



City Research Online

City St George's, University of London

Citation: Bergeron, M. (2010). Structured polyphonic patterns. (Unpublished Doctoral thesis, City University London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://city-ref-test.eprints-hosting.org/id/eprint/12097/>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

STRUCTURED POLYPHONIC PATTERNS

Mathieu Bergeron

Submitted for Examination of Doctor of Philosophy

Department of Computing
City University London
United Kingdom

May 31, 2010

À Lise et Roger...

Acknowledgments

I am infinitely indebted to my supervisor Darrell Conklin for his sustained enthusiasm and patience – and for his considerable help with the content of this dissertation and administrative matters.

I also wish to thank Christos Kloukinas and Alan Marsden for kindly accepting to examine this dissertation. I am especially proud and touched that my work has been reviewed by Dr Marsden. His book on musical time was my first contact with the field of music informatics: it provided the spark that initiated my research project and constant inspiration as I revisited – and keep revisiting – the book.

I thank my colleagues of the Music Informatics research group at City University: Tillman Weyde, Jens Wissmann, Kerstin Neubarth, Aline Honingh, Ruben Hillewaere, Michael Rausch and Édouard Gilbert. I will keep fond memories of many work sessions, reading group meetings, retreats... and teatime chats. Additional thanks to Christina Anagnostopoulou for her generous comments and encouragements early on.

I am also grateful to City University and FQRNT (Le Fonds québécois de la recherche sur la nature et les technologies) for their financial support.

Finally, I wish to thank Katherine Elliott for kindly accepting to proofread this dissertation.

* * *

On a more personal note, I want to warmly thank Lise and Roger who supported me greatly – and in many ways – throughout my doctoral studies. I will particularly remember their swift encouragements at the beginning of the process and their inextinguishable patience at the end.

I also wish to thank all my friends who graciously put up with the usual mood circumvolutions of a PhD student. In particular: Jens, Matilda, Helena, Ionana and

Theo, Manon, both François, Julien, Raveca, Nathalie, Marie-Pierre and Éliane. A very special mention to Éliane who shared part of my adventure in London. Some additional thanks to François (Perron) and Jens for their renewed enthusiasm towards ideas developed in this dissertation – and perhaps more importantly towards crazy ideas *not* developed here. Some additional thanks also to Theo and Ioana for their hospitality and heartwarming generosity.

Final thanks to *De L'onga* – Andrew and Pierre-Alexandre – whose album *À l'Abri du Temps* has been a trusty auditory companion throughout writing up this dissertation.

Declaration

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only copies made for study purposes, subject to normal conditions of acknowledgment.

Abstract

The present dissertation develops, applies and evaluates a novel method for the representation and retrieval of patterns in musical data. The method supports the typical polyphonic patterns that one finds in music theory textbooks. Most current computational methods to musical patterns are restricted to monophony (one melody at a time). The Structured Polyphonic Patterns method (*SPP*) applies to the general case of polyphonic music, where many melodies may unfold concurrently. Pattern components are conjunctions of features which encode properties of musical events, or relations that they form with other events. Relations between events that overlap in time but are not simultaneous are supported, enabling patterns to express key temporal relations of polyphonic music. Patterns are formed by joining and layering pattern components into sequences (horizontal structures) and layers (vertical structures). A layer specifies voicing in an abstract way, and the exploration of different voice permutations is handled automatically. The *SPP* method also provides a mechanism for defining new features. We evaluate *SPP* by developing a small catalog of musicologically relevant queries and analyzing the results on four corpora: 185 chorale harmonizations by J.S. Bach, Mozart Symphony no. 40, a small set of piano pieces by Chopin, and a collection of folk songs containing more than 8000 pieces – in addition to its size, demonstrating the scalability of the method, that latter corpus is interesting as it shows that *SPP* is also usable for monophony. Examining several corpora allows us to establish that some polyphonic patterns constitute salient properties of a corpus: they are over-represented in one corpus by comparison to the others. In addition, the queries we develop demonstrate that the *SPP* method possesses sufficient expressiveness to capture important music-theoretic notions. At the same time, we show how the method is more restrictive than some existing polyphonic pattern representations, hence providing a better approximation of the expressive power required for polyphonic patterns. It is a better candidate representation for music data mining, a difficult problem that has received significant attention for the monophonic case, but limited attention for

the more general polyphonic case.

Contents

1	Introduction	1
1.1	Computational musicology	4
1.2	Motivating examples	8
1.3	Aim and objectives	10
1.4	Summary	11
1.5	Outline	11
2	Polyphonic music	14
2.1	Music notation	15
2.2	Computer representations	20
2.3	Abstractions	22
2.3.1	Temporal relations	22
2.3.2	Musical relations	25
2.4	Motivating patterns	28
2.5	Requirements for a polyphonic pattern language	33
2.6	Summary	35
3	Related work	36
3.1	Expressive and precise approaches	36
3.1.1	Relational patterns	37
3.1.2	Humdrum	40
3.1.3	Structured Polyphonic Patterns	43

3.2	Restricted or approximate approaches	45
3.2.1	Vertical approaches	53
3.2.2	Graph approaches	54
3.2.3	Geometric approaches	55
3.2.4	Sequential approaches	56
3.3	Summary	57
4	Structured Polyphonic Patterns	58
4.1	Source	58
4.2	Components	61
4.3	The <i>SPP</i> language	62
4.4	Interpreting <i>SPP</i> patterns	67
4.5	Feature definition rules	71
4.6	Querying musical sources	72
4.7	Matching algorithm	73
4.8	Summary	74
5	Musicological queries in <i>SPP</i>	75
5.1	Overview	75
5.2	Sources	77
5.3	User-defined features	81
5.4	Queries	85
5.4.1	Compensated leap	88
5.4.2	Block chord	91
5.4.3	Parallel fifth	94
5.4.4	Suspension	97
5.4.5	Counterpoint	99
5.5	Summary	103

6	The expressiveness of polyphonic patterns	104
6.1	A hierarchy of polyphonic pattern languages	104
6.2	Relational patterns $\mathcal{R}$	108
6.3	From Humdrum to $\mathcal{H}$	109
6.3.1	Well-formedness	110
6.3.2	Interpretation	111
6.4	$\mathcal{SPP}$ and $\mathcal{SPP}_{\text{seq}}$	112
6.5	Comparing languages	113
6.5.1	$\mathcal{R}$ subsumes $\mathcal{H}$ and $\mathcal{SPP}$	113
6.5.2	$\mathcal{H}$ does not subsume $\mathcal{SPP}$	113
6.5.3	$\mathcal{SPP}$ does not subsume $\mathcal{H}$	114
6.5.4	$\mathcal{H}$ and $\mathcal{SPP}$ subsume $\mathcal{SPP}_{\text{seq}}$	115
6.6	Summary	119
7	Discussion	120
7.1	From requirements to $\mathcal{SPP}$	120
7.2	Beyond n-part polyphony	123
7.3	Motivating $\mathcal{SPP}$ semantics	124
7.4	Pattern matching efficiency	126
7.5	Mock $\mathcal{SPP}$ GUI	132
7.6	Suitability of $\mathcal{SPP}$ for data mining	133
7.7	Suitability of $\mathcal{SPP}$ for musicology	135
7.8	Summary	138
8	Conclusion	142
8.1	Summary and contributions	142
8.2	Anticipated developments	144
A	Musical concepts	145

CONTENTS

A.1 Overview	145
A.2 Glossary	149
B User-defined features	152
C Notation	158
D Matching algorithm	161
E Prolog implementation	166
F Additional tables and figures	170
F.1 Features	170
F.2 Queries	171
F.3 Musical excerpts for Chapter 5	173
F.4 Musical excerpts for Chapter 6	200
F.5 Musical excerpts for Chapter 7	206
G Accompanying CD	211

List of Figures

1.1	Written music	3
1.2	Bach chorale harmonization BWV 323. Two polyphonic patterns are highlighted: a parallel fifth in the eighth bar and a suspension in the final bar	9
2.1	First five bars of the tenor voice of BWV 323	15
2.2	First five bars of the tenor voice of BWV 323 in piano-roll notation .	16
2.3	First two bars of BWV 323	17
2.4	Illustration of the concept of sequence of layers	18
2.5	Bars three to five of BWV 323	19
2.6	The first two bars of BWV 323 as different mixtures of sequences and layers	21
2.7	Examples of monophonic transposition and transformation. a) and c): first five bars of the tenor voice of BWV 323; a) and d): transposed by a fifth down; b) and e): transformed while conserving pitch and duration contours	27
2.8	Examples of homophonic transposition and transformation. a) and c): first two bars of BWV 323; a) and d) transposed by a fifth down; b) and e): transformed so that chords are preserved and that the soprano voice forms the same harmonic intervals (albeit not necessarily with the same voice)	29
2.9	Eighth bar of BWV 323. The parallel fifth pattern is highlighted. Every dashed note added to the piano-roll would also form a parallel fifth	31

LIST OF FIGURES

2.10	Final bar of BWV 323. The suspension pattern is highlighted. Every dashed note added to the piano-roll would also form a suspension . . .	32
3.1	Prolog encoding of four notes forming a parallel fifth pattern	38
3.2	Prolog definition of the four temporal relations considered in this dissertation	38
3.3	The parallel fifth pattern in Prolog. Reproduced and adapted from Fitsioris and Conklin (2008)	39
3.4	Final bar of BWV 323 in Humdrum/kern encoding	40
3.5	Final bar of BWV 323 in Humdrum/kern encoding after the preprocessing required for the execution of a suspension query	41
3.6	Execution of a Humdrum suspension query	42
3.7	<i>SPP</i> encoding of four notes forming a parallel fifth pattern	43
3.8	Parallel fifth pattern in <i>SPP</i>	44
3.9	Suspension pattern in <i>SPP</i>	44
3.10	Example of an <i>SPP</i> feature definition rule	45
3.11	Overview of the information involved in the <i>SPP</i> suspension and parallel fifth patterns. Patterns are shown in CAPITALS. Features that are already present in the source are shown in bold . Features that are added to the source using <i>SPP</i> feature definition rules are shown in <i>italics</i>	46
4.1	<i>SPP</i> encoding of BWV 304, bar 21	60
5.1	Overview of the queries developed in this chapter. Queries are shown in CAPITALS. Features that are already present in the source are shown in bold . Features that are added to the source using <i>SPP</i> feature definition rules are shown in <i>italics</i>	76
5.2	<i>SPP</i> encoding of BWV 304, bar 21. The features of Table 5.3 are shown, except key : Dmaj , which should appear in every event but was omitted for readability. The basic features onset and offset shown in Figure 4.1 (page 60) are also omitted	80

LIST OF FIGURES

5.3	Frame-conditioned probabilities of metrical levels in Bach chorales, Symphony no.40, Chopin and folk songs	81
5.4	$\mathcal{SPP}$ encoding of BWV 304, bar 21. Only some of the user-defined features of Table 5.4 are shown. Features shown in figures 4.1 (page 60) and 5.2 are omitted	83
5.5	Frame-conditioned probabilities of melodic intervals in Bach chorales, Symphony no.40, Chopin and folk songs. Only intervals occurring over at least 1% of m temporal relations are considered	84
5.6	Examples of compensated leaps: a) BWV 273 bar 5 in the soprano voice; b) BWV 349 bar 13 in the alto voice; c) BWV 297 bar 3 in the tenor voice; and d) BWV 277 bar 1 in the bass voice. The excerpts are shown in piano-roll notation in figures a) F.3 on page 173; b) F.4 on page 174; c) F.5 on page 175; and d) F.6 on page 176	91
5.7	Examples of blocked I/i chords: a) BWV 292 bar 1; b) BWV 335 bar 4; c) BWV 415 bar 1; and d) BWV 438 bar 8. The excerpts are shown in piano-roll notation in figures a) F.7 on page 177; b) F.8 on page 178; c) F.9 on page 179; and d) F.10 on page 180	93
5.8	Examples of parallel fifths: a) BWV 263 bar 6; b) BWV 301 bar 3; c) BWV 323 bar 8; d) BWV 355 bar 15; and e) BWV 361 bar 12. The excerpts are shown in piano-roll notation in figures a) F.11 on page 181; b) F.12 on page 182; c) F.13 on page 183; d) F.14 on page 184; and e) F.15 on page 185	96
5.9	Examples of suspensions: a) BWV 259 bar 8; b) BWV 390 bar 12; c) BWV 393 bar 7; and d) BWV 285 bar 1. The excerpts are shown in piano-roll notation in figures a) F.16 on page 186; b) F.17 on page 187; c) F.18 on page 188; and d) F.19 on page 189	98
5.10	Examples of passing tones: a) BWV 253 bar 4; b) BWV 395 bar 9; c) BWV 262 bar 2; and d) BWV 436 bar 1. The excerpts are shown in piano-roll notation in figures a) F.20 on page 190; b) F.21 on page 191; c) F.22 on page 192; and d) F.23 on page 193	100

LIST OF FIGURES

5.11	Examples of two-voice counterpoint patterns: a) BWV 277 bar 3; b) BWV 285 bar 9; c) BWV 380 bar 3; and d) BWV 418 bar 7. The excerpts are shown in piano-roll notation in figures a) F.24 on page 194; b) F.25 on page 195; c) F.26 on page 196; and d) F.27 on page 197	102
5.12	Examples of three-voice counterpoint patterns: a) BWV 374 bar 7; and b) BWV 398 bar 11. The excerpts are shown in piano-roll notation in figures a) F.28 on page 198; and b) F.29 on page 199	102
6.1	A hierarchy of polyphonic pattern languages	105
6.2	Temporal networks for the following patterns: a) dislocated chord, b) parallel fifth, c) suspension, d) layered passing tones, and e) tritone resolution. The network d) can be expressed in $\mathcal{SPP}$ but not in $\mathcal{H}$. The network e) can be expressed in $\mathcal{H}$ but not in $\mathcal{SPP}$	106
6.3	Examples of dislocated $\mathbf{V}^7$ chords: a) BWV 285 bar 15; and b) BWV 318 bar 13. The excerpts are shown in piano-roll notation in figures a) F.30 on page 200; and b) F.31 on page 201	107
6.4	Examples of layered passing tones: a) BWV 255 bar 2; and b) BWV 320 bar 19. The excerpts are shown in piano-roll notation in figures a) F.32 on page 202; and b) F.33 on page 203	107
6.5	Examples of embellished tritone resolutions: a) BWV 257 bar 2; and b) BWV 315 bar 13. The excerpts are shown in piano-roll notation in figures a) F.34 on page 204; and b) F.35 on page 205	108
7.1	Suspension pattern in Chopin Waltz Op.64 Nr.2, bars 9 and 10	124
7.2	Comparing Humdrum and $\mathcal{SPP}$ against sources with an increasing number of events. Results are averaged over 1000 sources with 10 voices and an instance density of 0.1	128
7.3	Comparing Humdrum and $\mathcal{SPP}$ against sources with an increasing number of voices. Results are averaged over 500 sources with 10000 events and 1000 instances	128
7.4	Comparing Humdrum and $\mathcal{SPP}$ against sources with an increasing instance density. Results are averaged over 500 sources with 10 voices and 10000 events	129

LIST OF FIGURES

7.5	Comparing Humdrum and <i>SPP</i> preprocessing and matching times for a counterpoint query with varying number of features. Results are averaged over 10 sources with 4 voices and 10000 events	129
7.6	Comparing Humdrum and <i>SPP</i> preprocessing and matching times for a counterpoint query of varying sequence length. Results are averaged over 10 sources with 4 voices and 10000 events	130
7.7	Comparing Humdrum and <i>SPP</i> preprocessing and matching times for a counterpoint query of varying layer size. Results are averaged over 10 sources with 8 voices and 5000 events	131
7.8	The effect of ill-ordered literals in a Prolog query. Results are averaged over 1000 sources with 10 voices and an instance density of 0.1 . . .	131
7.9	The suspension pattern in a mock <i>SPP</i> GUI	132
7.10	Developing the suspension pattern in a mock <i>SPP</i> GUI	139
7.11	Transforming the suspension pattern into a block chord pattern in a mock <i>SPP</i> GUI	140
7.12	Examples of the rising fourths canon pattern: a) BWV 278 bar 8 between tenor and alto; b) BWV 328 bar 43 between bass and tenor; The excerpts are shown in piano-roll notation in figures a) F.36 on page 207; and b) F.37 on page 208	141
7.13	Examples of a two-voice module: a) BWV 328 bar 33 between bass and alto; b) BWV 382 bar 9, also between bass and alto; The excerpts are shown in piano-roll notation in figures a) F.38 on page 209; and b) F.39 on page 210	141
F.1	Frame-conditioned probabilities of scale degrees in Bach chorales, Symphony no.40, Chopin and folk songs	170
F.2	Frame-conditioned probabilities of harmonic intervals in Bach chorales, Mozart Symphony no.40 and Chopin piano pieces. Only intervals occurring over at least 1% of <i>st</i> temporal relations are considered . . .	171
F.3	BWV 273 (bar 5) with a highlighted compensated leap pattern	173
F.4	BWV 349 (bar 13) with a highlighted compensated leap pattern . . .	174
F.5	BWV 297 (bar 3) with a highlighted compensated leap pattern	175

LIST OF FIGURES

F.6	BWV 277 (bar 1) with a highlighted compensated leap pattern	176
F.7	BWV 292 (bar 1) with a highlighted I/i block chord pattern	177
F.8	BWV 335 (bar 4) with a highlighted I/i block chord pattern	178
F.9	BWV 415 (bar 1) with a highlighted I/i block chord pattern	179
F.10	BWV 438 (bar 8) with a highlighted I/i block chord pattern	180
F.11	BWV 263 (bar 6) with a highlighted parallel fifth pattern	181
F.12	BWV 301 (bar 3) with a highlighted parallel fifth pattern	182
F.13	BWV 323 (bar 8) with a highlighted parallel fifth pattern	183
F.14	BWV 355 (bar 15) with a highlighted parallel fifth pattern	184
F.15	BWV 361 (bar 12) with a highlighted parallel fifth pattern	185
F.16	BWV 259 (bar 8) with a highlighted suspension pattern	186
F.17	BWV 390 (bar 12) with a highlighted suspension pattern	187
F.18	BWV 393 (bar 7) with a highlighted suspension pattern	188
F.19	BWV 285 (bar 1) with a highlighted suspension pattern	189
F.20	BWV 253 (bar 4) with a highlighted passing tone pattern	190
F.21	BWV 395 (bar 9) with a highlighted passing tone pattern	191
F.22	BWV 262 (bar 2) with a highlighted passing tone pattern	192
F.23	BWV 436 (bar 1) with a highlighted passing tone pattern	193
F.24	BWV 277 (bar 3) with a highlighted two-voice counterpoint pattern .	194
F.25	BWV 285 (bar 9) with a highlighted two-voice counterpoint pattern .	195
F.26	BWV 380 (bar 3) with a highlighted two-voice counterpoint pattern .	196
F.27	BWV 418 (bar 7) with a highlighted two-voice counterpoint pattern .	197
F.28	BWV 374 (bar 7) with a highlighted two-voice counterpoint pattern .	198
F.29	BWV 398 (bar 11) with a highlighted two-voice counterpoint pattern	199
F.30	BWV 284 (bar 15) with a highlighted dislocated chord pattern	200
F.31	BWV 318 (bar 13) with a highlighted dislocated chord pattern	201
F.32	BWV 255 (bar 2) with a highlighted layered passing tones pattern . .	202
F.33	BWV 320 (bar 19) with a highlighted layered passing tones pattern .	203

LIST OF FIGURES

F.34 BWV 257 (bar 2) with a highlighted tritone resolution pattern	204
F.35 BWV 315 (bar 13) with a highlighted tritone resolution pattern . . .	205
F.36 BWV 278 (bar 8) with a highlighted rising fourths canon pattern . .	207
F.37 BWV 328 (bar 43) with a highlighted rising fourths canon pattern . .	208
F.38 BWV 328 (bar 33) with a highlighted two-voice module	209
F.39 BWV 382 (bar 9) with a highlighted two-voice module	210

List of Tables

3.1	Overview of existing polyphonic pattern approaches	47
4.1	Different types of <i>SPP</i> patterns and their basic interpretations . . .	64
5.1	Overview of the four corpora considered in this dissertation	78
5.2	Frame-conditioned probabilities of the st and sw temporal relations .	79
5.3	Basic features of source encodings	79
5.4	Examples of user-defined features	82
5.5	Frame-conditioned probabilities of the main polyphonic patterns analyzed in this chapter	85
5.6	The polyphonic patterns of this chapter expressed in <i>SPP</i>	87
5.7	The three most frequent melodic sequences in Bach chorales and their respective frame-conditioned probabilities. The frame-conditioned probability of the compensated leap pattern is also presented	90
5.8	The three most frequent four-part chords that use three scale degrees, as found in Bach chorales, and their frame-conditioned probabilities. Also, the two most frequent four-part chords that use four scale degrees and their frame-conditioned probabilities	92
5.9	Major chord with the root as the lowest note and its frame-conditioned probability with respect to all block chords. The following lines show the three most frequent permutations and their frame-conditioned probabilities with respect to other possible permutations	94
5.10	Most and least frequent parallel intervals and their frame-conditioned probabilities	97

LIST OF TABLES

5.11	Most frequent suspensions in Bach chorales and their frame-conditioned probabilities	98
A.1	Interpretation of absolute pitches in scientific notation in terms of frequency	146
A.2	Basic musical concepts	149
A.3	Examples of durations in common music notation	150
A.4	Musical concepts about texture	150
A.5	Musical concepts about pitch relations	151
A.6	Musical concepts about style	151
B.1	Features that encode properties of a single event	152
B.2	Features that encode melodic properties: relations of the current event with its direct predecessor	153
B.3	Definition rules for features that encode melodic properties (Table B.2)	153
B.4	Features that encode harmonic properties: relations of the current event with an overlapping event in some other voice $\mathbf{x}$	154
B.5	Definition rules for features that encode harmonic properties (Table B.4)	155
B.6	Features that encode complex properties: relations between more than two events	156
B.7	Definition rules for features that encode complex properties (Table B.6)	157
C.1	Syntax and meaning of $\mathcal{SPP}$ sources and components	158
C.2	Syntax and meaning of $\mathcal{SPP}$ patterns	159
C.3	Syntax and meaning of $\mathcal{SPP}$ feature definition rules	160
C.4	Syntax of $\mathcal{R}$ and $\mathcal{H}$	160
F.1	The three most frequent melodic sequences that begin with a step and their frame-conditioned probabilities	171

LIST OF TABLES

F.2	Minor chord with the root as the lowest note and its frame-conditioned probability with respect to all block chords. The following lines show the three most frequent permutations and their frame-conditioned probabilities with respect to other possible permutations	172
F.3	Polyphonic embellishments and their frame-conditioned probabilities .	172
G.1	Accompanying CD, tracks 1-12	211
G.2	Accompanying CD, tracks 13-25	212
G.3	Accompanying CD, tracks 26-37	213
G.4	Accompanying CD, tracks 38-50	214

Chapter 1

Introduction

This dissertation presents research concerned with the representation and retrieval of patterns in music. The notion of pattern (motif, regularity) is very common in science – including computer science – and various other human activities, including music. This chapter establishes the larger context of the current research, states its aim and objectives and offers an overview of the dissertation.

Patterns The world we live in is filled with regularities, whether manufactured by living agents – like us – or emerging from the interaction of inanimate matter. Describing those regularities requires a capacity for *abstraction*, a process by which the central characteristics of an object are highlighted and the details forgotten. Applying this process typically unifies different objects under the same description. This description, whether free-floating or explicitly written down in some notation, is what the current dissertation considers as a *pattern*. The inverse of abstraction consists of identifying objects that embody a particular pattern. We call this process *instantiation*; and objects identified as a result are said to be *instances* of the pattern.

One can easily recognize these processes of abstraction and instantiation in a variety of human affairs, perhaps most strikingly in science and the arts. As Alfred North Whitehead formulates it: “Art is the imposing of a pattern on experience, and our aesthetic enjoyment is recognition of the pattern” (Price, 1954). The importance of patterns is even more evident in science, with different goals and constraints. Scientific literature is filled with examples of simple, yet powerful descriptions of the world that are deemed important: for example, the double-helix pattern of DNA molecules, cold and warm fronts in meteorology, regularities of human languages

(e.g. consonants and vowels are thought to occur in all spoken languages), and cognitive biases such as the “primacy effect” (the tendency to recall more easily the initial elements of a sequence).

These patterns are often constituents of more complex laws or principles. In themselves, they do not explain a phenomenon, but can act as building blocks towards the concise formulation of an explanation (and possibly render the discovery of the explanation more likely). That is, the patterns concern the surface of things – what is directly observable – but constitute a first constructive step towards a deeper understanding.

Of course, it is a question of interpretation whether some fact truly lies on the surface or perhaps only appears as a consequence of some prior abstractions or indirect observations (vowels, for example, seem reasonably close to “direct experience”, where the double-helix structure of DNA relies on quite a few prior abstractions – about atoms and how they interact for example – and indirect observations involving sophisticated instruments). When thinking about patterns, it is important to be clear from the start about the “level of abstraction” at which we are working. The basic representation of the world one starts with before applying the process of abstraction, we call the *source*. Instances of a pattern are part of that source. The case of music is no exception and one has to carefully establish what one means by “the musical surface” (or musical source) in order to discuss the idea of musical pattern.

Musical structure There is a tradition in music theory to describe music in terms of “surface” and “background structures”. The surface consists of the notes actually played by a musician and heard by the audience. The background structures include a rich array of additional information that both musician and audience (more or less consciously) are constantly creating, updating and – to some extent – keeping in synchronization (sometimes by external means, e.g. through the gestures of a conductor). For example, these include tempo (the speed at which a piece of music is played), meter (the overall rhythmic feeling of a piece, e.g. a waltz versus a polka), orchestration (which instruments are used and with which “proportion”), melodic skeleton (which notes are considered essential to the melody and which are “facultative” ornamentations) and harmonic structure (roughly, how moments of tension and relaxation are distributed through the piece). While that information is not “in the music” per se, it is very often considered a legitimate starting point for the study of music.



Figure 1.1: Written music

Written music The research presented here, however, is concerned with the surface of music. We obtain a clear definition of what that surface is by building on the long tradition of written music: the transcription of a musical piece using a specialized graphical notation. Figure 1.1 shows a musical melody written as note symbols, with time flowing from left to right. The reader can listen to the melody by playing the first track of the accompanying CD. Written music depends on quite a few prior abstractions, namely those behind the idea of the musical note, which is based on a discretization of the elementary dimensions of the musical sound: pitch (how “high” or “low” the sound is), loudness, duration and timbre (the “texture” of the sound; what makes a trombone sound different than a guitar).

In addition, an estimation of the background structures is often notated in written music: for example, time signature, key signature, orchestration, and chords. Whether or not the estimation is accurate (i.e. whether or not it corresponds to what musicians and listeners “have in mind” while the music is played) is sometimes unclear. However, as the information is available in written music, it is considered in this dissertation as part of the musical source.

Patterns in music Contrary to other art forms (e.g. literature, painting), music usually does not refer to concepts that are meaningful on their own (see Nattiez, 1990, for a in-depth discussion). Perhaps as a consequence of this, a common theme in music theory is that a piece of music acquires meaning through the recurrence of similar passages. In the words of Meredith (2006), “a number of influential music analysts and music psychologists have stressed that discovering the important repetitions in a passage of music is an essential step towards achieving a rich understanding of it”.

Before attempting to analyze the deeper structures of a piece of music, a scholar typically gains familiarity with the piece by making observations about the musical surface. Cook (1987) suggests various ways to approach this familiarization process, of which “very often the analysis proper will emerge”. One such ways is to look for patterns, sequences or groups of notes that appear frequently in the piece.

In this viewpoint, the notion of pattern is based on musical expertise. The occurrences of a pattern can be presented in written music, and the relation between the occurrences discussed, but an exact definition of the pattern is not required. It is assumed to exist in the mind of a competent reader or listener.

For example, Schubert (2007) isolates various patterns in the written transcription of pieces by the sixteenth century Italian composer Palestrina. The patterns are classified according to how they are formed and Schubert discusses how they influence the larger structure of the pieces. Intuitively, the instances of the various patterns are clearly related, and it is hence not necessary to formally account for the variations exhibited from one instance to the other (e.g. a different melody, or some added notes).

Another example of this viewpoint on musical patterns is the work of Gjerdingen (2007) on the pedagogical tradition of Partimento. Gjerdingen discusses patterns that were used as templates. The student would learn how to create elaborations of the basic patterns, and how to combine them in interesting ways. Gjerdingen retraces examples of such patterns, and how they were used in real musical pieces. Again, the instances of the patterns are clearly related, even if they exhibit significant variations and elaborations.

Both Schubert (2007) and Gjerdingen (2007) discuss patterns that they have patiently isolated by hand. In the future, one can imagine that computer tools will help to make this task less cumbersome. These tools would however need to approximate the musical expertise that musicologists exhibit and this is a goal that computer music research has yet to reach (although some progress has been made towards it).

In the current dissertation, we focus instead on musical patterns that are easily amenable to a formal, computational representations. Such patterns could still be useful for traditional musicology as discussed and exemplified in Section 7.7. However, their main use lies in computational musicology, where the analysis of large amount of data is crucial, and automated techniques necessary.

1.1 Computational musicology

One could define computational musicology as the study of the musical phenomenon with the use of computers: for data representation and for various form of specialized

processing. With respect to musical patterns, two applications are predominant: pattern matching and pattern discovery (data mining).

Computing and music Because of the versatility of computers, computational techniques are applied to problems in many different fields. In return, computing has grown wider and wider, developing many new techniques. Music is no exception to that trend and provides many challenges for computer scientists. Applications of computers to music abound: for example, sound synthesis, computational composition and analysis, visualization, and music pedagogy. The idea of musical patterns is often at the heart of such music processing tools.

A large portion of computational approaches to musical patterns focus on the retrieval of musical sources (Downie, 2003). In this view, the representation of the source and its regularities does not have to be intelligible. There does not need to be a clear representation of the pattern since the product is a search engine meant to interact with the user at a very high level. This point of view is discussed and reviewed further in Chapter 3 when looking at existing computational approaches to patterns in music.

Representation The current research is mainly concerned with the representation of the temporal aspect of music and we draw inspiration and insight from Alan Marsden’s seminal book on this topic (Marsden, 2000). Focusing on the temporal aspect has one main advantage: our work is not necessarily limited to music. Although we do not pursue this path here, we could in principle use our approach for any problem that has a similar temporal structure to polyphonic music: concurrent streams of events where events have some duration and where events that overlap in time are particularly important. For example, the schedule of a projet, the execution trace of parallel programs, or the arrival of multiple requests to a server. The current dissertation focuses on music, however, as this is the application for which the approach is tailored. For the representation of the other aspects of music, we use basic notions of music theory. These mainly revolve around the representation of pitch. Although this is also a very active area of research – in particular the notions of pitch space and pitch distance (see e.g. Tymoczko, 2009, for a recent critical review) – we consider it to be outside the scope of the current dissertation.

Processing Significant attention has been invested into the reconstruction of hidden structures from the surface of music (see e.g. Temperley, 2001). These are

generally tailored for very specific tasks: for example, extracting the melodies (or voices) when they are not clear from notation (Cambouropoulos, 2006), segmenting melodies into phrases (M. T. Pearce and Wiggins, 2008), exploring the possible hierarchical organization of a given melodic phrase (Marsden, 2005), extracting the underlying chord sequence of a piece (Khadkevich and Omologo, 2009), performing some analysis on that chord sequence (Pachet, 2000), and finding the key of the piece (Hu and Saul, 2009).

Humdrum There are also some general “fact gathering” approaches. This corresponds to the idea of pattern matching: the expert user writes down what he or she considers as a key pattern and the computer searches for that exact pattern in a musical source, returning every instance. Typically, the user then analyzes the results in order to gain some deeper understanding of the musical source under examination. This is the point of view taken in the current dissertation. It was pioneered by musicologist David Huron, who developed Humdrum (Huron, 1999), the only general approach to polyphonic patterns to enjoy some lasting success.

Humdrum was developed in the late 1980s and has been used in many computational musicology projects. For example, pattern matching in Humdrum can be used to test hypotheses about music or musical styles. Jan (2004) establishes a link between the styles of Mozart and Hayden by showing that particular patterns can be found in selected works of both composers. Huron (2001a) analyzes the data gathered by matching several manually crafted patterns to support an argument about the perceptual independence of concurrent melodies. Huron (2001b) uses a similar empirical point of view to refine and criticize Alan Forte’s analysis of Brahms’s opus 51, No.1. Huron formally encodes patterns discussed by Forte and then proceeds to search for these patterns in other compositions by Brahms. The result is a measure of the *salience* of the patterns that Alan Forte identified, i.e. how representative they are of Brahms’ opus 51, No.1.

Modern approaches Humdrum is a typical application of its time: it uses a text-based file format with a somewhat loose syntax; it is based on Unix-like command-line tools for processing; and patterns are defined using regular expressions that are matched directly on the concrete syntax. As such, it lacks the efficiency and ease-of-use of modern computer tools. Approaches to musical patterns designed after Humdrum often focus on some “modernization” effort: give a clearer semantics to the idea of patterns, more efficient pattern matching, or automated pattern

manipulation. The latter is related to the idea of music data mining (see Rolland and Ganascia, 2002, for an overview). Rather than specifying a few patterns by hand, and gathering facts from the inspection of the results, the idea is to explore and compare large quantities of patterns and gather facts from the results of these comparisons.

Most of the modern approaches to musical patterns focus on the simpler, restricted case of monophonic music. Figure 1.1 is typical of monophonic music: it contains a single melody, easily represented as a sequence of notes. This simplifies temporal relations and, in general, the definition of patterns.

Some approaches consider the full polyphonic case, but without the generality of Humdrum (these are reviewed in Chapter 3). In the polyphonic case – reasonably the general case for western music – many melodies unfold at the same time. There cannot be a single structure – such as a sequence – to capture every piece of music. In addition, the temporal relations are more complex. Section 1.2 introduces some challenges that polyphonic music and polyphonic patterns entail.

Of all the recent work on musical patterns, we are particularly inspired by the work of Darrell Conklin and his viewpoint method. A viewpoint is a user-defined function that abstracts from the musical surface by recording only some of the available information, in some musically relevant way. In addition, viewpoints can be combined in an elegant modular way to capture progressively more complex musical notions. Over the years, the method has enabled the formal expression of many musical concepts and has been applied to many computational musicology projects (Conklin and Witten, 1995; Conklin and Anagnostopoulou, 2001; Conklin, 2002, 2010, 2008).

For example, Conklin (2008) revisits the analysis of Huron (2001b). Rather than specifying the patterns by hand, Conklin develops a method where patterns are discovered automatically and compared for salience. The result is twofold: i) the salience of Forte’s patterns is empirically validated, but more importantly ii) the work shows that Forte’s patterns can be discovered without prior knowledge and with only a few reasonable assumptions about what constitutes a musical pattern.

Conklin (2008) combines the versatility of viewpoints with the notion of *subsumption*: that a pattern can be described as more general or more specific than another pattern. This concept has been explored extensively in general data mining approaches such as Inductive Logic Programming (Nienhuys-Cheng and de Wolf, 1997), but very little in the case of music data mining. One recent application is the

discovery of patterns that are distinctive (Conklin, 2010): very general, yet over-represented in one corpus with respect to some comparison corpora. This is done through the notion of feature set (Conklin and Bergeron, 2008): a representation of music where a musical event is described by a set of viewpoint/value pairs – called features by Conklin and Bergeron (2008). The representation supports subsumption in a straightforward way: a pattern component is more general if it is subset of more specific pattern components.

Although the current dissertation is focused on pattern matching, our approach is designed to eventually support subsumption and pattern manipulation (this is discussed in Chapter 7). Our motivation is to build on the work of Conklin, in particular by extending it to the general case of polyphonic music, as it has been mainly applied to the restricted case of monophonic music. We achieve this by extending to polyphony the idea of viewpoint – called user-defined features here – and the notion of feature set. We focus on giving this extension a clear syntax and semantics. The result is something that was yet to be achieved: a method that is as general as Humdrum for polyphonic pattern matching but also conducive to music data mining. Section 1.3 details the objectives that we set out to achieve in this modernization effort of Humdrum and extension of Conklin’s work to polyphony.

1.2 Motivating examples

We examine two examples in order to better explain the challenges that polyphonic patterns can entail. Figure 1.2 shows a chorale melody harmonized by J.S. Bach, in common music notation. Two well-known polyphonic patterns are highlighted: a parallel fifth and a suspension. The reader can listen to this musical piece by playing the second track of the accompanying CD. The detailed explanation of common music notation is deferred until Chapter 2, which contains an overview of what a musical source contains and of the common abstractions over such a source. It suffices to say here that, as in the case of monophony, the spatial orientation of note symbols (e.g. ♯) indicates timing. In addition to notes that follow one another spatially and temporally, notes that are aligned vertically occur at the same time, concurrently (with the exception that every note in the top portion of the figure occurs before the notes of the bottom portion).

The first pattern is a parallel fifth. Using Prolog, Fitsioris and Conklin (2008) explored the occurrences of parallel fifths in Bach chorales. This is relevant as this



Figure 1.2: Bach chorale harmonization BWV 323. Two polyphonic patterns are highlighted: a parallel fifth in the eighth bar and a suspension in the final bar

pattern is often considered as prohibited. By using a pattern matching approach, Fitsioris and Conklin (2008) showed that the parallel fifth occurs more than one would expect. This fact – and other similar facts that can be gathered using polyphonic patterns – can help validate or invalidate (or at least mitigate) particular statements about musical style. Though understood clearly by every first year music student, the parallel fifth is remarkably difficult to express formally. One must express, for example, the fact that the first two notes of the pattern have to overlap in time (but not necessarily start at the same time as is the case in Figure 1.2). The next two notes must also overlap (and this time necessarily start at the same time). The pitch distance between the notes must correspond to the interval of a perfect fifth, or alternatively the interval of an octave plus a perfect fifth. There must be some melodic motion between two successive notes and, finally, both pairs of notes must involve the same voices (all of these concepts are explained in Chapter 2). The pattern can be captured using Humdrum, but it is difficult to do so with precision and confidence in the pattern matching results (see Bergeron and Conklin, 2008, or Chapter 3). Precision is important in a computational musicology setting. One needs to be certain of the pattern matching results in order to offer a credible commentary of statements of music theory. The approach of Fitsioris and Conklin (2008) achieves this goal. It is an ad hoc approach designed specifically to return

every instance of the parallel fifth pattern with a high degree of confidence in the results. In addition, the approach is easy to generalize and – for lack of existing competitors to Humdrum – we consider it to be a complete pattern matching approach in this dissertation.

The second pattern is a suspension. Contrary to the parallel fifth pattern, this pattern is common in western tonal music. It is part of a large collection of prescribed patterns called *polyphonic embellishments*. Using the Humdrum toolkit, Huron (2007) analyzed their occurrences in Bach chorales. Embellishments were marked up manually, allowing for maximum flexibility of interpretation. It is also possible to develop a polyphonic pattern that captures suspensions, as shown by Bergeron and Conklin (2008). This is a rigid definition, but it has the advantage of precision: results are definite facts about the source being queried.

As pointed out by Bergeron and Conklin (2008), the suspension pattern is interesting to consider as it further illustrates the level of sophistication required to capture polyphonic patterns, especially regarding temporal relations. The second note of a suspension must occur as the first note already sounds (in Figure 1.2, that first note is displayed with two note symbols linked by a *tie*; indicating that the note sounds continuously over the transition from one symbol to the other). Similarly, the third note of the pattern must occur while the second note is sounding. This temporal relation of “starting while” is generally not supported by existing approaches to polyphonic patterns. In addition, the suspension pattern requires us to measure pitch distance in an abstract way, by classifying distances in two categories: consonances and dissonances (again, these notions are explained in Chapter 2). Although it is quite common in music theory to use the dissonance and consonance concepts rather than “concrete” intervals, most existing approaches to polyphonic patterns do not support this level of abstraction.

1.3 Aim and objectives

The general aim of the current research is to create a modern pattern matching method for polyphonic music, in particular one that satisfies the following objectives:

1. Full support for written polyphonic music
2. Expressive and precise
3. Clear syntax and semantics

- 4. Flexible
- 5. Easy to use and efficient
- 6. Conducive to music data mining

1.4 Summary

To summarize, the current dissertation circumscribes the idea of pattern by defining the following four concepts:

- 1. Source: the data in which patterns are to be identified. This dissertation considers written polyphonic music
- 2. Abstractions: the kinds of simplifications that enable the identification of regularities in the source. This dissertation focuses on temporal relations, while supporting important musical relations
- 3. Pattern language: a way to write down those abstractions, for example as a set of constraints. This dissertation develops such a language and gives it a clear syntax and semantics.
- 4. Pattern matching: a mechanism to quickly and efficiently identify the instances of a particular pattern in a given source. This dissertation provides an efficient matching algorithm for the pattern language it develops.

We consider polyphonic patterns that have the level of sophistication of the parallel fifth and suspension. We consider and compare three main approaches to polyphonic patterns: i) a relational pattern language inspired by Fitsioris and Conklin (2008), ii) Humdrum, and iii) the current approach, Structured Polyphonic Patterns (Bergeron and Conklin, 2008).

1.5 Outline

This chapter introduced the idea of pattern as revolving around the concepts of source, abstractions, language, and matching. The next chapter, Chapter 2, introduces the particularities of the source examined in this research: polyphonic music.

It also introduces the common abstractions found in music theory and sketches a set of requirements for a polyphonic pattern language. Those requirements can be seen as a refinement of the objectives stated in Section 1.3. We see that polyphony is a difficult case in music and a challenge for computational musicology.

In Chapter 3, we present a review of existing computational approaches to patterns in polyphonic music. We see that only two approaches are comparable to the current research in terms of the requirements stated in Chapter 2: Humdrum and relational patterns. These are compared to the current approach in terms of expressiveness in Chapter 6 and in term of efficiency in Chapter 7. Other approaches generally lack expressiveness and cannot represent important musical abstractions with precision. In particular, they could not return every instance of the parallel fifth and suspension patterns discussed here and throughout the dissertation.

Chapter 4 gives the formal presentation of the approach we develop: Structured Polyphonic Patterns (*SPP*). It also defines a pattern matching algorithm. Chapter 5 applies the approach to a corpus of chorale harmonizations by J.S. Bach. The chapter can be read as a tutorial to *SPP*. We show how to use the method for musicological queries ranging from very simple to complex encodings of music-theoretical notions of style. To add some perspective to the results, we compare our main corpus with three corpora: a Mozart symphony, a collection of piano pieces by Chopin, and a collection of folk songs containing more than 8000 pieces – in addition to its size, demonstrating scalability of the method, that latter corpus is interesting as it shows that *SPP* is also usable for monophony. Examining several corpora allows us to establish that some polyphonic patterns constitute salient properties of a corpus: they are over-represented in one corpus by comparison to the others.

In Chapter 7, we motivate the semantics of *SPP* by discussing alternative semantics. We discuss the efficiency of *SPP* matching and compare it to Humdrum thoroughly. We see that *SPP* is a constructive step towards an approach to music data mining for the polyphonic case. The conclusion of this dissertation summarizes the contributions of the current research. In particular, Chapter 8 argues that the *SPP* matching method satisfies the high-level objectives stated in this introduction and the requirements elaborated in Chapter 2.

Finally, the current dissertation contains some Appendices that can prove helpful as reading companions. Appendix A presents a glossary of important musical terms and concepts. Appendix B offers a summary of how these musical concepts are formalized in order to appear in *SPP* patterns. Appendix C enumerates and explains

the notation we use in the following. Appendix D details the pattern matching algorithm that this dissertation develops. Appendix F presents additional results and figures. Finally, Appendix G enumerates the content of the accompanying CD: the reader can listen to every musical excerpt presented in the dissertation. Note also the presence of an index starting on page 222.

Chapter 2

Polyphonic music

This chapter motivates and describes the notions of musical source and musical patterns that are central to this dissertation. As explained in Chapter 1, we adopt a notion of musical source that directly follows western musical notation. We explain how the notation supports multiple textures of growing complexity: monophony, homophony, and polyphony. We review how this is reflected in some of the existing computational representations of music. The notion of pattern we adopt is constructed by surveying common abstractions over that musical source: from textbooks of music theory, from studies of music perception, and from existing literature in computational musicology. In addition, these abstractions are validated by revisiting the motivating examples of Chapter 1: the parallel fifth pattern and the suspension pattern. Finally, with the notions of source and pattern precisely defined, we can expand and refine the objectives stated in Chapter 1. This yields a set of requirements for a polyphonic pattern language against which existing computational approaches are evaluated in Chapter 3.



Figure 2.1: First five bars of the tenor voice of BWV 323

2.1 Music notation

As stated in Chapter 1, the current dissertation considers written music as the source from which musical patterns are extracted. In particular, we consider the common music notation developed in the western musical tradition. For western music, this constitutes a sound abstraction of what the musical phenomenon is. As Marsden puts it: “common music notation has evolved over centuries of use by performers and composers, and so can be regarded as having proved its fitness in use” (Marsden, 2000, p.119).

