntax. To distribute syllables of a word onto multiple notes, hyphens (-) may be used. Syllable assignments for specific notes can be skipped using underscores (_). As an example, the first measures of the song *Hey Jude* by the Beatles is used. Refer to Listing 4.16 and the corresponding representations in Figure 4.18.

```

1 composition
2 {
3   key F
4   {
5     rhythm (4) 2 _8 8 8 8 2 _2
6     {
7       pitches 4 2 2 4 5 1
8       {
9         lyrics "Hey Jude don't make it bad"
10      }
11    }
12
13    rhythm (8 8) 4 4. 8 8 8 8 16 16 2 _4
14    {
15      pitches 1 2 3 7 7 6 4 5 4 3 2
16      {
17        lyrics "take a sad song and make it bet-te--r"
18      }
19    }
20  }
21 }

```

Listing 4.16: Language representation of the first measures of *Hey Jude* by the Beatles demonstrating the specification of lyrics



(a) Context tree model representation



(b) Score representation

Figure 4.18: Score and context tree model of the first measures of *Hey Jude* by the Beatles demonstrating the specification of lyrics. Corresponding files are available on the accompanying CD under `Examples/Compositions/Beatles/Hey Jude/Leadsheet` (see Appendix [A](#)).

4.5.10 Custom Contexts

[MPS](#) offers a feature to create arbitrary custom contexts. An example is shown in Figure [4.19](#). The context tree model contains three sections in which individual moods are described by means of custom context nodes. Refer to Listing [4.17](#) for the corresponding language representation. Custom contexts are syntactically defined by the keyword *customContext*, followed by a context identifier (in this case *mood*) and a string literal representing an instance of the context.

Custom contexts are represented as individual layers in context layer models (see Chapter [3](#)). Scores generated from models containing custom contexts will contain textual annotations such as *Mood: vivid* at the top of the relevant staves.

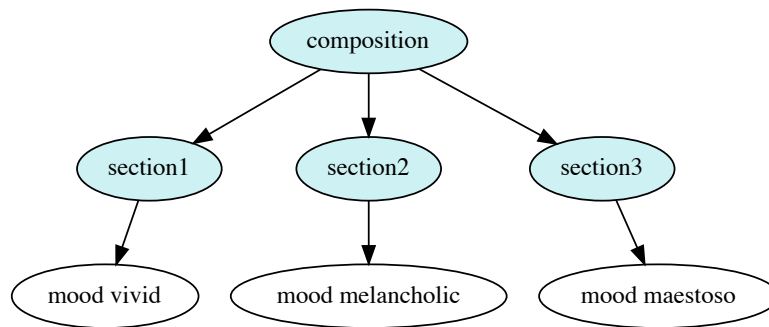


Figure 4.19: Context tree model introducing a new custom context type to describe moods of individual sections

```

1 composition
2 {
3   fragment section1
4   {
5     customContext mood "vivid"
6   }
7
8   fragment section2
9   {
10    customContext mood "melancholic"
11  }
12
13  fragment section3
14  {
15    customContext mood "maestoso"
16  }
17
18 }

```

Listing 4.17: Syntactical representation of the context tree model in Figure 4.19 which defines custom contexts to describe the mood of individual sections

4.6 Context Modifiers

Frequently, already introduced musical material is modified in the course of compositions. In these cases, no fundamentally new ideas are introduced, but existing ones are shaped. To account for this, so called *context modifiers* allow to adjust already existing musical material. Their functionality is explained in the following subsections.

4.6.1 Rhythmic Modifiers

Rhythmic context modifiers have the purpose of manipulating existing rhythmic contexts in a musical composition.

Augmentations and Diminutions

Rhythmic augmentation involves prolonging the note lengths of a given rhythm by multiplying the original lengths with a constant factor, typically 2. However, other scale factors are possible. A rhythmic diminution is considered as the opposite of a rhythmic augmentation, i.e. the note durations are not extended but shortened by a constant factor.

The following example demonstrates a model of a subject being transformed using diminution and inversion. It can be found in J. S. Bach's *The Art of Fugue, BWV 1080, Contrapunctus VII*. The language representation is shown in Listing 4.18, the corresponding model is visualized in Figure 4.20 and the resulting score is depicted in Figure 4.21.

```

1 composition
2 {
3   key Dm
4   {
5     parallel
6     {
7       fragmentRef soprano
8       fragmentRef tenor
9     }
10  }
11 }
12
13 fragment soprano
14 {
15   rhythm _1
16   inversion 11
17   {
18     fragmentRef subject
19   }
20 }
21
22 fragment tenor
23 {
24   diminution, scale melodicMinor
25   {
26     fragmentRef subject
27   }
28 }
29
30 fragment subject
31 {
32   rhythm 2 4. 8 4. 8 2 2 4. 8 5/8 8 8 8 4 _4 _2
33   {
34     pitches 0 4 3 2 1 0 -1 0 1 2 3 2 1 0
35   }
36 }

```

Listing 4.18: Language representation of an excerpt from J. S. Bach's *The Art of Fugue, BWV 1080, Contrapunctus VII*, in which diminution and inversion is applied

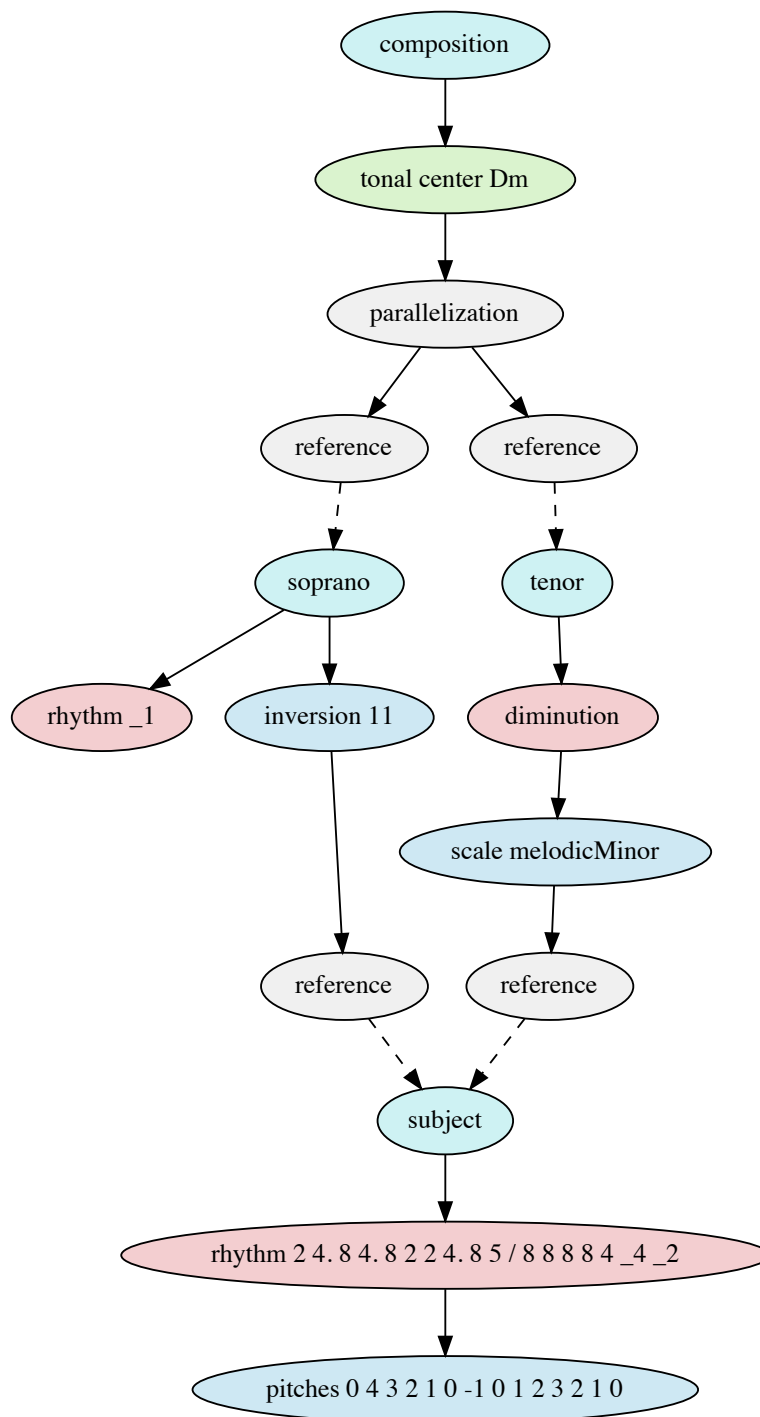


Figure 4.20: Context tree model of an excerpt from J. S. Bach's *The Art of Fugue*, BWV 1080, *Contrapunctus VII*, in which diminution and inversion is applied



Figure 4.21: Score of an excerpt from of J. S. Bach’s *The Art of Fugue*, *BWV 1080*, *Contrapunctus VII*, resulting from the context model in Figure 4.20. Corresponding files can be found on the accompanying CD under `Examples/Compositions/Bach/BWV1080_TheArtOfFugue/Contrapunctus7_Diminution_and_Inversion` (see Appendix A).

Rhythmic Extensions

Rhythmic extensions are used to extend the duration of the last note or rest in a rhythm. This modifier was already demonstrated in the context tree model for Beethoven’s *Symphony No. 5 in C Minor*, *Op. 67* in section 4.3. Syntactically, rhythmic extensions are specified using the keyword `rhythmicExtension`, followed by a note duration as explained in section 4.5.1. If the note duration is positive, the rhythm is extended. If the note duration is negative, the rhythm is shortened by the absolute value of the given negative duration.

Rhythmic Adjustments

Rhythmic adjustment modifiers allow to modify rhythms at the beginning and at the end. The modifications are specified by means of two durations for the beginning and the end of the rhythm, respectively. It is possible to specify both or only one of the parameters. Refer to Table 4.10 for detailed parameter descriptions.

Table 4.10: Rhythmic adjustment modifier parameters

Parameter	Description
<code>startDelta</code>	Specifies how the rhythm is modified at the beginning. If <code>startDelta</code> is positive, the rhythm will start from the given time, effectively shortening the rhythm by <code>startDelta</code> . If <code>startDelta</code> is negative, the first note or rest of the rhythm will be extended.
<code>endDelta</code>	Specifies a duration for the adjustment of the end of the rhythm. If <code>endDelta</code> is positive, the rhythm is extended; if <code>endDelta</code> is negative, the rhythm is shortened. The behaviour is identical with the <code>rhythmicExtension</code> modifier introduced in the previous section.

Rhythmic Insertions

This modifier inserts a rhythm into the contextually present rhythm. This can either happen in an additive manner, whereupon existing notes and rests are shifted to the

Table 4.11: Rhythmic insertion modifier parameters

Parameter	Description
<code>offset</code>	Specifies after which duration the insertion should be applied to the rhythm.
<code>rhythm</code>	Defines the rhythm to be inserted in the syntax introduced in section 4.5.1.
<code>mode</code>	Either <code>insert</code> to shift existing notes and rests after the insertion to the right or <code>overwrite</code> to overwrite existing elements.

Table 4.12: Rhythmic displacement modifier parameters

Parameter	Description
<code>offset</code>	Defines the rhythm translation offset. For positive durations, the rhythm is shifted to the right, for negative durations to the left.
<code>mode</code>	In <code>discard</code> mode, notes moved over the rhythm’s boundary are removed. In <code>wrap</code> mode, the notes are appended to the other end of the rhythm.

right, or in a destructive manner, whereupon existing elements are overwritten. A rhythmic insertion was already demonstrated in Queen’s *Bohemian Rhapsody*. Refer to the score in Figure 4.3 and compare the rhythms in the first and the second measure, which both start off with three eighth notes, but continue differently. In the model shown in Figure 4.4 this is expressed using a rhythmic insertion. It is used in the right subtree, which represents the specifics of the second measure. The rhythm $8\ 16\ 5/16$ is inserted into the basic rhythm $_8\ 8\ 8\ 8\ 4\ 4$ at offset 2, i.e. after the duration of a half note, effectively replacing the two quarter notes with the specified rhythm. Table 4.11 contains explanations for all parameters of this modifier.

Rhythmic Displacements

Rhythmic displacement modifiers are used to translate existing rhythms by moving them to the right or to the left in itself. The modifier takes a note duration offset and a mode specification as parameters, which is explained in detail in Table 4.12. As an example, consider Steve Reich’s composition *Clapping Music*, in which a rhythmic motif is repeatedly performed by two players. For the second player, the rhythm is iteratively shifted and wrapped, resulting in twelve rhythmic variations. See Figure 4.22 showing a context tree model containing a repeatedly applied rhythmic displacement modifier and the Figure 4.23 for the corresponding score. The syntactic representation of the model can be found in Listing 4.19.

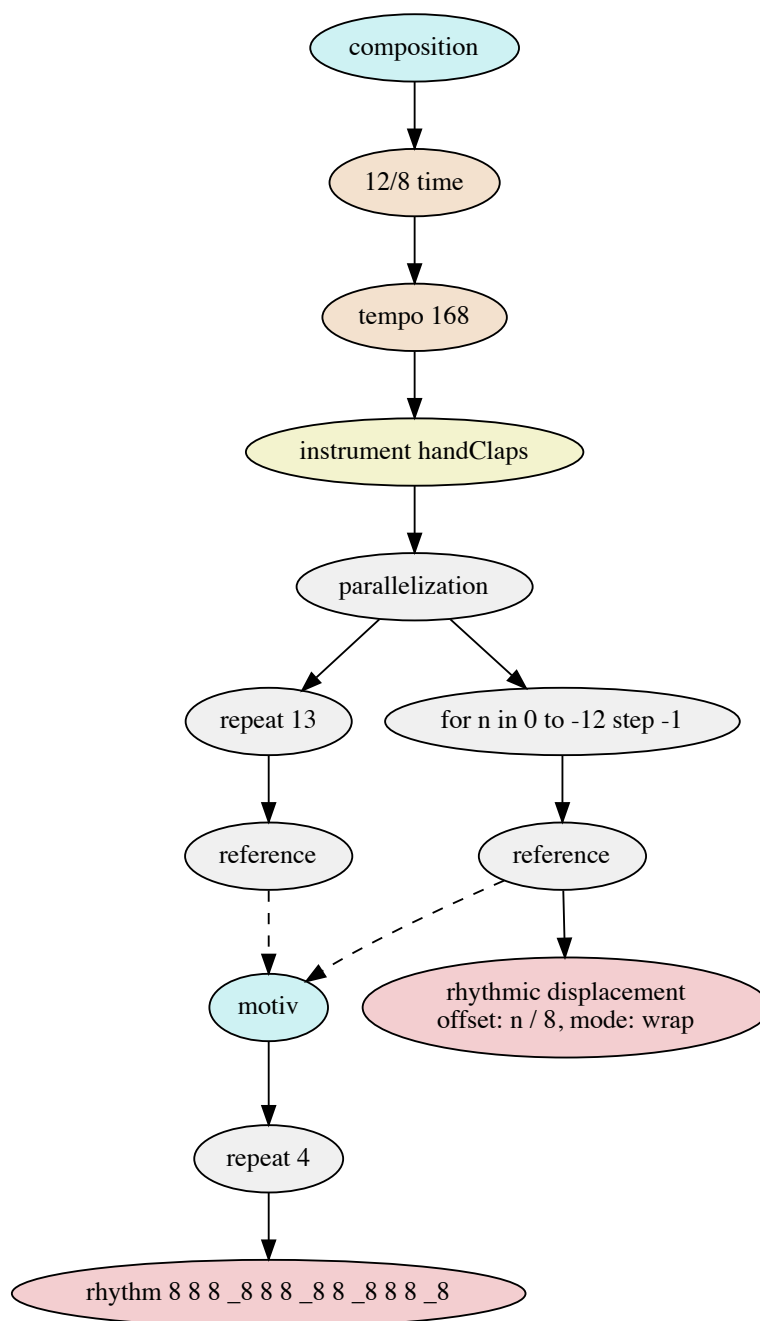


Figure 4.22: Context tree model of Steve Reich's *Clapping Music*, in which iterative rhythmic displacements are utilized

```

1 composition
2 {
3   time 12/8, tempo 168
4   {
5     instrument handClaps
6     {
7       parallel
8       {
9         repeat 13
10        {
11          fragmentRef motiv
12        }
13        for n in 0 to -12 step -1
14        {
15          fragmentRef motiv
16          {
17            rhythmicDisplacement mode wrap offset n/8
18          }
19        }
20      }
21    }
22  }
23 }
24
25 fragment motiv
26 {
27   repeat 4
28   {
29     rhythm 8 8 8 _8 8 8 _8 8 _8 8 8 _8
30   }
31 }

```

Listing 4.19: Language representation of Steve Reich’s *Clapping Music* model in Figure 4.22, in which iterative rhythmic displacements are utilized

4.6.2 Pitch Modifiers

Pitch modifiers are used for manipulating contexts in the musical pitch dimension.

Transpositions

Transpositions have the effect of modifying contextually available pitches. The modifier can be applied in three modes in order to support semitone-based transpositions, scale-based transpositions and octave translations. All parameters are explained in Table 4.13. Refer to Chapter 4.8.5 for an example demonstrating various transposition techniques.

Inversions

Inversions were already demonstrated in section 4.6.1 in conjunction with a diminution using J. S. Bach’s *The Art of Fugue, BWV 1080, Contrapunctus VII* as an example.



Figure 4.23: Score of Steve Reich’s *Clapping Music*. The score results from the context tree model in Figure 4.22, in which iterative rhythmic displacements are utilized. Corresponding files are available on the accompanying CD under Examples/Compositions/Reich/Clapping Music (see Appendix A).

Table 4.13: Transposition modifier parameters

Parameter	Description
mode	Defines the unit of the interval expression. Three modes are available: absolute for semitone-based transpositions, inScale to perform transpositions of scale degrees and octaves for octave translations. If the parameter is not specified, the default absolute will be used.
interval	Expression which must be interpretable as an integer number. The unit of this number is defined by the mode parameter.

Parallel Intervals

Parallel interval modifiers add simultaneously audible pitches in a specific interval to existing pitches. The intervals can be specified in terms of semitones, scale degrees or octaves. As an example, a model of the guitar introduction of Deep Purple’s *Smoke on the Water* is shown in Figure 4.24, the syntactical representation of which can be seen in Listing 4.20.

```

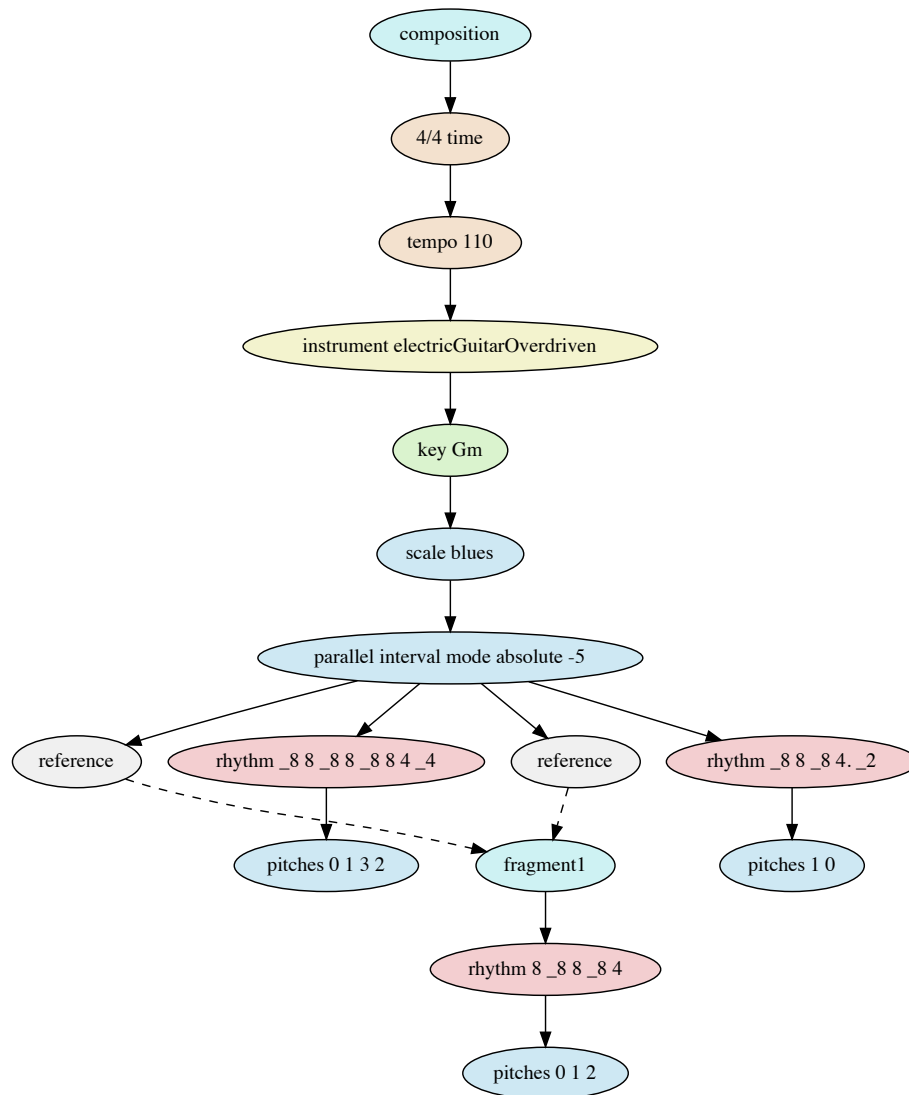
1 composition
2 {
3   time 4/4, tempo 110
4   {
5     instrument electricGuitarOverdriven
6     {
7       key Gm
8       {
9         scale blues
10        {
11          parallelInterval mode absolute -5
12          {
13            fragmentRef fragment1
14
15            rhythm _8 8 _8 8 _8 8 4 _4
16            {
17              pitches 0 1 3 2
18            }
19
20            fragmentRef fragment1
21
22            rhythm _8 8 _8 4. _2
23            {
24              pitches 1 0
25            }
26          }
27        }
28      }
29    }
30  }
31 }
32
33 fragment fragment1
34 {
35   rhythm 8 _8 8 _8 4
36   {
37     pitches 0 1 2
38   }
39 }

```

Listing 4.20: Language representation of the the guitar introduction of Deep Purple’s *Smoke on the Water*, in which a parallel interval modifier is used.

The main melodic motif is notated in terms of degrees on the G minor blues scale, which consists of the minor pentatonic scale with an added “blue note” between the third and fourth scale degree, as shown in Figure 4.25.

The upper notes of the famous *Smoke on the Water* riff can be specified in terms of scale degrees on the G minor blues scale. When analyzing the distance between the notes, it becomes apparent that the lower notes have a constant distance to the upper notes, namely five semitones or a perfect fourth. It is therefore convenient to



(a) Context tree model representation



(b) Score representation. The black notes were specified using the G minor blues scale (see Figure 4.25). The red notes were automatically generated using a parallel interval modifier which was configured with a constant interval of a descending perfect fourth.

Figure 4.24: Score and context tree model of the guitar introduction of Deep Purple's *Smoke on the Water*, in which a parallel interval modifier is used. Corresponding files are available on the accompanying CD under **Examples/Compositions/Deep Purple/Smoke On The Water** (see Appendix A).



Figure 4.25: G minor blues scale

Table 4.14: Parallel interval modifier parameters

Parameter	Description
mode	Specifies the interval unit. Available modes are absolute (in semi-tones), inScale (for scale-specific parallel intervals) and octaves .
interval	Expression to define the parallel interval. The expression must be interpretable as integer number. See section 4.9 for more details.

specify this circumstance rather than specifying each lower pitch manually. Refer to Table 4.14 for a detailed description of parallel interval modifier parameters. Note that the first and third measure are exactly identical, which is why the individual musical contexts of these measures were extracted to a fragment and referenced twice, as already described in section 4.4.5.

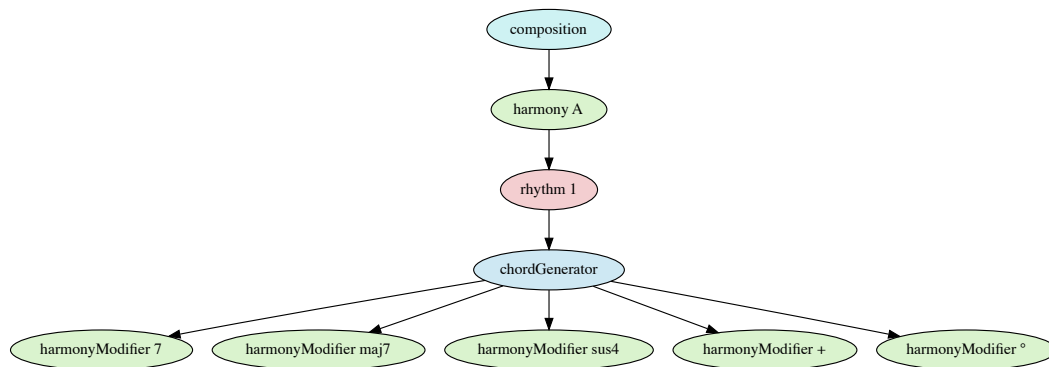
4.6.3 Harmonic Modifiers

Harmonic modifiers are used to extend or alter contextually accessible harmonies. In the model and corresponding score in Figure 4.26, various harmony modifications of the base harmony A major are demonstrated. The resulting chords of the modifications are: A major, A⁷, A^{maj7}, A augmented and A diminished. Refer to section 4.7.1 for details on chord generators.

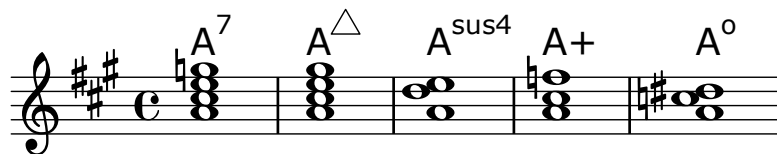
```

1 composition
2 {
3   harmony A
4   {
5     rhythm 1
6     {
7       chordGenerator
8       {
9         harmonyModifier 7
10        harmonyModifier maj7
11        harmonyModifier sus4
12        harmonyModifier +
13        harmonyModifier °
14      }
15    }
16  }
17 }
```

Listing 4.21: Language representation of the model in Figure 4.26a demonstrating various harmony modifiers.



(a) Context tree model representation



(b) Score representation

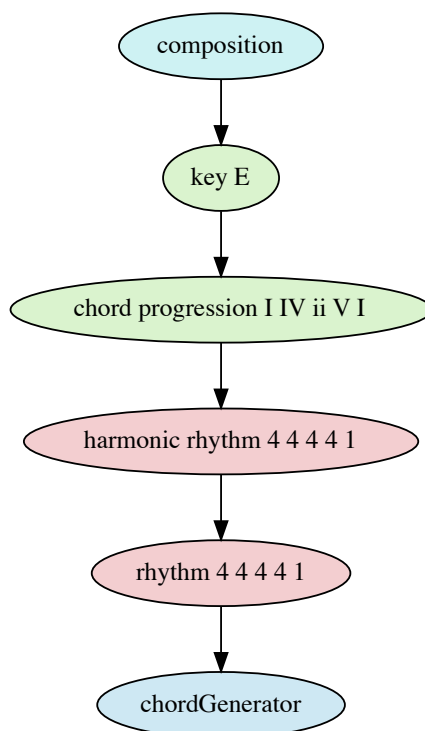
Figure 4.26: Score and context tree model demonstrating various harmony modifications. Corresponding files are available on the accompanying CD under **Examples/Model/HarmonyModifiers** (see Appendix [A](#)).

4.7 Context Generators

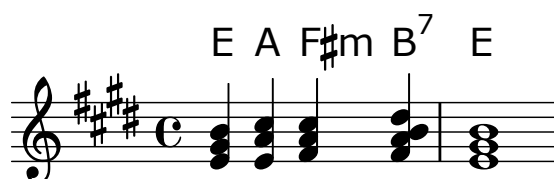
The purpose of context generators is to create new contexts based on already existing contexts. For example, pitch contexts can be built based on harmonic contexts, as explained in the following sections.

4.7.1 Chord Generators

Chord generators create pitch contexts representing specific chord inversions for contextually available harmonies. Refer to the model in Figure [4.27](#) for an example, in which an abstract chord progression is defined using Roman numerals. Concrete chord inversions are derived using a chord generator, resulting in the score shown in Figure [4.27b](#). Chord generators can be flexibly configured for various musical applications. All possible parameters are described in Table [4.15](#).



(a) Context tree model representation



(b) Score representation

Figure 4.27: Context tree model demonstrating a chord generator and the resulting score. Corresponding files are available on the accompanying CD under **Examples/Model/ChordGenerator/SimpleHarmonicProgression** (see Appendix [A](#)).

```

1 composition {
2   key E
3   {
4     harmonicProgression I IV ii V7 I
5     {
6       harmonicRhythm 4 4 4 4 1
7       {
8         rhythm 4 4 4 4 1
9         {
10          chordGenerator
11        }
12      }
13    }
14  }
15 }

```

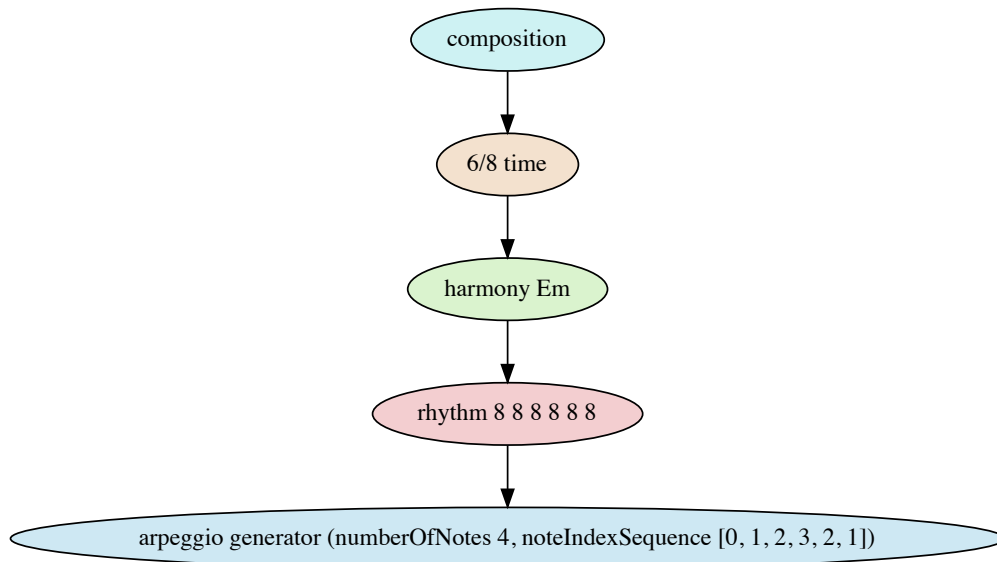
Listing 4.22: Language representation of the model in Figure 4.27a using a chord generator to create chord inversions for a given harmony progression.

Table 4.15: Chord generator parameters

Parameter	Description
<code>startOctave</code>	Defines the octave in which the lowest note of the first chord is generated.
<code>startInversion</code>	Specifies the default inversion of this chord generator. 0 corresponds to the root position, 1 to the first inversion etc.
<code>numberOfNotes</code>	Defines how many notes are generated for each chord. If this parameter is not specified, the minimum number of notes to express a harmony adequately are used. For example, three notes are used for major or minor chords but four notes for a dominant seventh chord.
<code>includeBassNote</code>	If set to <code>true</code> , the bass note (which in some cases can be different from the root note) is included in chords.
<code>findNearestInversion</code>	If set to <code>true</code> , the system will minimize the distance between successive chords. In other words, inversions with a minimum aggregated semitone distance to the previous chord will be used.

4.7.2 Arpeggio Generators

Arpeggio generators are specialized chord generators which allow to distribute individual notes of generated chords sequentially in time. An example is demonstrated in Figure 4.28 and the corresponding language representation is shown in Listing 4.23.



(a) Context tree model representation



(b) Score representation

Figure 4.28: Context tree model using an arpeggio generator and the resulting score. Corresponding files are available on the accompanying CD under `Examples/Model/ArpeggioGenerator` (see Appendix A).

Arpeggio generators determine concrete chord inversions just like chord generators. Consequently, all parameters of chord generators (see Table 4.15) can be applied to arpeggio generators. However, instead of generating simultaneously played notes, arpeggio generators produce sequentially played notes in a contextually available rhythm. The generator sequentially chooses notes from the current chord for this purpose. By default, notes are chosen in ascending order and this sequence is wrapped if more notes are required. For example, for a D major chord (D-F#-A) and a rhythm with four notes, the resulting arpeggio sequence would be D-F#-A-D. The sequence of the selected notes can be influenced with the so called *note index sequence*. Each note in the chord is assigned a zero-based index (e.g. for the above

```

1 composition
2 {
3   time 6/8, harmony Em
4   {
5     rhythm 8 8 8 8 8 8
6     {
7       arpeggioGenerator (numberOfNotes 4 noteIndexSequence 0 1 2 3 2 1)
8     }
9   }
10 }

```

Listing 4.23: Language representation of the model in Figure 4.28a using a generator to produce an E minor arpeggio.

mentioned example the indices would be: $D \Rightarrow 0$, $F\sharp \Rightarrow 1$, $A \Rightarrow 2$). To produce descending instead of ascending arpeggios, the default note index sequence 0 1 2 could be changed to 2 1 0. In the example in Figure 4.28a and Listing 4.23, the note index sequence 0 1 2 3 2 1 is used which results in an alternating ascending and descending arpeggio (see Figure 4.28b).

```

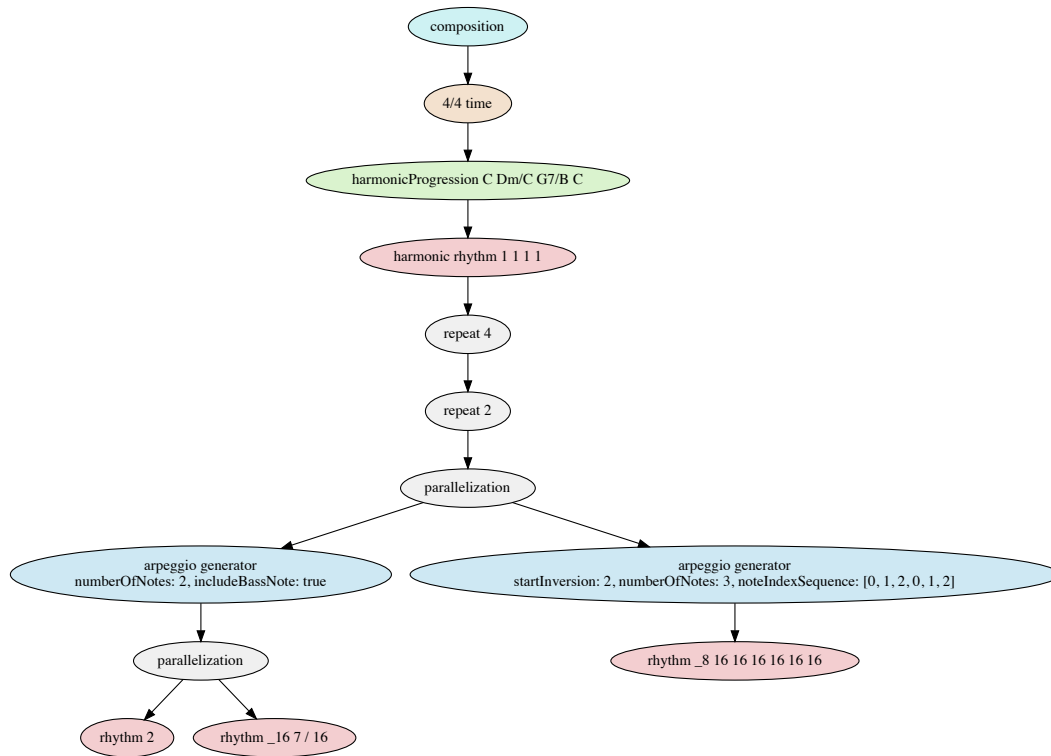
1 composition
2 {
3   time 4/4
4   {
5     harmonicProgression C Dm/C G7/B-B C, harmonicRhythm 1 1 1 1
6     {
7       repeat 4, repeat 2
8       {
9         parallel
10        {
11          arpeggioGenerator (numberOfNotes 2 includeBassNote true)
12          {
13            parallel
14            {
15              rhythm 2
16              rhythm _16 7/16
17            }
18          }
19          arpeggioGenerator (numberOfNotes 3 startInversion 2
20                             noteIndexSequence 0 1 2 0 1 2)
21          {
22            rhythm _8 16 16 16 16 16 16
23          }
24        }
25      }
26    }
27  }
28 }

```

Listing 4.24: Language representation of the model in Figure 4.29a producing J. S. Bach, *Prelude in C Major, BWV 846*, mm. 1–4.

A more complex example is demonstrated in Figure 4.29 and the corresponding Listing 4.24. The model produces the first four measures of J. S. Bach’s *Prelude in C Major, BWV 846*. Two separate arpeggio generators are used to generate independent arpeggios for the left and the right hand. An advanced feature is used in the third chord in the third measure. The harmony is specified as G^7 with B in

the bass. Additionally, a so called *note exclusion* with the syntax `-B` is specified. It instructs the compiler to skip the relevant note during the chord inversion computing process. As can be seen in measure 3 in the score (Figure 4.29b), the note B is not present in the arpeggio. To account for this, specific notes can be skipped in the chord generation process.



(a) Context tree model representation containing two arpeggio generators



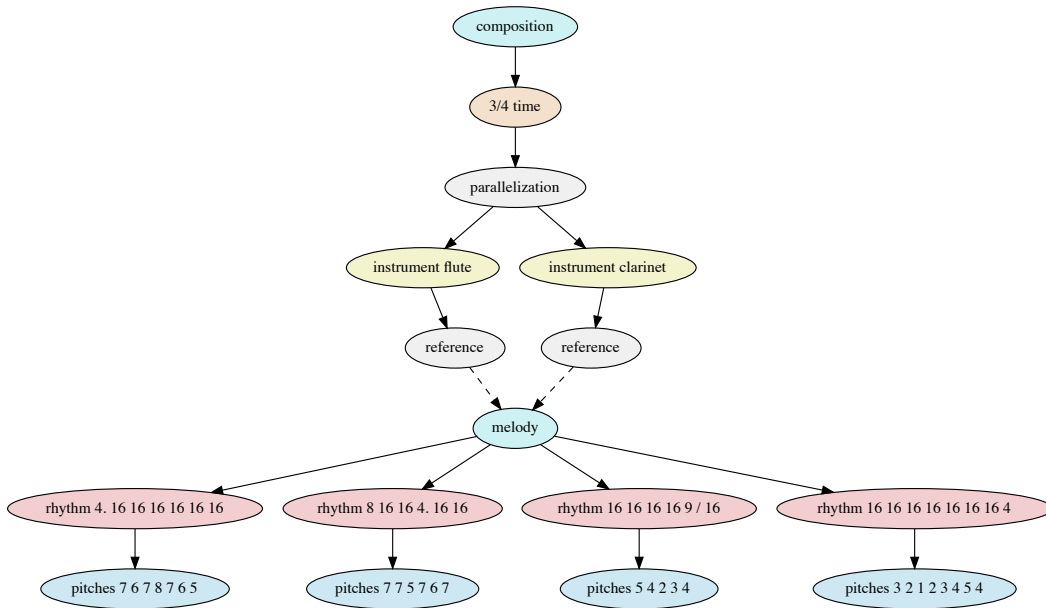
(b) Score representation

Figure 4.29: Score and context tree model of J. S. Bach, *Prelude in C Major, BWV 846*, mm. 1–4. Corresponding files can be found on the accompanying CD under **Examples/Compositions/Bach/BWV846_Prelude_in_C_Major** (see Appendix A).

4.8 Control Structures

Control structures are utilized to dynamically reuse contexts in context tree models with the help of loops, iterative modifications and other algorithmic constructs, which are explained in detail in this section.

4.8.1 Parallelizations



(a) Context tree model representation



(b) Score representation. Note that the clarinet is notated in Bb.

Figure 4.30: Score and context tree model of a simultaneously played excerpt from *Boléro* by Maurice Ravel. Corresponding files are available on the accompanying CD under Examples/Compositions/Ravel/Bolero (see Appendix [A](#)).

Parallelizations are used to indicate that nested tree branches are not to be evaluated sequentially, but in parallel. This results in individual musical streams resulting in multiple parts or voices being played simultaneously.

As an example, a parallel version of an already introduced context tree model is shown. The model in Figure [4.30](#) contains a *parallelization* node to cause the melody being played simultaneously by flute and clarinet. Syntactically, this is

```

1 composition
2 {
3     time 3/4
4     {
5         parallel
6         {
7             instrument flute
8             {
9                 fragmentRef melody
10            }
11            instrument clarinet
12            {
13                fragmentRef melody
14            }
15        }
16    }
17 }
18
19 fragment melody
20 {
21     rhythm 4. 16 16 16 16 16 16, pitches 7 6 7 8 7 6 5
22     rhythm 8 16 16 4. 16 16, pitches 7 7 5 7 6 7
23     rhythm 16 16 16 16 9/16, pitches 5 4 2 3 4
24     rhythm 16 16 16 16 16 16 16 4, pitches 3 2 1 2 3 4 5 4
25 }
    
```

Listing 4.25: Syntactical representation of the *Boléro* excerpt shown in Figure 4.30a

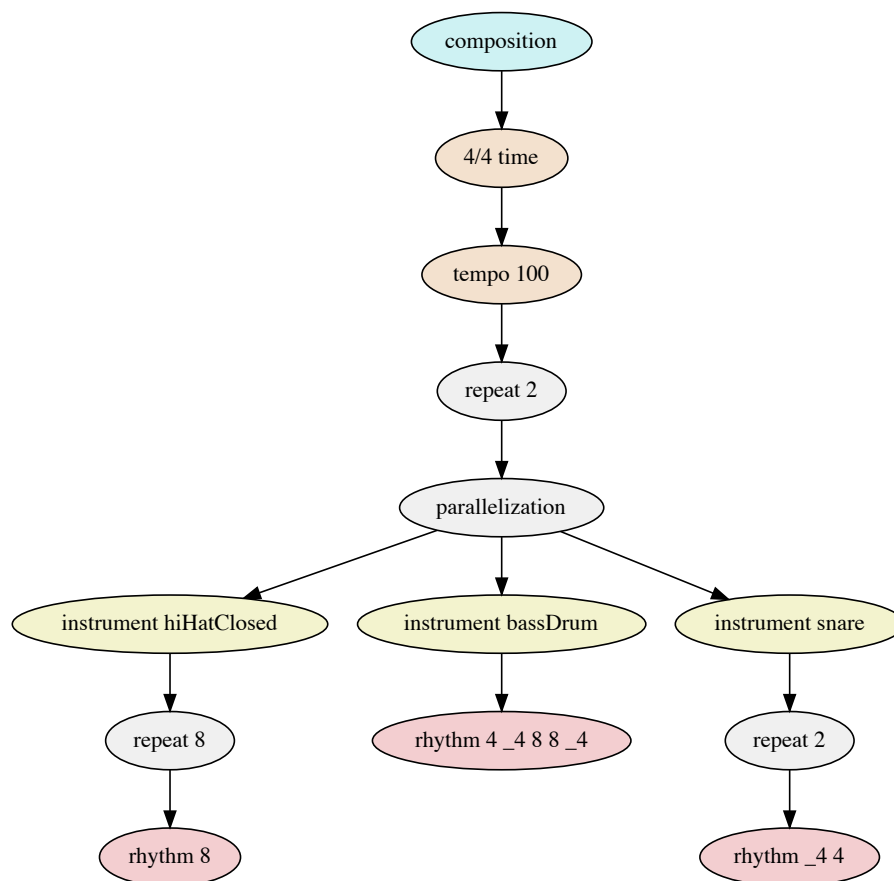
accomplished with the `parallel` keyword as demonstrated in Listing 4.25. Compare with the model already presented in Figure 4.11a, which results in sequentially played melodies.

4.8.2 Repetitions

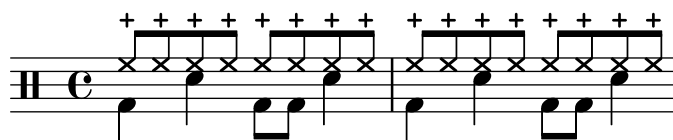
Repetition is a frequently utilized technique in music composition and is applied in a variety of forms. As Arnold Schoenberg puts it, “intelligibility in music seems to be impossible without repetition” (Ockelford 2016, I, p. 2). A common form of repetitions is known from musical scores, in which repeat signs indicate that a section of the score is to be played again (as an example, refer to score 4.23).

In MPS, arbitrary subtrees of contexts can be repeated, which can be applied to single contexts or combinations of musical contexts. Furthermore, repetitions can be nested hierarchically. This is demonstrated using a context tree model of a simple drum groove shown in Figure 4.31. The corresponding language representation is demonstrated in Listing 4.26.

The model contains nested control structures to repeat context subtrees. The outer structure (`repeat 2`) repeats the whole measure produced by the subtree below the `parallel` element. It produces musical material for closed hi-hats, bass drum and snare. A nested repetition resulting in 8 eighths notes is specified for the hi-hats. Also, the snare drum repeats the rhythmic pattern of a quarter rest followed by a quarter note (`rhythm _4 4`) twice, which is also expressed as a nested repetition.



(a) Context tree model representation



(b) Score representation

Figure 4.31: Score and context tree model of a simple drum groove containing nested repetitions. Corresponding files are available on the accompanying CD under **Examples/Model/Repetitions** (see Appendix [A](#)).

```

1 composition
2 {
3   time 4/4, tempo 100
4   {
5     repeat 2
6     {
7       parallel
8       {
9         instrument hiHatClosed
10        {
11          repeat 8
12          {
13            rhythm 8
14          }
15        }
16        instrument snare
17        {
18          repeat 2
19          {
20            rhythm _4 4
21          }
22        }
23        instrument bassDrum
24        {
25          rhythm 4 _4 8 8 _4
26        }
27      }
28    }
29  }
30 }

```

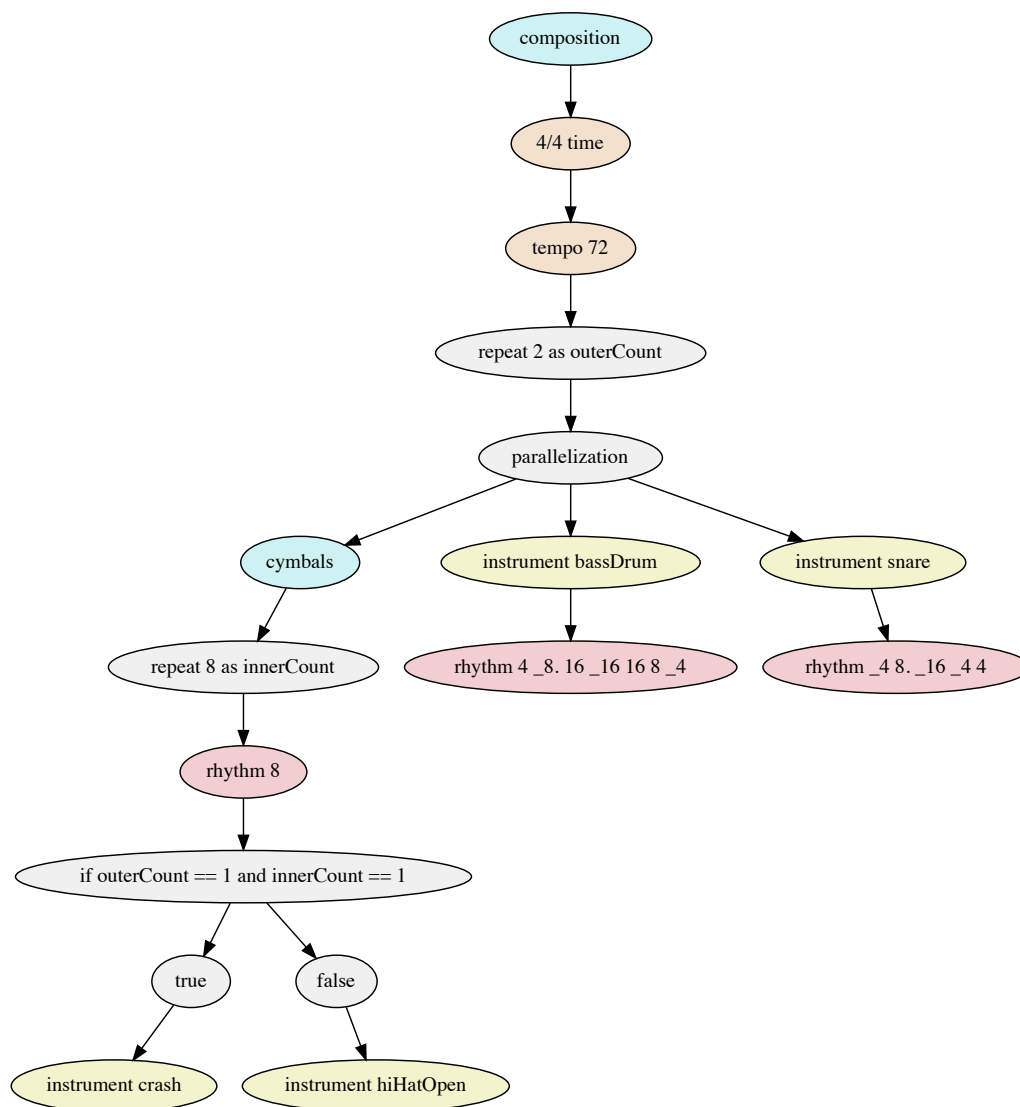
Listing 4.26: Syntactical representation of the context model shown in Figure 4.31a using nested repeated structures.

In this manner, repetitions of musical context subtrees can be hierarchically nested in arbitrary complexity. The repeat count can be bound to a variable, which can be utilized to introduce conditional contexts. This technique is demonstrated in the following section.

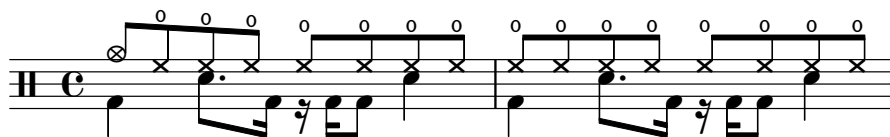
4.8.3 Conditions

Condition control structures are used to define conditional contexts. Therefore, an expression is defined which is evaluated to a boolean expression, yielding either **true** or **false**. Depending on the result, a different context tree branch is followed. This is illustrated in the model in Figure 4.32, which produces the drum introduction of *In My Place* by Coldplay.

The contexts for the cymbals are specified conditionally in this context model. A condition based on the current repetition counts of an outer and an inner **repeat** control structure is specified. It evaluates to true if both the outer and inner repetition count is 1. If this is the case, a crash cymbal is used as instrument context. In all other cases, open hi-hats are played. In the two measures shown in Figure 4.32b, it can be seen that the condition evaluates to **true** only in the first measure on the first beat, on which a crash cymbal is played. On all other beats, especially



(a) Context tree model representation



(b) Score representation

Figure 4.32: Context tree model and resulting drum introduction of Coldplay's *In My Place*. Corresponding files are available on the accompanying CD under **Examples/Compositions/Coldplay/In My Place** (see Appendix [A](#)).

```

1 composition
2 {
3   time 4/4, tempo 72
4   {
5     repeat 2 as outerCount
6     {
7       parallel
8       {
9         fragment cymbals
10        {
11          repeat 8 as innerCount
12          {
13            rhythm 8
14            {
15              if outerCount == 1 and innerCount == 1
16              {
17                instrument crash
18              }
19              else
20              {
21                instrument hiHatOpen
22              }
23            }
24          }
25        }
26
27        instrument bassDrum
28        {
29          rhythm 4 _8. 16 _16 16 8 _4
30        }
31
32        instrument snare
33        {
34          rhythm _4 8. _16 _4 4
35        }
36      }
37    }
38  }
39 }
40

```

Listing 4.27: Syntactical representation of the context tree model in Figure 4.32a producing the drum introduction of Coldplay’s *In My Place*

on the first beat in the second measure, an open hi-hat is played because the outer repetition count evaluates to 2 in the second measure.

Condition expressions can be based on arbitrary variables defined in any context nodes which are hierarchically placed above the current condition node. Notably, results of function calls can be used to create dynamically modeled compositions using conditional contexts. Refer to section 4.9.4 for more details.

4.8.4 Iterations

Iterations are used to create loops in which musical material is iteratively modified. The control structure resembles **for** loops in general purpose programming languages. Iterations define a control variable which typically changes its value in every loop iteration. The model in Figure 4.33 and the corresponding code in Listing 4.28 demonstrate an iteration producing a G minor blues scale (see Figure 4.25).