In the common music notation, a piece of music is written down by drawing *note* symbols (e.g. ♩) over a *staff* (five horizontal lines). Figure 2.1 illustrates this process along with other basic features of music notation: the *clef* (at the far left of the staff), the *key signature* (the three *sharp* symbols ♯ next to the clef), the *time signature* (the symbol *C* next to the key signature), and *bar lines* (vertical lines subdividing the staff into *bars*).

The placement of note symbols in Figure 2.1 reflects two important dimensions of music: the pitch of the notes and their unfolding in time. The vertical placement of a note indicates its pitch, with only a discrete number of pitches expressed: notes have to be drawn either on a staff line or between staff lines (with additional staff lines being added when necessary; for example to “support” the third note of Figure 2.1). The notated *pitch* of a note prescribes whether it shall sound high (like a female voice) or low (like a male voice). The horizontal placement of a note indicates when it shall be played, with time flowing from left to right.

However, the placement of notes in the common notation only imperfectly reflects those key dimensions. For a precise depiction, it can be useful to use an alternate notation called the piano-roll notation, which is more akin to a precise plot of the musical content. Figure 2.2 presents the material of Figure 2.1 in piano-roll notation. Notice how, for example, the pitch distance between adjacent notes varies in Figure 2.2, while it seems to be always the same in Figure 2.1. The information is the same, but visualized differently. The clef and key signature are also related to

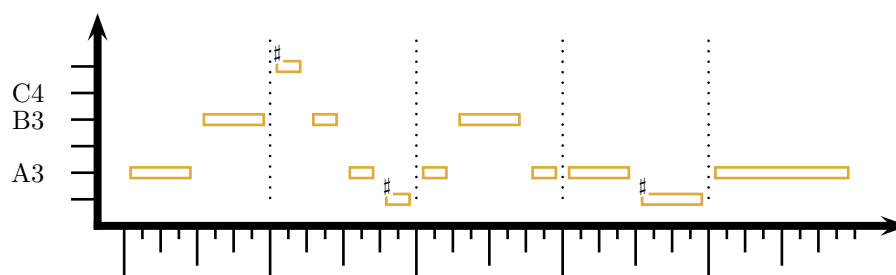


Figure 2.2: First five bars of the tenor voice of BWV 323 in piano-roll notation

the pitch dimension. These are discussed in Appendix A.

Similarly, the duration of notes in common music notation is slightly indirect, as it is represented by the shape of the note symbols: for example, ♩ for two beats, and ♪ for one beat. A *beat* corresponds to the time elapsed between two taps of the foot when following the tempo of a piece of music. Table A.3 in Appendix A presents some of the most common note shapes and their durations.

In the musical example of figures 2.1 and 2.2, the notes appear one after the other in sequence. This is the typical alignment for a musical *part* or *voice*: the notes that are played by the same instrument. When only one such part appears, the music is said to have a *monophonic* texture (literally, a one-voice texture).

The common music notation also accommodates pieces of music having multiple parts. Consider, for example, the musical excerpts of Figure 2.3. Two staves are used to help differentiate the parts. In addition, each staff holds respectively two parts, one with notes pointing upwards and the other with notes pointing downwards. In the piano-roll notation, we use different colors to distinguish the parts.

Figure 2.3 is an example of another well-known musical texture called *homophony*. It is characterized by the presence of multiple parts that all unfold in lockstep. As is clear from the piano-roll notation, there are, at any given time, four notes that start at the same time and have the same duration.

Homophonic music evokes a fundamental duality in how music for multiple parts is perceived and represented. Looking again at Figure 2.3, one can clearly imagine two ways to structure the music: either as four sequences of notes that unfold in lockstep or as one sequence in which every element contains four notes that sound at the same time. In the latter view, we say the four notes form a *layer* of notes. The whole excerpt is then imagined as a *sequence of layers*. This is the view one would take, for example, when thinking of block chords played on the piano or strummed

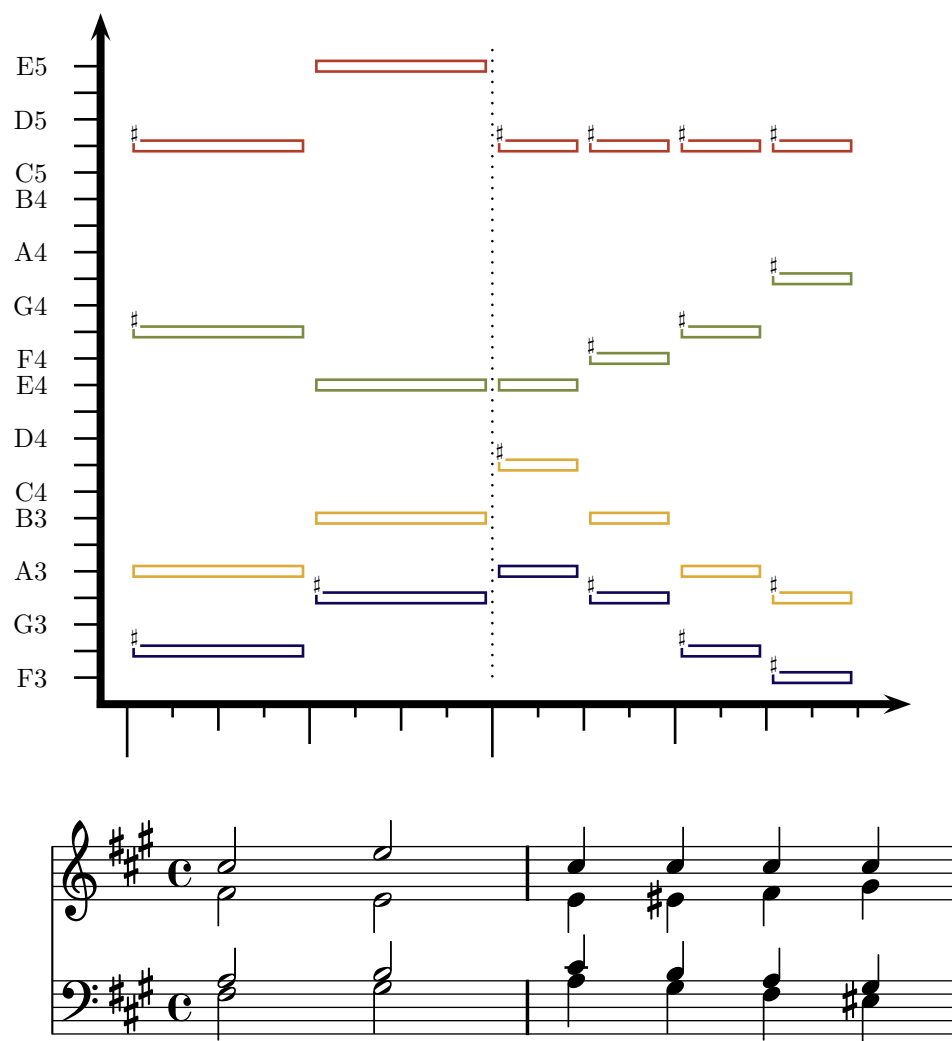


Figure 2.3: First two bars of BWV 323

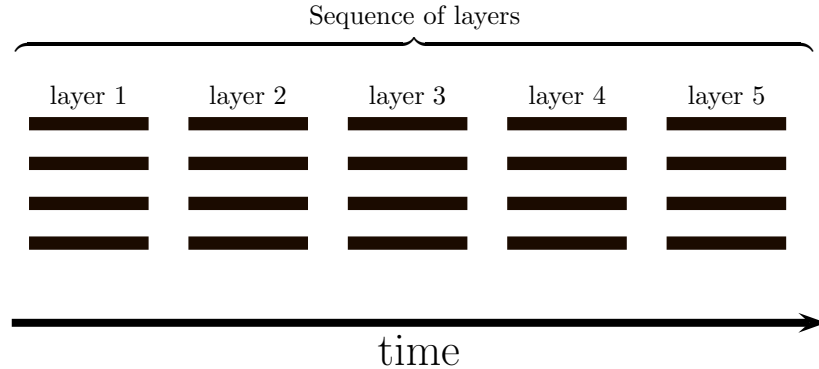


Figure 2.4: Illustration of the concept of sequence of layers

on a guitar. Every block chord would be considered as a layer where many musical events unfold at the same time. In the remainder of this dissertation, we use the phrase “sequence of layers” to refer to such cases where many layers appear one after the other sequentially in time. Figure 2.4 illustrates the meaning of the phrase.

Most music for multiple parts is not purely of homophonic texture (but some fragments or inner parts frequently are). When the parts are not unfolding in lockstep, the resulting texture is called *polyphony*. Consider for example Figure 2.3. Again, the common musical notation exhibits two staves, each respectively two parts. The parts sometimes unfold in lockstep (as in the last bar), but mostly do not.

In this dissertation, we only consider the notion of part and not voices. We sometimes refer to our point of view as *n*-part polyphony. This point of view has drawbacks; in particular it might not capture the most general case of western written music, where parts sometimes split up in many voices, or many melodies (think for example of the piano part). However, while parts are almost always clear from the notation, voices are not necessarily clearly identified or apparent in the score. Working with voices would require quite a bit of effort to automatically identify the voices, which is out of the scope of the current approach. Nonetheless, we briefly discuss how to extend the approach to accommodate voices in Section 7.2. In the remainder of this thesis, we use the words *part* and *voice* indiscriminately to refer to the concept of part. In the “worst” case, if parts and voices are not known, a piece of music can still be represented using a single part that contains all the notes of the piece. For example, this is how piano music is represented in Chapter 5.

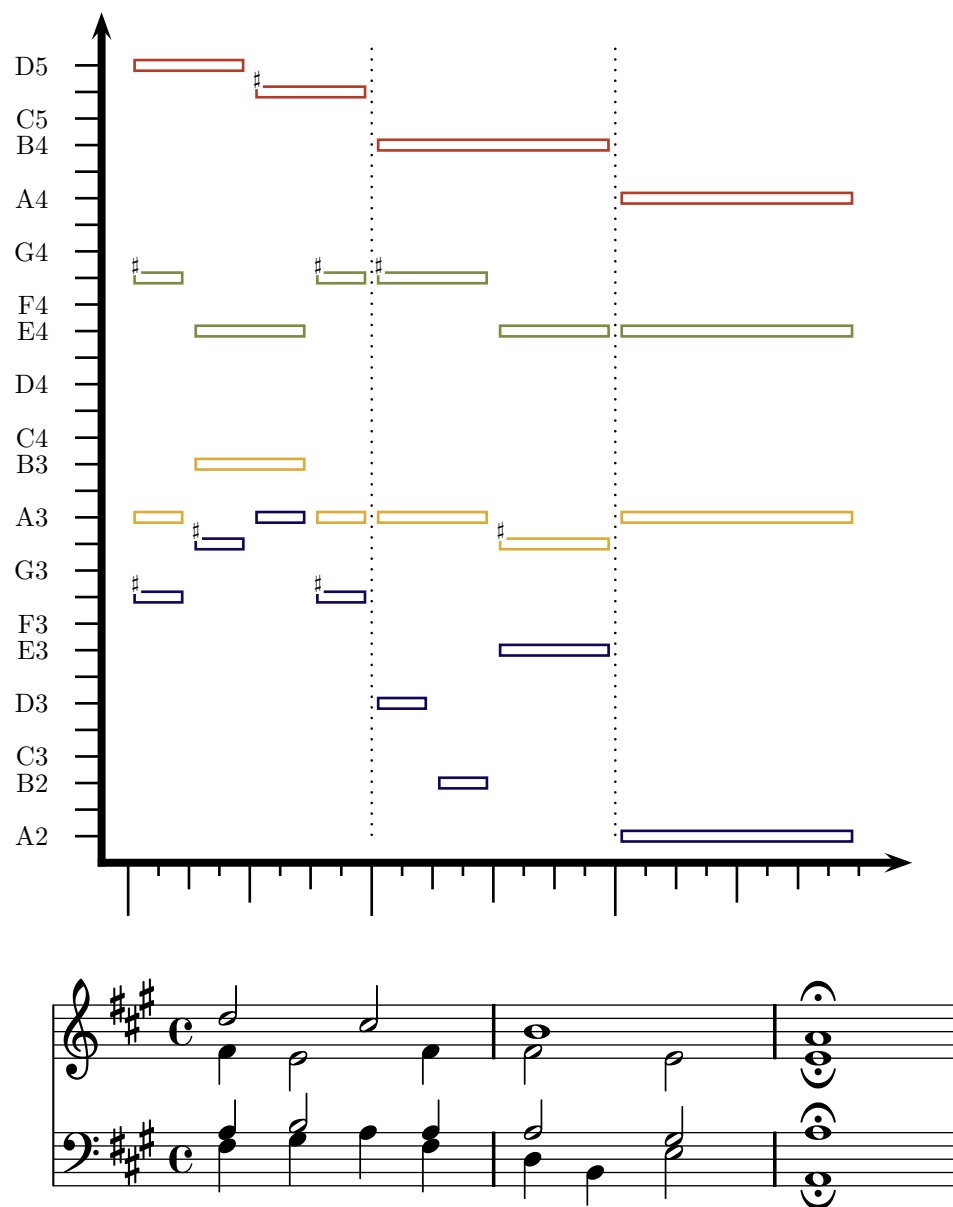


Figure 2.5: Bars three to five of BWV 323

2.2 Computer representations

The representation of written music has attracted significant interest over the years. Many file formats exist to store musical data on a computer: e.g. ESAC, DARMS, and MIDI (Selfridge-Field, 1997). As these have different motivations, they can often differ widely. The well-known MIDI format, for example, is a binary file format designed to transfer information from controllers (such as keyboards and touchpads) to synthesizers. It is not designed for manual edition or to transcribe a piece in common notation (although it is often used for this purpose). Some formats are designed to imitate notation, such as MusicXML (Good, 2001) and Humdrum (Huron, 1999) and aim at supporting as many features of common music notation as possible. Other formats are designed to quickly transcribe melodies and are restricted to monophonic music, e.g. ESAC.

The representations also differ significantly in how they represent pitch and duration or, in general, how they accommodate the concepts of music theory discussed in Section A.1. Most representations, however, have some notion of structure. In MIDI, MusicXML and Humdrum, parts can be specified. In Humdrum, one can even specify how parts appear and disappear as the piece unfolds.

Representations developed as part of research projects very often put the emphasis on structure. For example, Bel (1990) focuses on the concept of sequence and proposes a representation that supports the realization of a sequence in many different ways (by mapping its components to different onsets). In such a scheme, layers are formed in two ways: i) when sequences are synchronized, and ii) when events in the same sequence overlap as they are being mapped to neighboring onsets. One could say that layers are implicit for Bel (1990). Other representation methods typically use explicit layers and revolve around the duality between sequences and layers discussed in the previous section. For Balaban (1996), explicit music structures are formed in two basic ways: by concatenating sub-structures (forming sequences) and by specifying that sub-structures are simultaneous (forming layers). An important feature of the latter scheme is that musical structures are hierarchical. In this kind of representation, a sequence or a layer is itself a musical object that can be inserted wherever a note can occur. In the visual composition language Open Music (Agon et al., 1999), for example, the chord object (essentially a layer) can be reused anywhere a note is expected. In general, these can be embedded at will, as is clear from the algebraic data structures developed by Hudak et al. (1996) or Conklin (2002). For example, expressing sequences with square brackets `[/]` and layering with hor-

$$\begin{array}{lcl}
 \text{Layered sequences} & \frac{\frac{[C\sharp, E, C\sharp, C\sharp, C\sharp, C\sharp]}{[F\sharp, E, E, E\sharp, F\sharp, G\sharp]}}{\frac{[A, B, C\sharp, B, A, G\sharp]}{[F\sharp, G\sharp, A, G\sharp, F\sharp, E\sharp]}} & \\
 \text{Sequence of layers} & \left[\frac{\frac{C\sharp}{F\sharp}}{\frac{A}{F\sharp}}, \frac{\frac{E}{E}}{\frac{B}{G\sharp}}, \frac{\frac{C\sharp}{E}}{\frac{C\sharp}{A}}, \frac{\frac{C\sharp}{E\sharp}}{\frac{B}{G\sharp}}, \frac{\frac{C\sharp}{F\sharp}}{\frac{A}{F\sharp}}, \frac{\frac{C\sharp}{G\sharp}}{\frac{G\sharp}{E\sharp}} \right] & \\
 \text{Structures} & \left[\frac{\frac{C\sharp}{F\sharp}}{\frac{A}{F\sharp}}, \frac{\frac{E}{E}}{\frac{B}{G\sharp}}, \frac{\frac{[C\sharp, C\sharp, C\sharp, C\sharp]}{[E, E\sharp, F\sharp, G\sharp]}}{\frac{[C\sharp, B, A, G\sharp]}{[A, G\sharp, F\sharp, E\sharp]}} \right] &
 \end{array}$$

Figure 2.6: The first two bars of BWV 323 as different mixtures of sequences and layers

horizontal lines, the homophonic excerpt discussed in the previous section could be expressed in many different ways. Figure 2.6 presents three such ways: i) as layered sequences, where the individual parts are encoded as sequences and the four sequences are put in a layer; ii) as a sequence of layers, where the chords that the parts form by unfolding in lockstep are encoded as layers and put in a sequence and iii) as a mixture of both approaches, where the first bar is encoded as layered sequences and the second bar as a sequence of layers. The latter scheme shows how layers and sequences can be embedded freely. When such a free mixture of sequences and layers is used, we simply refer to this organization as *structured*.

The basic interpretation of such structures is in accordance with the homophonic texture : sequences hold notes or objects that are contiguous, one directly following the other; layers hold objects that start together and unfold at the same time. Some scheme offer more control over the interpretation. For example, the musical structures of Balaban (1996) can also contain markers to override the default onset of a structure. In addition, high-level operators are available to specify the repetition or stretching of a structure. The structural representation developed by Marsden (2000) provides elaborate control over how sequences and layers are formed. In this representation, musical objects are positioned against a metrical grid, their duration measured in beats. Every object has an upbeat portion (unfolding before the first beat of the object) and an after-beat portion (unfolding after the last beat of the object). The representation provides many ways to join objects in sequences: *regular*

(by allowing the overlap between the after-beat of the first object and the upbeat of the second, if any); *no overlap* (by adding space between the objects to avoid overlap, if any); *elide* (by clipping the after-beat and upbeats to avoid overlap); and *overlap* (forcing the overlap of after-beat and upbeat). Similarly, many possibilities are provided for the joining of two objects as a layer: *regular* (where the first beat of the first object is simply aligned with the first beat of the second object, without altering durations); and multiple ways to *stretch* the second object so that both objects have the same durations, e.g. by holding the last note, by padding with rests, by stretching the durations, by cycling or clipping the musical content.

In this dissertation, we focus on a simple interpretation of structures. In particular only four temporal relations are considered. The idea of using structures to express temporal relations has some history (see Marsden, 2000, chap.3, p.85-87). This is the central idea behind the *SPP* method developed in this dissertation. It allows us to bring the well-known sequence/layer paradigm into the realm of musical patterns, by allowing layers and sequences to encode abstractions over the musical source. Note that Marsden (2000, chap.4) also discusses how structures can represent durations, an idea that is not pursued here.

2.3 Abstractions

The notion of polyphonic pattern developed in this dissertation is based on abstractions over the musical source. We consider two types of abstractions: temporal relations and musical relations. The temporal relations are inspired by the musical textures and existing music representations. Musical relations stem from music theory and are closely related to notions of music notation such as the representation of pitch.

2.3.1 Temporal relations

There are many ways to define temporal relations in music. These typically depend on the ontology of musical time one chooses. Marsden (2000, chap.2) cites several properties of time to consider when choosing an ontology: its “shape” (whether linear, branching or circular), its “extent” (whether finite, unbounded or infinite), its “texture” (whether discrete, dense or continuous), and finally the individuals it is composed of: *points*, *periods* or *events*. Like most computer representations of

music, this dissertation adopts an ontology where time is linear, unbounded, and dense.

Linear time Linear time implies that there is a single past and single future. In contrast, branching time allows more than one past or more than one future (or both). Circular time removes the notion of past and future as every time point keeps occurring again and again. The notion has been applied in music to describe rhythm (Toussaint, 2010) and chord sequences (Pachet, 2000).

Unbounded time Unbounded time implies that there is no definite end of time. This does not mean that time extends infinitely, but simply that one cannot know in advance the extent of time. This is necessary to represent music as the length of musical pieces varies and cannot be bounded by some constant. In contrast, finite time possesses a clear beginning and end. It can be sufficient, for example, when representing rhythmic patterns of a known length.

Dense time In a dense time texture, any amount of time can be subdivided into a smaller amount. By contrast, a discrete time texture supports subdivisions only to a point, i.e. there is some smallest amount of time which cannot be subdivided. This does not mean that a dense texture can accommodate arbitrary subdivisions, however. A texture that supports arbitrary subdivisions is described as continuous. The difference is reminiscent of rational numbers and real numbers: rational numbers allow for infinite subdivision of the number 1, for example, but some subdivisions are not representable as ratios, e.g. the golden section.

Individuals Marsden (2000, p.33) cites three kinds of individuals that can constitute time: points, periods and events. Points are instantaneous moments in time, while periods have some extent: they last some amount of time. When using events, one switches the point of view somewhat and describes only the moments where something of importance occurs. The events can be described in terms of a period ontology – where an event is defined as unfolding during some period; or a time point ontology – where the event possess a starting point and ending point. The latter is the point of view taken in this dissertation: a musical piece contains events and a musical event, note or rest, has an onset (start time) and offset (end time). We do not refer to time outside the unfolding of events.

Temporal relations With events that extend in time, one can adopt the full range of temporal relations that are commonly defined to relate periods of time (Allen, 1983; Marsden, 2000, chap.3). We use a slightly different strategy here and focus on four temporal relations that we propose as particularly relevant for musical events. These are inspired by the musical texture discussed earlier. In the case of monophony, the events are aligned in a sequence. Every event directly follows its predecessor. This is the “meet” relation: an event **a** meets with **b**, written $m(a, b)$, if the offset of **a** coincides with the onset of **b**.

In homophony, the m temporal relation appears as well. In addition, events overlap in time and, in particular, start at the same time. This is the “start together” relation: two events **a** and **b** start together, written $st(a, b)$, if they share the same onset.

In polyphony, the m and st relations can still occur. In addition, there are cases where events overlap in time without starting together. To capture such cases, we introduce two temporal relations: an “overlap” relation that simply states that two events unfold together at some point in time and the “starts while” relation that specifies which of the overlapping events starts first and which starts second (assuming that they do not start together). More precisely, i) two events **a** and **b** overlap, written $ov(a, b)$, if there is some point in time where both events are unfolding and ii) an event **a** starts while **b** is unfolding, written $sw(a, b)$, if the onset of **a** occurs after the onset of **b** but before its offset. Notice that the sw is “strict”: it implies that events do not start together. This is mainly justified by the need to precisely represent the suspension pattern which is characterized by the fact that the notes forming the pattern do not start at the same time (see Section 2.4 below).

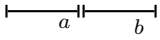
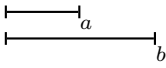
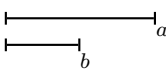
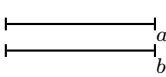
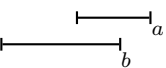
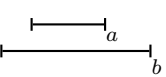
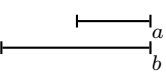
Using a “dot” notation to refer to the onset and offset of events, the four temporal relations we consider can be defined as follows:

Temporal relation Holds when

$m(a, b)$	$a.offset = b.onset$
$st(a, b)$	$a.onset = b.onset$
$sw(a, b)$	$a.onset > b.onset$ and $a.onset \leq b.offset$
$ov(a, b)$	there is a time t such that: $a.onset \leq t \leq a.offset \text{ and } b.onset \leq t \leq b.offset$

The relations can also be compared to the full catalog of period relations defined by Allen (1983) – as a general temporal knowledge representation scheme – and

re-investigated later for music by Marsden (2000, chap.3). They can be expressed in terms of the general period relations as follows:

$m(a, b)$	is	$a \text{ m } b$		a “meets” b
$st(a, b)$	covers	$a \text{ s } b$		a “starts” b
		$b \text{ s } a$		b “starts” a
		$a = b$		a “equals” b
$sw(a, b)$	covers	$b \text{ o } a$		b “overlaps with” a
		$a \text{ d } b$		a “during” b
		$a \text{ f } b$		a “finishes” b
$ov(a, b)$	covers	all the cases above (and their inverse), except $a \text{ m } b$		

Using the shorthand notation of Allen (1983), their expression is as follows (the suffix **i** expresses the inverse relation):

Temporal relation	Disjunction of period relations
$st(a, b)$	$a [s \text{ si } =] b$
$sw(a, b)$	$a [oi \text{ d f}] b$
$o(a, b)$	$a [s \text{ si } = \text{ o oi d di f fi}] b$

Notice that the temporal relations that we define are not specific about the end of the events.

2.3.2 Musical relations

Many aspects of music representation depend on concepts of music theory. The reader is referred to Appendix A or (Dowling and Harwood, 1986, chap.1) for an introduction to these notions. They mainly revolve around the organization of pitch and support different abstractions over musical data that play an important role in the definition of musical patterns.

Monophony For monophonic music, the notion of musical pattern has been explored in depth in recent years (see Cambouropoulos et al., 1999, for an overview). A strong consensus emerged that an approach to monophonic patterns must at least capture the notion of *transposition*. A melodic sequence is transposed if it starts on a different pitch than the original but preserves the distance between any two subsequent notes. Consider, for example, the monophonic excerpts of Figure 2.7. Melody d) is the transposition downward of melody c) by the interval of a perfect fifth (P5 or seven units in the piano-roll notation). The distance between subsequent notes stays the same, e.g. in both cases the first and second notes are separated by the interval of a major second (M2 or two units in the piano-roll notation). Appendix A explains the music-theoretic notion of *interval* and the values it can take.

Transposed melodies are usually considered to be equivalent to the original and listeners will remember and recognize a transposed melody as easily (Dowling and Harwood, 1986, chap.4). There is also evidence that a listener will easily recognize melodies that preserve the same contour but not necessarily the same exact intervals. Furthermore, listeners cannot easily distinguish between the transposed instances and the contour-preserving instances (Dowling and Harwood, 1986, chap.4, p.134). Figure 2.7 b) and e) present such a transformed melody. For pitch, the contour simply refers to how a note is higher, lower or equal to its predecessor. For duration, the contour refers to how a note is shorter, longer or equal to its predecessor.

The importance of contour calls for a slightly more subtle view of musical relations, even for the simple monophonic case. Rather than considering only transposition, many different relations of a note with its predecessor may be considered. The viewpoint method introduced by Conklin and Witten (1995) is perhaps the most developed around that theme. It would assign each note with a value – called *feature value* by Conklin and Bergeron (2008) – for different viewpoints – called *features* by Conklin and Bergeron (2008). This is illustrated by Figure 2.7 e). Every musical excerpt that is assigned the same sequence of feature values is considered as an instance of the same pattern.

In such a scheme, every note of a pattern is described according to the same features, here pitch contour and duration contour. It has been argued recently that a pattern language should support cases where different notes form different relations (Lartillot, 2004; Cambouropoulos et al., 2005; Conklin and Bergeron, 2008). This is accommodated by Conklin and Bergeron (2008) with a representation where notes are represented by *feature sets*, where notes of a pattern can be described by different mixtures of relations.

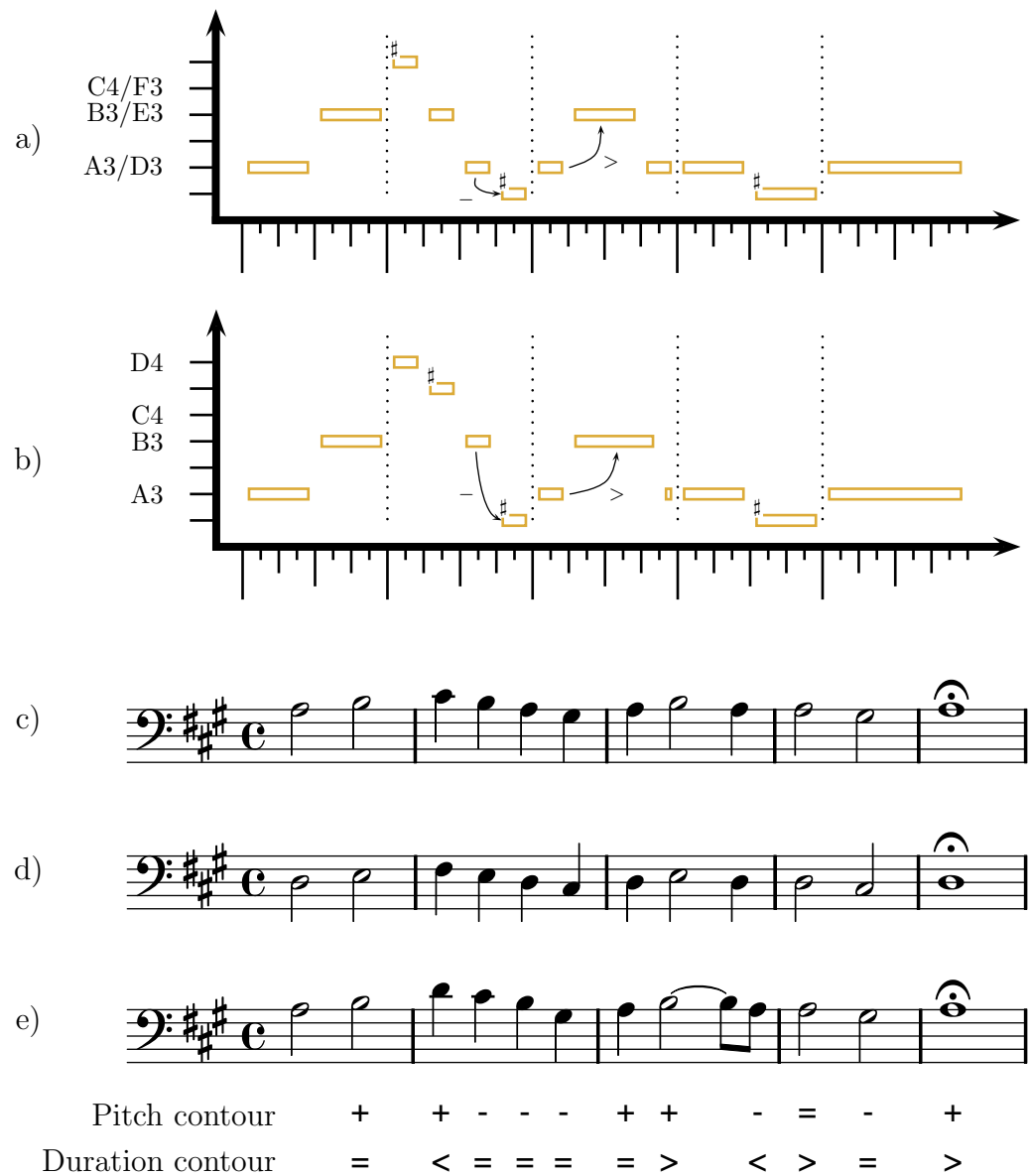


Figure 2.7: Examples of monophonic transposition and transformation. a) and c): first five bars of the tenor voice of BWV 323; a) and d): transposed by a fifth down; b) and e): transformed while conserving pitch and duration contours

Homophony and polyphony For homophonic and polyphonic music, the need for flexibility is understandingly greater than for monophonic music. Even if many approaches to polyphonic patterns still consider transposition alone (see Chapter 3), it is perhaps even more apparent for polyphony that patterns are very diverse and require great care in their representation. Consider the musical excerpts of Figure 2.8. Excerpt d) is the transposition downwards of excerpt c) by the interval of a perfect fifth (P5).

Figure 2.8 b) and e) depicts a transformation of the original excerpt that is likely to be considered as very similar, both perceptually and in terms of music-theoretic notions. Intervals between notes that sound together – *harmonic* intervals – are slightly more subtle than melodic intervals. Harmonic intervals obey the principle of octave equivalence (one of the three musical universals according to Dowling and Harwood, 1986, p.4): pitches separated by more than one octave are still considered to form intervals ranging from the unison (no interval) to the seventh. In particular, the interval of an octave is assimilated with the unison. Such octave-equivalent intervals are called *compound* intervals. Compound intervals are used to define the parallel fifth pattern introduced in Chapter 1. A parallel fifth pattern cannot be captured by transposition as it needs to encompass both the instances where fifths are repeated and where twelfths (an octave plus a fifth) are repeated.

The compound interval is the default interval in this dissertation. However, it is sometimes useful to include octaves in the notion of intervals. We call these *non-compound* intervals. To name non-compound intervals, one simply continues to count units after seven: an octave is an eighth, an octave plus a second is a ninth, and so on.

Consonance Harmonic intervals are typically classified into two categories: consonant and dissonant. The latter incite motion where the former are more restful (Piston, 1941). The suspension pattern introduced in Chapter 1 relies on this notion. Again, it cannot be captured by transposition, as intervals can vary from one suspension to another.

2.4 Motivating patterns

Using the temporal and musical relations introduced above, we can give a precise definition of the parallel fifth and suspension patterns introduced in Chapter 1. At

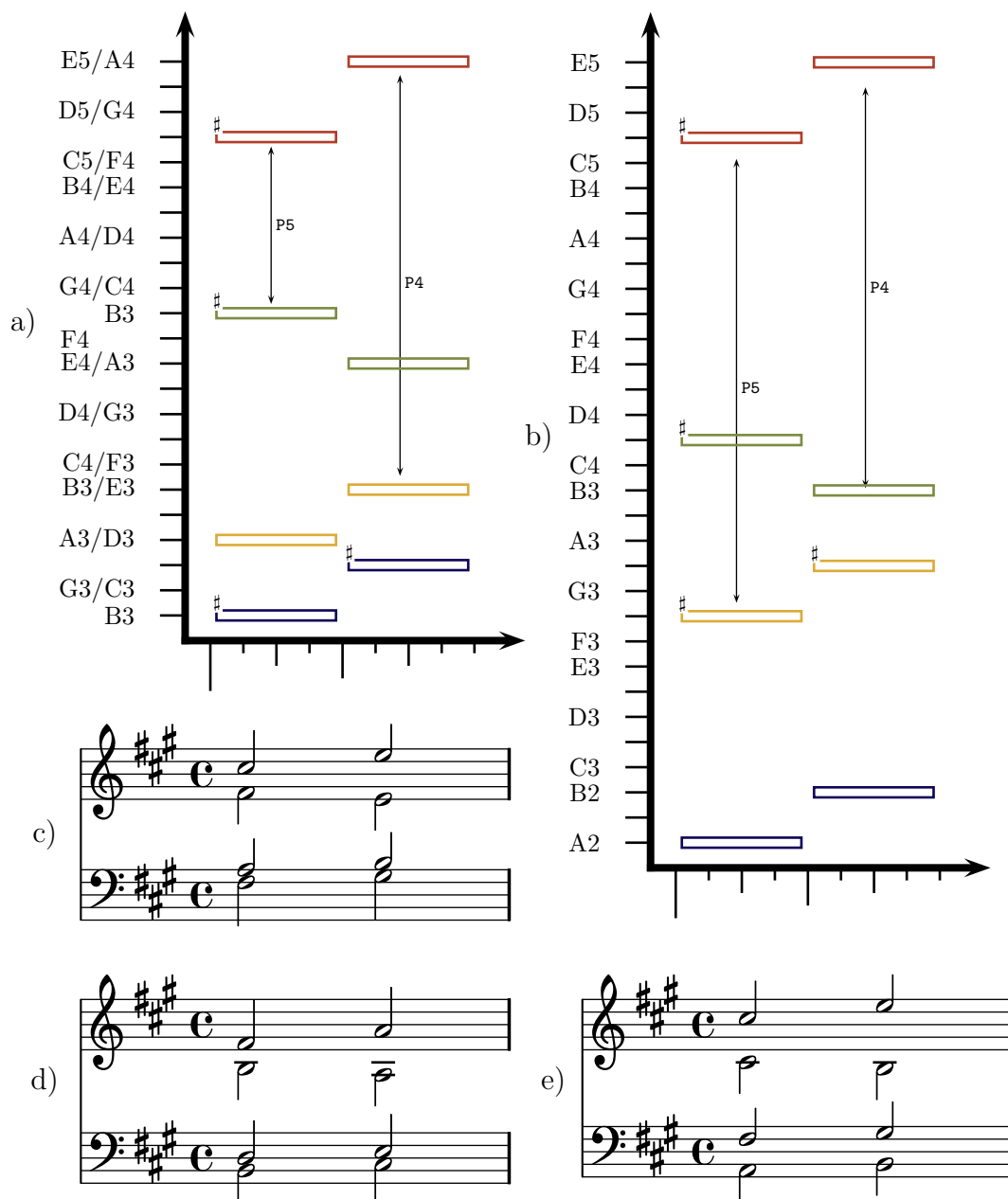


Figure 2.8: Examples of homophonic transposition and transformation. a) and c): first two bars of BWV 323; a) and d) transposed by a fifth down; b) and e): transformed so that chords are preserved and that the soprano voice forms the same harmonic intervals (albeit not necessarily with the same voice)

the same time, this illustrates the need for such precise relations and how challenging it can be to capture a polyphonic pattern.

Parallel fifth Consider the parallel fifth of Figure 2.9. We can think of a parallel fifth as a sequence of two layers. The first layer contains two notes, which form the diatonic interval of a perfect fifth. The second layer also contains two notes, which also form the diatonic interval of a perfect fifth. From the instance of Figure 2.9, a parallel fifth pattern can be developed by applying the following abstractions. First, we must capture cases where the fifth is compound, e.g. where two diatonic intervals of a twelfth are repeated.

Then, in the instance, the first and second layers both contain notes in the soprano and alto voices. In the pattern, these can be any two voices. The fact that the voices stay the same from the first layer to the second must however be enforced.

Finally, both layers exhibit an **st** (start together) temporal relation in the instance of Figure 2.9. This is not the general case for a parallel fifth, however. Following Fitsioris and Conklin (2008), we consider that the cases where the first layer exhibits an **ov** (overlap) temporal relation are also valid instances. Consequently, the pattern must express both the **ov** and **st** temporal relations. Figure 2.9 illustrates a case requiring the **ov** with dashed notes in the piano-roll notation.

Suspension The suspension pattern requires the same amount of sophistication. Consider, for example, the suspension of Figure 2.10. The pattern can be seen as layered sequences. In the instance, the top sequence holds two notes while the bottom sequence holds a singleton (the $F\sharp$ at the bottom of Figure 2.10). Again, the pattern must abstract the fact that the instance occurs between the tenor and bass voice and express that a suspension can occur between any two voices. At the heart of the suspension is the idea that the singleton note introduces a dissonance and that the final note of the pattern resolves that dissonance to a consonance. In the instance, the dissonance is the diatonic interval of a perfect fourth (between the $F\sharp$ in the bass voice and the B in the tenor voice) and the consonance is the diatonic interval of a major third (between the $F\sharp$ in the bass voice and the $A\sharp$ in the tenor voice). The pattern must abstract away from a particular pair of intervals and simply specify that a dissonance is followed by a consonance.

The next abstraction step is to specify the temporal relations involved. In the instance, the singleton note starts while the first note of the pattern is already

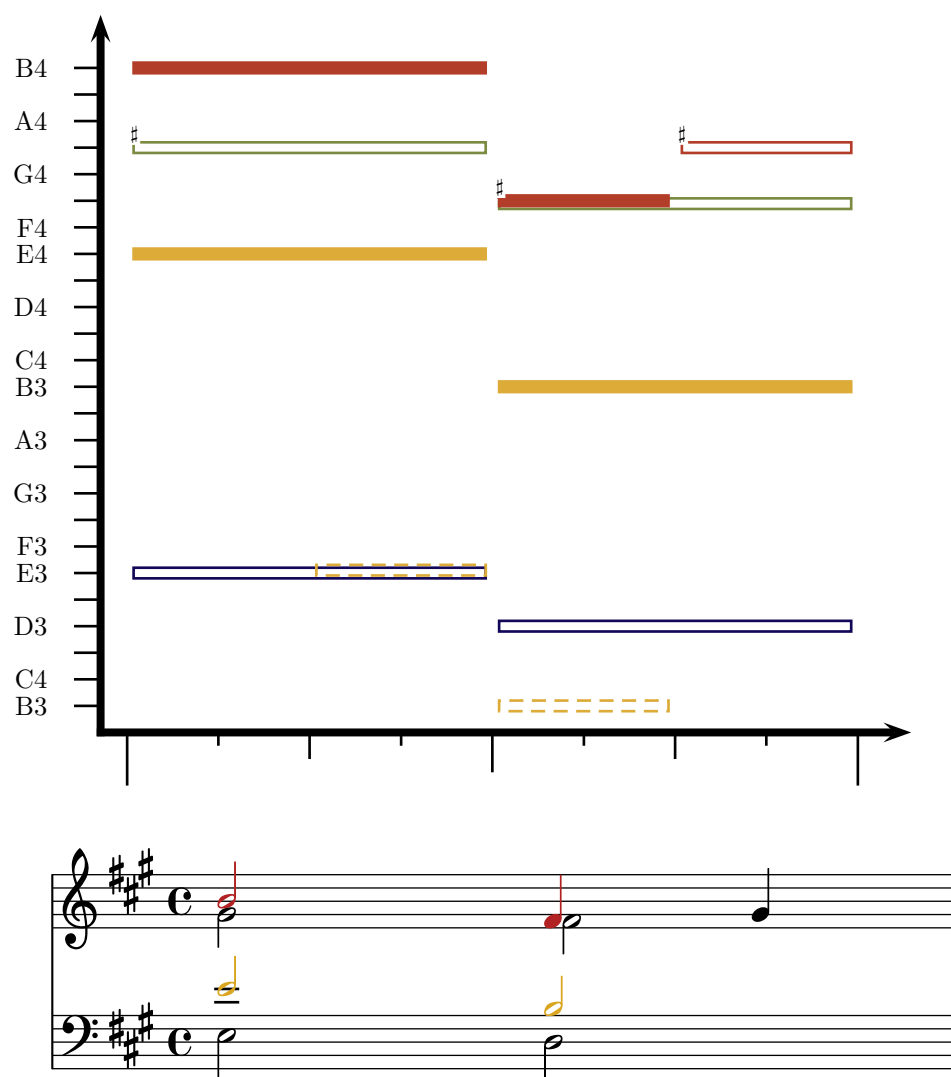


Figure 2.9: Eighth bar of BWV 323. The parallel fifth pattern is highlighted. Every dashed note added to the piano-roll would also form a parallel fifth

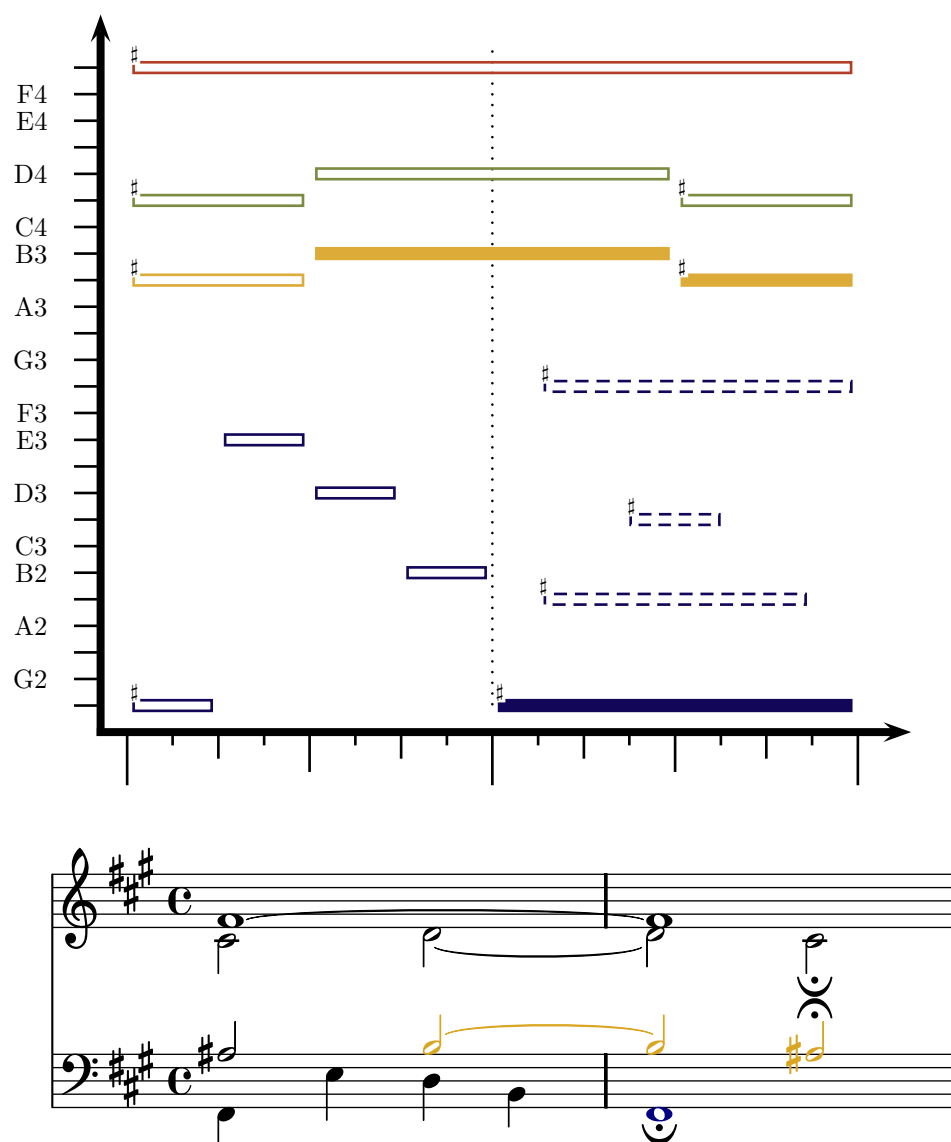


Figure 2.10: Final bar of BWV 323. The suspension pattern is highlighted. Every dashed note added to the piano-roll would also form a suspension

sounding. Similarly, the last note of the pattern starts while the singleton note is already sounding. This is characteristic of a suspension pattern. Figure 2.10 shows multiple cases with dashed notes that would also form a suspension. To express the suspension in every case, a pattern language must support the **sw** (start while) temporal relation. As we will see in Chapter 3, many existing approaches to polyphonic patterns were not designed to support this relation and hence cannot represent the suspension pattern.

2.5 Requirements for a polyphonic pattern language

Equipped with the musical concepts introduced in this chapter, we can refine the six objectives stated in Chapter 1. The result is precise enough to form a set of requirements for a polyphonic pattern language.

Requirement 1: Fully support n-part polyphony This is mainly a requirement about how musical data is encoded. In particular, the following must be satisfied:

- Encode musical events in an unrestricted way, i.e. the source should not be restricted to a monophonic or homophonic texture
- The source encoding should support an unlimited number of parts. Each musical event must belong to a part

Requirement 2: Expressive and precise The expressiveness requirement can be refined as follows:

- The pattern language should express at least the four temporal relations introduced in this chapter: **m**, **st**, **sw**, and **ov**
- The pattern language should express musical relations between contiguous notes. This includes, at least, notes that belong to the same part and meet; and notes that belong to different parts and overlap in time.

The precision requirement can be defined as follows:

- For every expressible pattern, the approach should return every instance of that pattern that exists in the source
- Every instance should precisely match the temporal and musical relations that the pattern expressed

The expressiveness and precision requirements can be reformulated using our motivating examples: the approach should be able to express and retrieve every parallel fifth and suspension in a corpus of polyphonic music.

Requirement 3: Clear syntax and semantics This requirement is two fold:

- Temporal and musical relations can be written down in a simple and unambiguous way
- The meaning of a pattern can be easily inferred from the syntax

Requirement 4: Flexible A flexible approach supports many different queries:

- The approach should include a mechanism to define musical relations
- The approach should support the definition of queries in a modular way: it should be possible to refine a query by adding a relation or slightly expanding its structure

Requirement 5: Easy to use and efficient This requirement is mainly qualitative. We argue, for example, that ease-of-use is increased by a clear syntax, especially if it reflects the temporal relations enforced by the pattern. Regarding efficiency, we argue that the pattern matching algorithm should have a linear or log-linear time complexity with respect to the number of musical events. This is discussed in detail in Chapter 7.

Requirement 6: Conducive to music data mining This refers to how the temporal and musical relations are expressed. In a sense, one can view polyphonic patterns as carefully crafted combinations of relations. If the space of combinations can be explored automatically, we say the language is conducive to music data mining. Ideally, the language should restrict the space of patterns as much as possible,

while still satisfying all previously stated requirements. This increases the possibilities for tractable pattern discovery algorithms. Although it is outside the scope of this dissertation to develop a music data mining method, we argue in Chapter 7 that the pattern matching language we propose is a good candidate language for such an application.

2.6 Summary

This chapter presented the notions of musical source and musical pattern in detail. Polyphonic patterns are a challenging application for computer science as they require us to precisely express different kinds of relations. We refined the objectives of our research into a set of requirements. In the next chapter, we review the existing approaches to the problem of querying polyphonic music data.

Chapter 3

Related work

This chapter discusses the existing computational approaches to the general task of querying polyphonic music data. With respect to the requirements for a polyphonic pattern language elaborated in Chapter 2, three approaches deserve more attention: i) relational patterns, ii) Humdrum and iii) Structured Polyphonic Patterns. In particular, these approaches are both expressive enough to represent the parallel fifth and suspension patterns discussed so far and precise enough to retrieve the exact list of all the instances of such patterns. Other approaches are either too restrictive as for example they cannot express the `sw` temporal relation, or are not precise enough as for example they use some kind of approximate pattern matching algorithm that is not guaranteed to retrieve all the instances of a pattern. We provide a classification of these approaches according to their underlying principles. For example, a number of approaches stem from the concept of vertical layers discussed in Chapter 2.

3.1 Expressive and precise approaches

In the following sections, we discuss in detail three approaches to polyphonic patterns: i) relational patterns inspired by (Fitsioris and Conklin, 2008), ii) Humdrum (Huron, 1999), and iii) Structured Polyphonic Patterns (Bergeron and Conklin, 2008). The three approaches possess the expressiveness required to represent the parallel fifth and suspension patterns. In particular, the temporal relations and musical relations introduced in Chapter 2 can be expressed unambiguously. This is not the case for most approaches to polyphonic pattern retrieval, either because

no mechanism is provided to abstractly specify the temporal aspect of a pattern or because some temporal relation is not supported. For example, in the case of approaches based on layers (or similar vertical structures), the **st** (start together) temporal relation is expressed but not the **sw** (start while) temporal relation, which is essential in expressing the suspension pattern. This is circumvented in some approaches (Huron, 1999; Dubnov et al., 2003) by using a continuation marker, a special symbol that indicates that some note in a layer does not start as the layer starts but is rather the continuation of some previously sounding note. In *SPP*, the concept of layer is extended in a different way, by using an onset modification operator. Rather than expressing the continuation of a musical note, it simply states that some note starts before the beginning of the layer. The issue of expressiveness with respect to temporal relations is formally analyzed in Chapter 6.

The other key property that relational patterns, Humdrum and *SPP* all possess is a precise matching algorithm that returns exactly all the instances of a pattern. This is very important for the kinds of applications that this dissertation considers, for example the testing of a musicological hypothesis or the gathering of facts about music. An approximate matching algorithm has other advantages however, and is suitable for other motivations reviewed in Section 3.2.

3.1.1 Relational patterns

Using the logic programming language Prolog, Fitsioris and Conklin (2008) search for every instance of the parallel fifth pattern in Bach chorale harmonizations. A representation for polyphonic music is developed encompassing both basic information about musical events and a series of relations that the events form. For example, Figure 3.1 shows the basic information of four notes: **a**, **b**, **c**, **d**. The statements concerning note **a** are simply read as follows: the onset of **a** is 0, the offset of **a** is 2, the pitch of **a** is B4, and **a** belongs to the **soprano** voice. Note that for readability, we adapted the representation of Fitsioris and Conklin (2008) and made it more consistent with the nomenclature of the current dissertation.

The relations that Fitsioris and Conklin (2008) propose are similar to the temporal relations and musical relations discussed in Chapter 2. These can either be represented as statements or given definition in the Prolog programming language. Figure 3.2 shows the latter case, where each temporal relation is defined as a rule. On the left-hand side of the rule is the target relation. On the right-hand side is a list of literals that must be satisfied for the rule to hold. For example, the **m** (meet)

<code>onset(a,0).</code>	<code>onset(c,2).</code>
<code>offset(a,2).</code>	<code>offset(c,3).</code>
<code>pitch(a,'B4').</code>	<code>pitch(c,'F4#').</code>
<code>voice(a,soprano).</code>	<code>voice(c,soprano).</code>
<code>onset(b,0).</code>	<code>onset(d,2).</code>
<code>offset(b,2).</code>	<code>offset(d,4).</code>
<code>pitch(b,'E4').</code>	<code>pitch(d,'B3').</code>
<code>voice(b,tenor).</code>	<code>voice(d,tenor).</code>

Figure 3.1: Prolog encoding of four notes forming a parallel fifth pattern

temporal relation holds between event A and B when the offset of A is X and the onset of B is also X. The `ov` (overlap) temporal relation is specified through four cases, by reusing the definition of the `st` and `sw` temporal relations.

<code>m(A,B) :-</code>	<code>st(A,B) :-</code>
<code>offset(A,X),</code>	<code>onset(A,X),</code>
<code>onset(B,X).</code>	<code>onset(B,X).</code>
<code>sw(A,B) :-</code>	
<code>onset(A,X),</code>	<code>ov(A,B) :- st(A,B).</code>
<code>onset(B,Y),</code>	<code>ov(A,B) :- st(B,A).</code>
<code>offset(B,Z),</code>	<code>ov(A,B) :- sw(A,B).</code>
<code>X > Y,</code>	<code>ov(A,B) :- sw(B,A).</code>
<code>X <= Z.</code>	

Figure 3.2: Prolog definition of the four temporal relations considered in this dissertation

Once temporal relations and musical relations are given a definition, the parallel fifth pattern is expressed by the rule shown in Figure 3.3. Again, the query is adapted from Fitsioris and Conklin (2008) to make it more consistent with the notation used in this dissertation. It can be read as follows: the notes A, B, C, D form a parallel fifth if the following temporal relations are satisfied: A and B overlap; A meets C and B meets D; and C and D start together. These are the same temporal relations discussed in Section 2.4. In addition, the following musical relations must be satisfied: the interval between C and D is a perfect fifth (`interval(C,D,'P5')`); there is some melodic motion involved (`not(motion(A,B,'='))`); the interval between A, B and C,

```

par5th(A,B,C,D) :-
    ov(A,B),          higher(C,D)
    m(A,C),           interval(C,D,'P5'),
    m(B,D),           not(motion(A,B,'=')),
    st(C,D),          interval_nc(A,B,I),
    voice(A,X),       interval_nc(C,D,I),
    voice(C,X),
    voice(B,Y),
    voice(D,Y),
    not(X=Y).

```

Figure 3.3: The parallel fifth pattern in Prolog. Reproduced and adapted from Fitsioris and Conklin (2008)

D is the same, even in the non-compound representation where octaves are explicitly taken into account (`interval_nc(A,B,I)` and `interval_nc(C,D,I)`). Finally, there are some constraints that are included to make the query completely precise: the fact that the pitch of `C` is higher than that of `D` (`higher(C,D)`) ensures that a parallel fifth is matched only once and constraints about voicing such as `voice(A,X)` ensure that the fifth is repeated by the same two voices.

Although Fitsioris and Conklin (2008) develop a single query, we can easily see how the approach could support many other queries. Logic programming languages such as Prolog are often used as generic query languages (De Raedt, 2002). The advantage of such an approach is its expressiveness. As Prolog is a general programming language, any discrete dataset can be represented and any computational query executed. In practice, however, an approach based on a logic programming language will focus on data that has a certain structure and on queries that are expressed using a limited set of relations. This is why we prefer to refer to this approach as relational patterns.

Executing a query in Prolog is like executing a program. The time complexity depends on the way the query is written and, in principle, the query could even be impossible to execute (i.e. the program could enter some kind of infinite loop). For these reasons, using Prolog or any constraint language may be easier with a intermediary language that restricts the kinds of queries that a user can define. With respect to temporal relations, such a restriction is proposed in Chapter 6.

Although it is in principle possible to design a music data mining method around relational queries, it is in practice difficult to do so. The problem lies in the fact that

the search space to explore is very big. Using *SPP* as a “tighter” approximation of the relevant search space to explore seems a promising avenue of research. This issue is further discussed in Section 7.6.

3.1.2 Humdrum

Humdrum (Huron, 1999) is centered around the representation of musical pieces as formatted text files. A Humdrum file contains multiple columns, called *spines*, one for each part of the piece it encodes. Consider for example Figure 3.4 showing the final bar of Bach chorale BWV 323 as a Humdrum file.

**kern	**kern	**kern	**kern
*I:[soprano]	*I:[alto]	*I:[tenor]	*I:[bass]
*k[f#c#g#]	*k[f#c#g#]	*k[f#c#g#]	*k[f#c#g#]
*M4/4	*M4/4	*M4/4	*M4/4
[1f#	2c#	2A#	4FF#
.	.	.	4E
.	[2d	[2B	4D
.	.	.	4BB
=	=	=	=
1f#]	2d]	2B]	1FF#
.	2c#	2A#	.

Figure 3.4: Final bar of BWV 323 in Humdrum/kern encoding

Time flows from top to bottom in a Humdrum file: each line can be considered as a layer. Any two *tokens* appearing on the same line are considered to start together. For example, the fifth line of Figure 3.4 encodes a layer where the soprano voice sounds a **F#**, the alto voice a **C#** and the tenor and bass voice respectively a **A#** and **F#**. The following line shows how a continuation token “.” is used in Humdrum to represent the polyphonic texture: the bass voice now sounds an **E** while all other voices are still sounding the same pitch. The particular encoding of pitches and durations can vary in Humdrum. The encoding of absolute pitches as letters and durations as numbers is called the *kern* encoding. Other symbols of Figure 3.4 include comments regarding the encoding used, voice names, key and time signatures (lines one to four), bar lines (line nine) and the opening and closing of ties [/] (e.g. line seven and ten).

The pattern matching method of Humdrum is centered around a set of command-line tools for processing and transforming Humdrum files. Most of the temporal and

musical relations presented in Chapter 2 can be expressed through careful preprocessing of the source. For example, to express an **sw** temporal relation in a Humdrum pattern, the command-line tool *ditto* has to be executed. Its effect is to replace every continuation token “.” with the token being continued in surrounding brackets. Figure 3.5 shows the results of applying *ditto* and other processing to the Humdrum file of Figure 3.4. The third line, for example, shows that the **A#** is the continuation of a previously sounding note. Another way to read this is that the **E** in the first column starts while the **A#** is unfolding.

bass	tenor	cons
FF#	A#	T
E	(A#)	F
D	B	T
BB	(B)	T
FF#	(B)	F
(FF#)	A#	T

Figure 3.5: Final bar of BWV 323 in Humdrum/kern encoding after the preprocessing required for the execution of a suspension query

Figure 3.5 also displays other transformations: the bass and tenor voices were extracted; durations, ties and bar lines were removed; an additional column with a consonance/dissonance feature was added. To do this in a general way, in particular to extract every possible two-voice combination, one can use a shell script as shown by Figure 3.6.

The last line of Figure 3.6 executes the pattern matching command. A pattern in Humdrum consists of a text file where every line contains a regular expression. This series of regular expressions matches a series of contiguous lines in a Humdrum file, where every regular expression matches a line. Again, the concept of layer is present, with each regular expression matching one layer in a sequence of layers. The content of the file **susp** in Figure 3.6 is as follows:

```
[a-g A-G]+ [- # n]* [^)]* [(] [a-g A-G]+ [- # n]* .* F$
[a-g A-G]+ [- # n]* [)] [^(]* [a-g A-G]+ [- # n]* .* T$
```

The first line matches the layer where a note starts in the first column while some note is already sounding. The expression `[a-g A-G]+ [- # n]*` at the beginning of the line matches any pitch followed by any accidentals (or none). The two key expressions of the line are `[^)]*` and `[(]`. The first one ensures that the first note

```
for i in soprano alto tenor bass do
  extract -i *I:[$i]      in.krn          > $i.krn
  extract -i **kern       $i.krn         > $i.krn

for i,j in soprano alto tenor bass do
  if i!=j then
    assemble              $i.krn $j.krn   > tmp.krn
    clean                 tmp.krn         > tmp.krn
    ditto -p              tmp.krn         > tmp.krn
    hint                  tmp.krn         > hint.txt
    recode -f cons        hint.txt         > cons.txt
    assemble              tmp.krn cons.txt > query.krn

pattern -f susp          query.krn        >> instances.txt
```

Figure 3.6: Execution of a Humdrum suspension query

of the layer is not a continuation, i.e. that the pitch is not enclosed in brackets as a result of executing the ditto command (the caret is the negation symbol in the regular expression notation used in Humdrum). The second expression expresses the converse: that the second note of the layer is a continuation, i.e. it was enclosed in brackets after the execution of a ditto command. The second line of the pattern matches the inverse situation, where the first note of the layer is being continued and the second note starts. This expresses the temporal relations of the suspension. To match the consonance and dissonance involved in the suspension pattern, the regular expressions in the pattern respectively end with the expressions `.F$` and `.T$`. This matches the additional column of Figure 3.5 that was specifically added to represent consonances and dissonances. The consonance feature could not be expressed through a regular expression. The same is true for most musical relations one could want to express in a query. This shows how much the pattern matching mechanism of Humdrum depends on an adequate preprocessing of the source.

Consequently, the semantics of a Humdrum query can be somewhat opaque and it can be hard to be confident in the precise meaning of a pattern. Also, to develop a query one needs to be familiar with command line and scripting tools. In the words of Jan (2004): “It is worth noting that effective use of the toolkit in its ‘raw’ state requires a degree of facility with UNIX that many musicologists without technical backgrounds are unable or unwilling to devote time to acquiring”.

3.1.3 Structured Polyphonic Patterns

The *SPP* pattern language (Bergeron and Conklin, 2008) is based on two ideas: the structuring of music into sequences and layers discussed in Chapter 2 and the concept of feature set introduced recently for music data mining (Conklin and Bergeron, 2008). A feature set is a representation of a musical event as a collection of attributes. Structured Polyphonic Patterns extend the concept in two ways: i) a feature is now labelled with a voice name, i.e. it belongs to a voice and ii) a feature can optionally refer to some other voice, hence encoding a relation between the current voice and that other voice. Figure 3.7 shows how to encode as *SPP* events the four notes encoded in Prolog in Figure 3.1.

$$\begin{aligned} &\{\text{onset} : 0, \text{offset} : 2, \text{pitch} : \text{B4}\}_{\text{soprano}} \\ &\{\text{onset} : 2, \text{offset} : 3, \text{pitch} : \text{F4}\sharp, \text{parallel}(\text{tenor}) : \text{T}\}_{\text{soprano}} \\ &\{\text{onset} : 0, \text{offset} : 2, \text{pitch} : \text{E4}\}_{\text{tenor}} \\ &\{\text{onset} : 2, \text{offset} : 4, \text{pitch} : \text{B3}, \text{parallel}(\text{soprano}) : \text{T}\}_{\text{tenor}} \end{aligned}$$

Figure 3.7: *SPP* encoding of four notes forming a parallel fifth pattern

The onset, offset and pitch appear as features within the set. The voice statement appears as an index, or voice label, appended to each set. In addition, the feature `parallel` captures a relation with another voice. The second event of the soprano voice, for example, exhibits parallel motion with the tenor voice.

The biggest difference between the relational representation of Figure 3.1 and *SPP* is that relations in *SPP* are encoded as features, or propositions. This point of view, called positionalization by Kramer (2000), is the main abstraction behind the concept of feature set. The abstraction enables faster pattern matching as the instantiation of pattern components is self-contained, while relations require the exploration of many events to determine the instances of a relation. Also, as argued by Conklin and Bergeron (2008), this facilitates the manipulation of patterns and restricts the pattern space in a way that renders data mining possible.

Finally, note how the source is not encoded with sequences and layers. In *SPP*, these concepts are used to define a pattern. For example, Figure 3.8 shows the parallel fifth pattern in *SPP*.

The two pattern components on the left form one layer and the two components on the right form a second layer. Layers are expressed via a layering operator “`==`”. In addition, the layers are placed in a sequence through the “`;`” operator. By

$$\frac{-\{\text{et_interval}(y) : \text{P5}\}_x}{-\{\}_y} ; \frac{\{\text{st_interval}(y) : \text{P5}, \text{parallel}(y) : \text{T}\}_x}{\{\}_y}$$

Figure 3.8: Parallel fifth pattern in *SPP*

default, a layer expresses the **st** temporal relation. In Figure 3.8, this is the case of the second layer. To specify other temporal relations, *SPP* extends the concept of layer through the introduction of an onset modification operator “−”. Conceptually, a component prefixed with that operator starts before the beginning of the layer. To specify the **ov** temporal relation between two components, it suffices to prefix both components with “−”. Conceptually, both components extend back in time past the beginning of the layer. Hence, is it unknown whether they start together or, if not, which starts first. All that is specified is that they overlap in time as they are both unfolding when the layer starts.

To specify the **sw** temporal relation, it suffices to prefix one of the components with the “−” operator and leave the other untouched. Conceptually, the former extends back past the beginning of the layer while the latter starts as the layer starts. This is the case of the second layer of the suspension pattern, shown in Figure 3.9.

$$\frac{-\{\}_x}{-\{\}_y} ; \frac{\{\text{sw_cons}(y) : \text{T}, \text{m_size} : \text{S}, \text{m_dir} : -\}_x}{-\{\text{sw_cons}(x) : \text{F}, \text{accented} : \text{T}\}_y}$$

Figure 3.9: Suspension pattern in *SPP*

Notice how the voice names present in the source encoding are replaced by voice variables in the patterns, here **x** and **y**. This is one of the key characteristics of *SPP*. By using variables, the voice combinations are accounted for in a declarative sense, in contrast to Humdrum in which they must be explicitly enumerated.

In Figure 3.9, for example, the voice variables **x** and **y** could be assigned to any combination of voice names, as long as **x** and **y** are assigned to different voices. In particular, the voice variable **x** appearing at the top of the layers could be assigned to a “low” voice such as the bass voice. The top-to-bottom ordering of a layer does not indicate how the pattern matches the different parts of a source. Also, when both the top and bottom components of a layer are labelled with the same voice variable, the whole layer will match events that are within the same part. This allows *SPP* to express, for example, block chord patterns over a source that has a

single part (e.g. an unvoiced piano piece).

Finally, notice how the *SPP* suspension pattern presented in Figure 3.9 is more precise than the Humdrum suspension pattern presented in Section 3.1.2. It uses more features, for example specifying that the final consonance is approached by a melodic interval of a step (feature `m_size : S`), with a downward direction (feature `m_dir : -`). In Humdrum, this would require adding more columns to the pre-processed source used for pattern matching. Every new column requires to extend the preprocessing script, which can be time-consuming. In *SPP* by contrast, the meaning of features is precisely expressed through feature definition rules. Figure 3.10 presents such a rule.

The rule applies everywhere the **where** clause matches and adds the feature defined by the **add** clause. These rules are applied to a source until it is saturated with features, i.e. until new features cannot be added using the rules. Alternatively, it is also possible to compute the rules in an on-the-fly manner, where the features are added to a source while the pattern is being matched.

Conceptually, the information involved in the suspension and parallel fifth pattern is presented by Figure 3.11.

```

where
{et_interval_nc(y) : I1}x ; {st_interval_nc(y) : I2, m_size : S}x*

add
parallel(y) :  if I1 = I2 ∧ S ≠ U   :  T
                else                  :  F

```

Figure 3.10: Example of an *SPP* feature definition rule

3.2 Restricted or approximate approaches

Reviewing the scientific literature more broadly reveals several computational approaches to the notion of musical patterns. Traditionally, the approaches were restricted to the monophonic case. Rolland and Ganascia (2002), for example, offer a good overview. A strong interest for the more general polyphonic case has emerged recently. There are around twenty distinct approaches to the problem of querying polyphonic music data and – except Humdrum – they were all developed in the last decade. Table 3.1 presents an overview of the approaches.

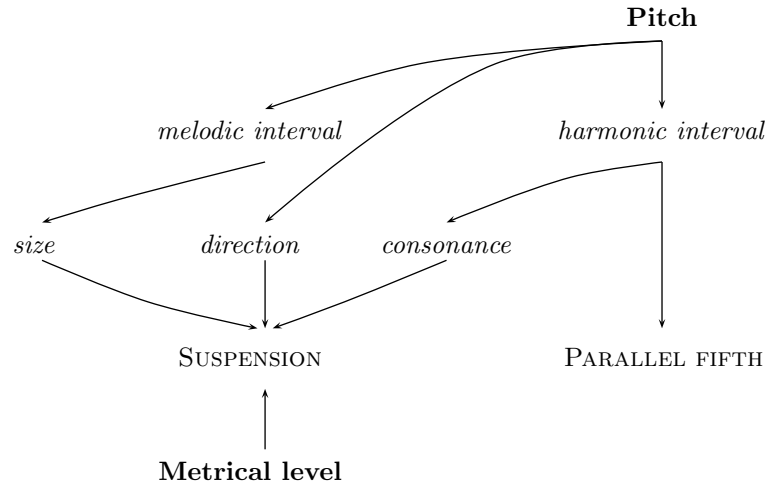


Figure 3.11: Overview of the information involved in the *SPP* suspension and parallel fifth patterns. Patterns are shown in CAPITALS. Features that are already present in the source are shown in **bold**. Features that are added to the source using *SPP* feature definition rules are shown in *italics*

The approaches can be classified according to their underlying principle: vertical structures (i.e. layers), graph representations, geometric representations and sequential approaches. Humdrum can be classified as a *vertical* approach. In general, the vertical approaches lack the expressiveness to represent the **sw** temporal relation and hence lack the expressiveness to represent the suspension pattern. Relational patterns can be classified as a *graph* approach where nodes correspond to events and edges correspond to relations. Existing graph approaches, however, use an approximate matching algorithm lacking the precision to return every instance of either a suspension or parallel fifth pattern. In general, *geometric* approaches are not designed to express temporal relations. Both source and patterns are encoded as sets of points in a geometric space where the dimensions are, for example, pitch and onset time. Finally, sequential approaches are designed to search for melodies within a polyphonic texture and hence cannot express polyphonic patterns.

Note that a restricted expressive power or an approximate matching mechanism are not necessarily drawbacks. It is usually a design choice, as for example restriction on the expressive power can yield faster matching algorithms. Also, approximate pattern matching is often required by the task at hand. For example, if we want to return sources or regions of a source that are approximately similar to some musical excerpt, precise matching is not wanted. The remainder of this section presents a more detailed analysis of the motivations and characteristics of existing approaches.

	Principle	Motivation	Source		Pattern						Data mining
			pre-processed	voiced	expressiveness	voiced	vertical relations	user features	syntax	precision	
(Huron, 1999)	vertical graph structured	musicology	•	•	Poly.	•	•	•	•	Exact	
(Fitsioris and Conklin, 2008)		musicology		•	Poly.	•	•	•	•	Exact	
(Bergeron and Conklin, 2008)		musicology		•	Poly.	•	•	•	•	Exact	
(Conklin, 2002)	vertical	musicology	•	•	Homo.	•	•	•	•	Exact	•
(Meudic and Staint-James, 2003)	vertical	musicology	•		Homo.					Approx.	•
(Rohrmeier and Cross, 2008)	vertical	musicology	•		Homo.					Approx.	•
(Dubnov et al., 2003)	vertical	generation	•		Poly.				•	Exact	•
(Hanna and Ferraro, 2007)	vertical	MIR			Homo.				•	Approx.	
(Pickens and Crawford, 2002)	vertical	MIR	•		Homo.					Approx.	
(Doraisamy and Rüger, 2003)	vertical	MIR	•		Homo.					Approx.	
(Szeto and Wong, 2006)	graph	musicology		•	Poly.	•			•	Exact	
(Madsen and Widmer, 2005)	graph	musicology		•	Poly.	•		•	•	Approx.	•
(Meredith, 2006)	geometric	N/A			Poly.					Exact	•
(Romming and Selfridge-Field, 2007)	geometric	MIR			Poly.					Approx.	
(Lubiw and Tanur, 2004)	geometric	MIR			Poly.					Approx.	
(Ukkonen et al., 2003)	geometric	MIR			Poly.					Approx.	
(Typke et al., 2004)	geometric	MIR			Poly.					Approx.	
(Dovey, 2001)	geometric	MIR			Poly.			•		Exact	
(Clausen et al., 2000)	geometric	MIR			Poly.		•			Exact	
(Utgoff and Kirlin, 2006)	sequential	N/A			Mono.					Approx.	•
(Pardo and Sanghi, 2005)	sequential	MIR			Mono.					Approx.	
(Lemström and Tarhio, 2000)	sequential	MIR	•		Mono.					Exact	

Table 3.1: Overview of existing polyphonic pattern approaches

Underlying principles

There are four main principles found in the literature to support different definitions of polyphonic patterns.

Vertical approaches A good number of approaches (Huron, 1999; Conklin, 2002; Meudic and Staint-James, 2003; Dubnov et al., 2003; Hanna and Ferraro, 2007; Pickens and Crawford, 2002; Doraisamy and Rüger, 2003) revolve around the idea that polyphonic patterns can be represented as a sequence of layers. Some components of the layers may be continuation markers that indicate that a previously occurring event unfolds across a layer boundary. This allows the representation of the **sw** temporal relation (Huron, 1999; Dubnov et al., 2003).

Graph approaches Some approaches (Szeto and Wong, 2006; Madsen and Widmer, 2005) define a polyphonic pattern as a graph. Nodes represent musical events and edges represent temporal relations between events. Similarly, the source is encoded as a graph. The expressiveness of such a pattern language can be tailored by adding new kinds of edges. Instances are simply subgraphs of the source. Szeto and Wong (2006) use a restricted form of graphs and Madsen and Widmer (2005) use a heuristics notion of graph similarity.

Geometric approaches Another popular principle is to define a polyphonic pattern as a set of points in a multidimensional space (Meredith, 2006; Romming and Selfridge-Field, 2007; Ukkonen et al., 2003; Typke et al., 2004; Dovey, 2001; Clausen et al., 2000). Typically, the dimensions used are onset and pitch and the source is also encoded as a set of points. A pattern instance is found when a translation exists that projects every point of the pattern to (or close to) a point of the source. This captures the musical concept of transposition. It fails to capture, however, the general concept of musical relation, which occurs locally, between two events of a pattern instance.

Sequential approaches Finally, some approaches (Utgoff and Kirlin, 2006; Pardo and Sanghi, 2005; Lemström and Tarhio, 2000) restrict the notion of pattern to a monophonic sequence of components to be matched against a polyphonic source. These approaches have the advantage of efficiency, but are limited in terms of applications.

Motivations

Significant differences between polyphonic pattern approaches arise from their respective motivations. This affects the kinds of features that the pattern language can capture. It also dictates important requirements such as efficiency, intelligibility (i.e. explicitly presenting patterns in some language) and flexibility (e.g. supporting user-defined features). We isolate three motivations: music information retrieval (MIR), automated composition and computational musicology. In addition, some approaches do not state any particular motivation.

Music information retrieval MIR approaches aim at quickly answering musical queries made by ordinary listeners. In one popular scenario, called *query by humming* (Pardo and Sanghi, 2005), the listener sings the melody of the piece of music he or she is searching for. The MIR system extracts melodic information from the user input, creates a melodic pattern and matches that pattern against a database of musical pieces. To be useful, such a system must tolerate errors in the query, as the user is not expected to sing the whole melody correctly. Also, it must execute quickly and return a list of possible answers the user can choose from. As execution time is crucial, errors in answering the query are also tolerated. The quality of the retrieval phase is evaluated by discussing its recall (the proportion of matches that were returned) and precision (the proportion of returned results that are true matches). Finally, there is no need in MIR to use an intelligible pattern language as the user is typically not interested in (and potentially not trained for) manually inspecting and crafting patterns.

Computational musicology Musical patterns are of interest both in general musicology (studying the music phenomena) and in the more specific discipline of music analysis (studying a particular piece or corpus). In the former, one can use patterns to formulate and verify hypotheses about music (Huron, 1999, 2001b). In the latter, one can use patterns to discover facts about a piece of music which can support the analysis of the piece (Conklin, 2002). Contrary to MIR, the user is presumably a music scholar whom is typically interested in defining detailed queries. A flexible and extensible approach is also required, as it is nearly impossible to anticipate every musical feature a scholar might want to use in a query. Also, as patterns are used to verify or discover facts, it is important for the pattern matching mechanism to be clearly and intelligibly defined. Consequently, musicological queries

are typically more demanding in terms of computational resources.

Automated composition Some approaches to automated composition directly use musical patterns (Cope, 1991). The idea is to extract patterns from existing musical material and compose new music by recombining and instantiating the patterns. A major difference with other approaches is that only a few patterns need to be extracted, rather than enumerating a list of results. While Cope (1991) uses monophonic patterns, the approach of Dubnov et al. (2003) applies to polyphony. It is based on a statistical model of musical style. Patterns are never used as such, but are rather implicit in the model being induced from examples. The approach differs from MIR or computational musicology as it does not only capture existing patterns, but also creates new patterns.

Source encoding

Pre-processing Different approaches use different representations of the source on which patterns are matched. Some approaches perform transformations on the source in order to facilitate pattern matching. These transformations, however, usually come at the price of losing some information about the musical surface. One popular idea is to represent polyphonic music as a sequence of layers. To do that, one must segment the musical surface according to some principle.

For example, Conklin (2002) and Rohrmeier and Cross (2008) create a sequence of layers by sampling the source at every beat. Meudic and Staint-James (2003) use a similar technique, but the onsets of beats are determined dynamically according to the material at hand. In polyphonic music, however, some notes may sound across a layer boundary. One way to deal with this problem is to segment long notes into a sequences of short notes. This is the approach taken, for example, by Huron (1999), Conklin (2002) and Rohrmeier and Cross (2008).

Voicing Some approaches consider that the musical material to be searched is voiced, i.e. that concurrent parts are identified (Huron, 1999; Conklin, 2002; Szeto and Wong, 2006; Madsen and Widmer, 2005). Other approaches consider the material to be unvoiced. Not representing voices affects significantly the kinds of patterns that one can search for and discover. For example, both the parallel fifth and suspension patterns discussed so far are expressed over explicitly identified voices.

Expressiveness and precision

Syntax Some approaches present the patterns in some abstract pattern language (Huron, 1999; Conklin, 2002; Dubnov et al., 2003; Hanna and Ferraro, 2007; Szeto and Wong, 2006; Madsen and Widmer, 2005). Other approaches (e.g. Meredith, 2006; Doraisamy and Rüger, 2003) define a pattern directly as the set of its instances, provided that the instances are similar to one another according to some similarity measure.

Contiguity A musical pattern is usually defined as a set of events that are contiguous in time. The exact definition of contiguity differs from one approach to another. Approaches like (Huron, 1999; Conklin, 2002; Madsen and Widmer, 2005) use a strict contiguity and events must meet (i.e. satisfy the `m` temporal relation). The approach of Utgoff and Kirlin (2006) uses a loose contiguity, where events can approximatively meet (with respect to some user-defined threshold). Finally, the approach of Meredith (2006) does not require events to be contiguous. Contiguous patterns are however considered more relevant.

Features All approaches consider musical relations in one form or another (with the sole exception of Dubnov et al., 2003). For approaches with an explicit pattern language, those features are usually named and values are taken from some discrete domain (Huron, 1999; Conklin, 2002; Madsen and Widmer, 2005; Fitsioris and Conklin, 2008). For approaches with no pattern language, the features are implicit and appear in how the similarity of pattern instances is computed. In both cases, the approach might or might not offer the user with the opportunity to define new features (or new components of the similarity measure).

Vertical relations Surprisingly, only three existing approaches support vertical relations (Huron, 1999; Conklin, 2002; Fitsioris and Conklin, 2008): musical relations formed by events that overlap in time. By contrast, a popular idea is to compare two polyphonic passages by first extracting a set of monophonic passages and comparing them using monophonic features (e.g. Madsen and Widmer, 2005; Meudic and Staint-James, 2003; Doraisamy and Rüger, 2003). The absence of vertical relations, however, decreases greatly the ability of an approach to represent meaningful polyphonic patterns like the suspension and parallel fifth.

Expressiveness An important characteristic of a musical pattern approach is its expressiveness. Some approaches will not support the definition of some kind of patterns, while others will only support the expression of a restricted form of the pattern. For example, the approach of Lemström and Tarhio (2000) only supports patterns that take the form of a sequence. Consequently, it is impossible to express a pattern of homophonic texture. In the approach of Conklin (2002), one can only express an approximation of the suspension pattern. As a result of prior transformation of the musical data, the approach cannot differentiate between homophonic texture and polyphonic texture, which is needed to fully express the suspension pattern. Note that this is not an intrinsic flaw of vertical approaches, however, as the combination of layers and continuation markers provides a polyphonic expressiveness.

In general, we say an approach has a monophonic expressiveness if it does not support a precise definition of homophonic patterns. Similarly, we say an approach has an homophonic expressiveness if it does not support a precise definition of polyphonic patterns.

Precision

Exact matching The pattern matching strategy differs from one approach to another. Some approaches use exact matching (e.g. Lemström and Tarhio, 2000; Dovey, 2001): each pattern component is matched to an event in a precise way.

Approximate matching In contrast, some approaches use some kind of fuzzy matching, where pattern component are matched to an event if some measure of similarity is higher than a predetermined threshold (e.g. Utgoff and Kirlin, 2006; Typke et al., 2004). In general, exact matching is used when the representation accommodates varying levels of abstraction, using abstract features. Fuzzy matching is interesting when patterns use a small number of fairly concrete features (e.g. pitch and duration).

Data mining

Only six approaches support the discovery of polyphonic patterns (Conklin, 2002; Meudic and Staint-James, 2003; Dubnov et al., 2003; Madsen and Widmer, 2005; Meredith, 2006). The work of Conklin (2002) is applicable to the discovery of patterns in a corpus of musical pieces. Other approaches are designed to analyze a single

piece. Two approaches perform an exhaustive exploration of the pattern space according to some well-defined criteria (Conklin, 2002; Meredith, 2006). For example, the search could focus on frequent patterns, long patterns, patterns with a limited set of features. Other approaches will perform a heuristics search, not exploring the whole pattern space but rather aiming at quickly discovering some reasonably interesting pattern according to some criteria such as statistical significance or models of music perception or cognition.

3.2.1 Vertical approaches

Most vertical approaches (Meudic and Staint-James, 2003; Dubnov et al., 2003; Hanna and Ferraro, 2007; Pickens and Crawford, 2002; Doraisamy and Rüger, 2003; Rohrmeier and Cross, 2008) define a pattern as a musical passage encoded at the same abstraction level as the source. One exception is Rohrmeier and Cross (2008) who propose some abstraction by considering sets of pitch classes (ignoring the octave of the notes). The other exception is the work of Conklin (2002), in which a pattern is a sequence of layered components described by any combination of abstract features (as long as each component uses the same combination). Each feature is defined through a function, called *viewpoint*, which takes the preceding musical context as argument and outputs a value. In addition, a viewpoint can be elegantly defined as a composition of multiple viewpoints through a set of viewpoint constructors. By contrast, one defines features in Humdrum (Huron, 1999) either by manually embedding the definition of the feature in the pattern (assuming it can be defined using regular expressions) or by using one of the many command-line tools provided in the toolkit. There is, however, no way to combine feature definitions.

Vertical features are supported by Fitsioris and Conklin (2008), Conklin (2002), and Huron (1999), but absent from other approaches. There is no way, however, for a feature to refer to a voice in Humdrum. As a consequence, if one wants to search, for example, for a parallel fifth pattern, one first needs to extract every possible two-voice combination in which the pattern might be present. Also, if one wants to search for a pattern where voices can be permuted, one first needs to extract the voices from the data and then compute the possible permutations. Although this does not influence the approach precision, it does make it less intelligible. A similar problem occurs in the approach of Conklin (2002), where voice permutations need to be encoded as a set of patterns, each encoding a particular permutation.

Only three vertical approaches (Conklin, 2002; Meudic and Staint-James, 2003;

Rohrmeier and Cross, 2008) support data mining for polyphonic music. Rohrmeier and Cross (2008) develop a pattern discovery framework specialized for the extraction of chord sequences. In the work of Meudic and Staint-James (2003), the similarity of two sequences is computed according to melodic intervals and durations. Pattern discovery is done as follows. First, small sequences are extracted from the piece. The small sequences are put in a matrix where cells represent the similarity between two sequences. Both axes of the matrix are ordered chronologically. The pattern space is explored by joining contiguous cells that are similar, hence creating longer patterns. One problem with this approach, however, is that the content of the small sequences used to create the matrix are not revisited. Consequently, the discovered pattern could exhibit very different musical relations across the boundary where two small sequences are joined. Furthermore, as the length of the small sequences is fixed, only patterns of certain lengths are visited. By contrast, Conklin (2002) uses a suffix tree data structure to enumerate all recurrent sequences. A statistical measure of significance is used to rank the patterns. A pattern is significant if it occurs more often than expected in a random corpus. The approach of Dubnov et al. (2003) uses a similar technique. However, discovered patterns are never explicitly extracted from the suffix tree. Rather, the tree is used as a statistical modeling of the source and then sampled to generate new musical compositions.

3.2.2 Graph approaches

Two approaches use graph-based representations of source and patterns (Madsen and Widmer, 2005; Szeto and Wong, 2006). Both are designed with musical analysis in mind, one focusing on pattern discovery (Madsen and Widmer, 2005) and the other on pattern matching (Szeto and Wong, 2006).

In Madsen and Widmer (2005), the source and pattern instances have the same representation. Nodes represent notes and edges represent the **st** and **m** temporal relations. The **sw** temporal relation is not used but could be added to the approach by adding a new kind of edge. Voicing is indicated by removing the **m** edges that do not occur within a voice. In the work of Szeto and Wong (2006), the nodes represent the pitch class of notes (ignoring octave). Edges represent the **m** and **ov** temporal relations. A pattern is defined as a restricted form of graph, where **m** edges must form “monophonic” voices and **ov** edges occur only between events in separate voices. An efficient pattern matching algorithm is given for that restriction. Madsen and Widmer (2005) also define a pattern as a graph, which must be included

in the set of instances. The instances must be similar to one another, according to a comparison of the sequences of notes found in the graph. The comparison is made with respect to user defined features directly inspired by the work of Conklin (2002). The comparison of Madsen and Widmer (2005) does not support, however, the definition of vertical relations. Patterns are discovered using a genetic search algorithm that maintains a set of candidate patterns and grows them by, for example, adding nodes or joining existing patterns.

3.2.3 Geometric approaches

The work of Meredith (2006) is an example of a geometric approach. The source is represented as a set of points in a multidimensional space. Dimensions correspond to concrete musical features such as onset, pitch and duration. Romming and Selfridge-Field (2007); Typke et al. (2004); Dovey (2001); Clausen et al. (2000) use a similar representation. Lubiw and Tanur (2004); Ukkonen et al. (2003) use an equivalent representation: duration is not considered, but events are represented as lines instead of points.

In the approach of Meredith (2006), a pattern is represented implicitly as the set of instances. Every instance is a subset of the points of the source. Other geometric approaches also use one instance to represent the pattern. In the approach of Romming and Selfridge-Field (2007), for example, a pattern is simply a set of points. Geometric approaches use the notion of translation to relate pattern instances. A translation along the onset dimension represents the fact that a repetition occurs after the first occurrence of the repeated material. A translation along the pitch dimension represents the fact that a repetition may be transposed with respect to the original occurrence. Using a logarithmic scale for durations, Romming and Selfridge-Field (2007) use translation to make sure pattern instances preserve duration ratios between consecutive events. Translation, however, cannot capture every relevant vertical relations, e.g. important relations such as consonance and dissonance are not captured. In addition to translation, some approaches use approximate matching. This is done with a Hausdorff distance by Romming and Selfridge-Field (2007) and using a weighted measure of distance by Lubiw and Tanur (2004) and Typke et al. (2004). Finally, the work of Dovey (2001) supports user-defined relations when computing the similarity of two points.

One characteristic of geometric approaches is that unvoiced polyphonic data can be searched. Voices, if present, can simply be represented as a dimension. Another

interesting characteristic is that it does not restrict patterns to a set of contiguous events. As argued by Meredith (2006), this can capture some forms of melodic elaboration. Not all forms are captured, however, as instances need to match in the onset dimension. The simple variation of modifying durations, for example, will not be captured.

3.2.4 Sequential approaches

Sequential approaches are centered around the matching and discovery of monophonic patterns in a polyphonic texture. Lemström and Tarhio (2000), for example, develop a series of string matching algorithms tailored for quickly retrieving melodies in a source of homophonic texture. The pattern is a sequence of pitches and the source material is a sequence of sets of pitches. The authors apply the technique to polyphonic material by sampling concurrent pitches of a piece at every new onset. Instances of a pattern all share the same sequence of melodic intervals, what Lemström and Tarhio (2000) call *transposition invariance*. The approach, however, does not support voices. Both Utgoff and Kirlin (2006) and Pardo and Sanghi (2005) use the same definition of pattern. The source, however, is not sampled to create layers. Rather, the m temporal relations of the pattern must match existing m relations in the source. Utgoff and Kirlin (2006) use an approximate definition of this relation.

If one searches for a melody, a sequential approach would return every instance of that melody, including those that move across voice boundaries. Moreover, by sampling the source at every onset, Lemström and Tarhio (2000) create notes that do not exist in the source. Consequently, the technique might report some melodic matches that simply do not exist in the source. The main advantage of the approach of Lemström and Tarhio (2000) is its efficiency. The algorithm is an adaptation of *bit-parallel* algorithms for string matching and runs in linear time with respect to the size of the music material to be searched.

Of the three sequential approaches, only Utgoff and Kirlin (2006) support pattern discovery. Patterns are represented directly as a set of instances. The pattern space is explored by clustering existing instances and removing the least relevant clusters before growing the instances of each cluster. The similarity measure used to form the cluster is a heuristic referring to melodic intervals and durations. Finally, the resulting patterns are ranked according to a heuristic interest measure that penalizes local jumps in the pitch and duration dimensions.