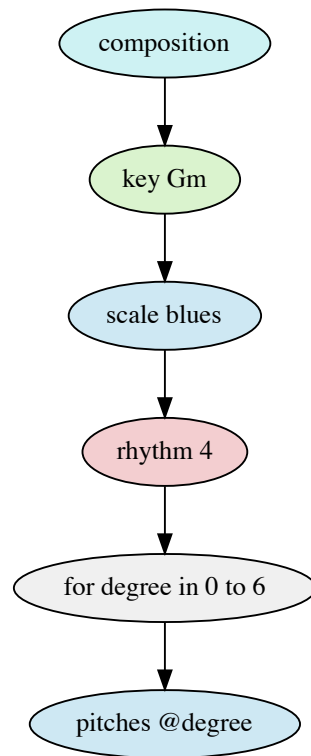


Figure 4.33: Context tree model using an iteration to produce a G minor blues scale.

```

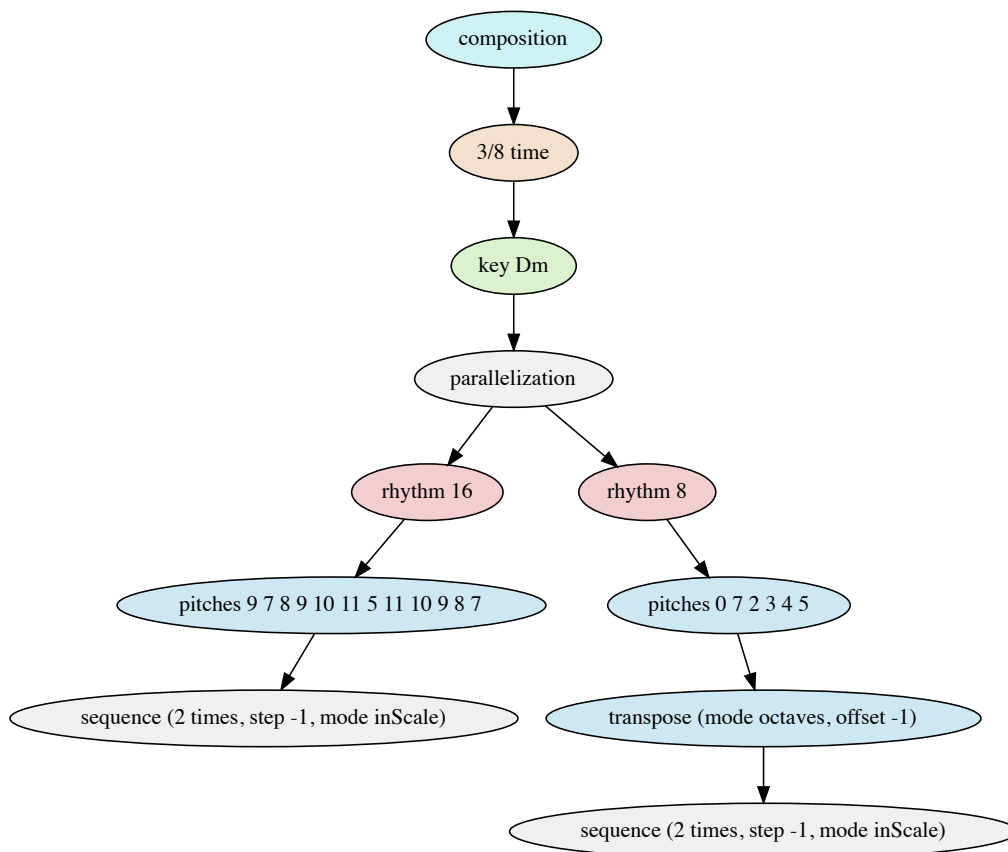
1 composition
2 {
3   key Gm
4   {
5     scale blues
6     {
7       rhythm 4
8       {
9         for degree in 0 to 6
10        {
11          pitches @degree
12        }
13      }
14    }
15  }
16 }
  
```

Listing 4.28: Iteration producing a G minor blues scale

Also refer to section [4.6.1](#) in which a rhythmic pattern is iteratively displaced using a corresponding control structure and a suitable rhythmic modifier.

4.8.5 Sequences

In melodic sequences, a musical phrase is repeatedly played with the same rhythm but transposed pitches, “the succession of pitch levels rising or falling by the same or similar intervals” (Randel 2003, p. 768). MPS provides a separate control structure for melodic sequences. Technically, melodic sequences are translated to an iteration with nested transpositions.



(a) Context tree model representation



(b) Score representation

Figure 4.34: Score and context tree model of a sequence from J. S. Bach’s *Invention No. 4 in D minor*, BWV 775, mm. 7–10. Corresponding files are available on the accompanying CD under Examples/Compositions/Bach/BWV775_Invention_No4_in_D_Minor/SequenceExample (see Appendix A).

Table 4.16: Sequence control structure parameters

Parameter	Description
times	Specifies how often the sequence is repeated.
step	Defines the offset of the iteratively applied transposition. The unit of this expression is defined by the mode parameter.
mode	Defines the unit of the interval expression. Three modes are available: absolute for semitone-based transpositions, inScale to perform transpositions of scale degrees and octaves for octave translations. If the parameter is not specified, the default absolute will be used.

The model in Figure 4.34 represents a sequence from J. S. Bach’s *Invention No. 4 in D minor, BWV 775*. The model can be syntactically represented as shown in Listing 4.29. Sequence control structures are applied both to the right hand and the left hand part. Both sequence control structures are applied twice (2 **times**). In the first iteration, the specified pitches are adopted without modification. In the second iteration, the pitches are transposed one step down. Consequently, the scale degrees of both parts are diatonically transposed down in parallel. Refer to Table 4.16 for detailed parameter descriptions.

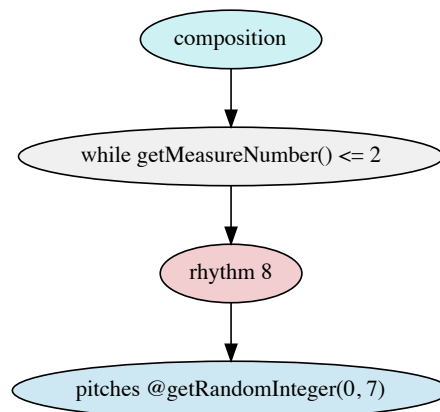
```

1 composition
2 {
3   time 3/8, key Dm
4   {
5     parallel
6     {
7       rhythm 16
8       {
9         pitches 9 7 8 9 10 11 5 11 10 9 8 7
10        {
11          sequence 2 times step -1 mode inScale
12        }
13      }
14      rhythm 8
15      {
16        pitches 0 7 2 3 4 5, transpose mode octaves -1
17        {
18          sequence 2 times step -1 mode inScale
19        }
20      }
21    }
22  }
23 }
```

Listing 4.29: Language representation of the context tree model of a melodic sequence from J. S. Bach’s *Invention No. 4 in D minor, BWV 775* shown in Figure 4.34a.

4.8.6 While-Loops

The contents of while-loops are applied as long as a specified condition is fulfilled. An example is demonstrated in Figure 4.35. The loop is applied while the measure number is less than or equal to 2 (i.e. in the first two measures). The current measure number can be retrieved using the function `getMeasureNumber()`. Pitches are chosen randomly using another function call to `getRandomInteger()`. Refer to section 4.9.4 for more details on function calls.



(a) Context tree model representation



(b) Score representation

Figure 4.35: Context tree model and resulting score demonstrating a while loop to generate randomly pitched notes. Corresponding files are available on the accompanying CD under `Examples/Model/While` (see Appendix A).

```

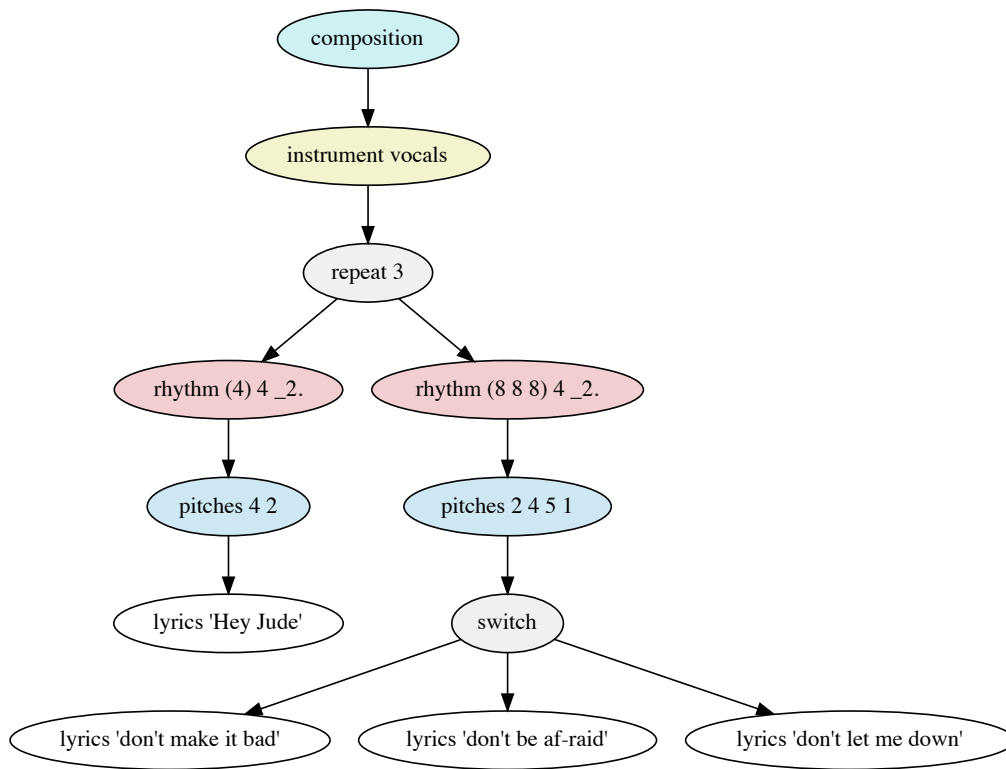
1 composition
2 {
3     while getMeasureNumber() <= 2
4     {
5         rhythm 8
6         {
7             pitches @getRandomInteger(0, 7)
8         }
9     }
10 }
    
```

Listing 4.30: Language representation of the context tree model in Figure 4.35a demonstrating a while loop to generate randomly pitched notes.

4.8.7 Switches

This control structure selects and processes only one of the specified child tree branches for each invocation. If the structure is encountered again (e.g. due to a repeat), the next child branch is processed. If no more child branches are available, processing continues from the first child branch again.

An example is provided in the context tree model in Figure 4.36, in which the same melody is repeated three times. The switch control structure applies three different lyric contexts for each loop iteration. The corresponding language representation is shown in Listing 4.31.



(a) Context tree model representation



(b) Score representation

Figure 4.36: Context tree model and resulting score demonstrating a switch control structure. Corresponding files are available on the accompanying CD under `Examples/Model/Switch` (see Appendix A).

```

1 composition
2 {
3     instrument vocals
4     {
5         repeat 3
6         {
7             rhythm (4) 4 _2.
8             {
9                 pitches 4 2
10                {
11                    lyrics "Hey Jude"
12                }
13            }
14            rhythm (8 8 8) 4 _2.
15            {
16                pitches 2 4 5 1
17                {
18                    switch
19                    {
20                        lyrics "don't make it bad"
21                        lyrics "don't be af-raid"
22                        lyrics "don't let me down"
23                    }
24                }
25            }
26        }
27    }
28 }

```

Listing 4.31: Language representation of the context tree model in Figure 4.36a demonstrating a switch control structure.

It is also possible to define non-consecutive processing sequences. This is done by specifying a so called *child index sequence*, as demonstrated in Listing 4.32.

```

1 switch childIndexSequence 0 0 1

```

Listing 4.32: Switch control structure defining a custom child index sequence

The switch specified in Listing 4.32 will process the first child branch twice, followed by the second child branch. If invoked again, processing will start over at the beginning of the custom sequence.

4.9 Expressions

Expressions are used to specify dynamic parameters in context tree models. These are especially useful for algorithmic composition, in which certain musical parameters are computed based on mathematical rules. MPS uses a custom expression language supporting logical and arithmetic expressions with variables and function calls.

Table 4.17: Expression language literals

Type	Description	Internal Type
boolean	Boolean value. Permitted literals are <code>true</code> and <code>false</code> .	<code>boolean</code>
integer	Integer number with optional negative sign, such as 42, -23 or 0.	<code>int</code>
float	Floating point number with optional negative sign, such as 3.1415 or -2.1.	<code>double</code>
fraction	Fraction represented by an integer numerator and integer denominator, for instance 1/4. Arithmetic divisions automatically result in a fraction if both operands are integer numbers.	<code>Fraction</code> ³
string	Represents a sequence of zero or more characters encoded in UTF-16 ⁴	<code>String</code> ⁵

4.9.1 Literals

A basic unit of information in the expression language is given in the form of literals. Refer to Table 4.17 containing a summary of available literal types.

4.9.2 Operators

The system supports boolean operators, comparison operators and arithmetic operators. The operators are listed in Table 4.18 ordered by operator priority, from highest to lowest precedence. Parentheses may be used for custom operator prioritization, for example:

```
1 (2 + 3) * 4
```

Listing 4.33: Example expression containing a parenthesized expression

In the expression in Listing 4.33, the term 2+3 is evaluated first and the result is multiplied with 4. If no parentheses would be used, 3*4 would be evaluated first due to higher precedence of the multiplication operator.

Table 4.18: Expression language operators ordered by priority

Operator	Description
!	Unary boolean negation operator. For example, <code>!true</code> evaluates to <code>false</code> .
-	Unary arithmetic negation. For example, <code>-(2+1)</code> evaluates to <code>-3</code> .
*	Arithmetic multiplication

Continued on next page

Table 4.18 – *Continued from previous page*

Operator	Description
/	Arithmetic division. Results in a fraction if both operands are integer numbers.
%	Modulo operator
+	Arithmetic addition. May also be used to concatenate strings.
-	Arithmetic subtraction
==	Evaluates to true if the left operand is equal to the right operand.
!=	Evaluates to true if the left operand is not equal to the right operand.
<	Evaluates to true if the left operand is less than the right operand.
>	Evaluates to true if the left operand is greater than the right operand.
<=	Evaluates to true if the left operand is equal to or less than the right operand.
>=	Evaluates to true if the left operand is equal to or greater than the right operand.
and	Boolean <i>and</i> operator. Result of the expression is true if and only if both operands evaluate to true .
or	Boolean <i>or</i> operator. Result of the expression is true if at least one of the operands evaluates to true .

4.9.3 Type Conversions

Expressions are dynamically cast if required. For example, to evaluate the expression in Listing 4.34, several dynamic type casts are applied.

```
1 1 + 0.7 > 3/4 and !(n % 2)
```

Listing 4.34: Expression requiring dynamic type casts

To sum $1 + 0.7$, 1 is implicitly converted to a floating point number. To evaluate the comparison $1.7 > 3/4$, 1.7 is automatically converted to the fraction $\frac{17}{10}$. The result of the left-hand comparison $\frac{17}{10} > \frac{3}{4}$ yields **true**. The modulo operation on the right hand side results in the remainder of n being divided by 2 (Flanagan 2005, p. 33). The remainder is wrapped in a boolean negation. This implies that the remainder must implicitly be cast to a boolean expression. It evaluates to **false** if the remainder is equal to zero and to **true** otherwise. The boolean result of this implicit cast is negated and then used as right operand for the *and* conjunction.

Table 4.19: Implicit type conversion rules. If two different types are combined as operands of an operator, one operand will implicitly be converted to the resulting type.

Type 1	Type 2	Resulting Type
boolean	integer	integer
boolean	float	float
boolean	fraction	fraction
boolean	string	string
integer	float	float
integer	fraction	fraction
integer	string	string
float	fraction	fraction
float	string	string
fraction	string	string

The right hand side of the *and*-operator can also be read as: ‘if n is dividable by 2’. Refer to Table 4.19 for an overview of implicit type conversion rules and Table 4.21 for an explanation of the applied transformations.

4.9.4 Function Calls

Functions are used to dynamically retrieve musical context information. They are evaluated during the compilation process (see section 5.2). The returned values depend on the given parameters, the stream context and the temporal context in which they are invoked. Refer to Table 4.20 for an overview of available functions.

Table 4.20: Table of available functions

Signature	Return Type	Description
<code>getRootNote()</code>	<code>NoteReference</code>	Returns the root note of the current context harmony.
<code>getBassNote()</code>	<code>NoteReference</code>	Returns the bass note of the current context harmony, which can in some cases be different from the root note.
<code>getRandomBoolean()</code>	<code>boolean</code>	Returns a random boolean value, i.e. <code>true</code> or <code>false</code> .
<code>getRandomInteger(min, max)</code>	<code>integer</code>	Returns a random integer value between <code>min</code> (inclusive) and <code>max</code> (exclusive).

Continued on next page

Table 4.20 – *Continued from previous page*

Operator	Description	
<code>getRandomDouble(min, max)</code>	<code>double</code>	Returns a random double value between <code>min</code> (inclusive) and <code>max</code> (exclusive).
<code>getTime()</code>	<code>fraction</code>	Returns the current time in the current stream in terms of note duration (e.g. after a quarter note, the elapsed time is $\frac{1}{4}$). Refer to section 3.4 for more details.
<code>getTimeSignature()</code>	<code>TimeSignature</code>	Returns the current time signature.
<code>isInFragmentContext(string)</code>	<code>boolean</code>	Returns <code>true</code> if the current context stack contains the fragment with the given name, <code>false</code> otherwise.

4.10 Implementation Details

The last part of this chapter provides technical details about the architecture and implementation of the context tree composition model and the language infrastructure in [MPS](#).

4.10.1 Composition Domain Model

The core domain model for [MPS](#) contains all basic data structures which are needed to represent musical compositions. It is comprised of about 125 data structure types (also referred to as *classes* in object-oriented programming) and about 15 enumerations of domain-specific concepts. Its class diagram showing all data types and associations can be found on the accompanying CD (see Appendix [A](#)). Due to its huge dimensions, printing is not possible using common page layouts.

The model was developed using the [Eclipse Modeling Framework \(EMF\)](#)⁶, which is a toolkit for Java developers to create and edit data model definitions. Moreover, the framework offers the functionality to generate programming code from model definitions, which can later be extended easily. By this means, developers do

⁶<https://www.eclipse.org/modeling/emf/>

Table 4.21: Type cast specifications. Source types are listed on the left, target types are listed in the columns on top.

	boolean	integer	float	fraction	string
boolean	-	false \Rightarrow 0 true \Rightarrow 1	false \Rightarrow 0.0 true \Rightarrow 1.0	false $\Rightarrow \frac{0}{1}$ true $\Rightarrow \frac{1}{1}$	false \Rightarrow 'false' true \Rightarrow 'true'
integer	false if equal to 0, true otherwise	-	as specified by <code>doubleValue</code> ⁷	$\frac{n}{1}$	as specified by <code>valueOf</code> ⁸
float	false if equal to 0.0, true otherwise	$\lfloor f \rfloor$, i.e. the nearest integer below the value of the floating point number f	-	Nearest computable fraction as specified by <code>Fraction</code> ⁹	as specified by <code>valueOf</code> ¹⁰

Continued on next page

⁷[https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#doubleValue\(\)](https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#doubleValue())

⁸[https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#valueOf\(int\)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#valueOf(int))

⁹<https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#>

`Fraction(double)`

¹⁰[https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#valueOf\(double\)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#valueOf(double))

Table 4.21 – *Continued from previous page*

	boolean	integer	float	fraction	string
fraction	false if fraction is equal to $\frac{0}{1}$, true otherwise	Whole number part of the fraction as specified by intValue ¹¹	as specified by doubleValue ¹²	-	as specified by toString ¹³
string	false if string is empty, true otherwise	as specified by parseInt ¹⁴	as specified by parseDouble ¹⁵	Supported if string contains two integer numbers separated by a slash (/) or a single integer number	-

¹¹[https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#intValue\(\)](https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#intValue())

¹²[https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#doubleValue\(\)](https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#doubleValue())

¹³[https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#toString\(\)](https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/fraction/Fraction.html#toString())

¹⁴[https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#parseInt\(java.lang.String\)](https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#parseInt(java.lang.String))

¹⁵[https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html#parseDouble\(java.lang.String\)](https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html#parseDouble(java.lang.String))

not have to write circumstantial programming code (such as methods to access or change model attributes and infrastructure for notifications of model changes) for each model element themselves. Instead, developers are able to focus on essential programming tasks (Steinberg et al. [2008](#)). Moreover, [EMF](#) provides facilities for model object persistence, i.e. saving and loading models to/from files.

The composition domain model forms the basis of all other models developed in this dissertation, namely the context layer model (see Chapter [3](#)), the context tree model (introduced in this chapter) and the score model (to be described in Chapter [5.3](#)).

4.10.2 Domain-Specific Composition Language

The domain-specific description language for context tree composition models MC²L proposed in this dissertation was designed using the framework Xtext^{[16](#)}. Xtext is a powerful framework for developing programming languages and domain-specific languages.

In order to develop a computer language, developers provide a grammar in a custom Xtext syntax. It contains grammar rules based on the [EBNF](#) and bindings to a data model. MC²L is based on the composition domain model introduced in the previous section. The complete language grammar is attached in full length in Appendix [B.1](#). Based on the grammar, Xtext generates a sophisticated language infrastructure including lexer, parser, serializer, validation facilities and an editor for the language with basic syntax coloring, automatic code completion, outline view, hyperlinking and refactoring support. Based on the generated code, all components of the language infrastructure can further be refined and extended. Refer to Figure [4.37](#) for a visual impression of the editor.

4.11 Summary

Context tree composition models accommodate hierarchically arranged elements for describing musical structures, according modifications and algorithmic processes. Musical information can be reused in various ways, including inheritance, polymorphism, auto expansion and modularization. Expressions enable the specification of dynamic musical parameters. An advanced graphical editor is available for the development of composition models in a corresponding domain-specific language. The model representations proposed in the previous two chapters allow novel approaches for musical applications, which are demonstrated in the following chapters.

¹⁶<http://www.eclipse.org/Xtext/>

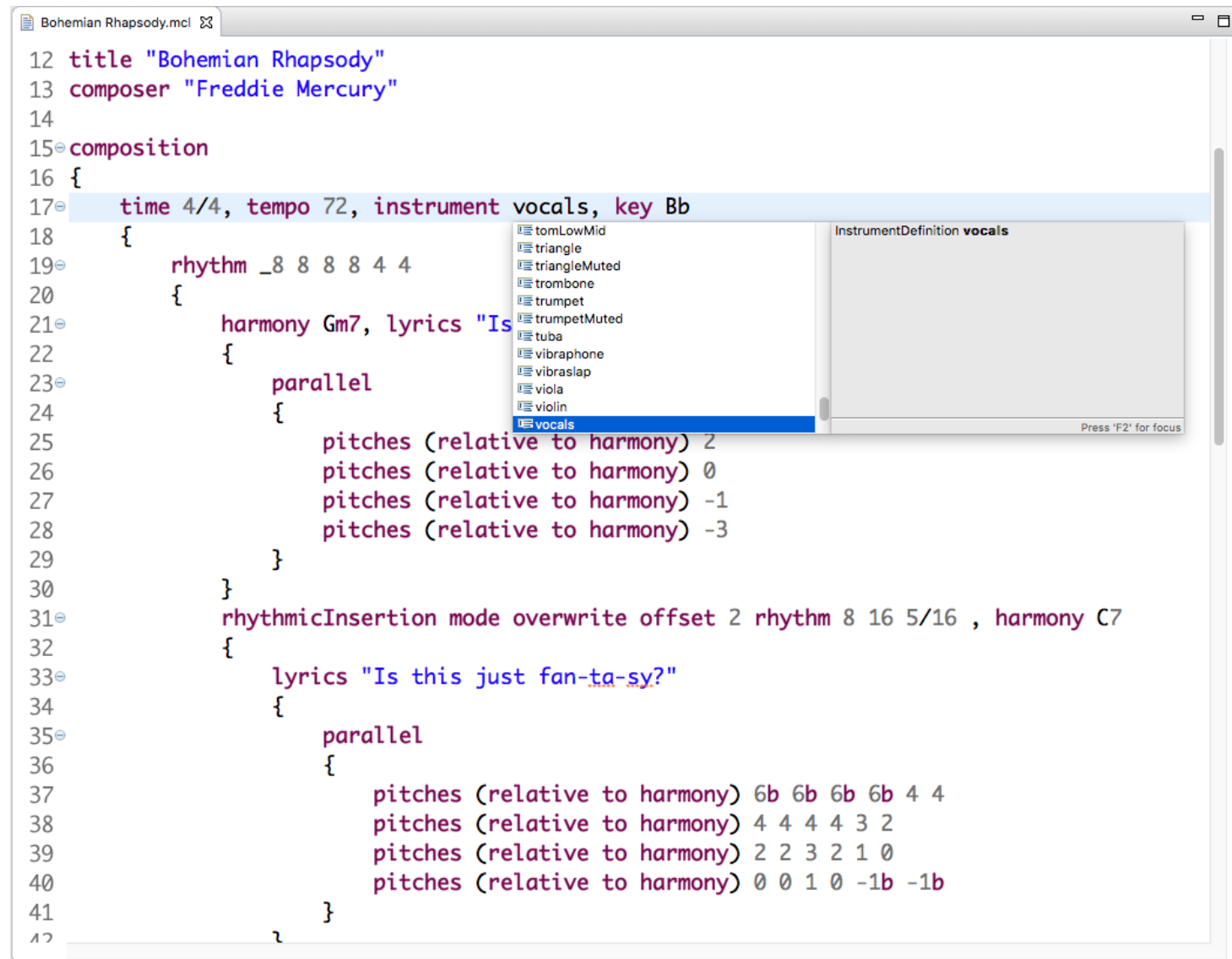


Figure 4.37: Screenshot of the editor for the musical context composition language. Automatic code completion is currently invoked.

Part II

System Applications

Chapter 5

Model Transformations

The old idea of a composer suddenly having a terrific idea and sitting up all night to write it is nonsense. Nighttime is for sleeping.

— Benjamin Britten

Music Processing Suite provides an extensive infrastructure for music format conversions. This includes conversions between the music model representations introduced in the previous chapters. Furthermore, these model representations can be derived from standard formats such as `MIDI` and MusicXML. This chapter includes in-depth descriptions for all relevant transformations.

5.1 Transformation Infrastructure Overview

An overview of possible transformations is shown in Figure `5.1`. The upper left part of the illustration relates to the internal model representations of `MPS`, which can be translated to `MC2L` language representations and stored in corresponding files. An appropriate parser is responsible for translating language representations into equivalent context tree representations. In the other direction, a serializer performs the task of translating a context tree representation into its corresponding language representation. Implementation details of these transformations were already described in Chapter `4.10`.

The left side of the Figure presents possible transformations regarding symbolic standard formats for music, namely `MIDI` and MusicXML. With appropriate parsers, scores available in the named formats can be converted to context layer models. These transformations are covered in Chapters `5.5` and `5.6`.

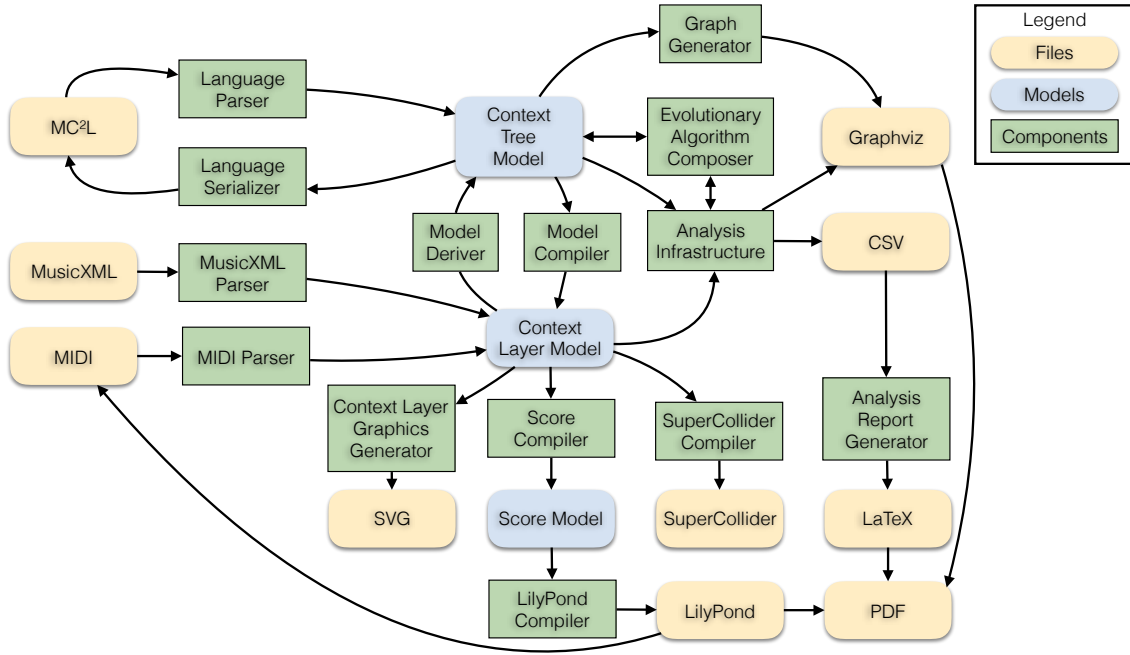


Figure 5.1: Music Processing Suite transformation infrastructure

In the center of the diagram, internal model transformations are shown. This involves transforming context tree models into context layer models (covered in section 5.2) and various formats into which context layer models can be converted, such as musical scores (described in Section 5.3) and the SuperCollider language (Section 5.4). Finally, an algorithm is proposed for deriving context tree models from stream models, which closes the circle of the most important conversions available in MPS. The other components visible in Figure 5.1, namely the analysis infrastructure and the evolutionary composer component, are explained in Chapters 7 and 8, respectively.

5.2 Transforming Context Tree Models to Context Layer Models

A crucial transformation in MPS is the translation of context tree models into context layer models, which can be described as an expansion of a compressed composition model into an equivalent time-based representation. This functionality is provided by a compiler. Its mode of operation is explained using the context tree model and resulting data structures in Figure 5.2.

The compiler traverses the context tree recursively from top to bottom. Each encountered node is put on a stack until the compiler reaches a leaf node (i.e. a node which has no child nodes). At each leaf node, the current musical contexts are de-

terminated. This can be achieved by looking for the following contexts in the stack, returning the first context found of the specified type, respectively:

1. Harmonic Progressions
2. Harmonic Rhythms
3. Rhythms
4. Pitches
5. Instruments
6. Keys
7. Harmonies
8. Lyrics
9. Custom Contexts

In case a context is not found in the stack, a default context will be assigned if applicable. Default contexts are listed in Table 5.1.

Table 5.1: Default context values

Context	Default Value
Instrument	Piano
Key	C Major
Harmony	C Major
Time Signature	$\frac{4}{4}$
Tempo	120 BPM

If the compiler encounters a parallelization control structure (the semantics of which were introduced in section 4.8.1), it is put on a separate stack. The corresponding stack entry also contains the current parallelization index, indicating which child structure of the parallelization is currently processed.

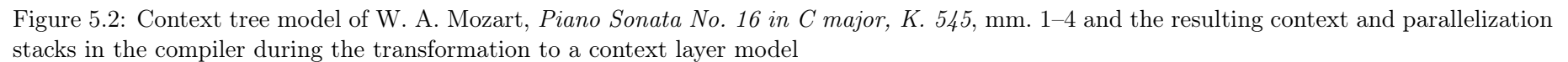


Figure 5.2: Context tree model of W. A. Mozart, *Piano Sonata No. 16 in C major*, K. 545, mm. 1–4 and the resulting context and parallelization stacks in the compiler during the transformation to a context layer model

As illustrated in Figure 5.2, the context stack contains the following items at the leftmost leaf node:

1. `harmonic progression C G7 C F C G7-F4 C`
2. `harmonic rhythm 1 2 2 2 2 2 2`
3. `rhythm 2 4 4 4. 16 16 4 _4`
4. `pitches (startOctave 5) 0 2 4 -1 0 1 0`

Furthermore, the following default contexts are added:

1. `instrument piano`
2. `key C`
3. `time 4/4`
4. `tempo 120`

First, the harmonic progression context layer is generated. To determine the duration of the given harmonies, a harmonic rhythm context is required, as already explained in section 4.5.8. Subsequently, rhythm-based events are generated. Therefore, rhythmic notes are matched with pitches and lyric syllables, if applicable. *Auto expansion* is applied if possible, as introduced in section 4.4.4. The remaining contexts are added to the resulting context layer model with the duration of the currently processed rhythm (in the example, the duration of the rhythm `2 4 4 4. 16 16 4 _4` is 2 measures). The intermediate result after processing the leftmost leaf node is shown in Figure 5.3.

The context stack at the second leaf node contains a context modifier, namely a rhythmic insertion. If one or more modifiers are present in the stack, they are applied after the context identification process in the order they appear in the context tree model (i.e. the reverse order they appear in the context stack). In the case of the example shown in Figure 5.2, the second half of the original rhythm used in the first two measures is replaced.

Finally, context generators are executed if present in the stack at a leaf node. In the example, this is the case at the third, rightmost leaf node. It contains an *arpeggio generator* which creates the left hand accompaniment. It is invoked 8 times as the compiler unfolds the corresponding `repeat 8` control structure. The semantics and parameters of arpeggio generators were already explained in section 4.7.2. The resulting context layer model of the transformation is shown in Figure 5.4.

Stream 1	Meter (4)	4/4 time				4/4 time																			
	Key (1)	C																							
	Harmony (7)	C				G7				C				F		C		G7-F		C					
	Harmonic Rhythm (7)	1				2				2				2		2		2		2					
	Rhythm (17)	2		4		4		4		16		16		4		_4									
	Scale (1)	major																							
	Degrees (17)	0		2		4		-1		0		1		0											
	Pitches (17)	C5		E5		G5		B4		C5		D5		C5											
	Loudness (1)	loudness mf																							
	Time (Measures)				1				2				3				4								
	Time (Absolute)				0				1				2				3								

Figure 5.3: Intermediate context layer model of W. A. Mozart, *Piano Sonata No. 16 in C major, K. 545*, mm. 1–4, after processing the leftmost leaf node of the model shown in Figure 5.2

Stream 1	Meter (4)	4/4 time										4/4 time										4/4 time																								
	Key (1)	C																																												
	Harmony (7)	C										G7										C					F					C					G7-F					C				
	Harmonic Rhythm (7)	1										2										2					2					2					2									
	Rhythm (17)	2		4				4		4		16		16		4		_4		2		4		4		4		8		16		16		4		_4										
	Scale (1)	major																																												
	Degrees (17)	0		2				4		-1		0		1		0				5		4		7		4		3		2		3		2												
	Pitches (17)	C5		E5				G5		B4		C5		D5		C5				A5		G5		C6		G5		F5		E5		F5		E5												
	Loudness (1)	loudness mf																																												
Stream 2	Meter (4)	4/4 time										4/4 time										4/4 time																								
	Key (1)	C																																												
	Harmony (7)	C										G7										C					F					C					G7-F					C				
	Harmonic Rhythm (7)	1										2										2					2					2					2					2				
	Rhythm (32)	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8											
	Scale (1)	major																																												
	Pitches (32)	C4	G4	E4	G4	C4	G4	E4	G4	D4	G4	F4	G4	C4	G4	E4	G4	C4	A4	F4	A4	C4	G4	E4	G4	B3	G4	D4	G4	C4	G4	E4	G4													
	Loudness (1)	loudness mf																																												
	Time (Measures)		1										2										3										4													
Time (Absolute)		0										1										2										3														

Figure 5.4: Context layer model of W. A. Mozart, *Piano Sonata No. 16 in C major, K. 545*, mm. 1–4. Corresponding input and output files are available on the accompanying CD under Examples/Compositions/Mozart/KV545_SonataFacile (see Appendix A).

5.3 Transforming Context Layer Models to Score Representations

The previously described context layer model transformation is already an essential intermediate step in order to produce musical scores. The *score compiler* is responsible for converting all time-based context layer model events with score-specific events such as time signature changes, key changes, tempo specifications, notes and rests.

The implementation uses stream sequencers to traverse each available stream, which results in a sequence of stream events (see Chapter 3.6). Each stream event is converted to a score-specific event. In most cases, this results in a note or a rest. Otherwise, instructions for metric changes, key/harmony changes and clefs are added. This results in an abstract score model which is convertible to any symbolic score format. The structure of the abstract score model is illustrated in Figure 5.5¹.

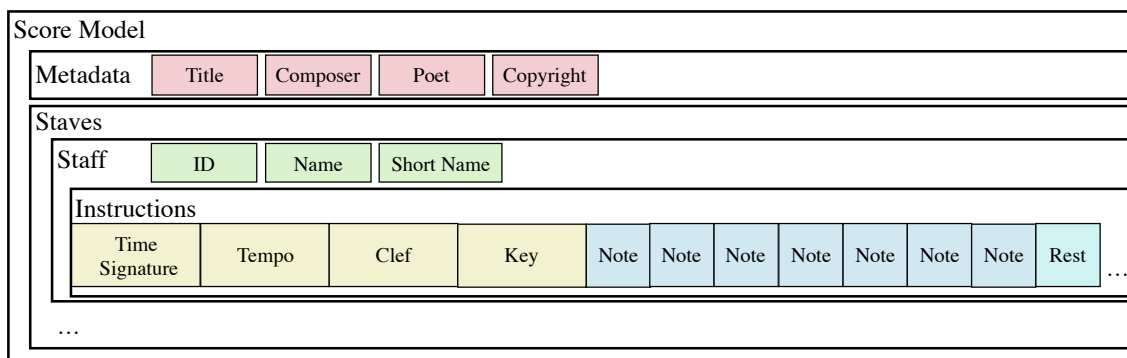


Figure 5.5: Abstract score model structure

5.3.1 LilyPond Compiler

MPS supports the compilation of abstract score models to the LilyPond markup language (Nienhuys and Nieuwenhuizen 2003). The compiler outputs each instruction and note in a LilyPond-specific syntax and writes the results to a LilyPond file. These in turn can be converted using a LilyPond compiler, resulting in PDF files containing the visual score and MIDI files for playback of the piece. The corresponding LilyPond code for the previously introduced example is shown in Listing 5.1. Figure 5.6 shows the PDF file resulting from the LilyPond compilation of the example shown in the last sections.

¹The class diagram of the score model is available on the accompanying CD, see Appendix A

```

1 \version "2.12.0"
2 #(set-default-paper-size "a4")
3
4 \header {
5   title = "Sonata Facile (KV 545), 1st movement"
6   composer = "W. A. Mozart"
7 }
8
9 \score {
10   <<
11
12     \new Staff {
13       \set Staff.midiInstrument = #"acoustic grand"
14       \clef treble
15       \time 4/4
16       \key c \major
17       c''2 e''4 g'',
18       b'4. c''16 d'' c''4 r
19       a''2 g''4 c'',
20       g'' f''8 e''16 f'' e''4 r
21     }
22
23     \new Staff {
24       \set Staff.midiInstrument = #"acoustic grand"
25       \clef treble
26       \time 4/4
27       \key c \major
28       c'8 g' e' g' c' g' e' g',
29       d' g' f' g' c' g' e' g',
30       c' a' f' a' c' g' e' g',
31       b g' d' g' c' g' e' g'
32     }
33
34   >>
35
36   \midi {
37     \context {
38       \Score
39       tempoWholesPerMinute = #(ly:make-moment 120 4)
40     }
41   }
42   \layout {
43     indent = 0\cm
44   }
45 }

```

Listing 5.1: LilyPond representation of W. A. Mozart's *Piano Sonata No. 16 in C major*, K. 545, mm. 1–4

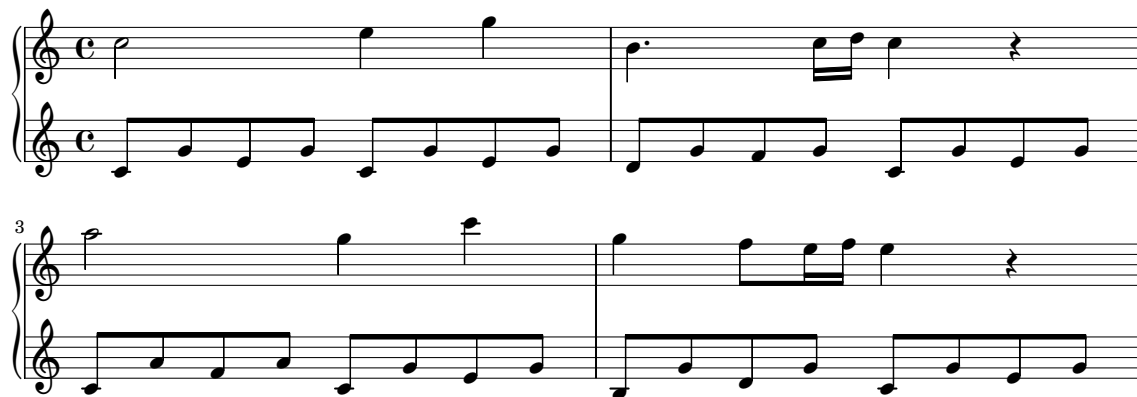


Figure 5.6: Score of W. A. Mozart, *Piano Sonata No. 16 in C major*, K. 545, mm. 1–4

5.4 Transforming Context Layer Models to SuperCollider

SuperCollider is a sound synthesis environment featuring a “domain-specific programming language specialized for sound but with capabilities to rival any general-purpose language” (Wilson et al. 2011, p. xiii). A crucial data structure for sound production in SuperCollider are so called *events*, which are represented by a collection of key/value pairs. Common examples for keys are `duration` (`dur`), `loudness` (`amp`), `instrument` or `sustain`. Events can be generated using higher-level data structures called *event patterns* (Wilson et al. 2011, p. 182). Predefined pattern classes are available to model parallel processes (represented by the class `Ppar`) and sequential processes (implemented by the class `Pseq`).

A compiler was implemented transforming context layer model representations to nested SuperCollider pattern structures. The current implementation produces code which invokes the MIDI engine of SuperCollider to produce MIDI events. The compiled SuperCollider code of the previously introduced example is demonstrated in Listing 5.2.

```

1  MIDIClient.init;
2  m = MIDIOut(0);
3  TempoClock.default.tempo = 120/60/4;
4
5  Pbindef(\p0,
6      \type, \midi,
7      \midiout, m,
8      \chan, 0,
9      \amp, 60/127,
10     \midinote, Pseq([72, 76, 79, 71, 72, 74, 72, -1, 81, 79, 84, 79, 77, 76,
11     77, 76, -1]),
12     \dur, Pseq([1/2, 1/4, 1/4, 3/8, 1/16, 1/16, 1/4, 1/4, 1/2, 1/4, 1/4, 1/4,
13     1/8, 1/16, 1/16, 1/4, 1/4])
14 );
15
16 Pbindef(\p1,
17     \type, \midi,
18     \midiout, m,
19     \chan, 1,
20     \amp, 60/127,
21     \midinote, Pseq([60, 67, 64, 67, 60, 67, 64, 67, 62, 67, 65, 67, 60, 67,
22     64, 67, 60, 69, 65, 69, 60, 67, 64, 67, 59, 67, 62, 67, 60, 67, 64, 67]),
23     \dur, 1/8
24 );
25
26 Pdef(\p0).reset;
27 Pdef(\p1).reset;
28 Pdef(\p0).play;
29 Pdef(\p1).play;

```

Listing 5.2: SuperCollider code of W. A. Mozart, *Piano Sonata No. 16 in C major*, K. 545, mm. 1–4

In the first two lines in Listing 5.2 the `MIDI` client is initialized and the first output

is assigned to the global variable *m*. SuperCollider provides a global `TempoClock`, which schedules events on a time grid of beats and bars. The clock can be configured to have a tempo (e.g. 120 BPM) and a certain number of beats per bar (e.g. 4). To compute the amount of time (in seconds) between two beats, the following formula can be applied (where *b* are the number of beats per minute):

$$\frac{b \text{ beats}}{\text{minute}} = \frac{b \text{ beats}}{60 \text{ s}} = \frac{\frac{60}{b} \text{ s}}{\text{beat}} \quad (5.1)$$

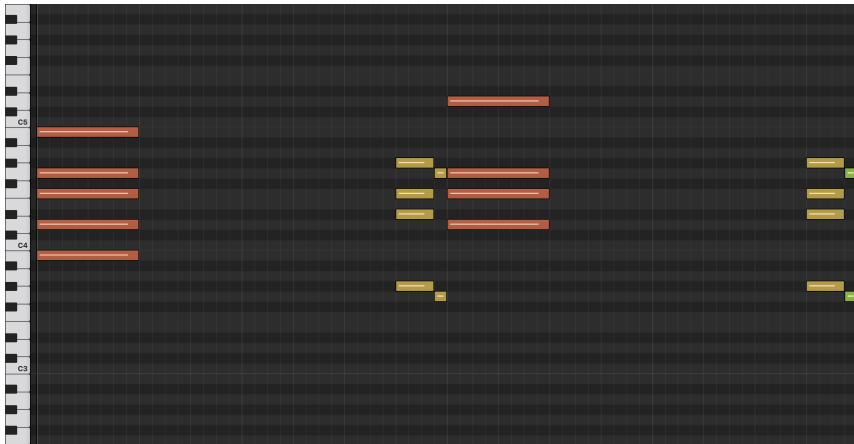
The time interval between two beats is computed in line 3 in Listing 5.2 and the default `TempoClock` is configured accordingly. Below, two pattern definitions were generated, containing musical parameters resulting in sequences of events. Apart from MIDI-specific configuration data, the patterns are populated with a constant loudness (`\amp;`) and sequences of pitches (`\midinote`) and durations (`\dur`).

The utilization of SuperCollider synthesizer definitions (`SynthDefs`) could be a promising future extension. This would enable the composition of electroacoustic music, which makes use of sound synthesis processes with individually defined sound parameters. These parameters could be mapped to musical context layers (as introduced in Chapter 3.3) or to custom context layers (see Chapter 4.5.10).

5.4.1 Immediate Compilation and Execution

By default, the compiler writes the resulting SuperCollider code to a file with the extension `*.scd`. To execute this code, it has to be loaded manually into a SuperCollider interpreter. Additional functionality was developed, with which the code is automatically executable in a running SuperCollider instance. This feature enables the immediate playback of a composition model directly from the MPS application. This was implemented using the data transmission protocol `Open Sound Control (OSC)`. Using this protocol, MPS is able to communicate with a SuperCollider software instance. In particular, the compiled code is transmitted to SuperCollider over a local `User Datagram Protocol (UDP)` socket. The implementation of this functionality is provided by the library `JavaOSC`². The `UDP` protocol does not verify the delivery of network packets and does not guarantee that network packets arrive in the exact order in which they were sent (Bagad and Dhotre 2009, p. 1-8). However, other protocols supporting these criteria, such as `Transmission Control Protocol (TCP)`, can be used in conjunction with `OSC` (Schmeder et al. 2010).

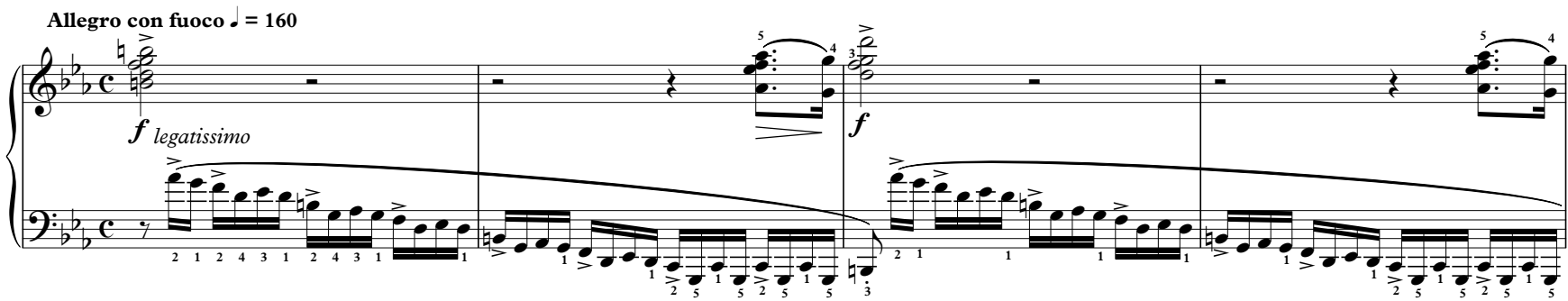
²<https://www.illposed.com/software/javaosc.html>



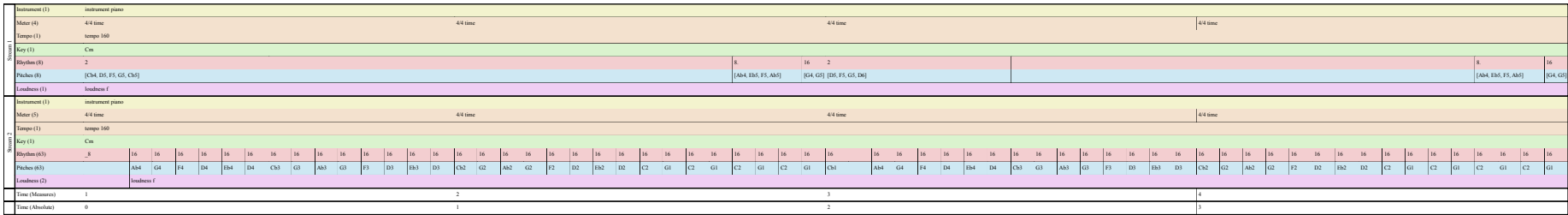
(a) Piano roll representation of the upper part



(b) Piano roll representation of the lower part



(c) Score representation. Edited by Michael Kravchuk and published in the Public Domain.



5.5 Transforming MIDI Files to Context Layer Models

The conversion from `MIDI` files to context layer models is illustrated with the first four measures of Frédéric Chopin’s *Étude Op. 10, No. 12 in C minor (Revolutionary Étude)* as example (see Figure 5.7). Apart from correlating *note on* and *note off* events and interpreting these as notes, the algorithm also evaluates relevant *meta events*, such as time signatures, tempo specifications, key signatures and lyrics (see Chapter 2.1.1). As already mentioned in section 2.1.1, pitches are represented in terms of integer notes in `MIDI`, corresponding to semitone keys on the piano keyboard. However, enharmonic spellings can not be encoded with this representation. For example, it can not be distinguished between the note $F\sharp$ and $G\flat$. `MPS` leverages key signature meta information for making suitable enharmonic decisions. This succeeds in the majority of cases. However, correct decisions can not be assured in the case of harmonic modulations which are not explicitly encoded in the `MIDI` file. As an example, refer to the first chord in the right hand in Figure 5.7c, which can be interpreted as a G^7 chord. In the `MIDI` representation, the exact spelling of the note B is discarded, only the corresponding integer note numbers are stored. `MPS` interprets these notes as $C\flat$, since in the available harmonic context (the key C minor) B is not a feasible spelling option.

In conclusion, the resulting context layer model contains all information derivable from the data and metadata in the input `MIDI` file, which is only entirely “correct” if the relevant context information is properly encoded in the input file. Because some information can not be encoded at all in `MIDI` files, `MPS` additionally supports processing MusicXML files. The corresponding transformations are described in the following section.

5.6 Transforming MusicXML Files to Context Layer Models

As already outlined in section 2.1.7, MusicXML is a music representation format which includes not only exact specifications of musical parameters, but also notation-specific information. In the conversion process from MusicXML to `MPS` context layer models, all represented information have to be combined appropriately. This is not always trivial, because sometimes required information is not available in the current context in the document, but has to be combined with other data located somewhere else in the MusicXML file.

The MusicXML code of J. S. Bach’s *Sinfonia 1*, BWV 787, mm. 1–2, which is used subsequently as an example, is attached to this document in Appendix [B.2](#). The first step in the transformation process is parsing the model into an in-memory object representation. This is accomplished with the library ProxyMusic³. The transformation algorithm subsequently traverses the object structure. After reading meta data (such as title and composer), the part-list is processed (see lines 52–67 in Appendix [B.2](#)).

For each voice contained in the MusicXML file, a separate stream is created in the resulting context layer model. The example file contains only one part (beginning in line 68) containing two measures (at lines 69 and 384, respectively). The notes and rests contained in the measures have individual `<voice>` elements, which makes it possible to encode multiple voices in one part explicitly. In the demonstrated example, the three composed voices of *Sinfonia 1* are encoded as follows:

- The upper voice is encoded as voice 1 (e.g. line 114)
- Voice number 2 was assigned to the middle voice (e.g. line 447)
- The lower voice has number 5 (e.g. line 312)

This results in three individual streams in the context layer model, as shown in Figure [5.8](#). If a voice contains simultaneously sounding notes with different durations, these are automatically extracted to further individual streams. For instance, this is required for some drum and percussion instrument parts, in which multiple instruments are frequently notated in one part.

The transformation algorithm also evaluates time signatures (e.g. line 87), key signatures (e.g. line 84), harmonic progressions and lyrics in MusicXML files. The unfolding of repeats was also implemented in the MusicXML transformation process, and can be activated or deactivated using a configuration flag.

5.7 Deriving and Compressing Context Tree Composition Models

Deriving context tree representations from time-based context layer model representations is implemented in the form of a compression algorithm, which aims to create a compact tree representation of the input model. In other words, the number of tree nodes required to represent a composition is minimized. By definition, when expanding the resulting context tree model representation (the expansion process is

³<https://github.com/Audiveris/proxymusic>

Stream 2	Instrument (1)	instrument piano																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Meter (2)	4/4 time										4/4 time																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Key (1)	C																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Harmony (1)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
	Harmonic Rhythm (1)	_2!																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Rhythm (19)	_16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	2~	4	4																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
	Pitches (19)		G4	A4	B4	C5	D5	E5	F5	G5	F5	G5	A5	F5	A5	G5	F5	E5	E5	F#5																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
Loudness (2)		loudness mf																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
Stream 3	Instrument (1)	instrument piano																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Meter (2)	4/4 time										4/4 time																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Key (1)	C																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Harmony (1)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
	Harmonic Rhythm (1)	_2!																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Rhythm (14)	4		_8	8	8	8	8	8	4		_8	8	8	8	8	8																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
	Pitches (14)	C3			C4	B3	G3	A3	B3	C4			B3	A3	G3	A3	D3																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
Loudness (1)	loudness mf																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
Stream 4	Instrument (1)	instrument piano																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Meter (2)											4/4 time																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Key (1)	C																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
	Harmony (2)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
	Harmonic Rhythm (2)	_1										_1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
	Rhythm (19)											_16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16

Figure 5.8: Context layer model of J. S. Bach, *Sinfonia 1*, BWV 787, mm. 1–2, resulting from the MusicXML code in Appendix B.2

explained in section 5.2), the result must be equivalent to the original input. Thus, the proposed algorithm applies *lossless* compression to musical context tree models (Hankerson et al. 2003, p. 119).

5.7.1 Related Work

Some aspects of the proposed algorithm are reminiscent of Huffman coding, a lossless compression method using tree structures for constructing codes for arbitrary input sequences (Salomon 2007, pp. 74ff.). The aim of Huffman encoding usually is to encode symbols occurring with high relative frequencies (i.e. *probabilities*) with short binary sequences, whereas low probability input symbols are encoded with longer codes. Figure 5.9 illustrates the construction of a code for the input sequence *abcbcbadae*. The symbol probabilities are: $P(a) = 0.4$, $P(b) = 0.2$, $P(c) = 0.2$, $P(d) = 0.1$ and $P(e) = 0.1$.

However, there are a number of differences between Huffman coding and the proposed compression method:

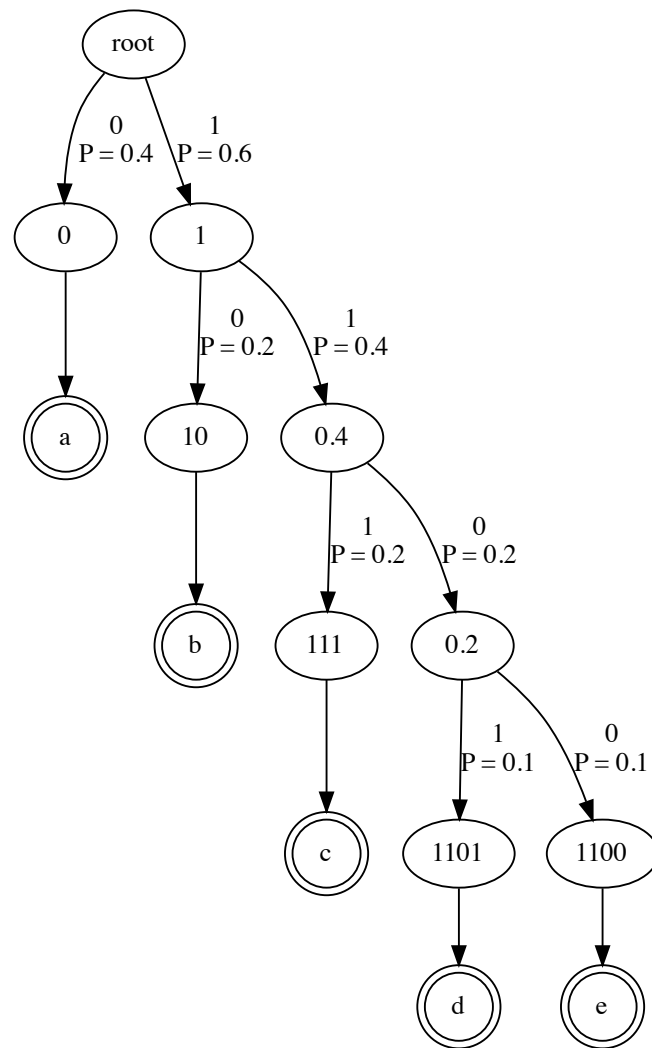


Figure 5.9: Huffman coding tree illustrating the construction of a possible code for the input sequence **abcabcadae**. The resulting binary encoding is: $a \hat{=} 0$, $b \hat{=} 10$, $c \hat{=} 111$, $d \hat{=} 1101$, $e \hat{=} 1100$

- The input data is not a sequence of symbols, but a tree model
- The output is not a binary sequence, but an optimized tree
- Huffman coding typically utilizes binary or ternary trees (Salomon [2007](#), pp. 82ff.), context tree models can have arbitrary tree shapes
- Context tree models support distinct mechanisms for internal redundancy optimization such as inheritance (see Chapter [4.4.2](#)), polymorphism (see Chapter [4.4.3](#)) auto expansion (see section [4.4.4](#)) and fragments (Chapter [4.4.5](#)).
- In context tree models, the child node orders can affect the semantics of the resulting music, which has to be taken into account when compressing the tree model

More details on Huffman coding similarities and differences can be found in the following section.

5.7.2 Compression Algorithm

The model deriving and compression process is divided into five phases:

1. Segmentation of the input context layer model
2. Context tree conversion of the segmented parts
3. Optimizations using auto expansion
4. Optimizations using inheritance
5. Optimizations using control structures and fragment extractions

To illustrate the model compressing process, the first eight measures of *Piano Sonata No. 21 in C major, Op. 53* known as *Waldstein* by Ludwig van Beethoven are used as an example. The score is shown in Figure [5.10](#).

Segmentation Phase

In the segmentation phase, each stream (or voice) in the given context stream model is divided into n sections. The default behaviour is to segment the stream measure-wise. Using a configuration parameter, the segmentation granularity can be adjusted to multiples of measure lengths (e.g. $\frac{1}{2}$, $\frac{1}{4}$ or 2 times the measure length). A possible future enhancement for the algorithm is to determine the optimal segmentation size automatically.



Figure 5.10: Ludwig van Beethoven, *Piano Sonata No. 21 in C major, Op. 53* (“Waldstein”), mm. 1–8

In a second segmentation process, repetitive sequences of time signatures, instruments, keys, rhythms, pitches and lyric syllables are identified. If applicable, rhythmic sequences and pitch sequences are divided in such a way that as many sequences as possible contain identical pitches and rhythms. In a later stage, these redundant sequences are eliminated by using a custom type of **Run-Length Encoding (RLE)** (Salomon 2007, pp. 22ff.).

Tree Conversion

Each segmented part is translated to a corresponding tree representation. A straightforward approach for this translation is creating corresponding tree nodes for each context layer. The resulting redundant subtrees are inserted in the created context tree model below a parallelization control structure, which represents individual voices in the composition. For illustration, a redundant context tree model is shown in Figure 5.11.

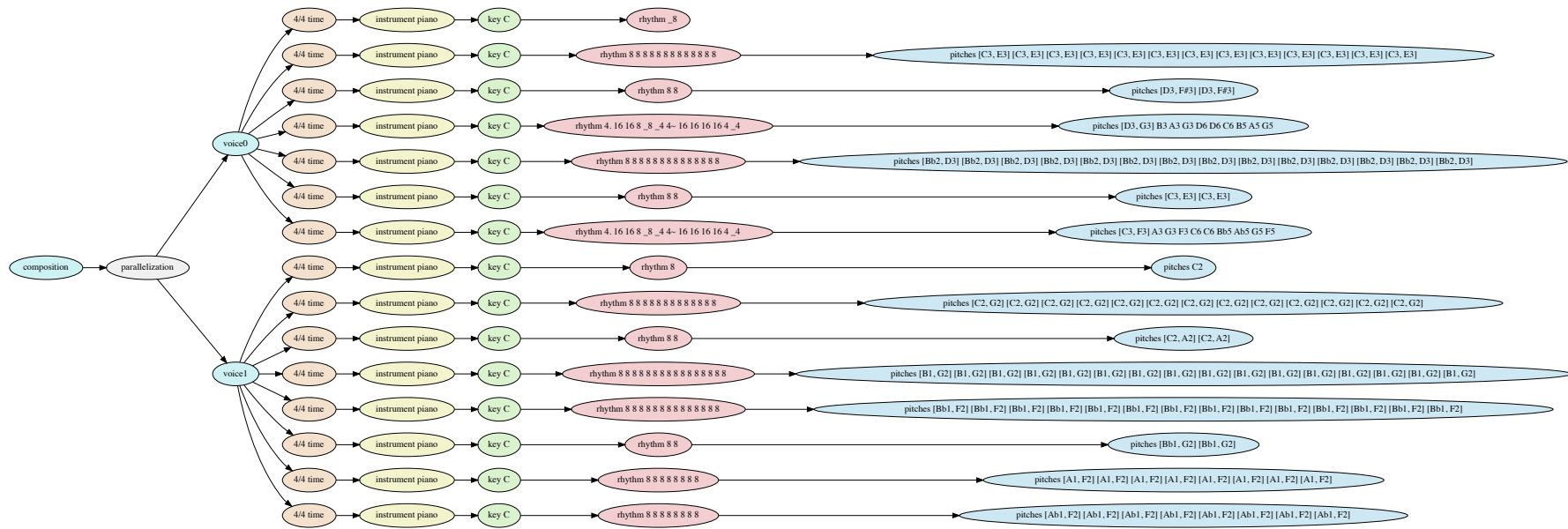


Figure 5.11: Redundant context tree model of Ludwig van Beethoven, *Piano Sonata No. 21 in C major, Op. 53* (“Waldstein”), mm. 1–8

Auto Expansion Optimization

The first optimization phase leverages the functionality of *auto expansion*, which was explained in Chapter 4.4.4. In the example, repetitive pitch sequences are reduced to sequences with only one item. After the first phase, the model looks as shown in Figure 5.13. In comparison to the state in Figure 5.11, redundant information was removed at nodes *l*, *m*, *o*, *r*, *s*, *t*, *u*, *v*, *w* and *x*.

Inheritance Optimization

In the next step, the hierarchical model structure is utilized for inheritance-based optimizations (see Chapter 4.4.2). In particular, the aggregated node constellations at the leaf nodes of the tree are analyzed and common nodes are identified. For this purpose, the tree nodes are represented in a matrix, which is illustrated in Figure 5.12.

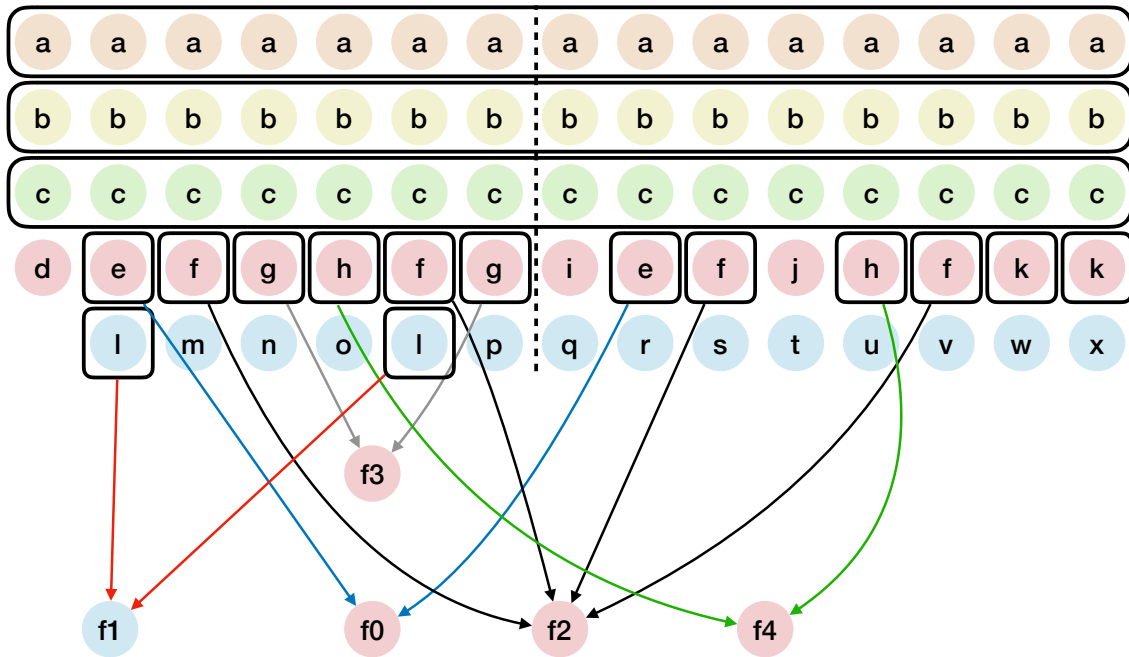


Figure 5.12: Node matrix used for computing the hierarchical tree arrangement of the compressed model. Refer to Figure 5.13 for a visualization of the nodes *a* to *x*. The nodes *a*, *b* and *c* are combined to three single nodes and arranged as common parent nodes in the tree, as shown in Figure 5.14. This graphic also illustrates the extraction of fragments, the result of which can be seen in Figure 5.15.

For each node in the tree, successive equivalent nodes are identified and grouped. The longer the group length, the more often a particular node is reused. Therefore, the group lengths can be interpreted as indications for the hierarchy levels in the compressed tree. The higher the number, the higher is the hierarchy level in the resulting tree, and the higher is the degree of inheritance optimization. The analysis

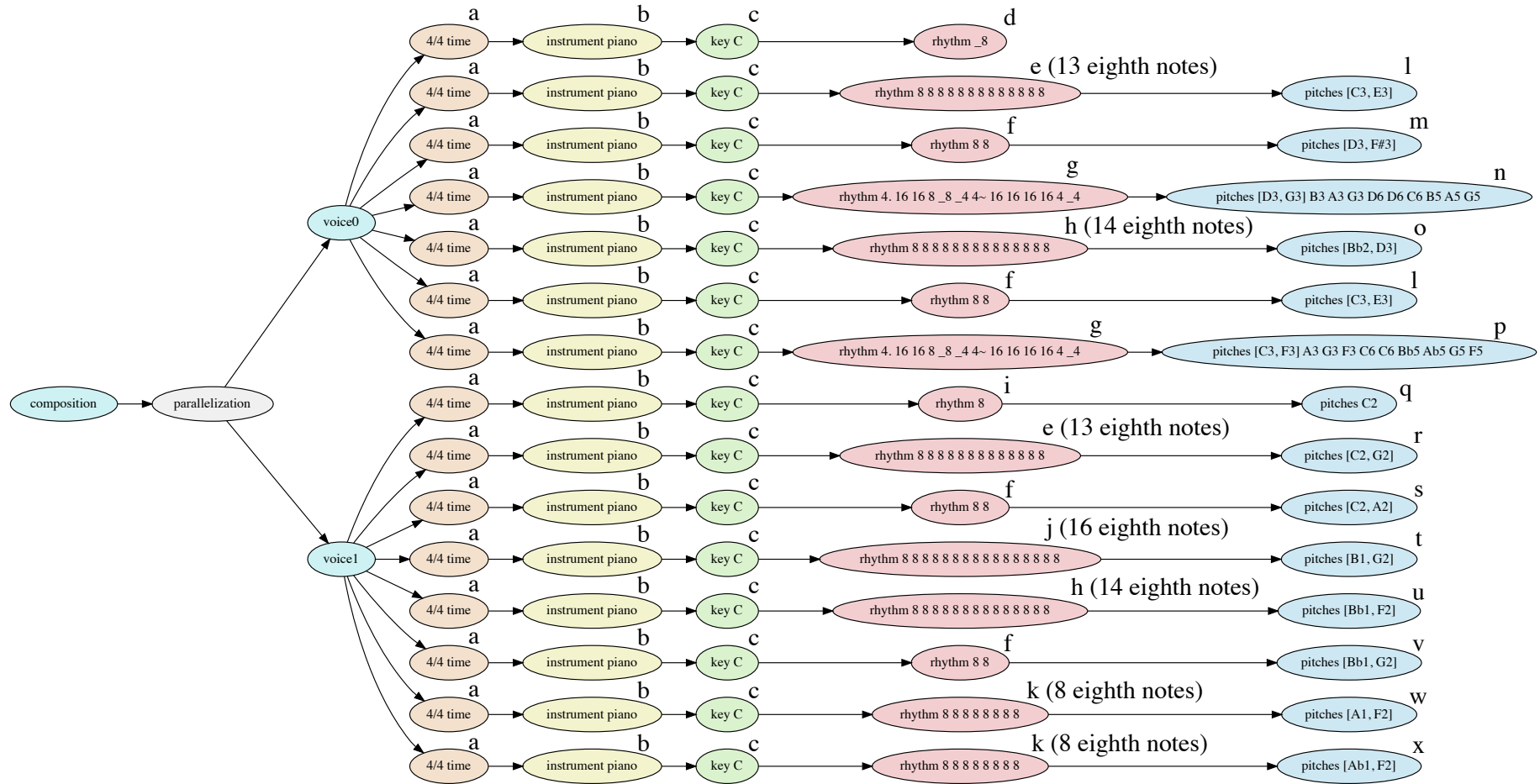


Figure 5.13: Context tree model of Ludwig van Beethoven, *Piano Sonata No. 21 in C major, Op. 53* (“Waldstein”), mm. 1–8, after applying *auto expansion* optimizations. This results to the removal of redundant information at nodes *l*, *m*, *o*, *r*, *s*, *t*, *u*, *v*, *w* and *x*.

of distinct node occurrence counts and the positioning of often occurring nodes near the top of trees is an analogy to Huffman coding. However, an essential difference is that in this step only consecutive nodes are taken into account. This is required to retain the temporal semantics of the composition model. After the optimization, the model is structured as shown in Figure 5.14.

Fragment Extraction Optimization

In the last optimization phase, redundant subtrees are identified and extracted (see Chapter 4.4.5). This is repeated until no more optimizations are possible. This step eliminates redundant information of non-consecutive node structures, which can not be achieved using the previously described inheritance optimization. For instance, the two rhythm nodes labeled k in Figure 5.13 can be combined to one node with the two child nodes w and x . In contrast, consider the nodes labeled f in Figure 5.13. These can not be combined to one node while preserving the temporal semantics of the composition model, at least not in the common hierarchy level. Therefore, the strategy of extracting fragments is applied, as introduced in Chapter 4.4.5. After performing this last step, the composition model is redundancy-optimized as shown in Figure 5.15.

5.7.3 Future Work

The current implementation performs model data compressions by identifying equivalent nodes. A future enhancement for the algorithm to be considered is the detection of similar nodes, which could be eliminated by describing suitable modification procedures, as explained in Chapter 4.6. By this means, composition models could be reduced to essential musical information and the description of modification processes applied throughout the composition, which conforms to the way most human composers shape musical works.

Considering the presented example, the algorithm then should be able to detect the pitch relationships of the first and second half of the excerpt, which is in essence expressible as a transposition by two descending semitones. In addition, the highly similar rhythmic structure of the second excerpt half could be represented in terms of simple rhythmic insertions in the rhythms of the first half.

To summarize, the proposed algorithm is not only capable of transforming time-based music representations to tree-based models, it also identifies redundant information and re-formulates models in such a way that information specified more than once is extracted and reused in order to ensure a concise music representation.

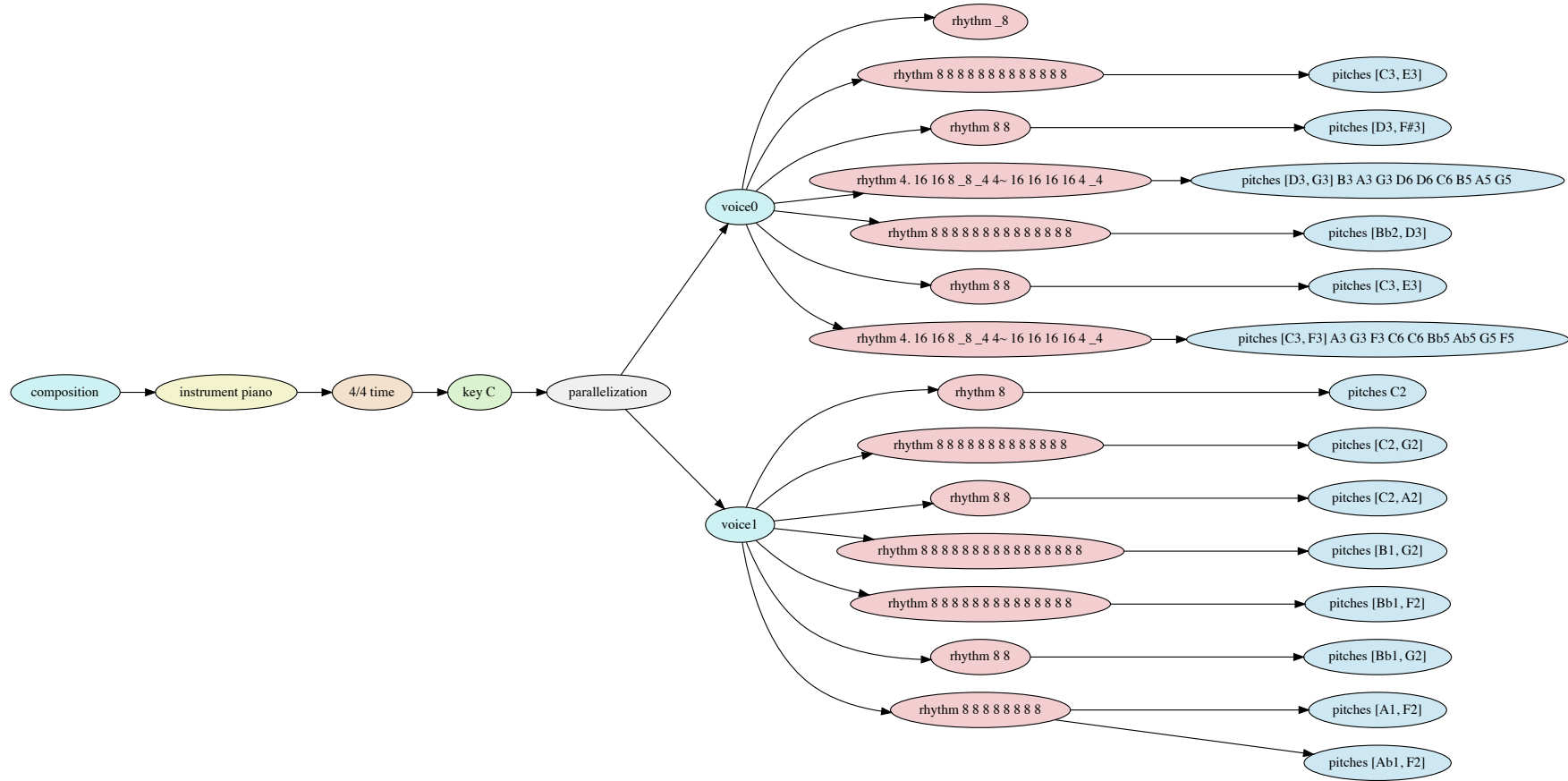


Figure 5.14: Context tree model of Ludwig van Beethoven, *Piano Sonata No. 21 in C major, Op. 53* (“Waldstein”), mm. 1–8, after optimizing the tree structure utilizing inheritance

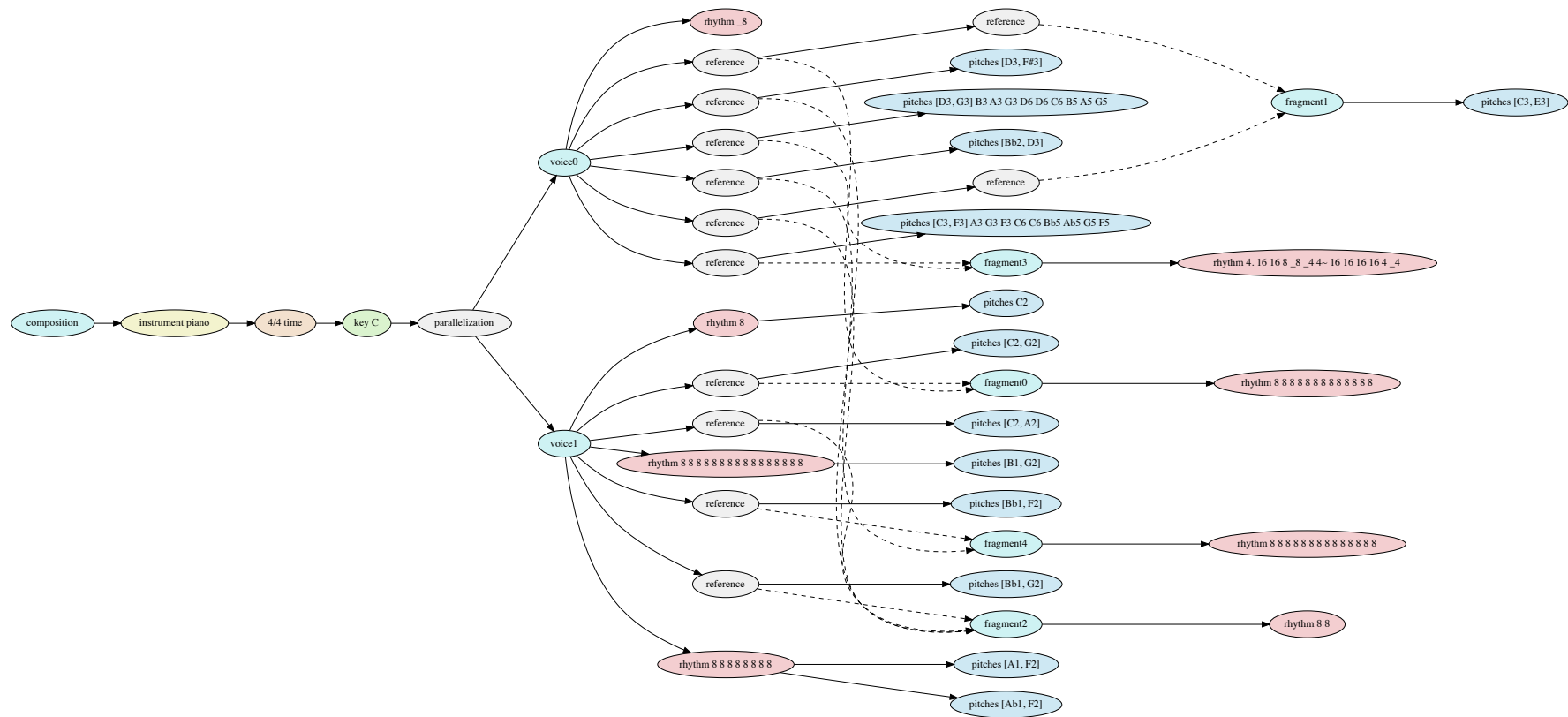


Figure 5.15: Context tree model of Ludwig van Beethoven, *Piano Sonata No. 21 in C major, Op. 53* (“Waldstein”), mm. 1–8, after extracting fragments

5.8 Graphical User Interface

All transformations introduced in this chapter can be conveniently invoked using graphical user interfaces in **MPS**. Figure 5.16 shows the **MPS** toolbar, which is located at the top of the user interface of the application (see Figure 1.2). The toolbar provides buttons with which the user can invoke transformations with one simple click.

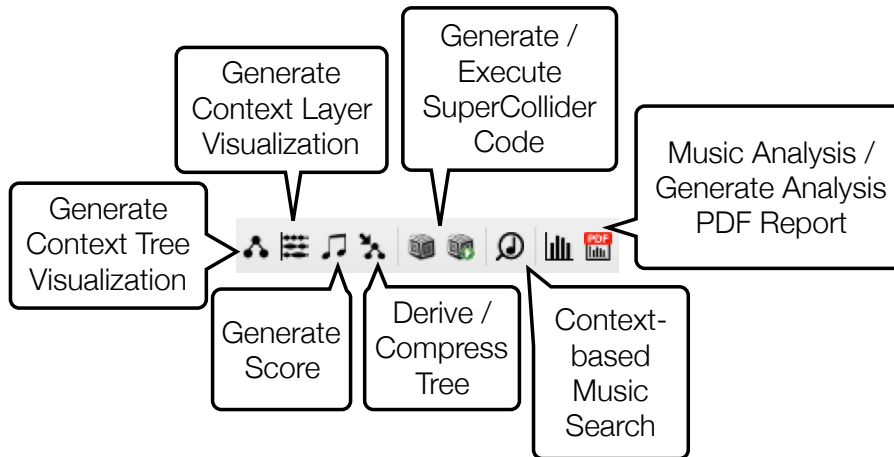


Figure 5.16: Toolbar of the graphical Music Processing Suite user interface. Available actions are: generate context tree visualizations (see Chapter 4), generate context layer model visualizations (see Chapter 3), generate scores (see Chapter 5.3), derive / compress tree models (see Chapter 5.7), context-based music search (will be introduced in Chapter 6) and music analysis (which will be demonstrated in Chapter 7).

The results of the corresponding transformations are stored in files with a suitable file name suffix. For example, refer to Figure 1.2, in which the current project contains a composition language file named `BeethovenSymphony5.mc1`. Next to this file, other files with the same base name are stored. Score-related files, specifically the corresponding LilyPond, **MIDI** and **PDF** files, start with the file name prefix `BeethovenSymphony5_Score.*`.

5.9 Summary

Transformation algorithms which enable the conversion between selected symbolic music representation formats have been introduced. **MPS** gains flexibility by being capable of representing music both in time-dependent models and in a concise tree representations. The transformation infrastructure is capable of converting in both directions: from time-based representations (including the symbolic music standard formats **MIDI** and MusicXML) via context layer models to context tree models, and vice versa. The capability of importing standardized symbolic music formats

is an important feature of [MPS](#) as musicians and researchers can build upon a vast repertoire of existing material. The introduced representation forms have certain advantages for specific computational music processing applications, as will be shown in the following chapters. Moreover, the conversions between context layer models and context tree models are an integral part of the automated composition algorithm, which will be introduced in [Chapter 8](#).

Chapter 6

Context-based Corpus Search

Elwood: What kind of music do you usually have here?

Claire: Oh, we got both kinds. We got country *and* western.

— from the movie *The Blues Brothers*

In this chapter, another application of the context-based composition model is introduced: searching for musical fragments in a corpus of pieces. Advanced techniques for retrieving musical information in musical corpora are possible due to the model design: since all available context information is accessible at any point of time in the composition, search criteria can be specified that go beyond simple musical sequences. The main focus in this chapter is on the advantages of the underlying model, and not on the search algorithm itself, which is implemented in a straightforward fashion.

6.1 Motivation

The search for a musical phrase or fragment in a musical corpus can be performed in many different ways. In most cases, the available material is scanned for a specific musical sequence of notes and rests. As already discussed in the introduction of Chapter 3, this methodology is not sufficient in all cases. The already presented example is repeated in Figure 6.1 for convenience.



Figure 6.1: Ludwig van Beethoven, *Symphony No. 5 in C Minor, Op. 67*, Mv. I, motif

Depending on the application, it might be desirable to specify multiple search criteria or to abstract the search query. Aspects to consider are, for instance:

1. Do we want to limit the search results to specific instruments?
2. Should the initial rest be taken into account for matching the rhythm?
3. Which time signatures are relevant? Only $\frac{2}{4}$ time or other metric contexts as well?
4. Is the temporal position of the rhythm in the respective measures relevant?
5. Do we search for the exact pitch sequence $G-G-G-E\flat$ or for three equal pitches in a row followed by a descending major third?
6. Should the pitch sequence be abstracted to scale degrees in order to find the degree combination (namely $5-5-5-3$, or zero-based $4-4-4-2$) on other scales?
7. How important is the key context? Should the motif be located only in C minor contexts or other keys as well?
8. Should the harmonic context consider the key only or also local harmonic contexts (local keys, contextual harmonies)?

The previously mentioned variants are based on a selection of common musical parameters or context layers. Since the model accommodates more context layers as described in section [3.3](#), a large number of further abstractions and contextual matching scenarios are possible.

6.2 Formulating Musical Search Queries

The formulation of musical search queries is easily possible with the domain-specific composition language introduced in Chapter [4](#). Although the original purpose of the language is to describe musical compositions, a given composition model is also interpretable as a musical search query. For instance, if we only look for the rhythm of Beethoven’s motif in a $\frac{2}{4}$ metric context, we could simply express this search query as:

```

1 composition
2 {
3     time 2/4, rhythm _8 8 8 8 2
4 }
    
```

Listing 6.1: Syntactical representation of Beethoven’s 5th Symphony motif rhythm and metric context, which can be interpreted as search query.

Queries can be formulated using arbitrary musical context combinations introduced in chapters 3 and 4, including instruments, time signatures, rhythms, absolute pitches, degree-based pitches, keys, harmonies and lyrics.

6.3 Search Methodology

The search for the queried musical fragment is performed at the context layer model level. Particularly, the search query model is transformed into a context layer model containing only the queried contexts, meaning that no default contexts will be used (see Chapter 5.2). The query layer model will subsequently be matched against layer model representations of all compositions in the corpus. An overview of this concept is illustrated in Figure 6.2.

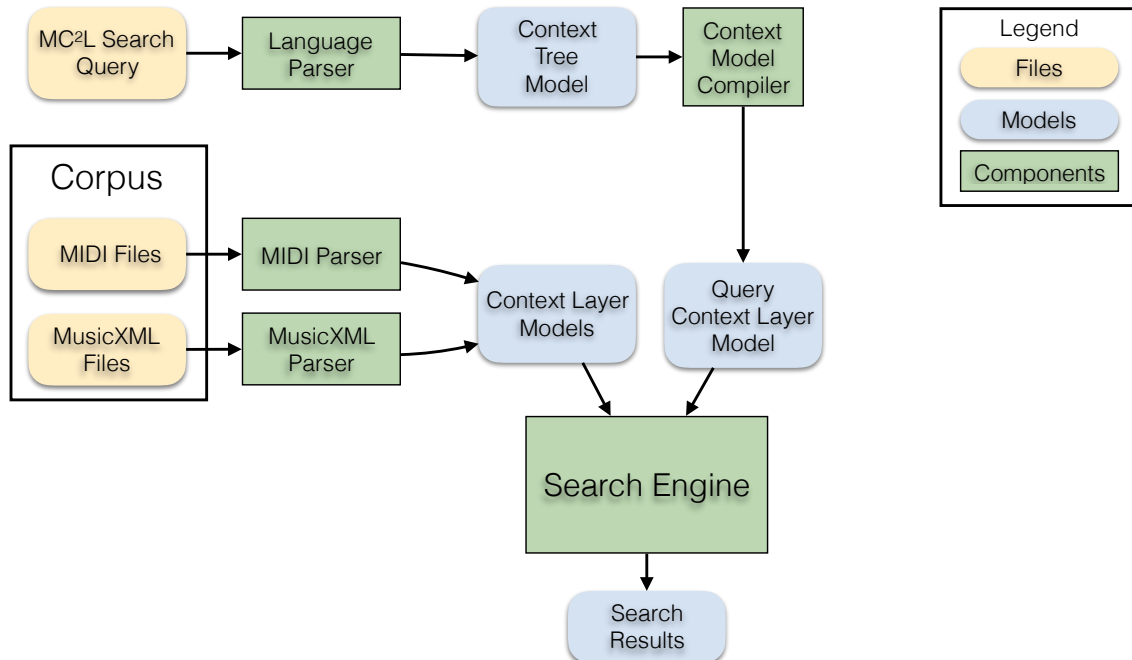


Figure 6.2: Overview of the context-based search infrastructure

6.4 Search Query Context Layer Models

The first step performed by the algorithm is producing a context layer model representation of the search query. This step is handled by the context tree model compiler (see section 5.2) with a special configuration. It ensures that no default context layers will be generated, resulting in a model containing only the types of contexts specified in the search query. To illustrate, various search queries and resulting context layer model representations are shown in Table 6.1.

6.5 Search Algorithm

The algorithm compares the query context layer model patterns resulting from the transformation described in the previous section with context layer model representations of the corpus. Therefore, each input file is transformed to a context layer model using suitable parsers and converters (for details, refer to sections 5.5 and 5.6). Each corpus layer model is segmented into stream events using stream sequencers (explained in section 3.6). The algorithm iterates through all stream events, comparing if a stream event matches the first query stream event in all aspects (i.e. all context layers). If this is the case, the query stream events are iterated in parallel to the corpus layer model and are compared layer by layer. If the query stream iteration is finished without any discrepancy being detected, a match is registered. The pseudocode of the algorithm is shown in Listing 6.2.

```
1  searchResults = empty list
2  queryStreamModel = construct stream model for search query
3  queryStreamEvents = compute segmentation of queryStreamModel
4  inputFiles = scan corpus for MIDI, MusicXML and MC2L files
5  foreach input file
6  {
7      inputStreamModel = convert file to stream model
8      streamEvents = compute segmentation of inputStreamModel
9      foreach stream event in streamEvents
10     {
11         eventIndex = index of current stream event
12         foreach query stream event
13         {
14             if query stream event does not match event with eventIndex:
15                 cancel comparison and continue at next stream event (outer loop)
16             else:
17                 increment eventIndex
18         }
19         add result to searchResults
20     }
21 }
22 return searchResults
```

Listing 6.2: Context-based search algorithm

Table 6.1: Search queries and corresponding query stream model patterns

Description	Query Syntax	Query Context Layer Model																		
Search for the rhythm $\text{♩} \text{♩} \text{♩} \text{♩}$ only.	rhythm _8 8 8 8 2	<table><tr><td>Rhythm (5)</td><td>_8</td><td>8</td><td>8</td><td>8</td><td>2</td></tr></table>	Rhythm (5)	_8	8	8	8	2												
Rhythm (5)	_8	8	8	8	2															
Search for the explicit pitch sequence $G G G Eb$ only.	pitches G G G Eb	<table><tr><td>Pitches (4)</td><td>G</td><td>G</td><td>G</td><td>Eb</td></tr></table>	Pitches (4)	G	G	G	Eb													
Pitches (4)	G	G	G	Eb																
Search for the rhythm $\text{♩} \text{♩} \text{♩} \text{♩}$ in combination with the explicit pitch sequence $G G G Eb$.	rhythm _8 8 8 8 2, pitches G G G Eb	<table><tr><td>Rhythm (5)</td><td>_8</td><td>8</td><td>8</td><td>8</td><td>2</td></tr><tr><td>Pitches (5)</td><td></td><td>G4</td><td>G4</td><td>G4</td><td>Eb4</td></tr></table>	Rhythm (5)	_8	8	8	8	2	Pitches (5)		G4	G4	G4	Eb4						
Rhythm (5)	_8	8	8	8	2															
Pitches (5)		G4	G4	G4	Eb4															
Search for the zero-based degrees $4 4 4 2$ in an arbitrary scale context.	pitches 4 4 4 2	<table><tr><td>Degrees (4)</td><td>4</td><td>4</td><td>4</td><td>2</td></tr></table>	Degrees (4)	4	4	4	2													
Degrees (4)	4	4	4	2																
Search for the zero-based degrees $4 4 4 2$ in a C minor key context.	key Cm, pitches 4 4 4 2	<table><tr><td>Key (1)</td><td colspan="5">Cm</td></tr><tr><td>Degrees (4)</td><td>4</td><td>4</td><td>4</td><td>2</td></tr></table>	Key (1)	Cm					Degrees (4)	4	4	4	2							
Key (1)	Cm																			
Degrees (4)	4	4	4	2																
Search for the zero-based degrees $4 4 4 2$ in a C minor key context with the rhythm $\text{♩} \text{♩} \text{♩} \text{♩}$.	key Cm, rhythm _8 8 8 8 2, pitches 4 4 4 2	<table><tr><td>Key (1)</td><td colspan="5">Cm</td></tr><tr><td>Rhythm (5)</td><td>_8</td><td>8</td><td>8</td><td>8</td><td>2</td></tr><tr><td>Degrees (5)</td><td></td><td>4</td><td>4</td><td>4</td><td>2</td></tr></table>	Key (1)	Cm					Rhythm (5)	_8	8	8	8	2	Degrees (5)		4	4	4	2
Key (1)	Cm																			
Rhythm (5)	_8	8	8	8	2															
Degrees (5)		4	4	4	2															

6.6 Search Result Presentation

Match results are displayed in a separate view in the graphical user interface of **MPS**. An example is presented in Figure 6.3.

File	Stream	Measure	Beat in Measure	Absolute Time
▶ /Analysis_LargeCorpus/LargeCorpus/Beatles/Do You Want To Know A Secret (Leadsheet).mxl	---0	36, 38		
▼ /Analysis_LargeCorpus/LargeCorpus/Beatles/I Want To Hold Your Hand (Leadsheet).mxl	---0	32, 33		
▶ I Want To Hold Your Hand (Leadsheet).mxl	---0	32	1 / 2	63 / 2
▶ I Want To Hold Your Hand (Leadsheet).mxl	---0	33	1 / 2	65 / 2
▼ /Analysis_LargeCorpus/LargeCorpus/Beatles/You Won't See Me (Leadsheet).mxl	---0	1, 5, 9, 13		
▶ You Won't See Me (Leadsheet).mxl	---0	1	1 / 2	1 / 2
▶ You Won't See Me (Leadsheet).mxl	---0	5	1 / 2	9 / 2
▶ You Won't See Me (Leadsheet).mxl	---0	9	1 / 2	17 / 2
▶ You Won't See Me (Leadsheet).mxl	---0	13	1 / 2	25 / 2
▶ /Analysis_LargeCorpus/LargeCorpus/Beatles/Hello, Goodbye (Leadsheet).mxl	---0	10, 13		
▶ /Analysis_LargeCorpus/LargeCorpus/Beatles/Here There and Everywhere (Leadsheet).mxl	---0	24, 25		
▶ /Analysis_LargeCorpus/LargeCorpus/Beatles/A Hard Day's Night (Score).mxl	---1	2		

Figure 6.3: Search result presentation in the graphical user interface. The screenshot shows results for the rhythm of Beethoven’s 5th Symphony motif in songs by the Beatles. Results are grouped by source file. The following items are displayed for each search result: resource path, zero-based stream number, measures, beats and absolute time.

The results are hierarchically presented in a table and grouped by source file. Each search result displays the workspace-relative source file path, the zero-based stream number in which the result was found, the measure number, the beat (i.e. the zero-based start time in the corresponding measure) and the absolute time, expressed in terms of a fraction (see section 3.4 for details).

6.7 Results

The following search results were produced by evaluating selected search queries presented in section 6.4 against a corpus of 1122 MusicXML files containing pieces by the following composers or musical styles, respectively:

- Johann Sebastian Bach (545 pieces)
- The Beatles (36 pieces)
- Ludwig van Beethoven (163 pieces)
- Johannes Brahms (40 pieces)
- Frédéric Chopin (101 pieces)
- Claude Debussy (20 pieces)
- Jazz Standards (24 pieces)

- Scott Joplin (18 pieces)
- Wolfgang Amadeus Mozart (66 pieces)
- Franz Schubert (51 pieces)
- Antonio Vivaldi (58 pieces)

The corpus was compiled from open internet sources and platforms providing MusicXML files. The platform on which most high-quality scores were found was musicxml.org. Basic quality and plausibility checks for the selected files were performed by reviewing the duration of the pieces to sort out fragments and work in progress files, assessing the overall notation quality and by listening to selected sections of the pieces to assure a certain musical coherence and to avoid pitch discrepancies for transposed instruments. Despite these efforts, it is impossible to guarantee that each and every note in the corpus is correctly notated. A copy of the corpus is provided on the accompanying CD, see Appendix [A](#).

Rhythm Search Results


Querying the corpus for the rhythm  from Beethoven's *Symphony No. 5 in C minor, Op. 67* motif produces matches in 83 pieces. A selection of matches is shown in Table [6.2](#).

Table 6.2: Selected search results for the rhythm of Beethoven's 5th *Symphony* motif

Composer	Work Number	Title	Measure(s)
Bach	BWV 537	<i>Fantasia and Fugue in C minor</i>	92, 95, 137, 138
Bach	BWV 540	Tocatta and Fugue in F major	598
Bach	BWV 848	The Well-Tempered Clavier Book I: Fugue No. 3 in C♯ major	55
Bach	BWV 858	The Well-Tempered Clavier Book I: Fugue No. 13 in F♯ major	23

Continued on next page

Table 6.2 – *Continued from previous page*

Composer	Work Number	Title	Measure(s)
Bach	BWV 880	The Well-Tempered Clavier Book II: Prelude No. 11 in F major	2, 4, 5, 7, 9, 12, 16, 18, 20, 21, 23, 25, 27, 29, 30, 34, 37, 46, 58, 60, 61, 68
Bach	BWV 892	The Well-Tempered Clavier Book II: Prelude No. 23 in B major	12, 23
Beatles		A Hard Day's Night	2
Beatles		Here, There and Everywhere	25
Beatles		You Won't See Me	13
Joplin		Augustan Club Waltz	100
Beethoven	Op. 13	Piano Sonata No. 8 ("Pathétique"), Movement III	79
Beethoven	Op. 14 No. 1	Piano Sonata No. 9 in E major, Movement I	60, 83, 85, 87, 155
Beethoven	Op. 21	Symphony No. 1, Movement IV	189
Beethoven	Op. 30 No. 2	Violin Sonata No. 7 in C minor	17, 147
Beethoven	Op. 55	Symphony No. 3 ("Eroica"), Movement I	556
Beethoven	Op. 55	Symphony No. 3 ("Eroica"), Movement IV	332
Beethoven	Op. 60	Symphony No. 4, Movement II	34, 36, 89, 91
Beethoven	Op. 60	Symphony No. 4, Movement IV	214
Schubert	D. 547	<i>An die Musik</i>	20, 21, 40, 41
Schubert	D. 530	<i>An eine Quelle</i>	21, 22, 23

Continued on next page

Table 6.2 – *Continued from previous page*

Composer	Work Number	Title	Measure(s)
Schubert	D. 930	<i>Der Hochzeitsbraten</i>	206, 367, 432
Debussy	CD 82	<i>Suite Bergamasque</i> , Movement IV: “Passepied”	110

Absolute Pitch Search Results

The pitch sequence $G\ G\ G\ Eb$ is found in 20 pieces in the corpus. These are listed in Table 6.3. The first movement of the 5th Symphony itself is excluded.

Table 6.3: Search results for the absolute pitch sequence of Beethoven’s 5th *Symphony* motif


Composer	Work Number	Title	Measure(s)
Bach	BWV 14 No. 5	Chorale from Cantata <i>Wär Gott nicht mit uns diese Zeit</i>	1, 5
Bach	BWV 93 No. 7	Chorale from Cantata <i>Wer nur den lieben Gott läßt walten</i>	5
Bach	BWV 345	Chorale <i>Ich bin ja, Herr, in deiner Macht</i>	13
Bach	BWV 403	Chorale <i>O Mensch, schau Jesum Christum an</i>	1
Beethoven	Op. 10 No. 2	Sonata No. 6, Movement III	37
Beethoven	Op. 55	Symphony No. 3 (“Eroica”), Movement IV	80
Brahms	Op. 36	<i>String Sextet No. 2 in G major</i> , Movement I	4, 17, 53
Mozart	KV 427 No. 1	<i>Kyrie from Great Mass in C minor</i>	1, 86

Continued on next page

Table 6.3 – *Continued from previous page*

Composer	Work Number	Title	Measure(s)
Mozart	KV 427 No. 18	<i>Agnus Dei</i> from <i>Great Mass in C minor</i>	1, 86
Mozart	KV 458	<i>String Quartet No. 17 in B-flat major</i> (“ <i>The Hunt</i> ”)	242, 250, 253
Mozart	KV 563	<i>Divertimento in E-flat major</i> , Movement I	174
Schubert	D. 955	<i>Glaube, Hoffnung und Liebe</i>	8, 76
Vivaldi	RV 297	<i>Concerto No. 4 in F minor</i> (“ <i>L’inverno</i> ”), Movement I	41
Vivaldi	RV 578	<i>Concerto for 2 Violins and Cello in G minor</i> , Movement I	7
Vivaldi	RV 531	<i>Concerto for 2 Cellos in G minor</i> , Movement I	61
Vivaldi	RV 728	Aria “ <i>Sol da te, mio dolce amore</i> ” from opera <i>Orlando</i>	22, 24, 37, 38, 40

Combined Rhythm and Pitch Search Results

A search for the rhythm  combined with the absolute pitch sequence $G\ G\ G\ E_b$ yields no results (except for the first movement of the 5th Symphony). The same holds true if the initial rest is removed from the rhythmic part of the search query. This indicates that no literal copy of the motif is contained in the corpus.

Combined Rhythm and Key Search Results


Searching for the rhythm  in the key context $C\ minor$ surprisingly only produces search results in two pieces, which are shown in Table [6.4](#).

Table 6.4: Search results for the rhythm of Beethoven’s 5th *Symphony* motif in the key C minor

Composer	Work Number	Title	Measure(s)
Brahms	Op. 51 No. 1	<i>String Quartet No. 1 in C minor</i> , Movement I	179, 180, 189, 190, 191, 229, 233
Brahms	Op. 51 No. 1	<i>String Quartet No. 1 in C minor</i> , Movement IV	42, 43, 44, 45, 46, 48, 60, 64

Investigating the source of the problem reveals that nearly all MusicXML files in the corpus do not make use of the appropriate encoding for minor key signatures. Specifically, most pieces with C minor key signatures are encoded as follows:

```

1 <key>
2   <fifths>-3</fifths>
3 </key>

```

Listing 6.3: Key specification in MusicXML without **mode** element

The correct way encoding a minor key would be the following:

```

1 <key>
2   <fifths>-3</fifths>
3   <mode>minor</mode>
4 </key>

```

Listing 6.4: Key specification in MusicXML containing a **mode** element

The root of the problem is that most music notation applications provide the option to select the number of accidentals used (e.g. three flats), but do not provide a selection whether the key is a major or minor key. Thus, the selected key signatures are interpreted and stored as major keys. However, this issue is rooted in inappropriately encoded score data, not in a deficiency in the search algorithm. After adjusting the harmonic context to the parallel key Eb major, more matches are detected, which are listed in Table [6.5](#).

Table 6.5: Search results for the rhythm of Beethoven’s 5th *Symphony* motif in the key Eb major

Composer	Work Number	Title	Measure(s)
Bach	BWV 537	<i>Fantasia and Fugue in C minor</i>	92, 95, 137, 138
Beethoven	Op. 10 No. 1	Sonata No. 5, Movement III	55, 56
Beethoven	Op. 13	Piano Sonata No. 8 (“Pathétique”), Movement III	79
Beethoven	Op. 30 No. 2	Violin Sonata No. 7 in C minor	17, 147
Beethoven	Op. 55	Symphony No. 3 (“Eroica”), Movement I	556
Beethoven	Op. 55	Symphony No. 3 (“Eroica”), Movement IV	332
Beethoven	Op. 60	Symphony No. 4, Movement II	34, 36, 89, 91
Brahms	Op. 51 No. 1	<i>String Quartet No. 1 in C minor</i> , Movement I	35, 36, 45, 46, 47
Mozart	KV 427 No. 1	<i>Kyrie from Great Mass in C minor</i>	62
Mozart	KV 427 No. 18	<i>Agnus Dei from Great Mass in C minor</i>	62
Kern		<i>All The Things You Are</i> , Big Band Arrangement	63, 65
Sampson		<i>Stompin’ at the Savoy</i> , Big Band Arrangement	25

Degree-based Search Results

Searching for the zero-based degree sequence 4 4 4 2 in the corpus yields results in 233 pieces. However, verifying the results reveals that most of them are false positives. This is due to the incorrect encoding of minor keys in the MusicXML files in the corpus, as pointed out in the previous section. The scale degrees 4 4 4 2 in the C minor context are interpreted as 2 2 2 0 in the Eb major context,

resulting mostly in false positive matches. In order to make use of combinations of degree- and key-related search queries, the specifications of minor keys would have to be corrected in all 1122 pieces in the corpus. Afterwards, the full flexibility of degree-dependent search queries could be used. The implementation and evaluation of this procedure is open for future research.

6.8 Conclusion

The search infrastructure implemented in [MPS](#) provides possibilities to formulate and perform context-dependent musical search queries in a user-provided corpus of compositions. Instead of note sequences, combinations of musical aspects can be specified and are translated to search patterns in individual musical context layers. Each composition in the corpus is translated into a corresponding layer representation (see Chapter [3](#)), in which the pattern matching is performed.

The search was successfully applied for queries incorporating rhythms and absolute pitch specifications. Queries including key contexts and pitch specifications relative to keys only partially yielded correct results. However, this is not due to a malfunctioning algorithm, but because of incorrect encodings of keys in the corpus source files.

Since search criteria can be specified in each available musical context layer, a fine-grained and application-specific search is possible, making [MPS](#) a powerful tool for finding specific musical fragments in musical corpora. Future work could address enhancements of the query language, e.g. by introducing boolean operators and advanced options for more flexible search queries. Furthermore, the system could be optimized with regard to performance by employing appropriate database and indexing technologies as proposed in related literature (Sapp et al. [2004](#); Typke et al. [2005](#); Cuthbert, Ariza, Cabal-Ugaz, et al. [2011](#); Viro [2011](#); Sampaio et al. [2013](#)).

Chapter 7

Music Analysis

There is two kinds of music, the good,
and the bad. I play the good kind.

— Louis Armstrong

The context-based models introduced in chapters 3 and 4 constitute an expedient basis for music analysis applications. Large efforts are put into manually performed music analysis in order to find out more about musical structures, patterns, parallel developments, variations and harmonic relations and progressions, to name but a few. Recently, the usage of computer programs was proven effective for a number of these tasks (Cope 2009). An extensive music analysis framework was developed as part of MPS, which facilitates statistical and musicological analysis.

7.1 Motivation

Several aspects have motivated the development of the MPS analysis tool:

1. The underlying context-based model allows developing new analysis methods which benefit from the available context information. Not only explicit information, but also implicit and contextual data can be taken into account and relations between individual context layers can be explored.
2. Once certain analysis algorithms are developed, they can be applied not only to one musical piece, but also easily to hundreds or thousands of pieces. This way, the processing power of computers is leveraged and human research is disburdened from laborious manual work.

3. The proposed system does not require the knowledge of programming skills. Users only have to provide **MIDI** and/or MusicXML files. The system produces output files in **Comma Separated Values (CSV)** format which can be opened with any regular spreadsheet application. Furthermore, **MPS** generates clearly arranged **PDF** analysis reports containing comprehensible data tables and graphical plots of analysis results.
4. By interpreting musical analysis results, insights can be gained into characteristic properties of individual pieces, composers and styles. By using the processing capabilities of computers, statistical commonalities and distinctive features of musical compositions can be explored.
5. The findings obtained from analysis results can be applied in the design of an automated composition system, which is introduced in Chapter **8**.
6. Experiments with the *music21* framework (Cuthbert and Ariza **2010**), conducted in late 2015 prior to developing the analysis system, yielded that a) local corpus access mechanisms were not properly implemented yet and b) that the analysis of a large corpus with more than 1000 pieces took about half a day. A more efficient implementation was required for the composition algorithm developed in Chapter **8**.

7.2 Analysis Scopes

To analyze musical compositions, one or more files containing symbolic music data in **MIDI** or MusicXML format have to be provided. The analysis and export process is configurable on the basis of different analysis scopes. Either a single piece, a collection of pieces or a corpus of pieces can be analyzed, which is visualized in Figure **7.1**.