3.3 Summary

This chapter reviewed existing computational approaches to the representation and retrieval of polyphonic patterns in polyphonic music. The notions of source and pattern introduced in Chapter 2 can vary depending on the motivations and underlying principles of an approach. Three approaches correspond well to the aims and requirements established in the previous chapter: relational patterns, Humdrum and Structured Polyphonic Patterns. The latter is formally defined in the following chapter and some practical applications are illustrated in Chapter 5. The three approaches are further compared in Chapter 6, where their expressiveness in terms of temporal relations is examined.

Chapter 4

Structured Polyphonic Patterns

This chapter formally introduces the *SPP* pattern matching method. It explains how musical sources are encoded in the method, gives a formal definition of the *SPP* language and explains how queries are executed. Patterns are built by joining basic components either vertically as layers, horizontally as sequences, or as an unrestricted mixture of both. In addition, an onset modification operator enables the representation of layers containing asynchronous components. In *SPP*, a component generalizes over musical events by representing many possible “concrete” events. The structure of a pattern provides further abstraction by representing temporal relations between events. This chapter gives a formal definition of what it means for an *SPP* pattern to be instantiated by a collection of events. We explain how user-defined features are integrated in the query mechanism. Finally, we introduce an efficient matching algorithm.

4.1 Source

An *SPP* source is an encoding of a piece of music. It simply contains all the musical events of the piece.

Definition 1 A *source* s is a set of *events* e :

$$s ::= \{e_1, e_2, \dots\}$$

An *SPP* event encodes a single musical event, for example a note or a rest. The event must belong to a voice. In addition, an event contains features that represent its properties, such as its **onset** and **offset**.

Definition 2 A *event* e is a set of *event features* f , labelled with a *voice name* n :

$$e ::= \{f_1, f_2, \dots\}_n$$

There are two kinds of event features. The first kind encodes a property of the event itself or a musical relation with an event in the same voice. The second kind encodes a musical relation that the event forms with some event in another voice.

Definition 3 An *event feature* f is defined according to the following syntax, where τ is a *feature name*, v is a *feature value* and n is an optional *voice name*:

$$f ::= \tau : v \quad | \quad \tau(n) : v$$

Consider the following example of an *SPP* event:

$$(E1) \quad \{\text{pitch} : F\sharp 4, \text{interval}(\text{bass}) : M3\}_{\text{soprano}}$$

It encodes a musical note that belongs to the **soprano** voice. The note has a pitch of $F\sharp 4$ and forms a major third with some note in the **bass** voice.

Musical properties such as **interval** are typically the most relevant features when defining a query. However, an *SPP* source does not necessarily possess such features. For an *SPP* source to be *well-formed*, two features are required to be present in every event: the **onset** of an event (the time point at which it starts) and its **offset** (the time point at which it stops).

Figure 4.1 shows a well-formed encoding of bar 21 of Bach chorale BWV 304.

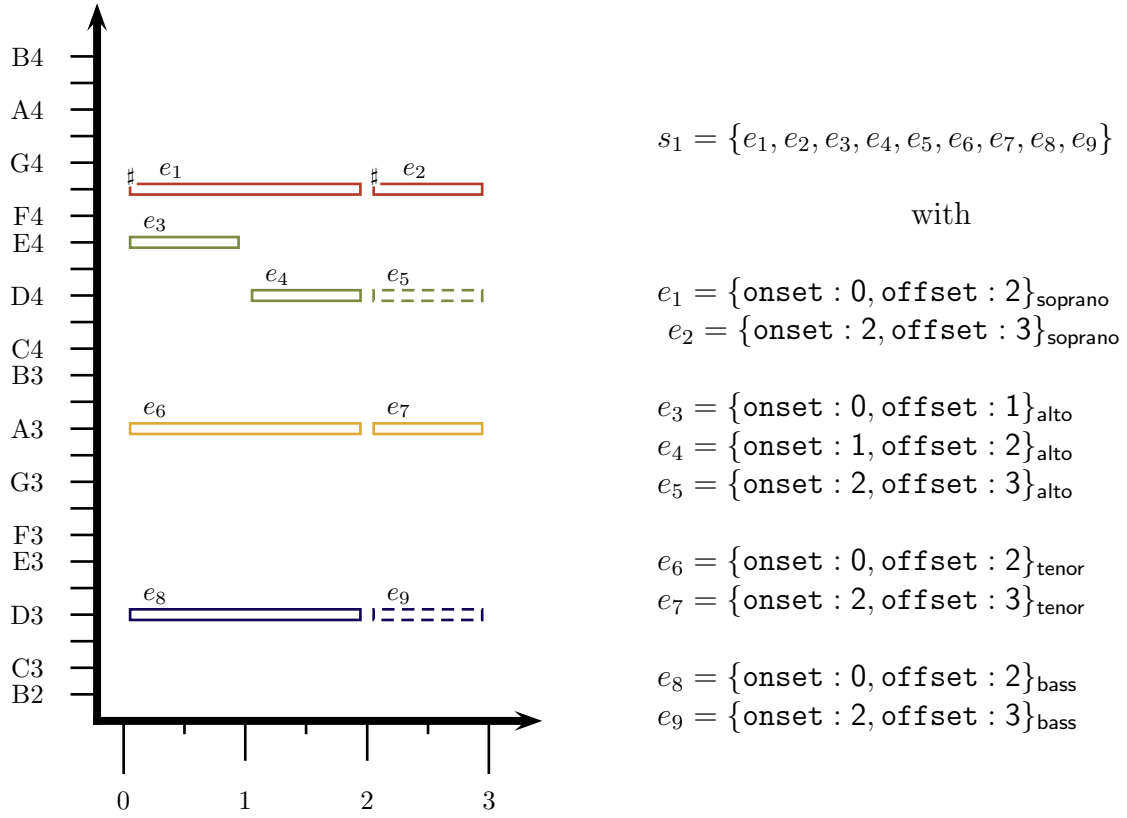


Figure 4.1: *SPP* encoding of BWV 304, bar 21

4.2 Components

As a basic abstraction mechanism, the $\mathcal{SP}\mathcal{P}$ language provides a way to describe many different events through a syntactic construct called *component*.

Definition 4 A *component* κ is a set of *component features* $\widehat{f}$, labelled with either a *voice name* n or a *voice variable* γ :

$$\begin{aligned} \kappa \quad ::= \quad & \{\widehat{f}_1, \widehat{f}_2, \dots\}_n \\ & | \quad \{\widehat{f}_1, \widehat{f}_2, \dots\}_\gamma \end{aligned}$$

If labelled with a voice variable, a component can describe events that belong to different voices. Similarly, a component feature act as a descriptor for many different event features.

Definition 5 A *component feature* $\widehat{f}$ is defined according to the following syntax, where τ is a *feature name*, v is a *feature value*, α is a *value variable*, n is a *voice name* and γ is a *voice variable*:

$$\begin{aligned} \widehat{f} \quad ::= \quad & \tau : v \\ & | \quad \tau : \alpha \\ & | \quad \tau(n) : v \\ & | \quad \tau(\gamma) : v \\ & | \quad \tau(n) : \alpha \\ & | \quad \tau(\gamma) : \alpha \end{aligned}$$

Again, using variables instead of values or voice names provides a way to represent many different events. For example, consider the following component:

$$(E2) \quad \{\text{pitch} : P, \text{interval}(y) : I\}_x$$

It describes the case of *E1*, where an event forms a major third with some event in the **bass** voice. Similarly, it can describe nearly all events of Figure 4.1, assuming they were first augmented with the **pitch** and **interval** features. Rest events are exceptions, as these do not possess a **pitch** or form any **interval**. When a particular

event is described by a component, we say that the component matches that event, or equivalently that the event instantiates the component.

4.3 The $\mathcal{SPP}$ language

The second abstraction mechanism that the $\mathcal{SPP}$ language provides is the ability to specify temporal relations. The four temporal relations presented in Chapter 2 are supported:

- $\mathbf{m}(a, b)$: *a meets b* (*a* ends as *b* starts)
- $\mathbf{st}(a, b)$: *a and b start together*
- $\mathbf{sw}(a, b)$: *a starts while b is unfolding*
- $\mathbf{ov}(a, b)$: *a and b overlap* (at some point in time, both *a* and *b* are unfolding)

In $\mathcal{SPP}$, these temporal relations are specified structurally, via two basic operators: the sequencing operator “;” and the layering operator “==”. In addition, an onset modification operator “-” can be applied directly to a component, altering the interpretation of the corresponding layer.

Definition 6 An $\mathcal{SPP}$ pattern ϕ is defined according to the following syntax:

$$\begin{array}{lcl} \phi & ::= & \kappa \\ & | & -\kappa \\ & | & \phi ; \phi \\ & | & \frac{\phi}{\phi} \end{array}$$

The “;” operator is used to build sequences of components, thus enforcing the $\mathbf{m}$ temporal relation. The “==” operator is used to build layers of simultaneous events, thus enforcing the $\mathbf{st}$ relation. In addition, the “-” operator can be used in conjunction with the “==” operator to represent events that overlap in time without being simultaneous, thus enforcing the $\mathbf{sw}$ and $\mathbf{ov}$ temporal relations. Consider, for example, the $\mathcal{SPP}$ pattern below.

$$(E3) \quad \frac{\{\text{duration} : 1\}_x}{-\{\text{onset} : 0\}_y ; \{\text{pitch} : X\}_y} ; \quad \{\text{duration} : 1\}_x$$

By virtue of belonging to a sequence, components $\{\text{onset} : 0\}_y$ and $\{\text{pitch} : X\}_y$ at the bottom of the layer are interpreted as following one another, hence specifying an **m** temporal relation. By virtue of being layered, components $\{\text{duration} : 1\}_x$ and $\{\text{onset} : 0\}_y$ on the left of the pattern would normally be interpreted as specifying an **st** temporal relation. However, as the “ $-$ ” operator is used, the onset of the modified component is conceptually extended backward in time, resulting in the specification of an **sw** temporal relation: the component at the bottom of the layer is interpreted as starting earlier and hence is already unfolding when the component at the top of the layer starts.

Notice how example *E3* above contains two identical components of the form $\{\text{duration} : 1\}_x$. When isolated, these would match the same events. In the context of a pattern, however, they are meant to match different events: both with the same duration, but one following the other temporally. To make this distinction effective, we need to make sure that every component in an *SPP* pattern can be uniquely identified. This is done through an indexing. The indexing is not part of *SPP* syntax per se, but is necessary to describe and manipulate *SPP* patterns in an unambiguous way.

Definition 7 Given an *SPP* pattern ϕ , we assume an *indexing* that assigns a unique *index* ϵ to every component κ in ϕ . We write $\bar{\epsilon}$ to refer to the component indexed by ϵ .

For example, the pattern of *E3* can be represented with indices as follows:

$$(E4) \quad \frac{a}{-b ; c} ; \quad d$$

We have $\bar{a} = \bar{d} = \{\text{duration} : 1\}_x$, $\bar{b} = \{\text{onset} : 0\}_y$, and $\bar{c} = \{\text{pitch} : X\}_y$. Henceforth, we freely refer to *SPP* patterns as having either indices or components, depending on context.

The indexing is useful when discussing the structure of *SPP* patterns. Consider for example the patterns of Table 4.1. The interpretation refers to how the structure

Type of pattern	Example ϕ	Interpretation of ϕ
Sequence of components	$a ; b ; c$	a starts ϕ c ends ϕ
Layer of components	$\begin{array}{c} \underline{\underline{a}} \\ \underline{\underline{b}} \\ \underline{\underline{c}} \end{array}$	a, b and c start ϕ a, b and c end ϕ
Sequence of layers	$\frac{a}{b} ; \frac{c}{d}$	a, b start ϕ c, d end ϕ a and c are aligned b and d are aligned
Layered sequences	$\begin{array}{c} \underline{\underline{a ; b}} \\ \underline{\underline{c ; d}} \\ \underline{\underline{e ; f ; g}} \end{array}$	a, c, e start ϕ b, d, g end ϕ
Structures	$\frac{a}{b} ; \frac{c ; d}{e ; \frac{f}{g}}$	a, b start ϕ d, f, g end ϕ a and c are aligned b and e are aligned e and f are aligned

 Table 4.1: Different types of $\mathcal{SPP}$ patterns and their basic interpretations

of a pattern indicates the temporal relations to be enforced. When components start a pattern, or a subpattern, they are conceptually simultaneous (via the **st** temporal relation) or overlapping (via the **sw** or **ov** temporal relations). Two components are aligned if one ends a subpattern on the left of a “;” operator and the other starts a subpattern on the right of the same operator. In addition, aligned components have a similar location in their respective layers (a component at the top of one layer is aligned with a component at the top of the other layer). Aligned components are interpreted as being directly one after the other in sequence (via the **m** temporal relation).

Notice that Table 4.1 contains several pattern where the binary operators “;” and “=” are seemingly used to relate more than two terms. This relies on an implicit bracketing of the terms of a pattern. The interpretation of the $\mathcal{SPP}$ patterns presented in this dissertation is always clear from the vertical and horizontal place-

ment of the terms. To improve readability, we hence present the patterns without brackets. For completeness, the definition below indicates the implicit position of brackets in a $\mathcal{SP}\mathcal{P}$ pattern.

Definition 8 When interpreting $\mathcal{SP}\mathcal{P}$ patterns, the following assumptions are made:

1. The layering operator “ $\equiv$ ” has precedence over the sequencing operator “ $;$ ”
2. The layering operator “ $\equiv$ ” is top-associative
3. The sequencing operator “ $;$ ” is left-associative

The first rule of Definition 8 simply states that when sequences and layers are mixed, the brackets are implicitly forming around the layers:

$$(E5) \quad \frac{a}{\equiv} ; \frac{c}{\equiv} \quad \text{is implicitly bracketed as} \quad \left(\frac{a}{\equiv} \right) ; \left(\frac{c}{\equiv} \right)$$

The second rule of Definition 8 states that the terms at the top of a layer are grouped first:

$$(E6) \quad \frac{\frac{a}{\equiv}}{\equiv} \quad \text{is implicitly bracketed as} \quad \frac{\left(\frac{a}{\equiv} \right)}{\equiv}$$

Similarly, the last rule of Definition 8 states that the terms at the beginning of sequence are grouped first:

$$(E7) \quad a ; b ; c \quad \text{is implicitly bracketed as} \quad (a ; b) ; c$$

It is also important to notice how the interpretation of a pattern allows for some commutativity. For example, the following two patterns have the same interpretation:

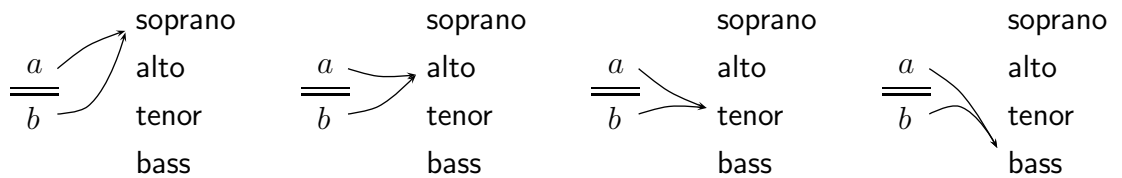
$$\frac{a}{\equiv} ; \frac{c}{\equiv} \quad \text{is equivalent to} \quad \frac{b}{\equiv} ; \frac{d ; e}{\equiv}$$

It is not relevant, for example, that components a and c appear on top of the layers in the first pattern and on the bottom in the second pattern. In both cases, the component a is aligned with the component c . It is not true however that the layering operator “ $\equiv$ ” is always commutative. For example, if the first layer above was to be permuted and not the second, the interpretation of the pattern would change by virtue of the component a now being aligned with d in the second pattern:

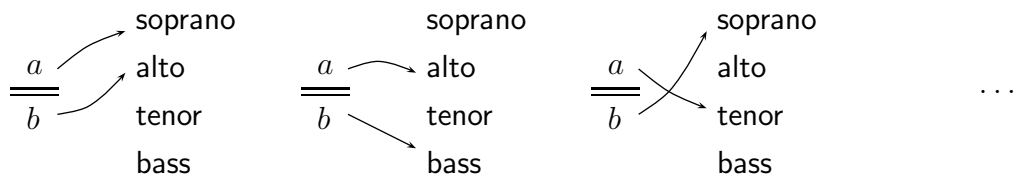
$$\frac{a}{b} \equiv \frac{c}{d; e} \quad \text{is *not* equivalent to} \quad \frac{b}{a} \equiv \frac{c}{d; e}$$

Finally, note that the interpretation of a pattern does not indicate voicing. In particular, a component appearing at the top of a layer can match any voice and does not necessarily match a voice that would be considered “higher” than the voice matched by a component appearing at the bottom of the layer.

To illustrate this, imagine a layer with two components and a source with four parts (ordered from highest to lowest): **soprano**, **alto**, **tenor** and **bass**. When the components have the same voice label, they will match events in the same part:



When the components have different voice labels, they will match events in any two distinct parts:



By abstracting away from the ordering of the components of a layer, the *SPP* method automates the exploration of voice combinations.

The following section builds on the intuitive interpretation of *SPP* patterns discussed so far and gives a formal definition of how an *SPP* pattern is instantiated by a collection of events.

4.4 Interpreting $\mathcal{SPP}$ patterns

Conceptually, an instance of an $\mathcal{SPP}$ pattern ϕ is determined by a mapping of the components of ϕ to the events of a source, along with a mapping of the variables of ϕ to appropriate values. The former shows “where” in the source the instance occurs and the latter shows “how” a particular component relates to the event it matches. As explained in Section 4.3, the former can only be defined if the components are indexed. An instance hence consists of a set of pairs I that relate indices and events, along with a variable assignment. For example, the following is an instance of the pattern $E3$ in the source of Figure 4.1:

$$(E8) \quad \begin{aligned} I &= \{(a, e_4), (b, e_6), (c, e_7), (d, e_5)\} \\ \theta &= \{(\mathbf{x}, \mathbf{A3}), (\mathbf{x}, \mathbf{alto})(\mathbf{y}, \mathbf{tenor})\} \end{aligned}$$

Formally, a variable assignment is defined as follows.

Definition 9 A *variable assignment* θ for ϕ is a set of pairs acting as a mapping of value variables and voice variables to feature values and voice names, such that:

- Every variable of ϕ appears exactly once in θ
- No other variables appear in θ
- θ always maps a value variable to a value
- θ always maps a voice variable to a voice name
- The mapping of voice variables to voice names is injective: no two voice variables are mapped to the same voice name. This enforces the intuition that when two components are labelled with distinct voice variables, they belong to distinct voices.

An instance is defined as follows.

Definition 10 Given a variable assignment θ for ϕ , a θ -*instance* I of ϕ is a set of pairs acting as a mapping of component indices to events:

- Every component index ϵ of ϕ appears exactly once in I
- No other component index appears in I
- The mapping of indices to events is injective: no two indices are mapped to the same event. This enforces the intuition that a pattern with n components will match n events.

In addition, the following conditions must hold:

- **If ϕ is of the form ϵ or $-\epsilon$, then:**

$$I = \{(\epsilon, e)\}$$

e is a θ -instance of $\bar{\epsilon}$

(Definition 11)

- **If ϕ is of the form $\phi_1 ; \phi_2$, then:**

$$I = I_1 \cup I_2$$

I_1 is a valid θ -instance of ϕ_1

I_2 is a valid θ -instance of ϕ_2

and:

For every component ϵ_1 *ending* ϕ_1

(Definition 12)

And every component ϵ_2 *starting* ϕ_2

(Definition 13)

If ϵ_1 and ϵ_2 are *aligned*

(Definition 14)

then I *enforces* $\mathbf{m}(\epsilon_1, \epsilon_2)$

(Definition 15)

- **If ϕ is of the form $\frac{\phi_1}{\phi_2}$, then:**

$$I = I_1 \cup I_2$$

I_1 is a valid θ -instance of ϕ_1

I_2 is a valid θ -instance of ϕ_2

and:

For all components ϵ_1, ϵ_2 *starting* ϕ_1, ϕ_2

(Definition 13)

For pairs of the form ϵ_1, ϵ_2 , I *enforces* $\mathbf{st}(\epsilon_1, \epsilon_2)$

(Definition 15)

For pairs of the form $\epsilon_1, -\epsilon_2$, I *enforces* $\mathbf{sw}(\epsilon_1, \epsilon_2)$

(Definition 15)

For pairs of the form $-\epsilon_1, \epsilon_2$, I *enforces* $\mathbf{sw}(\epsilon_2, \epsilon_1)$

(Definition 15)

For pairs of the form $-\epsilon_1, -\epsilon_2$, I *enforces* $\mathbf{ov}(\epsilon_1, \epsilon_2)$

(Definition 15)

Consider the pattern $E3$ and the instance of example $E8$. Components a and b start the layer on the left-hand side of the “;” operator. As a “—” operator is used,

the instance must enforce $\mathbf{sw}(a, b)$. According to the instance, a is mapped to e_4 and b is mapped with e_6 . We can verify on Figure 4.1 that when e_4 starts, the event e_6 is already unfolding. Similarly, we can verify that the semantics is respected for components a and d on both side of the “;” operator. As a ends the left-hand side of the operator and b starts the right-hand side (and they are aligned), the instance must enforce the $\mathbf{m}$ temporal relation. Component d is mapped to event e_5 . Again, we can verify on Figure 4.1 that event e_4 ends as event e_5 starts. The remainder of the semantics can also be easily verified. Note that the fact that events e_4 and e_5 belong to the same voice is enforced by the variable assignment, not by the structure of the pattern.

The remainder of this section contains formal definitions that are referenced by Definition 10. First, we define what it means for an event to instantiate a component.

Definition 11 Given a θ -instance I , a component $\bar{e}$ and an event e , we say that e *θ -instantiates* $\bar{e}$ if:

- The voice label of $\bar{e}$ and e match, that is:
 - If the voice label of $\bar{e}$ is n , then the voice label of e is n
 - If the voice label of $\bar{e}$ is γ , then the voice label of e is n , and $\theta(\gamma) = n$
- For every component feature $\hat{f} \in \bar{e}$, there exists an event feature $f \in e$ such that:
 - If $\hat{f}$ is $\tau : v$, then f is $\tau : v$
 - If $\hat{f}$ is $\tau : \alpha$, then f is $\tau : v$, and $\theta(\alpha) = v$
 - If $\hat{f}$ is $\tau(n) : v$, then f is $\tau(n) : v$
 - If $\hat{f}$ is $\tau(\gamma) : v$, then f is $\tau(n) : v$, and $\theta(\gamma) = n$
 - If $\hat{f}$ is $\tau(n) : \alpha$, then f is $\tau(n) : v$, and $\theta(\alpha) = v$
 - If $\hat{f}$ is $\tau(\gamma) : \alpha$, then f is $\tau(n) : v$, and $\theta(\alpha) = v$, $\theta(\gamma) = n$

The following three definitions concern the conceptual interpretation of $\mathcal{SPP}$ patterns that was discussed at the end of Section 4.3.

Definition 12 Given an $\mathcal{SPP}$ pattern ϕ , we say the index ϵ *ends* ϕ exactly in the following cases:

- ϕ is of the form ϵ or $-\epsilon$
- ϕ is of the form $\phi_1 ; \phi_2$ and ϵ ends ϕ_2
- ϕ is of the form $\frac{\phi_1}{\phi_2}$ and ϵ ends ϕ_1 or ϵ ends ϕ_2

Definition 13 Given an $\mathcal{SPP}$ pattern ϕ , we say the index ϵ *starts* ϕ exactly in the following cases:

- ϕ is of the form ϵ or $-\epsilon$
- ϕ is of the form $\phi_1 ; \phi_2$ and ϵ starts ϕ_1
- ϕ is of the form $\frac{\phi_1}{\phi_2}$ and ϵ starts ϕ_1 or ϵ starts ϕ_2

Definition 14 Given an $\mathcal{SPP}$ pattern $\phi_1 ; \phi_2$, a index ϵ_1 ending ϕ_1 and a index ϵ_2 starting ϕ_2 , we say that ϵ_1, ϵ_2 are *aligned* exactly in the following cases:

- ϕ_1 is of the form ϵ_1 or $-\epsilon_1$, and:
 - ϕ_2 is of the form ϵ_1 or $-\epsilon_1$
 - ϕ_2 is of the form $\phi'_2 ; \phi''_2$ and ϵ_1, ϵ_2 are aligned in $\epsilon_1 ; \phi'_2$
 - ϕ_2 is of the form $\frac{\phi'_2}{\phi''_2}$ and ϵ_1, ϵ_2 are aligned in $\epsilon_1 ; \phi'_2$
- ϕ_1 is of the form $\phi'_1 ; \phi''_1$, and ϵ_1, ϵ_2 are aligned in $\phi'_1 ; \phi_2$
- ϕ_1 is of the form $\frac{\phi'_1}{\phi''_1}$, and:
 - ϕ_2 is of the form ϵ_2 or $-\epsilon_2$ and ϵ_1, ϵ_2 are aligned in $\phi'_1 ; \epsilon_2$
 - ϕ_2 is of the form $\phi'_2 ; \phi''_2$ and ϵ_1, ϵ_2 are aligned in $\phi'_1 ; \phi'_2$
 - ϕ_2 is of the form $\frac{\phi'_2}{\phi''_2}$ and ϵ_1, ϵ_2 are aligned in $\phi'_1 ; \phi'_2$ or $\phi''_1 ; \phi'_2$

Finally, the precise definition of the four temporal relations that an $\mathcal{SPP}$ pattern can enforce is as follows. We use a “dot” notation to refer to the value of event features. For example, we write $e.\text{onset}$ to refer to the onset of some event e .

Definition 15 Given an instance I containing the pairs (ϵ_1, e_1) and (ϵ_2, e_2) , we say that:

- I enforces $\mathbf{m}(\epsilon_1, \epsilon_2)$ if:
 - $e_1.\mathbf{offset} = e_2.\mathbf{onset}$
- I enforces $\mathbf{st}(\epsilon_1, \epsilon_2)$ if:
 - $e_1.\mathbf{onset} = e_2.\mathbf{onset}$
- I enforces $\mathbf{sw}(\epsilon_1, \epsilon_2)$ if:
 - $e_1.\mathbf{onset} > e_2.\mathbf{onset}$, and
 - $e_1.\mathbf{onset} \leq e_2.\mathbf{offset}$
- I enforces $\mathbf{ov}(\epsilon_1, \epsilon_2)$ if there exists a point in time t such that:
 - $e_1.\mathbf{onset} \leq t \leq e_1.\mathbf{offset}$, and
 - $e_2.\mathbf{onset} \leq t \leq e_2.\mathbf{offset}$

4.5 Feature definition rules

As explained in Section 4.4, the $\mathcal{SPP}$ language offers two abstraction mechanisms: the description of collections of events through components and the specification of temporal relations between events through syntactic constructions. A pattern ϕ , however, can only refer to the information that is already present in a source. Typically, the basic information of a musical source, such as pitch and duration, will not suffice to express meaningful queries.

To deal with this issue, the $\mathcal{SPP}$ method provides a systematic way to expand the information of a source, in particular by adding features that encode musical concepts such as scale degrees and relations between notes such as melodic and harmonic intervals.

Definition 16 A *feature definition rule* consists of two clauses, the **where** clause and the **add** clause:

$$\begin{array}{ll} \mathbf{where} & \mathbf{add} \\ \phi[\epsilon^*] & \widehat{f}[\Omega] \end{array}$$

- The **where** clause contains an $\mathcal{SPP}$ pattern ϕ in which exactly one component ϵ^* is distinguished.
- The **add** clause contains a component feature $\hat{f}$ in which the value is replaced by a function Ω defined over the variables of ϕ and returning a value when evaluated in the context of a variable assignment θ .

A feature definition rule applies to every θ -instance I of its **where** clause ϕ . For every such instance, an event feature is built from the **add** clause as follows. First, its voice variable (if any) is replaced by the appropriate voice name, according to θ . Then, the value of the feature is replaced by the result of evaluating the function Ω in the context of θ . The resulting event feature is added to the event e that I associates with the distinguished component ϵ^* . For example, the following definition rule would add the **interval** feature appearing in example *E1*.

$$(E9) \quad \begin{array}{cc} \text{where} & \text{add} \\ \frac{\{\text{pitch} : P1\}_x^*}{\{\text{pitch} : P2\}_y} & \text{interval}(y) : \text{diatonic_interval}(P1, P2) \end{array}$$

The pattern of the **where** clause will match every case where two notes start together. For every such case, the pitches of the notes are recorded in the value variables $P1$ and $P2$ and their respective voices are recorded in the voice variables x and y . This information is used to transform the **add** clause into a valid event feature by assigning y to a voice name and evaluating **diatonic_interval**. The feature is then added to the note that corresponds to the distinguished component $\{\text{pitch} : P1\}_x^*$.

4.6 Querying musical sources

Using the $\mathcal{SPP}$ method to build a musicological query typically implies the following steps:

1. Encode the musical pieces to query as well-formed $\mathcal{SPP}$ sources
2. Write down feature definition rules to capture musical concepts on which the query depends

3. Apply every rule to the sources, hence saturating it with the relevant features (alternatively, it is also possible to compute the rules in an on-the-fly manner, where the features are added to a source while the pattern is being matched)
4. Write down an $\mathcal{SPP}$ pattern that captures the query. The pattern typically refers to the features defined above
5. Compute the instances of the pattern in the saturated sources
6. Inspect the resulting instances
7. Optionally, the user can perform some quantitative analysis of the instances

Notice that Step 1 assumes that more than one source is of interest as, typically, a query is applied to a *corpus*: a collection of musical pieces. Applying a query to a corpus is simply done by applying the query separately to every piece in the corpus. This is done by steps 3 and 5, which can be performed efficiently using the same algorithm, described in Section 4.7 and Appendix D. Chapter 5 illustrates how to build and apply musicological queries in $\mathcal{SPP}$, including some examples of simple quantitative analysis of the results.

4.7 Matching algorithm

A naive implementation of $\mathcal{SPP}$ would yield an algorithm operating in quadratic time: joining two subpatterns into a sequence or layer would require the exploration of the Cartesian product of their instances. This section introduces the $\mathcal{SPP}$ -join algorithm, operating in log-linear time. It takes as input a well-formed $\mathcal{SPP}$ source s and an $\mathcal{SPP}$ pattern ϕ and returns as output the collection of all θ -instances of ϕ in s . Following Definition 10, the collection of instances can be built by recursively analyzing ϕ . To compute the instances of a pattern ϕ with subpatterns ϕ_1 and ϕ_2 , the algorithm proceeds as follows. First the instances of ϕ_1 and ϕ_2 are computed. Then, pairs of instances of ϕ_1 and ϕ_2 are iterated. Every pair is joined to form a candidate instance of ϕ (this simply consists of the union of ϕ_1 and ϕ_2 respective instances). The candidate instance is tested against Definition 10 to determine if it is a valid instance of ϕ . A naive implementation would iterate all pairs of instances, for example by iterating every instance of ϕ_1 and for every such instance, iterating all instances of ϕ_2 . This would clearly yield a quadratic time complexity in terms of the number of instances. The efficient $\mathcal{SPP}$ -join algorithm developed

in Appendix D is based on the simple observation that all valid instances of an *SPP* pattern are contiguous in the time dimension. When joining two collections of instances, the *SPP-join* algorithm directly selects a subset of instance pairs. The subset corresponds to those instances of ϕ_1 and ϕ_2 that may yield a valid instance of ϕ with respect to the temporal relations it enforces. This is achieved by storing instances according to their start time and accessing only the candidate instances that have a relevant start time. Using a balanced tree, this can be done in log-linear time.

4.8 Summary

This chapter presented the *SPP* method in detail. The method satisfies the requirements for polyphonic pattern language stated in Chapter 2. It differs from the existing approaches reviewed in Chapter 3 in many ways. Contrary to Humdrum, for example, it has clear and precise semantics and, contrary to a relational language, it is specialized for music with a syntax that illustrates the structure of the query. In terms of formal expressiveness, the differences between these three approaches are explored further in Chapter 6. The following chapter illustrates the utility of *SPP* as a polyphonic pattern language by developing musicological queries and analyzing the results on four different musical corpora.

Chapter 5

Musicological queries in $\mathcal{SPP}$

This chapter demonstrates the practical applicability of $\mathcal{SPP}$. We show how to develop musicologically meaningful queries and analyze the results obtained when applying the queries to real musical sources. The queries we develop show a wide range of patterns, from basic queries that could be used for information gathering to complex queries that can capture precepts of musical styles, such as patterns of second species counterpoint. The queries are typical of n-part polyphony. For qualitative evaluation of the queries, for example when examining particular instances of a pattern, we focus on a corpus of chorale harmonizations by J.S. Bach. These pieces are largely homophonic, but contain enough examples of polyphonic material for our purposes. For quantitative analysis, we consider three additional corpora: folk songs, Mozart Symphony no.40 and Chopin piano music. Examining several corpora enables us to establish the salience of some patterns: over-represented in one corpus by comparison to the others.

5.1 Overview

In this chapter, we encode four corpora as well-formed $\mathcal{SPP}$ sources, using such additional musical features as **pitch**, **key**, **duration** and **meter**. From these basic features, we derive more meaningful features using the feature definition mechanism introduced in Chapter 4. Figure 5.1 shows this transformation as arrows from basic features (in **bold**) to user-defined features (in *italics*). The latter include, for example, the scale degree of notes and different types of melodic and harmonic intervals.

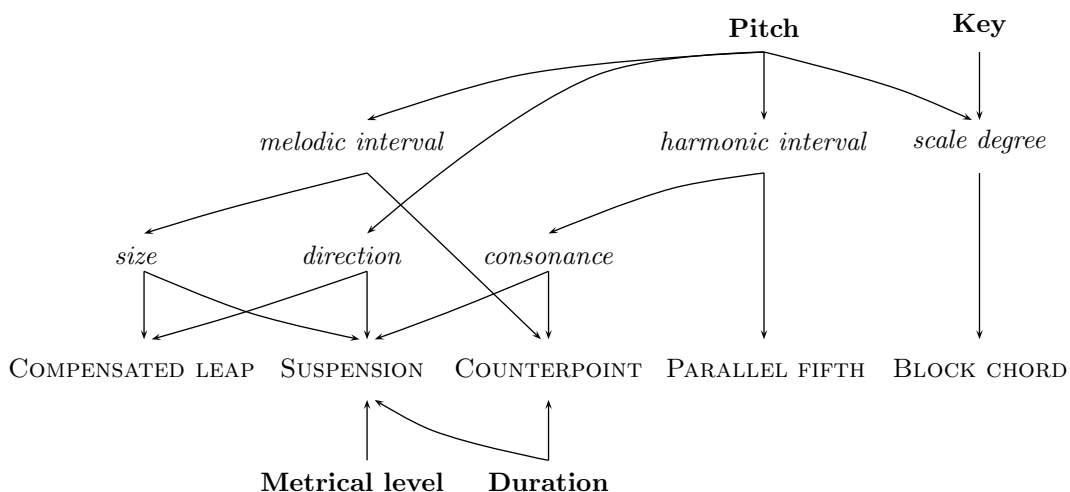


Figure 5.1: Overview of the queries developed in this chapter. Queries are shown in CAPITALS. Features that are already present in the source are shown in **bold**. Features that are added to the source using *SPP* feature definition rules are shown in *italics*

By using such features in basic queries and analyzing the result, we obtain a simple way to compare the different corpora. This is described in Section 5.3.

Figure 5.1 also shows how features are used in queries (in CAPITALS). Arrows show how different queries depend on different features. These queries were selected as they illustrate different kinds of *SPP* patterns: sequences, layers, sequence of layers (with different ways to combine the onset modification operator “—”), and free mixtures of sequences and layers.

For each query, we compare two simple quantities: how many positions in the source are consistent with the *frame* of the query and how many of those positions are actual instances of the query. The frame of a query typically captures the same structure, e.g. in terms of temporal relations, but without musical relations. It gives us an approximation of how probable a particular pattern is in a specific corpus, e.g. how probable it is to encounter an actual suspension when notes that exhibit the structure of a suspension are encountered. We call this the *frame-conditioned probability* of a pattern. It gives us a way to normalize the number of occurrences of a pattern in a corpus, in order to render the comparison with other corpora relevant.

For example, the frame-conditioned probability of the suspension pattern in Bach chorale harmonizations is 18.88%. This means that nearly one fifth of all positions that exhibit the temporal relations of a suspension also exhibit the consonance and dissonance pattern of a suspension. By comparison, this is only 3.03% in Mozart

Symphony no.40 and 2.28% in Chopin piano music. The comparison tells us that the suspension pattern is *salient* in Bach chorales. Salience can refer to how perceptually prominent a pattern is or how frequent it is (Huron, 2001b); or alternatively salience can refer to a measure of relative probability (Conklin, 2008, 2010). This dissertation adopts a simple definition of salience, inspired by (Conklin, 2010) : a pattern is salient in a corpus, with respect to a comparison corpus, when it has a relatively high probability in that corpus and relatively low probability in the comparison corpus.

In Section 5.4, we analyze the frame-conditioned probabilities and salience of the queries of Figure 5.1. In addition, musical examples from Bach chorale harmonizations are examined and variations and extensions of the queries are proposed. The additional queries illustrate how to reuse existing features and patterns when building new queries in a modular way.

To improve readability, some results are deferred to Appendix F, which contains additional figures, in particular renderings of musical excerpts in piano-roll notation.

5.2 Sources

The results presented in this chapter concern four corpora: i) a collection of 185 chorale harmonizations by Johann Sebastian Bach (henceforth called *Bach chorales*), ii) the four movements of Symphony No.40 in G minor, KV. 550, by Wolfgang Amadeus Mozart (henceforth called *Symphony no.40* or *no.40*), iii) a collection of 83 pieces for piano by Fryderyk Chopin (henceforth called *Chopin*) and iv) a collection of folk songs largely composed of the Essen Folksong Collection (Schaffrath, 1995). Table 5.1 presents an overview of the corpora in terms of number of sources, events and parts.

All corpora were retrieved in Humdrum format. For each piece, its key was estimated using the Humdrum key command, which is based on a key finding algorithm by Krumhansl and Kessler (1982). The estimation includes modulations (regions of a piece where the key temporarily changes).

These corpora were selected to illustrate the applicability of *SPP* to different types of sources: monophonic (Folk songs), 4-part polyphony (Bach chorales), orchestral (Symphony no.40), and piano music (Chopin).

Corpus	Sources	Events	% Rests	Parts
Chorales	185	44465	2.73%	4
Symphony no.40	4	31363	28.09%	10
Chopin	83	90497	7.02%	1
Folk songs	8413	456827	5.51%	1

Table 5.1: Overview of the four corpora considered in this dissertation

Temporal relations The number of parts in Table 5.1 indicates that the corpora might differ in texture, for example in how many simultaneous notes a single part can contain. We can get a better idea by executing two simple queries:

$$(P1) \quad \frac{\{\}_x}{\{\}_y}$$

$$(P2) \quad \frac{\{\}_x}{-\{\}_y}$$

The queries *P1* and *P2* will respectively return every instance of the **st** (start together) and **sw** (start while) temporal relations. This reveals that Symphony no.40 and Chopin exhibit simultaneous notes within parts. For example, a note can form up to sixteen **st** relations and thirteen **sw** relations in Symphony no.40, even if there are only ten parts in the source. This is probably due to the fact that, at given times, many parts are playing block chords (through double stops in the violin parts, for example). For Chopin, pieces were encoded as having a single part (i.e. every note of the piano is in the same part). It is therefore expected to find simultaneous notes within that part: up to eight **st** relations and six **sw** can be formed by a note in that corpus. This is different for Bach chorales, where notes are never simultaneous within the same part. To compare the proportion of **st** and **sw** in each corpus, we use the following frame query:

$$(P3) \quad \frac{-\{\}_x}{-\{\}_y}$$

The query *P3* will return every instance of the **ov** temporal relation. This gives us

the frame-conditioned probability of the **st** and **sw** relations, as shown by Table 5.2. The **st** temporal relation is more frequent than the **sw** temporal relation in all three polyphonic corpora we consider.

	Chorales	no.40	Chopin
st	65.83%	60.60%	66.39%
sw	34.17%	39.40%	33.61%

Table 5.2: Frame-conditioned probabilities of the **st** and **sw** temporal relations

Note that some care is required when working with queries such as *P1*, *P2* and *P3*. Queries *P1* and *P3*, for example, will match every **st/ov** relation twice, once as the top component matches the first event involved in the relation and once again as the same component matches the second event. Another issue is how both queries return only instances of relations formed between different parts. This is so because two voice variables are used: **x** and **y**. In order to get the instances that occur within a part, we must use a similar query that uses only one voice variable. These issues and similar issues were properly addressed when producing and analyzing the results of this chapter and are not discussed in the remainder.

Musical features In addition to exhibiting similar proportions of **st/sw** temporal relations, the corpora are very similar in terms of the musical information available. All corpora were originally encoded in the Humdrum format, and retrieved from the site www.kern.humdrum.org (Sapp, 2005). Each piece (or movement) was translated as a collection of events. Each event contains the basic features required for a well-formed *SPP* source: **onset** and **offset**. In addition, the four musical features of Figure 5.3 were extracted from the Humdrum encoding. Figure 5.2 illustrates the encoding scheme on the excerpt of Figure 4.1 (page 60).

Feature name	Description	Examples
pitch	Spelled pitch	F4, G#3, Eb3
key	Tonic and mode	Cmaj, Amin
duration	Duration of an event	2, 1, $\frac{1}{2}$
meter	Metrical position of an event	0 (strong), 3 (weak).

Table 5.3: Basic features of source encodings

To compare the corpora further, we can formulate queries based on the features of Table 5.3. For example, the following query will record every metrical level found in a corpus:

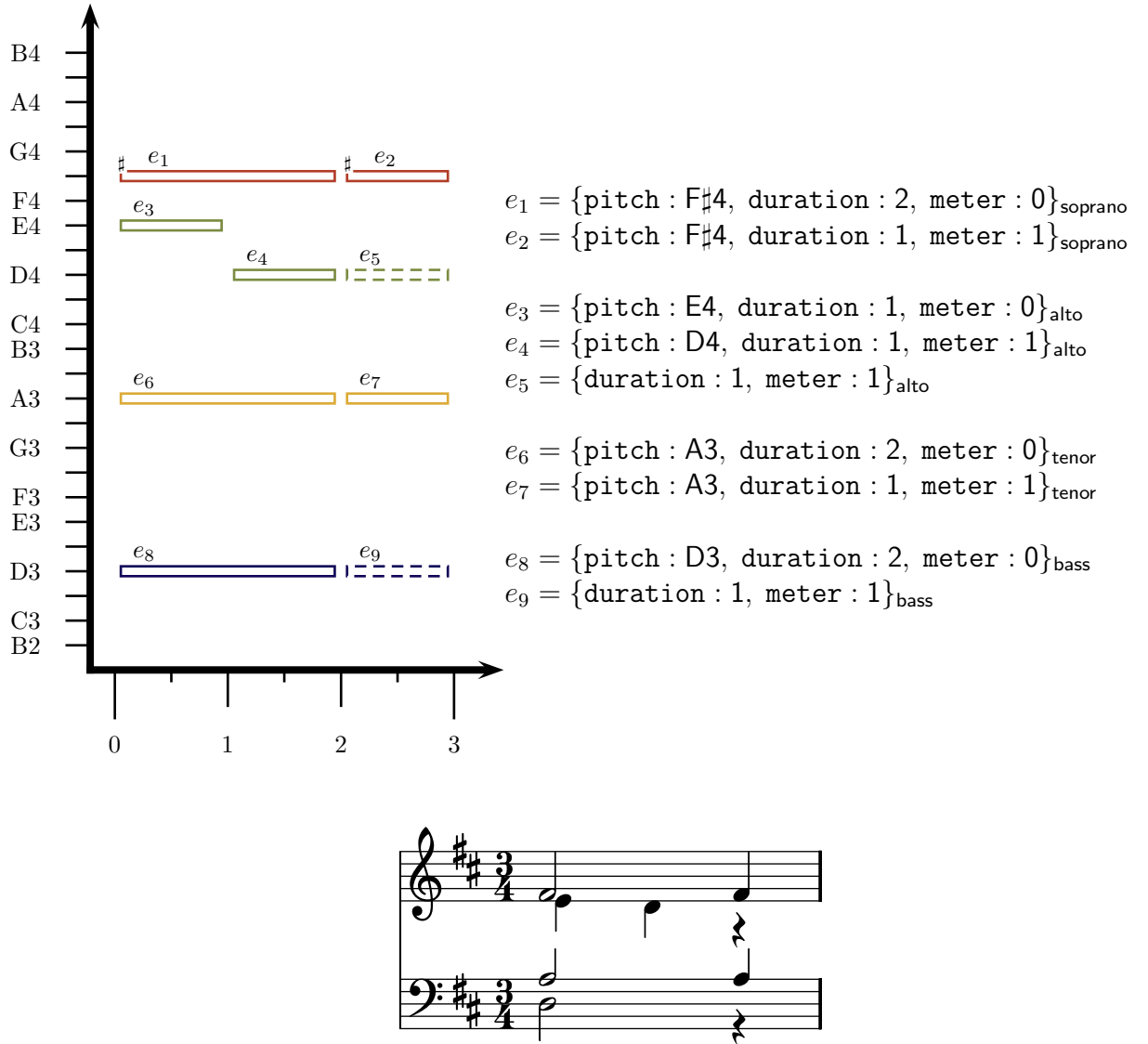


Figure 5.2: *SPP* encoding of BWV 304, bar 21. The features of Table 5.3 are shown, except **key** : Dmaj, which should appear in every event but was omitted for readability. The basic features **onset** and **offset** shown in Figure 4.1 (page 60) are also omitted

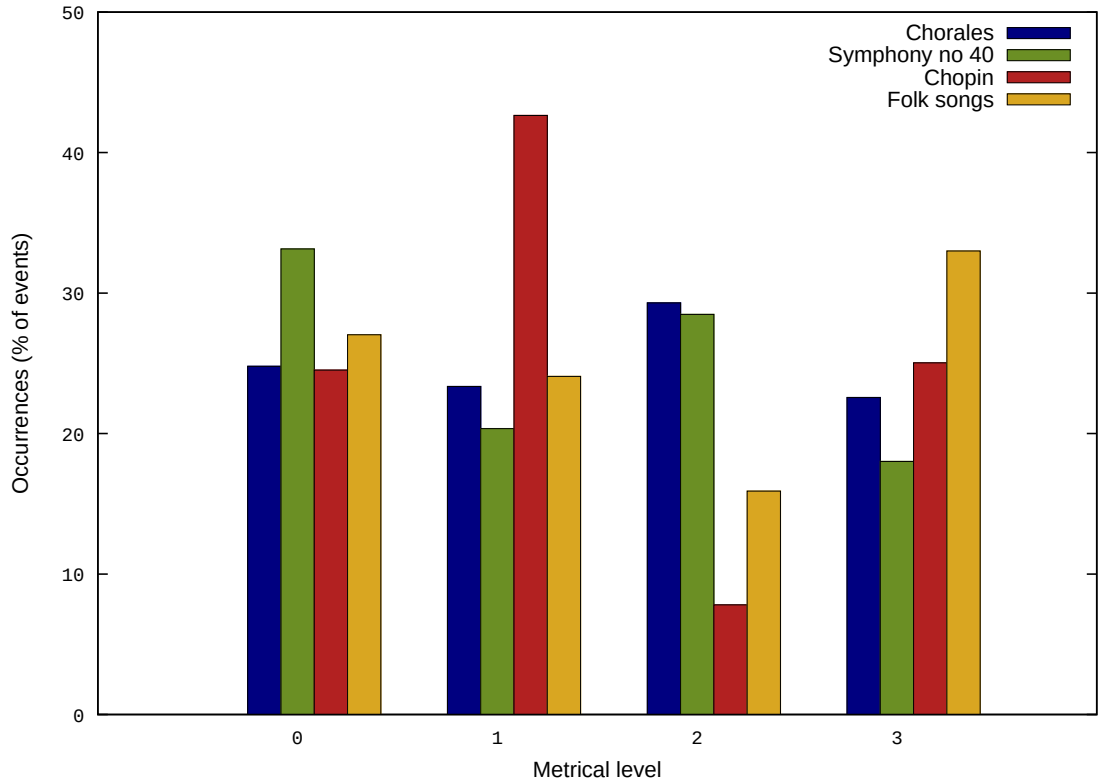


Figure 5.3: Frame-conditioned probabilities of metrical levels in Bach chorales, Symphony no.40, Chopin and folk songs

$$(P_4) \quad \{\text{meter} : X\}_x$$

Applying the query to the four corpora and analyzing the results yields Figure 5.3. The frame query is simply the empty component, i.e. we normalize the occurrences of a particular metrical level by dividing it by the total number of events. We clearly see, for example, that metrical level 1 (the second strongest level) is used significantly more in Chopin than in the other corpora. It is salient in Chopin by comparison to Bach chorales, Symphony no.40 and folk songs. The following section, Section 5.3, develops similar queries for user-defined features.