The first musical analysis scope to be introduced is the single piece scope. It is used to gain insight into a specific musical composition by performing either global analyses or voice-specific analyses. It is also possible to combine the aforementioned analysis modes. Furthermore, musically meaningful sections can be specified in order to perform section-wise analyses, which can also be combined with the other analysis modes.

Collection analysis is suitable for comparing multiple musical pieces among each other. For this purpose, analysis results are shown next to each other in combined representations. Corpus analysis goes one more step further, allowing the comparison of multiple collections of compositions, e.g. folders containing multiple composi-

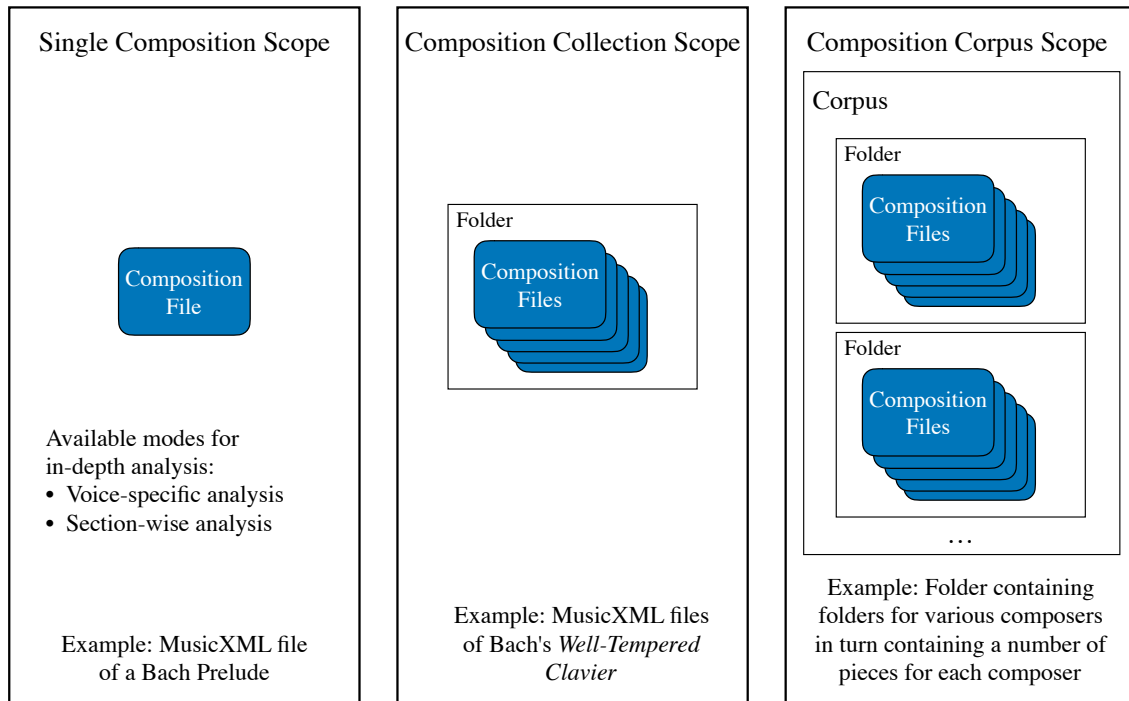


Figure 7.1: Analysis scopes

tions of individual composers. Comparative analyses are explained in greater detail in section [7.6](#).

To demonstrate these analysis techniques for the single piece scope, the first movement of Ludwig van Beethoven's *Piano Sonata No. 14 in C# minor*, commonly known as *Moonlight Sonata*, is used as an example. The musical aspects and relations surveyed by the analysis process are explained and illustrated below.

7.3 Rhythmic Analysis

7.3.1 Note Duration Analysis

In order to get a first notion of the rhythmical characteristics of a piece, a note duration analysis is performed. The result visualizes how often specific note durations are used in a composition. Refer to Figure [7.2](#), which shows a histogram of note durations used in Beethoven's *Piano Sonata No. 14 in C# minor*, Mv. I. Rather than visualizing absolute note duration counts, this figure shows the relative frequencies of the note durations, i.e. percentages which add up to 1.

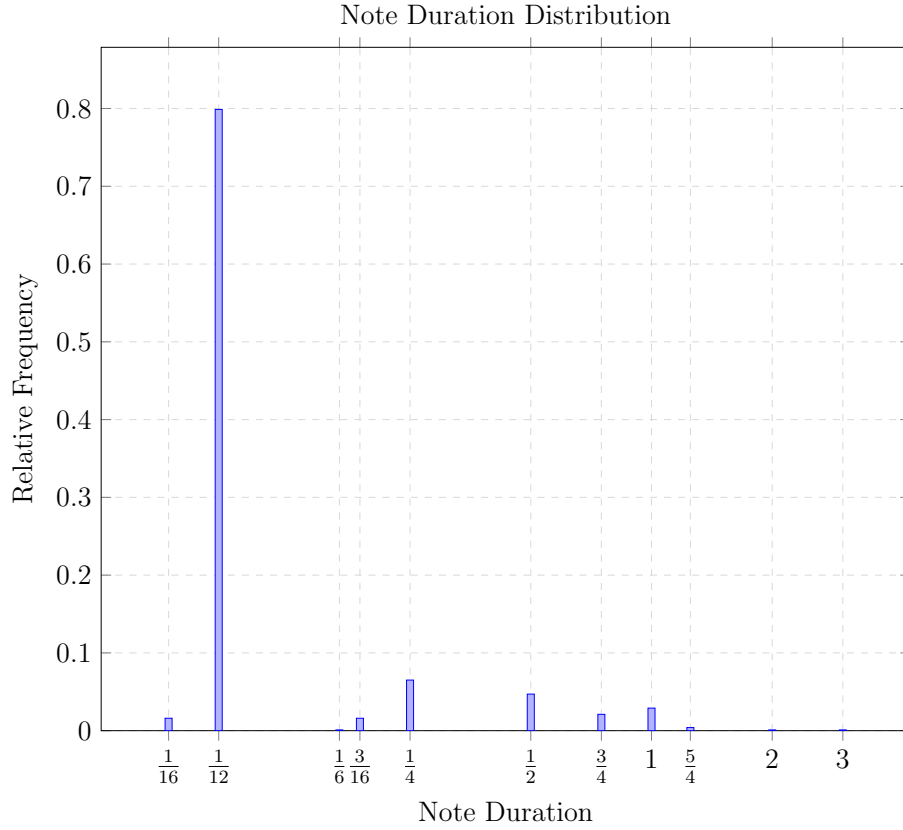


Figure 7.2: Note duration distribution of Ludwig van Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1. The distribution reveals that the composition is heavily based on eighth triplets, which have a fractional duration of $\frac{1}{12}$ and constitute about 80% of the notes in the piece. About 7% of the notes are quarter notes and about 5% are half notes. Tied notes longer than a whole note do only occur rarely.

7.3.2 Note Density Analysis

Another aspect regarding durations is the ratio between durations of notes and rests in the piece. In [MPS](#) this ratio is called *Note Density*. If the composition or a stream (voice) contains notes exclusively, the note density is 100%. The more rests are contained in a composition or voice, the lower the note density. The note density ρ_d can be computed by dividing the total duration of notes by the total duration of notes and rests. Mathematically this is represented in Equation [7.1](#), where d_i is the duration of a stream event with index i in a context layer model stream containing N stream events (see section [3.6](#)).

$$\rho_d = \frac{\sum_{i=0}^N \begin{cases} d_i, & \text{if stream event represents a note} \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=0}^N d_i} \quad (7.1)$$

The note densities for the individual voices in the presented example are: 0.98 (arpeggi), 0.99 (bass accompaniment) and 0.65 (melody).

7.3.3 Beat Analysis

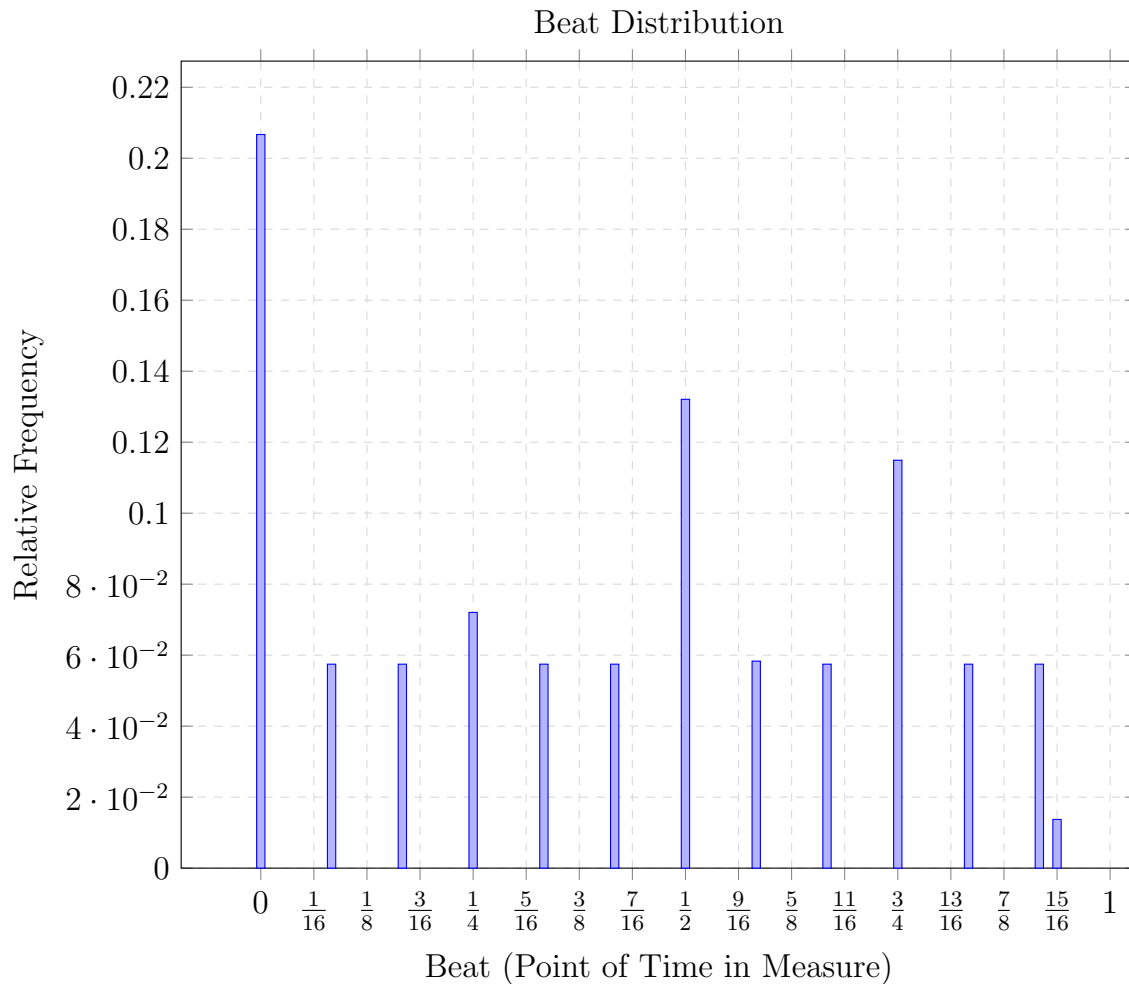


Figure 7.3: Beat distribution visualizing the relative frequencies of note onset times relative to the beginning of the respective measures in Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1

When analyzing music rhythmically, it is very important to consider the metric context of the notes. Two notes with the same duration can have fundamentally different musical meanings when placed on two different beats, i.e. different points of time in a measure. Figure [7.3](#) illustrates how the notes of the first movement of the *Moonlight Sonata* are distributed over the measures.

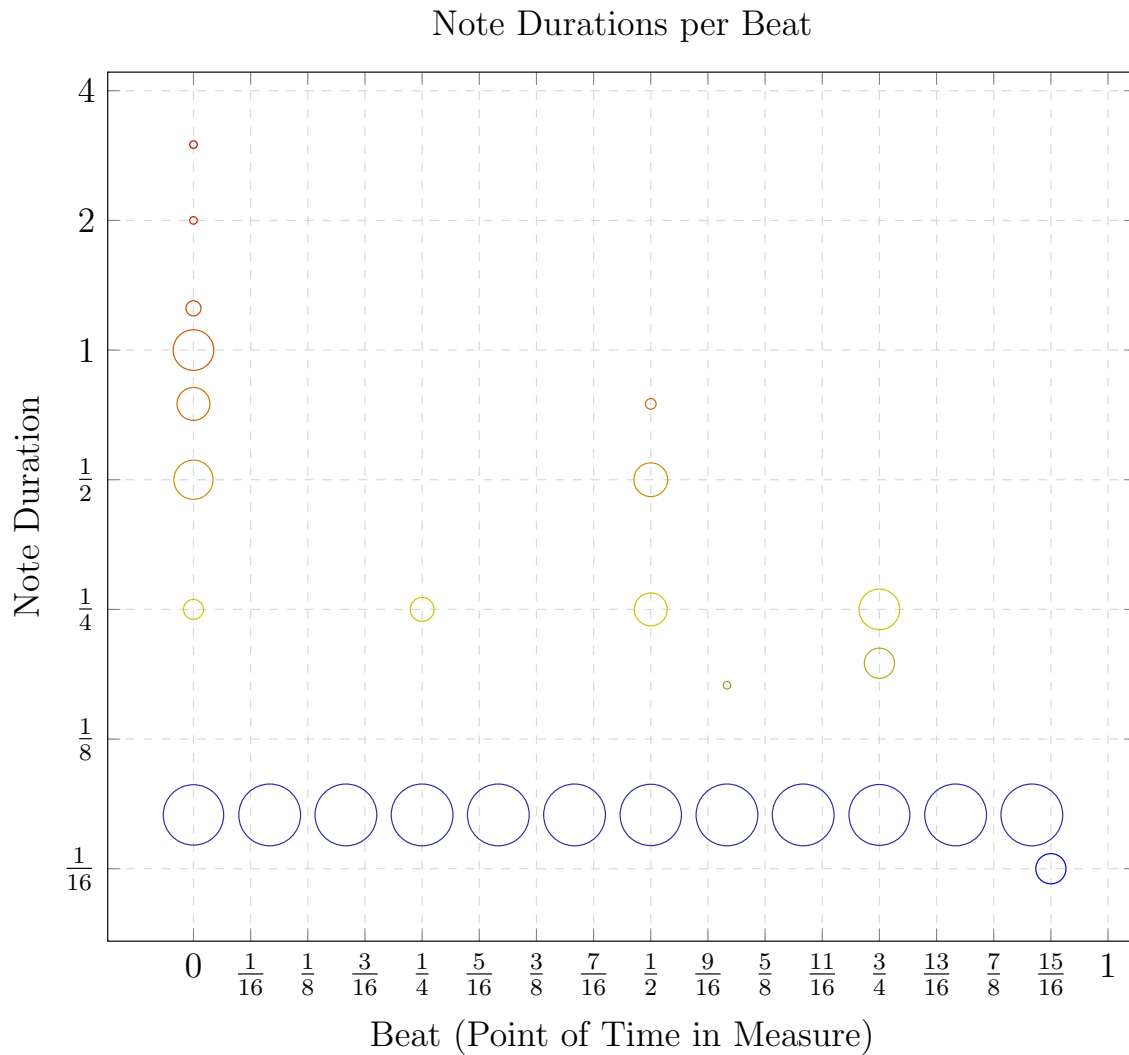


Figure 7.4: Note duration distribution dependent on beats of Ludwig van Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1. Note durations are additionally visualized by means of colors, where blue corresponds to short durations and red to long durations.

7.3.4 Combined Note Duration and Beat Analysis

An even more meaningful representation of the analysis results can be obtained by combining duration and beat analysis in one plot, as demonstrated in Figure 7.4. It depicts a three-dimensional representation of the data visualizing triplets of note durations, beats and their respective relative frequency. The latter is visualized by the area of the corresponding circle, the radius of which is proportional to the square root of the relative frequency.

Voice-specific Note Duration and Beat Analysis

By narrowing down the analysis scope even more, more complex correlations can be visualized. Figure 7.5 shows the relationship between note durations, beats, voices

and relative frequencies of combinations of the aforementioned elements. By means of the resulting figure, rhythmical similarities and differences between individual voices can be detected.

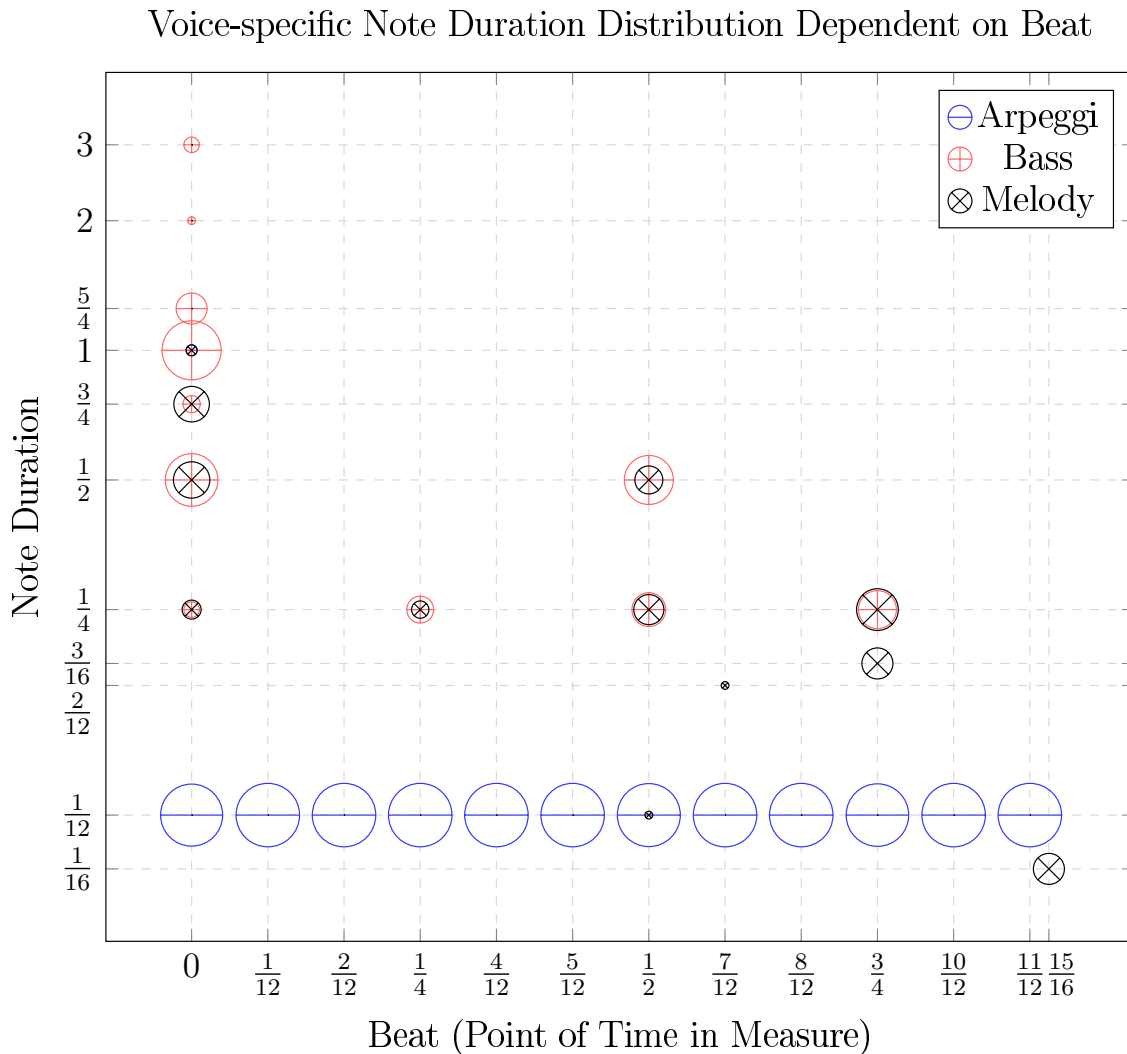


Figure 7.5: Voice-specific note duration distribution dependent on beats of Ludwig van Beethoven's *Piano Sonata No. 14 in C# minor*, Mv. 1

7.4 Pitch Analysis

The previous sections were concerned with rhythm analysis, thus far intentionally omitting the pitch dimension. This section explicitly covers pitch-related analysis.

7.4.1 Piano Roll Representations

A piano roll representation provides a simple yet accurate overview over the pitches used in a composition. It can be considered a very simple score containing only

simple pitch information dependent on time. Figure 7.6 depicts a piano roll representation of the first movement of the *Moonlight Sonata*, in which absolute and relative pitch maxima and minima can easily be identified. Furthermore, piano roll representations are useful for identifying repeating parts and sections of musical pieces visually.

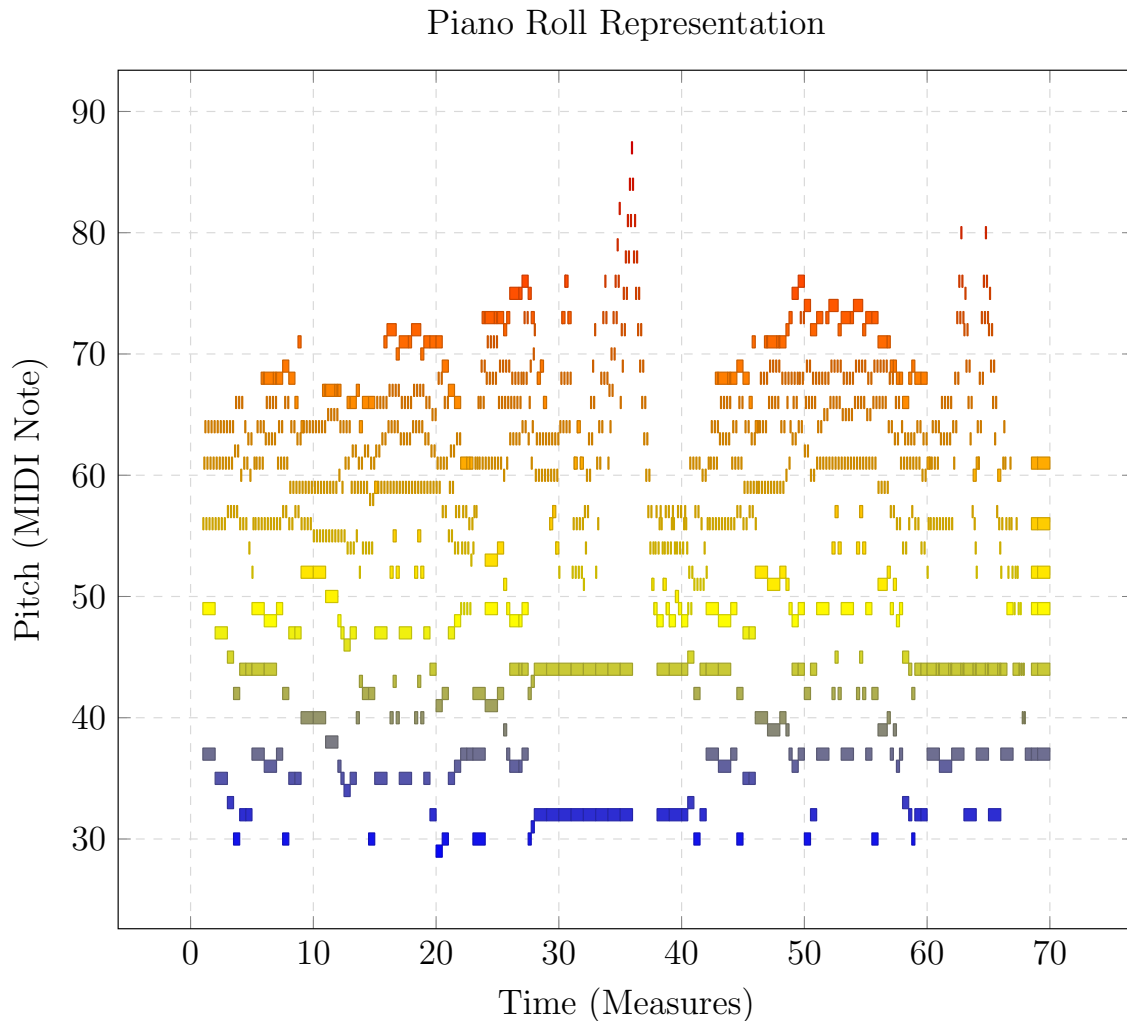


Figure 7.6: Piano roll representation of Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1

7.4.2 Pitch Distributions

Pitch distributions indicate how often specific pitches are used throughout a musical piece. An example is depicted in Figure 7.7.

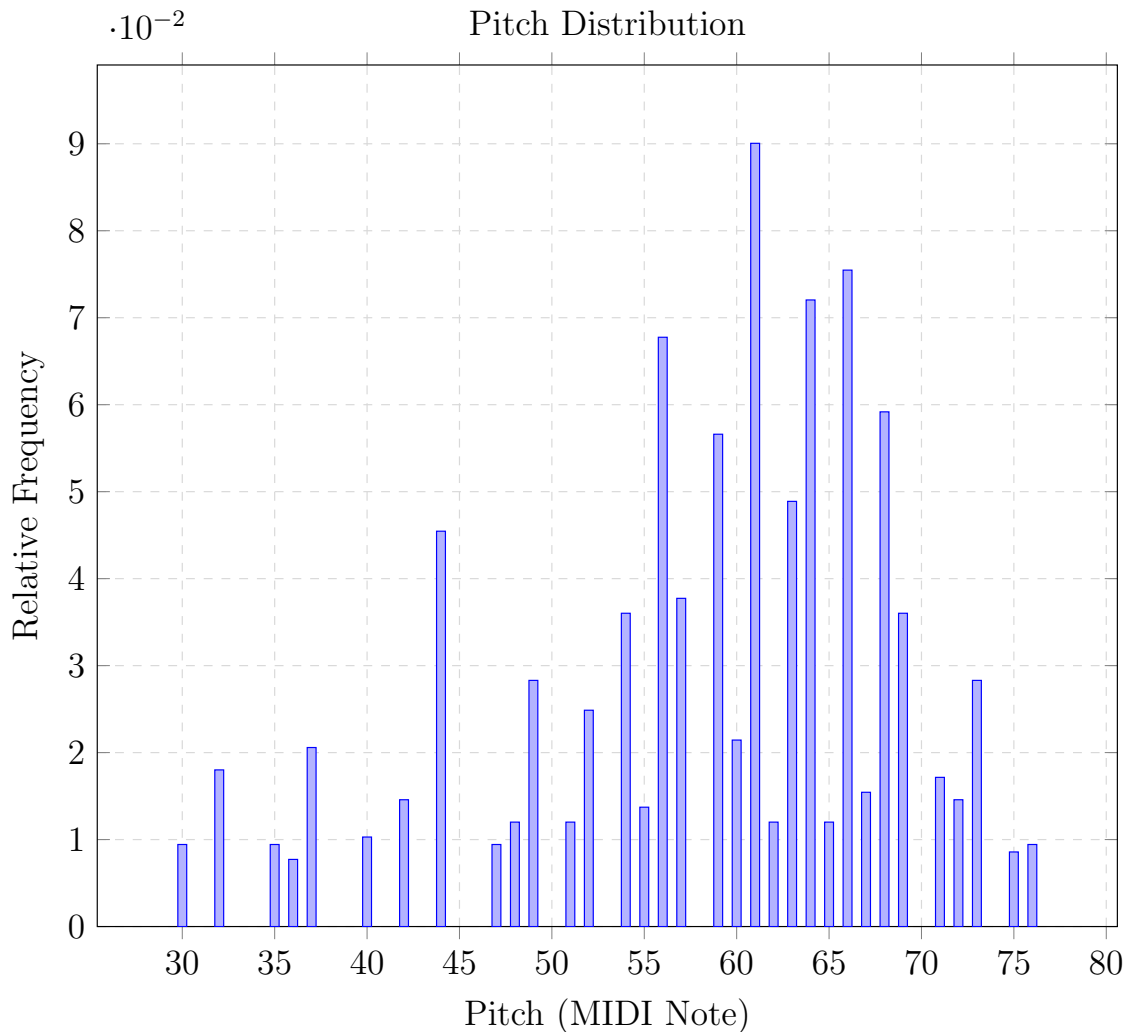


Figure 7.7: Pitch distribution of Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1

7.4.3 Interval Analysis

Not only is it relevant to analyze individual pitches in music, but also to analyze combinations of pitches and their relations. Musically speaking, the distance between two pitches is referred to as an interval (Randel [2003](#), pp. 413ff.). Intervals can be analyzed a) by examining consecutively audible pitches or b) by considering simultaneously sounding pitches. Biles refers to intervals in case a) as *horizontal intervals* and in case b) as *vertical intervals* (Biles [2007a](#)). Both analysis techniques are introduced in the following sections.

Horizontal Interval Analysis

This section covers the analysis of successive pitches in musical compositions, which is visualized in Figure [7.8](#). When summarizing the counted interval leaps in a histogram as depicted in Figure [7.9](#), conclusions can be drawn about the pitch progres-

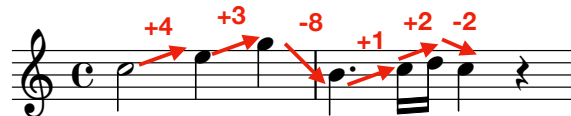


Figure 7.8: Analyzing interval leaps by computing semitone differences to the respective preceding pitches. Example shows excerpt from *Piano Sonata No. 16 in C major, K. 545* by W. A. Mozart

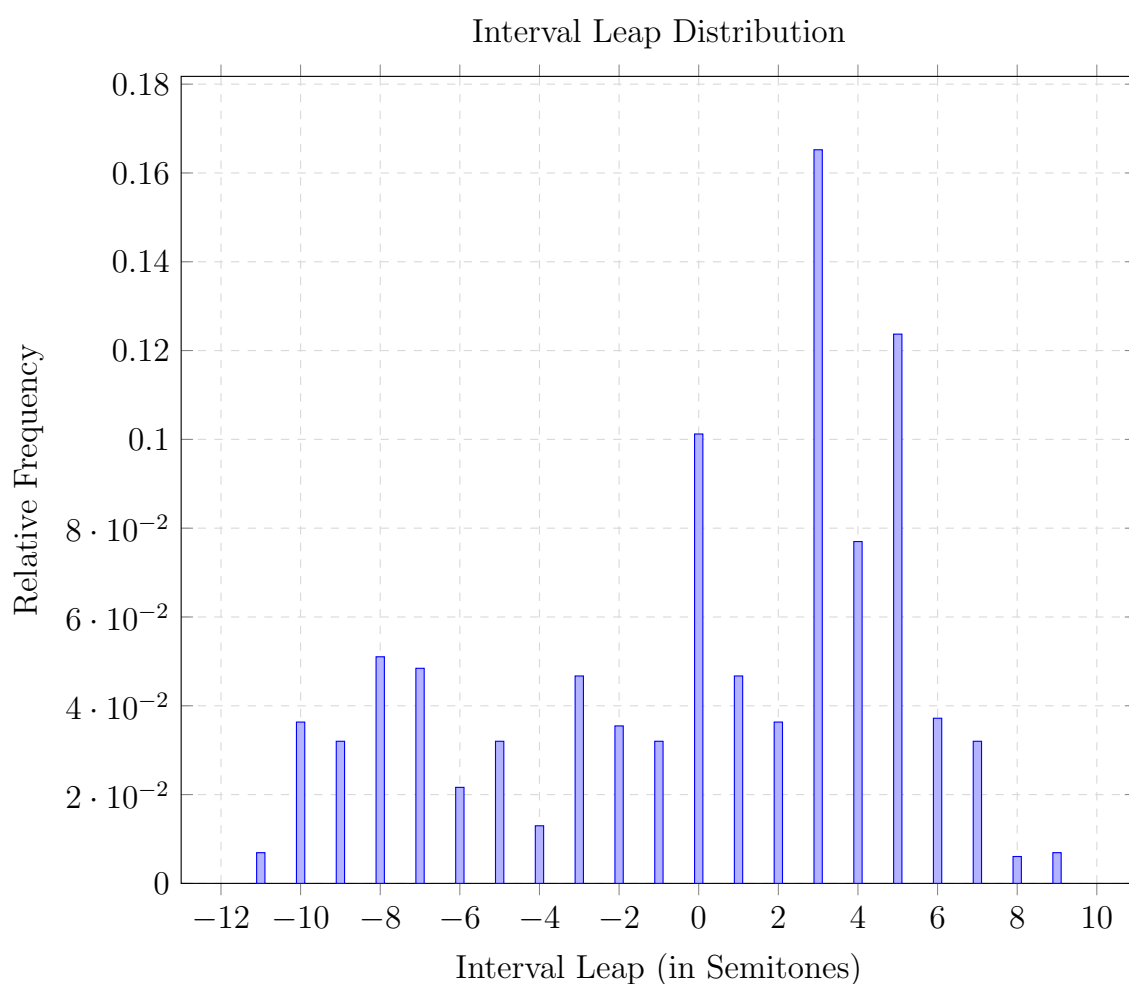


Figure 7.9: Interval leap distribution of Beethoven's *Piano Sonata No. 14 in C# minor, Mv. 1*. The most frequently used interval is an ascending minor third (+3 semitones), followed by ascending perfect fourths (+5 semitones) and repeated pitches (interval leap of 0 semitones).

Vertical Interval Analysis

The combinations of simultaneously audible pitches in musical pieces play a major role in music analysis, because they have a pivotal impact on the listener's auditory impression. Pitch constellations are especially important to determine consonance, dissonance and harmony in musical compositions, which is explained in detail below.

7.4.4 Dissonance Analysis

Whenever one or multiple pieces sound concurrently, they produce a certain consonance or dissonance. To a certain extent, the degree of consonance or dissonance can be physically explained and mathematically computed. The level of dissonance of two tones with individual fundamental frequencies f_1 and f_2 can be analyzed by comparing the frequency ratio of f_1 and f_2 . Figure 7.11 illustrates the dissonance values of all semitone intervals within two octaves. The ratios can also be derived from the string lengths required to produce two pitches with the given interval distance, and are also dependent on the tuning system used (Randel 2003, pp. 414ff.). For example, a major ninth has a frequency ratio of 9 : 4. The dissonance for this interval can be obtained by computing the *Tenney Height* or *Tenney Harmonic Distance*, which is defined in Equation 7.2, where a and b are the numerator and the denominator of a frequency ratio, respectively (M. M. Deza and E. Deza 2016, p. 415).

$$\text{HD}(a, b) = \log_2(a \cdot b) \quad (7.2)$$

It follows that the dissonance of a major ninth is approximately 5.17, as shown in Equation 7.3.

$$\log_2(9 \cdot 4) = \log_2(36) \approx 5.17 \quad (7.3)$$

In case of more than two notes sounding at the same time, all interval combinations are taken into account. For example, in the D major chord (consisting of the pitches D, F \sharp and A) there are three interval combinations: D-F \sharp , F \sharp -A and D-A. In general, a chord consisting of n notes contains $\frac{n \cdot (n-1)}{2}$ intervals pairs, as shown in Equation 7.4 (Neapolitan 2015, p. 96ff.).

$$\binom{n}{2} = \frac{n!}{2! \cdot (n-2)!} = \prod_{j=1}^2 \frac{n+1-j}{j} = \frac{n}{1} \cdot \frac{n-1}{2} = \frac{n \cdot (n-1)}{2} \quad (7.4)$$

The dissonance value of n simultaneously sounding notes is defined as the average dissonance of all interval combinations, the formula of which is given in Equation 7.5.

Dissonance Values of Simultaneously Sounding Intervals Within Two Octaves

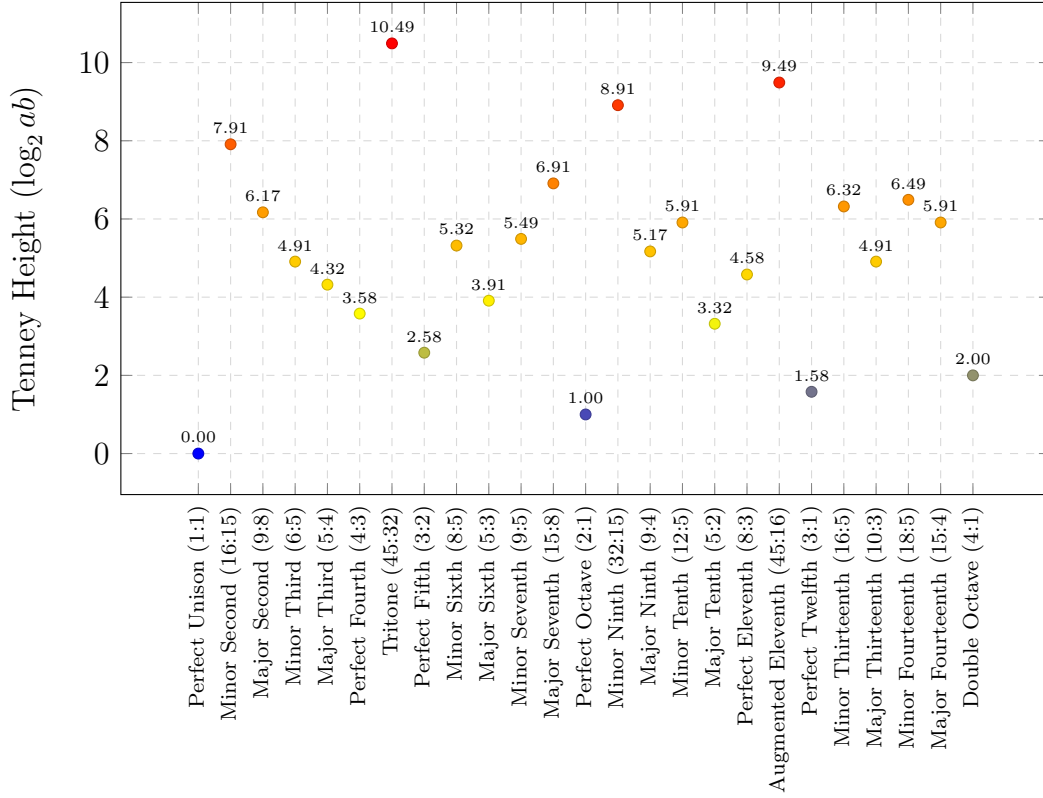


Figure 7.11: Dissonance values of simultaneously audible intervals within two octaves

The variables a_k and b_k designate the numerator and the denominator of the k^{th} interval frequency ratio, respectively.

$$\frac{1}{n} \sum_{k=1}^n \text{HD}(a_k, b_k) = \frac{1}{n} \sum_{k=1}^n \log_2(a_k \cdot b_k) \quad (7.5)$$

Excursus: Relationship between Harmony and Polymetre

Pitches and rhythms are closely related to each other. The fundamental frequencies of tones are physically described in terms of a frequency in Hertz (Goldstein 2013, pp. 263-265). Oscillations, occurring a certain number of times per second, can also be interpreted as a regular rhythm on a small time scale. Consider the two rhythms in Figure 7.12, representing the proportions of three beats against two. If these are played at rapid speed, two tones would be audible in the interval of a perfect fifth. This interval is characterized by the frequency ratio 3 : 2, just as the rhythm shown in Figure 7.12. It follows that pitches can be interpreted as rhythms on a different time scale than conventional rhythms. Based on this observation, Barlow has explored the relationships between harmony and polymetre (Barlow 2012, p. 47).

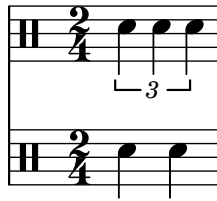


Figure 7.12: Rhythm representing the ratio 3 : 2. When played at very high speed the interval of a perfect fifth, which has the same frequency ratio, is audible. Barlow therefore proposed that harmony is a special case of polymetre (Barlow [2012](#), p. 47).

Dissonance Over Time

[MPS](#) analyzes each combination of simultaneously audible pitches in musical pieces, resulting in a series of dissonance values over time. A graphical representation of such a progression is shown in Figure [7.13](#). This representation allows to identify positions with notably high consonance or dissonance in a piece.

7.4.5 Harmonic Analysis

A particularly important field in music analysis is harmonic analysis. It involves deriving designations of chords being formed by simultaneously audible notes in compositions. This is a task which can be handled conveniently by computers. Based on the already introduced context layer model (see Chapter [3](#)), the analysis algorithm can access relevant harmonic context information in an individual layer, if this information is contained in the corresponding input files. To this end, MusicXML supports the encoding of keys and local context harmonies. If no explicit contextual harmony information is available, MPS attempts deriving the harmonies from simultaneously audible pitches, which are referred to as *implicit harmonies*.

Harmony Distributions

The distributions of harmonies (either explicitly given or implicit harmonies) can be visualized in histograms as shown in Figure [7.14](#). Note that the program does not simply count the number of times a harmony is encountered, but takes the duration of individual harmonic contexts into account when computing the histogram.

Beat-dependent Harmony Distributions

Another interesting question regarding harmonic analysis is the temporal distribution of harmony changes dependent on the time in the respective measures. In Figure [7.15](#), the three-dimensional interrelations between harmony changes, beats and relative quantities of harmony-beat pairs are illustrated.

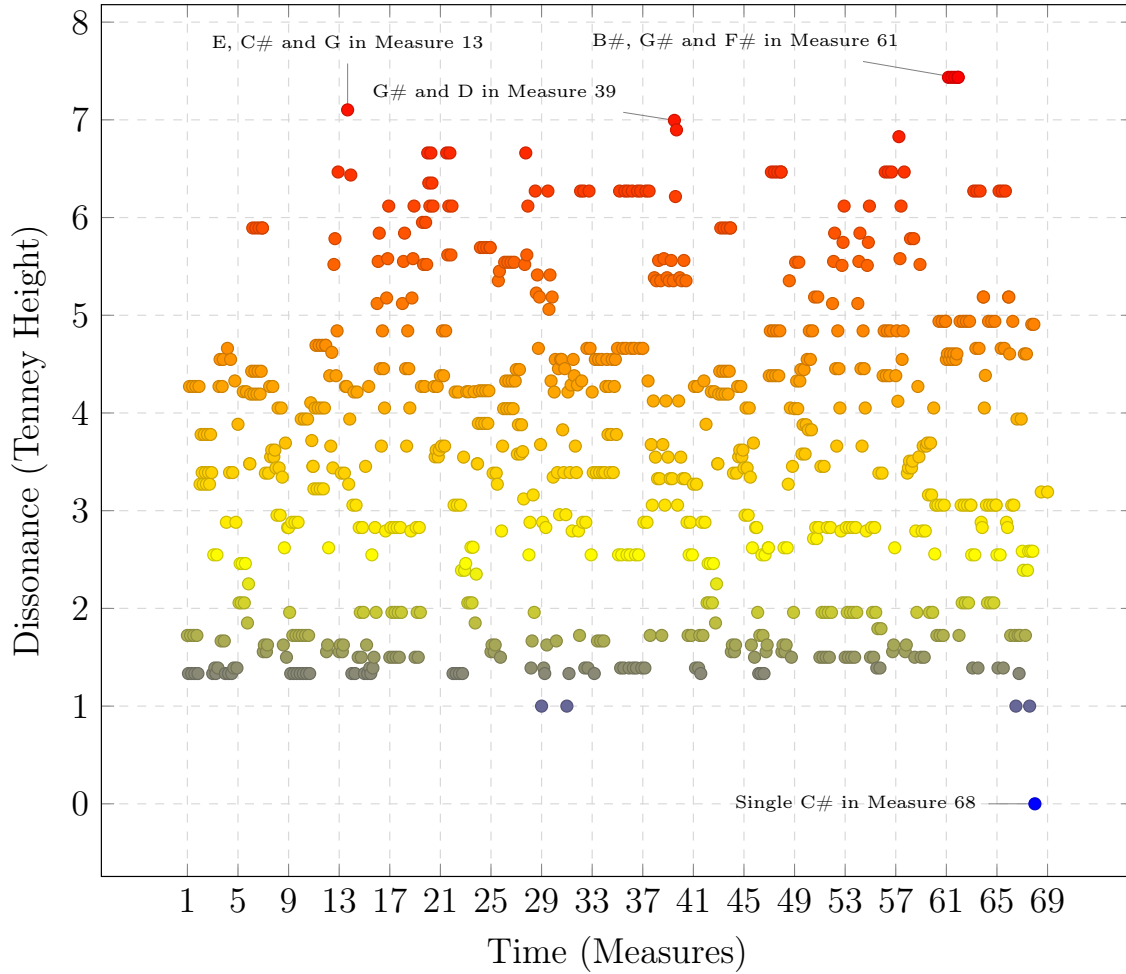


Figure 7.13: Dissonance plot of Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1. Dissonance values are additionally visualized by means of colors, where blue corresponds to consonant and red to dissonant. Local and global dissonance minima and maxima are marked.

Harmony Compliance Analysis

Each note is analyzed in relation to its corresponding harmonic context. In particular, the ratio of harmony compliant notes c_h is computed by dividing the number of notes belonging to the context harmony by the total number of notes (see Equation [7.6](#)). This value (between 0 and 1) provides information about how harmony becomes manifested in the notes being played. In the presented example, the harmony compliance is 0.83.

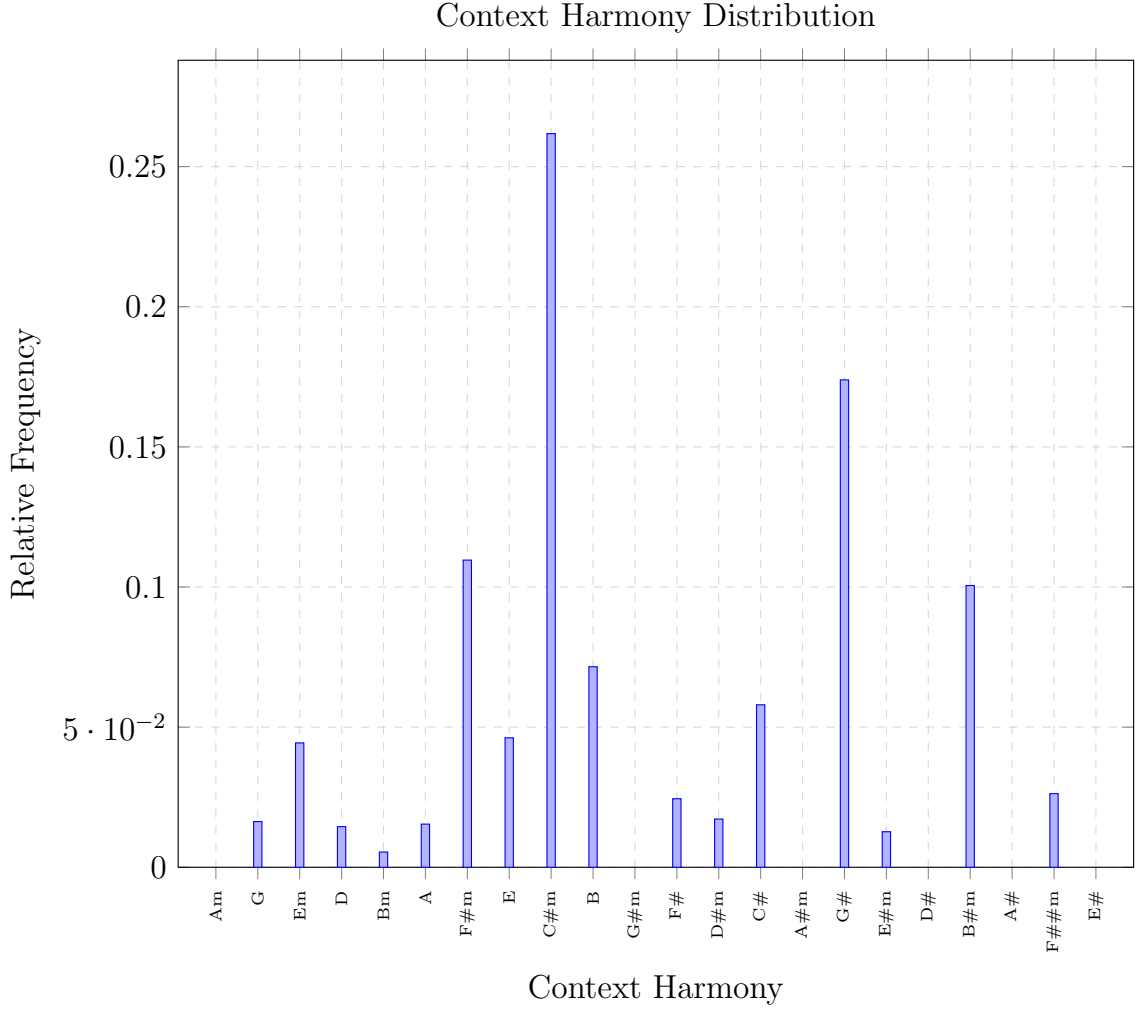


Figure 7.14: Harmony distribution of Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1. The x-axis is sorted according to the circle of fifths. Note that chords with the same root note and kind are combined for clarity, e.g. $C\sharp m$ and $C\sharp m/B$ are both associated with the $C\sharp m$ bin.

$$c_h = \frac{\sum_{i=0}^N \begin{cases} 1, & \text{if stream event contains pitch which belongs to context harmony} \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=0}^N \begin{cases} 1, & \text{if stream event contains pitch} \\ 0, & \text{otherwise} \end{cases}} \quad (7.6)$$

Scale Compliance Analysis

The same concept introduced in the previous section is applied here to contextual scales. The ratio of scale-compliant notes is determined both for the scale matching the current key and for the scale matching the current context harmony (see

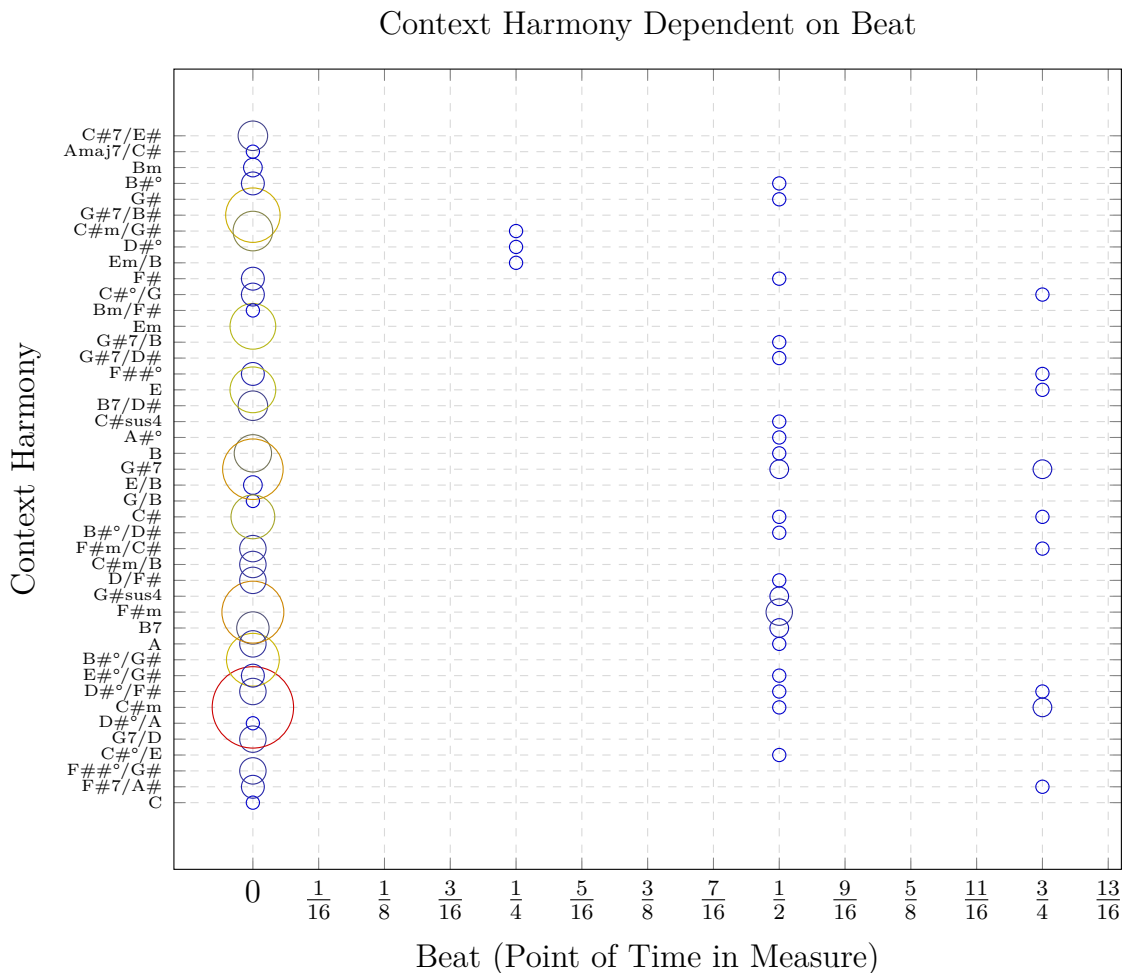


Figure 7.15: Beat-dependent harmony distribution of Beethoven's *Piano Sonata No. 14* in *C# minor*, Mv. 1. Circle areas are proportional to the relative frequencies of the corresponding beat-harmony pairs. Furthermore, circles are colored dependent on relative frequencies for better visual grouping, where blue corresponds to low percentages and red to high percentages. Most harmony changes happen on the very first beat of the measures, C#m being the most frequently used chord. Fewer changes take place on the 3rd and 4th quarter beat of the measures. Only very few changes are placed on the second quarter beat. No harmony changes happen between quarter beats.

equations [7.7](#) and [7.8](#), yielding values between 0 and 1. High values indicate a very strict adherence to tonality. Values are lower for more experimental and free music styles such as jazz or progressive genres.

$$c_{sk} = \frac{\sum_{i=0}^N \begin{cases} 1, & \text{if stream event contains pitch which belongs} \\ & \text{to scale corresponding to current key} \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=0}^N \begin{cases} 1, & \text{if stream event contains pitch} \\ 0, & \text{otherwise} \end{cases}} \quad (7.7)$$

$$c_{sh} = \frac{\sum_{i=0}^N \begin{cases} 1, & \text{if stream event contains pitch which belongs to scale} \\ & \text{corresponding to current harmony} \\ 0, & \text{otherwise} \end{cases}}{\sum_{i=0}^N \begin{cases} 1, & \text{if stream event contains pitch} \\ 0, & \text{otherwise} \end{cases}} \quad (7.8)$$

7.5 Progression Analysis

The [MPS](#) analysis tool also provides facilities to visualize musical progressions. Due to the underlying layer-based model, progressions can be visualized for every available musical dimension. In this section, examples are demonstrated for harmonic and lyric progressions. The system offers three modes to render progression graphs, in which the edge labels have individual meanings:

- Occurrence count: displays how often a transition was detected
- Measure numbers: shows the measure numbers in which the corresponding transition was detected
- Markov model: shows transition probabilities

7.5.1 Harmonic Progression Graphs

A beneficial representation for studying harmonic progressions in musical compositions are harmonic progression graphs generated by [MPS](#), as demonstrated in [Figure 7.16](#). The graphs can also be rendered as Markov models, as already demonstrated in [section 2.3.3](#). An alternative projection of harmonic progression graphs on the circle of fifths is presented in [Figure 7.17](#). The harmonic progressions of Paul Desmond's

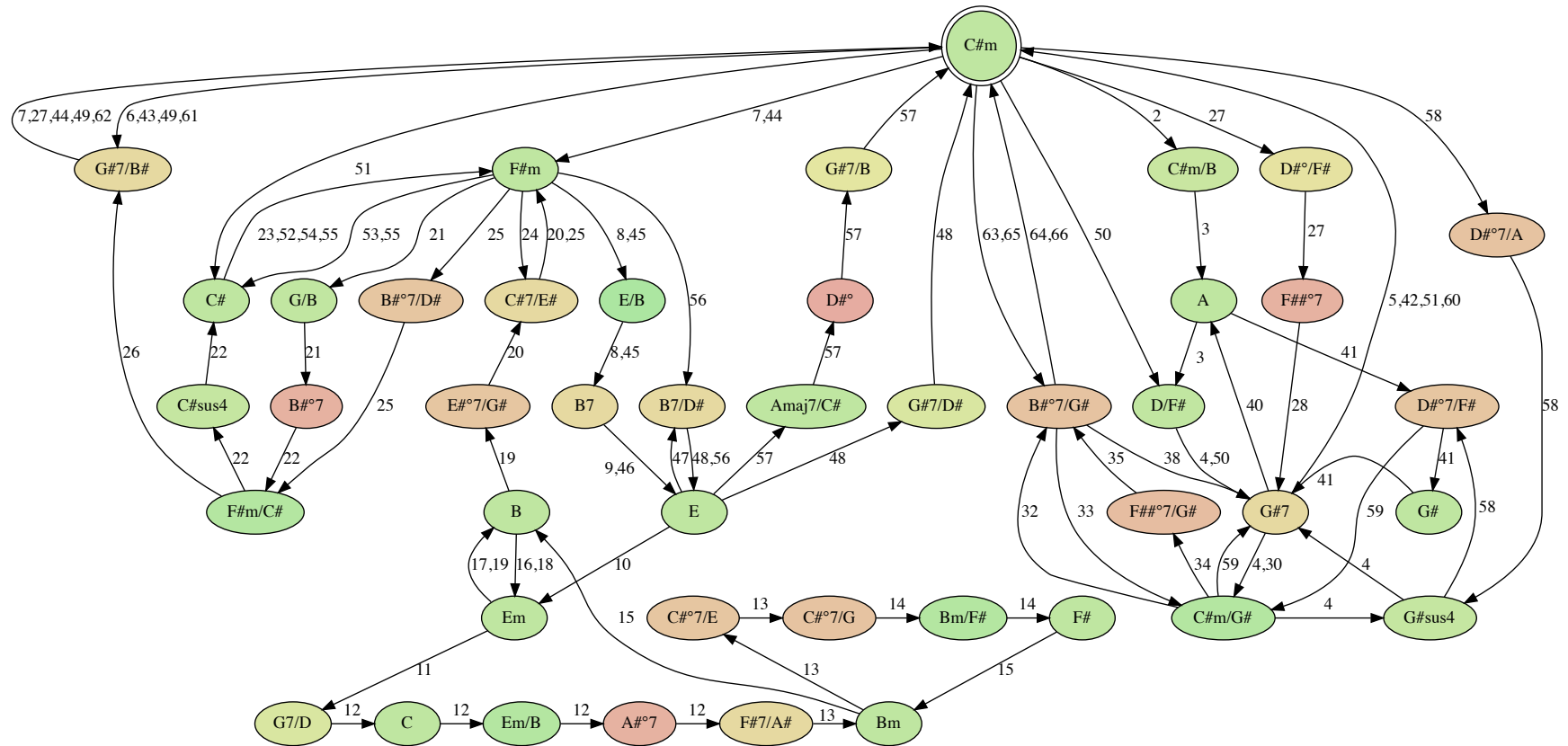
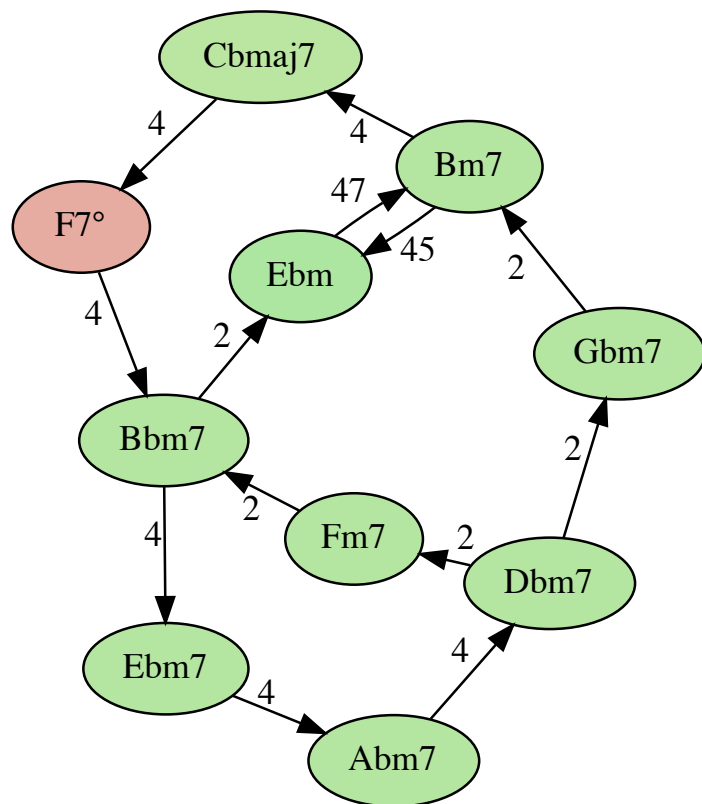
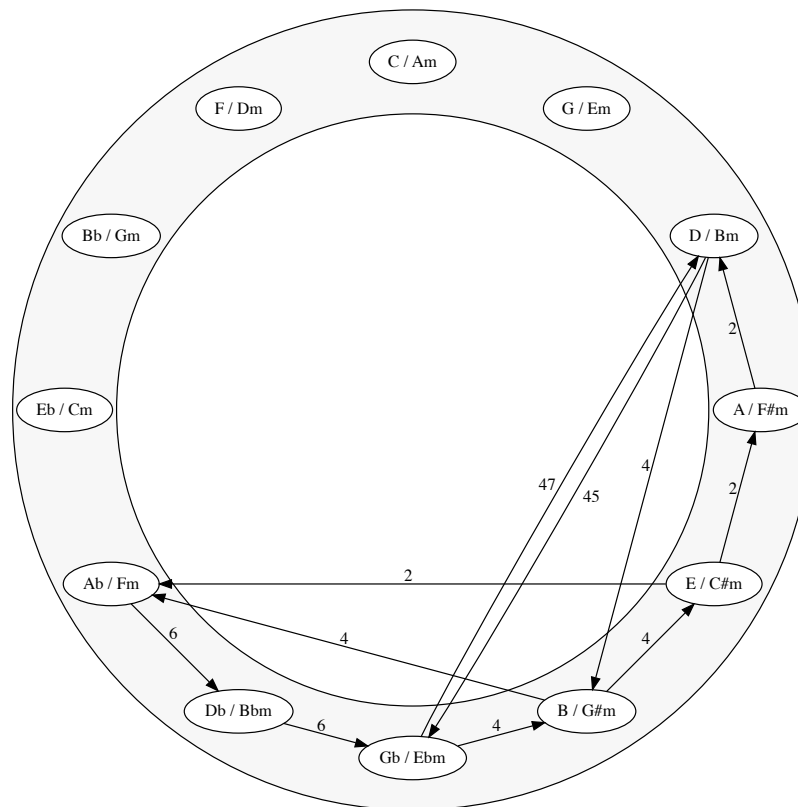


Figure 7.16: Chord progression graph of Beethoven’s *Piano Sonata No. 14 in C# minor*, Mv. 1. The numbers specify the measures in which the corresponding harmonic change was detected. The colors of the chords encode the consonance or dissonance of the relevant chord (green corresponds to consonant and red to dissonant). The graph reveals which harmonic progressions are only used once and which are used multiple times. The latter are easily identifiable due to comma-separated enumerations of measure numbers in which the corresponding transitions occur.



(a) Chord progression graph



(b) Circle of fifths projection of the chord progression graph. It provides an intuitive way of assessing the harmonic complexity of a piece.

Figure 7.17: Harmonic progression graph of Paul Desmond's *Take Five* projected onto the circle of fifths. Edge labels indicate the absolute number of occurrences of the corresponding harmonic transition.

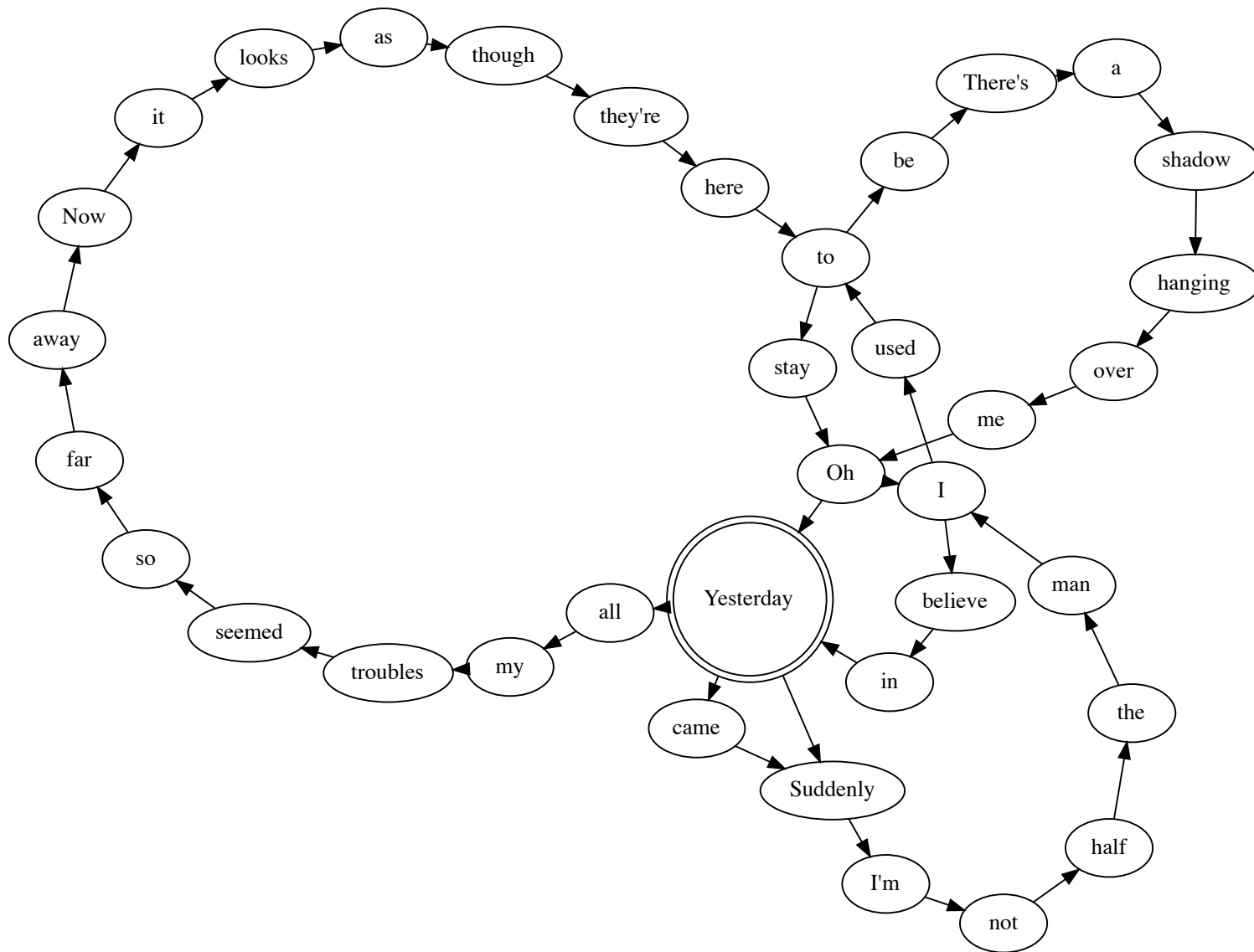


Figure 7.18: Graph visualizing lyric progressions of the first two verses in *Yesterday* by the Beatles

Take Five are used in this example. The circle of fifths projection is useful for assessing the harmonic complexity of a piece visually.

7.5.2 Lyric Progression Graphs

To demonstrate the versatility of the progression graph generator for other context layers, Figure [7.18](#) shows a graph of lyric progressions in *Yesterday* by the Beatles.

7.6 Comparative Analysis of Large Corpora

When studying musical compositions, new insights can also be gained by comparing multiple pieces. Music Processing Suite allows to juxtapose collections and large corpora of compositions.

7.6.1 Comparing Composition Collections

To demonstrate the analysis process and visualisations for multiple compositions, all 24 preludes of Johann Sebastian Bach's *The Well-Tempered Clavier, Book I* were analyzed. Selected statistical plots are demonstrated and discussed.

Note Duration Distributions (Collection Scope)

Figure [7.19](#) contains the combined note duration distributions of all 24 preludes of J. S. Bach's *The Well-Tempered Clavier, Book I*. By identifying markers with unique positions and a certain distance to other markers, exceptional preludes regarding note durations can be identified. For example, the sixteenth triplet notes in preludes 6 and 15 are notated as regular sixteenth notes, as can be seen in Figure [7.20](#).

Key Distributions (Collection Scope)

An aggregated plot visualizing the notated keys used in the 24 preludes of the first book of the *Well-Tempered Clavier* by J. S. Bach is presented in Figure [7.21](#). It becomes apparent that not all keys are continuously used. Furthermore, the plot reveals an asymmetry of keys with sharps and keys with flats (12 preludes with up to 7 sharps, but only 10 preludes with up to 6 flats).

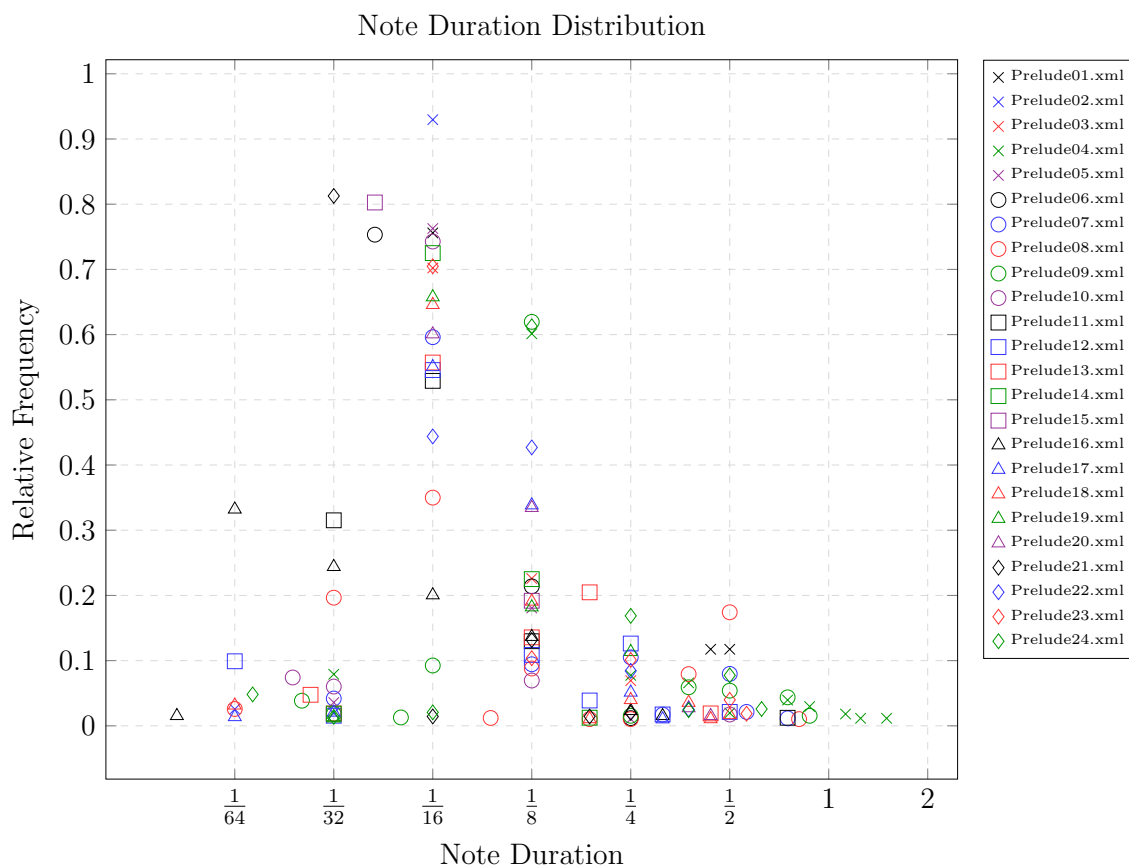


Figure 7.19: Aggregated note duration distribution of the 24 preludes in Johann Sebastian Bach’s *The Well-Tempered Clavier, Book I*. Prelude 2 in C minor is the rhythmically most uniform piece containing about 92% 16th notes, followed by Prelude 21 with about 81% 32nd notes. The markers for Prelude 6 and 15 are quite interesting. These preludes contain many 16th triplet notes, explaining why these markers are located between 16th and 32nd notes. See Figure [7.20](#) for an excerpt showing the notation of the corresponding notes.



Figure 7.20: J. S. Bach, *Prelude No. 15 in G Major, BWV 860*, m. 1. The notes in the right hand are notated as regular sixteenth notes but are actually sixteenth triplets.

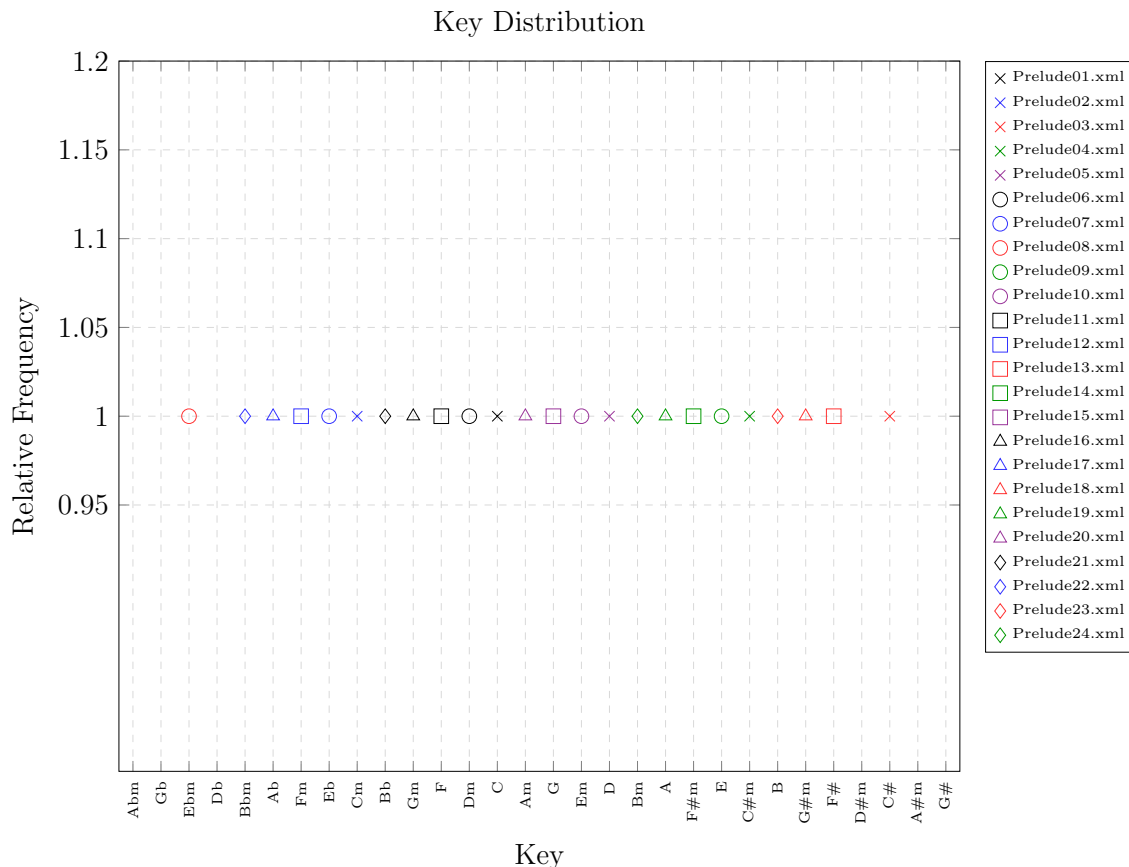


Figure 7.21: Aggregated key distribution of the 24 preludes in Johann Sebastian Bach’s *The Well-Tempered Clavier, Book I*. Not all keys are continuously covered by the preludes. In particular, the keys Db major and D♯ minor were left out. Furthermore, there is an asymmetry between preludes with sharps and preludes with flats: there are 12 preludes up to 7 sharps but only 10 preludes up to 6 flats.

7.6.2 Analyzing Large Corpora

To conduct musical analyses in an even broader scope, large corpora of musical works can be processed using **MPS**. A typical scenario would be to analyze a large corpus containing selected works of various composers, styles and eras in order to find similarities and differences. In the following examples, the corpus introduced in Chapter **6** was analyzed. A copy of the corpus are available on the accompanying CD (see Appendix **A**). The complete analysis of all 1,122 pieces in the corpus containing 99,700 measures, 7,271 separate voices and 2,067,580 notes, took about one and a half hours.

Note Duration Distributions (Corpus Scope)

An aggregated note duration distribution for various composers is shown in Figure **7.22**. It allows a comparison of note duration usage for the individual composers.

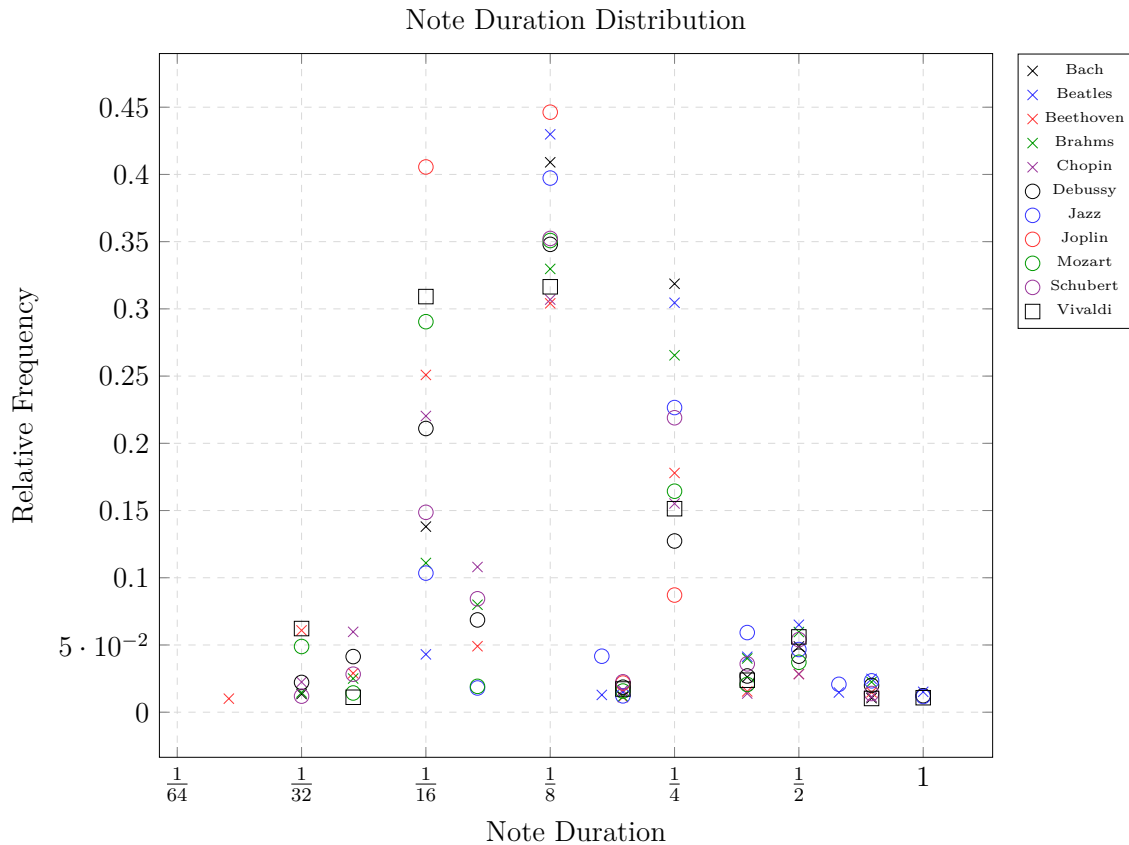


Figure 7.22: Aggregated note duration distributions analyzed from a large corpus containing compositions of various composers. Scott Joplin uses significantly more 16th notes than the other analyzed composers. The most frequently used note durations are eighth notes, which are consistently used in a relative quantity over 30% by all composers. J. S. Bach uses most quarter notes of all composers in the corpus. Eighth and sixteenth triplets are most frequently used by Chopin.

Beat Distributions (Corpus Scope)

In order to explore on which points of time in measures individual composers place notes, an aggregated beat distribution as depicted in Figure 7.23 is generated.

Interval Leap Distributions (Corpus Scope)

A comparative visualization of horizontal interval leaps detected in the large corpus is shown in Figure 7.24. It reveals differences in the composition of successive pitches, especially in the middle and by the margins of the histogram.

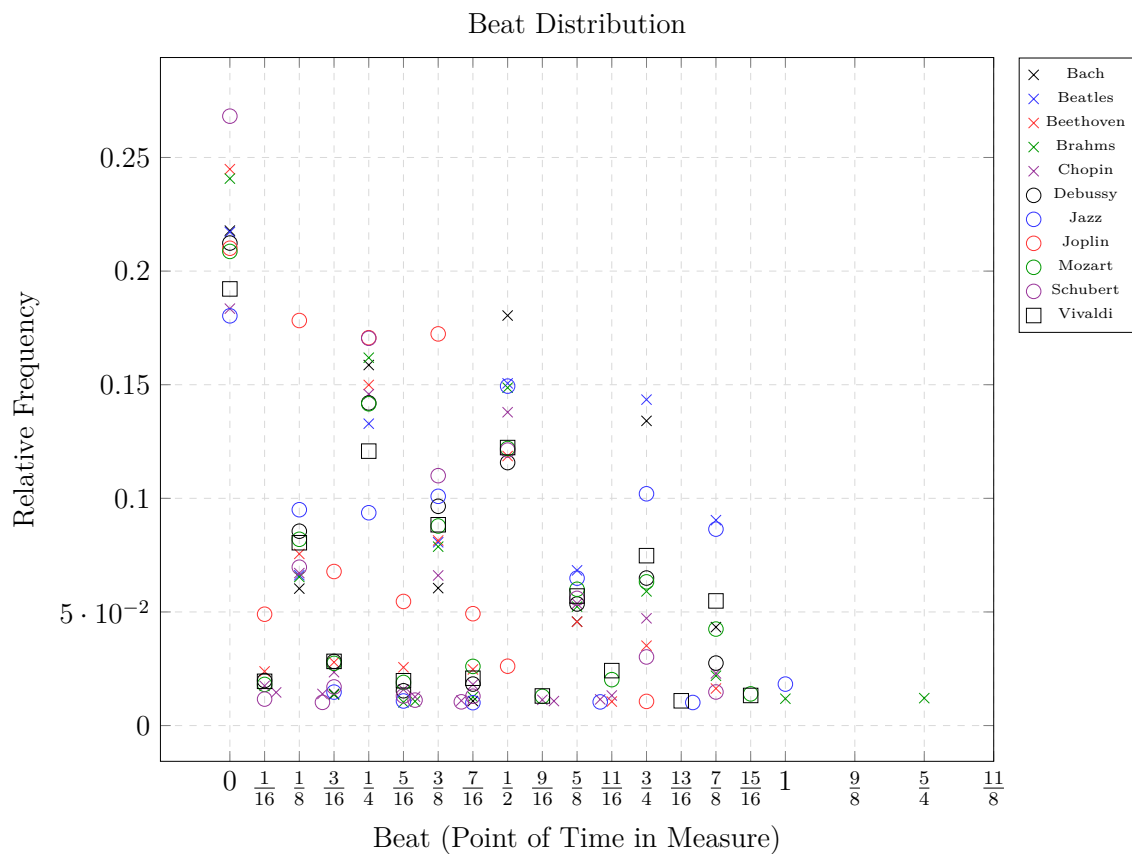


Figure 7.23: Aggregated beat distributions analyzed from a large corpus containing compositions of various composers. Generally, the increasing use of syncopation is visible over time, most notably when comparing Scott Joplin's beat distribution to the ones of other composers.

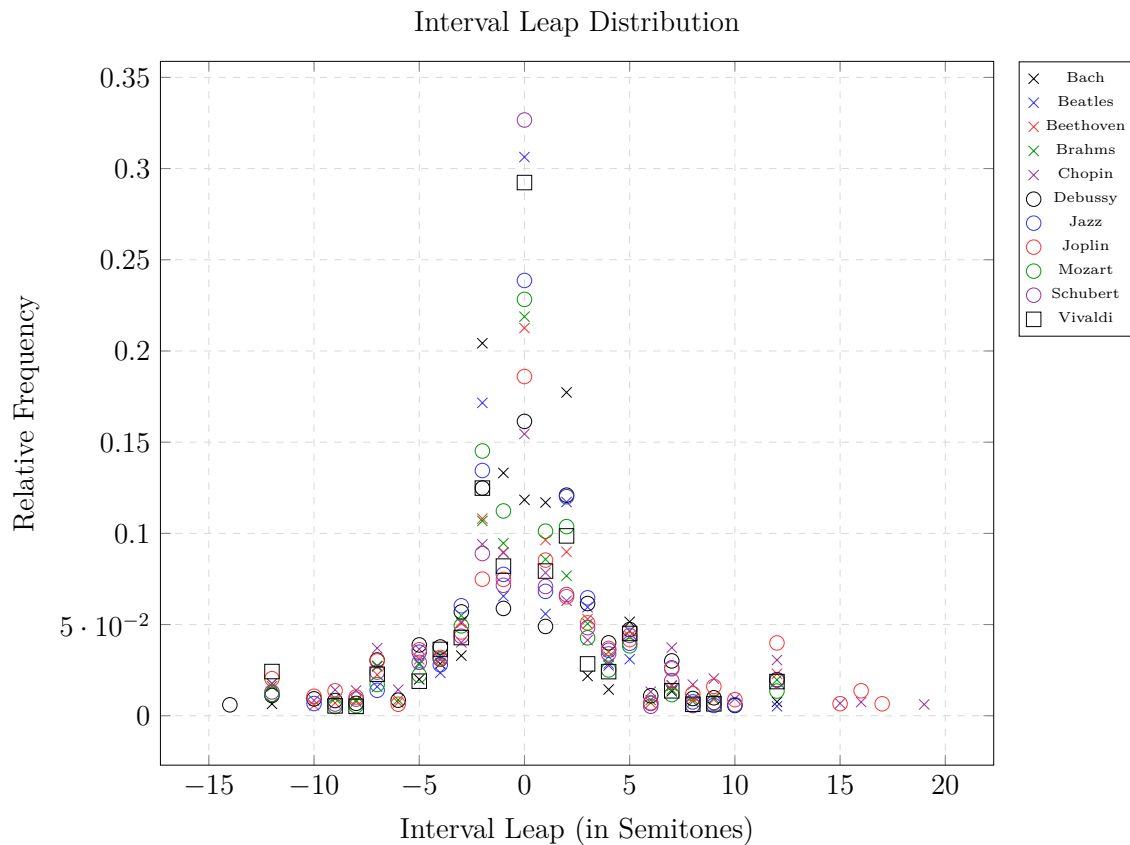


Figure 7.24: Aggregated interval leap distributions analyzed from a large corpus containing compositions of various composers. Debussy is the only composer using large descending interval leaps which are greater than an octave. The only composers using very high ascending leaps are Chopin and Joplin. The interval leap distribution of J. S. Bach is unique: He uses significantly less pitch repetitions (interval leap of 0) than the other composers, but on the other hand uses more descending and ascending minor and major seconds than his colleagues.

7.7 Conclusion

MPS provides extensive tools for music analysis, which can be applied in different scopes. In-depth analyses for single pieces can be performed, as well as comparative analyses of multiple pieces. No programming knowledge is required, but the analysis procedure is invoked with the simple click of a button. Analysis report **PDFs** can be generated automatically, which contain comprehensive statistical plots, graphs and analysis data. Example analysis reports can be found on the accompanying CD (see Appendix **A**). Future work could address further analysis algorithms and performance optimizations. The analysis system provides an important foundation for the automated composition algorithm, which is introduced in the following chapter.

Part III

Automated Composition

Chapter 8

Evolutionary Composition Algorithm

I was obliged to be industrious.
Whoever is equally industrious will
succeed equally well.

— Johann Sebastian Bach

This chapter is concerned with the design and implementation of an algorithm capable of generating musical material. It is built on the foundations of the context-based models of musical compositions introduced in Chapters [3](#) and [4](#). Combined with the proposed evolutionary algorithm, the system can be utilized for the following applications:

- Creating variations of existing compositions
- Recombining multiple compositions (composition crossover)
- Style imitations (i.e. imitating certain musical styles or composers)
- Generating compositions with predefined constraints (e.g. instruments or harmonic progressions)
- Generating pieces with multiple sections with individual musical properties

8.1 Motivation

A number of automated composition approaches have been introduced in Chapter [2.3](#). The main motivation of the proposed system is to demonstrate that the musical quality of automated composition systems does not only depend on the employed algorithm, but also on the structure and complexity of the underlying music model.

The context-based models proposed in Chapters 3 and 4 are used as the foundation of the automated composition system. Due to the ability to produce unexpected and astonishing results while being relatively simple to retrace, EAs were selected as the basic algorithmic method for the system. The GP approach, which uses tree-based structures to represent solutions, is especially suited for the system: the evolutionary process is adapted to operate on context tree composition models (see Chapter 4). These are processed in an artificial evolutionary process, involving continuous analysis, crossbreeding, mutation and recombination of pieces.

8.2 Composition vs. Improvisation

Composition is defined as “The activity or process of creating music, and the product of such an activity. The term belongs to a large class of English nouns derived from the participial stems of Latin verbs (here *composit-*, from *componere*: ‘put together’) followed by the suffix *-io/-ionem*” (Sadie and Tyrrell 2001, p. 186).

Improvisation is a spontaneous form of composition happening in real time, and is “considered by many as an extraordinary skill, a sort of magic” (Pachet 2012, p. 119). In contrast, compositional processes in the primary sense include the recurrent refinement of the created musical material.

Both the creation and the interpretation of compositions in this restrictive sense are commonly distinguished from improvisation, in which decisive aspects of composition occur during performance. The distinction hinges on what performers are expected to do in various situations and on how they prepare themselves to meet such expectations. (Sadie and Tyrrell 2001, p. 186)

An essential characteristic of the designed system is that it is a composition system, *not* an improvisation system, implying that the algorithm does not produce a complete result in one iteration, but potentially modifies the generated results many times before yielding a final result. This approach resembles a kind of creativity which is based on trying out different possibilities and combinations:

The ‘hard work’ type of creativity often involves trying many different combinations and choosing one over the others. It seems natural to express this iterative task as a computer algorithm. The implementation issues can be reduced to two components: how to understand one’s own creative process well enough to reproduce it as an algorithm, and how to program a computer to differentiate between ‘good’ and ‘bad’ music. (Jacob 1996)

The two main issues addressed in the previous quote are discussed in the following sections and possible solution approaches are elaborated.

8.3 Composition Algorithm

The **GP**-based algorithm and its implementation is described in depth in the following sections.

8.3.1 Overview

A flow chart depicting the basic algorithm structure is shown in Figure 8.1.

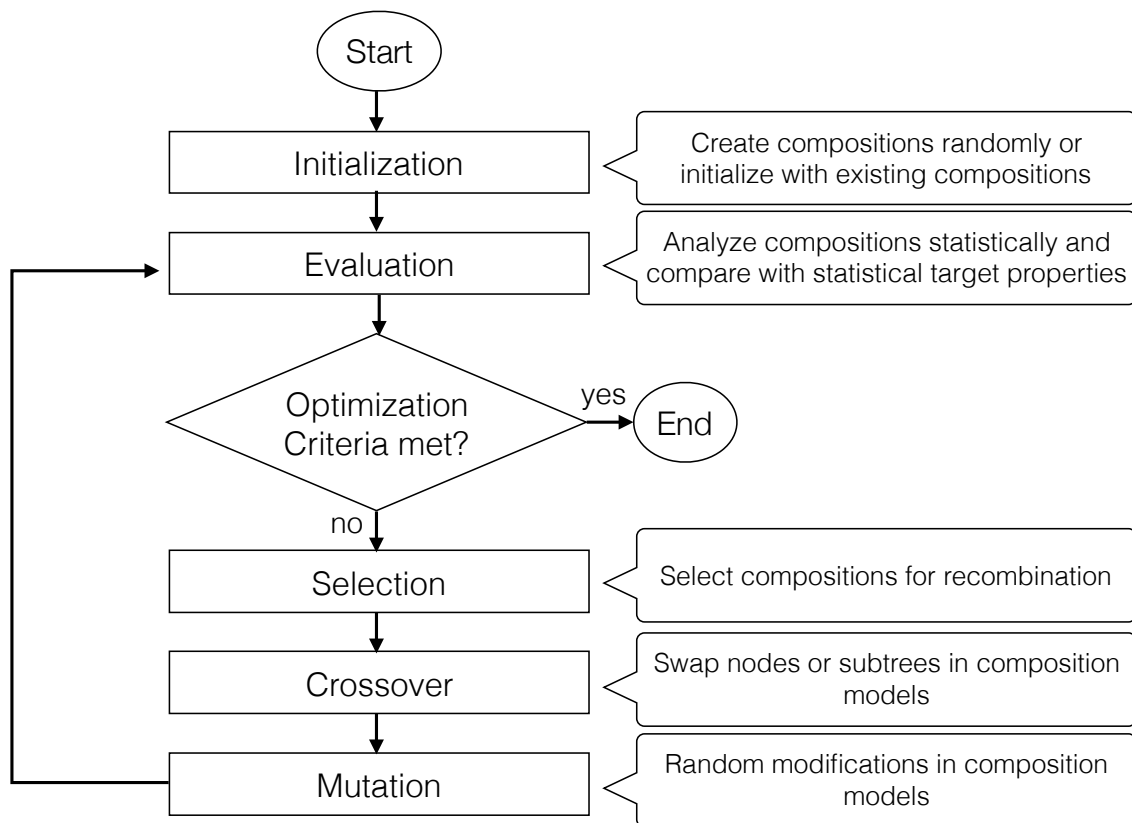


Figure 8.1: Flowchart depicting the basic structure of the evolutionary composition algorithm

The algorithm embodies a typical **EA** which was translated to the domain of music. The genotypes used in the evolutionary process are musical compositions, which are evaluated and modified according to the **GP** paradigm.

In the initialization phase, a set of initial compositions is created. Depending on the algorithm application, the initial population is either formed from existing musical material or generated randomly.

The evaluation process involves assigning ratings to each composition, which is accomplished by performing statistical analyses, which are introduced in detail in the following section.

If the optimization criteria are met, the algorithm terminates and outputs the best-rated compositions. Possible criteria are:

- The accumulated differences between the statistical distributions of a generated composition and the target distributions are below a specified threshold
- A configured maximum number of generations is exceeded
- The algorithm did not achieve improvements in terms of fitness for n generations

In case the optimization criteria are not met, the program produces a further generation of compositions. This is achieved by repeating the following procedure until the next generation reaches the configured population size:

1. Select two compositions from the current generation using roulette wheel selection (see section [2.3.7](#))
2. Create a new composition by performing a crossover between the two selected compositions (see section [8.3.4](#))
3. Perform random modifications in the resulting composition as explained in section [8.3.5](#)

8.3.2 Fitness Function

A considerable challenge in the process of designing [EAs](#) is the specification of suitable fitness functions. The purpose of fitness functions is assigning comparable ratings to individuals in the population. The evaluation of individual solutions is typically straight-forward in fields such as engineering, in which the quality of solutions can be explicitly expressed mathematically. In the case of automated music generation, the quantitative rating of music quality is a particularly difficult task:

The range of theories regarding the bases of aesthetic value, judgement and criticism is extraordinary, and the debates show no signs of near-term resolution. This presents a problem for AI scientists wishing to produce computational artists: How do we know when we've got one? How do we know if version A is better than version B, or vice versa? Without the

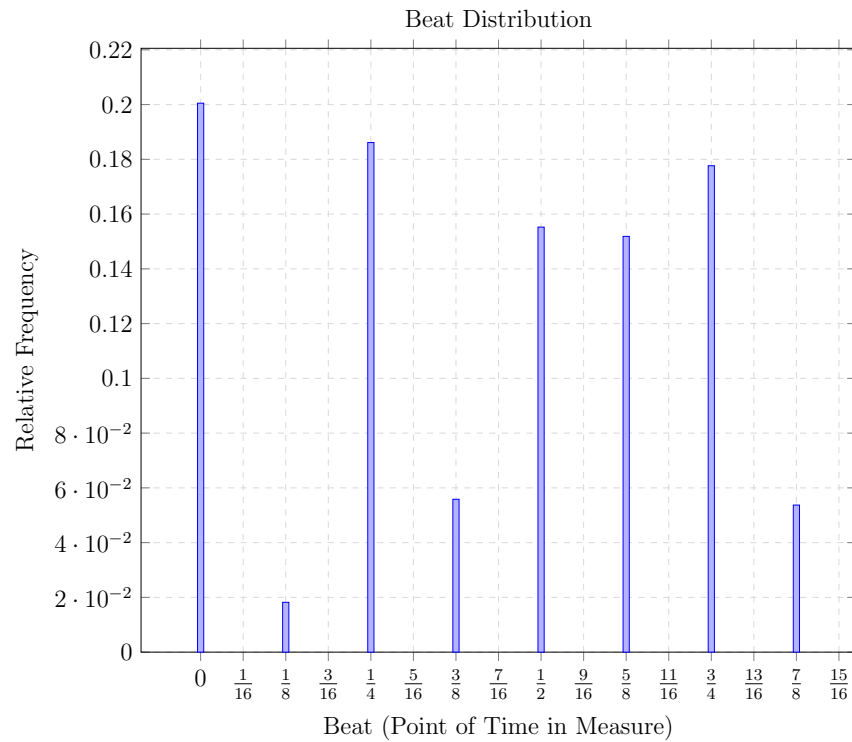
ability to answer such questions the science of artist construction cannot proceed, and these questions seem to be inseparably linked to the murky issues of aesthetic judgement. (Spector and Alpern [1994](#), p. 3)

The approach proposed in this dissertation is to use statistical analysis as the vehicle for composition evaluation. In particular, the statistical analyses introduced in Chapter [7](#) are performed for each composition generated in the evolutionary process. Each statistical distribution or feature, represented by a histogram or a real number value, respectively, is compared with a set of target distributions and values, which define the desired statistical properties of the composition to be generated. The target distributions and values can either be chosen from previously analyzed musical pieces (which effectively results in style imitations), or can in turn be generated algorithmically according to mathematical rules. The assumption is that certain statistical distributions of certain musical aspects and particular combinations of these result in pleasant musical compositions.

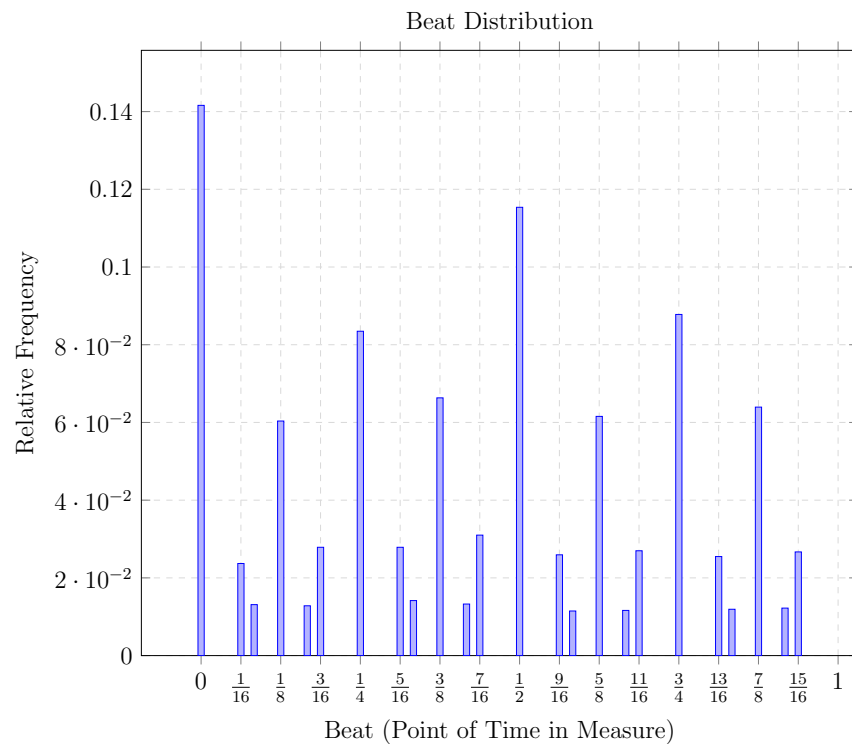
The goal of the algorithm is to minimize the difference between the actual analyzed statistical distributions and the desired target distributions. Consider the beat distributions in Figure [8.2](#) showing the relative frequency distributions of note onsets relative to the beginning of corresponding measures. Mathematically, these distributions can be considered as *histograms* of so called *probability mass functions* of discrete random variables defined as $X : S \rightarrow \mathbb{R}$, where S represents a set of all distinct observed symbols, each of which is assigned a probability (Pruim [2018](#), p. 57). The probability is obtained by computing the relative frequency of each symbol, i.e. the absolute number of occurrences of a symbol divided by the total number of occurrences. Formally, the probability mass function is a function $f : \mathbb{R} \rightarrow [0, 1]$, where $f(x) = P(X = x)$; $x \in \mathbb{R}$. The probabilities sum up to 1 by definition, as shown in Equation [8.1](#) (Pruim [2018](#), p. 57).

$$\sum_{s \in S} f(X(s)) = 1 \tag{8.1}$$

The advantage of using relative frequencies rather than absolute occurrences is that histograms can be compared efficiently, because the probabilities are normalized. This would not be possible by comparing absolute numbers. The difference between two discrete relative frequency histograms $X : A \rightarrow \mathbb{R}$ and $Y : B \rightarrow \mathbb{R}$ can be computed as shown in Equation [8.4](#), i.e. by aggregating the absolute difference between both histogram probabilities for the combined symbol set of both histograms. If an element of the combined set is not present in one of the histograms, the probability value in the respective histogram is zero. This is expressed in the functions g and h shown in Equations [8.2](#) and [8.3](#).



(a) Beat distribution histogram of *Help!* by the Beatles



(b) Beat distribution histogram of Ludwig van Beethoven's *Piano Sonata No. 21 in C major ("Waldstein"), Op. 53, Mv. I*

Figure 8.2: Comparison between two beat distribution histograms

$$g(s) = \begin{cases} f(X(s)) & \text{if } s \in A \\ 0 & \text{otherwise} \end{cases} \quad (8.2)$$

$$h(s) = \begin{cases} f(Y(s)) & \text{if } s \in B \\ 0 & \text{otherwise} \end{cases} \quad (8.3)$$

$$\sum_{s \in A \cup B} |g(s) - h(s)| \quad (8.4)$$

The goal of the algorithm is to minimize the computed histogram differences to zero. The computed value also serves as a scalar distance metric, enabling to compare compositions to each other: the smaller the distance to the optimum value zero, the better the rating of a composition in terms of fitness in the evolutionary process.

8.3.3 Multi-objective Optimization

The system was designed to support the optimization of multiple statistical aspects of music simultaneously (Hofmann 2015). In EA jargon, this is called *multi-objective optimization*. For example, not only rhythmic parameters are optimized during the evolutionary process, but also pitch-related and harmony-related structures. Technically, this is achieved by adding up the differences of all statistical comparisons and minimizing the resulting sum. Furthermore, individual fitness functions can be weighted, which makes it possible to specify the importance of the corresponding musical aspects. Equation 8.5 indicates how multiple fitness functions f_i can be weighted with the coefficients w_i and combined to a composite fitness function f (Poli et al. 2008, p. 76).

$$f = \sum_i w_i f_i \quad (8.5)$$

In the following, the music-specific fitness functions implemented in MPS are explained in detail. These are divided into two categories: fitness functions optimizing statistical feature values (shown in Table 8.1) and fitness functions optimizing statistical distributions (listed in Table 8.2).

Table 8.1: Fitness functions for statistical feature values

Fitness Function	Description
Duration	Optimizes the absolute duration of the piece.

Continued on next page

Table 8.1 – *Continued from previous page*

Fitness Function	Description
Number of Measures	Optimizes the number of measures in the piece.
Number of Voices	Optimizes the number of voices in the piece.
Number of Notes	Optimizes the total number of notes in the piece.
Pitch Range	Minimizes the number of notes which can not be played by the used instruments.
Note density	Optimizes the percentage of notes compared to the total available duration.
Chord Compliance	Optimizes the percentage of notes belonging to the current harmony.
Scale Compliance (Harmony)	Optimizes the percentage of notes belonging to the current scale relative to the current harmony.
Scale Compliance (Key)	Optimizes the percentage of notes belonging to the current scale relative to the current key.
Root Compliance (Harmony)	Optimizes the percentage of notes which represent the root note of the current harmony.
Bass Note Compliance (Harmony)	Optimizes the percentage of notes which represent the bass note of the current harmony. Note that the bass note can be different from the root note of a harmony (see section 4.5.8).
Harmonic progression length	Optimizes the number of harmonies in harmonic progressions.
Superfluous Elements	Used to remove superfluous elements in generated compositions, e.g. pitches for percussion instruments.
Tree Depth	Optimizes the tree depth of the generated context tree model representing the composition.

Table 8.2: Fitness functions for statistical distributions

Fitness Function	Description
Note Duration Distribution	Optimizes the distribution of note durations.
Rest Duration Distribution	Optimizes the distribution of rest durations.
Time Signature Distribution	Optimizes the distribution of time signatures used in the composition.

Continued on next page

Table 8.2 – *Continued from previous page*

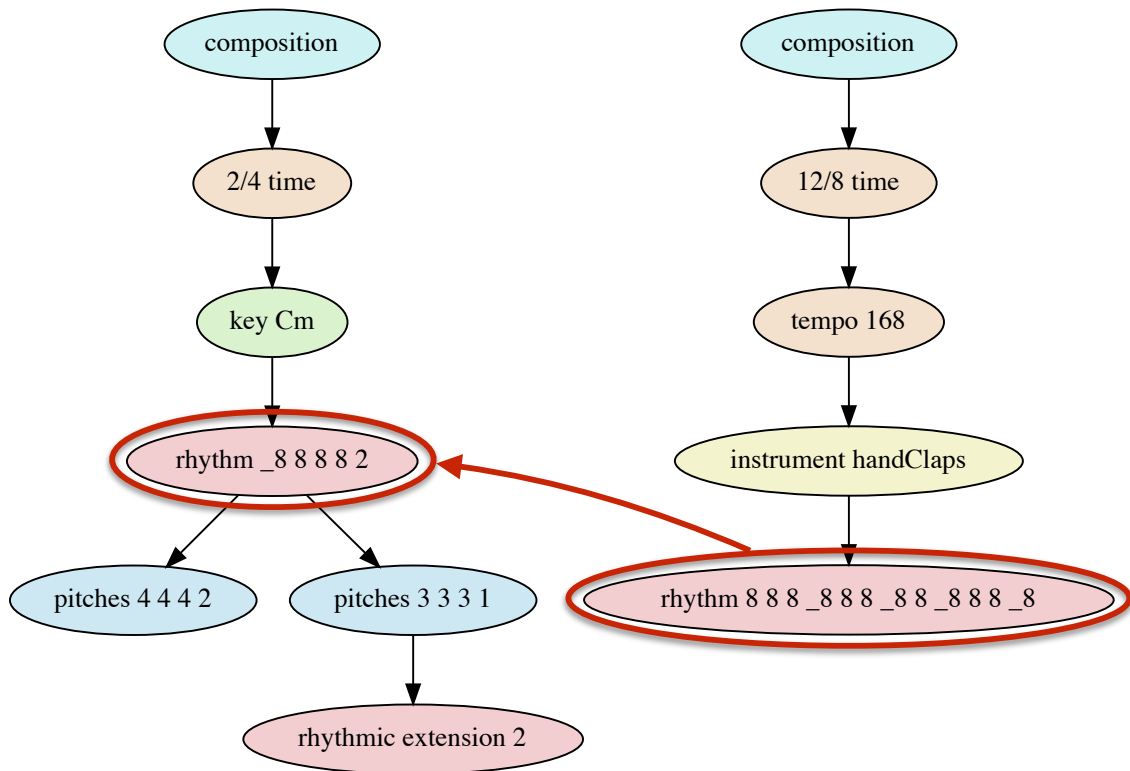
Fitness Function	Description
Beat Distribution	Optimizes the distribution of note onset times in the corresponding measures.
Dissonance Distribution	Optimizes the distribution of dissonance values. See section 7.4.4 for details.
Instrument Distribution	Optimizes the distribution of instruments used in the composition.
Simultaneous Interval Distribution	Optimizes the distribution of simultaneously played pitch intervals.
Interval Leap Distribution	Optimizes the distribution of successive pitch interval leaps.
Number of Simultaneous Notes Distribution	Optimizes the distribution of the number of notes being played simultaneously in the composition.
Circle of Fifths Distance Distribution	Optimizes the distances according to the circle of fifths of harmonic progressions.