5.3 User-defined features

Queries presented in the remainder of this chapter all depend on one or more user-defined features. These features capture music-theoretical concepts that are reused

Feature name	Description	Examples
degree	Scale degree	1, 2, 3, ...
st_interval(x)	Interval over the st temporal relation	m3, P5, A4
sw_cons(x)	Consonance over the sw temporal relation	T, F
m_interval	Interval over the m temporal relation	m3, P5, ...
m_dir	Direction of melodic motion	=, +, -
m_size	Size of melodic motion	U, S (step), L (leap)

Table 5.4: Examples of user-defined features

in many different queries. Table 5.4 presents the most important features. They are all defined using feature definition rules. For example, the scale **degree** feature is defined directly from the **pitch** and **key** of a note:

	where	add
(<i>R1</i>)	{ pitch : P, key : K} _x *	degree : scale_degree (P, K)

We can easily query the sources for the values of a user-defined feature. For example, pattern *P5* below returns every value of the **degree** feature. Again, the frame query is the empty component. Results are shown in Figure F.1 on page 170.

(<i>P5</i>)	{ degree : X} _x
---------------	-----------------------------------

The feature definition rule *R1* combines the information from two basic features to create a new feature **degree**. Another typical usage of definition rules is to combine the information from two adjacent events to create a new feature. When a specific temporal relation is involved, we include it in the name of the feature:

	where	add
(<i>R2</i>)	{ pitch : P1} _x ; { pitch : P2} _x *	m_interval : diatonic_interval (P1, P2)

This **m_interval** feature uses compound intervals: for intervals that are bigger than an octave, the octave is ignored and only the remainder is considered. For example, the interval of a tenth (an octave plus a third) is classified as a third. Unless specified otherwise, other interval features in this chapter work the same way. Again,

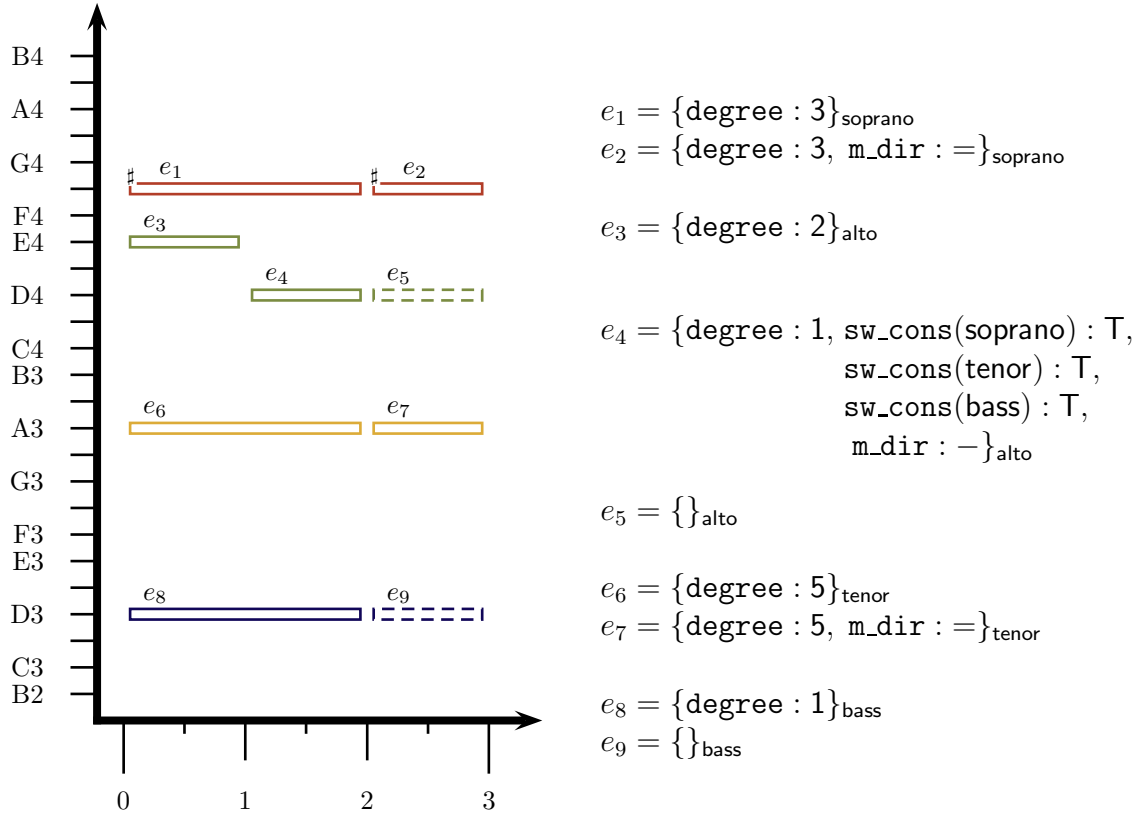


Figure 5.4: *SPP* encoding of BWV 304, bar 21. Only some of the user-defined features of Table 5.4 are shown. Features shown in figures 4.1 (page 60) and 5.2 are omitted

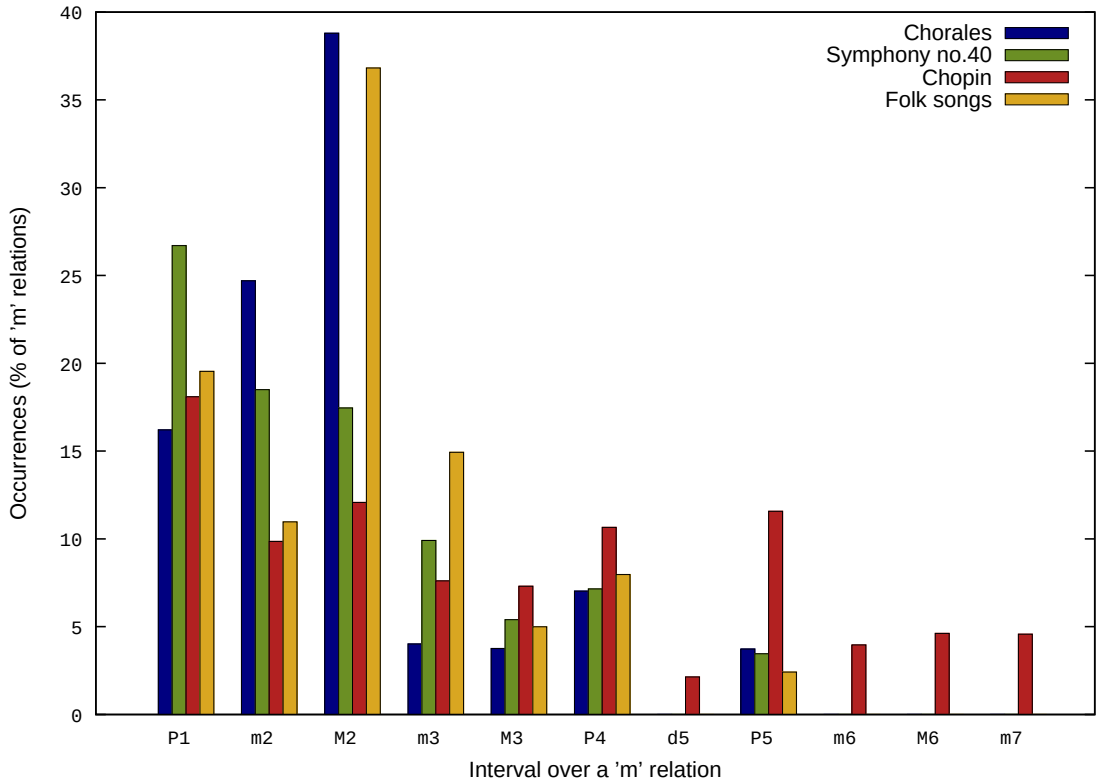


Figure 5.5: Frame-conditioned probabilities of melodic intervals in Bach chorales, Symphony no.40, Chopin and folk songs. Only intervals occurring over at least 1% of *m* temporal relations are considered

we can examine the values of the newly defined feature *m_interval*. This time, the frame query is a sequence of empty components, i.e. occurrences of melodic intervals are normalized against the number of *m* (meet) relations in a corpus. The results are shown in Figure 5.5. Interestingly, Bach chorales and folk songs contain more of the small intervals. These intervals are salient properties of these two corpora when compared with Symphony no.40 and Chopin. This could be explained by the fact that these corpora are meant to be sung, and these intervals are easier to sing. Conversely, the piano can easily realize large leaps and it is consequently not surprising that Chopin contains more of the large melodic intervals.

Computing the interval over the *st* temporal relation is similar to *R2*, except that the **where** clause contains a layer:

$$\begin{array}{c}
 \text{where} \qquad \qquad \qquad \text{add} \\
 (R3) \quad \frac{\{\text{pitch} : P1\}_x^*}{\{\text{pitch} : P1\}_y} \quad \text{st_interval}(y) : \text{diatonic_interval}(P1, P2)
 \end{array}$$

Pattern	Chorales	no.40	Chopin	Folk songs
Compensated leap	3.99%	3.50%	4.13%	4.26%
I/i chord	21.44%	2.59%	5.60%	N/A
Parallel fifth	0.12%	3.07%	10.15%	N/A
Suspension	18.88%	3.03%	2.28%	N/A
Counterpoint	65.74%	10.23%	2.33%	N/A

Table 5.5: Frame-conditioned probabilities of the main polyphonic patterns analyzed in this chapter

Other temporal relations are accommodated in the same fashion. Figure 5.4 illustrates how the source of Figure 5.2 is extended with new features after applying feature definition rules.

Again, we can examine the values of the newly defined feature `st_interval`. The frame query is *P1*, a layer of empty components, i.e. occurrences of harmonic intervals over `st` are normalized against the total number of `st` relations in a corpus. The results are shown in Figure F.2 on page 171.

5.4 Queries

Once the appropriate musical concepts are captured with user-defined features, a musicological query is expressed as an *SPP* pattern. The central queries of this chapter are shown in Table 5.6 on page 87. Their frame-conditioned probabilities are shown in Table 5.5.

The first pattern of Table 5.1 captures a melodic sequence consisting of a *compensated leap*: the melodic motion of a leap up followed by a step down. The pattern illustrates how the sequencing operator of *SPP* is used to query the melodic content of a musical source. It occurs with roughly the same probability in every corpus we consider, i.e. it is not a salient property of any of the corpus under examination. The pattern is further motivated and analyzed in Section 5.4.1.

The second pattern of Table 5.6 illustrates the use of layers in queries. By using scale `degree`, it captures the idea of a *I/i block chord*: notes that sound the scale degrees 1,3 and 5 simultaneously (the pattern is a special case where the first scale degree appears in two voices). The chord plays a central role in harmony, where it is considered to be the most stable and restful (an explanation of why pieces frequently

end on that chord). The frame-conditioned probability of the I/i chord is computed by comparing its occurrences with the total number of four-note layers. The second line of Table 5.5 shows that this central chord is salient in Bach Chorales when compared to Symphony no.40 and Chopin. This could be explained by the fact that Symphony no.40 and Chopin use significantly more combinations of scale degrees in block chords, which lowers the probabilities of any specific combination such as the 1,3,5 combination of the I/i chord. This pattern is further elaborated and analyzed in Section 5.4.2.

Compensated leap	$\{\}_x ; \{\text{m_size} : L, \text{m_dir} : +\}_x ; \{\text{m_size} : S, \text{m_dir} : -\}_x$
I/i chord	$\frac{\frac{\frac{\{\text{degree} : 5\}_z}{\{\text{degree} : 3\}_y}}{\{\text{degree} : 1\}_x}}{\{\text{degree} : 1\}_w}$
Parallel fifth	$\frac{-\{\text{et_interval}(y) : P5\}_x}{-\{\}_y} ; \frac{\{\text{st_interval}(y) : P5, \text{parallel}(y) : T\}_x}{\{\}_y}$
Suspension	$\frac{-\{\}_x}{-\{\}_y} ; \frac{\{\text{sw_cons}(y) : T, \text{m_size} : S, \text{m_dir} : -\}_x}{-\{\text{sw_cons}(x) : F, \text{accented} : T\}_y}$
Counterpoint	$\frac{\{\text{strong}(y) : T\}_x ; \{\text{weak}(y) : T\}_x}{\{\}_y} ; \frac{\{\text{strong}(y) : T\}_x ; \{\text{weak}(y) : T\}_x}{\{\}_y}$

Table 5.6: The polyphonic patterns of this chapter expressed in *SPP*

The third and fourth patterns of Table 5.6 illustrate sequences of layers. The *parallel fifth* pattern is an illustration of how combining two onset modification operators “—” in the same layer expresses the *ov* (overlap) temporal relation. The *suspension* pattern illustrates how to represent an *sw* temporal relation by using only one “—” in a layer. These patterns are discussed in detail in sections 5.4.3 and 5.4.4. As Table 5.5 illustrates, the suspension pattern is salient in Bach chorales by comparison to Symphony no.40 and Chopin. Conversely, the parallel fifth pattern is more probable in Symphony no.40 and Chopin by comparison to Bach chorales. This could be explained by the fact that Bach chorales are representative of a style where parts consist of independent melodies. The parallel fifth pattern is considered to impair part independence and has a low frame-conditioned probability in Bach chorales. The suspension pattern, on the other hand, is considered to improve part independence and has a high frame-conditioned probability in the chorales. In the other corpora, a higher probability of parallel fifths and a lower probability of suspensions is indicative of a texture where there are fewer melodies and more accompaniment material.

Finally, the last pattern of Table 5.5 illustrates a free mixture of the sequencing and layering operators. It captures some of the rules of second species counterpoint (Fux, 1965, orig. 1725), where two melodies overlap in a two notes against one fashion: two notes unfold in one melody while a single note is unfolding in the other melody. Again, the pattern is salient in Bach chorales when compared to Symphony no.40 and Chopin, as shown in Table 5.5. This could also be indicative that Bach chorales exhibit a texture in which parts contain independent melodies. The pattern is further discussed in Section 5.4.5.

5.4.1 Compensated leap

In *SPP*, a melodic pattern such as the compensated leap pattern consists of a sequence of components. In principle, a sequence of components matches any sequence of musical events that satisfy a chain of *m* temporal relations. Certainly, not every sequence would be considered a musically satisfactory melody. Music theory textbooks such as Piston (1949) do not attempt to define strict rules for the identification or composition of melodies. They do however discuss common properties of melodies. We consider two such properties here, both captured as *SPP* features: i) *m_dir*: the direction of melodic motion (whether a note has a higher pitch than its predecessor) and ii) *m_size*: the size of melodic motion (whether there is a large

pitch difference between successive notes). The first one is captured by the following feature definition rule:

	where		add
(R4)	$\{\text{pitch} : P1\}_x ; \{\text{pitch} : P2\}_x^*$	$\text{m_dir} :$	if $P2 > P1$: + else if $P2 < P1$: - else : =

The operator $<$ is given a meaning on spelled pitches by mapping the pitches to natural numbers. For example, C4 is mapped to 60, C#4 and Db4 are mapped to 61, D4 is mapped to 62, etc.

The second feature, `m_size` is defined by reusing another user-defined feature: `m_interval_nc`, the interval between successive notes.

	where		add
(R5)	$\{\text{m_interval_nc} : I\}_x^*$	$\text{m_size} :$	if $I < m2$: U else if $I < m3$: S else : L

The `m_interval_nc` feature is the non-compound version of `m_interval`: intervals bigger than the octave are not represented in their compound form, e.g. a perfect eleventh (the interval obtained by adding an octave and a fourth) is represented as P11 and not as P4. It is necessary to use `m_interval_nc` rather than `m_interval` in order to not confuse intervals bigger than the octave with a step. We say that there is no melodic motion if the interval is strictly smaller than a minor second (e.g. the diminished second or the unison). This is the first case above, where the value is U (for unison). We say that the melodic motion is a *step* if the interval is smaller than a minor third (e.g. a minor or major second). This is the second case above, where the value is S. Finally, intervals bigger than a second are classified as a *leap* or L.

Again, the operator $>$ is given a meaning on diatonic intervals by mapping the intervals to natural numbers. For example, the unison P1 is mapped to 0, a major second M2 is mapped to 2, a major third M3 is mapped to 4, and so on (these values correspond to semitone distances, see Appendix A). The value of minor and diminished intervals are computed by subtracting 1 or 2 to the value of the corresponding major interval. Finally, if an interval has a negative value (e.g.

(S1, D1)	(S2, D2)	Chorales	no.40	Chopin	Folk songs
(S, −)	(S, −)	17.07%	5.43%	1.34%	11.55%
(S, +)	(S, +)	13.07%	3.05%	1.08%	6.19%
(S, −)	(S, +)	8.04%	3.66%	1.06%	5.13%
(L, +)	(S, −)	4.00%	3.44%	3.42%	4.26%

Table 5.7: The three most frequent melodic sequences in Bach chorales and their respective frame-conditioned probabilities. The frame-conditioned probability of the compensated leap pattern is also presented

$d1 = -1$), we use the absolute value.

The frame query for the compensated leap pattern is as follows:

$$(P6) \quad \{\}_x ; \{\mathbf{m_size} : S1, \mathbf{m_dir} : D1\}_x ; \{\mathbf{m_size} : S2, \mathbf{m_dir} : D2\}_x$$

The frame query simply replaces every feature value with a value variable. This ensures a basic resemblance with the original pattern, hence returning positions in the corpus where the pattern might have occurred. In this case, it ensures that the frame query returns instances where there is melodic motion. Using empty components would also return sequences containing rests, which are clearly positions where the compensated leap pattern cannot occur. In the remainder of this chapter, frame queries are all constructed in the same fashion, by replacing feature values by feature variables.

Typically, melodies are expected to employ more steps than leaps. Using the frame query *P6*, we can test this idea. By looking at three note sequences where the last two notes both contain an **m_dir** feature and an **m_size** feature, we get a list of all pairs of melodic motion. Table 5.7 shows the three most frequent pairs in Bach chorales, along with their frame-conditioned probabilities in Symphony no.40 and Chopin. Indeed, we see that pairs where only steps are used are more frequent. When a leap occurs, it is most frequently upwards and followed by a step downwards. This is exactly what the compensated leap pattern captures. Figure 5.6 shows examples of compensated leap in Bach chorales.

Using the same query, we can also show that compensated leaps are not significantly more frequent than other kinds of leaps. This is shown in Table F.1 on page 171.

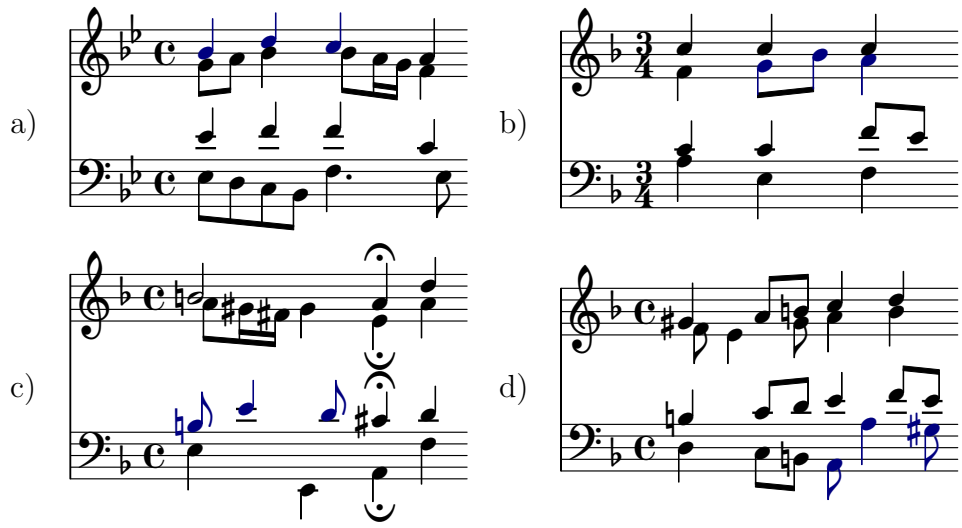


Figure 5.6: Examples of compensated leaps: a) BWV 273 bar 5 in the soprano voice; b) BWV 349 bar 13 in the alto voice; c) BWV 297 bar 3 in the tenor voice; and d) BWV 277 bar 1 in the bass voice. The excerpts are shown in piano-roll notation in figures a) F.3 on page 173; b) F.4 on page 174; c) F.5 on page 175; and d) F.6 on page 176

5.4.2 Block chord

In *SPP*, a block chord pattern consists of a layer of components. It matches events that overlap in time and satisfy the *st* temporal relation. There are many ways to describe and classify chords. This usually forms the introductory material of harmony textbooks such as Piston (1941). One way to proceed is to look at the scale degrees that form the chord. In *SPP*, this can be achieved through the frame query of the I/i block chord pattern, which is as follows:

$$(P7) \quad \begin{array}{c} \underline{\underline{\{\text{degree} : Z\}_z}} \\ \underline{\underline{\{\text{degree} : Y\}_y}} \\ \underline{\underline{\{\text{degree} : X\}_x}} \\ \underline{\underline{\{\text{degree} : W\}_w}} \end{array}$$

The pattern *P7* simply matches all cases when four distinct parts exhibit simultaneous notes. This includes cases where a scale degree is repeated, for example a chord where the third scale degree appears in two voices. Table 5.8 shows the result of running such a query on the three polyphonic corpora we consider. The results are post-processed in two ways: i) only the scale degrees are recorded (the exact variable

Scale degrees	Chorales	no.40	Chopin
1 3 5	21.44%	2.59%	5.60%
3 5 7	7.73%	0.71%	0.71%
2 5 7	7.66%	0.32%	1.57%
1 2 4 6	1.76%	0.16%	0.11%
1 2 3 5	1.11%	0.49%	0.08%

Table 5.8: The three most frequent four-part chords that use three scale degrees, as found in Bach chorales, and their frame-conditioned probabilities. Also, the two most frequent four-part chords that use four scale degrees and their frame-conditioned probabilities

assignments are ignored); and ii) the repetitions of scale degrees are ignored (e.g. the chord 1,1,3,5 is mapped to 1,3,5). The difference in frame-conditioned probability can be explained in part by the number of chords that each corpora uses. Bach chorales use 286 different block chords, while Symphony no.40 and Chopin use respectively 1460 and 1299 distinct chords. Finally, note that four-part chords that use only two scale degrees are also present in the results (data not shown in Table 5.8), e.g. a chord where only the first and third scale degree appear has a frame-conditioned probability of 0.65% in the Chorales, 2.70% in Symphony no.40 and 0.70% in Chopin.

The I/i block chord pattern introduced earlier is a special case of *P7* as it only matches cases where the first scale degree is repeated. Examples of I/i block chords in the chorales are shown in Figure 5.7. Notice how the scale degrees do not always appear in the same order. For example, in excerpt a) the scale degree 1 appears in the bass and tenor voices while it appears in the bass and alto voices in excerpt b). This illustrates how a layer in *SPP* can match different permutations of a source voices. Different permutations create different intervals between the notes of the block chord and this is a second way to describe different chords.

The alternative description of chords is invariant over transposition: the chord may be translated up and down in the pitch dimension, and the intervals between the constituent notes stay the same. By using the `st_interval` feature rather than a scale `degree` feature, we can query the sources for this alternative description of chords:

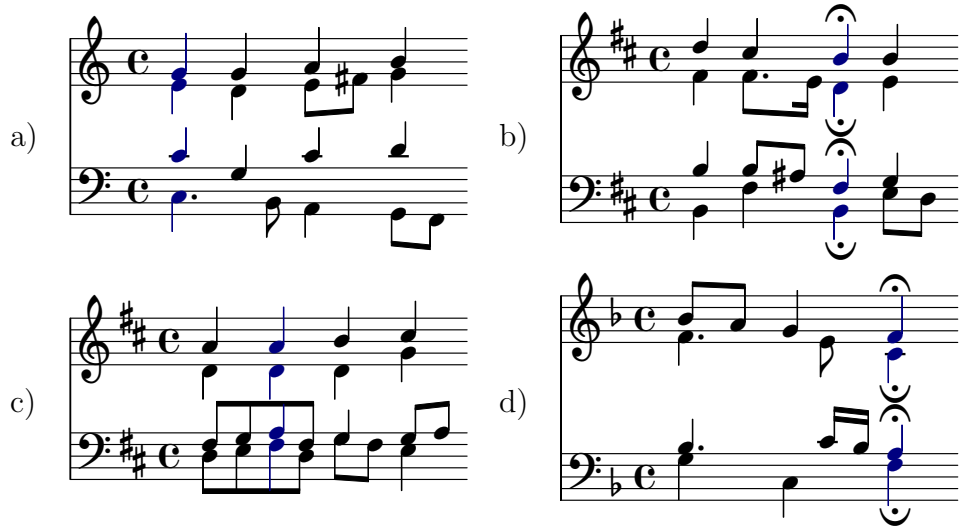


Figure 5.7: Examples of blocked I/i chords: a) BWV 292 bar 1; b) BWV 335 bar 4; c) BWV 415 bar 1; and d) BWV 438 bar 8. The excerpts are shown in piano-roll notation in figures a) F.7 on page 177; b) F.8 on page 178; c) F.9 on page 179; and d) F.10 on page 180

$$(P8) \quad \frac{\frac{\frac{\{\text{st_interval}(y) : Y\}_z}{\{\text{st_interval}(x) : X\}_y}}{\{\text{st_interval}(w) : W\}_x}}{\{\}_w}$$

This records the interval between the different parts appearing in the chord. We can also look at the intervals from a single part to all other parts using the same features and a single component:

$$(P9) \quad \{\text{st_interval}(x) : X, \text{st_interval}(y) : Y, \text{st_interval}(z) : Z\}_w$$

Assuming part *w* of *P9* is the lowest note of the chord, the two most frequent chords in Bach chorales are formed by the intervals (P1, M3, P5) and (P1, m3, P5). These respectively correspond to the *major chord* and *minor chord* in a specific configuration where the *root* of the chord appears twice: once as the lowest note and once through the interval P1 (with the lowest note). In this configuration, the major chord forms 5.18% of all four-part block chords in the chorales as shown by Table 5.9. The table also shows the most frequent permutations of this chord.

Chord	Chorales	no.40	Chopin
Major	5.19%	0.25%	0.18%
P5, P4, M3	25.51%	18.48%	16.56%
M3, m3, P4	20.92%	13.86%	15.00%
P1, M3, m3	19.23%	27.60%	28.40%

Table 5.9: Major chord with the root as the lowest note and its frame-conditioned probability with respect to all block chords. The following lines show the three most frequent permutations and their frame-conditioned probabilities with respect to other possible permutations

These are obtained by executing query *P8* and looking at intervals between parts. For example, the most frequent permutation has the fifth occurring first (between *w* and *x*), then the octave (between *w* and *y*, which is equivalent to fourth between *y* and *x* as shown in the table) and finally the major third (between *w* and *z*).

Using queries *P8* and *P9*, a similar analysis can easily be performed for other types of chords. For example, Table F.2 on page 172 shows the result for the minor chord with its root as the lowest note.

Note that our analysis of queries *P8* and *P9* is based on post-processing of *SPP* pattern matching results. In particular, the possible permutations of the major chord shown in Table 5.9 are not isolated directly in *SPP* but rather with a specialized post-processing script. We employed a similar strategy to isolate the lowest note of a block chord. However, it would be possible to use a *SPP* feature for this purpose, for example a feature that is true only when the note belongs to a four-part layer and has a smaller pitch than the pitches of the other three notes of the layer.

5.4.3 Parallel fifth

In *SPP*, a pattern describing how melodies interact with each other can be specified as a sequence of layers. The onset modification operator “—” can be used to specify the temporal relations between the notes of the melodies. Features are used, for example, to specify how the notes relate in terms of pitch. Music textbooks contain comments on how two melodies should and should not interact. This section describes the parallel fifth pattern, a polyphonic interaction that is considered as best avoided when melodies are meant to be independent.

A parallel fifth occurs when two melodies are separated by a perfect fifth and exhibit the same melodic motion. In the parallel fifth pattern shown in Table 5.6,

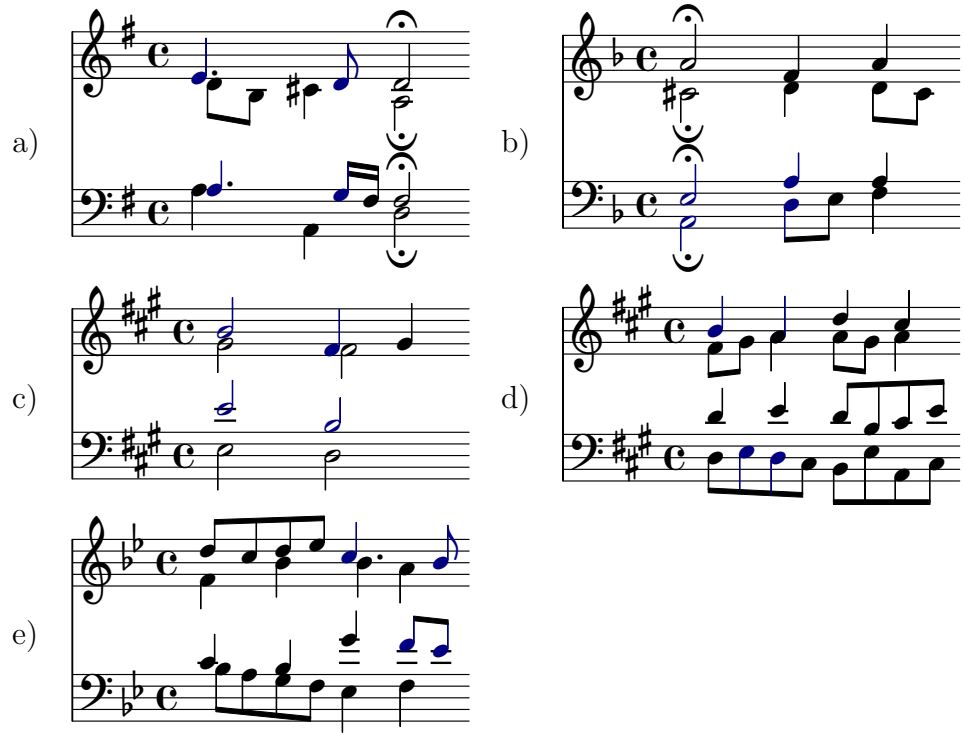


Figure 5.8: Examples of parallel fifths: a) BWV 263 bar 6; b) BWV 301 bar 3; c) BWV 323 bar 8; d) BWV 355 bar 15; and e) BWV 361 bar 12. The excerpts are shown in piano-roll notation in figures a) F.11 on page 181; b) F.12 on page 182; c) F.13 on page 183; d) F.14 on page 184; and e) F.15 on page 185

are consistent with the results of Fitsioris and Conklin (2008).

We can generalize the parallel fifth query and look for any parallel interval:

$$\begin{array}{lcl}
 & & \frac{-a}{-b} ; \frac{c}{d} \\
 (P10) & \bar{a} & = \{\text{et_interval}(y) : I\}_x \\
 & \bar{b} & = \{\}_y \\
 & \bar{c} & = \{\text{st_interval}(y) : I, \text{parallel}(y) : T\}_x \\
 & \bar{d} & = \{\}_y
 \end{array}$$

The results are as follows: parallel fifths and octaves (unisons) are very rare in Bach chorales, while parallel minor thirds and major sixths are the most frequent. This is shown in Table 5.10. The contrast with Symphony no.40 and Chopin is also interesting. As one might expect, parallel fifths and octaves are not avoided in these

Interval	Chorales	no.40	Chopin
m3	42.24%	6.33%	14.70%
M6	20.81%	5.27%	12.67%
P4	14.89%	3.83%	11.06%
P5	0.12%	3.07%	10.15%
P1	0.12%	35.35%	20.56%
d5	0.02%	0.78%	2.45%

Table 5.10: Most and least frequent parallel intervals and their frame-conditioned probabilities

corpora and, on the contrary, are salient by comparison to Bach chorales. This is probably due to a strong presence of accompaniment material in these corpora, e.g. sequences of chords.

5.4.4 Suspension

The suspension pattern is similar in structure to the parallel fifth pattern of Section 5.4.3. It concerns two melodies and how they interact. As such it is expressed in *SPP* as a sequence of layers. Contrary to the parallel fifth pattern, the suspension is commonly used to make two melodies more independent. Another difference with the parallel fifth pattern is that it does not refer to a particular interval. Instead, two classes of intervals are used: consonant intervals and dissonant intervals. The former are considered stable (or restful) and the latter unstable. In *SPP*, this is defined as follows:

	where	add
(R7)	{sw_interval(y) : I} _x	if I ∈ {P1, m3, M3, P5, m6, M6} then T else F
	sw_cons(y) :	

The rule is a simple transformation that classifies some intervals as consonances: unison, minor and major thirds, perfect fifths and minor and major sixths. Other intervals are classified as dissonant. When a dissonance appears, it is often said to be approached in a particular way. When a dissonance is replaced by a consonance, it is said to be resolved. The suspension pattern is one of many ways to approach and resolve a dissonance in a polyphonic texture. The key for a suspension is that



Figure 5.9: Examples of suspensions: a) BWV 259 bar 8; b) BWV 390 bar 12; c) BWV 393 bar 7; and d) BWV 285 bar 1. The excerpts are shown in piano-roll notation in figures a) F.16 on page 186; b) F.17 on page 187; c) F.18 on page 188; and d) F.19 on page 189

Intervals	Chorales	no.40	Chopin
M2 m3	29.05%	10.00%	1.99%
P4 M3	15.32%	13.33%	7.07%
P4 P5	11.62%	3.33%	6.19%

Table 5.11: Most frequent suspensions in Bach chorales and their frame-conditioned probabilities

the dissonance is both introduced and resolved through an *sw* temporal relation. Figure 5.9 shows examples of suspension in Bach chorales.

Pattern *P11* below generalizes the suspension pattern and look for the exact intervals that form the dissonance and its resolution to a consonance. Results are shown in Table 5.4.4.

$$\frac{-a}{-b} ; \frac{c}{-d}$$

$$\begin{aligned}
 (P11) \quad \bar{a} &= \{ \}_x \\
 \bar{b} &= \{ \}_y \\
 \bar{c} &= \{ \text{sw_interval}(y) : \text{I2}, \text{m_size} : \text{S}, \text{m_dir} : - \}_x \\
 \bar{d} &= \{ \text{sw_interval}(x) : \text{I1}, \text{accented} : \text{T} \}_y
 \end{aligned}$$

Most other ways of approaching and resolving a dissonance do not exhibit the **sw** temporal relation as the suspension does. Rather, they can be described with the **st** temporal relation and the **et** (end together) temporal relation. We can represent these additional ways as an $\mathcal{SPP}$ sequence:

$$\begin{array}{lcl}
 & & a ; b ; c \\
 (P12) \quad \bar{a} & = & \{\}_x \\
 & \bar{b} & = \{\text{et_cons}(y) : F, \text{m_size} : S1, \text{m_dir} : D1, \text{accented} : A\}_x \\
 & \bar{c} & = \{\text{st_cons}(y) : T, \text{m_size} : S2, \text{m_dir} : D2\}_x
 \end{array}$$

Explicitly, the sequence encodes a single melody. We use features to encode the temporal relations that are formed with the second melody. The voice variable y above refers to that second melody.

By setting different values for the variable representing melodic motion and size in the pattern $P12$ above, we can query the sources for many types of polyphonic embellishments (Huron, 2007): for example, passing tone, neighbor tone, and appoggiatura. The feature **accented** simply refers to the metrical level of b (the note that introduces the dissonance): if it is stronger than the metrical level of c (the note that resolves the dissonance), we say the embellishment is accented. Figure 5.10 shows examples of accented and unaccented passing tones in Bach chorales.

For completeness, Table F.3 on page 172 gives a list of all other embellishments we analyzed using query $P12$.

5.4.5 Counterpoint

The patterns of sections 5.4.3 and 5.4.4 concern the case where melodies interact in a note against note fashion. Other cases can be captured in $\mathcal{SPP}$ as well, by using a free mixture of the sequencing and layering operators. The last pattern of Table 5.6 (page 87) illustrates the case where melodies interact in a two notes against one fashion: two notes unfold in the first melody while only one note is unfolding in the second melody. The first note is said to be in the *strong* position. It forms an **st** temporal relation and, according to the rules of counterpoint, it has to form a consonance. The second note is said to be in the *weak* position. It forms an **sw** temporal relation and it can form a dissonance only if it acts as a passing tone between two strong notes. Otherwise, it has to form a consonance. By following counterpoint treatises such as Fux (1965, orig. 1725), one can formulate many rules

where **add**

(R9) $\{\text{st_cons}(y) : C, \text{par1_5}(y) : P\}_x^*$ $\text{strong}(y) : \begin{array}{ll} \text{if } C = T \wedge P = F & : T \\ \text{else} & : F \end{array}$

Weak positions can act as passing tones. To encode such a rule, we reuse the polyphonic embellishment pattern *P12* of Section 5.4.4. Again, it is transformed into a feature:

where

$a ; b^* ; c$

$\bar{a} = \{\}_x$
 $\bar{b} = \{\text{et_cons}(y) : C1, \text{m_size} : S1, \text{m_dir} : D1\}_x$
(R10) $\bar{c} = \{\text{st_cons}(y) : C2, \text{m_size} : S2, \text{m_dir} : D2\}_x$

add

$\text{passing}(y) : \begin{array}{ll} \text{if } C1 = F \wedge C2 = T \wedge S1 = S \\ \quad \wedge S2 = S \wedge D1 = D2 & : T \\ \text{else} & : F \end{array}$

Using this **passing** feature, we can define a feature **weak** to capture the acceptable weak positions.

where **add**

(R11) $\{\text{st_cons}(y) : C, \text{passing}(y) : P\}_x^*$ $\text{weak}(y) : \begin{array}{ll} \text{if } C = T \vee P = T & : T \\ \text{else} & : F \end{array}$

Equipped with the features **strong** and **weak** and the structure shown in Table 5.6 (page 87), we capture some basic rules of counterpoint. Figure 5.11 shows some examples from Bach chorales.

More rules could be added following Fux (1965, orig. 1725), simply by defining more features. Another interesting query consists of adding a third part. This is easy to achieve in $\mathcal{SPP}$ by keeping the same features and extending structure:

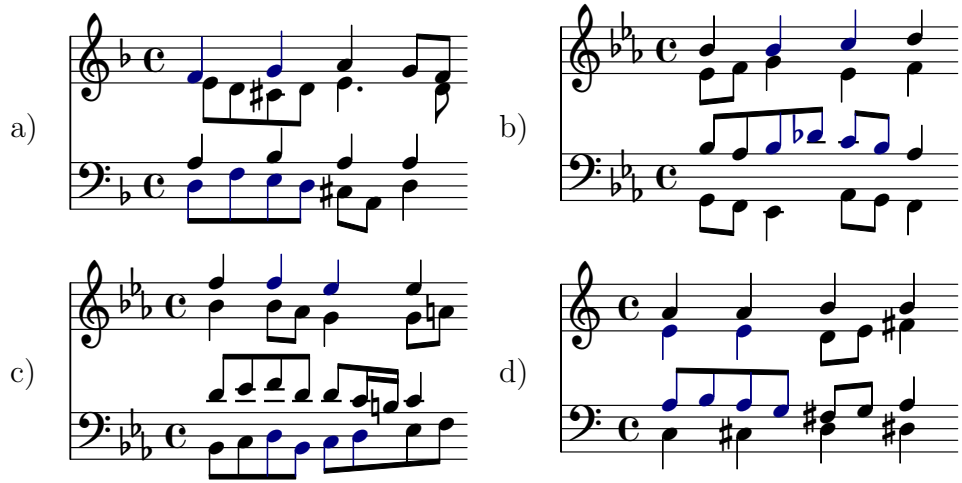


Figure 5.11: Examples of two-voice counterpoint patterns: a) BWV 277 bar 3; b) BWV 285 bar 9; c) BWV 380 bar 3; and d) BWV 418 bar 7. The excerpts are shown in piano-roll notation in figures a) F.24 on page 194; b) F.25 on page 195; c) F.26 on page 196; and d) F.27 on page 197

$$\begin{array}{c}
 \begin{array}{cc}
 \frac{a ; b}{\frac{a' ; b'}{c}} & ; \frac{d ; e}{\frac{d' ; e'}{f}}
 \end{array} \\
 (P13) \\
 \begin{array}{ll}
 \bar{a}, \bar{d} = \{\text{strong}(y) : T\}_x & \bar{a'}, \bar{d'} = \{\text{strong}(y) : T\}_w \\
 \bar{b}, \bar{e} = \{\text{weak}(y) : T\}_x & \bar{b'}, \bar{e'} = \{\text{weak}(y) : T\}_w \\
 \bar{c}, \bar{f} = \{\}_y &
 \end{array}
 \end{array}$$

Figure 5.12 shows examples of the three-voice case from Bach chorales.

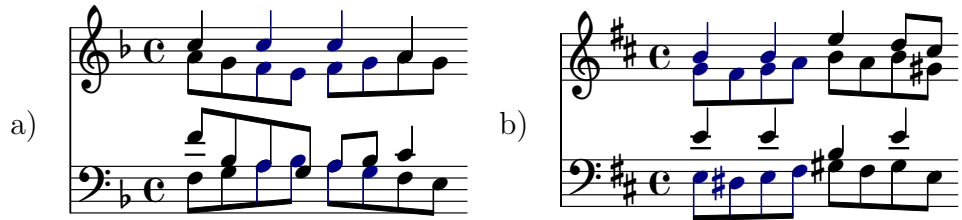


Figure 5.12: Examples of three-voice counterpoint patterns: a) BWV 374 bar 7; and b) BWV 398 bar 11. The excerpts are shown in piano-roll notation in figures a) F.28 on page 198; and b) F.29 on page 199

5.5 Summary

This chapter explored many different applications of the *SPP* method. We showed that it is well-suited to capture musical knowledge and patterns, especially those which are associated with n-part polyphony. The queries developed in this chapter were also applied to musical sources of other genres and textures, demonstrating the versatility of the method. The chapter shows how different *SPP* structures correspond well to different musical concepts. Also, we showed how queries can be built in a modular way, by expanding on previously defined features and patterns. It is also easy to expend a query by adding a feature or new components. The next chapter focuses on the expressiveness of *SPP* in terms of temporal relations and provides a thorough comparison with relational patterns and Humdrum. Then, Chapter 7 discusses further properties of the method such as efficiency and suitability for data mining.

Chapter 6

The expressiveness of polyphonic patterns

This chapter compares the expressiveness of three polyphonic pattern languages: relational patterns, Humdrum, and $\mathcal{SPP}$. The comparison is restricted to the temporal aspect of patterns. The result is a hierarchy of languages, with relational patterns strictly subsuming Humdrum and $\mathcal{SPP}$. In addition, we define a fourth language that characterizes the intersection of Humdrum and $\mathcal{SPP}$. This language, $\mathcal{SPP}_{\text{seq}}$, is a restriction of $\mathcal{SPP}$ where a sequence is never embedded in a layer (a pattern is always structured as a sequence of layers). Both the parallel fifth and suspension patterns are expressible in this restricted $\mathcal{SPP}_{\text{seq}}$. This suggests that it is a good candidate pattern language to use for polyphonic music data mining.

6.1 A hierarchy of polyphonic pattern languages

This chapter establishes the hierarchy shown in Figure 6.1. Four pattern languages are compared: $\mathcal{R}$, $\mathcal{H}$, $\mathcal{SPP}$, and $\mathcal{SPP}_{\text{seq}}$. The languages $\mathcal{R}$ and $\mathcal{H}$ respectively correspond to interpretations of relational patterns and Humdrum that focus on the temporal aspect only. Similarly, this chapter considers only the temporal aspect of $\mathcal{SPP}$ and its restriction $\mathcal{SPP}_{\text{seq}}$. We focus on these *temporal patterns* (Bergeron and Conklin, 2009) as they show how the approaches differ. In terms of musical relations, the approaches are nearly equivalent, with the exception that $\mathcal{SPP}$ encodes

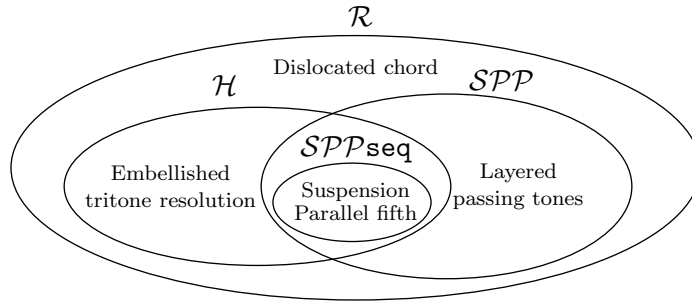


Figure 6.1: A hierarchy of polyphonic pattern languages

musical relations as features (this is discussed further in Chapter 7).

The methodology used to obtain the hierarchy of Figure 6.1 is threefold. First, we developed a baseline notation for the representation of temporal patterns. Then, all languages are interpreted with respect to that baseline notation. Finally, we provide example patterns that distinguish the many languages, e.g. the layered passing tones pattern is expressible in SPP but not in $\mathcal{H}$. This enables us to assert the various statements expressed by the hierarchy: i) that $\mathcal{R}$ subsumes both $\mathcal{H}$ and SPP (Section 6.5.1), ii) that $\mathcal{H}$ does not subsume SPP (Section 6.5.2), iii) that SPP does not subsume $\mathcal{H}$ (Section 6.5.3), iv) and finally that both $\mathcal{H}$ and SPP subsume $SPPseq$ (Section 6.5.4).

The baseline notation depends on the four temporal relations introduced in this dissertation: **m** (meet), **st** (start together), **sw** (start while), and **ov** (overlap). A temporal pattern can be written as a network of these temporal relations or *temporal network*. Nodes in the network are component indices $a, b, c, \dots$ and have the same meaning as in Chapter 4: unique identifiers for the components of a pattern. A pattern component matches a musical event, either a note or a rest.

Figure 6.2 presents the temporal networks corresponding to the examples of Figure 6.1. Consider for example the parallel fifth pattern (Figure 6.2b). It uses four component indices and hence matches four events. The network stipulates that events a and c must overlap in time while events b and d must start together. In addition, events a, b and c, d must respectively meet. Nothing else is specified by the pattern, in particular no musical relations are specified. Throughout this chapter, we nonetheless discuss temporal patterns as having musical meaning, and provide musical illustrations.

Even with the simple patterns of Figure 6.2, we can establish that the four pattern

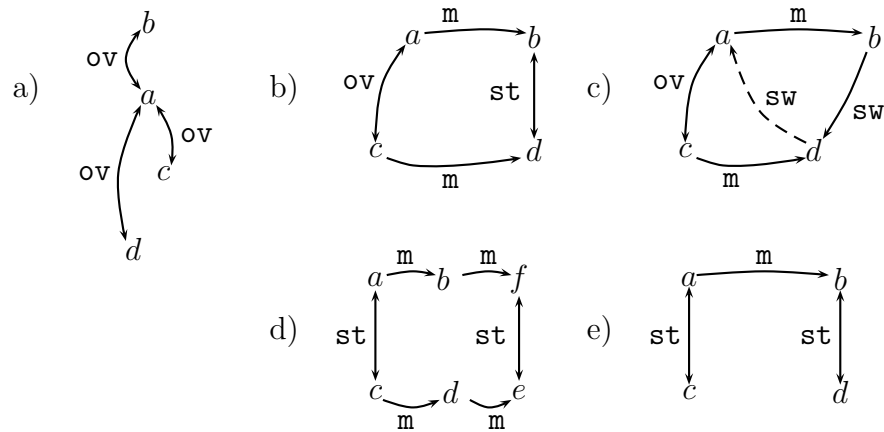


Figure 6.2: Temporal networks for the following patterns: a) dislocated chord, b) parallel fifth, c) suspension, d) layered passing tones, and e) tritone resolution. The network d) can be expressed in SPP but not in $\mathcal{H}$. The network e) can be expressed in $\mathcal{H}$ but not in SPP .

languages analyzed in this chapter do not possess the same expressiveness. This is mainly done by showing that one language can express some pattern that another cannot. In addition, the examples are useful to understand how the languages differ.

The first example, Figure 6.2a, is called a *dislocated chord* pattern. It could capture notes that overlap with the root of a chord without necessarily overlapping with one another. Figure 6.3 shows two instances of this pattern. In both cases, the note in the bass voice corresponds to a . The temporal network does not differentiate the order in which the other notes sound: in the first case they mostly form a layer, but in the second case none of the notes start at the same time. This pattern cannot be represented in $\mathcal{H}$ or SPP . In both cases, the only way to specify that events overlap is to place all the events in a layer. But then, every event overlaps with every event in the layer. This would not capture either case of Figure 6.3, for example in the second case the C in the tenor voice (note b) does not overlap with the $F\sharp$ in the alto voice (note d).

The two following temporal networks, Figure 6.2b and Figure 6.2c, correspond to the parallel fifth and suspension patterns. As discussed in Chapter 5, these are expressed as a sequence of two layers. The first layer expresses the ov temporal relation between a and c . The second layer expresses the st/sw temporal relation between b and d . The sequencing of these layers expresses the m temporal relations between a, b and c, d . These networks are hence expressible in SPP_{seq} , the restriction of SPP where a layer never contains a sequence. The temporal relation $sw(d, a)$ is shown with a dash line as it is *implicit* in the network: it can be inferred from

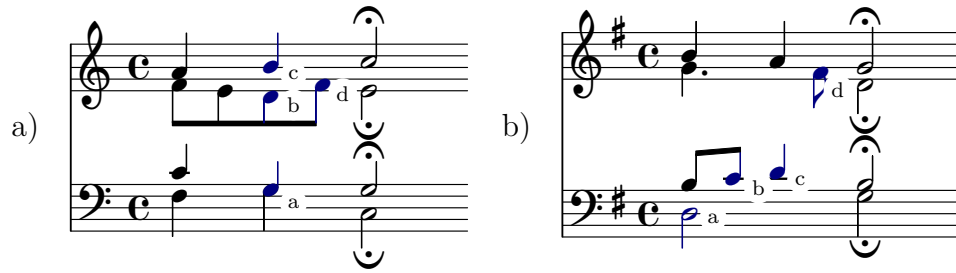


Figure 6.3: Examples of dislocated V^7 chords: a) BWV 285 bar 15; and b) BWV 318 bar 13. The excerpts are shown in piano-roll notation in figures a) F.30 on page 200; and b) F.31 on page 201

the other temporal relations of the network. We say that the network without the implicit relation is equivalent to the network that possesses it.

The fourth temporal network, Figure 6.2d, represents a *layered passing tones* pattern. It consists of two sequences: a, b, f and c, d, e . With the appropriate musical relations, each sequence could capture a passing tone pattern as described in Chapter 5. Figure 6.4 shows instances of the pattern. According to the temporal network, the first notes of the sequence a, c start together, as do the last notes f, e . However, the intermediary notes b, d do not form a temporal relation. They start together in the first instance of Figure 6.4, but this is not the case in the second instance. This can be captured in SPP using layered sequences, where the beginning of the sequences have to start together but the remainder of the sequence unfold independently. In $\mathcal{H}$, however, this is impossible to represent. A pattern with overlapping sequences has to express temporal relations between the components of the sequence.

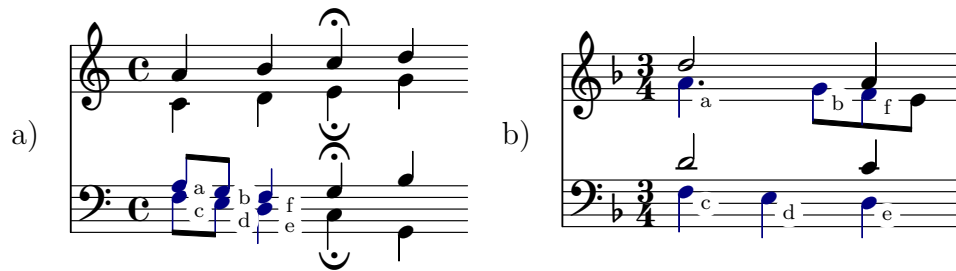


Figure 6.4: Examples of layered passing tones: a) BWV 255 bar 2; and b) BWV 320 bar 19. The excerpts are shown in piano-roll notation in figures a) F.32 on page 202; and b) F.33 on page 203

The fifth temporal network illustrates how in Humdrum one can use regular expressions to indifferently match either the start of an event or the continuation of

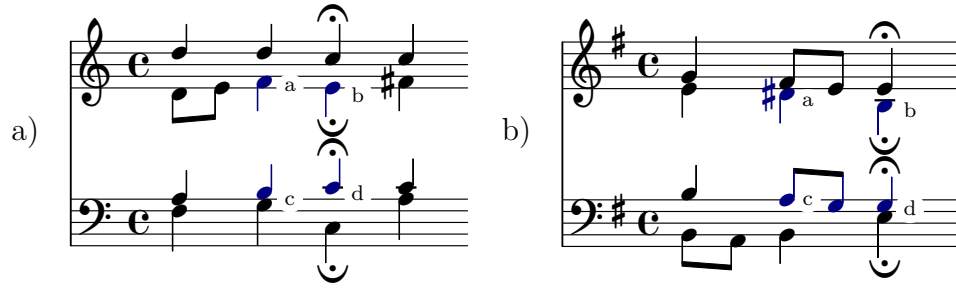


Figure 6.5: Examples of embellished tritone resolutions: a) BWV 257 bar 2; and b) BWV 315 bar 13. The excerpts are shown in piano-roll notation in figures a) F.34 on page 204; and b) F.35 on page 205

an event. In the temporal restriction $\mathcal{H}$, this is captured by a special “don’t care” symbol $\star$. Using such a symbol, one can specify a “gaped” sequence, where the relation between some component in the sequence and its successor is not specified. Figure 6.2 e) can be read as layered sequences. The $\mathbf{m}$ temporal relation appears in the upper sequence, but is absent from the lower sequence. The result is that an intervening note could appear between c and d in the lower sequence and the pattern would still match. With the appropriate musical relations, this could represent an *embellished tritone resolution* pattern, when the interval of a tritone resolves to the interval of a third, with a facultative embellishment. The instances of Figure 6.5 show how the pattern matches with and without this embellishment.

This cannot be expressed in $\mathcal{SPP}$. There is a mechanism to express gaped sequences in $\mathcal{SPP}$, using a sequence of layers where the size of layers varies. As a result, some of the components do not align with preceding or following components, and this creates gaps. This is however less flexible than $\mathcal{H}$ “don’t care” symbol and does not apply in cases where the layers are of the same size. In that case, every component of the layer is aligned with a predecessor. This is the case of Figure 6.2e. The $\mathbf{m}$ relation between c and d would have to be specified in $\mathcal{SPP}$.

The remainder of this chapter develops the arguments presented here more formally.

6.2 Relational patterns $\mathcal{R}$

When restricted to express only temporal relations, the relational pattern language presented in Chapter 3 can be defined as follows.

Definition 17 A relational pattern $r \in \mathcal{R}$ is defined as a set of temporal relations ω expressed over component indices ϵ :

$$r ::= \omega, \dots, \omega \quad \mathbf{with} \quad \omega ::= \begin{array}{l} \mathbf{m}(\epsilon, \epsilon) \\ | \quad \mathbf{ov}(\epsilon, \epsilon) \\ | \quad \mathbf{st}(\epsilon, \epsilon) \\ | \quad \mathbf{sw}(\epsilon, \epsilon) \end{array}$$

This is equivalent to the baseline representation of temporal networks. This explains that $\mathcal{R}$ subsumes all other languages discussed in Figure 6.1.

6.3 From Humdrum to $\mathcal{H}$

As described in Chapter 3, a Humdrum pattern is a list of regular expressions. Typically, such regular expressions are designed to match multiple columns. The initial columns match event tokens and the remaining columns match properties of these events, for example encodings of the musical relations they form. An event token is either the beginning of an event or the continuation of an event. In addition, the regular expression can formulate a special “don’t care” option that matches any token. When ignoring the musical relations, a Humdrum pattern can be construed as a matrix, where lines are layers containing events and continuations of events, and columns are sequences.

Definition 18 A pattern $h \in \mathcal{H}$ is defined as a matrix where every position h_{ij} is a token of the form:

$$h_{ij} ::= \begin{array}{l} \epsilon \\ | \quad (\epsilon) \\ | \quad \star \end{array}$$

A token of the form ϵ matches the beginning of an event. A token of the form (ϵ) matches the continuation of an event. The token $\star$ is the special “don’t care” symbol that matches in any circumstance. For example, the suspension pattern presented in Chapter 3 can be expressed as the following matrix, where only the temporal aspect is considered:

$$(E10) \quad \begin{bmatrix} d & (a) \\ (d) & b \end{bmatrix}$$

Notice how only three events are present. This is slightly different from the $\mathcal{SPP}$ suspension pattern that uses four notes.

6.3.1 Well-formedness

Not every pattern in $\mathcal{H}$ is consistent with the Humdrum patterns as described above. To make sure that every matrix can be interpreted as a Humdrum pattern, we impose the following well-formedness constraints.

Definition 19 A pattern $h \in \mathcal{H}$ is well-formed if it satisfies the following constraints:

- Except for the first line, a continuation (ϵ) may only occur if it follows the start of event ϵ , either directly or after a sequence of continuations (ϵ) .
- A component index ϵ may only occur more than once if it is part of a sequence of continuations.
- Except for the first line, there is no line that contains only continuation symbols.

The first constraint simply enforces that continuation symbols are used in context, with the event being continued clearly identified. We make an exception for the first line: if the pattern starts with the continuation of an event, the beginning cannot be identified. The second constraint ensures that a component index always denotes a single component. The third constraint forbids lines that do not add any information to the pattern. As continuation symbols are used in context, a line that contains only continuations does not add more information than the preceding lines. Again, we make an exception for the first line, as the context is not available. For example, the patterns below are not well-formed:

$$(E11) \quad \begin{bmatrix} a & b \\ c & (d) \\ (c) & (d) \end{bmatrix} \quad \begin{bmatrix} a & b \\ (a) & c \\ d & b \end{bmatrix}$$

In the first example above, the continuation token (d) is not preceded by the beginning of d . Also, there is a line containing only continuation tokens. That line implies that c and d overlap. Clearly, this does not make the pattern anymore precise, as the preceding line already implies that c starts while d is unfolding. The second case of example *E11* is not well-formed as the token b appears twice. This suggests that the event b starts twice, at different times, which cannot be given a reasonable interpretation.

6.3.2 Interpretation

The interpretation of a pattern in $\mathcal{H}$ is relatively simple. It consists of looking at pairs of tokens and how they form temporal relations. Tokens that follow one another express the $\mathbf{m}$ temporal relation. Tokens that are on the same line specify either $\mathbf{st}$, $\mathbf{sw}$ or $\mathbf{ov}$ depending on the combination of continuation markers.

Definition 20 A pattern $h \in \mathcal{H}$ is interpreted as follows with respect to the temporal relations it enforces:

- For any two tokens h_{ij}, h_{ik} on the same line, the interpretation is as follows:
 - $\epsilon_1 \quad \epsilon_2$ form the relation $\mathbf{st}(\epsilon_1, \epsilon_2)$
 - $\epsilon_1 \quad (\epsilon_2)$ form the relation $\mathbf{sw}(\epsilon_1, \epsilon_2)$
 - $(\epsilon_1) \quad \epsilon_2$ form the relation $\mathbf{sw}(\epsilon_2, \epsilon_1)$
 - $(\epsilon_1) \quad (\epsilon_2)$ form the relation $\mathbf{ov}(\epsilon_1, \epsilon_2)$
- For any two tokens h_{ij}, h_{i+1j} in the same column and successive lines, the interpretation is as follows:
 - ϵ_1
 ϵ_2 form the relation $\mathbf{m}(\epsilon_1, \epsilon_2)$
 - (ϵ_1)
 ϵ_2 form the relation $\mathbf{m}(\epsilon_1, \epsilon_2)$

- Any other pair does not form a relation. In particular, the “don’t care” token $\star$ never forms a relation.

For example, the suspension pattern is interpreted as follows:

$$(E12) \quad \begin{bmatrix} d & (a) \\ (d) & b \end{bmatrix} \rightsquigarrow \begin{array}{l} \mathbf{sw}(d, a), \quad \mathbf{sw}(b, d), \\ \mathbf{m}(a, b) \end{array}$$

6.4 $\mathcal{SPP}$ and $\mathcal{SPPseq}$

The interpretation of $\mathcal{SPP}$ in terms of temporal networks is straightforward. It suffices to apply the definitions of Chapter 4, while considering only the statements that concern temporal relations. In addition, we define the language $\mathcal{SPPseq}$, a restriction of $\mathcal{SPP}$ where a layer never contains a sequence. The restriction intentionally excludes patterns that are not expressible in $\mathcal{H}$. This yields a way to characterize the intersection between $\mathcal{SPP}$ and $\mathcal{H}$. The restriction is defined syntactically:

Definition 21

An $\mathcal{SPPseq}$ pattern φ is defined according to the following syntax:

$$\begin{array}{ll} \varphi ::= & \psi \quad \text{with} \quad \psi ::= \quad \epsilon \\ & | \varphi ; \psi \quad \quad \quad | -\epsilon \\ & \quad \quad \quad \quad \quad | \frac{\psi}{\psi} \end{array}$$

In addition, the pattern must satisfy the following constraint: except for the first layer, a layer contains at most one component of the form $-\epsilon$. This well-formedness constraint is deliberately formulated to ensure that every $\mathcal{SPPseq}$ pattern has an equivalent in $\mathcal{H}$. The rationale is simple: when a layer expresses the **ov** temporal relation in $\mathcal{SPP}$, the relation does not need to be refined into an **st** or **sw** relation. This is different in $\mathcal{H}$, as the continuation markers that express **ov** are necessarily preceded by the start of their respective event, the relation has to be refined into either **st** or **sw**. The first line of an $\mathcal{H}$ is an exception and, correspondingly, we make an exception for the first layer of an $\mathcal{SPPseq}$ pattern.

The interpretation of an SPP_{seq} pattern is the same as that of an SPP pattern. As every SPP_{seq} pattern is also an SPP pattern, it follows trivially that SPP subsumes SPP_{seq} .

SPP_{seq} can express most patterns presented in Chapter 5, including the parallel fifth and suspension. Only the counterpoint pattern is not expressible as a sequence of layers.

6.5 Comparing languages

We compare languages by comparing which temporal networks they support. We say that a pattern can be expressed in another language if there exists a pattern in that language that enforces the same temporal network, or a network that we can prove to be equivalent. The equivalence of two temporal networks can be obtained by showing that some temporal relation is implicit in the network.

6.5.1 $\mathcal{R}$ subsumes $\mathcal{H}$ and SPP

The result concerning $\mathcal{R}$ is straightforward to obtain.

Claim 1. $\mathcal{R}$ subsumes both $\mathcal{H}$ and SPP : for any pattern in $\mathcal{H}$ or SPP , the temporal network resulting from the interpretation of the pattern is expressible by some pattern in $\mathcal{R}$.

The pattern language $\mathcal{R}$ is equivalent to the baseline representation of temporal networks. As any pattern in $\mathcal{H}$ and SPP can be expressed as a temporal network, it is also expressible in $\mathcal{R}$. $\square$

6.5.2 $\mathcal{H}$ does not subsume SPP

This result is obtained by examining the layered passing tones example discussed above.

Claim 2. $\mathcal{H}$ does not subsume $\mathcal{SPP}$: there exists at least one pattern in $\mathcal{SPP}$ that has no equivalent in $\mathcal{H}$.

In $\mathcal{SPP}$, the layered passing tones pattern is expressed as follows:

$$(P14) \quad \frac{a ; b}{c ; d} ; \frac{f}{e} \rightsquigarrow \begin{array}{lll} \text{st}(a, c), & \mathbf{m}(a, b), & \mathbf{m}(c, d) \\ \text{st}(f, e), & \mathbf{m}(b, f), & \mathbf{m}(d, e) \end{array}$$

Consider the temporal network of the layered passing tones pattern. To capture the $\text{st}(a, c)$, $\mathbf{m}(a, b)$ and $\mathbf{m}(c, d)$ temporal relations, the following three $\mathcal{H}$ patterns are possible:

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \quad \begin{bmatrix} a & c \\ (a) & d \\ b & \star \end{bmatrix} \quad \begin{bmatrix} a & c \\ b & (c) \\ \star & d \end{bmatrix}$$

All of the above patterns enforce an additional temporal relation that is not enforced by the $\mathcal{SPP}$ pattern, respectively $\text{st}(b, d)$, $\text{sw}(d, a)$ and $\text{sw}(b, c)$. $\square$

6.5.3 $\mathcal{SPP}$ does not subsume $\mathcal{H}$

This result is obtained by analyzing the embellished tritone resolution pattern.

Claim 3. $\mathcal{SPP}$ does not subsume $\mathcal{H}$: there exists at least one pattern in $\mathcal{H}$ that has no equivalent in $\mathcal{SPP}$.

In $\mathcal{H}$, the embellished tritone resolution is expressed by the following pattern:

$$(P15) \quad \begin{bmatrix} a & c \\ (a) & \star \\ b & d \end{bmatrix} \rightsquigarrow \begin{array}{ll} \text{st}(a, c), & \text{st}(b, d), \\ & \mathbf{m}(a, b) \end{array}$$

The pattern enforces both $\text{st}(a, c)$ and $\text{st}(b, d)$. The only way to do that in $\mathcal{SPP}$ is to layer a, c and b, d with the “=” operator. As pattern $P15$ also enforces $\mathbf{m}(a, b)$, these two must be joined by the “;” operator:

$$\frac{a}{c}; \frac{b}{d}$$

But then, the $\mathcal{SPP}$ pattern will also enforce the temporal relation $\mathfrak{m}(c, d)$ which is clearly not enforced by pattern $P15$. $\square$

6.5.4 $\mathcal{H}$ and $\mathcal{SPP}$ subsume $\mathcal{SPPseq}$

This result is obtained by showing that every pattern in $\mathcal{SPPseq}$ is also expressible in $\mathcal{H}$ and $\mathcal{SPP}$.

Claim 4. Both $\mathcal{H}$ and $\mathcal{SPP}$ subsume $\mathcal{SPPseq}$: for every $\mathcal{SPPseq}$ pattern, there exists a pattern in $\mathcal{SPP}$ with the same network and a pattern in $\mathcal{H}$ with an equivalent network.

The first part of the proof is trivial: any pattern in $\mathcal{SPPseq}$ is also a pattern in $\mathcal{SPP}$ and has the same temporal network.

The second part of the proof proceeds by structural induction of an $\mathcal{SPPseq}$ pattern φ along the “;” operator. We show that as the sequence of layers grows, there is always a pattern in $\mathcal{H}$ that has an equivalent temporal network.

Base case: ψ In the base case, the sequence of layers contains only one layer. We show that any layer has an equivalent in $\mathcal{H}$ by induction along the “=” operator.

The base case is a single component, possibly prefixed by a “−” operator: ϵ or $-\epsilon$. Both cases are respectively covered by the following $\mathcal{H}$ patterns:

$$\left[\epsilon \right] \quad \left[(\epsilon) \right]$$

For the induction step, consider two layers ψ_1 and ψ_2 covered by two equivalent $\mathcal{H}$ patterns h, j .

$$\left[h_1 \ h_2 \ \dots \ h_n \right] \quad \left[j_1 \ j_2 \ \dots \ j_m \right]$$

By the induction hypothesis, both patterns capture the same temporal networks as the layers ψ_1 and ψ_2 . In particular, pairs of components of the following forms are covered by appropriate tokens, capturing the appropriate relations:

$\mathcal{SPPseq}$	$\mathcal{H}$	Temporal relations
ϵ_1, ϵ_2	ϵ_1, ϵ_2	$\mathbf{st}(\epsilon_1, \epsilon_2)$
$-\epsilon_1, \epsilon_2$	$(\epsilon_1), \epsilon_2$	$\mathbf{sw}(\epsilon_2, \epsilon_1)$
$\epsilon_1, -\epsilon_2$	$\epsilon_1, (\epsilon_2)$	$\mathbf{sw}(\epsilon_1, \epsilon_2)$
$-\epsilon_1, -\epsilon_2$	$(\epsilon_1), (\epsilon_2)$	$\mathbf{ov}(\epsilon_1, \epsilon_2)$

Then, joining ψ_1 and ψ_2 with “=” is covered by the $\mathcal{H}$ pattern formed by concatenating h and j :

$$\frac{\psi_1}{\psi_2} \quad \left[\begin{array}{cccccc} h_1 & h_2 & \dots & h_n & j_1 & j_2 & \dots & j_m \end{array} \right]$$

The pairs of components from ψ_1 and ψ_2 are necessarily covered the same way as above. Note that the resulting $\mathcal{H}$ pattern is well-formed as it consists of a single layer, which can take any form. Also, the pair $-\epsilon_1, -\epsilon_2$ can appear as it is the first layer of the sequence. In the following layers of a well-formed $\mathcal{SPPseq}$ pattern, at most one component can be of the form $-\epsilon$, hence that case will not be considered.

Induction step: joining φ and ψ The induction step over the “;” operator consists of joining a sequence of layers φ and a layer ψ . By hypothesis, there exists a pattern h in $\mathcal{H}$ that covers φ and a pattern j that covers ψ .

$$\left[\begin{array}{cccc} \vdots & & & \\ h_1 & h_2 & \dots & h_n \end{array} \right] \quad \left[\begin{array}{cccc} j_1 & j_2 & \dots & j_n \end{array} \right]$$

For simplicity, we assume the patterns above have the same width. If one is bigger, the smaller one is simply padded with columns filled with $\star$ symbols. The $\mathcal{H}$ pattern covering the joined $\varphi ; \psi$ is built as follows. First, h and j are reordered so that pairs of components that are aligned in $\varphi ; \psi$ appear in the same column in h and j . For example, if ϵ_1 at the end of φ and ϵ_2 in ψ are aligned, then these will appear in the corresponding column.

$$\left[\begin{array}{cccc} \vdots & & & \\ \star & h_1 & \epsilon_1 & \dots & h_n \end{array} \right] \\ \left[\begin{array}{cccc} j_1 & \star & \epsilon_2 & \dots & j_n \end{array} \right]$$

This ensures that the $\mathbf{m}$ temporal relations that the $\mathcal{SPPseq}$ pattern expresses are covered by the $\mathcal{H}$ pattern. In addition, components that are not aligned are placed in columns where a $\star$ symbol appears, as shown above. This ensures that no $\mathbf{m}$ relation is formed by these components, as is the case in $\mathcal{SPP}$ when components are not aligned with a predecessor/successor.

Then, an additional line is added between h and j . This line contains the start of any continued event in j , and the continuation of all other columns from h . This ensures that the resulting $\mathcal{H}$ pattern is well-formed. For example, supposing that ϵ_2 is affixed with the “ $-$ ” operator in ψ , we would add the following line:

$$\begin{bmatrix} \vdots \\ \star & h_1 & \epsilon_1 & \dots & h_n \end{bmatrix}$$

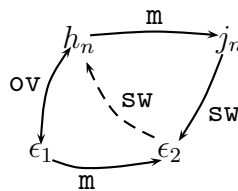
$$\begin{bmatrix} \star & (h_1) & \epsilon_2 & \dots & (h_n) \end{bmatrix}$$

$$\begin{bmatrix} j_1 & \star & (\epsilon_2) & \dots & j_n \end{bmatrix}$$

However, the added line forms temporal relations that are not expressed by the $\mathcal{SPPseq}$ pattern:

$$\dots ; \frac{\vdots}{\frac{\epsilon_1}{h_n}} ; \frac{\vdots}{\frac{-\epsilon_2}{j_n}}$$

In the $\mathcal{SPPseq}$ pattern above, the $\mathbf{sw}$ temporal relations are formed only by components of the same layer, but not between components of different layers. In the $\mathcal{H}$ pattern, however, the relation $\mathbf{sw}(\epsilon_2, h_n)$ appears. This relation can be shown to be implicit in the temporal network that the $\mathcal{SPPseq}$ pattern does express. Clearly, the temporal network of the $\mathcal{SPPseq}$ pattern will express the following relations: $\mathbf{m}(h_n, j_n)$, $\mathbf{m}(\epsilon_1, \epsilon_2)$, and $\mathbf{sw}(j_n, \epsilon_2)$. In addition, depending on the last layer of φ , the $\mathcal{SPPseq}$ pattern expresses one of the following: $\mathbf{st}(h_n, \epsilon_1)$, $\mathbf{sw}(\epsilon_1, h_n)$, $\mathbf{sw}(h_n, \epsilon_1)$, or $\mathbf{ov}(h_n, \epsilon_1)$. Without loss of generality, we can assume that $\mathbf{ov}(h_n, \epsilon_1)$ is expressed as it subsumes the other relations. This corresponds to the temporal network below.



The fact that the $\mathbf{sw}(\epsilon_2, h_n)$ relation is implicit relies on the following observations. Firstly, as j_n starts while ϵ_2 is unfolding, it follows that ϵ_2 starts before j_n . As h_n meets with j_n , we have that ϵ_2 starts before the end of h_n . Secondly, as the direct predecessor of ϵ_2 overlaps with h_n it follows that ϵ_2 starts after the beginning of h_n . Thus, we have that ϵ_2 starts before the beginning of j_n but after the beginning of h_n , and this implies that it must start while h_n is unfolding. This implicit $\mathbf{sw}(\kappa_2, h_n)$ relation is shown by a dashed line in the network above.

Similarly, all temporal relations specified by the $\mathcal{H}$ pattern as a result of the “additional” line can be shown to be implicit in the temporal network of the $\mathcal{SPPseq}$ pattern. Hence the networks are equivalent for $\varphi ; \psi$ and the $\mathcal{H}$ pattern we constructed. By induction, this demonstrates that there is an equivalent $\mathcal{H}$ pattern for any sequence of layers in $\mathcal{SPPseq}$. $\square$

We illustrate the argument above for the two voice case. The proof proceeds by structural induction over the “;” operator (i.e. the claim holds as the sequence grows). The base cases are:

$$\frac{a}{c} \quad \frac{-a}{c} \quad \frac{a}{-c} \quad \frac{-a}{-c}$$

These are clearly covered by the following $\mathcal{H}$ patterns:

$$\left[\begin{array}{cc} a & c \end{array} \right] \quad \left[\begin{array}{cc} (a) & c \end{array} \right] \quad \left[\begin{array}{cc} a & (c) \end{array} \right] \quad \left[\begin{array}{cc} (a) & (c) \end{array} \right]$$

Now, suppose there exists a pattern $h \in \mathcal{H}$ that covers the $\mathcal{SPPseq}$ pattern φ . The induction cases are as follows (the case with two modified events $-\epsilon$ does not appear; by definition of $\mathcal{SPPseq}$, this is only allowed in the first layer):

$$\varphi ; \frac{a}{c} \quad \varphi ; \frac{-a}{c} \quad \varphi ; \frac{a}{-c}$$

Suppose h has n lines. The induction cases are covered by:

$$\begin{array}{ccc} h & \left[\begin{array}{cc} \vdots & \\ h_{n1} & h_{n2} \end{array} \right] & \left[\begin{array}{cc} \vdots & \\ h_{n1} & h_{n2} \end{array} \right] \\ \cdot & \cdot & \cdot \\ \left[\begin{array}{cc} a & c \end{array} \right] & \left[\begin{array}{cc} a & (h_{n2}) \\ (a) & c \end{array} \right] & \left[\begin{array}{cc} (h_{n1}) & c \\ a & (c) \end{array} \right] \end{array}$$

The last two cases enforce an extra temporal relation (respectively $\mathbf{sw}(a, h_{n2})$ and $\mathbf{sw}(c, h_{n1})$) that the $\mathcal{SPPseq}$ pattern does not enforce. However, that relation can be inferred by the temporal relations that the $\mathcal{SPPseq}$ pattern does enforce. That is, whenever the relations $\mathbf{m}(a, b)$, $\mathbf{m}(d, e)$, $\mathbf{sw}(b, e)$ and $\mathbf{ov}(a, d)$ are present, then $\mathbf{sw}(e, a)$ can be inferred.

6.6 Summary

This chapter formally compared the three approaches to polyphonic patterns discussed in detail in Chapter 3: relational patterns, Humdrum, and $\mathcal{SPP}$. In terms of the temporal network they can express, relational patterns were shown to be the most expressive. Humdrum and $\mathcal{SPP}$ can be considered as subsets of relational patterns, with a non-trivial intersection that this chapter characterized as $\mathcal{SPPseq}$: a restriction of $\mathcal{SPP}$ to sequences of layers. Having a restricted expressiveness is advantageous in terms of data mining, as the space of patterns to explore is reduced. Furthermore, $\mathcal{SPPseq}$ is expressive enough to capture the parallel fifth and suspension patterns discussed throughout this dissertation. The possibility of using $\mathcal{SPP}$ or $\mathcal{SPPseq}$ for music data mining is discussed in the next chapter. In addition, Humdrum and $\mathcal{SPP}$ are compared in terms of ease of use and efficiency.

Chapter 7

Discussion

This chapter revisits the requirements for a polyphonic pattern language elaborated in Chapter 2. We further establish how *SPP* fully satisfies the requirements and, in particular, how it possesses some advantages over relational patterns and Humdrum: clearer syntax and semantics, more efficient, more amenable to a user-friendly tool, and more conducive to music data mining. Some limitations and caveats are also discussed.

7.1 From requirements to *SPP*

This dissertation developed the Structured Polyphonic Patterns approach around a set of requirements. These can be seen as design principles that motivate the particular way in which *SPP* is formulated and presented.

Requirement 1: Fully support n-part polyphony The main objective of *SPP* was to support polyphonic music. In particular, this meant representing the kinds of complex polyphonic patterns that textbooks discuss, e.g. the parallel fifth and suspension patterns. These patterns typically involve well-defined parts. Hence, support for polyphonic patterns requires the precise representation of multiple parts. As these patterns also involve precise temporal relations, the multiple parts of a polyphonic piece must also be represented in a general way, without assuming a particular structure, as is the case of approaches restricted to homophony. The *SPP* method accomplishes this by representing the source as an unstructured collection of musical events. At the same time, events are required to contain the attributes

that make it possible to define meaningful polyphonic patterns: onset, offset, and voice label. Similarly, *SPP* patterns can refer to voices. However, these voices are not *dynamic*: they cannot capture the case where voices appear and disappear as the piece unfolds. This limitation is discussed further in Section 7.2

Requirement 2: Expressive and precise This requirement relates to the expression of temporal and musical relations. *SPP* directly supports four temporal relations: **m** (meet), **st** (start together), **sw** (start while), and **ov** (overlap). The relation **m** is expressed through the sequencing operator of *SPP*, and by aligning components in the adjacent layers of a sequence of layers. The three vertical relations **st**, **sw** and **ov** are expressed through the concept of layer, with different combinations of the special onset modification operator “—”. Musical relations are accommodated through a feature definition mechanism. A feature can refer to a voice, which enables the expression of precise relations. The same musical relations can be defined in relational patterns, Humdrum, and *SPP*. For relational patterns, the relations need to be implemented directly in the underlying programming language, e.g. Prolog. Humdrum has dedicated tools, but these do not necessarily cover every possible relation. For other relations, the user must program his or her own specialized command-line tools.

Relational patterns and Humdrum can, however, represent musical relations with greater precision. In *SPP*, relations are encoded as features that forget with which event the relation is formed, a strategy referred to as positionalization (Kramer, 2000). Only the voice of that event is recorded. In contrast, Humdrum can keep a precise record of the two events that form the relation: the events always appear on the same line as the encoding of the relation. In relational patterns, the events are explicitly recorded through variables. In practice, this does not seem to be a significant limitation of *SPP*. By distinguishing features formed over different temporal relations as is done in Chapter 5, the necessary precision is obtained. In addition, the positionalization of relations reduces the space of patterns and makes *SPP* more conducive to data mining. One possible drawback is that positionalization can reduce the readability of patterns, as the reader sometimes needs to infer the precise meaning of a feature from its name and the context in which it appears. For example, consider the following simplification of the *SPP* suspension pattern developed in Chapter 5:

$$\frac{-\{\}_x^*}{-\{\}_y} ; \frac{\{\}_x}{-\{\text{sw_cons}(x) : F\}_y}$$

The `sw_cons` feature encodes a relation between the bottom right component (which contains the feature) and the top left component (distinguished by a `*` symbol above). This has to be inferred from the name of the feature, which explicitly states that a musical relation over an `sw` temporal relation is encoded, and the context in which the feature appears: in this case, the bottom right component extends past the start time of its layer and, consequently, must start before the end of the preceding layer, hence starting while the distinguished top left component unfolds.

Requirement 3: Clear syntax and semantics The syntax and semantics of *SPP* is defined in Chapter 4 and illustrated in Chapter 5. The syntax revolves around a few simple constructs: sets of features denote musical events and sequencing and layering operators are used to build patterns. It has the advantage that it embodies the structure of a pattern: sequences are displayed as horizontal lines and layers as vertically aligned components. The syntax of Humdrum is similar, but with time flowing from top to bottom: sequences are displayed vertically and layers horizontally. By comparison, a relational pattern is always a list of literals. Even with simple patterns, it can be difficult to understand the structure of such a pattern: whether it consists of sequences, layers, or a mix of the two.

The semantics of *SPP* patterns is also based on the idea of sequence and layer, and directly follows from the structure of a pattern. To denote events that start before the beginning of a layer, the “`-`” evokes the extension of an event backward in time. Section 7.3 discusses some choices that were made when formulating the semantics of *SPP*. The semantics of relational patterns is also clear, with every instance satisfying every literal of the pattern. By contrast, the semantics of Humdrum patterns can be difficult to grasp, as it depends on prior transformations of the source.

Requirement 4: Flexible The flexibility of *SPP* comes from the feature definition rules developed in Chapter 4 and illustrated in Chapter 5. These ensure that new queries can be accommodated easily.

In addition, as Chapter 5 demonstrates, it is easy to develop an *SPP* query in a modular way. First, by reusing the same user-defined features in many queries. Also, variations on a query are easily achieved, by using variables instead of values

or by adding features in order to obtain more details on the context in which instances occur. The structure of a pattern can also be changed easily, by adding new components, or possibly reordering the components. It is much more difficult to do this using relational patterns or Humdrum. Finally, it is also easy to create a new pattern by joining two existing patterns via the “;” or “=” operators; the semantics is modular: the meaning of subpatterns does not change as they are joined.

The flexibility that *SPP* offers over Humdrum could prove particularly useful for musicological research. For example, if a set of related patterns needs to be constructed, it would be less time-consuming to do it in *SPP*. Section 7.7 discusses a case where that would be relevant to musicological research.

Requirement 5: Easy to use and efficient With a clear syntax and semantics, it is easy to develop queries in *SPP*. To make this more concrete, we develop a mock graphical interface for *SPP* in Section 7.5. Regarding efficiency, the *SPP* matching algorithm is more efficient than Humdrum. We have developed multiple implementations of the algorithm, using the Ocaml (Leroy et al., 2003) and Prolog programming languages. The Prolog implementation is more efficient for matching only, but it is more efficient to use the Ocaml implementation to apply feature definition rules. This is discussed in detail in Section 7.4.

Requirement 6: Conducive to music data mining *SPP* is inspired by the feature set representation of pattern components, which in turned was inspired by the viewpoint method of Conklin (2002). The feature set representation was introduced specifically for music data mining (Conklin and Bergeron, 2008). Moreover, the fact that *SPP* patterns are structured around syntactic constructs provides a convenient way to explore the pattern space. This is discussed further in Section 7.6.

7.2 Beyond n-part polyphony

The queries presented in this dissertation are mainly relevant for pieces of music where a fixed number of static voices is used. It is possible to extend *SPP* to dynamically voiced music. This requires giving some structure to the domain of voice names. For example, such a structure could be a graph where nodes represent voices and edges represent how voices are divided into sub-voices, and possibly merged back to high-level voices. Consequently, the definition of how components are matched

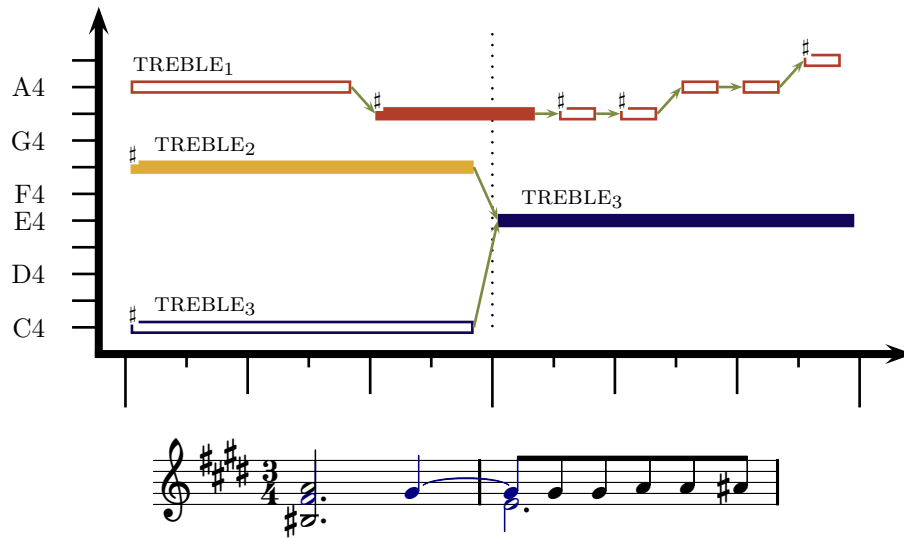


Figure 7.1: Suspension pattern in Chopin Waltz Op.64 Nr.2, bars 9 and 10

would need to be extended. Rather than requiring that identical voice labels are mapped to the same voice name, it would be required that they are mapped to voice names that are linked by a path in the graph of voices.

For example, consider an instance of the suspension pattern displayed in Figure 7.1. The piano-roll has been annotated with voice names and arrows indicating one possible underlying graph of voices. Notice how the voice `treble2` is merged with the voice `treble3`. As the suspension pattern occurs over that merge – the first note is in voice `treble2` and the second in `treble3` – the suspension instance would not be captured with the current semantics of *SPP*.

An extension to the semantics would require us to extend the matching algorithm as well. When joining two instances, the condition for joining two voice variable assignments would be modified. Instead of requiring that any two assignments of the same voice variable are equal, it would be required that they are consistent with the graph of voices. For example, one assignment could be `treble2` and the other `treble3`. As these are consistent, the resulting assignment would be set to the higher-level voice `treble3`.