8.3.4 Crossover Operators

As already explained in Chapter 2.3.7, the default crossover method used in GP is *subtree crossover*. It can also be applied to context tree composition models. However, subtree crossover in its basic form does not guarantee musically meaningful results. For this reason, crossover operators tailored to context tree models were developed for the proposed algorithm.

The first operator is called *node type crossover*. It analyzes which types of musical contexts (e.g. rhythms, pitches, time signatures, harmonies) are available in the two parent compositions. A random type is selected, and the candidates of the selected type are filtered from both parents. After selecting the crossover points in each parent compositions (by selecting a random candidate of the chosen type in each parent), the node in the first composition is replaced by the node in the second composition. The existing child nodes of the insertion point node are kept in place during this operation. An example of a node type crossover of two musical compositions is shown in Figure 8.3b, in which Beethoven’s Symphony 5 motif is crossed with Steve Reich’s *Clapping Music*.

Another crossover operator is an extension of the previously introduced crossover technique and is called *node type and contained sequence crossover*. It works as follows: if the selected nodes contain musical sequences such as rhythms, pitches



(a) Node crossover of rhythms illustrated using context tree models of Beethoven's Symphony 5 motif (left) and Steve Reich's Clapping Music (right)



(b) Score resulting from the crossover shown above

Figure 8.3: Node crossover between Beethoven's Symphony 5 motif and Steve Reich's *Clapping Music*. Corresponding files are available on the accompanying CD under `Composer/NodeCrossover` (see Appendix A).

or harmonic progressions, then the sequences themselves are crossed by using a one-point crossover technique, which was already illustrated in Chapter 2.3.7. This operator effectively combines tree-based crossover from GP with sequence-based crossover from GAs.

8.3.5 Mutation Operators

The biological equivalent to genetic mutations is imitated in evolutionary algorithms through random modifications in chromosomes (see section 2.3.7). In the case of context tree models, a number of model-specific mutations were implemented, which are listed in Table 8.3.

Table 8.3: Mutation operators

Operator	Description
Add Rhythm	Inserts a new rhythm.
Add Time Signature	Inserts a new time signature.
Add Instrument	Inserts a new instrument context.
Add Loudness	Inserts a new loudness context.
Add Harmonic Progression	Inserts a new harmonic progression.
Add Harmonic Rhythm	Inserts a new harmonic rhythm.
Add Pitches	Inserts a new pitch sequence.
Add Scale	Inserts a new explicit scale context.
Add Key	Inserts a new explicit key context.
Add Rhythm and Pitches	Inserts a new rhythm combined with a new pitch sequence.
Add Fragment	Inserts a new fragment.
Add Fragment Reference	Inserts a reference to an existing fragment.
Add Repetition	Inserts a new repetition control structure.
Add Parallelization	Inserts a new parallelization control structure.
Add Transposition	Inserts a new transposition modifier.
Add Rhythmic Displacement	Inserts a new rhythmic displacement modifier.
Add Rhythmic Insertion	Inserts a new rhythmic insertion modifier.
Add Chord Generator	Inserts a new chord generator.
Add Arpeggio Generator	Inserts a new arpeggio generator.
Replaces Note Index Sequence	Replaces the note index sequence of an arpeggio generator (see section 4.7.2).
Add Harmony	Inserts a new harmony into a harmonic progression.
Replace Harmony	Replaces a harmony in a harmonic progression.
Remove Harmony	Removes a harmony from a harmonic progression.
Add Rhythmic Note	Inserts a new note or rest into a rhythm.
Modify Rhythmic Note Duration	Changes the duration of a note or rest in a rhythm.
Replace Rhythmic Note	Replaces a note or rest in a rhythm.

Continued on next page

Table 8.3 – *Continued from previous page*

Operator	Description
Swap Rhythmic Notes	Swaps two notes or rests in a rhythm.
Add Pitch	Inserts a new pitch into a pitch sequence.
Replace Pitch	Replaces a pitch in a pitch sequence.
Replace Subtree	Replaces a random subtree with a randomly generated subtree (equals to <i>subtree mutation</i> , see 2.3.7).
Replace Node	Replaces a single node with a randomly generated node.
Replace Rhythm	Replaces a rhythm with a randomly generated rhythm.
Replace Pitches	Replaces a pitch sequence with a randomly generated pitch sequence.
Replace Instrument	Replaces an instrument context with a randomly selected instrument context.
Remove Node	Removes a random node from the composition model.
Remove Subtree	Removes a random subtree from the composition model.
Swap Children	Swap the position of two child nodes relative to the common parent node.
Move Subtree	Moves a subtree to a random position in the tree.
Move Node	Moves a single node to a random position in the tree.

8.3.6 Parameters

The evolutionary composition process can be adjusted using the parameters listed in Table [8.4](#).

Table 8.4: Parameters for the evolutionary composition process

Parameter	Description
Number of Generations	Maximum number of generations after which to stop the algorithm, even if no optimal solution was found.
Population Size	Number of compositions in each generation.
Number of Offspring	Number of offspring to produce before forming a new generation (defaults to population size).
Number of Elitism Individuals	Number of the best n individuals which are copied to the next generation without modification.
Crossover Rate	Probability for performing crossover.
Mutation Rate	Probability for performing mutation.
Fitness Threshold	The algorithm terminates if the specified fitness value is reached.

8.3.7 Genetic Programming Specifics

The proposed algorithm is based on the [GP](#) paradigm, which operates on tree-based structures. Compared to traditional [GAs](#), some specifics have to be considered. First, the commonly applied selection algorithm is *tournament selection*, which involves selecting a specified number of distinct individuals in the population in order to select the best-rated one. The number of randomly selected individuals t is also referred to as *tournament size*. A popular setting is $t = 2$, which results in low selective pressure, because relatively low-rated individuals can still win a tournament. Another popular setting for [GP](#) systems is $t = 7$ (Luke [2013](#), p. 48). As t increases, the probability of low-rated individuals winning a tournament decreases, because the likelihood of higher-rated compositions in the tournament increases. On the one hand, this results in a higher selection pressure which causes the algorithm to converge more quickly. On the other hand, higher selection pressure quickly reduces the diversity of the population, which can lead to an inadequate exploration of the search space. For the proposed system, experiments showed that lower selection pressure ($t = 2$) is preferable.

Another specific aspect of [GP](#) is the composition of crossover and mutation operators. Conventional [EAs](#) usually apply mutation operators directly after crossover operators. In contrast, the application of crossover and mutation operators is typ-

ically mutually exclusive in GP systems (Poli et al. 2008, p. 17). The probability of employing either one of the operators is defined by the *crossover rate* P_c and *mutation rate* P_m , respectively. In some GP scenarios, mutation is not even required, because subtree crossover inherently has “mutative” properties, resulting in crossover rates near 100% (Luke 2013, p. 48).

Furthermore, another operator is introduced: *reproduction* involves inserting a copy of a selected individual into the next generation without modification. The probability of reproduction P_r is defined by the difference between the combined crossover and mutation rate to 100%, as shown in Equation 8.6 (Poli et al. 2008, p. 17).

$$P_c + P_m + P_r = 1 \Leftrightarrow P_r = 1 - P_c - P_m \quad (8.6)$$

8.3.8 Example Evolutionary Algorithm Run

The functionality of the evolutionary algorithm is illustrated with a simple scenario. The goal of the following evolutionary process is to evolve the opening rhythm ♩ from Beethoven’s *Symphony No. 5 in C Minor, Op. 67*, using statistical distributions. The fitness function is composed of the following objectives, which can be obtained using the analysis tools introduced in Chapter 7:

- Duration: 2 measures
- Number of voices: 1
- Note density: $\frac{7}{8}$
- Note duration distribution: 75% eighth notes, 25% half notes
- Time signature distribution: 100% $\frac{2}{4}$ time
- Beat distribution: first beat in the measure: 25%, second eighth beat: 25%, third eighth beat: 25%, last eighth beat: 25%

For this simple example, a minimal evolutionary algorithm configuration was used which is shown in table 8.5. A graphical illustration of the evolutionary process is shown in Figure 8.4.

Table 8.5: Parameters for the evolutionary composition process used for the run depicted in Figure 8.4

Parameter	Value
Number of Generations	10
Population size	10
Number of Offspring	9
Number of Elitism Individuals	1
Crossover Rate	0.5
Mutation Rate	0.4
Fitness Threshold	0

In the evolutionary algorithm run depicted in Figure 8.4, an optimal solution representing the rhythm of the motif was found four generations after the randomly generated initial population, shown as *Generation 0*. Using the depicted graph, which can be considered a type of family tree, the origins of specific solutions can be retraced. The found optimal solution, which is marked with a double circle in the graph, can be traced back to the second individual with rating 0.53 in the initial population. In this instance of the run, the final solution was found as follows:

1. The initial population was generated with random individuals. The second composition with the rating 0.53 contained the rhythm ♪♪♪♪.
2. A mutation operator which swaps notes or rest was applied, yielding the rhythm ♪♪♪♪ in generation 1, which also has the rating 0.53 according to the fitness function.
3. Another mutation operator was applied, which turns a note into a rest or vice versa. In this case, the last eighth note was selected and converted into a rest, resulting in the rhythm ♪♪♪♪. This became the best-rated individual in generation 2 with a rating of 0.5.
4. Due to elitism, the best-rated composition was directly copied to the next generation. Furthermore, the same individual was selected for reproduction, which resulted in two copies of the rhythm in the third generation.
5. Another mutation operator moved the eighth rest to another position in the rhythm, which resulted in ♪♪♪♪, which is exactly the rhythm of Beethoven's famous motif.

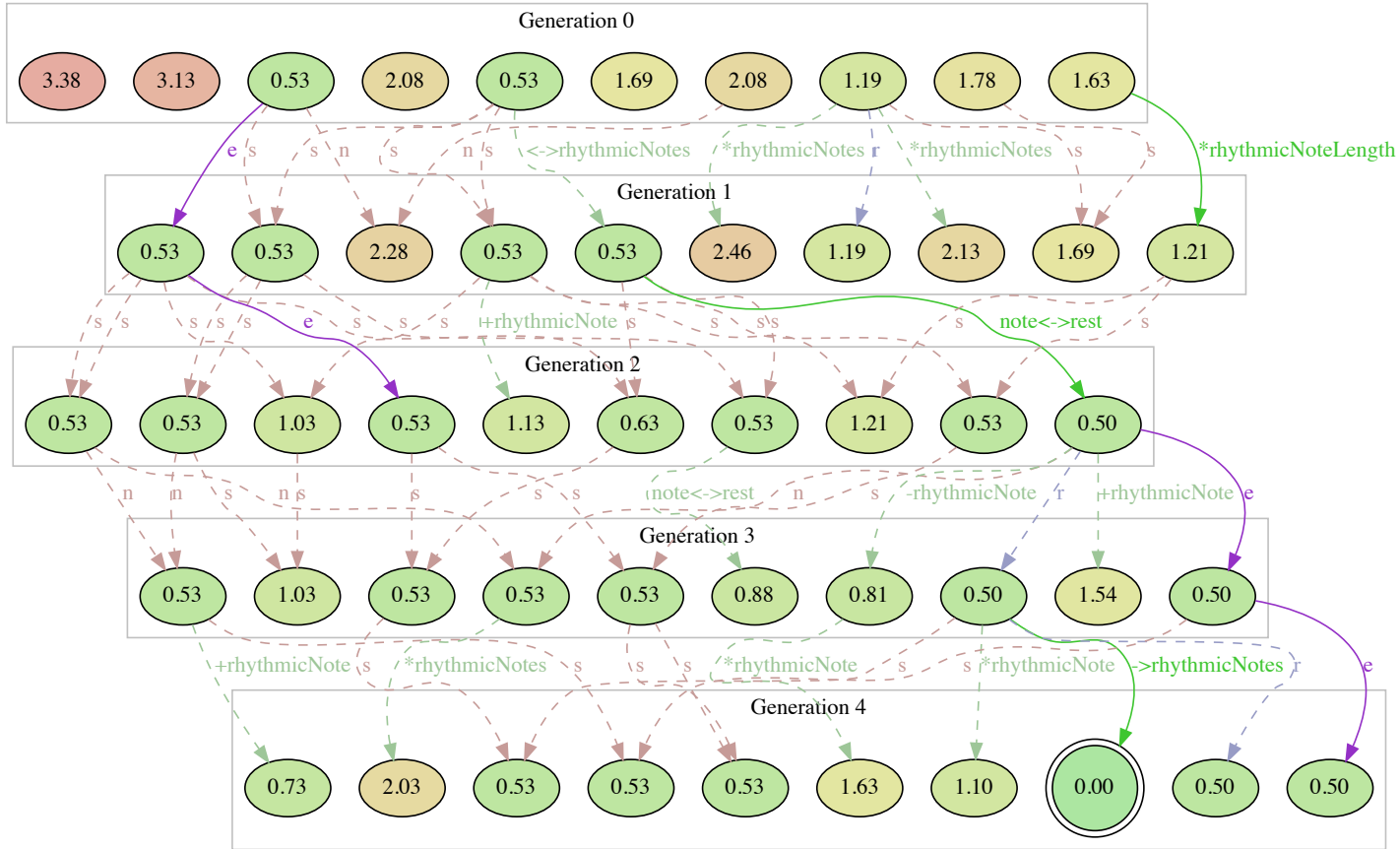


Figure 8.4: Graph illustrating a small-scale evolutionary process with the objective to reconstruct the rhythm $\text{♩} \text{♩} \text{♩} \text{♩}$ from the motif of Beethoven's *Symphony No. 5 in C Minor, Op. 67*. The individual solutions are labeled and colored according to their aggregated fitness function rating, where lower ratings are considered better because the fitness function is minimized. The process found an optimal solution (indicated by the double-circled node with rating 0.00) satisfying all statistical constraints after 4 generations. The graph visualizes which genetic operators were used: **e** $\hat{=}$ direct elite insertion, **r** $\hat{=}$ reproduction, **s** $\hat{=}$ subtree crossover, **n** $\hat{=}$ node crossover, **+rhythmicNote** $\hat{=}$ add note or rest, **+rhythmicNotes** $\hat{=}$ add multiple notes or rests, **-rhythmicNote** $\hat{=}$ remove note or rest, **-rhythmicNotes** $\hat{=}$ remove multiple notes or rests, ***rhythmicNote** $\hat{=}$ replace note or rest, ***rhythmicNotes** $\hat{=}$ replace multiple notes or rests, ***rhythmicNoteLength** $\hat{=}$ change note or rest length, **note<->rest** $\hat{=}$ turn note into rest and vice versa, **->rhythmicNote** $\hat{=}$ move note or rest, **->rhythmicNotes** $\hat{=}$ move multiple notes or rests, **<->rhythmicNotes** $\hat{=}$ swap notes or rests

Note that in this example crossover operators were not capable of improving the fitness ratings significantly, because the tree structures used in this example are very simple and therefore the number of possible insertion points is limited. In this case, subtree crossovers either result in nested structures which are too complicated for the simple optimization problem presented, or cause a complete replacement of the tree structure in the target model.

8.4 Applications

In the following sections, larger-scale applications of the proposed algorithm are demonstrated. These include recombining existing compositions, generating style imitations and composing music using musical and statistical constraints.

8.4.1 Composition Crossover and Variations

Composition crossover involves recombining the musical material of one or multiple input compositions to one resulting composition. If only one input composition is supplied, the result is effectively a variation of the given piece. Otherwise, an arbitrary number of pieces can be recombined to a new composition.

Motivation

To demonstrate the potentials of composition crossover, a manually compiled model in which eleven compositions were combined is presented in Figure [8.5¹](#). Elements of the following compositions are present in this model:

- Harmonic progression from *Stairway to Heaven* by Led Zeppelin
- Harmonic rhythm from *Piano Sonata No. 16 in C major, K. 545* by W. A. Mozart
- Trumpet Instrumentation from *Penny Lane* by the Beatles
- Augmented rhythm from *Golliwogg's Cake Walk* by Claude Debussy
- Pitch degrees from *Don't Stop Me Now* by Queen
- Arpeggio pattern from *Prelude in C Major, BWV 846* by J. S. Bach
- Bass rhythm from Herbie Hancock's *Chameleon*

¹An audio sample resulting from this model is available on the accompanying CD under `Audio/Crossover/Crossover.mp3`

- Bass pitch pattern of *Hey Jude* by the Beatles
- First measure of Beethoven's *Symphony No. 5 in C Minor, Op. 67* motif
- Hi-hat and bass drum pattern from *One* by Metallica
- Snare Drum pattern from *With Arms Wide Open* by Creed

The model in Figure [8.5](#) illustrates powerful possibilities of the context-based model. By selecting and recombining individual musical aspects of compositions, which happens on fine-grained musical levels (and not on the basis of notes and rests), structurally complex and musically coherent results can be produced with high likelihood. Due to the context-dependent interpretation of all musical aspects, the model structure inherently ensures a certain musical plausibility.

Implementation

In order to cross a given set of input compositions, the evolutionary algorithm is run with the following configuration:

- The initial generation is populated with copies of the input compositions. Each node in each composition is tagged with an identifier corresponding to the respective source file.
- The crossover probability is set to a high value (i.e. slightly less than or equal to 100%), whereas the mutation probability is set to zero.
- The goal of the algorithm is to perform crossover operations until the following criteria are met as closely as possible:
 - The number of measures of the resulting composition is equal to a pre-defined number.
 - Composition source tags are distributed equally (e.g. when crossing two compositions, tags should be distributed fifty-fifty, for three compositions each should have a ratio of $\frac{1}{3}$, etc.).
 - The overall note density of the resulting compositions is equal to the average note density of the given input compositions.

This algorithm configuration can also be used to generate variations of a given composition. In this special case, the initial population is filled with copies of one single piece. Consequently, the composition will be crossed with itself throughout the evolutionary process.

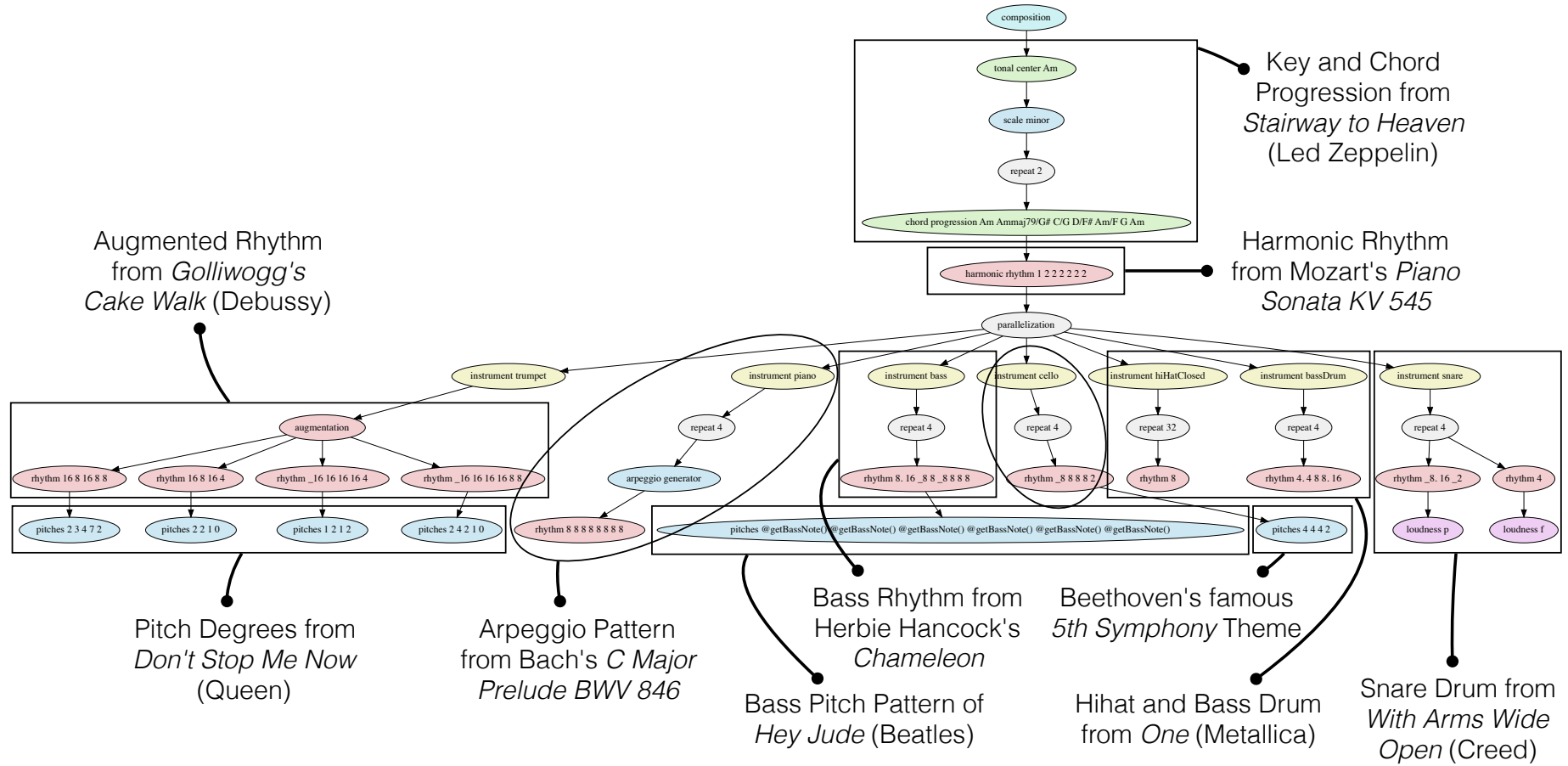


Figure 8.5: Manually compiled model recombining musical fragments of eleven different compositions. This model demonstrates both the expressiveness of the proposed model and the potential of algorithmic recombination of musical compositions by putting individual aspects of pieces into new musical contexts. Corresponding files are available on the accompanying CD under **Composer/ManualCrossover** (see Appendix [A](#)).

Graphical User Interface

MPS provides a user interface to configure crossover runs. It is presented in Figure 8.6. It allows users to configure input compositions, a destination folder, the desired number of measures of the output composition and parameters for the evolutionary algorithm.

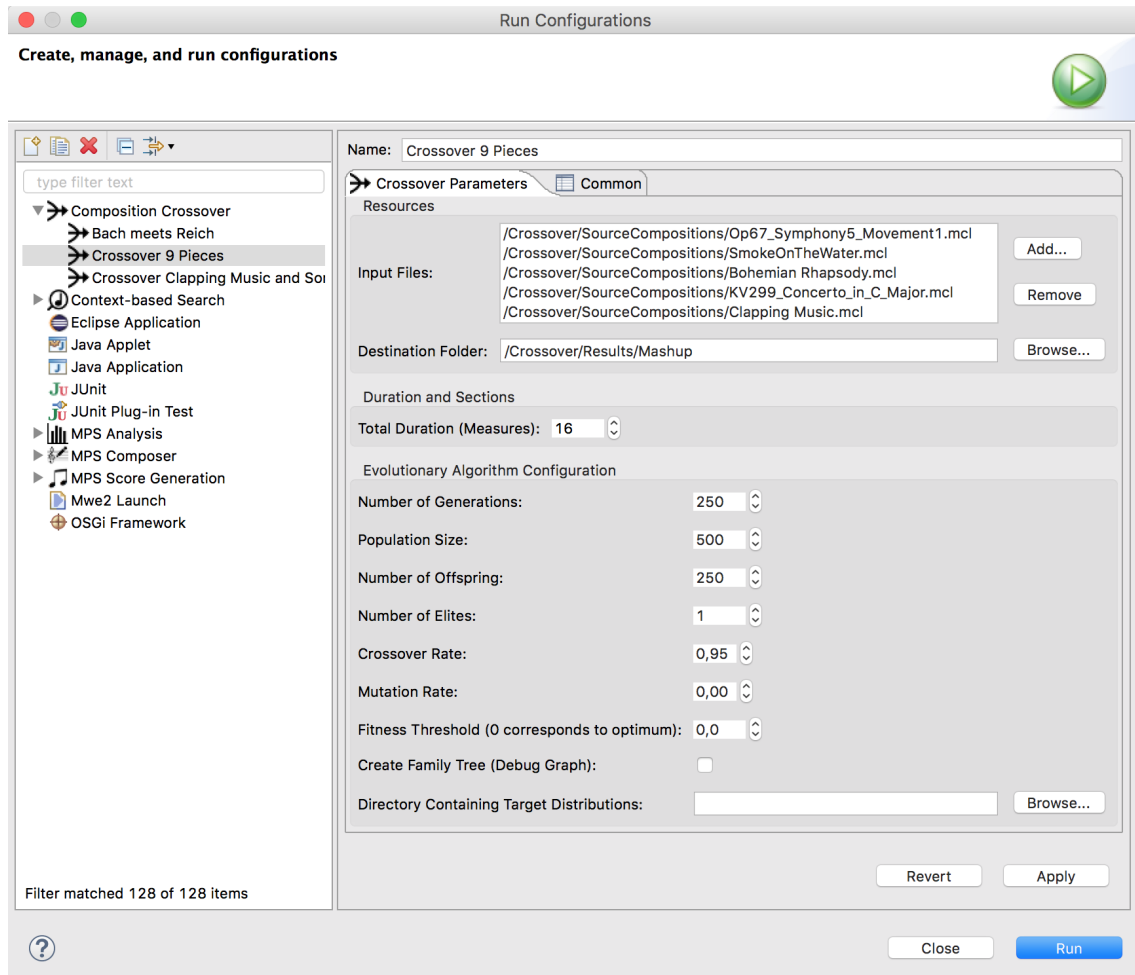


Figure 8.6: User interface to configure the evolutionary algorithm for composition crossover. At the top, input compositions and a destination directory are specified. Below that, the desired number of measures of the crossed composition is specified. Evolutionary algorithm parameters can be set in the lower section. It is also possible to supply custom statistical target distributions in CSV format, which are supplied in a separate target distribution folder.

Results

A composition model generated by the evolutionary algorithm only by means of crossover operations is presented in Figure 8.7. The following nine composition models were given as input:

- J. S. Bach, *Prelude in C Major*, BWV 846, mm. 1–4
- The Beatles, *Hey Jude*
- Ludwig van Beethoven, *Symphony No. 5 in C Minor*, Op. 67, Mv. I, mm. 1–21
- Ludwig van Beethoven, *Piano Sonata No. 14 in C# minor*, Mv. I, mm. 1–4
- Deep Purple, *Smoke on the Water*, main guitar riff
- Wolfgang Amadeus Mozart, *Flute and Harp Concerto in C major*, K. 299/297c, opening theme
- Wolfgang Amadeus Mozart, *Piano Sonata No. 16 in C major*, K. 545, mm. 1–8
- Queen, *Bohemian Rhapsody*, vocal introduction
- Steve Reich, *Clapping Music*



Figure 8.7: Context tree model of a composition generated by the evolutionary algorithm by recombining existing compositions. It was produced by applying only crossover operations starting from seven existing musical pieces. The resulting score is shown in Figure [8.8](#).

The target fitness functions listed in Table [8.6](#) were specified. The algorithm produced a combined composition with 8 measures after 250 generations. Only seven



Figure 8.8: Score of a composition generated by the evolutionary algorithm through recombining existing compositions. It was generated by applying only crossover operations starting from seven existing musical pieces. The corresponding model is shown in Figure 8.7. Corresponding files are available on the accompanying CD under `Composer/AutomatedCrossover` (see Appendix A).

of the nine given composition are a part of the final result, the distribution of which is listed in Table 8.7

Table 8.6: Fitness functions for the evolutionary crossover process

Fitness Function	Weight	Target Value	Actual Value	Distance
Number of Measures	1	8	8	0
Note Density	10	0.9	0.82	0.78
Chord Compliance	2	0.75	0.94	0.38
Number of Voices	5	4	3	5
Superfluous Model Elements	1	0	0	0

Continued on next page

Table 8.6 – Continued from previous page

Fitness Function	Weight	Target Value	Actual Value	Distance
Source Tag Distribution	10	Equal ratio of $\frac{1}{9}$ for each input composition	see Table 8.7	8.03
Total Distance		0	14.19	14.19

Table 8.7: Ratios of origin composition elements in the crossover composition

Composition	Percentage
J. S. Bach, <i>Prelude in C Major, BWV 846</i>	41.0%
The Beatles, <i>Hey Jude</i>	15.4%
Ludwig van Beethoven, <i>Piano Sonata No. 14 in C# minor</i>	15.4%
Queen, <i>Bohemian Rhapsody</i>	12.8%
Wolfgang Amadeus Mozart, <i>Piano Sonata No. 16 in C major, K. 545</i>	7.7%
Ludwig van Beethoven, <i>Symphony No. 5 in C Minor, Op. 67</i>	5.1%
Deep Purple, <i>Smoke on the Water</i>	2.6%

The resulting piece is mainly reminiscent of Bach’s *Prelude in C Major, BWV 846*, since it is based on its arpeggio structure, which is dependent on a changing harmonic context. This also aligns with the distribution of source compositions shown in Table 8.7, which shows that 41% of the piece originate from Bach’s prelude. Table 8.7 also reveals that the algorithm was not able to construct a context tree model containing the same amount of musical material from all nine input compositions, at least not in 250 generations. Overall, the resulting composition can be considered ‘creative’, demonstrating that at least some aspects of combinational creativity (see Chapter 2.2.3) can be simulated by means of EAs.

8.4.2 Style Imitations

Another possible application of the evolutionary algorithm is generating musical material imitating a specific musical styles.

Implementation

Style imitations are produced as follows:

1. An existing piece or a collection of pieces is analyzed statistically as demonstrated in Chapter [7](#)
2. A desired subset of the analysis results is selected
3. The selected analysis results are used as target values and/or distributions for the evolutionary algorithm
4. The evolutionary algorithm is utilized to produce compositions with similar or equal statistical properties as the analyzed music

Results

The algorithm is capable of generating short musical phrases resembling the style of pre-analyzed pieces. As an example, a style imitation of the first four measures of W. A. Mozart's *Piano Sonata No. 16 in C major, K. 545* is generated. The methodology is presented in Figure [8.9](#).

In the first step, both voices in the input are analyzed separately (for details about analysis scopes, refer to section [7.2](#)). Subsequently, a subset of desired statistical features is selected. In the example, the following features were chosen:

- Note density
- Harmony compliance (the percentage of notes which belong to the current harmonic context, which is notated above the score)
- Note duration distribution
- Beat distribution (note onset times relative to the beginning of a measure)
- Interval leap distribution (relative distance of successive pitches)

The selected features are used as input for the evolutionary algorithm in order to construct a fitness function. The goal of the optimization process is to minimize the differences between the actual distributions (which are analyzed for each generated composition) and the given distributions. The harmonic progression shown in [8.9](#) was predefined, as described in detail in section [8.4.3](#). One of the generated results complying to the given features is presented in Figure [8.10](#).

The generated melody is reminiscent of the original right hand motif by Mozart. As regards the left hand, the algorithm generated a nearly identical accompaniment.

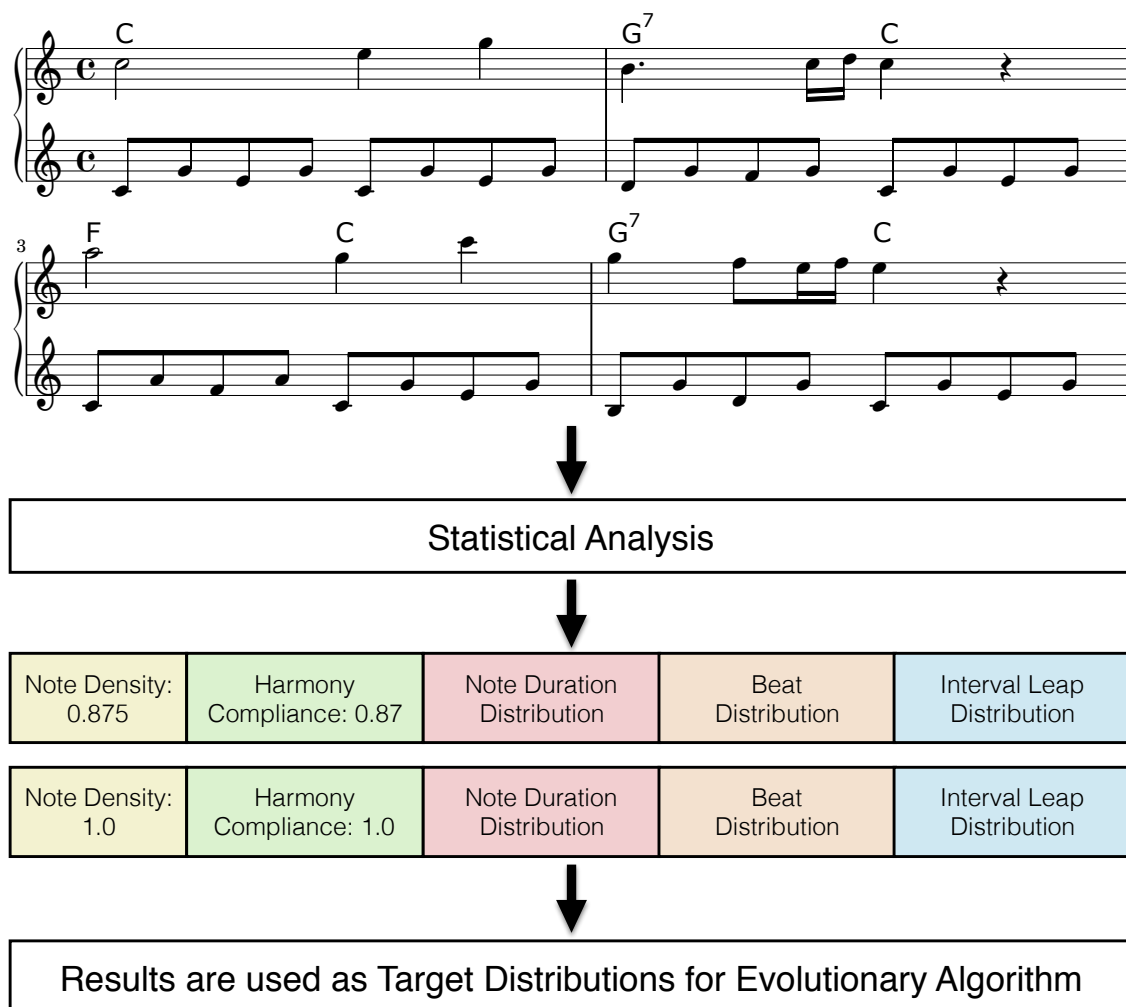


Figure 8.9: Methodology applied to generate style imitations. The analyzed example is W. A. Mozart's *Piano Sonata No. 16 in C major, K. 545*, mm. 1–4. Corresponding files are available on the accompanying CD under `Examples/Compositions/Mozart/KV545_SonataFacile` (see Appendix [A](#)).



Figure 8.10: Style imitation of W. A. Mozart, *Piano Sonata No. 16 in C major, K. 545*, mm. 1–4, generated by the evolutionary algorithm. Corresponding files can be found on the accompanying CD under `Composer/StyleImitation` (see Appendix [A](#)).

The only difference is the beginning of the last measure, in which Mozart used B and D as lower notes, whereas the algorithm used D and F. The latter is indeed 100% harmony compliant, however the chord generated by the algorithm does not fully embody the dominant seventh chord intended by Mozart. Overall, the generated piece is a plausible style imitation of the given input composition.

8.4.3 Generating Compositions with Predefined Structures

```

1 composition final
2 {
3   key C
4   {
5     repeat 2
6     {
7       harmonicProgression I7 IV7 I7 IV7 I7 V7 IV7 I7 V7
8       {
9         harmonicRhythm 1 1 2! 2! 2! 1 1 1 1
10        {
11          parallel
12          {
13            scale blues
14            {
15              fragmentRef lead
16            }
17            fragmentRef accompaniment
18          }
19        }
20      }
21    }
22  }
23 }
24
25 fragment lead fixed
26 fragment accompaniment fixed

```

Listing 8.1: Template composition model specifying **fixed** nodes and **final** subtrees which are not modified by the evolutionary algorithm

Under certain conditions it is eligible to predefine structures of generated compositions. Examples for these are sections of pieces (such as verse or chorus), repetitions, scales to be used or harmonic constraints such as keys and harmonic progressions. In the following example, a blues composition is generated by the evolutionary algorithm. For that purpose, a typical harmonic blues progression is predefined. This is accomplished by providing a template composition model, which can be specified using the composition language introduced in Chapter 4. Two additional keywords are utilized to instruct the algorithm that certain nodes in the composition model must not be removed and/or modified during the evolutionary process: the keyword **fixed** indicates that child nodes can be appended, but the node must not be removed. When specifying a **final** node, the complete subtree starting at the corresponding node must not be removed and no child nodes can be appended.

Refer to Listing 8.1, in which a template composition model for a blues is demonstrated. The corresponding context tree model is depicted in Figure 8.11.

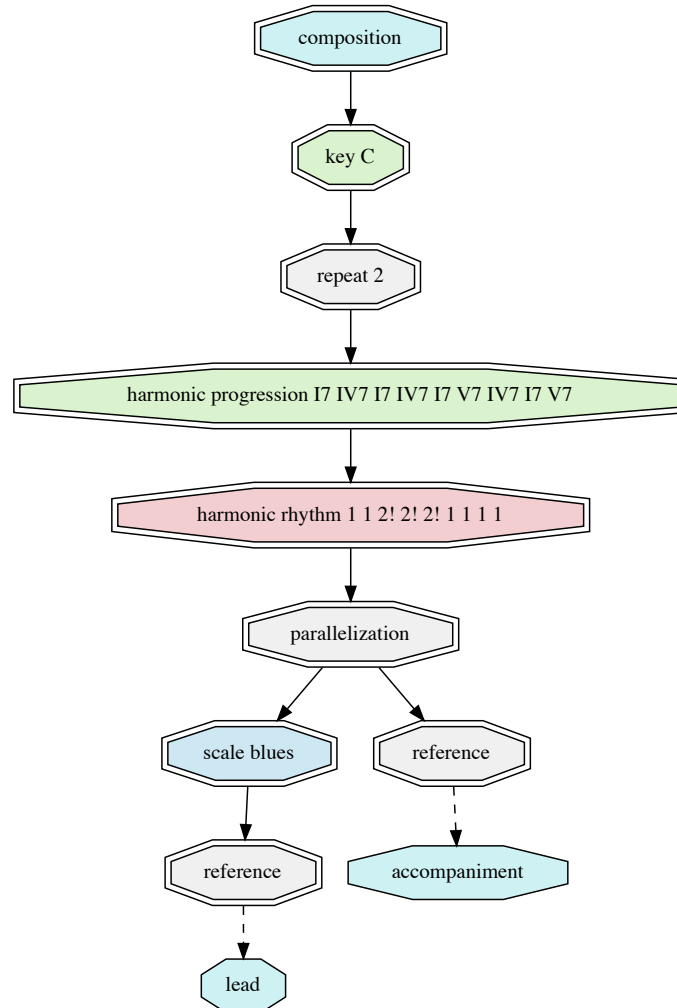


Figure 8.11: Context tree model of a template composition for a blues corresponding to the language representation in Listing 8.1. Fixed nodes are surrounded by octagons and final subtrees are marked with double octagons.

Implementation

If a template model is given as input, the initial generation is populated with complemented copies of the given template model. During the evolutionary process, the **fixed** and **final** annotations in the model are considered whenever crossover or mutation operators are applied. In particular, crossover point selection disregards all fixed and final nodes. With regard to mutation, no mutations are permitted for final nodes and mutation actions for fixed nodes are limited to child node additions.

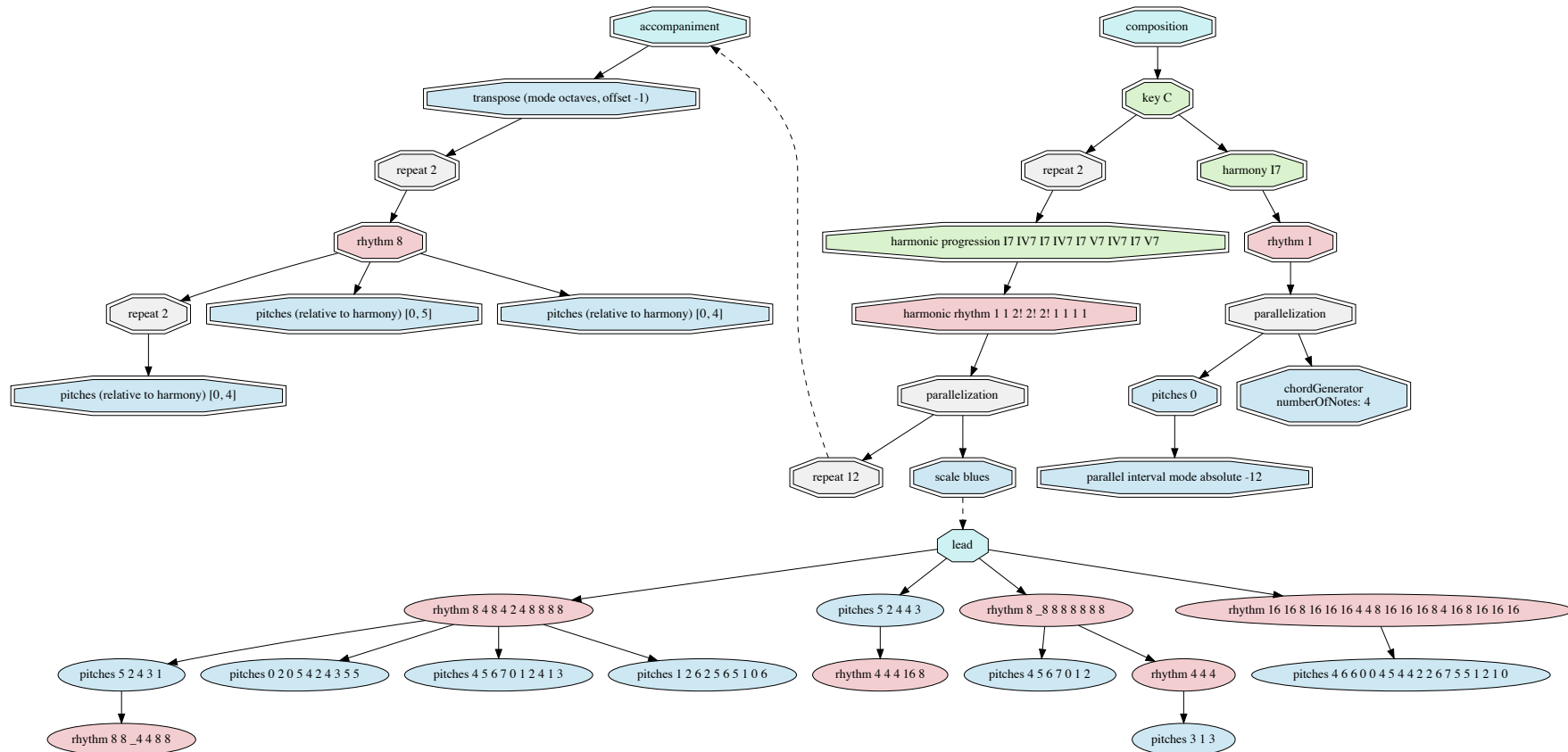


Figure 8.12: Context tree model of a blues composition generated by the evolutionary algorithm. Nodes surrounded by octagons were predefined. The subtree below the **lead** node was generated by the evolutionary algorithm. Corresponding files are available on the accompanying CD under **Composer/Blues** (see Appendix [A](#)).

Results

An extended template composition model based on the previously presented model was used to generate a blues composition. In comparison to the presented model, a simple piano accompaniment and a final chord was added. The complete resulting model containing both predefined and generated nodes is presented in Figure 8.12. The corresponding score is shown in Figure 8.13.

The program also generated a blues composition with four parts, namely a lead voice, piano, bass and drums. The corresponding model and audio files are available on the accompanying CD under `Composer/BluesFourParts` (also see Appendix A).



Figure 8.13: Blues composition generated by the evolutionary algorithm resulting from the context tree model in figure 8.12. The accompaniment in the left hand and the final chord were predefined. The melody in the right hand part was generated by the algorithm. Corresponding files can be found on the accompanying CD under `Composer/Blues` (see Appendix A).

8.4.4 Generating Compositions with Multiple Sections

Using the methodologies presented so far, the algorithm generates musical material that retains certain statistical properties, hence lacking in desirable musical developments. Against the background of identifying individual sections with different statistical attributes, these can be identified when analyzing human compositions. This is illustrated in Figure 8.14.

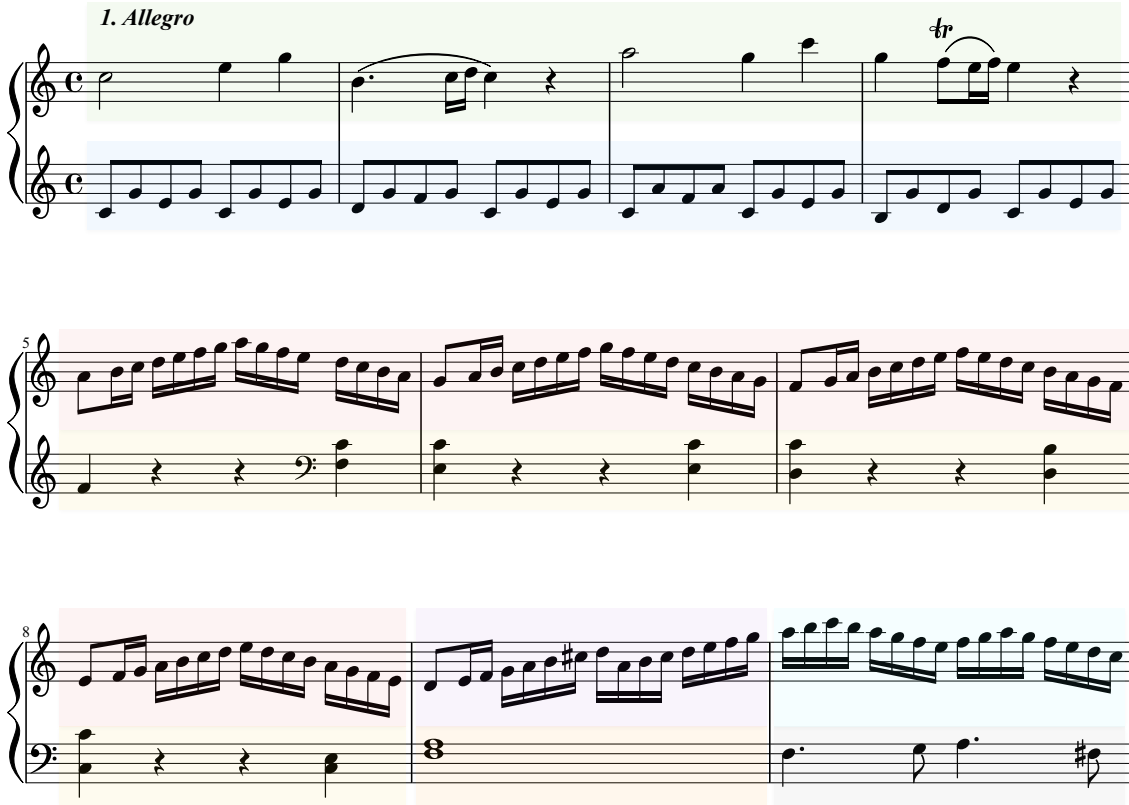


Figure 8.14: Sections with individual statistical attributes in W. A. Mozart’s *Piano Sonata No. 16 in C major, K. 545*. Score edited by RSB and made available at imslp.org under the [Creative Commons Attribution 3.0 Unported License](https://creativecommons.org/licenses/by/3.0/).

An approach for generating compositions with multiple sections, which was developed in the scope of this dissertation, is the division of the fitness function into sections and voices, to which individual statistical target distributions can be assigned. The developed data structure is illustrated in Figure 8.15.

Global Target Features			
	Section 1	Section 2	...
	Section-specific Target Features	Section-specific Target Features	...
Voice 1	Voice-specific Target Features	Voice-specific Target Features	...
Voice 2	Voice-specific Target Features	Voice-specific Target Features	...
...

Figure 8.15: Fitness function defining statistical target features on three different hierarchy levels

The data structure contains statistical target features on three different hierarchy levels:

1. Global target features valid for the whole composition
2. Section-specific target features
3. Voice-specific target features which can be assigned to each section in each voice.

Using this data structure, musical developments in the generated compositions are possible, since consecutive sections can have different statistical properties. Furthermore, the development of polyphonic music is possible, since concurrent voices can have different characteristics as well.

Graphical User Interface

The graphical user interface for configuring the evolutionary algorithm for section-wise composition is depicted in Figure 8.16. Compared to the interface in Figure 8.6, the dialog contains additional settings for specifying the number of sections and voices to be composed and an optional list of instruments to be used.

The statistical target features and distributions are supplied in the form of CSV files. These have to be stored in a predefined folder structure, which is shown in Figure 8.17. Depending on the location of the CSV files, the scope of the corresponding fitness function is determined. A fitness function structure according to Figure 8.15 is constructed from the given files.

Implementation

In a first attempt, an EA optimizing all statistical target features in all voices and sections in a large composition model was implemented. This implementation was not productive due to the fact that the optimization process in most cases was not able to reach optimal solutions in acceptable time. The problem is rooted in the temporal semantics of the context tree composition model. In large trees, small local changes can have large global impacts. For example, if a note is added or removed at the very beginning of a composition, all following notes are shifted, causing previously computed onset positions relative to measure beginnings and simultaneously audible note combinations to become invalid. Consequently, these have to be optimized again. In this process, music is shifted again and the algorithm gets stuck in a loop.

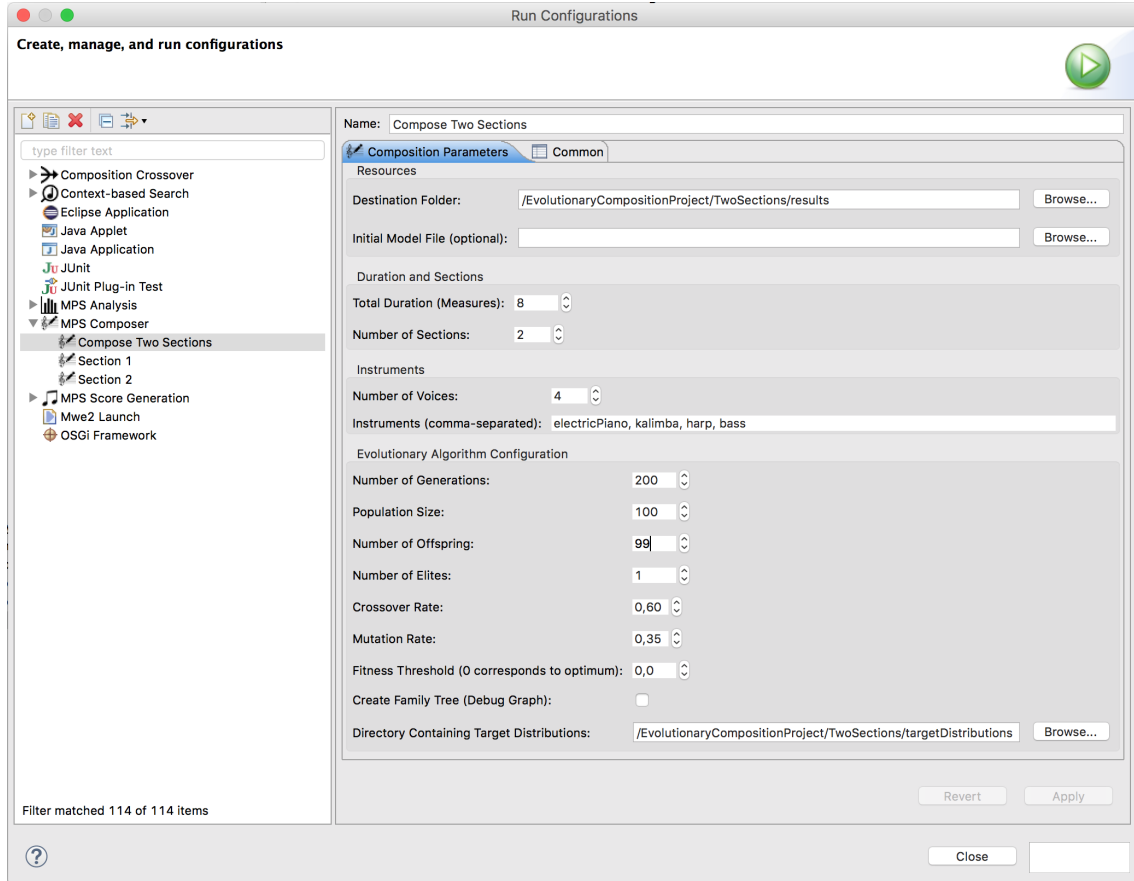


Figure 8.16: Graphical user interface for section-wise composition generation

To counteract this issue, the following strategy was implemented: Each part (i.e. each section for each voice) is independently evolved in separate evolutionary processes. The target distributions for each part are determined by combining global, section-specific and the respective voice-specific target feature sets. In case target features of the same type are present in multiple layers, the innermost target feature set is used.

The part-specific evolutionary processes are in turn split into multiple stages to generate the following musical contexts:

- Instrumentation
- Metric contexts
- Chord progressions
- Rhythms
- Pitches

In each stage, only relevant target distributions are considered, reducing the number of optimization goals for the algorithm. Furthermore, after each stage is finished,

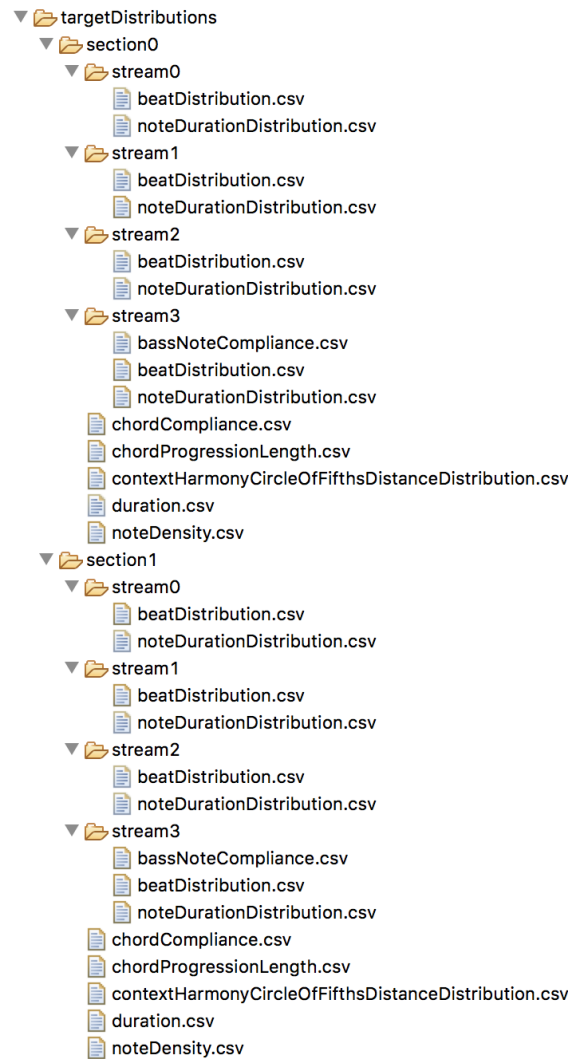


Figure 8.17: Folder structure containing CSV files for target features and distributions, from which a section-wise fitness function is built according to Figure 8.15

the results of the corresponding stage are ‘frozen’ and not modified again in later stages. This avoids the mentioned problems when optimizing all musical aspects at the same time.

Results

Using the proposed approach, the system is capable of generating polyphonic musical compositions with varying musical characteristics. This is achieved by generating multiple sections containing voices with individual statistical properties. The following result was generated by specifying arbitrary statistical target feature values (shown in Table 8.8) and statistical target distributions (shown in Figures 8.18, 8.19, 8.20, 8.21 and 8.22) as input. These target values and distributions can either be adopted from music analysis results (see Chapter 7) or be specified in an

arbitrary manner. In the latter case, the computer generates music without any prior knowledge or analysis of existing musical compositions. One possible resulting composition is shown in Figure [8.23](#).

Table 8.8: Target values and actual values of statistical features of the generated composition

Feature	Scope	Target Value	Actual Value
Chord Compliance	Section 1	1.0	0.944
Chord Compliance	Section 2	1.0	0.938
Chord Progression Length	Section 1	4	4
Chord Progression Length	Section 2	4	4
Duration	Section 1	4.0	4.0
Duration	Section 2	4.0	4.0
Note Density	Section 1	1.0	1.0
Note Density	Section 2	0.9	0.89

The resulting compositions are musically more interesting than the ones generated based on the techniques presented in the previous chapters due to the fact that musical developments and changes are possible. One thing to consider for future developments is that the transitions between the sections are sometimes abrupt, since the sections are generated independently from each other and therefore no coherence between the units is guaranteed. However, musical coherence within the individual sections can be accomplished using fitness functions concerning consonance of simultaneously audible notes and harmonic compliance values. Apart from the amendable section transitions, the program demonstrates that interesting musical output can be generated according to statistical properties.

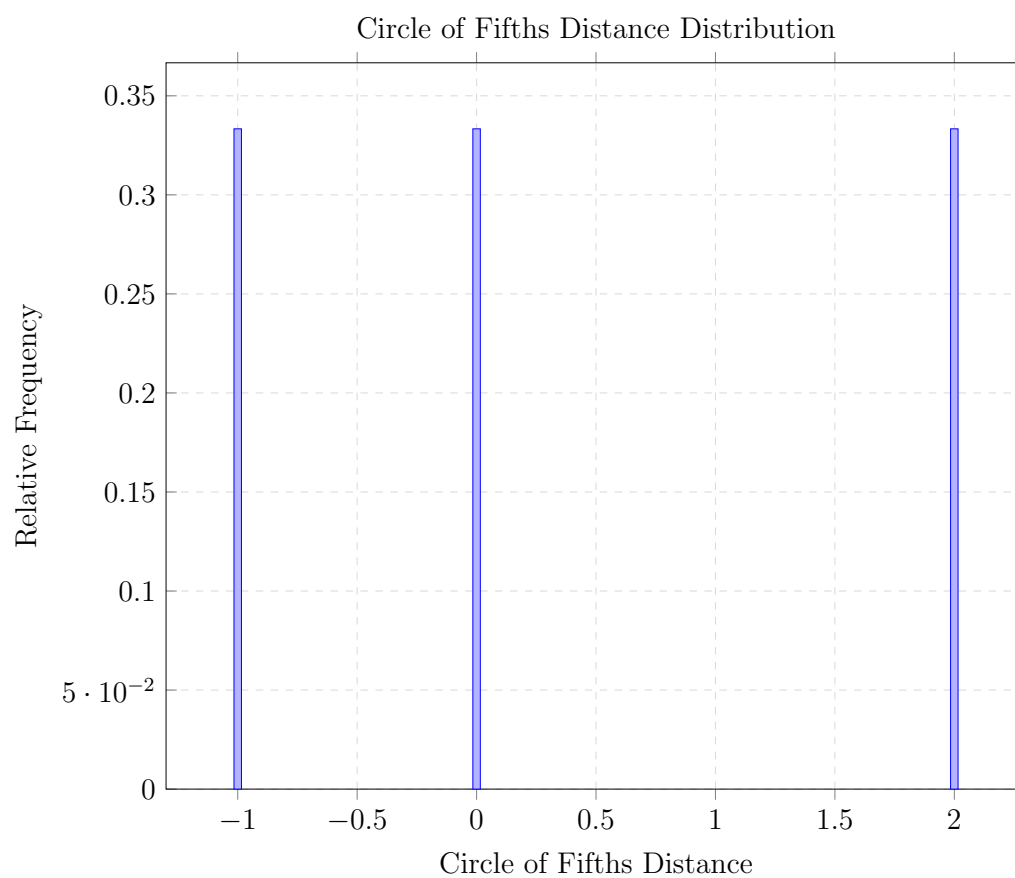


Figure 8.18: Circle of fifths distance distribution used to generate harmonic progressions

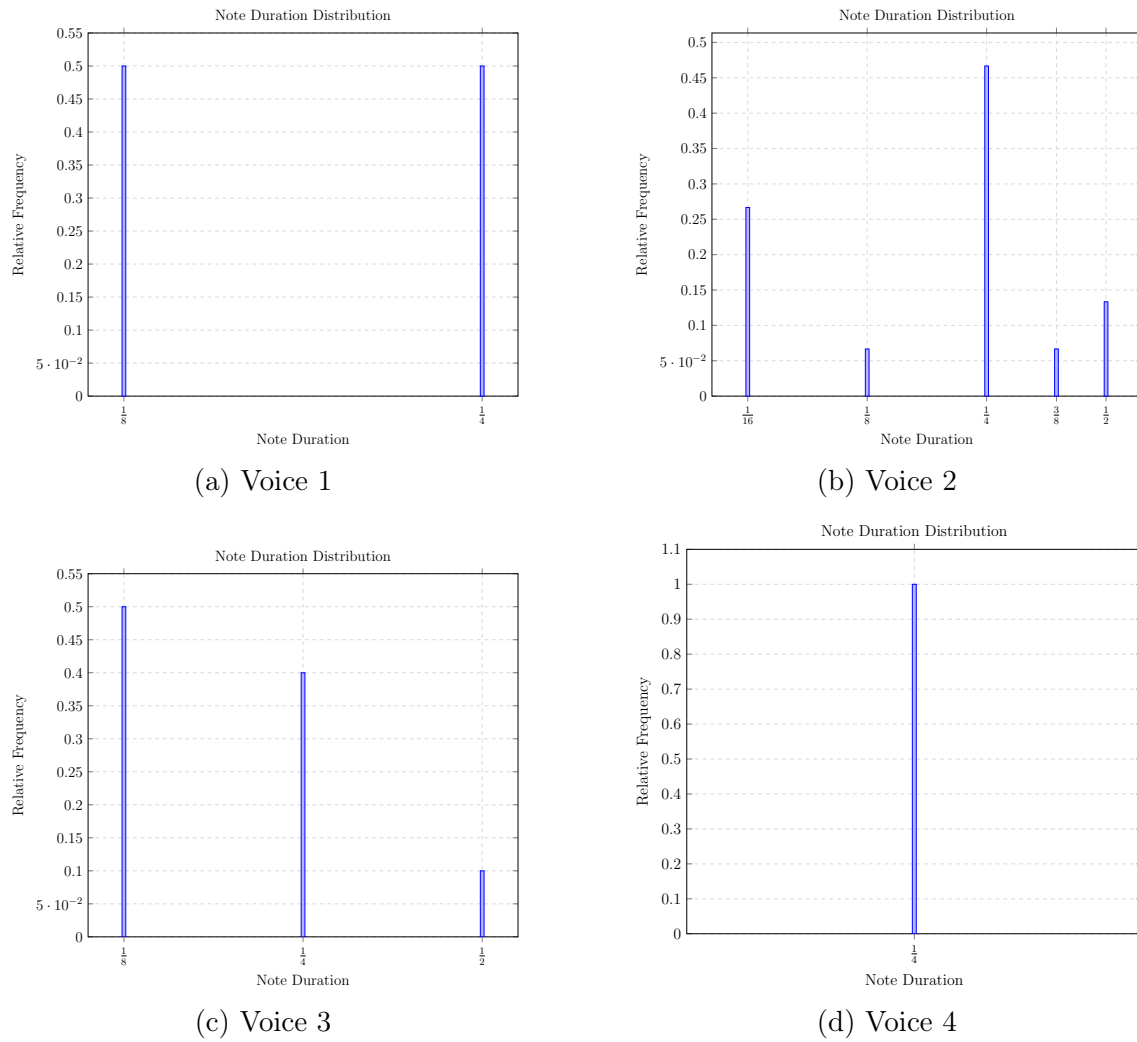
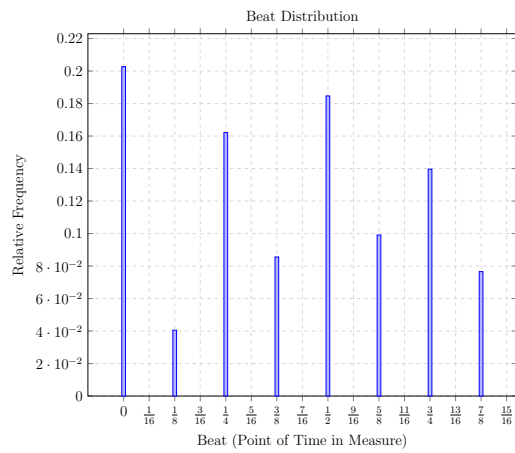
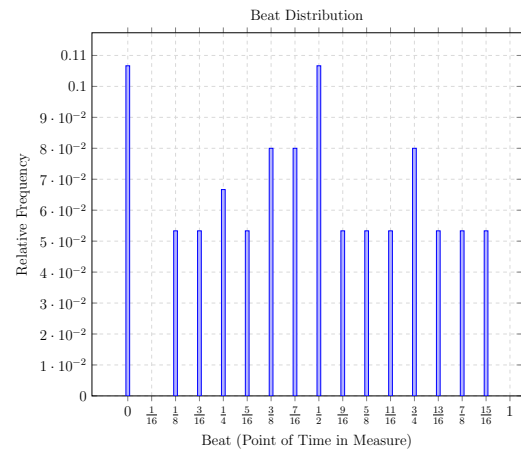


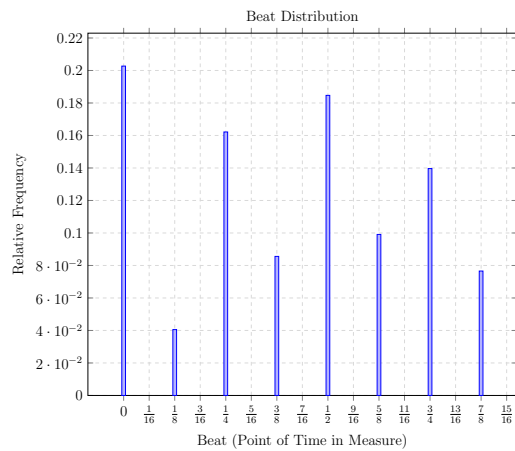
Figure 8.19: Target note duration distributions for the first section



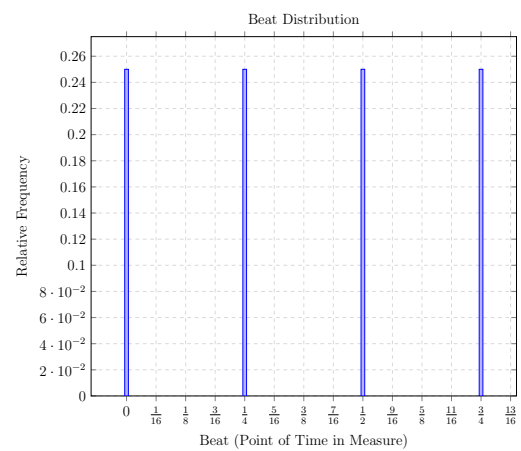
(a) Voice 1



(b) Voice 2

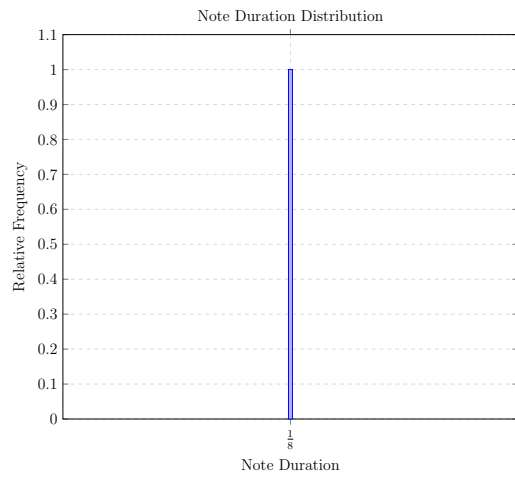


(c) Voice 3

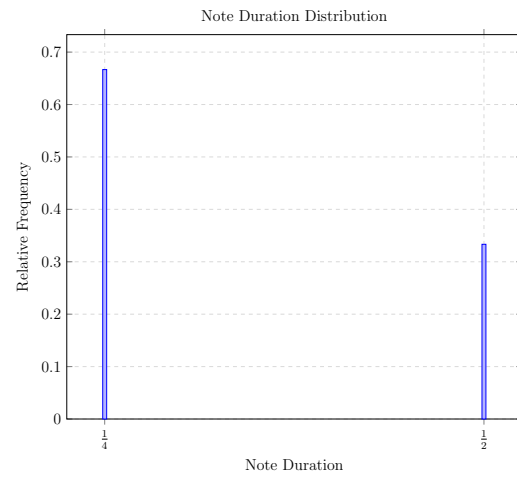


(d) Voice 4

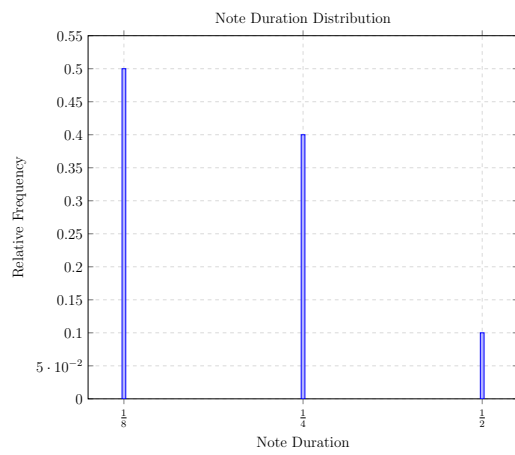
Figure 8.20: Target beat distributions for the first section



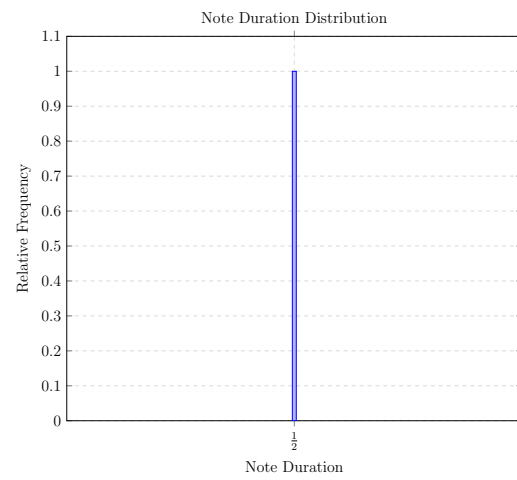
(a) Voice 1



(b) Voice 2

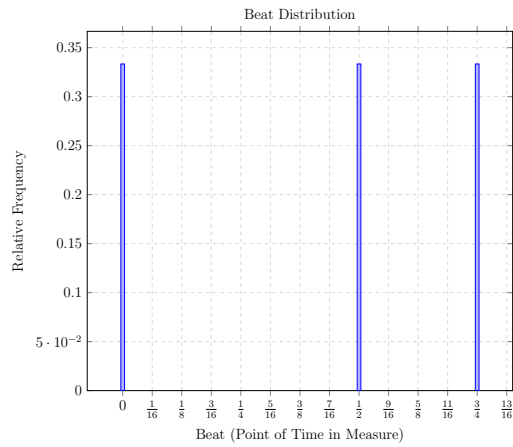


(c) Voice 3

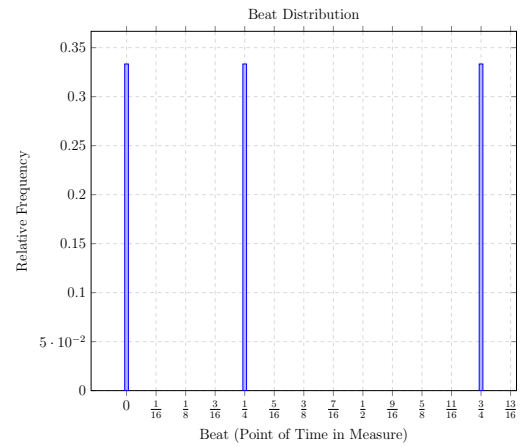


(d) Voice 4

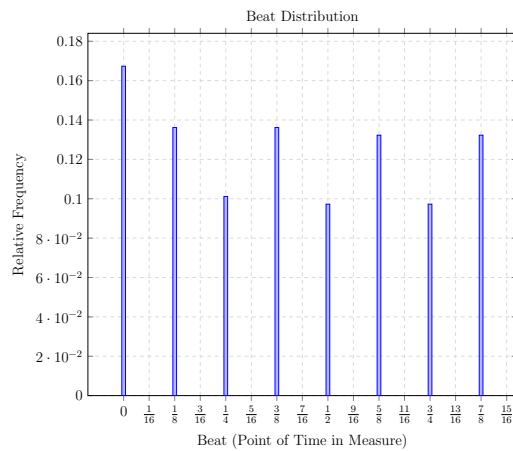
Figure 8.21: Target note duration distributions for the second section



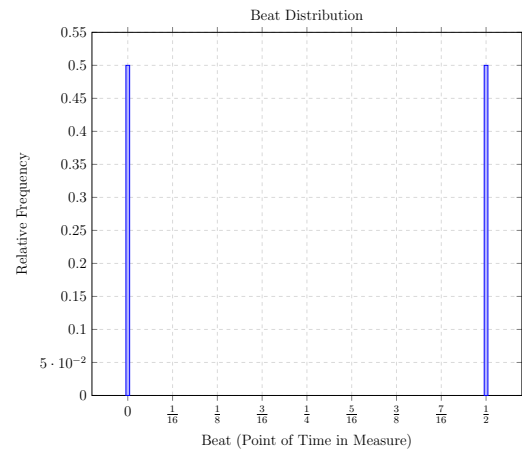
(a) Voice 1



(b) Voice 2

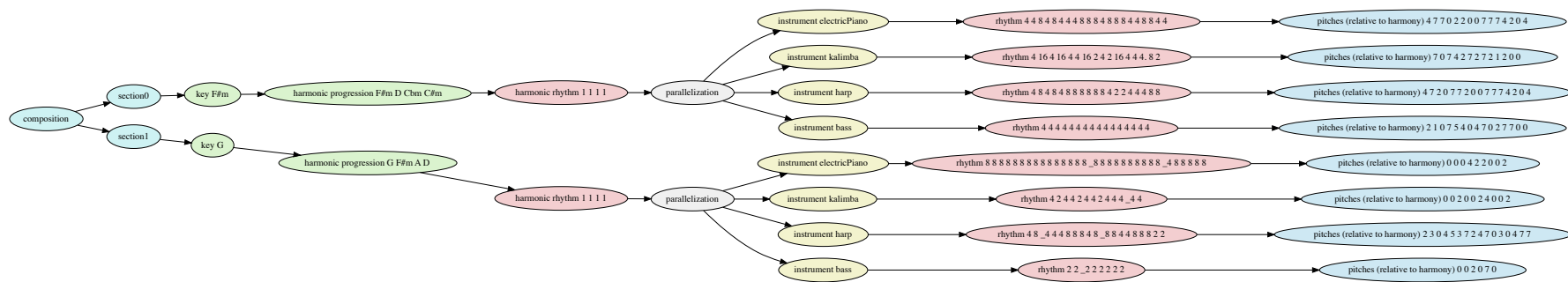


(c) Voice 3



(d) Voice 4

Figure 8.22: Target beat distributions for the second section



(a) Context tree model representation (in left-to-right layout for better readability)

(b) Score representation

Figure 8.23: Composition generated by the evolutionary algorithm containing two sections with different statistical and musical properties. Corresponding files are available on the accompanying CD under **Composer/SectionWiseComposition** (see Appendix **A**).

8.5 Summary

Four automated music composition scenarios based on an [EA](#) were presented in this chapter. The algorithm is not only capable of combining an arbitrary number of compositions to a new piece by recombining musical material, it can also be used to generate style imitations and original compositions with or without predefined musical structures. Although the musical quality of the compositions is not comparable to notable human composers, the algorithm is very versatile and pursuing the further development of the program is promising.

In future revisions, the algorithm could be extended to utilize more musical modifiers and control structures, which were introduced in Chapter [4](#). This could result in musically even more plausible compositions. Moreover, the exploration of statistical distribution combinations yielding musically interesting results deserve study. Why not combine the rhythmic qualities of Bach with jazz harmonies and Indian melody intervals? For this purpose, the development of a [GUI](#) replacing the current file-based interface (see Figure [8.17](#)) would be preferable.

Part IV

Conclusions and Appendices

Chapter 9

Conclusions

Do not fear mistakes. There are none.

— Miles Davis

In this work, various symbolic music processing applications and algorithms were developed in combination with new music representation models, which are capable of representing various musical aspects individually. This type of representation offers new possibilities for transforming, searching, analyzing and algorithmically generating music.

Two different models were proposed: a layer-based model, comprised of one or multiple streams containing musical context dimension layers, and a tree-based model, which has multiple advantages: musical information can be reused elegantly in order to minimize redundancy, musical modifications can be specified, and control structures are available, which can be used in conjunction with expressions. A corresponding domain-specific composition language for textual representation of tree-based composition models was developed to facilitate the work with textual representations of context tree models.

As a means to apply, test, and validate the proposed models, algorithms and the language, these were combined to a software application named MPS, which can be used by composers, musicians and scientists for composing, analyzing and generating music.

A key aspect of this work is to demonstrate that appropriate data representation models are equally important as the employed algorithms. For many music research applications, interpreting music as sequences of notes and rests is not sufficient. The multi-layered models introduced in this dissertation provide fine-grained access to individual musical dimensions, which are arbitrarily extensible.

Applications of the proposed models introduced in this work include importing sym-

bolic music standard formats, namely MIDI and MusicXML, and conversions between the proposed music representation models. The application is also capable of exporting LilyPond and SuperCollider files, which enables users to produce and play back scores.

A music search algorithm was proposed, which allows finding specific musical material in certain musical contexts. This advanced context-based search functionality goes beyond simple sequence-based search scenarios, because pattern combinations of individual musical aspects can be specified. This work also demonstrated advanced music analysis capabilities leveraging information extraction and combination of individual musical context layers in the proposed models.

Finally, an innovative evolutionary algorithm was introduced. It can be configured for various applications, including crossover of an arbitrary number of compositions, style imitations and section-wise composition generation. The algorithm reveals that some aspects of musical creativity can be modeled artificially. However, science is still far away from understanding and reproducing all aspects of human creativity with regard to music. The research contributions presented in this dissertation, however, describe some approaches in this direction.

Future enhancements of the system could be: supporting new standard formats for importing and exporting music, extending the model and the language with more musical contexts and elements, supporting electroacoustic compositions, improving percussion instrument support, adding new analysis algorithms and visualizations, optimizing the performance of the proposed algorithms, and providing graphical user interfaces for statistical features and distributions for the evolutionary algorithm.

In conclusion, **MPS** is a comprehensive software system for symbolic music processing, which can be used as a versatile tool for a broad range of applications, as shown in this dissertation. The software is freely available for Windows, Linux and Mac OS X at musicprocessing.net. It is designed to be a flexible platform for music notation, composition, analysis, generation and future music-related research projects.

Bibliography

- Alfonseca, Manuel et al. (2006). “A Fitness Function for Computer-generated Music using Genetic Algorithms”. In: *WSEAS Transactions on Information Science and Applications* 3 (3), pp. 518–525.
- Alfonseca, Manuel et al. (2007). “A Simple Genetic Algorithm for Music Generation by Means of Algorithmic Information Theory”. In: *IEEE Congress on Evolutionary Computation, CEC 2007* (Singapore), pp. 3035–3042.
- Allan, Moray (2002). “Harmonising Chorales in the Style of Johann Sebastian Bach”. PhD thesis. University of Edinburgh.
- Alpaydin, Ethem (2014). *Introduction to Machine Learning*. 3rd ed. MIT Press. ISBN: 9780262028189.
- Arbib, Michael A., ed. (2013). *Language, Music, and the Brain: A Mysterious Relationship*. Cambridge, Massachusetts: MIT Press. ISBN: 9780262314138.
- Ariza, Christopher and Michael Scott Cuthbert (2010). “Modeling Beats, Accents, Beams, and Time Signatures Hierarchically with music21 Meter Objects”. In: *Proceedings of the International Computer Music Conference, ICMC 2010* (New York, USA).
- Ariza, Christopher and Michael Scott Cuthbert (2011). “The music21 Stream: A New Object Model for Representing, Filtering, and Transforming Symbolic Musical Structures”. In: *Proceedings of the International Computer Music Conference, ICMC 2011* (Huddersfield, UK).
- Azevedo, Frederico A.C. et al. (2009). “Equal Numbers of Neuronal and Nonneuronal Cells Make the Human Brain an Isometrically Scaled-Up Primate Brain”. In: *Journal of Comparative Neurology* 513.5, pp. 532–541. DOI: [10.1002/cne.21974](https://doi.org/10.1002/cne.21974).
- Bagad, Vilas S. and Iresh A. Dhotre (2009). *Network Programming & Management*. Technical Publications. ISBN: 9788184317565.
- Barlow, Clarence (2012). *On Musiquantics*. Tech. rep. 51. Johannes Gutenberg University Mainz.
- Baroni, Mario et al. (1982). “A Grammar for Melody: Relationships between Melody and Harmony”. In: *Musical Grammars and Computer Analysis*. Ed. by Mario

- Baroni and Laura Callegari. Florence, Italy: Casa Editrice Leo S. Olschki. ISBN: 9788822232298.
- Batlle, Eloi and Pedro Cano (2000). “Automatic Segmentation for Music Classification using Competitive Hidden Markov Models”. In: *Proceedings of the First International Conference on Music Information Retrieval, ISMIR 2000* (Plymouth, Massachusetts).
- Becker, Judith and Alton Becker (1979). “A Grammar of the Musical Genre Srepegan”. In: *Journal of Music Theory* 23.1, pp. 1–43.
- Bel, Bernard and Jim Kippen (1992). “Modeling Music with Grammars: Formal Language Representation in the Bol Processor”. In: *Computer Representations and Models in Music*. Ed. by A. Marsden and A. Pople. London: Academic Press, pp. 207–238.
- Bellgard, Matthew I. and Chi Ping Tsang (1994). “Harmonizing Music the Boltzmann Way”. In: *Connection Science* 6.2-3, pp. 281–297.
- Bellini, Pierfrancesco et al. (2006). “Using MPEG Symbolic Music Representation in MPEG-4”. In: *Proceedings of the 2nd International Conference on Automated Production of Cross Media Content for Multi-channel Distribution, Axmedis 2006*. Firenze University Press, pp. 73–77.
- Bianchi, Filippo M. et al. (2017). *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. Springer.
- Biles, John A. (1994). “GenJam: A Genetic Algorithm for Generating Jazz Solos”. In: *Proceedings of the International Computer Music Conference, ICMC 1994* (Aarhus, Denmark), pp. 131–137.
- Biles, John A. (2007a). “Evolutionary Computation for Musical Tasks”. In: *Evolutionary Computer Music*. Ed. by Eduardo R. Miranda and John A. Biles. London: Springer. Chap. 2, pp. 28–51.
- Biles, John A. (2007b). “Improvising with Genetic Algorithms: GenJam”. In: *Evolutionary Computer Music*. Ed. by Eduardo R. Miranda and John A. Biles. Springer. Chap. 7, pp. 137–169.
- Blacking, John (1970). “Tonal Organization in the Music of Two Venda Initiation Schools”. In: *Ethnomusicology* 14.1, pp. 1–56. ISSN: 00141836.
- Bloch, Joshua (2018). *Effective Java*. 3rd ed. Addison-Wesley. ISBN: 978-0134685991.
- Boden, Margaret A. (1994). “Creativity and Computers”. In: *Artificial Intelligence and Creativity: An Interdisciplinary Approach*. Ed. by T. Dartnall. Space Technology Library. Dordrecht: Kluwer Academic Publishers, pp. 3–26. ISBN: 9780792330615.
- Boden, Margaret A. (2004). *The Creative Mind: Myths and Mechanisms*. Routledge. ISBN: 9780415314527.

- Boden, Margaret A. (2010). *Creativity and Art: Three Roads to Surprise*. Oxford University Press. ISBN: 9780199659395.
- Brooks, Frederick P. et al. (1957). “An Experiment in Musical Composition”. In: *IRE Transactions on Electronic Computers* EC-6.3, pp. 175–182.
- Broze, Yuri and Daniel Shanahan (2012). *The iRb Corpus in **jazz format*. URL: https://csml.som.ohio-state.edu/home/index.php/iRb_Jazz_Corpus (visited on 07/23/2018).
- Burnson, William A. et al. (2010). “Automatic Notation of Computer-Generated Scores for Instruments, Voices and Electro-Acoustic Sounds”. In: *Proceedings of the International Computer Music Conference, ICMC 2010* (New York, USA).
- Butchers, Christopher (1968). “The Random Arts: Xenakis, Mathematics and Music”. In: *Tempo* 85, pp. 2–5.
- Camilleri, Lelio (1982). “A Grammar of the Melodies of Schubert’s Lieder”. In: *Musical Grammars and Computer Analysis*. Ed. by Mario Baroni and Laura Callegari. Florence, Italy: Casa Editrice Leo S. Olschki, pp. 229–236. ISBN: 9788822232298.
- Chai, Wei and Barry Vercoe (2001). “Folk Music Classification Using Hidden Markov Models”. In: *Proceedings of the International Conference on Artificial Intelligence*. Vol. 6. 4.
- Cheng, Heng-Tze et al. (2008). “Automatic Chord Recognition for Music Classification and Retrieval”. In: *IEEE International Conference on Multimedia and Expo, ICME 2008* (Hannover, Germany), pp. 1505–1508.
- Choi, Keunwoo et al. (2016). “Text-based LSTM Networks for Automatic Music Composition”. In: *Proceedings of the 1st Conference on Computer Simulation of Musical Creativity* (Huddersfield, UK).
- Chomsky, Noam (1957). *Syntactic Structures*. Reprinted by Walter de Gruyter, Berlin, 2002. Den Haag: Mouton. ISBN: 9783110218329.
- Chomsky, Noam (1959). “On Certain Formal Properties of Grammars”. In: *Information and Control* 2.2, pp. 137–167. ISSN: 0019-9958.
- Cilibrasi, Rudi and Paul M.B. Vitányi (2005). “Clustering by Compression”. In: *IEEE Transactions on Information Theory* 51.4, pp. 1523–1545.
- Cilibrasi, Rudi, Paul M.B. Vitányi, and Ronald de Wolf (2004). “Algorithmic Clustering of Music Based on String Compression”. In: *Computer Music Journal* 28.4, pp. 49–67.
- Coca, Andrés E. et al. (2013). “Computer-aided Music Composition with LSTM Neural Network and Chaotic Inspiration”. In: *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2013* (Dallas, Texas). IEEE, pp. 270–276.

- Coello Coello, Carlos A. and Gary B. Lamont, eds. (2004). *Applications of Multi-objective Evolutionary Algorithms*. Vol. 1. Advances in Natural Computation. World Scientific. ISBN: 9789812567796.
- Collins, N. (2010). *Introduction to Computer Music*. Chichester: John Wiley & Sons. ISBN: 9780470714553.
- Cooper, G. and L.B. Meyer (1960). *The Rhythmic Structure of Music*. Phoenix Books. University of Chicago Press. ISBN: 9780226115221.
- Cope, David (1991). *Computers and Musical Style*. Oxford University Press. ISBN: 0-19-816274-X.
- Cope, David (1996). *Experiments in Musical Intelligence*. Middleton, Wisconsin: A-R Editions.
- Cope, David (2000). *The Algorithmic Composer*. Madison, Wisconsin: A-R Editions. ISBN: 9780895794543.
- Cope, David (2004). *Virtual Music: Computer Synthesis of Musical Style*. Cambridge, Massachusetts: MIT Press. ISBN: 9780262532617.
- Cope, David (2005). *Computer Models of Musical Creativity*. Cambridge, Massachusetts: MIT Press. ISBN: 0-262-03338-0.
- Cope, David (2009). *Hidden Structure: Music Analysis Using Computers*. Middleton, Wisconsin: A-R Editions. ISBN: 9780895796400.
- Cunha, Uraquitan Sidney and Geber Ramalho (1999). “An Intelligent Hybrid Model for Chord Prediction”. In: *Organised Sound* 4.2, pp. 115–119.
- Cuthbert, Michael Scott and Christopher Ariza (2010). “music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data”. In: *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010* (Utrecht, The Netherlands), pp. 637–642.
- Cuthbert, Michael Scott, Christopher Ariza, Jose Cabal-Ugaz, et al. (2011). “Hidden Beyond MIDI’s Reach: Feature Extraction and Machine Learning with Rich Symbolic Formats in music21”. In: *Proceedings of the 4th International Workshop on Machine Learning and Music, held in Conjunction with the 25th Annual Conference on Neural Information Processing Systems, NIPS 2011* (Sierra Nevada, Spain).
- Cuthbert, Michael Scott, Christopher Ariza, and Lisa Friedland (2011). “Feature Extraction and Machine Learning on Symbolic Music using the music21 Toolkit”. In: *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011* (Miami, Florida), pp. 387–392.
- Dannenbergh, Roger et al. (1997). “A Machine Learning Approach to Musical Style Recognition”. In: *Proceedings of the International Computer Music Conference, ICMC 1997* (Thessaloniki, Greece).

- Darwin, Charles (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: John Murray.
- Deza, Michel Marie and Elena Deza (2016). *Encyclopedia of Distances*. 4th ed. Berlin, Heidelberg: Springer. ISBN: 9783662443422.
- Eck, Douglas and Jasmin Lapalme (2008). *Learning Musical Structure Directly from Sequences of Music*. Tech. rep. 1300. University of Montreal, Department of Computer Science.
- Eck, Douglas and Jürgen Schmidhuber (2002a). *A First Look at Music Composition Using LSTM Recurrent Neural Networks*. Tech. rep. IDSIA-07-02. Manno, Switzerland: Istituto Dalle Molle di studi sull’intelligenza artificiale.
- Eck, Douglas and Jürgen Schmidhuber (2002b). “Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks”. In: *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pp. 747–756.
- Eigenfeldt, Arne (2009). “The Evolution of Evolutionary Software: Intelligent Rhythm Generation in Kinetic Engine”. In: *Applications of Evolutionary Computing*. Ed. by Mario Giacobini et al. Berlin, Heidelberg: Springer, pp. 498–507. ISBN: 978-3-642-01129-0.
- Eigenfeldt, Arne (2012). “Corpus-based Recombinant Composition Using a Genetic Algorithm”. In: *Soft Computing* 16.12, pp. 2049–2056. ISSN: 1433-7479. DOI: [10.1007/s00500-012-0871-z](https://doi.org/10.1007/s00500-012-0871-z).
- Eigenfeldt, Arne and Philippe Pasquier (2012). “Populations of Populations: Composing with Multiple Evolutionary Algorithms”. In: *Proceedings of the First International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design, EvoMUSART 2012* (Málaga, Spain). Ed. by Penousal Machado et al. Berlin, Heidelberg: Springer, pp. 72–83. ISBN: 978-3-642-29142-5.
- Fenwick, Peter (2014). *Introduction to Computer Data Representation*. Bentham Science Publishers. ISBN: 9781608058822.
- Flanagan, David (2005). *Java in a Nutshell*. O’Reilly. ISBN: 9780596007737.
- Fogel, Lawrence J. et al. (1966). *Artificial Intelligence Through Simulated Evolution*. New York: John Wiley & Sons.
- Fox, Charles (2006). “Genetic Hierarchical Music Structures”. In: *Proceedings of the 19th International FLAIRS Conference*. Menlo Park, California: AAAI Press.
- Friedberg, Richard M. (1958). “A Learning Machine: Part I”. In: *IBM Journal of Research and Development* 2.1, pp. 2–13. ISSN: 0018-8646. DOI: [10.1147/rd.21.0002](https://doi.org/10.1147/rd.21.0002).

- Friedberg, Richard M. et al. (1959). “A Learning Machine: Part II”. In: *IBM Journal of Research and Development* 3.3, pp. 282–287. ISSN: 0018-8646. DOI: [10.1147/rd.33.0282](https://doi.org/10.1147/rd.33.0282).
- Fu, Zhouyu et al. (2011). “A Survey of Audio-based Music Classification and Annotation”. In: *IEEE Transactions on Multimedia* 13.2, pp. 303–319.
- Geiringer, Karl (1969). “Der Einfluss der Aufklärung auf J. S. Bachs künstlerisches Denken”. In: *Studia Musicologica Academiae Scientiarum Hungaricae* 11.1/4, pp. 201–206. ISSN: 15882888.
- Gentle, James E. (2006). *Random Number Generation and Monte Carlo Methods*. New York: Springer. ISBN: 9780387216102.
- Ghanea-Hercock, Robert (2003). *Applied Evolutionary Algorithms in Java*. Springer. ISBN: 9780387955681.
- Gibson, P.M. and J.A. Byrne (1991). “NEUROGEN: Musical Composition using Genetic Algorithms and Cooperating Neural Networks”. In: *Proceedings of the Second International IEEE Conference on Artificial Neural Networks*, pp. 309–313.
- Glashoff, Klaus (2003). *Gottfried Wilhelm Leibniz – die Utopie der Denkmachine*. URL: <http://s371741603.online.de/glashoffnet/logicglashoffnet/Texte/GottfriedWilhelmLeibniz6.pdf> (visited on 05/13/2018).
- Goldstein, E.B. (2013). *Sensation and Perception*. Cengage Learning. ISBN: 9781133958499.
- Good, Michael (2001). “MusicXML: An Internet-Friendly Format for Sheet Music”. In: *Proceedings of the International XML Conference and Expo, XMLEdge 2001* (Santa Clara, California).
- Goodfellow, Ian et al. (2016). *Deep Learning*. Cambridge, Massachusetts: MIT Press. ISBN: 978-0-262-03561-3.
- Han, Te Sun and Kingo Kobayashi (2007). *Mathematics of Information and Coding*. Providence, Rhode Island: American Mathematical Society. ISBN: 9780821842560.
- Hankerson, Darrel R. et al. (2003). *Introduction to Information Theory and Data Compression*. 2nd ed. Boca Raton, Florida: CRC Press. ISBN: 9781584883135.
- Hawthorne, Curtis et al. (2018). “Onsets and Frames: Dual-Objective Piano Transcription”. In: *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018* (Paris, France).
- Hewlett, Walter B. (1997). “MuseData: Multipurpose Representation”. In: *Beyond MIDI*. Ed. by Eleanor Selfridge-Field. Cambridge, Massachusetts: MIT Press, pp. 402–447.
- Hild, Hermann et al. (1992). “HARMONET: A Neural Net for Harmonizing Chorales in the Style of J.S. Bach”. In: *Advances in Neural Information Processing Systems*, pp. 267–274.

- Hiller, Lejaren A. and Leonard M. Isaacson (1958). “Musical Composition with a High-Speed Digital Computer”. In: *Journal of the Audio Engineering Society* 6.3, pp. 154–160.
- Hiller, Lejaren A. and Leonard M. Isaacson (1959). *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill.
- Hinton, Geoffrey E. and Terrence J. Sejnowski (1986). “Learning and Relearning in Boltzmann Machines”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 1.282-317, p. 2.
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780.
- Hofmann, David M. (2015). “A Genetic Programming Approach to Generating Musical Compositions”. In: *Proceedings of the 4th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design, EvoMUSART 2015* (Copenhagen, Denmark). Vol. 9027. LNCS. Springer, pp. 89–100.
- Hofmann, David M. (2016). “Introducing a Context-based Model and Language for Representation, Transformation, Visualization, Analysis and Generation of Music”. In: *Proceedings of the 42nd International Computer Music Conference, ICMC 2016*. Ed. by Hans Timmermans. Utrecht, The Netherlands: HKU University of the Arts, pp. 381–387.
- Hofstadter, Douglas R. (1979). *Gödel, Escher, Bach: An Eternal Golden Braid*. Twentieth-anniversary Edition. New York: Basic Books.
- Holland, John H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, Massachusetts: MIT Press. ISBN: 0262082136.
- Holtzman, Steven R. (1980). “A Generative Grammar Definition Language for Music”. In: *Interface* 9.1, pp. 1–48. DOI: [10.1080/09298218008570279](https://doi.org/10.1080/09298218008570279).
- Hoos, Holger H., Keith A. Hamel, et al. (1998). “The GUIDO Notation Format — A Novel Approach for Adequately Representing Score-Level Music”. In: *Proceedings of the International Computer Music Conference, ICMC 1998* (Ann Arbor, Michigan), pp. 451–454.
- Hoos, Holger H., Kai Renz, et al. (2001). “GUIDO/MIR — An Experimental Musical Information Retrieval System based on GUIDO Music Notation”. In: *Proceedings of the 2nd International Symposium on Music Information Retrieval, ISMIR 2001* (Bloomington, Indiana), pp. 41–50.
- Hopfield, John J. (1982). “Neural Networks and Physical Systems with Emergent Collective Computational Abilities”. In: *Proceedings of the National Academy of Sciences* 79.8, pp. 2554–2558.

- Horáček, Petr et al. (2011). *Augmented Transition Networks*. URL: <https://pdfs.semanticscholar.org/presentation/6ef1/2e49ce3f3c33c5c475d5b025995b0e1c423d.pdf> (visited on 05/09/2018).
- Horner, Andrew and David E. Goldberg (1991). “Genetic Algorithms and Computer-Assisted Music Composition”. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. Ed. by R. Belew and L. Booker. San Mateo, California: Morgan Kaufmann, pp. 437–441.
- Hughes, David W. (1988). “Deep Structure and Surface Structure in Javanese Music: A Grammar of Gendhing Lampah”. In: *Ethnomusicology* 32.1, pp. 23–74. ISSN: 00141836.
- Hughes, David W. (1991). “Grammars of Non-Western Musics: A Selective Survey”. In: *Representing Musical Structure*. Ed. by P. Howell et al. Cognitive Science Series. London: Academic Press. ISBN: 9780123571717.
- Huron, David (1995). *The Humdrum User Guide*. URL: <http://www.humdrum.org/guide/> (visited on 04/10/2018).
- Huron, David (2002). “Music Information Processing Using the Humdrum Toolkit: Concepts, Examples, and Lessons”. In: *Computer Music Journal* 26.2, pp. 11–26. DOI: [10.1162/014892602760137158](https://doi.org/10.1162/014892602760137158).
- Huron, David (2006). *Sweet Anticipation: Music and the Psychology of Expectation*. MIT Press. ISBN: 9780262083454.
- Husbands, Phil et al. (2007). “An Introduction to Evolutionary Computing for Musicians”. In: *Evolutionary Computer Music*. Ed. by Eduardo R. Miranda and John A. Biles. London: Springer, pp. 1–27. ISBN: 978-1-84628-600-1. DOI: [10.1007/978-1-84628-600-1_1](https://doi.org/10.1007/978-1-84628-600-1_1).
- IEEE, Computer Society Standards Committee (1985). *IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*. Institute of Electrical and Electronics Engineers (IEEE). DOI: [10.1109/ieeestd.1985.82928](https://doi.org/10.1109/ieeestd.1985.82928).
- Jacob, Bruce L. (1995). “Composing with Genetic Algorithms”. In: *Proceedings of the International Computer Music Conference, ICMC 1995* (Banff, Alberta, Canada), pp. 452–455.
- Jacob, Bruce L. (1996). “Algorithmic Composition as a Model of Creativity”. In: *Organised Sound* 1(3), pp. 157–165.
- Johanson, Brad and Riccardo Poli (1998). “GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters”. In: *Proceedings of the Third Annual Conference on Genetic Programming, GP’98*. Ed. by John R. Koza et al. San Francisco, California: Morgan Kaufmann, pp. 181–186.

- Johnson-Laird, Philip N. (1991). “Jazz Improvisation: A Theory at the Computational Level”. In: *Representing Musical Structure*. Ed. by P. Howell et al. London: Academic Press. ISBN: 9780123571717.
- Johnson-Laird, Philip N. (2002). “How Jazz Musicians Improvise”. In: *Music Perception: An Interdisciplinary Journal* 19.3, pp. 415–442.
- Jones, Kevin (1981). “Compositional Applications of Stochastic Processes”. In: *Computer Music Journal* 5.2, pp. 45–61. ISSN: 15315169.
- Kiernan, Francis J. (2000). “Score-based Style Recognition Using Artificial Neural Networks”. In: *Proceedings of the First International Conference on Music Information Retrieval, ISMIR 2000* (Plymouth, Massachusetts).
- Koenig, Gottfried Michael (1971). “Serielle und aleatorische Verfahren in der elektronischen Musik”. In: *Ästhetische Praxis. Texte zur Musik* 2, pp. 1962–1967.
- Kohonen, Teuvo (1982). “Self-organized Formation of Topologically Correct Feature Maps”. In: *Biological Cybernetics* 43.1, pp. 59–69.
- Kohonen, Teuvo (1989). “A Self-learning Musical Grammar, or ‘Associative Memory of the Second Kind’”. In: *Proceedings of the International Joint Conference on Neural Networks, IJCNN 1989* (Washington, D.C.). San Diego, California: IEEE, pp. 1–5.
- Kostelanetz, Richard (2003). *Conversing with Cage*. Second Edition. Routledge. ISBN: 9780415937924.
- Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press. ISBN: 0-262-11170-5.
- Krawczak, Maciej (2013). *Multilayer Neural Networks: A Generalized Net Perspective*. Studies in Computational Intelligence. Springer International Publishing. ISBN: 9783319002484.
- Kumar, Rajendra (2010). *Theory Of Automata, Languages and Computation*. New Delhi: Tata McGraw-Hill. ISBN: 9780070702042.
- Lerdahl, Fred (2001). *Tonal Pitch Space*. Oxford University Press. ISBN: 9780195346374.
- Lerdahl, Fred and Ray Jackendoff (1983). *A Generative Theory of Tonal Music*. MIT Press. ISBN: 9780262260916.
- Li, Ming, Xin Chen, et al. (2004). “The Similarity Metric”. In: *IEEE transactions on Information Theory* 50.12, pp. 3250–3264.
- Li, Ming and Paul Vitányi (2013). *An Introduction to Kolmogorov Complexity and its Applications*. New York: Springer. ISBN: 9781475738605.
- Loughran, R. et al. (2015). “Tonality Driven Piano Compositions with Grammatical Evolution”. In: *IEEE Congress on Evolutionary Computation, CEC 2015*, pp. 2168–2175. DOI: [10.1109/CEC.2015.7257152](https://doi.org/10.1109/CEC.2015.7257152).

- Luke, Sean (2013). *Essentials of Metaheuristics*. Second Edition. Lulu. URL: <http://cs.gmu.edu/~sean/book/metaheuristics/>
- Manaris, Bill, Penousal Machado, et al. (2005). “Developing Fitness Functions for Pleasant Music: Zipf’s Law and Interactive Evolution Systems”. In: *EvoWorkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC* (Lausanne, Switzerland). Vol. 3449. LNCS, pp. 498–507.
- Manaris, Bill, Patrick Roos, et al. (2007). “A Corpus-based Hybrid Approach to Music Analysis and Composition”. In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence* (Vancouver, British Columbia, Canada). Menlo Park, California: AAAI Press, pp. 839–845. ISBN: 978-1-57735-323-2.
- Manaris, Bill, Dallas Vaughan, et al. (2003). “Evolutionary Music and the Zipf-Mandelbrot Law: Developing Fitness Functions for Pleasant Music”. In: *EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM* (Essex, UK). Vol. 2611. LNCS. Springer, pp. 522–534.
- Mandelbrot, Benoît B. (2003). “Multifractal Power Law Distributions: Negative and Critical Dimensions and other ‘Anomalies,’ Explained by a Simple Example”. In: *Journal of Statistical Physics* 110.3-6, pp. 739–774.
- Marchal, Benoît (2002). *XML by Example*. 2nd ed. Que. ISBN: 9780789725042.
- McCormack, Jon and Mark d’Inverno, eds. (2012). *Computers and Creativity*. Springer. ISBN: 9783642317279.
- McDonald, Chris (2000). “Exploring modal subversions in alternative music”. In: *Popular Music* 19.3, pp. 355–363.
- Miranda, Eduardo R. (2001). *Composing Music with Computers*. Routledge. ISBN: 9781136120930.
- Mozer, Michael (1994). “Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing”. In: *Connection Science* 6 (1), pp. 247–280.
- Müller, Meinard (2015). *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer. ISBN: 9783319219455.
- Neapolitan, Richard E. (2015). *Foundations of Algorithms*. 5th ed. Burlington, Massachusetts: Jones & Bartlett Learning. ISBN: 9781284049206.
- Nevill-Manning, Craig G. and Ian H. Witten (1997). “Identifying Hierarchical Structure in Sequences: A Linear-time Algorithm”. In: *Journal of Artificial Intelligence Research* 7, pp. 67–82.
- Nienhuys, Han-Wen and Jan Nieuwenhuizen (2003). “LilyPond, a System for Automated Music Engraving”. In: *Proceedings of the XIV Colloquium on Musical Informatics*. Ed. by Nicola Bernardini et al. Florence, Italy: Associazione Italiana di Musica Informatica, pp. 167–172.