7.3 Motivating *SPP* semantics

When developing *SPP*, many issues regarding the syntax and semantics were explored. This section summarizes the most important choices. The “design” princi-

ples behind those choices were as follows:

- Keep expressiveness well adjusted for polyphonic music (i.e. avoid unnecessary expressive power, as this renders the language less suitable for data mining)
- Keep syntax and semantics as clear as possible
- Use as few operators as possible: keep the language simple

Contiguous patterns There is no mechanism in $\mathcal{SPP}$ to express relations between events that i) do not meet, ii) are not overlapping or iii) are not related by a chain of events that meet or overlap. One such mechanism, for example, would be to include a Kleene star operator (similar to regular expressions), e.g. $\epsilon_1 ; \epsilon_2^* ; \epsilon_3$ would match an event e_1 instantiating ϵ_1 , then a sequence of events – all instances of ϵ_2 – then an event instantiating ϵ_3 . One polyphonic pattern that might require such non-local relations is the *pedal point*. A pedal point is a note that unfolds while an undetermined number of notes is unfolding in another voice. It can be approximated in $\mathcal{SPP}$ by matching only for the pedal point, and implicitly matching the other notes through features.

Conjunctive components Currently, all the features of a component need to be instantiated, i.e. a component is a conjunction of features. One can easily imagine a more complex definition of components inspired by propositional logic. In such a scheme, a component could also express implication and disjunction. The following component, for example, would match a note with a pitch of either C or G and that has a duration of 2 when it occurs on a strong beat and any duration otherwise:

$$\{(\text{pitch} : \text{C} \vee \text{pitch} : \text{G}) \wedge (\text{ml} : 0 \Rightarrow \text{duration} : 2)\}$$

Again, it is unclear if such expressive power is necessary and supporting it would make the language and its semantics more complex. One possible application of propositional logic could be the precise definition of consonance in a four voice texture (Piston, 1941, chap. 2). It requires us to classify the interval of a perfect fourth as a consonance only if the interval of a fifth or an octave occurs below it. Such a constraint could be precisely defined with feature implication. Instead, this dissertation uses a somewhat simplified definition of consonance where the interval of a perfect fourth is always classified as a dissonance.

Onset modification One can imagine two alternative interpretations of the onset modification operator “−”. Consider the pattern below.

$$\frac{a}{\{ \}}; \frac{-b}{\{ \}}$$

In *SPP*, the meaning of “−” is to modify the onset of the event matched by *b*, namely by shifting it back in time. One could also imagine the “−” operator acting as a tie of *a* and *b*. In that case, *a* and *b* would match only one event. That would be similar to the continuation mechanism of Humdrum. Consequently, it would have a similar expressiveness to that of Humdrum in terms of temporal relations. As Chapter 6 demonstrates, this is not superior to the expressiveness of *SPP*. In addition, it would make the semantics and algorithm more intricate and less directly tied to the concept of layer, hence possibly less intuitive.

7.4 Pattern matching efficiency

The implementations of Humdrum and *SPP* differ in ways that have an impact on efficiency. The central difference is how the source is scanned and how instances are recorded. In Humdrum, the source is scanned in a linear fashion, from start to finish. Instances are simply printed on screen as the pattern is being matched. In *SPP*, the algorithm proceeds in two phases. In a first phase, the source is scanned in a linear fashion and the instances of the components of a pattern are computed and stored. In a second phase, the instances of the whole pattern are computed by joining the instance lists of its subpatterns. This can be done efficiently by ordering instances by their onset times as explained in Section 4.7.

Another difference is how voice combinations are handled. In Humdrum, there is no abstract representation of how a pattern can match many different combinations of voices. Consider for example a two-voice pattern and a four-part source. To match such a pattern, one must first extract every two-part combinations from the source and create artificial two-part sources (including permutations, e.g. where the lowest part appears first, and then where the lowest part appears second). Then, the Humdrum pattern matching command can be used on every artificial source. It follows that every possible voice combinations are explored, regardless of the presence of instances.

In *SPP*, the voice combinations are explored as the instance lists are being joined. For example, when two components are joined by a layer operator “=”, the possible assignations of their voice variables are merged and tested for consistency. This is done only for instances that have already been tested for their temporal validity (e.g. having the same onset if the layer enforces the *st* temporal relation). The advantage of this approach is that voice combinations are explored only when necessary. For example, if a particular voice combination does not contain instances, it will not be explored.

The principal motivation for using instance lists in *SPP* is to support pattern discovery. In such a scheme, new patterns are being created by joining two previously processed patterns. When a new pattern is being created, its instances need to be computed. With the Humdrum approach, one would need to re-scan the source for every new pattern. With the *SPP* algorithm, one can reuse the instance lists of the previously processed patterns and quickly obtain the instances of the new pattern.

In practice, for typical patterns and sources, the differences discussed here should not have a significant impact. To evaluate this claim, we compare *SPP* and Humdrum in two ways. First, we look at various ways the source can grow: increasing number of events, increasing number of voices, and increasing number of instances. Then, we look at various ways the pattern can grow: number of features, sequence length, and layer size.

Growing sources We consider two scenarios: i) feature definition rules need to be applied before executing pattern matching and ii) only pattern matching need to be executed. The first scenario occurs whenever the user has defined some features that are not in the source. This is the most likely scenario for musicological queries, where the query is being developed from scratch. The second scenario would occur, for example, if a source or corpus is already saturated with musical features and published as such. In that scenario, the query is executed without applying any feature definition rule.

In the first scenario, we compare Humdrum to three different implementations of *SPP*: i) an Ocaml implementation of the algorithm presented in Chapter 4, a Prolog implementation of the semantics presented in Chapter 4 and iii) a hybrid implementation where feature definition rules are applied in Ocaml and pattern matching is done in Prolog.

The comparisons are based on experiments executed with the suspension pattern

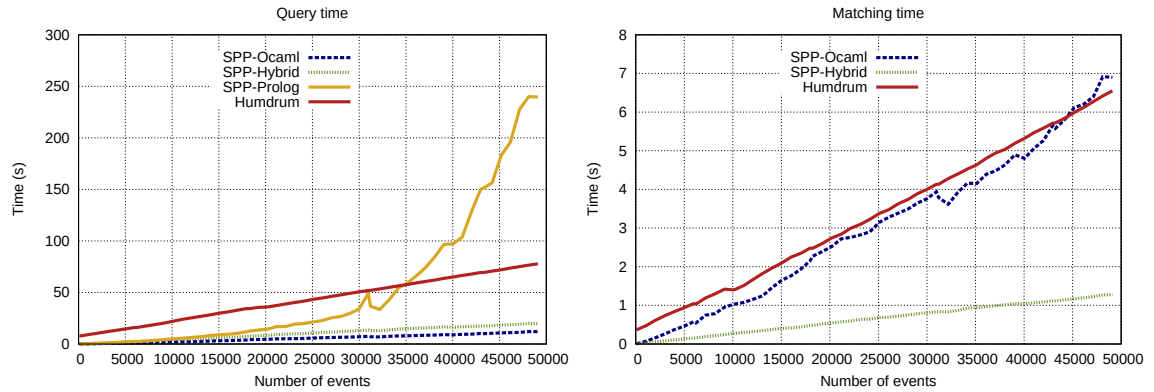


Figure 7.2: Comparing Humdrum and SPP against sources with an increasing number of events. Results are averaged over 1000 sources with 10 voices and an instance density of 0.1

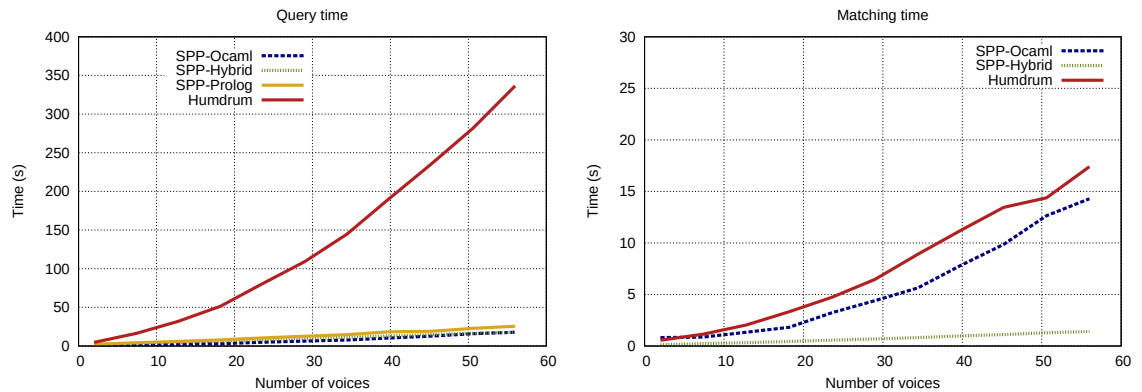


Figure 7.3: Comparing Humdrum and SPP against sources with an increasing number of voices. Results are averaged over 500 sources with 10000 events and 1000 instances

on artificial sources. The sources were created by recombining excerpts selected from Bach chorale harmonizations. Figure 7.2 shows the results for sources with an increasing number of events. The number of voices is kept constant to 10 voices. Also, the density of instances (the number of instances divided by the number of events) is kept constant to 0.1. In the first scenario, the Ocaml implementation is the most efficient. In the second scenario, the hybrid implementation is more efficient.

Figure 7.3 shows that Humdrum is far less robust with increasing voices. Figure 7.4 shows that all approaches are not significantly affected by an increase in the number of instances, with the exception of the SPP -Prolog implementation, where both the computation of feature definition rules and the pattern matching is done in Prolog.

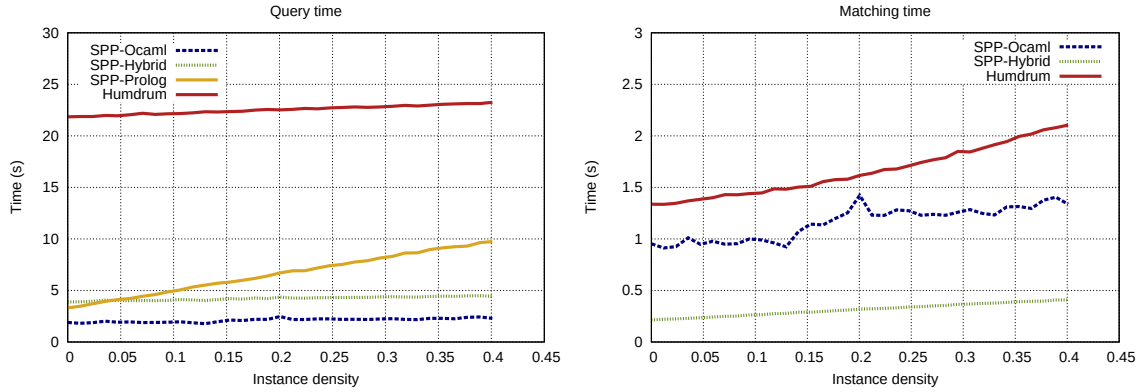


Figure 7.4: Comparing Humdrum and $\mathcal{SPP}$ against sources with an increasing instance density. Results are averaged over 500 sources with 10 voices and 10000 events

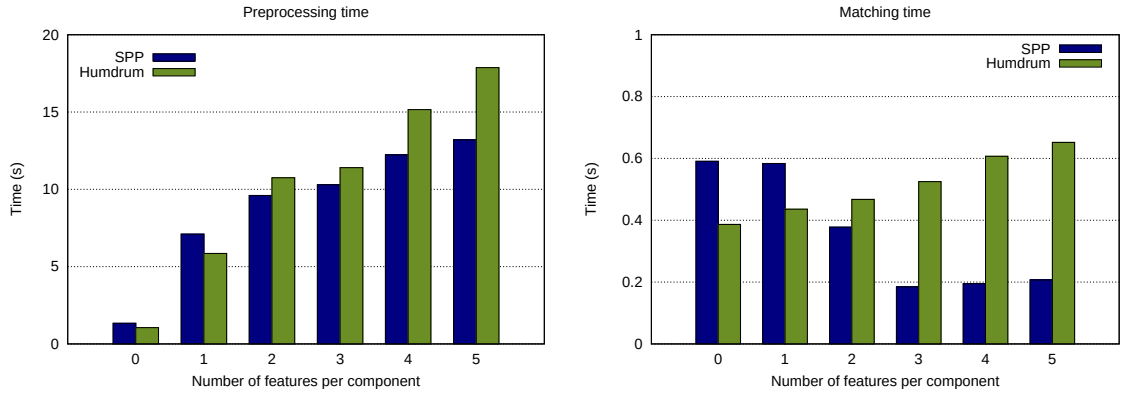


Figure 7.5: Comparing Humdrum and $\mathcal{SPP}$ preprocessing and matching times for a counterpoint query with varying number of features. Results are averaged over 10 sources with 4 voices and 10000 events

Growing patterns Here, we assume the first scenario and look at both preprocessing time and matching time. Experiments concern a counterpoint pattern similar to the one developed in Chapter 5. We compare the efficiency of Humdrum and the hybrid Ocaml/Prolog implementation of $\mathcal{SPP}$ against patterns with varying properties: growing number of features, growing sequence length, and growing layer size. Figure 7.5 shows the results for a growing number of features. Notice how $\mathcal{SPP}$ matching time can decrease as features are added. This is because each new feature adds a constraint and decreases the number of instances. The joining of instance lists used in the $\mathcal{SPP}$ algorithm hence becomes quicker.

Figure 7.6 shows the result for patterns of increasing sequence length. This does not affect preprocessing time. Also, Humdrum matching time is not affected significantly, while there is a slight decrease in $\mathcal{SPP}$ matching time. Again, this is

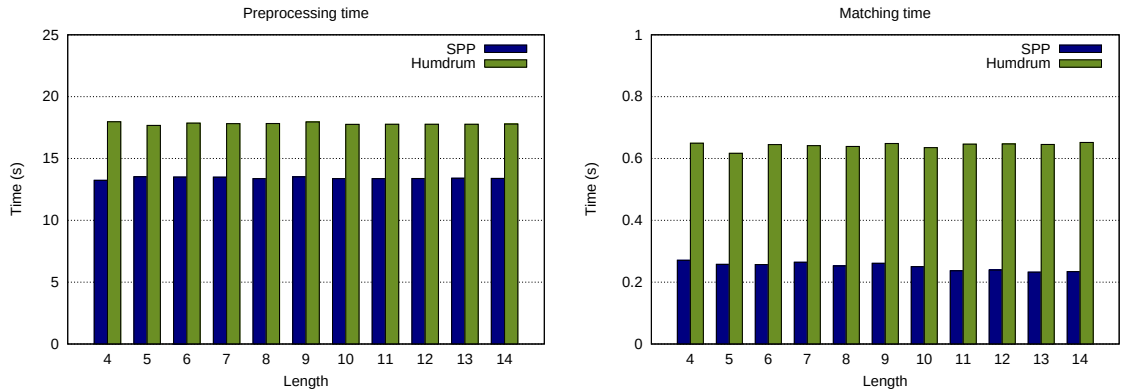


Figure 7.6: Comparing Humdrum and *SPP* preprocessing and matching times for a counterpoint query of varying sequence length. Results are averaged over 10 sources with 4 voices and 10000 events

due to the fact that *SPP* matching uses instance lists. The difference is however negligible.

Figure 7.7 shows the results for patterns with layers of increasing size. Both approaches suffer from combinatorial explosion (note that time is shown according to a logarithmic scale). *SPP* is slightly more robust, allowing for up to seven components per layer with reasonable matching time, while Humdrum pattern matching becomes impractical for patterns with more than four components per layer. Also, Humdrum becomes inefficient for preprocessing and matching, while *SPP* preprocessing remains efficient. In *SPP*, preprocessing depends only on the properties of the feature definition rules. It is independent of the pattern to match. By contrast, voice exploration in Humdrum has to be done for preprocessing also, whenever the pattern refers to more than one voice. In addition, as discussed earlier, Humdrum always displays “the worst case” with respect to voice exploration, but *SPP* can alleviate some of the combinatorial explosion: when instances are found quickly, the collections of instances that the algorithm joins get pruned early on and fewer instance joins have to be explored. Still, the only way to guarantee efficient pattern matching in Humdrum and *SPP* is to keep the layers small. In practice, this is not a significant limitation as musical patterns tend to refer to a small number of parts, often two or four as shown in Chapter 5.

A note on Prolog implementations It is important to note that the Prolog implementation is efficient because *SPP* patterns were interpreted in Prolog in a way that ensured a good ordering of the literals. When using Prolog for arbitrary relational patterns, it is only possible to obtain efficient matching if such a good

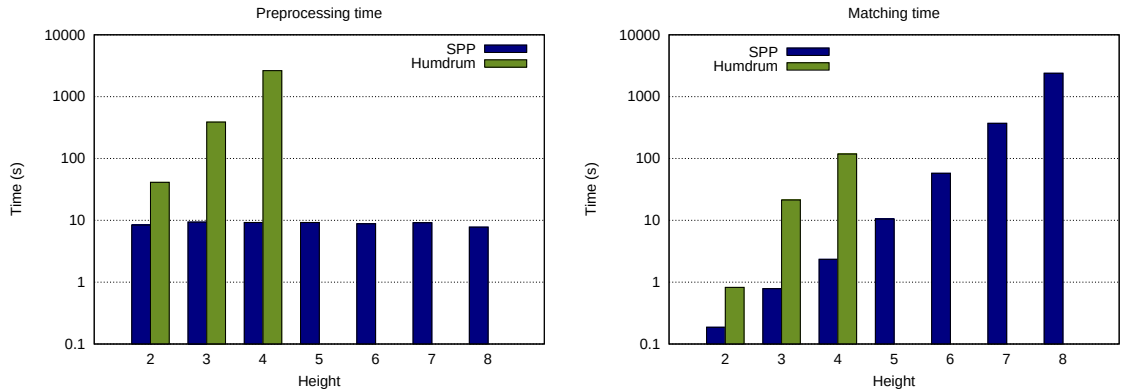


Figure 7.7: Comparing Humdrum and *SPP* preprocessing and matching times for a counterpoint query of varying layer size. Results are averaged over 10 sources with 8 voices and 5000 events

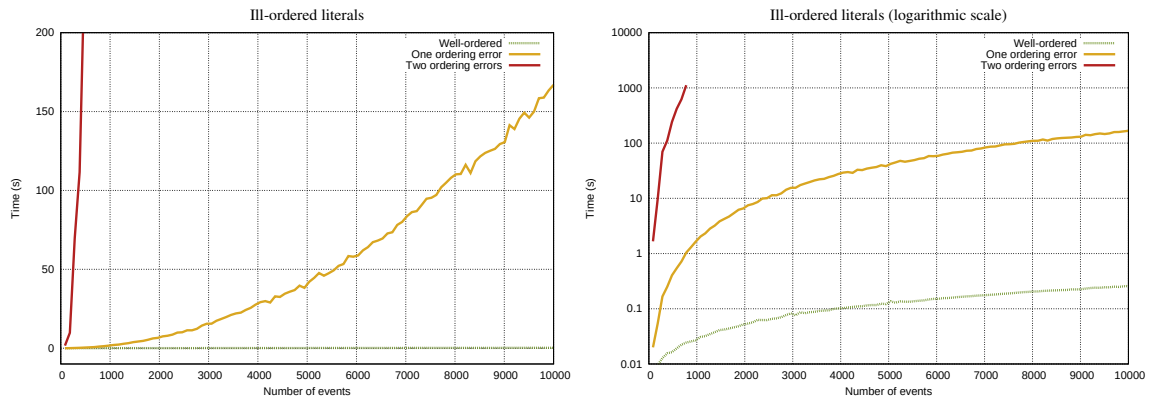
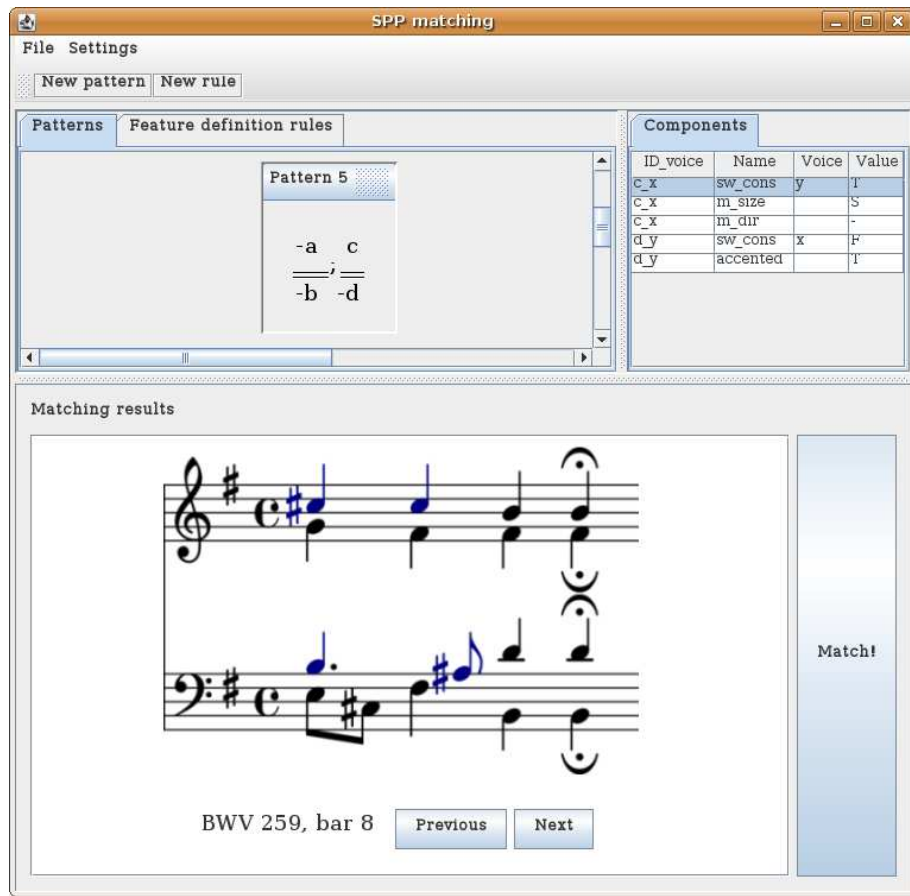


Figure 7.8: The effect of ill-ordered literals in a Prolog query. Results are averaged over 1000 sources with 10 voices and an instance density of 0.1

ordering exists. In particular, when writing the query directly in Prolog, one must be very careful not to begin a query with literals that refer to distinct event variables. Such an ordering will cause the source to be scanned in a quadratic fashion, while three ill-ordered literals will cause a cubic execution time, and so forth. Figure 7.8 shows how inefficient this can be in practice.

Interpretation *SPP* is more robust than Humdrum with respect to most experimental conditions explored here. The algorithm we presented in Chapter 4 is comparable in speed to the slightly more efficient Prolog implementation. The choice of an implementation depends on the needs at hand. The biggest advantage of the algorithm of Chapter 4 is that it can be reused for pattern discovery, and this is a further contribution of the current work.

The *SPP* language could also be construed as a useful intermediary pattern

Figure 7.9: The suspension pattern in a mock *SPP* GUI

representation. The direct use of Prolog for music pattern matching can prove cumbersome and requires some care when designing the query to avoid a bad ordering of literals that can cause serious efficiency problems. The Prolog implementation of *SPP* is an automatic translator of *SPP* patterns into Prolog and can help mitigate those problems. Appendix E presents this translation.

7.5 Mock *SPP* GUI

Developing a query in *SPP* is facilitated by a clear syntax and semantics and by the fact that it can be done in multiple steps. With a good graphical interface, this can be made intuitive. Figure 7.9 shows an example of such an interface for the suspension pattern. The structure of the pattern is expressed graphically, while the features are written down as separate fields.

With a clear syntax, the manipulation of $\mathcal{SPP}$ patterns is made easy. Figure 7.10 on page 139 shows one possible sequence of simple operations that will yield the structure of the suspension pattern. Figure 7.11 on page 140 shows a sequence of operations aimed at transforming the suspension pattern into a block chord pattern.

7.6 Suitability of $\mathcal{SPP}$ for data mining

A crucial idea in data mining is the efficient enumeration of the pattern space. One solution is to define a pattern language such that the space is small and can be enumerated exhaustively. However, as discussed in Chapter 3, this can prove too strong a restriction on expressiveness to satisfy the requirements for polyphonic patterns that this dissertation set out to fulfil. Another approach is to use a more expressive language and to restrict the enumeration to a subset of patterns, typically those with a high level of interest (according to some measure), or high frequency. As discussed in Chapter 3, Humdrum (Huron, 1999) is expressive enough for polyphonic queries. However, it does not provide a mechanism to enumerate the space of patterns. The formal syntax and semantics of $\mathcal{SPP}$ enable the definition of such a mechanism.

Component subsumption The first step is to define how to compare patterns directly from their syntax. In particular, we want to quickly determine if a component is more general than another. This enables the enumeration of the pattern space in general-to-specific manner. When interested in patterns that occur frequently, as is typically the case, this allows to cut whole regions of the search space as specializations of a component cannot be more frequent than some previously explored component. The property of a component being more general than another, or *subsumption*, is easy to define on $\mathcal{SPP}$ components. For example, the following subsumption holds under renaming of x to y and assignment of X to 60:

$$(E13) \quad \{\text{pitch} : X\}_x \text{ subsumes } \{\text{pitch} : 60, \text{duration} : 24\}_y$$

Enumerating frequent components Once component subsumption is defined, efficient enumeration of frequent components is possible. As variables are involved, this would constitute an extension of the technique developed by Conklin and Bergeron (2008).

Enumerating frequent patterns The enumeration of frequent components is a required first step for an efficient exploration of the pattern space. Once this is done, the space of frequent patterns can be explored by joining components with the “,” and “=” operators. Initially, the component would be layered with the “–” operator, as this is more general.

$$(E14) \quad \begin{array}{ccc} \frac{-a}{=} & \text{subsumes} & \frac{-a}{=} \\ -b & & b \end{array}$$

$$\begin{array}{ccc} \frac{-a}{=} & \text{subsumes} & \frac{a}{=} \\ -b & & b \end{array}$$

Bigger patterns can be explored by joining smaller patterns that were discovered to be frequent. Every time two patterns would be joined, their instances would be joined in the same efficient way that was developed for pattern matching in Section 4.7.

As this section sketches, the research proposed in this dissertation is clearly a constructive first step towards data mining of polyphonic music. In addition, Chapter 6 introduces SPP_{seq} , a restriction of SPP to sequences of layers that is expressive enough to capture most of the patterns presented in this dissertation. Such as restriction makes the approach even more conducive to data mining as the pattern space of the restricted language is smaller to explore. This is a significant contribution as polyphonic pattern discovery is a topic that has yet to be explored for languages satisfying the requirements of Chapter 2.

There have been some attempts, however, to use inductive logic programming (ILP) engines to discover musical rules from existing musical material (Ramirez, 2003; Pompe et al., 1996). This differs from the pattern discovery approach, as the target patterns are given to the learning engine, which searches for a succinct description of the contexts in which these patterns occur.

There are also some ILP tools that are tailored for pattern discovery. One successful application of ILP to music data mining concerns the discovery of chord sequence patterns (Anglade and Dixon, 2008). By using the chord sequence abstraction, musically significant patterns can be discovered in large polyphonic corpora. The abstraction significantly reduces the pattern space as i) only sequences of components are allowed and ii) the components contain very few features, essentially

chord names. There are a few issues with such an approach, however, as the corpus needs to be segmented and labelled with chord names prior to pattern discovery. It is still an open problem to reliably find such a segmentation and labelling (Anglade and Dixon, 2008, use two corpora where this has been done manually).

In the context of detailed polyphonic patterns, we have experimented with the ILP tool Warmr (King et al., 2001), and concluded that it is currently impractical to apply this tool for polyphonic pattern discovery. The problems stem from the fact that logic programming clauses are too expressive for the task at hand. To circumvent the issue, one must define a search bias. For example, features are defined by requiring that at least one note of a relation corresponds to a note that was already introduced in previous exploration of the pattern space (this limits the search to contiguous patterns, a reasonable restriction). There are, however, limitations to the kind of bias one can define with tools like Warmr.

In addition, there are efficiency problems that are very difficult to avoid when using the bias language of Warmr. For example, there is no obvious way to instruct Warmr to join previously discovered patterns rather than grow them by adding literals to their corresponding clauses. This creates a search space that is too fine-grained for our needs. In a homophonic texture, for example, a pattern representing two successive layers will be built in an exponential number of ways (first specifying the two upper notes satisfy m , then specializing so the upper and the middle note satisfy m and so on in every possible way). Similarly, there is no obvious way to first search for frequent components and then search for frequent structures (e.g. sequence of components). This means that the search space of components (which consists of every possible combination of features) is interleaved with the search space of structures (which consists of every possible combination of components). These problems could be avoided in $\mathcal{SPP}$, for example, by first discovering frequent components, then joining them in sequences and layers and then joining these to form sequences of layers or other mixtures of sequences and layers.

7.7 Suitability of $\mathcal{SPP}$ for musicology

As discussed in Chapter 1, the notion of musical pattern is usually simpler and more formal in computational musicology than traditional musicology. In the latter, patterns are illustrated by examples, and the musical expertise of the reader is called upon to understand how the examples are related. Although intuitively related, the

examples can be significantly different: added notes, different melodies, etc. The works of Gjerdingen (2007) and Schubert (2007) are good examples of this viewpoint on musical patterns.

Even if $\mathcal{SPP}$ was designed with computational musicology in mind, it could be useful for that type of research. For example, Gjerdingen (2007) presents a rising fourths canon pattern where two voices alternate the melodic motion of a rising fourth. Numerous variations and elaborations of the pattern are explored. $\mathcal{SPP}$ could be used to isolate places in a large corpus where such variations might occur, for example by using a set of very general patterns. The following pattern is one possible member of such a set:

$$\begin{array}{lcl} & \frac{a}{b} & ; \frac{c}{d} ; \frac{e}{f} \\ (P16) \quad \bar{a} & = & \{\text{st_interval}(y) : I1\}_x \\ & \bar{b} & = \{\}_y \\ & \bar{c} & = \{\text{st_interval}(y) : I2, \text{m_interval} : P4, \text{m_dir} : +\}_x \\ & \bar{d} & = \{\}_y \\ & \bar{e} & = \{\text{st_interval}(y) : I3, \text{m_interval} : D, \text{m_dir} : -\}_x \\ & \bar{f} & = \{\}_y \end{array}$$

The top sequence captures the rising melodic interval of a fourth, followed by any falling melodic interval. The variable D captures the falling interval, which can be either a major or minor third: M3 or m3. Similarly, variables are used to capture different sequences of harmonic intervals (and again, the third can be major or minor). This is useful as Gjerdingen (2007) describes two version of the pattern, one with that starts with the intervals P1, M3 (m3), P1 and one that starts with the intervals M3 (m3), P5, M3 (m3). The second voice is left blank in the pattern above, but it follows from the harmonic intervals that it must also display the melodic motion of a rising fourth. Figure 7.12 on page 141 shows instances of both versions of the pattern in Bach chorale harmonizations. The instances correspond to the case where there is no intervening notes in any of the two voices. Gjerdingen (2007) also presents elaborations where, for example, one voice exhibits denser melodic material.

As $\mathcal{SPP}$ is not designed to search for elaborations or variations on a theme, one would need to write patterns specifically to capture every type of expected elaborations. This might prove labor-intensive and has limitations as it is difficult

to anticipate every possible type of elaboration. However, such an approach can still prove very useful when very large databases are to be searched. In addition, one can imagine an extension of $\mathcal{SPP}$ that handles approximate matching and ranking of the results by relevance. The result would be a list of potential instances of the rising fourths canon pattern for the musicologist to inspect and classify. Such a semi-automated approach to pattern matching can provide a way for the musicologist to explore more patterns and also a much bigger number of musical pieces.

It is less obvious how pattern matching could have contributed to the work of Schubert (2007), as one of the crucial aspects of this work is that patterns are very specific to a single piece and the contribution lies in the discovery of the patterns.

For example, Schubert discusses the following pattern, partially capture in $\mathcal{SPP}$ here, as reoccurring frequently in a particular piece, and hence influencing its structure:

$$(P17) \quad \frac{\{\}_x}{-\{\}_y} ; \frac{\{\mathbf{m_dir} : -\}_x}{-\{\mathbf{m_dir} : +\}_y} ; \frac{\{\mathbf{m_dir} : -\}_x}{\{\mathbf{m_dir} : +\}_y}$$

It is a simple counterpoint pattern where two voices interact in a precise manner: voice x exhibits falling melodic motion while voice y exhibits rising melodic motion. In addition, the voices exhibit very specific temporal relations, as the first two notes of voice y start while the corresponding notes in voice x are unfolding.

Figure 7.13 on page 141 shows instances of the pattern in Bach chorales harmonizations. The pattern only appears six times in the corpus and never twice in the same piece. This in itself an interesting result as it confirms that the Chorale corpus does not contain a piece that uses the pattern discovered by Schubert (2007) as a structuring element. In addition, pattern matching could prove useful in revealing the presence of such a piece in another corpus, hence establishing strong structural similarity between two pieces. Perhaps that an $\mathcal{SPP}$ encoding of the patterns presented in Schubert (2007) would prove fruitful in that respect. To fully support his approach, however, $\mathcal{SPP}$ would need to support discovery of new counterpoint patterns. This topic is still under investigation, but early results are promising (Conklin and Bergeron, 2010).

7.8 Summary

This chapter discussed in detail the requirements stated in the introduction and elaborated in Chapter 2. The limits of $\mathcal{SP}\mathcal{P}$ were also discussed. The next chapter concludes this dissertation by summarizing the contributions it has made. Prospects for future research are also discussed.

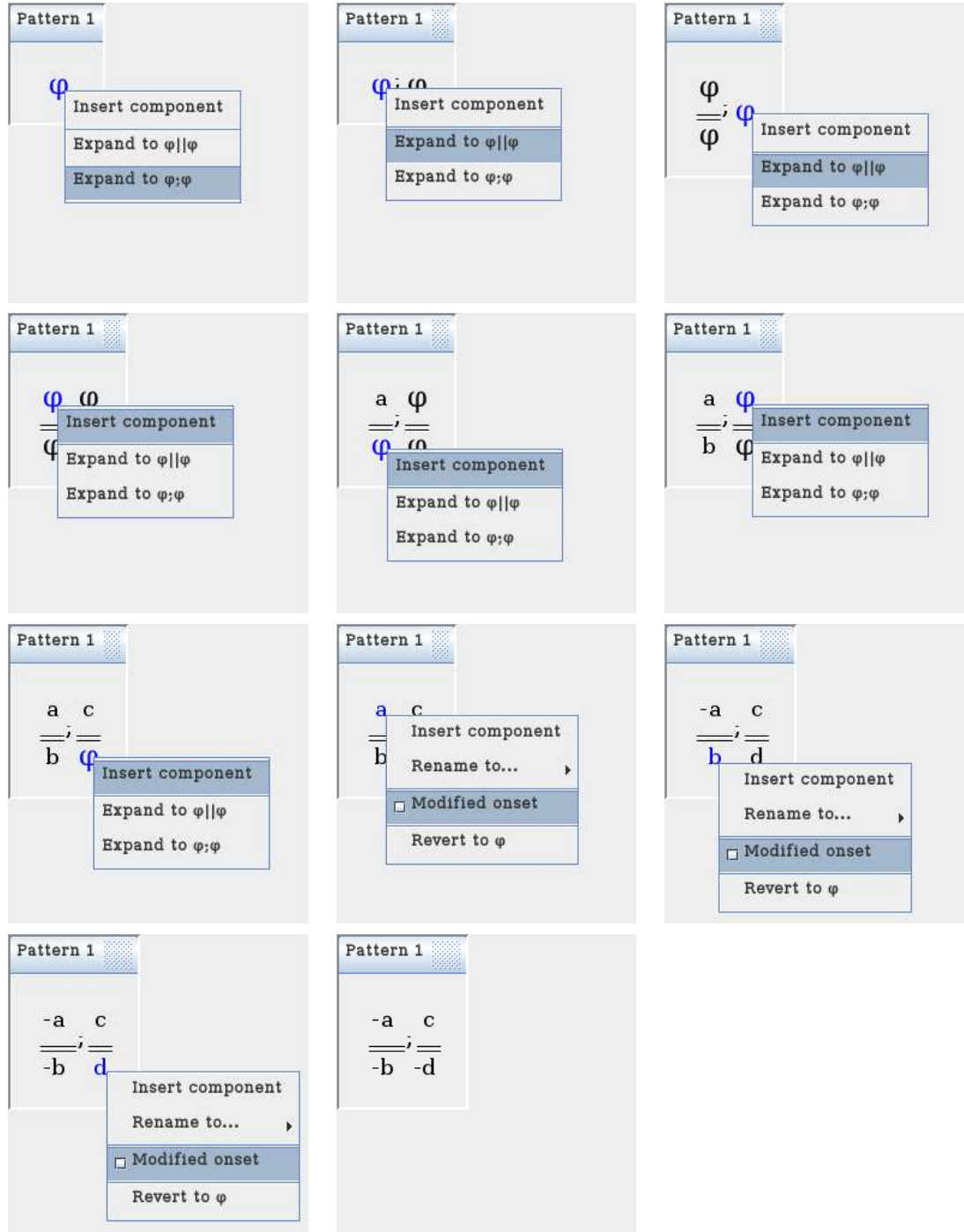


Figure 7.10: Developing the suspension pattern in a mock *SPP* GUI

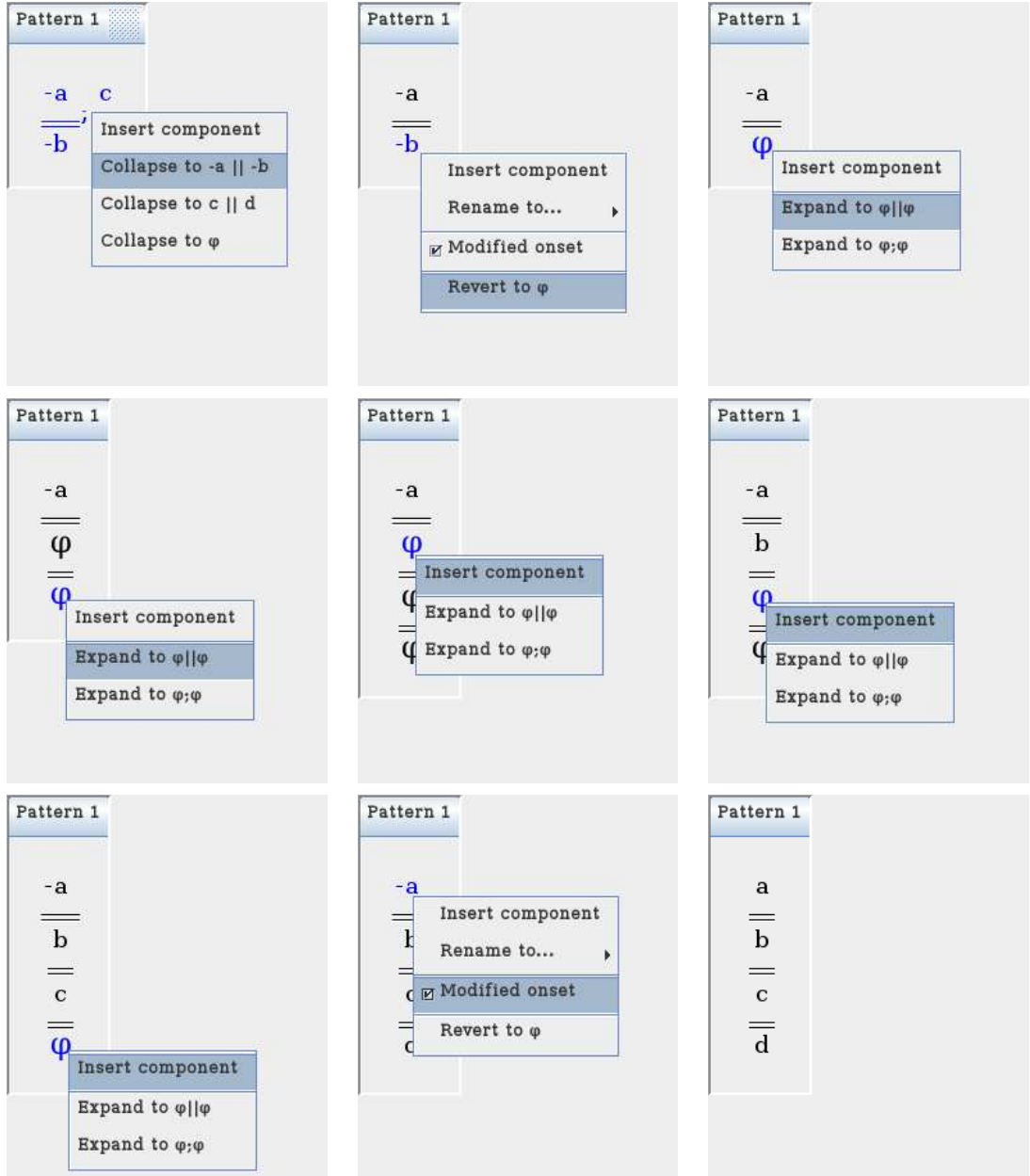


Figure 7.11: Transforming the suspension pattern into a block chord pattern in a mock *SPP* GUI



Figure 7.12: Examples of the rising fourths canon pattern: a) BWV 278 bar 8 between tenor and alto; b) BWV 328 bar 43 between bass and tenor; The excerpts are shown in piano-roll notation in figures a) F.36 on page 207; and b) F.37 on page 208

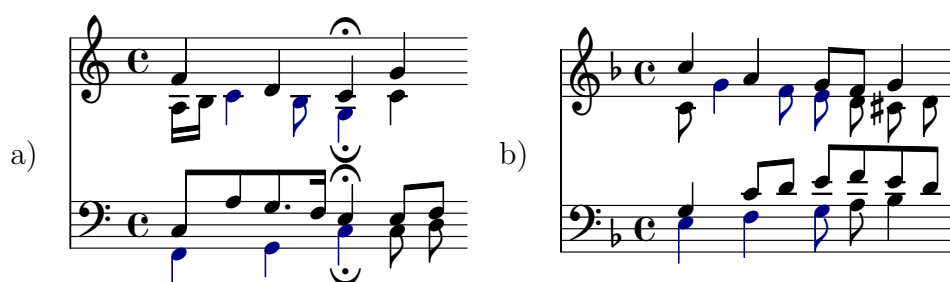


Figure 7.13: Examples of a two-voice module: a) BWV 328 bar 33 between bass and alto; b) BWV 382 bar 9, also between bass and alto; The excerpts are shown in piano-roll notation in figures a) F.38 on page 209; and b) F.39 on page 210

Chapter 8

Conclusion

8.1 Summary and contributions

The research presented in this dissertation is a contribution to the field of computational musicology. Our approach is inspired by the work of David Huron (Huron, 2001a,b) and Darrell Conklin (Conklin, 2002; Fitsioris and Conklin, 2008). As stated in Chapter 1, the general aim of the research was to create a modern pattern matching method for polyphonic music. By studying existing computer representations of polyphonic music and examples of polyphonic patterns from music theory textbooks, Chapter 2 developed a series of requirements for a polyphonic pattern language. In particular, the chapter included a proposition for a restricted set of temporal relations, one that is specialized for music: **m** (meet), **st** (start together), **sw** (start while), and **ov** (overlap). The requirements and temporal relations are contributions of this dissertation: an attempt to clearly circumscribe the idea of patterns in polyphonic music.

Using these requirements, Chapter 3 compared existing approaches to polyphonic patterns. We reviewed three approaches that are expressive and precise enough to capture the polyphonic patterns of parallel fifth and suspension: i) relational patterns, ii) Humdrum and iii) Structured Polyphonic Patterns. The latter approach, *SPP*, is a novel approach that this dissertation contributed.

In Chapter 4, we gave a formal definition of *SPP*. The novel approach combines advantages of relational patterns and Humdrum:

- Like Humdrum, the syntax is specialized for music. This is not the case for a relational language implemented for example in Prolog. The user has to create

his or her own syntax by specifying appropriate relations. Also, the syntax of a relational pattern does not illustrate the structure of the pattern

- Like relational patterns, the semantics of *SPP* is clear. This is not the case for Humdrum, as the meaning of a pattern depends on prior transformations of the source
- Voicing is handled in an abstract way. This is similar to relational patterns, where voices are represented with variables. By contrast, Humdrum uses columns to represent voices and the source has to be manipulated to account for combinations of voices. In *SPP*, the combinations of voices are explored automatically

SPP also adds several advantages that none of the existing approaches exhibit:

- A clear feature definition mechanism
- A clear demarcation between semantics and implementation. By contrast, the efficiency of a Prolog query is subject to large variations depending on exactly how it is written (see for example the discussion of Section 7.4)
- A dedicated matching algorithm. As shown in Chapter 7, it out-performs Humdrum in most cases. Also, the algorithm can be reused for data mining
- A “compositional” semantics. The language can be extended by adding new operators
- A syntax and semantics that is conducive to music data mining

In addition to this informal comparison, Chapter 6 presented a formal comparison of the approaches in terms of temporal patterns. The methodology developed to compare the languages is a further contribution of this dissertation. From this comparison emerged a new language: *SPPseq*. Again, this is a further contribution of our work. The new language is less expressive, but might be conducive to a better, more efficient pattern matching algorithm; and to a more efficient data mining method for polyphonic music.

Finally, Chapter 5 showed how to use *SPP* for simple and complex queries on diverse musical corpora. We showed how some polyphonic patterns are salient in Bach chorales, i.e. more probable in Bach chorales than in the other corpora. As well

as illustrating the usefulness of *SPP*, the chapter contributed to some degree to the empirical study of music by validating well-known claims about musical style, such as the relatively low probability of the parallel fifth pattern in chorale harmonizations, while the pattern is significantly more probable in orchestral and piano music.