- Nierhaus, Gerhard (2009). *Algorithmic Composition: Paradigms of Automated Music Generation*. Vienna, Austria: Springer. ISBN: 9783211755402.
- Nilsson, Nils J. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.
- Ochoa, Gabriela (1998). “On Genetic Algorithms and Lindenmayer Systems”. In: *Parallel Problem Solving from Nature — PPSN V* (Amsterdam, The Netherlands). Ed. by Agoston E. Eiben et al. Berlin, Heidelberg: Springer, pp. 335–344.
- Ockelford, Adam (2016). *Repetition in Music: Theoretical and Metatheoretical Perspectives*. Routledge. ISBN: 978-0-7546-3573-4.
- Olson, Harry F. (1967). *Music, Physics and Engineering*. 2nd ed. Originally published under the title *Musical Engineering* by McGraw-Hill Book Company in 1952. New York: Dover Publications. ISBN: 9780486217697.
- Oore, Sageev et al. (2017). “Learning to Create Piano Performances”. In: *Proceedings of the Thirty-first Annual Conference on Neural Information Processing Systems, NIPS 2017* (Long Beach, California).
- Ortega, Alfonso et al. (2002). “Automatic Composition of Music by Means of Grammatical Evolution”. In: *Proceedings of the 2002 Conference on APL*. New York: ACM Press, pp. 148–155.
- Pachet, François (1999). “Surprising Harmonies”. In: *International Journal of Computing Anticipatory Systems* 4, pp. 139–161.
- Pachet, François (2012). “Musical Virtuosity and Creativity”. In: *Computers and Creativity*. Berlin, Heidelberg: Springer. Chap. 5, pp. 115–146.
- Papadopoulos, George and Geraint Wiggins (1998). “A Genetic Algorithm for the Generation of Jazz Melodies”. In: *Proceedings of STeP 98*.
- Patel, Aniruddh D. (2008). *Music, Language, and the Brain*. New York: Oxford University Press. ISBN: 9780195123753.
- Patterson, David A. and John L. Hennessy (2008). *Computer Organization and Design: The Hardware/Software Interface*. Elsevier Science. ISBN: 9780080922812.
- Pelinski, Ramon (1982). “A Generative Grammar of Personal Eskimo Songs”. In: *Musical Grammars and Computer Analysis*. Ed. by Mario Baroni and Laura Callegari. Florence, Italy: Casa Editrice Leo S. Olschki, pp. 229–236. ISBN: 9788822232298.
- Poli, Riccardo et al. (2008). *A Field Guide to Genetic Programming*. Lulu. ISBN: 978-1-4092-0073-4. URL: <http://www.gp-field-guide.org.uk>.
- Polito, John et al. (1997). “Musica Ex Machina: Composing 16th-century Counterpoint with Genetic Programming and Symbiosis”. In: *Proceedings of the Sixth*

- Annual Conference on Evolutionary Programming* (Indianapolis, Indiana). Ed. by Peter J. Angeline et al. Springer.
- Pollastri, Emanuele and Giuliano Simoncelli (2001). “Classification of Melodies by Composer with Hidden Markov Models”. In: *First International Conference on Web Delivering of Music*. IEEE, pp. 88–95.
- Ponsford, Dan et al. (1999). “Statistical Learning of Harmonic Movement”. In: *Journal of New Music Research* 28, pp. 150–177.
- Priddy, Kevin L. and Paul E. Keller (2005). *Artificial Neural Networks: An Introduction*. Bellingham, Washington: International Society for Optical Engineering. ISBN: 9780819459879.
- Pruim, Randall (2018). *Foundations and Applications of Statistics: An Introduction Using R*. Second Edition. Providence, Rhode Island: American Mathematical Society. ISBN: 9781470428488.
- Prusinkiewicz, Przemyslaw and Aristid Lindenmayer (1990). *The Algorithmic Beauty of Plants*. New York: Springer. ISBN: 9781461384762.
- Putnam, Jeffrey (1996). “A Grammar-Based Genetic Programming Technique Applied to Music Generation”. In: *Proceedings of the Fifth Annual Conference on Evolutionary Programming* (San Diego, California). Cambridge, Massachusetts: MIT Press, pp. 277–286.
- Rader, Gary M. (1974). “A Method for Composing Simple Traditional Music by Computer”. In: *Communications of the ACM* 17.11, pp. 631–638.
- Ralley, David (1995). “Genetic Algorithm as a Tool for Melodic Development”. In: *Proceedings of the International Computer Music Conference, ICMC 1995* (Banff, Alberta, Canada), pp. 501–502.
- Ramalho, Geber and Jean-Gabriel Ganascia (1994). “Simulating Creativity in Jazz Performance”. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI’94* (Seattle, Washington). Menlo Park, California: AAAI Press, pp. 108–113.
- Randel, Don M. (2003). *The Harvard Dictionary of Music*. 4th ed. Harvard University Press Reference Library. Belknap Press of Harvard University Press. ISBN: 9780674011632.
- Rechenberg, Ingo (1973). *Evolutionsstrategie — Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog.
- Roads, Curtis (1979). “Grammars as Representations for Music”. In: *Computer Music Journal* 3.1, pp. 48–55.
- Roberts, Adam et al. (2018). “A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music”. In: *Proceedings of the Thirty-fifth International Conference on Machine Learning, ICML 2018* (Stockholm, Sweden).

- Rohrmeier, Martin (2011). “Towards a Generative Syntax of Tonal Harmony”. In: *Journal of Mathematics and Music* 5.1, pp. 35–53.
- Romero, Juan J. and Penousal Machado, eds. (2008). *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Berlin, Heidelberg: Springer. ISBN: 978-3-540-72876-4.
- Rumelhart, David E. et al. (1986). “Learning Representations by Back-propagating Errors”. In: *Nature* 323, pp. 533–536.
- Sadie, Stanley and John Tyrrell, eds. (2001). *The New Grove Dictionary of Music and Musicians*. Second Edition. Vol. 6. Macmillan Publishers.
- Safranek, Milos (2013). *Bohuslav Martinu — The Man and His Music*. Read Books Limited. ISBN: 9781447493044.
- Saichev, Alexander I. et al. (2009). *Theory of Zipf’s Law and Beyond*. Berlin, Heidelberg: Springer. ISBN: 9783642029462.
- Salomon, David (2007). *Data Compression: The Complete Reference*. Fourth Edition. London: Springer. ISBN: 9781846286032.
- Sampaio, Marcos da Silva et al. (2013). “The Implementation of a Contour Module for Music21”. In: *ART Music Review* 24.
- Sapp, Craig Stuart et al. (2004). “Search Effectiveness Measures for Symbolic Music Queries in Very Large Databases.” In: *Proceedings of the 5th International Conference on Music Information Retrieval, ISMIR 2004* (Barcelona, Spain).
- Sarker, Ruhul and Carlos A. Coello Coello (2003). “Assessment Methodologies for Multiobjective Evolutionary Algorithms”. In: *Evolutionary Optimization*. Springer, pp. 177–195.
- Scaringella, Nicolas et al. (2006). “Automatic Genre Classification of Music Content: A Survey”. In: *IEEE Signal Processing Magazine* 23.2, pp. 133–141.
- Schenker, Heinrich (1935). *Der freie Satz*. Vol. 3. Neue musikalische Theorien und Phantasien. Vienna: Universal-Edition.
- Schmeder, Andrew et al. (2010). “Best Practices for Open Sound Control”. In: *Proceedings of the 2010 Linux Audio Conference* (Utrecht, the Netherlands).
- Shao, Xi et al. (2004). “Unsupervised Classification of Music Genre using Hidden Markov Model”. In: *IEEE International Conference on Multimedia and Expo, ICME’04* (Taipei, Taiwan). Vol. 3. IEEE, pp. 2023–2026.
- Simon, Herbert A. and Allen Newell (1971). “Human Problem Solving: The State of the Theory in 1970”. In: *American Psychologist* 26.2, pp. 145–159.
- Sivanandam, S.N. and S.N. Deepa (2007). *Introduction to Genetic Algorithms*. Berlin; Heidelberg: Springer. ISBN: 9783540731900.
- Spector, Lee and Adam Alpern (1994). “Criticism, Culture, and the Automatic Generation of Artworks”. In: *Proceedings of the Twelfth National Conference on*

- Artificial Intelligence, AAAI'94* (Seattle, Washington). Menlo Park, California: AAAI Press.
- Spector, Lee and Adam Alpern (1995). "Induction and Recapitulation of Deep Musical Structure". In: *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI 1995* (Montreal, Quebec, Canada), pp. 20–25.
- Steedman, Mark J. (1984). "A Generative Grammar for Jazz Chord Sequences". In: *Music Perception: An Interdisciplinary Journal* 2.1, pp. 52–77. ISSN: 15338312.
- Steinberg, Dave et al. (2008). *EMF: Eclipse Modeling Framework*. 2nd ed. Addison-Wesley. ISBN: 9780132702218.
- Stewart, William J. (2009). *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*. Princeton University Press. ISBN: 9781400832811.
- Storer, James A. (2002). *An Introduction to Data Structures and Algorithms*. Springer. ISBN: 9781461200758.
- Sundberg, Johan and Björn Lindblom (1976). "Generative Theories in Language and Music Descriptions". In: *Cognition* 4.1, pp. 99–122.
- Svegliato, Justin (2017). *Can a deep neural network compose music?* URL: <https://towardsdatascience.com/can-a-deep-neural-network-compose-music-f89b6ba4978d> (visited on 07/16/2018).
- Temperley, David and Daniel Sleator (1999). "Modeling Meter and Harmony: A Preference-Rule Approach". In: *Computer Music Journal* 23.1, pp. 10–27. DOI: [10.1162/014892699559616](https://doi.org/10.1162/014892699559616).
- Thywissen, Kurt (1996). "GeNotator: An Environment for Investigating the Application of Genetic Algorithms in Computer Assisted Composition". In: *Proceedings of the International Computer Music Conference, ICMC 1996* (Hong Kong, China).
- Thywissen, Kurt (1999). "GeNotator: An Environment for Exploring the Application of Evolutionary Techniques in Computer Assisted Composition". In: *Organised Sound* 4 (2), pp. 127–133.
- Tillmann, Barbara et al. (2000). "Implicit Learning of Tonality: A Self-organizing Approach". In: *Psychological Review* 107.4, p. 885.
- Todd, Peter M. (1989). "A Connectionist Approach to Algorithmic Composition". In: *Computer Music Journal* 13.4, pp. 27–43.
- Troche, Sarah (2018). "Cage as Frankenstein: Monstrosity and Indeterminacy of Performance". In: *Tacet - Experimental Music Review*. Ed. by Matthieu Saladin. Les presses du réel. ISBN: 9782840664734.

- Typke, Rainer et al. (2005). “A Survey of Music Information Retrieval Systems”. In: *Proceedings of the 6th International Conference on Music Information Retrieval, ISMIR 2005* (London, UK), pp. 153–160.
- Veerarajan, T (2002). *Probability, Statistics And Random Processes*. New Delhi: Tata McGraw-Hill. ISBN: 9780070494824.
- Viro, Vladimir (2011). “Peachnote: Music Score Search and Analysis Platform”. In: *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011* (Miami, Florida), pp. 359–362.
- Walshaw, Chris (2011). *The abc Music Standard 2.1*. URL: <http://abcnotation.com/wiki/abc:standard:v2.1> (visited on 04/14/2018).
- Warford, J. Stanley (2017). *Computer Systems*. 5th ed. Jones & Bartlett Learning. ISBN: 9781284079630.
- Waschka II, Rodney (2007). “Composing with Genetic Algorithms: GenDash”. In: *Evolutionary Computer Music*. Ed. by Eduardo R. Miranda and John A. Biles. London: Springer. Chap. 6, pp. 117–136.
- Weihs, Claus et al. (2016). *Music Data Analysis: Foundations and Applications*. CRC Press. ISBN: 9781315353838.
- White, Harvey E. and Donald H. White (2014). *Physics and Music: The Science of Musical Sound*. Originally published by Saunders College / Holt, Rinehart and Winston, Philadelphia, in 1980. Dover Publications. ISBN: 9780486794006.
- Wilson, Scott et al. (2011). *The SuperCollider Book*. The MIT Press. ISBN: 9780262232692.
- Wright, David (2009). *Mathematics and Music*. Providence, Rhode Island: American Mathematical Society. ISBN: 9780821848739.
- Xenakis, Iannis (1966). “The Origins of Stochastic Music”. In: *Tempo* 78, pp. 9–12.
- Xenakis, Iannis (1992). *Formalized Music: Thought and Mathematics in Composition*. Pendragon Press. ISBN: 9781576470794.
- Xu, Changsheng et al. (2005). “Automatic Music Classification and Summarization”. In: *IEEE Transactions on Speech and Audio Processing* 13.3, pp. 441–450.
- Yu, Tina (1999). “Structure Abstraction and Genetic Programming”. In: *Proceedings of the 1999 Congress on Evolutionary Computation, CEC’99* (Washington, D.C.). IEEE, pp. 652–659.
- Ziv, Jacob and Abraham Lempel (1978). “Compression of individual sequences via variable-rate coding”. In: *IEEE Transactions on Information Theory* 24.5, pp. 530–536.

Appendix A

Contents of the Accompanying CD

The accompanying CD contains accessory digital material relating to this dissertation. Refer to table [A.1](#) for a detailed description of the CD contents.

Table A.1: CD Contents

Folder Name	Description
Analysis	Analysis reports generated with MPS .
Class Diagrams	Class diagrams of data representation models developed for MPS .
Corpus	Corpus of compositions in MusicXML format used for analysis and search applications.
Composer	Compositions generated by the evolutionary algorithm and related files.
Documentation	The complete MPS user documentation and reference in PDF and Hypertext Markup Language (HTML) format.
Examples	Example compositions in MC²L format along with corresponding tree model and stream model visualizations, scores, MIDI files and audio files in MPEG Audio Layer III (MP3) format.
MIDI	MIDI file examples and corresponding MP3 audio files.
Source Code	The complete source code of MPS including auxiliary files and build scripts.

Appendix B

Code Examples

B.1 Composition Language Grammar

```
1  /**
2   * Grammar of the Musical Context Composition Language (MC2L)
3   * Component of Music Processing Suite (MPS)
4   *
5   * Created by David Hofmann <dev@davehofmann.de>
6   */
7  grammar eu.hfm.mps.dsl.MusicalCompositionLanguage with org.eclipse.xtext.common.
    Terminals
8
9  import "http://www.hfm.eu/mps/compositionContextModel"
10 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
11
12 /*
13  * Root parser rule for MCL files
14  */
15 Model:
16     imports += Import*
17     headers += Header*
18     (
19         rootNode = RootNode
20         nodes += Node*
21     )?;
22
23 /*
24  * Terminal for integer numbers
25  */
26 terminal INT returns ecore::EInt: '-'? ('0'..'9')+;
27
28 /*
29  * Terminal for identifiers
30  */
31 terminal ID: ('a'..'z') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
32
33 /*
34  * Terminal for note names (can have zero or n #/b suffixes)
35  */
```

```

36 terminal NOTENAME: ('C' | 'D' | 'E' | 'F' | 'G' | 'A' | 'B')(( '#' )+ | ('b' )+)?;
37
38 /*
39  * datatype rule for floating point numbers
40  */
41 FLOAT returns ecore::EDouble: (INT '.' INT);
42
43 /*
44  * need this datatype rule in order to allow keywords as identifiers, too
45  */
46 IDENTIFIER: ID | 'percussion' | 'synth' | 'treble' | 'alto' | 'tenor' | 'bass';
47
48 Import:
49     'import' importURI=STRING;
50
51 Header:
52     Definition | Metadata;
53
54 RootNode returns Node:
55     {Node} 'composition' (fixed ?= 'fixed')? (final ?= 'final')? ('{' children +=
56         Node* '}' )?;
57
58 Node:
59     (Context | Modifier | ControlStructure | Generator)
60     (fixed ?= 'fixed')?
61     (final ?= 'final')?
62     ('{' children += Node* '}' | ',' children += Node)?;
63
64 Context:
65     TonalSystem | TimeSignature | Tempo | HarmonicContext | Key | DynamicContext |
66     Fragment | FragmentReference | Instrument | Scale | HarmonicRhythm |
67     HarmonicProgression | Pitches | Rhythm | Envelope | Lyrics | CustomContext;
68
69 Modifier:
70     RhythmicModifier | TonalModifier | HarmonicModifier;
71
72 RhythmicModifier:
73     RhythmicDisplacement | RhythmicInsertion | Augmentation | Diminution |
74     RhythmicExtension | RhythmicAdjustment;
75
76 TonalModifier:
77     Transposition | Inversion | ParallelInterval;
78
79 HarmonicModifier:
80     HarmonyModifier;
81
82 ControlStructure:
83     Repetition | Parallelization | Iteration | Condition | While | FunctionCall |
84     Sequence | Switch | Counter;
85
86 Generator:
87     ChordGenerator | ArpeggioGenerator;
88
89 // ***** Metadata *****
90
91 Metadata:
92     Title | Composer | Poet | Copyright;

```

```

89 Title:
90     'title' title=STRING;
91
92 Composer:
93     'composer' composer=STRING;
94
95 Poet:
96     'poet' poet=STRING;
97
98 Copyright:
99     'copyright' copyright=STRING;
100
101 // ***** Definitions *****
102
103 Definition:
104     InstrumentDefinition | InstrumentSetDefinition | TonalSystemDefinition |
105     ScaleDefinition;
106
107 // ***** Instrument Definitions *****
108
109 InstrumentDefinition:
110     'instrumentDef' name=IDENTIFIER '{'
111
112     ('type' instrumentType=InstrumentType)?
113     & ('percussionMidiNote' percussionMidiNote=INT)?
114     & ('pitchRange' pitchRange=Range)?
115     & ('maxSimultaneousNotes' maxSimultaneousNotes=INT)?
116     & ('events' '{' eventParameters += InstrumentParameter+ '}' )?
117     & ('continuous' '{' continuousParameters += InstrumentParameter+ '}' )?
118     & ('scoreLabel' scoreLabel=STRING)?
119     & ('lilyPondInstrumentName' lilyPondInstrumentName=STRING)?
120     & ('defaultClef' defaultClef=Clef)?
121     & ('defaultOctave' defaultOctave=INT)?
122     & ('notationOctaveOffset' notationOctaveOffset=Expression)?
123
124     '}' ;
125
126 enum InstrumentType:
127     DEFAULT='default' | PERCUSSION='percussion' | SYNTH='synth';
128
129 InstrumentParameter:
130     modifiers += ParameterModifer* type=ParameterType name=IDENTIFIER range=Range (
131     'default' defaultValue=Expression)?;
132
133 Range:
134     '[' lowerBound=Expression '..' upperBound=Expression ']';
135
136 enum ParameterModifer:
137     OPTIONAL='optional';
138
139 enum ParameterType:
140     INT='int' | FLOAT='float' | BOOLEAN='boolean';
141
142 enum Clef:
143     TREBLE='treble' | ALTO='alto' | TENOR='tenor' | BASS='bass';
144
145 InstrumentParameterReference:
146     parameter=[InstrumentParameter|IDENTIFIER];

```

```

144
145 InstrumentSetDefinition:
146     'instrumentSetDef' name=IDENTIFIER
147     '{'
148     (instruments += [InstrumentDefinition|IDENTIFIER])+
149     '}' ;
150
151 // ***** Tonal Systems *****
152
153 TonalSystemDefinition:
154     'tonalSystemDef' name=IDENTIFIER
155     '{'
156     'stepsPerOctave' stepsPerOctave=INT
157     notes += TonalSystemNote*
158     midiMapping=TonalSystemMidiMapping
159     '}' ;
160
161 TonalSystemMidiMapping:
162     'midiMapping' 'step' step=INT '->' midiNote=INT;
163
164 TonalSystemNote:
165     'note' name=NOTENAME 'step' step=INT;
166
167 Lyrics:
168     'lyrics' lyrics=STRING;
169
170 CustomContext:
171     'customContext' name=ID value=Expression;
172
173
174 // ***** Scales *****
175
176 ScaleDefinition:
177     'scaleDef' name=IDENTIFIER '{'
178     'degrees' (degrees += INT)+
179     '}' ;
180
181 // ***** Control Structures *****
182
183 Fragment:
184     'fragment' name=IDENTIFIER ('scoreLabel' scoreLabel=STRING)? ('internalSectionId'
        internalSectionId=STRING)?;
185
186 FragmentReference:
187     'fragmentRef' part=[Fragment|IDENTIFIER];
188
189 Parallelization:
190     {Parallelization} 'parallel';
191
192 Repetition:
193     'repeat' times=Expression ('as' variableDeclaration=SimpleVariableDeclaration)?;
194
195 Sequence:
196     'sequence' times=Expression 'times' 'step' step=Expression ('mode'
        transpositionMode=IntervalMode)?;
197
198 Iteration:

```

```

199     'for' variable=SimpleVariableDeclaration 'in' startValue=Expression 'to' endValue
      =Expression ('step' step=Expression)?;
200
201 Condition:
202     'if' condition=Expression
203     '{'
204         trueNodes += Node*
205     '}'
206     (
207         'else'
208         '{'
209             falseNodes += Node*
210         '}'
211     )?
212     ;
213
214 While:
215     'while' condition=Expression
216     '{'
217         children += Node+
218     '}'
219     ;
220 Switch:
221     {Switch}'switch'
222     ('childIndexSequence' (childIndexSequence += INT)*)?;
223
224 Counter:
225     'counter' variableDeclaration=SimpleVariableDeclaration;
226
227 SimpleVariableDeclaration returns VariableDeclaration:
228     name=IDENTIFIER;
229
230 // ***** Contexts *****
231
232 HarmonicContext:
233     'harmony' harmony=AbstractHarmony;
234
235 AbstractHarmony:
236     Harmony | Figure;
237
238 Harmony:
239     ((root=OctavelessNoteReference (minor ?= 'm')?))
240     (additions += HarmonyAddition)*
241     ('/' (bassNote=OctavelessNoteReference))?
242     (resolutionHints += HarmonyResolutionHint*);
243
244 HarmonyResolutionHint:
245     NoteExclusion;
246
247 NoteExclusion:
248     '- ' noteReference=NoteReference;
249
250 Figure:
251     (rootAccidental = Accidental)?
252     rootFigure=RomanNumeral
253     (additions += AdditionalHarmonyNote)*
254     ('/' (bassAccidental = Accidental)? bassFigure=RomanNumeral)?
255     ;

```

```

256
257 enum RomanNumeral:
258     TONICMAJOR='I' | TONICMINOR='i' |
259     SECONDMAJOR='II' | SECONDMINOR='ii' |
260     THIRDMAJOR='III' | THIRDMINOR='iii' |
261     FOURTHMAJOR='IV' | FOURTHMINOR='iv' |
262     FIFTHMAJOR='V' | FIFTHMINOR='v' |
263     SIXTHMAJOR='VI' | SIXTHMINOR='vi' |
264     SEVENTHMAJOR='VII' | SEVENTHMINOR='vii';
265
266 HarmonyAddition:
267     AdditionalHarmonyNote | HarmonySpecification;
268
269 AdditionalHarmonyNote:
270     (accidental=Accidental)? step=INT;
271
272 Accidental: '#' | 'b';
273
274 HarmonySpecification:
275     chordSymbol=ChordSymbol;
276
277 enum ChordSymbol:
278     MAJ7='maj7' | SUS4='sus4' | DIMINISHED='°' | AUGMENTED='+' | POWER='p' | SUS2='
279     sus2' | M7='m7';
280
281 Key:
282     'key' harmony=Harmony;
283
284 HarmonicProgression:
285     'harmonicProgression' harmonies += AbstractHarmony+;
286
287 Rhythm:
288     'rhythm' (partial=Partial (notes += RhythmicNote)* | (notes += RhythmicNote)+);
289
290 HarmonicRhythm:
291     'harmonicRhythm' notes += RhythmicNote+;
292
293 RhythmicNote:
294     RhythmNote | Rest | Tuplet;
295
296 RhythmNote:
297     noteDuration=NoteDuration (tiedToNextNote ?= '~')?;
298
299 Rest:
300     '_' noteDuration=NoteDuration;
301
302 Tuplet:
303     '(' numActualNotes=INT '/' numInsteadOfNotes=INT ':' notes += RhythmicNote+ ')';
304
305 Partial:
306     '(' notes += RhythmicNote+ ')';
307
308 NoteDuration:
309     CanonicalNoteDuration | FractionNoteDuration;
310
311 /*
312  * Integer value means reciprocal value, e.g. 4 for a quarter note.

```

```

312 * If an exclamation mark is added, it means no reciprocal (e.g. 2! means a
    duration of 2 whole notes)
313 * A dotted note is expanded to the 1.5-fold duration (can be applied multiple
    times)
314 */
315 CanonicalNoteDuration:
316     number=INT (notReciprocal ?= '!')? (dots = Dots)?;
317
318 // need an enum here due to ambiguity problems in grammar, ('.')+ does not work
319 enum Dots:
320     ONE='.' | TWO='..' | THREE='...' | FOUR='....';
321
322 /*
323 * Alternatively, the note length can be given as fraction (e.g. 3/4).
324 */
325 FractionNoteDuration:
326     numerator=INT '/' denominator=INT;
327
328 TimeSignature:
329     'time' numerator=INT '/' denominator=INT;
330
331 Tempo:
332     'tempo' startTempo=Expression ('->' endTempo=Expression)? ('noteDuration'
        noteDuration=CanonicalNoteDuration)?;
333
334 Instrument:
335     'instrument' instrumentDefinition=[InstrumentDefinition|IDENTIFIER] ('staffId'
        staffId=STRING)?;
336
337 TonalSystem:
338     'tonalSystem' tonalSystem=[TonalSystemDefinition|IDENTIFIER];
339
340 Pitches:
341     'pitches' ((' (parameters += PitchParameter)+ ')')? pitches += Pitch+;
342
343 PitchParameter:
344     HarmonicReference | StartOctave | FindNearestOctave;
345
346 FindNearestOctave:
347     'findNearestOctave' findNearestOctave ?= 'true';
348
349 HarmonicReference:
350     'relative' 'to' harmonyReference=HarmonyReference;
351
352 enum HarmonyReference:
353     KEY='key' | HARMONY='harmony';
354
355 Pitch:
356     SinglePitch | Chord;
357
358 SinglePitch:
359     NoteReference | DegreeReference | NoteExpression;
360
361 NoteExpression:
362     '@' expression=Expression;
363
364 DegreeReference:
365     degree=INT (accidental=Accidental)? (octave=Octave)?;

```

```

366
367 Chord:
368     '[' chordNotes += SinglePitch+ ']' ;
369
370 NoteReference:
371     tonalSystemNote=[TonalSystemNote|NOTENAME] (octave=Octave)?;
372
373 OctavelessNoteReference:
374     tonalSystemNote=[TonalSystemNote|NOTENAME];
375
376 Octave:
377     '_' number=INT;
378
379 DynamicContext:
380     LoudnessContext | SuddenDynamics;
381
382 LoudnessContext:
383     'loudness' loudness=Loudness ('->' targetLoudness=Loudness)?;
384
385 enum Loudness:
386     CURRENT='current' | SILENCE='silence' | FORTE='f' | FORTISSIMO='ff' |
387         FORTFORTISSIMO='fff' | FFFF='ffff' | PIANO='p' | PIANISSIMO='pp' |
388         PIANOPIANISSIMO='ppp' | PPPP='pppp' | MEZZOFORTE='mf' | MEZZOPIANO='mp';
389
390 SuddenDynamics:
391     'sudden' type=SuddenDynamicsType;
392
393 enum SuddenDynamicsType:
394     SFORZATO='sf' | RINFORZANDO='rf' | FORTEPIANO='fp';
395
396 Scale:
397     'scale' scale=[ScaleDefinition|IDENTIFIER];
398
399 Envelope:
400     'envelope' 'for' instrumentParameterReference=InstrumentParameterReference '
401         startValue' startValue=Expression
402         points += EnvelopePoint+;
403
404 EnvelopePoint:
405     'point' value=Expression 'time' time=Expression ('curve' curve=InterpolationType)
406         ?;
407
408 enum InterpolationType:
409     STEP='step' | LINEAR='linear' | EXPONENTIAL='exponential' ;
410
411 // ***** Modifiers *****
412
413 // Tonal Modifiers
414
415 Transposition:
416     'transpose' ('mode' transpositionMode=IntervalMode)? interval=Expression;
417
418 enum IntervalMode:
419     ABSOLUTE='absolute' | INSCALE='inScale' | OCTAVES='octaves';
420
421 Inversion:
422     'inversion' referenceDegree=Expression;

```

```

420
421 ParallelInterval:
422     'parallelInterval' ('mode' mode=IntervalMode)? interval=Expression;
423
424 // Rhythmic Modifiers
425
426 RhythmicDisplacement:
427     'rhythmicDisplacement' ('mode' mode=DisplacementMode)? 'offset' offset=Expression
428     ;
429
430 enum DisplacementMode:
431     DISCARD='discard' | WRAP='wrap';
432
433 RhythmicInsertion:
434     'rhythmicInsertion' ('mode' mode=RhythmicInsertionMode)? 'offset' offset=
435     NoteDuration rhythm=Rhythm ;
436
437 enum RhythmicInsertionMode:
438     INSERT='insert' | OVERWRITE='overwrite';
439
440 Augmentation:
441     {Augmentation} 'augmentation' (factor=Factor)?;
442
443 Diminution:
444     {Diminution} 'diminution' (factor=Factor)?;
445
446 Factor:
447     'factor' expression=Expression;
448
449 RhythmicExtension:
450     'rhythmicExtension' duration=NoteDuration;
451
452 RhythmicAdjustment:
453     'rhythmicAdjustment'
454     (
455         ('startDelta' startDelta=NoteDuration) |
456         ('endDelta' endDelta=NoteDuration) |
457         ('startDelta' startDelta=NoteDuration 'endDelta' endDelta=NoteDuration)
458     );
459
460 // Harmonic Modifiers
461
462 HarmonyModifier:
463     'harmonyModifier' harmonyAdditions += HarmonyAddition+;
464
465 // ***** Generators *****
466
467 StartOctave:
468     'startOctave' octave=Expression;
469
470 ChordGenerator:
471     {ChordGenerator} 'chordGenerator'
472     (
473         ('numberOfNotes' numberOfNotes=Expression)?
474         & ('includeBassNote' includeBassNote ?= 'true')?
475         & ('startInversion' startInversion=Expression)?
476         & ('findNearestInversion' findNearestInversion=BooleanLiteral)?

```

```

476         & ('startOctave' startOctave=Expression)?
477         & ('resetOnContextChange' resetOnContextChange?='true')?
478     );
479
480 ArpeggioGenerator:
481     {ArpeggioGenerator} 'arpeggioGenerator'
482
483     (
484         ('numberOfNotes' numberOfNotes=Expression)?
485         & ('includeBassNote' includeBassNote ?= 'true')?
486         & ('startInversion' startInversion=Expression)?
487         & ('findNearestInversion' findNearestInversion=BooleanLiteral)?
488         & ('startOctave' startOctave=Expression)?
489         & ('resetOnContextChange' resetOnContextChange?='true')?
490         & ('noteIndexSequence' noteIndexSequence += INT+)?
491     );
492
493 // ***** Expression Language *****
494
495 Expression:
496     BooleanExpression;
497
498 BooleanExpression returns Expression:
499     Comparison
500     (({AndOrExpression.left=current} op=BooleanOperator) right=Comparison)*;
501
502 BooleanOperator:
503     "or" | "and";
504
505 Comparison returns Expression:
506     Equals
507     (({Comparison.left=current} op=ComparisonOperator ) right=Equals)*;
508
509 ComparisonOperator:
510     "<" | ">" | "<=" | ">=";
511
512 Equals returns Expression:
513     Addition
514     (({Equals.left=current} op=EqualsOperator ) right=Addition)*;
515
516 EqualsOperator:
517     '==' | '!=';
518
519 Addition returns Expression:
520     Multiplication
521     (({Addition.left=current} op=AdditiveOperator) right=Multiplication)*;
522
523 AdditiveOperator:
524     '+' | '-';
525
526 Multiplication returns Expression:
527     UnaryExpression (({Multiplication.left=current} op=MultiplicativeOperator)
528         right=UnaryExpression)*;
529
530 MultiplicativeOperator:
531     "*" | "/" | "%";
532
533 UnaryExpression returns Expression:

```

```

533     {BooleanNegation} =>"!" expression=UnaryExpression | /* right associativity,
        can be recursive */
534     {ArithmeticSigned} =>"-" expression=Atomic | /* right associativity */
535     Atomic;
536
537 Atomic returns Expression:
538     '(' Expression ')' |
539     Literal |
540     VariableReference |
541     FunctionCall;
542
543 FunctionCall:
544     name=IDENTIFIER '(' (parameters += Expression (',' parameters += Expression)*)? ')' ;
545
546 Literal:
547     IntegerLiteral |
548     FloatLiteral |
549     StringLiteral |
550     BooleanLiteral;
551
552 VariableReference:
553     variable=[VariableDeclaration|IDENTIFIER];
554
555 IntegerLiteral:
556     value=INT;
557
558 StringLiteral:
559     value=STRING;
560
561 FloatLiteral:
562     value=FLOAT;
563
564 BooleanLiteral:
565     {BooleanLiteral} ((value != 'true') | 'false');

```

Listing B.1: Musical Context Composition Language Grammar

B.2 MusicXML Code Example

The following listing contains the MusicXML code of J. S. Bach's *Sinfonia 1*, BWV 787, measures one and two.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.0 Partwise//EN" "http
   ://www.musicxml.org/dtds/partwise.dtd">
3 <score-partwise>
4   <work>
5     <work-title>Sinfonia 1</work-title>
6   </work>
7   <identification>
8     <creator type="composer">J. S. Bach</creator>
9     <encoding>
10       <software>MuseScore 2.1.0</software>
11       <encoding-date>2018-03-13</encoding-date>

```

```

12     <supports element="accidental" type="yes"/>
13     <supports element="beam" type="yes"/>
14     <supports element="print" attribute="new-page" type="yes" value="yes"/>
15     <supports element="print" attribute="new-system" type="yes" value="yes"/>
16     <supports element="stem" type="yes"/>
17     </encoding>
18 </identification>
19 <defaults>
20   <scaling>
21     <millimeters>7.05556</millimeters>
22     <tenths>40</tenths>
23   </scaling>
24   <page-layout>
25     <page-height>1683.36</page-height>
26     <page-width>1190.88</page-width>
27     <page-margins type="even">
28       <left-margin>56.6929</left-margin>
29       <right-margin>56.6929</right-margin>
30       <top-margin>56.6929</top-margin>
31       <bottom-margin>113.386</bottom-margin>
32     </page-margins>
33     <page-margins type="odd">
34       <left-margin>56.6929</left-margin>
35       <right-margin>56.6929</right-margin>
36       <top-margin>56.6929</top-margin>
37       <bottom-margin>113.386</bottom-margin>
38     </page-margins>
39   </page-layout>
40   <word-font font-family="FreeSerif" font-size="10"/>
41   <lyric-font font-family="FreeSerif" font-size="11"/>
42 </defaults>
43 <credit page="1">
44   <credit-words default-x="595.44" default-y="1626.67" justify="center" valign="
45     top" font-size="24">Sinfonia 1</credit-words>
46   </credit>
47 <credit page="1">
48   <credit-words default-x="595.44" default-y="1569.97" justify="center" valign="
49     top" font-size="14">BWV 787</credit-words>
50   </credit>
51 <credit page="1">
52   <credit-words default-x="1134.19" default-y="1526.67" justify="right" valign="
53     bottom" font-size="12">J. S. Bach</credit-words>
54   </credit>
55 <part-list>
56   <score-part id="P1">
57     <part-name>Piano</part-name>
58     <part-abbreviation>Pno.</part-abbreviation>
59     <score-instrument id="P1-I1">
60       <instrument-name>Piano</instrument-name>
61     </score-instrument>
62     <midi-device id="P1-I1" port="1"></midi-device>
63     <midi-instrument id="P1-I1">
64       <midi-channel>1</midi-channel>
65       <midi-program>1</midi-program>
66       <volume>78.7402</volume>
67       <pan>0</pan>
68     </midi-instrument>
69   </score-part>

```

```

67     </part-list>
68 <part id="P1">
69     <measure number="1" width="523.57">
70         <print>
71             <system-layout>
72                 <system-margins>
73                     <left-margin>21.00</left-margin>
74                     <right-margin>-0.00</right-margin>
75                 </system-margins>
76                 <top-system-distance>170.00</top-system-distance>
77             </system-layout>
78             <staff-layout number="2">
79                 <staff-distance>65.00</staff-distance>
80             </staff-layout>
81         </print>
82         <attributes>
83             <divisions>8</divisions>
84             <key>
85                 <fifths>0</fifths>
86             </key>
87             <time symbol="common">
88                 <beats>4</beats>
89                 <beat-type>4</beat-type>
90             </time>
91             <staves>2</staves>
92             <clef number="1">
93                 <sign>G</sign>
94                 <line>2</line>
95             </clef>
96             <clef number="2">
97                 <sign>F</sign>
98                 <line>4</line>
99             </clef>
100         </attributes>
101         <note>
102             <rest/>
103             <duration>2</duration>
104             <voice>1</voice>
105             <type>16th</type>
106             <staff>1</staff>
107         </note>
108         <note default-x="107.21" default-y="-30.00">
109             <pitch>
110                 <step>G</step>
111                 <octave>4</octave>
112             </pitch>
113             <duration>2</duration>
114             <voice>1</voice>
115             <type>16th</type>
116             <stem>up</stem>
117             <staff>1</staff>
118             <beam number="1">begin</beam>
119             <beam number="2">begin</beam>
120         </note>
121         <note default-x="134.86" default-y="-25.00">
122             <pitch>
123                 <step>A</step>
124                 <octave>4</octave>

```

```

125         </pitch>
126         <duration>2</duration>
127         <voice>1</voice>
128         <type>16th</type>
129         <stem>up</stem>
130         <staff>1</staff>
131         <beam number="1">continue</beam>
132         <beam number="2">continue</beam>
133     </note>
134 <note default-x="162.51" default-y="-20.00">
135     <pitch>
136         <step>B</step>
137         <octave>4</octave>
138     </pitch>
139     <duration>2</duration>
140     <voice>1</voice>
141     <type>16th</type>
142     <stem>up</stem>
143     <staff>1</staff>
144     <beam number="1">end</beam>
145     <beam number="2">end</beam>
146 </note>
147 <note default-x="190.16" default-y="-15.00">
148     <pitch>
149         <step>C</step>
150         <octave>5</octave>
151     </pitch>
152     <duration>2</duration>
153     <voice>1</voice>
154     <type>16th</type>
155     <stem>down</stem>
156     <staff>1</staff>
157     <beam number="1">begin</beam>
158     <beam number="2">begin</beam>
159 </note>
160 <note default-x="217.81" default-y="-10.00">
161     <pitch>
162         <step>D</step>
163         <octave>5</octave>
164     </pitch>
165     <duration>2</duration>
166     <voice>1</voice>
167     <type>16th</type>
168     <stem>down</stem>
169     <staff>1</staff>
170     <beam number="1">continue</beam>
171     <beam number="2">continue</beam>
172 </note>
173 <note default-x="245.46" default-y="-5.00">
174     <pitch>
175         <step>E</step>
176         <octave>5</octave>
177     </pitch>
178     <duration>2</duration>
179     <voice>1</voice>
180     <type>16th</type>
181     <stem>down</stem>
182     <staff>1</staff>

```

```

183     <beam number="1">continue</beam>
184     <beam number="2">continue</beam>
185 </note>
186 <note default-x="273.11" default-y="0.00">
187   <pitch>
188     <step>F</step>
189     <octave>5</octave>
190   </pitch>
191   <duration>2</duration>
192   <voice>1</voice>
193   <type>16th</type>
194   <stem>down</stem>
195   <staff>1</staff>
196   <beam number="1">end</beam>
197   <beam number="2">end</beam>
198 </note>
199 <note default-x="300.76" default-y="5.00">
200   <pitch>
201     <step>G</step>
202     <octave>5</octave>
203   </pitch>
204   <duration>2</duration>
205   <voice>1</voice>
206   <type>16th</type>
207   <stem>down</stem>
208   <staff>1</staff>
209   <beam number="1">begin</beam>
210   <beam number="2">begin</beam>
211 </note>
212 <note default-x="328.41" default-y="0.00">
213   <pitch>
214     <step>F</step>
215     <octave>5</octave>
216   </pitch>
217   <duration>2</duration>
218   <voice>1</voice>
219   <type>16th</type>
220   <stem>down</stem>
221   <staff>1</staff>
222   <beam number="1">continue</beam>
223   <beam number="2">continue</beam>
224 </note>
225 <note default-x="356.07" default-y="5.00">
226   <pitch>
227     <step>G</step>
228     <octave>5</octave>
229   </pitch>
230   <duration>2</duration>
231   <voice>1</voice>
232   <type>16th</type>
233   <stem>down</stem>
234   <staff>1</staff>
235   <beam number="1">continue</beam>
236   <beam number="2">continue</beam>
237 </note>
238 <note default-x="383.72" default-y="10.00">
239   <pitch>
240     <step>A</step>

```

```

241         <octave>5</octave>
242     </pitch>
243     <duration>2</duration>
244     <voice>1</voice>
245     <type>16th</type>
246     <stem>down</stem>
247     <staff>1</staff>
248     <beam number="1">end</beam>
249     <beam number="2">end</beam>
250 </note>
251 <note default-x="411.37" default-y="0.00">
252     <pitch>
253         <step>F</step>
254         <octave>5</octave>
255     </pitch>
256     <duration>2</duration>
257     <voice>1</voice>
258     <type>16th</type>
259     <stem>down</stem>
260     <staff>1</staff>
261     <beam number="1">begin</beam>
262     <beam number="2">begin</beam>
263 </note>
264 <note default-x="439.02" default-y="10.00">
265     <pitch>
266         <step>A</step>
267         <octave>5</octave>
268     </pitch>
269     <duration>2</duration>
270     <voice>1</voice>
271     <type>16th</type>
272     <stem>down</stem>
273     <staff>1</staff>
274     <beam number="1">continue</beam>
275     <beam number="2">continue</beam>
276 </note>
277 <note default-x="466.67" default-y="5.00">
278     <pitch>
279         <step>G</step>
280         <octave>5</octave>
281     </pitch>
282     <duration>2</duration>
283     <voice>1</voice>
284     <type>16th</type>
285     <stem>down</stem>
286     <staff>1</staff>
287     <beam number="1">continue</beam>
288     <beam number="2">continue</beam>
289 </note>
290 <note default-x="494.32" default-y="0.00">
291     <pitch>
292         <step>F</step>
293         <octave>5</octave>
294     </pitch>
295     <duration>2</duration>
296     <voice>1</voice>
297     <type>16th</type>
298     <stem>down</stem>

```

```

299     <staff>1</staff>
300     <beam number="1">end</beam>
301     <beam number="2">end</beam>
302   </note>
303   <backup>
304     <duration>32</duration>
305   </backup>
306   <note default-x="79.56" default-y="-130.00">
307     <pitch>
308       <step>C</step>
309       <octave>3</octave>
310     </pitch>
311     <duration>8</duration>
312     <voice>5</voice>
313     <type>quarter</type>
314     <stem>up</stem>
315     <staff>2</staff>
316   </note>
317   <note>
318     <rest/>
319     <duration>4</duration>
320     <voice>5</voice>
321     <type>eighth</type>
322     <staff>2</staff>
323   </note>
324   <note default-x="245.46" default-y="-95.00">
325     <pitch>
326       <step>C</step>
327       <octave>4</octave>
328     </pitch>
329     <duration>4</duration>
330     <voice>5</voice>
331     <type>eighth</type>
332     <stem>down</stem>
333     <staff>2</staff>
334   </note>
335   <note default-x="300.76" default-y="-100.00">
336     <pitch>
337       <step>B</step>
338       <octave>3</octave>
339     </pitch>
340     <duration>4</duration>
341     <voice>5</voice>
342     <type>eighth</type>
343     <stem>down</stem>
344     <staff>2</staff>
345     <beam number="1">begin</beam>
346   </note>
347   <note default-x="356.07" default-y="-110.00">
348     <pitch>
349       <step>G</step>
350       <octave>3</octave>
351     </pitch>
352     <duration>4</duration>
353     <voice>5</voice>
354     <type>eighth</type>
355     <stem>down</stem>
356     <staff>2</staff>

```

```

357     <beam number="1">continue</beam>
358   </note>
359 <note default-x="411.37" default-y="-105.00">
360   <pitch>
361     <step>A</step>
362     <octave>3</octave>
363   </pitch>
364   <duration>4</duration>
365   <voice>5</voice>
366   <type>eighth</type>
367   <stem>down</stem>
368   <staff>2</staff>
369   <beam number="1">continue</beam>
370 </note>
371 <note default-x="466.67" default-y="-100.00">
372   <pitch>
373     <step>B</step>
374     <octave>3</octave>
375   </pitch>
376   <duration>4</duration>
377   <voice>5</voice>
378   <type>eighth</type>
379   <stem>down</stem>
380   <staff>2</staff>
381   <beam number="1">end</beam>
382 </note>
383 </measure>
384 <measure number="2" width="532.92">
385   <note default-x="12.64" default-y="-5.00">
386     <pitch>
387       <step>E</step>
388       <octave>5</octave>
389     </pitch>
390     <duration>16</duration>
391     <tie type="start"/>
392     <voice>1</voice>
393     <type>half</type>
394     <stem>up</stem>
395     <staff>1</staff>
396     <notations>
397       <tied type="start"/>
398     </notations>
399   </note>
400   <note default-x="259.84" default-y="-5.00">
401     <pitch>
402       <step>E</step>
403       <octave>5</octave>
404     </pitch>
405     <duration>8</duration>
406     <tie type="stop"/>
407     <voice>1</voice>
408     <type>quarter</type>
409     <stem>up</stem>
410     <staff>1</staff>
411     <notations>
412       <tied type="stop"/>
413     </notations>
414   </note>

```

```

415 <note default-x="391.19" default-y="0.00">
416   <pitch>
417     <step>F</step>
418     <alter>1</alter>
419     <octave>5</octave>
420   </pitch>
421   <duration>8</duration>
422   <voice>1</voice>
423   <type>quarter</type>
424   <accidental>sharp</accidental>
425   <stem>up</stem>
426   <staff>1</staff>
427 </note>
428 <backup>
429   <duration>32</duration>
430 </backup>
431 <note>
432   <rest>
433     <display-step>C</display-step>
434     <display-octave>4</display-octave>
435   </rest>
436   <duration>2</duration>
437   <voice>2</voice>
438   <type>16th</type>
439   <staff>1</staff>
440 </note>
441 <note default-x="43.85" default-y="-50.00">
442   <pitch>
443     <step>C</step>
444     <octave>4</octave>
445   </pitch>
446   <duration>2</duration>
447   <voice>2</voice>
448   <type>16th</type>
449   <stem>down</stem>
450   <staff>1</staff>
451   <beam number="1">begin</beam>
452   <beam number="2">begin</beam>
453 </note>
454 <note default-x="74.71" default-y="-45.00">
455   <pitch>
456     <step>D</step>
457     <octave>4</octave>
458   </pitch>
459   <duration>2</duration>
460   <voice>2</voice>
461   <type>16th</type>
462   <stem>down</stem>
463   <staff>1</staff>
464   <beam number="1">continue</beam>
465   <beam number="2">continue</beam>
466 </note>
467 <note default-x="105.56" default-y="-40.00">
468   <pitch>
469     <step>E</step>
470     <octave>4</octave>
471   </pitch>
472   <duration>2</duration>

```

```

473     <voice>2</voice>
474     <type>16th</type>
475     <stem>down</stem>
476     <staff>1</staff>
477     <beam number="1">end</beam>
478     <beam number="2">end</beam>
479     </note>
480 <note default-x="136.42" default-y="-35.00">
481     <pitch>
482         <step>F</step>
483         <octave>4</octave>
484     </pitch>
485     <duration>2</duration>
486     <voice>2</voice>
487     <type>16th</type>
488     <stem>down</stem>
489     <staff>1</staff>
490     <beam number="1">begin</beam>
491     <beam number="2">begin</beam>
492     </note>
493 <note default-x="167.27" default-y="-30.00">
494     <pitch>
495         <step>G</step>
496         <octave>4</octave>
497     </pitch>
498     <duration>2</duration>
499     <voice>2</voice>
500     <type>16th</type>
501     <stem>down</stem>
502     <staff>1</staff>
503     <beam number="1">continue</beam>
504     <beam number="2">continue</beam>
505     </note>
506 <note default-x="198.13" default-y="-25.00">
507     <pitch>
508         <step>A</step>
509         <octave>4</octave>
510     </pitch>
511     <duration>2</duration>
512     <voice>2</voice>
513     <type>16th</type>
514     <stem>down</stem>
515     <staff>1</staff>
516     <beam number="1">continue</beam>
517     <beam number="2">continue</beam>
518     </note>
519 <note default-x="228.98" default-y="-20.00">
520     <pitch>
521         <step>B</step>
522         <octave>4</octave>
523     </pitch>
524     <duration>2</duration>
525     <voice>2</voice>
526     <type>16th</type>
527     <stem>down</stem>
528     <staff>1</staff>
529     <beam number="1">end</beam>
530     <beam number="2">end</beam>

```

```

531     </note>
532 <note default-x="259.84" default-y="-15.00">
533     <pitch>
534         <step>C</step>
535         <octave>5</octave>
536     </pitch>
537     <duration>2</duration>
538     <voice>2</voice>
539     <type>16th</type>
540     <stem>down</stem>
541     <staff>1</staff>
542     <beam number="1">begin</beam>
543     <beam number="2">begin</beam>
544 </note>
545 <note default-x="290.69" default-y="-20.00">
546     <pitch>
547         <step>B</step>
548         <octave>4</octave>
549     </pitch>
550     <duration>2</duration>
551     <voice>2</voice>
552     <type>16th</type>
553     <stem>down</stem>
554     <staff>1</staff>
555     <beam number="1">continue</beam>
556     <beam number="2">continue</beam>
557 </note>
558 <note default-x="321.54" default-y="-15.00">
559     <pitch>
560         <step>C</step>
561         <octave>5</octave>
562     </pitch>
563     <duration>1</duration>
564     <voice>2</voice>
565     <type>32nd</type>
566     <stem>down</stem>
567     <staff>1</staff>
568     <beam number="1">continue</beam>
569     <beam number="2">continue</beam>
570     <beam number="3">begin</beam>
571 </note>
572 <note default-x="340.83" default-y="-20.00">
573     <pitch>
574         <step>B</step>
575         <octave>4</octave>
576     </pitch>
577     <duration>1</duration>
578     <voice>2</voice>
579     <type>32nd</type>
580     <stem>down</stem>
581     <staff>1</staff>
582     <beam number="1">continue</beam>
583     <beam number="2">continue</beam>
584     <beam number="3">end</beam>
585 </note>
586 <note default-x="360.11" default-y="-10.00">
587     <pitch>
588         <step>D</step>

```

```

589         <octave>5</octave>
590     </pitch>
591     <duration>2</duration>
592     <voice>2</voice>
593     <type>16th</type>
594     <stem>down</stem>
595     <staff>1</staff>
596     <beam number="1">end</beam>
597     <beam number="2">end</beam>
598 </note>
599 <note default-x="391.19" default-y="-15.00">
600     <pitch>
601         <step>C</step>
602         <octave>5</octave>
603     </pitch>
604     <duration>2</duration>
605     <voice>2</voice>
606     <type>16th</type>
607     <stem>down</stem>
608     <staff>1</staff>
609     <beam number="1">begin</beam>
610     <beam number="2">begin</beam>
611 </note>
612 <note default-x="422.05" default-y="-5.00">
613     <pitch>
614         <step>E</step>
615         <octave>5</octave>
616     </pitch>
617     <duration>2</duration>
618     <voice>2</voice>
619     <type>16th</type>
620     <stem>down</stem>
621     <staff>1</staff>
622     <beam number="1">continue</beam>
623     <beam number="2">continue</beam>
624 </note>
625 <note default-x="452.90" default-y="-5.00">
626     <pitch>
627         <step>E</step>
628         <octave>5</octave>
629     </pitch>
630     <duration>1</duration>
631     <voice>2</voice>
632     <type>32nd</type>
633     <stem>down</stem>
634     <staff>1</staff>
635     <beam number="1">continue</beam>
636     <beam number="2">continue</beam>
637     <beam number="3">begin</beam>
638 </note>
639 <note default-x="472.19" default-y="-10.00">
640     <pitch>
641         <step>D</step>
642         <octave>5</octave>
643     </pitch>
644     <duration>1</duration>
645     <voice>2</voice>
646     <type>32nd</type>

```

```

647     <stem>down</stem>
648     <staff>1</staff>
649     <beam number="1">continue</beam>
650     <beam number="2">continue</beam>
651     <beam number="3">end</beam>
652   </note>
653 <note default-x="491.47" default-y="-15.00">
654   <pitch>
655     <step>C</step>
656     <octave>5</octave>
657   </pitch>
658   <duration>2</duration>
659   <voice>2</voice>
660   <type>16th</type>
661   <stem>down</stem>
662   <staff>1</staff>
663   <beam number="1">end</beam>
664   <beam number="2">end</beam>
665 </note>
666 <backup>
667   <duration>32</duration>
668 </backup>
669 <note default-x="13.00" default-y="-95.00">
670   <pitch>
671     <step>C</step>
672     <octave>4</octave>
673   </pitch>
674   <duration>8</duration>
675   <voice>5</voice>
676   <type>quarter</type>
677   <stem>down</stem>
678   <staff>2</staff>
679 </note>
680 <note>
681   <rest />
682   <duration>4</duration>
683   <voice>5</voice>
684   <type>eighth</type>
685   <staff>2</staff>
686 </note>
687 <note default-x="198.13" default-y="-100.00">
688   <pitch>
689     <step>B</step>
690     <octave>3</octave>
691   </pitch>
692   <duration>4</duration>
693   <voice>5</voice>
694   <type>eighth</type>
695   <stem>down</stem>
696   <staff>2</staff>
697 </note>
698 <note default-x="259.84" default-y="-105.00">
699   <pitch>
700     <step>A</step>
701     <octave>3</octave>
702   </pitch>
703   <duration>4</duration>
704   <voice>5</voice>

```

```

705     <type>eighth</type>
706     <stem>down</stem>
707     <staff>2</staff>
708     <beam number="1">begin</beam>
709     </note>
710     <note default-x="321.54" default-y="-110.00">
711         <pitch>
712             <step>G</step>
713             <octave>3</octave>
714         </pitch>
715         <duration>4</duration>
716         <voice>5</voice>
717         <type>eighth</type>
718         <stem>down</stem>
719         <staff>2</staff>
720         <beam number="1">continue</beam>
721     </note>
722     <note default-x="391.19" default-y="-105.00">
723         <pitch>
724             <step>A</step>
725             <octave>3</octave>
726         </pitch>
727         <duration>4</duration>
728         <voice>5</voice>
729         <type>eighth</type>
730         <stem>down</stem>
731         <staff>2</staff>
732         <beam number="1">continue</beam>
733     </note>
734     <note default-x="452.90" default-y="-125.00">
735         <pitch>
736             <step>D</step>
737             <octave>3</octave>
738         </pitch>
739         <duration>4</duration>
740         <voice>5</voice>
741         <type>eighth</type>
742         <stem>down</stem>
743         <staff>2</staff>
744         <beam number="1">end</beam>
745     </note>
746     <barline location="right">
747         <bar-style>light-heavy</bar-style>
748     </barline>
749 </measure>
750 </part>
751 </score-partwise>

```

Listing B.2: MusicXML representation of J. S. Bach, *Sinfonia 1*, BWV 787, mm. 1–2