8.2 Anticipated developments

The main future development is a method for the discovery of new polyphonic patterns. Pattern discovery is an important part of computational musicology research. As Chapter 3 shows, there have been very few attempts at pattern discovery for polyphonic music. The *SPP* language developed in this thesis can be applied to such a difficult problem, as discussed in Section 7.6.

Further developments may include extensions to the language, for example to accommodate dynamically voiced music, where keeping track of how melodies appear and disappear is required. Another avenue is to apply the language to other domains than music. Any source that can be represented as concurrent streams of events could be construed as n-part polyphony. Recall that *SPP* does not require the events to be musical. However, abstractions over such a non-musical source would need to be defined in terms of the temporal relations that *SPP* expresses.

Finally, extending the work of Chapter 7 on a mock *SPP* GUI, a complete pattern matching tool can be developed. It could also support the other pattern languages developed in Chapter 6: $\mathcal{R}$ (relational patterns) and $\mathcal{H}$ (based on Humdrum).

Appendix A

Musical concepts

A.1 Overview

Most music-theoretic concepts revolve around the notion of pitch. In this dissertation, we use the scientific notation (Young, 1939): a pitch is encoded as a name (a letter), an optional accident symbol (either a sharp $\sharp$ or a flat $\flat$) and an octave number, e.g. $D\flat 4$. In terms of auditory perception, pitch is usually conceived as an abstract description of the frequency of a sound wave. The octave number refers to a particular frequency bandwidth, with each octave encompassing frequencies that are twice as big as those of the preceding octave. For example, Table A.1 shows how some pitches in scientific notation are related to frequencies. Notice how, for example, the $C\sharp$ in octave number 4 ($C\sharp 4$) has twice the frequency of the $C\sharp$ in octave number 3 ($C\sharp 3$).

Octave The concept of *octave* plays an important role in music theory. With respect to notation, it can be defined as the labelling of pitches with double or half the frequency with the same name. But the notion is also a basic fact of music perception: pitches that are an octave apart are heard as very similar, even equivalent. This notion of octave equivalence is described by Dowling and Harwood (1986, p.4) as one of three musical concepts that are valid across cultural boundaries. The second one is how the octave is perceptually divided in a discrete number of pitches. This is consistent with the fact that music notation does not accommodate arbitrary pitches. Finally, a piece of music is typically organized around a small number of focal pitches: between four and seven per octave.

Pitch	Frequency by octave (Hz)		
	3	4	5
C	130.81	261.63	523.25
C $\sharp$ /D $\flat$	138.59	277.18	554.37
D	146.83	293.66	587.33
D $\sharp$ /E $\flat$	155.56	311.13	622.25
E	164.81	329.63	659.26
F	174.61	349.23	698.46
F $\sharp$ /G $\flat$	185.00	369.99	739.99
G	196.00	392.00	783.99
G $\sharp$ /A $\flat$	207.65	415.30	830.61
A	220.00	440.00	880.00
A $\sharp$ /B $\flat$	233.08	466.16	932.33
B	246.94	493.88	987.77

Table A.1: Interpretation of absolute pitches in scientific notation in terms of frequency

Clef and key signature In common music notation, these focal pitches are indicated by the clef and the key signature. The clef simply specifies that some staff line is a reference point that corresponds to a given pitch. For example, the bass clef (a stylized cursive F) indicates that the second topmost line of the staff corresponds to an F. Lines and spaces of the staff can then be assigned pitches by moving step by step on the staff while moving step by step in the opposite direction in Table A.1; jumping over lines that display a sharp symbol $\sharp$ or a flat symbol $\flat$; and wrapping around the table if necessary. The key signature indicates that some of the staff lines correspond to pitches that are either raised by a sharp symbol $\sharp$ or lowered by a flat symbol $\flat$.

Scale In addition, the focal pitches denoted by the clef and key signature correspond to the notion of scale. Starting from any particular focal pitch, a *scale* is a process by which six other focal pitches are identified in a particular order. For example, starting from G, the *major* scale identifies the following pitches: A, B, C, D, E, and F $\sharp$. The pitch G is then called the first *degree* of the scale, the pitch A its second degree, and so forth. The first degree is also called the *tonic*. It is considered as the most important pitch of the scale and melodies in a particular scale very often start or end on that note.

The logic behind the major scale is expressed in terms of semitone. A *semitone* is the pitch distance between any two contiguous lines in Table A.1, e.g. from C to

C $\sharp$, or from E to F. Starting from any pitch in Table A.1, the major scale names pitches distanced by respectively 2,4,5,7,9, and 11 semitones; in that order, moving downwards in the list of pitch names and wrapping around the table if necessary (as is the case for the G major scale above). The *minor* scale is a similar process, using the semitone distances 2,3,5,7,8, and 10 (or 11 in some variations). When a piece of music is mostly organized around the major scale, it is said to be in *major mode*. Conversely, if it uses the minor scale, it is said to be in *minor mode*. These two modes are largely predominant in western concert music.

There are many competing explanations for the historical predominance of these two modes, how to build the scales and how this affects the exact frequencies that a pitch should represent. The frequencies of Table A.1 are only one possibility and, for example, it is usually considered that pitch names appearing on the same line in Table A.1 should in fact be differentiated in term of frequency. These confounded names, e.g. C $\sharp$ /D $\flat$ are called *enharmonics*. When the differentiation is available in the data, i.e. when we can be confident that the $\sharp$ and $\flat$ symbols reflect the intention of the composer or editor of the piece, we say the pitches are *spelled*. When spelled pitches are available, one typically prefers not to use semitones as a measure of pitch distance. The notion of distance based on the major scale and called *diatonic interval* is far more common in the music-theoretic literature.

Diatonic intervals A diatonic interval, simply called interval in this dissertation, is formed between any two spelled pitches and labelled according to the distance between the pitch names, expressed in how many degrees of the major scale separate the names. For example, as D is the fifth degree in the G major scale, the interval G-D $\flat$ is labelled as a fifth. In addition, when the interval is shortened or lengthened by the presence of accidentals that were not part of the scale, the interval is affixed with a *quality*. The intervals of the unison, fourth and fifth are called *perfect* intervals. When shortened, they become *diminished* and when lengthened, they become *augmented*. The interval G-D $\flat$ above is hence a diminished fifth. The intervals of the second, third, sixth and seventh are the major intervals. When shortened, they become *minor* intervals; if shortened twice, they become *diminished* intervals. When lengthened, they become *augmented* intervals.

The notion of interval differentiates between enharmonic pitches: the distance between G-D $\flat$ and G-C $\sharp$ is three six semitones in both cases, but in the first case it consists of a diminished fifth, while in the latter case it consists of an augmented fourth.

In addition, the intervals have the property of obeying the principle of octave equivalence. Pitches separated by more than one octave are still considered to form intervals ranging from the unison to the seventh. In particular, the interval of an octave is assimilated with the unison. Such octave-equivalent intervals are called *compound* intervals. This is the most common form of interval. However, it is sometimes useful to include octaves in the notion of intervals. In this dissertation, we call these *non-compound* intervals (abbreviated **nc** in Chapter 5). To name non-compound intervals, one simply continues to count scale degrees higher than seven: an octave is a eighth, an octave plus a second is a ninth, and so on.

Consonance In music theory, the intervals are classified as consonant or dissonance. A *consonance* is considered to be a stable sonority, one on which a piece can plausibly end. A *dissonance* is considered to be an unstable sonority, one that invites motion. The motion from a dissonance to a consonance, called a *resolution*, is a recurrent theme in music theory. Following Piston (1941, chap. 2), the intervals considered as consonant in this dissertation are the unison, the minor and major third, the perfect fifth, and the minor and major sixth. All other intervals are considered to be dissonant.

Metrical levels Finally, common music notation supports an indication of temporal organization called *meter*. Indicated by the time signature, the meter groups beats into bars, which divide a piece of music into segments. Typically, notes at the beginning of a bar are considered to be metrically accented and are meant to be highlighted by the performer in one way or another. The exact meaning of time signatures vary, but they all correspond to some labelling of the notes with metrical accents of varying intensity, e.g. 0 for the strongest, 1 for intermediary and 2 for the lowest. We call this the *metrical level* of a note, a notion that appears in some music-theoretic patterns.

A.2 Glossary

Concept	Definition
Note	Unit of sound; symbol with which the unit is written down
Beat	Time unit; corresponds to the time elapsed between two taps when following the tempo of a piece of music
Onset	Start time of a note; measured in number of beats since start time of the piece
Offset	End time of a note; measured in number of beats since start time of the piece
Duration	Time elapsed between the onset and offset of a note; measured in beats
Pitch	Auditory property of a note that make it sound higher (like a female voice) or lower (like a male voice); pitch is measured in frequency and referred to with pitch names
Octave	Pitches that have twice or half the frequency; when separated by an octave, pitches are considered very similar
Sharp	Symbol that raises the pitch of a note
Flat	Symbol that lowers the pitch of a note
Staff	Five horizontal lines on which notes are drawn
Staff line	Each line corresponds to a specific pitch; spaces between the lines also correspond to specific pitches
Clef	Identifies one of the staff lines as a reference pitch
Key signature	Identifies the prescribed pitch of other staff lines, e.g. by assigning them implicit sharp symbols
Time signature	Identifies how beats are counted, e.g. by groups of three
Bar	Group of beats; useful to segment the piece into small units (typically two to four beats)
Bar line	Visual depiction of the bars
Metrical level	Position of a note within a bar; typically notes in specific positions are accented

Table A.2: Basic musical concepts

Symbol	Duration	Dotted	Duration	Tied	Duration
	4		$4 + 4 \times \frac{1}{2}$		$4 + 2$
	2		$2 + 2 \times \frac{1}{2}$		$2 + 1$
	1		$1 + 1 \times \frac{1}{2}$		$1 + 1$
	$\frac{1}{2}$		$\frac{1}{2} + \frac{1}{2} \times \frac{1}{2}$		$\frac{1}{2} + \frac{1}{2}$

Table A.3: Examples of durations in common music notation

Concept	Definition
Part	The notes that one instrument or section plays, e.g. the piano part or the soprano section of a choir
Voice	When a part can be mentally divided in subparts, each is said to be a voice; in this dissertation however, the notions of part and voice are used without distinction
Melody	A sequence of notes that is perceived as salient in a piece; often the melody is what a listener remembers and recognizes later on
Transposition	Translation of a melody upwards or downwards in the pitch dimension. Transposed melodies are perceptually very similar
Independence	When multiple melodies can be mentally segregated
Sequence	A list of notes, occurring directly one after the other
Layer	A collection of notes, all starting at the same time; in this dissertation we extend the idea to notes that overlap without necessarily starting at the same time
Chord	A layer with specific pitches that are identified as having an important role
Monophony	Musical texture that consists solely of sequences of notes
Homophony	Musical texture that consists of sequences of layers
Polyphony	Musical texture that consists of overlapping melodies or, in general, of any structural organization of the notes

Table A.4: Musical concepts about texture

Concept	Definition
Scale	Identifies seven pitches that are predominant in a piece
Degree	Labelling of the notes with their position in the scale
Tonic	First degree of a scale; typically, it plays an important role in how the piece is structured and described
Key	Indication of the mode and tonic
Mode	Major or minor; this corresponds to two different scales
Semitone	Smallest pitch distance
Enharmonics	Pitches that sound very similar or even the same on some instruments, e.g. the piano
Pitch spelling	Differentiating enharmonics by using different pitch names
Diatonic interval	Measure of pitch distance
Quality	Diminished, minor, major, perfect or augmented; qualifies the diatonic intervals
Perfect	The unison (no interval), fourth, fifth and octave
Major	The second, third, sixth and seventh
Minor	When a major interval is shortened
Augmented	When a major or perfect interval is lengthened
Diminished	When a minor or perfect interval is shortened
Compound	Intervals that forget octaves; intervals bigger than an octave are considered as equivalent to their remainder after an octave, e.g. the interval of an octave plus a fifth is equated with a fifth; this is the default case in this dissertation
Non-compound (nc)	Intervals that take the octave into account, e.g. the interval of an octave and a fifth is encoded as a twelfth

Table A.5: Musical concepts about pitch relations

Concept	Definition
Consonance	Intervals that are considered as stable, i.e. a piece is likely to end on such intervals: the unison, the perfect fifth, the major and minor third and sixth
Dissonance	Intervals that are considered as unstable, i.e. a piece is likely to continue after sounding those intervals; all intervals except the ones mentioned above
Preparation	Introducing a dissonance
Resolution	Moving from a dissonance to a consonance

Table A.6: Musical concepts about style

Appendix B

User-defined features

Feature	Values	Description
onset	0, 1, ...	Start time of an event
offset	0, 1, ...	End time of an event
duration	0, 1, ...	Elapsed time between onset and offset
pitch	C4, C#4, ...	Spelled pitch of a note
key	Cmaj, Cmin, ...	Tonic and mode
meter	0, 1, ...	Metrical level (0 for strongest)
degree	1, 2, 3, ...	Scale degree

Table B.1: Features that encode properties of a single event

Feature	Values	Description
<code>m_interval</code>	<code>P1, m2, M2, ...</code>	Melodic interval: diatonic interval over the <code>m</code> temporal relation
<code>m_interval_nc</code>	<code>P1, m2, M2, ...</code>	Non-compound melodic interval: the octave is recorded
<code>m_size</code>	<code>U, S, L</code>	Classifying melodic intervals with respect to small intervals (steps) and big intervals (leaps)
<code>m_dir</code>	<code>=, +, -</code>	Direction of melodic motion
<code>accented</code>	<code>T, F</code>	True if the current note has a stronger metrical level than its predecessor

Table B.2: Features that encode melodic properties: relations of the current event with its direct predecessor

Where	Add
<code>{pitch : P1}_x ; {pitch : P2}_x^*</code>	<code>m_interval : diatonic_interval(P1, P2)</code>
<code>{pitch : P1}_x ; {pitch : P2}_x^*</code>	<code>m_interval_nc : diatonic_interval_nc(P1, P2)</code>
<code>{m_interval_nc : I}_x^*</code>	<code>m_size :</code> if <code>I < m2</code> then <code>U</code> if <code>I < m3</code> then <code>S</code> else <code>L</code>
<code>{pitch : P1}_x ; {pitch : P2}_x^*</code>	<code>m_dir :</code> if <code>P2 < P1</code> then <code>-</code> if <code>P2 > P1</code> then <code>+</code> else <code>=</code>
<code>{ml : M1}_x ; {ml : M2}_x^*</code>	<code>accented :</code> if <code>M2 < M1</code> then <code>T</code> else <code>F</code>

Table B.3: Definition rules for features that encode melodic properties (Table B.2)

Feature	Values	Description
<code>st_interval(x)</code>	<code>P1, m2, M2, ...</code>	Harmonic interval for notes that start together: diatonic interval over the <code>st</code> temporal relation
<code>sw_interval(x)</code>	<code>P1, m2, M2, ...</code>	Harmonic interval when the current note starts while some note in voice <code>x</code> is unfolding: diatonic interval over the <code>sw</code> temporal relation
<code>et_interval(x)</code>	<code>P1, m2, M2, ...</code>	Harmonic interval for notes that end together: diatonic interval over the <code>et</code> temporal relation
<code>st_interval_nc(x)</code>	<code>P1, m2, M2, ...</code>	Non-compound harmonic interval over <code>st</code> : octaves are recorded
<code>et_interval_nc(x)</code>	<code>P1, m2, M2, ...</code>	Non-compound harmonic interval over <code>et</code> : octaves are recorded
<code>st_cons(x)</code>	<code>T, F</code>	Consonance or dissonance over the <code>st</code> temporal relation; the unison (octave), the perfect fifth, the major and minor third and sixth are considered as consonance; other intervals are considered as dissonant
<code>sw_cons(x)</code>	<code>T, F</code>	Consonance or dissonance over the <code>sw</code> temporal relation
<code>et_cons(x)</code>	<code>T, F</code>	Consonance or dissonance over the <code>et</code> temporal relation

Table B.4: Features that encode harmonic properties: relations of the current event with an overlapping event in some other voice `x`

Where	Add
$\frac{\{\text{pitch} : P1\}_x^*}{\{\text{pitch} : P2\}_y}$	$\text{st_interval}(y) : \text{diatonic_interval}(P1, P2)$
$\frac{\{\text{pitch} : P1\}_x^*}{-\{\text{pitch} : P2\}_y}$	$\text{sw_interval}(y) : \text{diatonic_interval}(P1, P2)$
$\frac{-\{\text{pitch} : P1\}_x^*}{-\{\text{pitch} : P2\}_y} ; \frac{\{\}_x}{\{\}_y}$	$\text{et_interval}(y) : \text{diatonic_interval}(P1, P2)$
$\frac{\{\text{pitch} : P1\}_x^*}{\{\text{pitch} : P2\}_y}$	$\text{st_interval_nc}(y) : \text{diatonic_interval_nc}(P1, P2)$
$\frac{\{\text{pitch} : P1\}_x^*}{-\{\text{pitch} : P2\}_y}$	$\text{sw_interval_nc}(y) : \text{diatonic_interval_nc}(P1, P2)$
$\frac{-\{\text{pitch} : P1\}_x^*}{-\{\text{pitch} : P2\}_y} ; \frac{\{\}_x}{\{\}_y}$	$\text{et_interval_nc}(y) : \text{diatonic_interval_nc}(P1, P2)$
$\{\text{st_interval}(y) : I\}_x$	$\text{st_cons}(y) :$ if $I \in \{P1, m3, M3, P5, m6, M6\}$ then T else F
$\{\text{sw_interval}(y) : I\}_x$	$\text{sw_cons}(y) :$ if $I \in \{P1, m3, M3, P5, m6, M6\}$ then T else F
$\{\text{et_interval}(y) : I\}_x$	$\text{et_cons}(y) :$ if $I \in \{P1, m3, M3, P5, m6, M6\}$ then T else F

Table B.5: Definition rules for features that encode harmonic properties (Table B.4)

Feature	Values	Description
<code>parallel(x)</code>	T, F	When two voices move in the same direction and repeat the same harmonic interval
<code>par1_5(x)</code>	T, F	When two voices exhibit parallel motion for the intervals of a unison or perfect fifth
<code>passing(x)</code>	T, F	True if the current note is a passing tone with respect to voice <code>x</code>
<code>strong(x)</code>	T, F	Counterpoint property: true if the current note is a valid strong note with respect to voice <code>x</code> ; it must be a consonance and cannot exhibit a parallel fifth or octave
<code>weak(x)</code>	T, F	Counterpoint property: true if the current note is a valid weak note with respect to voice <code>x</code> ; it must either be a consonance or be a passing tone

Table B.6: Features that encode complex properties: relations between more than two events

Where	Add
$\{\text{et_interval_nc}(y) : I1\}_x$ $; \{\text{st_interval_nc}(y) : I2, \text{m_size} : S\}_x^*$	$\text{parallel}(y) :$ $\text{if } I1 = I2$ $\quad \wedge S \neq U$ $\text{then } T$ $\text{else } F$
$\{\text{et_interval}(y) : I\}_x$ $; \{\text{st_interval}(y) : I, \text{parallel}(y) : P\}_x^*$	$\text{par1_5}(y) :$ $\text{if } (I = P1 \vee I = P5)$ $\quad \wedge P = T$ $\text{then } T$ $\text{else } F$
$a ; b^* ; c$ $\bar{a} = \{\}_x$ $\bar{b} = \{\text{et_cons}(y) : C1, \text{m_size} : S1,$ $\quad \text{m_dir} : D1\}_x$ $\bar{c} = \{\text{st_cons}(y) : C2, \text{m_size} : S2,$ $\quad \text{m_dir} : D2\}_x$	$\text{passing}(y) :$ $\text{if } C1 = F \wedge C2 = T$ $\quad \wedge S1 = S \wedge S2 = S$ $\quad \wedge D1 = D2$ $\text{then } T$ $\text{else } F$
$\{\text{st_cons}(y) : C, \text{par1_5}(y) : P\}_x^*$	$\text{strong}(y) :$ $\text{if } C = T$ $\quad \wedge P = F$ $\text{then } T$ $\text{else } F$
$\{\text{st_cons}(y) : C, \text{passing}(y) : P\}_x^*$	$\text{weak}(y) :$ $\text{if } C = T$ $\quad \vee P = T$ $\text{then } T$ $\text{else } F$

Table B.7: Definition rules for features that encode complex properties (Table B.6)

Appendix C

Notation

Concept	Examples	Definition
Source	s s_1, s_2	A collection of musical events
Event	e e_1, e_2	A set of event features; the set is labelled with a voice name
Event feature	f $\text{pitch} : G\sharp,$ $\text{parallel}(\text{bass}) : T$	A feature name and its value; the feature possibly encodes a relation with an event in some other voice
Feature name	τ $\text{pitch}, \text{interval}$	
Voice name	n bass, alto	
Feature value	v $G\sharp, T$	
Component	κ κ_1, κ_2	Abstraction of events; values and voice names can be replaced with variables
Component feature	$\hat{f}$ $\text{pitch} : P,$ $\text{parallel}(x) : X$	P and X are value variables; x is a voice variable
Voice variable	γ x, y	
Value variable	α P, X	

Table C.1: Syntax and meaning of $\mathcal{SPP}$ sources and components

	Concept	Example	Definition
Component index	ϵ	a, b	Unique identifiers for components
Meet		$\mathbf{m}(a, b)$	a meets b (a ends as b starts)
Start together		$\mathbf{st}(a, b)$	a and b start together
Starts while		$\mathbf{sw}(a, b)$	a starts while b is unfolding
Overlap		$\mathbf{ov}(a, b)$	a and b overlap; at some point in time, both a and b are unfolding
SPP pattern	ϕ	$a ; b$ $\frac{a}{b}$	Sequence of components, layer of components, or a mixture of both structures
Sequence operator	“,”	$a ; b$	b directly follows a ; the relation $\mathbf{m}(a, b)$ holds
Layer operator	“=”	$\frac{a}{b}$	a and b start at the same time; the relation $\mathbf{st}(a, b)$ holds
Onset operator	“—”	$\frac{a}{-b}$ $\frac{-a}{-b}$	Conceptually, b is extended backward in time: the relation $\mathbf{sw}(a, b)$ holds When both a and b are extended backward in time, the relation $\mathbf{ov}(a, b)$ holds
Assignment	θ		Assigns values to variables
Instance	I		Maps components to events

 Table C.2: Syntax and meaning of $\mathcal{SPP}$ patterns

Concept	Example	Definition
where $\phi[\epsilon^*]$	$\{\text{pitch} : P1\} ; \{\text{pitch} : P2\}^*$	The where clause specifies the locations in the source where the rule applies
ϵ^*	$\{\text{pitch} : P2\}^*$	One component in ϕ is distinguished by a $*$ symbol. The event matching this component will be added a feature
add $\widehat{f}[\Omega]$	$\text{m_interval} : \text{interval}(P1, P2)$	The add clause specifies the feature to add
Ω	$\text{interval}(P1, P2)$	The function Ω takes the value of the variables $P1$ and $P2$ and returns the value of the feature to add

 Table C.3: Syntax and meaning of $\mathcal{SPP}$ feature definition rules

Concept	Example	Definition
$\mathcal{R}$ pattern	$r \quad \mathbf{m}(a, b), \mathbf{sw}(d, a), \mathbf{sw}(b, d)$	A pattern in $\mathcal{R}$ is a set of temporal relations
$\mathcal{H}$ pattern	$h \quad \begin{bmatrix} d & (a) \\ (d) & b \end{bmatrix}$	A pattern in $\mathcal{H}$ is a matrix of tokens
Event	$\epsilon \quad d \text{ and } b$	The start of an event in an $\mathcal{H}$ pattern
Continuation	$(\epsilon) \quad (a) \text{ and } (d)$	The continuation of an event in an $\mathcal{H}$ pattern
Don't care	$\star$	The special “don't care” symbol in an $\mathcal{H}$ pattern

 Table C.4: Syntax of $\mathcal{R}$ and $\mathcal{H}$

Appendix D

Matching algorithm

Following Definition 10 (Chapter 4, page 68), the collection of instances can be built by recursively analyzing ϕ .

Definition 22 Given a pattern ϕ , the *SPP*-join algorithm returns a collection of instances by executing the following computation:

$SPP\text{-join}(\phi) =$
 If ϕ **is of the form** ϵ , **then:**
 $instances_{\epsilon} := \text{iterate_events}(\epsilon)$ (Definition 24)
 return $instances_{\epsilon}$

If ϕ **is of the form** $\phi_1 ; \phi_2$, **then:**
 $instances_{\phi_1} := SPP\text{-join}(\phi_1)$
 $instances_{\phi_2} := SPP\text{-join}(\phi_2)$
 $instances_{\phi} := \text{iterate_seq}(\phi, \phi_1, \phi_2)$ (Definition 25)
 return $instances_{\phi}$

If ϕ **is of the form** $\frac{\phi_1}{\phi_2}$, **then:**
 $instances_{\phi_1} := SPP\text{-join}(\phi_1)$
 $instances_{\phi_2} := SPP\text{-join}(\phi_2)$
 $instances_{\phi} := \text{iterate_layer}(\phi, \phi_1, \phi_2)$ (Definition 29)
 return $instances_{\phi}$

To compute the instances of a pattern ϕ with subpatterns ϕ_1 and ϕ_2 , the algo-

rithm proceeds as follows. First the instances of ϕ_1 and ϕ_2 are computed. Then, pairs of instances of ϕ_1 and ϕ_2 are iterated (this is done respectively by `iterate_seq` and `iterate_layer`). Every pair is joined to form a candidate instance of ϕ (this simply consists of the union of ϕ_1 and ϕ_2 respective instances). The candidate instance is tested against Definition 10 to determine if it is a valid instance of ϕ . The efficient iteration of candidate instances is done using two specialized data structures. The first is used to store instances in such a way that an instance can be accessed directly via the point in time at which it starts.

Definition 23 For every pattern or subpattern ϕ , *SPP*-join builds a mapping from time points t to the set of instances that start at that point:

- $\text{instances}_\phi(t) = \{(I_1, \theta_1), \dots, (I_n, \theta_n)\}$
- All such mappings can be efficiently implemented as balanced trees, using the usual $\leq$ operator for comparisons.
- Notice that a θ -instance I is represented explicitly as a pair containing the set I and the variable assignment θ .

For the first case in Definition 22, `iterate_events`, the mapping is simply built as the source is scanned.

Definition 24 Given an *SPP* source s (Definition 1, Chapter 4, page 58), the function `iterate_events` is defined as follows:

```

iterate_events( $\epsilon$ ) =
  instances_ $\epsilon$  := {}
  for all  $e \in s$  do
     $t := e.\text{onset}$ 
    for all  $\theta$  such that  $e$  is a  $\theta$ -instance of  $\epsilon$  do
       $I := \{(\epsilon, e)\}$ 
      instances_ $\epsilon$ ( $t$ ) := instances_ $\epsilon$ ( $t$ )  $\cup \{(I, \theta)\}$ 

  return instances_ $\epsilon$ 

```

The second case in Definition 22, `iterate_seq`, takes advantage of the mapping to improve efficiency. We know that according to Definition 10, a valid instance of

$\phi_1 ; \phi_2$ has to enforce the $\mathbf{m}$ temporal relation between components that end ϕ_1 and those that start ϕ_2 . Consequently, for every instance of ϕ_1 , we can compute the point in time where the instance ends and directly access the relevant instances of ϕ_2 . Every resulting instance of ϕ is then stored in the appropriate mapping using the starting point of ϕ_1 (if ϕ is of the form $\phi_1 ; \phi_2$, it necessarily starts when ϕ_1 starts).

Definition 25

```

iterate_seq( $\phi, \phi_1, \phi_2$ ) =
  instances $_{\phi} := \{\}$ 
  for all  $(I_1, \theta_1) \in \text{instances}_{\phi_1}$  do
     $t :=$  the time at which  $I_1$  starts (Definition 26)
     $t' :=$  the time at which  $I_1$  ends (Definition 27)
    for all  $(I_2, \theta_2) \in \text{instances}_{\phi_2}(t')$  do
       $I_{12} := I_1 \cup I_2$ 
       $\theta_{12} := \theta_1 \cup \theta_2$ 
      if  $I_{12}$  is a valid  $\theta_{12}$ -instance of  $\phi$  then (Definition 10)
        instances $_{\phi}(t) := \text{instances}_{\phi}(t) \cup \{(I_{12}, \theta_{12})\}$ 

  return instances $_{\phi}$ 

```

Formally, we define what it means for an instance to start or end as follows.

Definition 26 Given an instance I of ϕ , we say that I *starts* at t if:

- There is a component ϵ in ϕ such that:
 - ϵ starts ϕ
 - ϵ would be aligned with ϵ' in $\epsilon' ; \phi$
- The instance I maps the component ϵ to the event e
- The time point t is the onset of e , i.e. $t = e.\text{onset}$

Definition 27 Given an instance I of ϕ , we say that I *ends* at t if:

- There is a component ϵ in ϕ such that:
 - ϵ ends ϕ
 - ϵ would be aligned with ϵ' in ϕ ; ϵ'
- The instance I maps the component ϵ to the event e
- The time point t is the offset of e , i.e. $t = e.\text{offset}$

The third case of Definition 22, `iterate_layer` is slightly more subtle. Again, we want to iterate the instances of ϕ_1 and directly access the instances of ϕ_2 . In some cases, the instance of ϕ_1 will start at the same time as the instance of ϕ_2 . This is the case where the **st** temporal relation is enforced. The mapping of Definition 23 suffices for such cases. For other cases, however, when the **sw** or **ov** temporal relations are enforced, we need to explore the set of instances that overlap with the current instance. This can be done efficiently provided the following mapping is already initialized:

Definition 28 The *SPP*-join algorithm assumes a mapping of all time points t to the set of time points where some event might overlap with an event found at t :

- $\text{overlaps}(t) = \{t_1, \dots, t_n\}$
- Again, the mapping can be efficiently implemented as a balanced tree, using the usual $\leq$ operator for comparisons.

Assuming the `overlaps` mapping, `iterate_layer` is defined as follows.

Definition 29

```

iterate_layer( $\phi, \phi_1, \phi_2$ ) =
  instance $_{\phi} := \{\}$ 
  for all  $(I_1, \theta_1) \in \text{instances}_{\phi_1}$  do
     $t :=$  the time at which  $I_1$  starts
    for all  $t' \in \text{overlaps}(t)$  do
      for all  $(I_2, \theta_2) \in \text{instances}_{\phi_2}(t')$  do

```

(Definition 26)

```

 $I_{12} := I_1 \cup I_2$ 
 $\theta_{12} := \theta_1 \cup \theta_2$ 
if  $I_{12}$  is a valid  $\theta_{12}$ -instance of  $\phi$  then (Definition 10)
     $\text{instances}_\phi(t) := \text{instances}_\phi(t) \cup \{(I_{12}, \theta_{12})\}$ 

return  $\text{instances}_\phi$ 

```

For completeness, the definition below shows how to initialize the `overlaps` mapping.

Definition 30

```

overlaps := {}
current_spans := {}
Sort source  $s$  is ascending order of onset and, for identical onsets,
    in descending order of offset
for all  $e \in s$  do
    if there is a  $f$  such that  $(e.\text{onset}, f) \in \text{current\_spans}$  then
        do nothing
    else
         $\text{current\_spans} := \text{current\_spans} \cup \{(e.\text{onset}, e.\text{offset})\}$ 
        for all  $(o, f) \in \text{current\_spans}$  such that  $o \leq e.\text{onset} \leq f$  do
             $\text{overlaps}(o) := \text{overlaps}(o) \cup \{e.\text{onset}\}$ 
        for all  $(o, f) \in \text{current\_spans}$  such that  $f > e.\text{onset}$  do
             $\text{current\_spans} := \text{current\_spans} \setminus (o, f)$ 

```

Appendix E

Prolog implementation

The Prolog implementation of $\mathcal{SPP}$ query consists of three elements: i) a translation of a well-formed $\mathcal{SPP}$ source to a set of Prolog facts (Definition 31), ii) a translation of an $\mathcal{SPP}$ pattern to a Prolog rule (Definition 33), which in turn relies on iii) an implementation as a set of Prolog rules of the four temporal relations supported by $\mathcal{SPP}$: $\mathbf{m}$, $\mathbf{st}$, $\mathbf{sw}$, $\mathbf{ov}$ (Definition 36). Given these three elements, the query is executed using a simple Prolog query that directly refers to the rule corresponding to the $\mathcal{SPP}$ pattern.

The first translation relies on attributing a unique index to every event in a $\mathcal{SPP}$ source.

Definition 31 Assuming an indexing function $\mathbf{index}(\cdot)$ that assigns a unique Prolog atom to every event of a $\mathcal{SPP}$ source s , the Prolog facts of that source are as follows:

$$\mathbf{source_facts}(s) = \{\mathbf{event_facts}(e, \mathbf{e}) \mid e \in s \wedge \mathbf{e} = \mathbf{index}(e)\}$$

The translation of an $\mathcal{SPP}$ event is a straightforward rewriting of the features of the event into Prolog syntax. For simplicity, we assume that every feature name τ and every voice name n is consistent with the syntax of Prolog atoms.

Definition 32 Given an $\mathcal{SPP}$ event e and its Prolog index $\mathbf{e}$, the translation of event e into Prolog facts is given as follows:

$$\begin{aligned}
 \text{event_facts}(e, \mathbf{e}) = & \\
 & \{ \\
 & \quad \text{If } f \text{ is } \tau : v, \quad \text{then } \text{feature}(\mathbf{e}, \tau, v) \\
 & \quad \text{If } f \text{ is } \tau(n) : v, \quad \text{then } \text{feature_voice}(\mathbf{e}, \tau, n, v) \\
 & \\
 & | f \in e \}
 \end{aligned}$$

To translate an *SPP* pattern ϕ to a Prolog rule, we assume a mapping of the components of ϕ to Prolog variables of the form **CompX** (where **X** is an integer). Every such variable will appear in the head of the rule. In the body of the rule, the same variable will appear in clauses specifying constraints about the relevant component: the features it contains and the temporal relation it forms. Similarly, we assume a mapping of the value variables of ϕ to Prolog variables of the form **ValX** and a mapping of the voice variables of ϕ to Prolog variables of the form **VoiceX**.

Definition 33 Given an *SPP* pattern ϕ with i components, j value variables and k voice variables, the Prolog rule that captures ϕ is as follows:

$$\begin{aligned}
 & \text{pattern}(\text{Comp1}, \dots, \text{CompI}, \text{Val1}, \dots, \text{ValJ}, \text{Voice1}, \dots, \text{VoiceK}) : - \\
 & \quad \text{pattern_clauses}(\phi)
 \end{aligned}$$

The exact clauses corresponding to a *SPP* pattern are obtained by recursively analyzing the pattern.

Definition 34 Assuming a mapping **comp_var** from component indices to Prolog variables of the form **CompX**, a mapping **value_var** from value variables to Prolog variables of the form **ValX** and a mapping **voice_var** from voice variables to Prolog variables of the form **VoiceX**, the clauses C of pattern ϕ are as follows:

$\text{pattern_clauses}(\phi) = C$ where C is as follows:

- If ϕ is of the form ϵ or $-\epsilon$, then:

$$\begin{aligned} C &= \text{component_clauses}(\bar{\epsilon}, \text{CompX}) \\ &\text{with } \text{CompX} = \text{comp_var}(\epsilon) \end{aligned} \quad (\text{Definition 11})$$

- If ϕ is of the form $\phi_1 ; \phi_2$, then:

$$\begin{aligned} C &= C_1 \cup C_2 \\ C_1 &= \text{pattern_clauses}(\phi_1) \\ C_2 &= \text{pattern_clauses}(\phi_2) \end{aligned}$$

and:

For every component ϵ_1 *ending* ϕ_1 (Definition 12)

And every component ϵ_2 *starting* ϕ_2 (Definition 13)

If ϵ_1 and ϵ_2 are *aligned* (Definition 14)

$$\begin{aligned} \text{then } C &= C \cup \{\text{m}(\text{CompX}, \text{CompY})\} \\ &\text{with } \text{CompX} = \text{comp_var}(\epsilon_1) \\ &\text{and } \text{CompY} = \text{comp_var}(\epsilon_2) \end{aligned}$$

- If ϕ is of the form $\frac{\phi_1}{\phi_2}$, then:

$$\begin{aligned} C &= C_1 \cup C_2 \\ C_1 &= \text{pattern_clauses}(\phi_1) \\ C_2 &= \text{pattern_clauses}(\phi_2) \end{aligned}$$

and:

For all components ϵ_1, ϵ_2 *starting* ϕ_1, ϕ_2 (Definition 13)

For pairs of the form ϵ_1, ϵ_2 , $C = C \cup \{\text{st}(\text{CompX}, \text{CompY})\}$

For pairs of the form $\epsilon_1, -\epsilon_2$, $C = C \cup \{\text{sw}(\text{CompX}, \text{CompY})\}$

For pairs of the form $-\epsilon_1, \epsilon_2$, $C = C \cup \{\text{sw}(\text{CompY}, \text{CompX})\}$

For pairs of the form $-\epsilon_1, -\epsilon_2$, $C = C \cup \{\text{ov}(\text{CompX}, \text{CompY})\}$

$$\begin{aligned} &\text{with } \text{CompX} = \text{comp_var}(\epsilon_1) \\ &\text{and } \text{CompY} = \text{comp_var}(\epsilon_2) \end{aligned}$$

The clauses of a pattern component are computed as defined below.

Definition 35 Assuming a mapping `value_var` from value variables to Prolog variables of the form `ValX` and a mapping `voice_var` from voice variables to Prolog

variables of the form **VoiceX**, we obtain the clauses of component κ associated with Prolog variable **CompX** as follows:

```

component_clauses( $\kappa$ , CompX) =
{
    If  $\widehat{f}$  is  $\tau : v$ ,      then  feature(CompX,  $\tau$ ,  $v$ )
    If  $\widehat{f}$  is  $\tau : \alpha$ ,    then  feature(CompX,  $\tau$ , ValY)
    If  $\widehat{f}$  is  $\tau(n) : v$ ,  then  feature_voice(CompX,  $\tau$ ,  $n$ ,  $v$ )
    If  $\widehat{f}$  is  $\tau(\gamma) : v$ , then  feature_voice(CompX,  $\tau$ , VoiceZ,  $v$ )
    If  $\widehat{f}$  is  $\tau(n) : \alpha$ , then  feature_voice(CompX,  $\tau$ ,  $n$ , ValY)
    If  $\widehat{f}$  is  $\tau(\gamma) : \alpha$ , then  feature_voice(CompX,  $\tau$ , VoiceZ, ValY)

    with ValY = value_var( $\alpha$ )
    and  VoiceZ = voice_var( $\gamma$ )

    |  $\widehat{f} \in \kappa$  }

```

Finally, the temporal relations supported by $\mathcal{SP}\mathcal{P}$ are defined below.

Definition 36

<pre> m(CompX, CompY) :- feature(CompX, offset, A), feature(CompY, onset, A). </pre>	<pre> st(CompX, CompY) :- feature(CompX, onset, A), feature(CompY, onset, A). </pre>
<pre> sw(CompX, CompY) :- feature(CompX, onset, A), feature(CompY, onset, B), feature(CompY, offset, C), A > B, A <= C. </pre>	<pre> ov(CompX, CompY) :- st(CompX, CompY). ov(CompX, CompY) :- st(CompY, CompX). ov(CompX, CompY) :- sw(CompX, CompY). ov(CompX, CompY) :- sw(CompY, CompX). </pre>

Appendix F

Additional tables and figures

F.1 Features

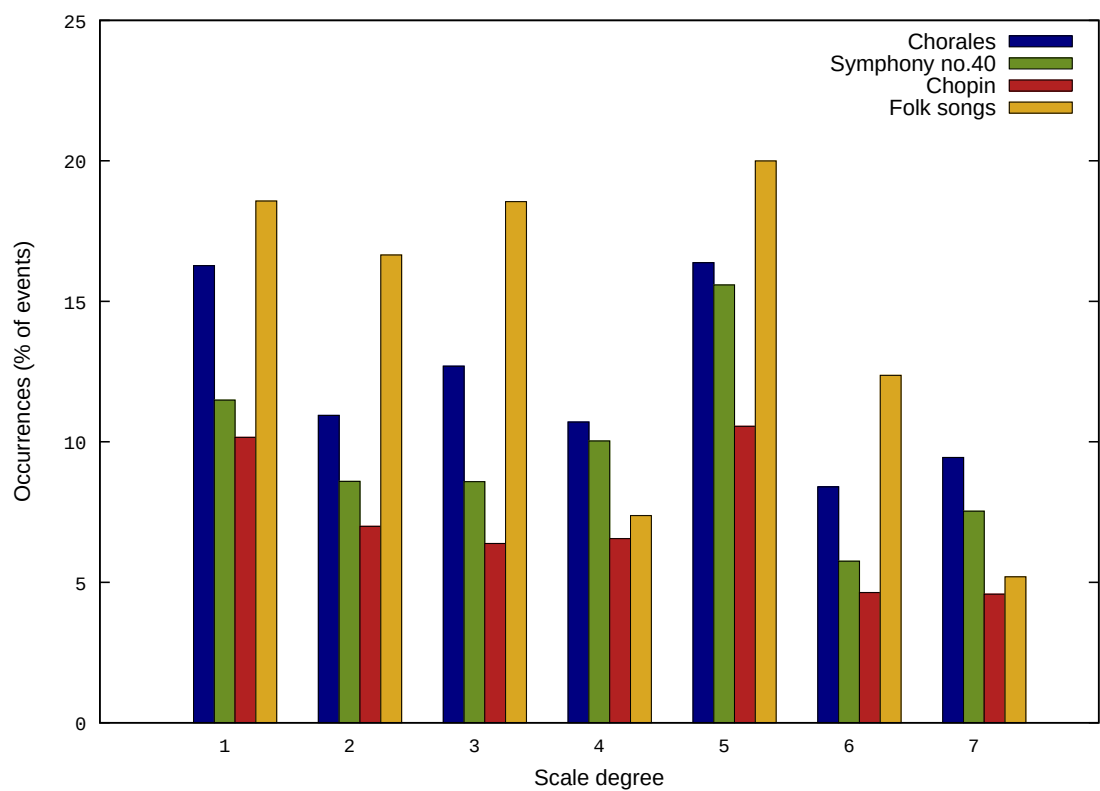


Figure F.1: Frame-conditioned probabilities of scale degrees in Bach chorales, Symphony no.40, Chopin and folk songs

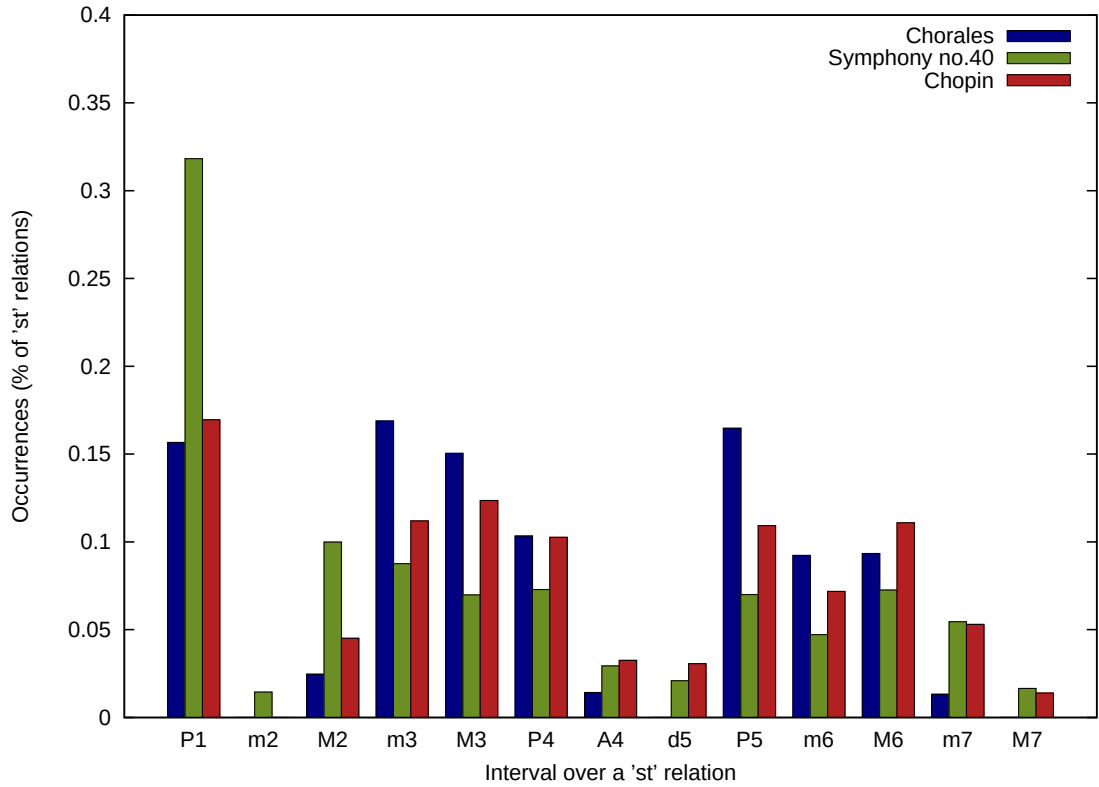


Figure F.2: Frame-conditioned probabilities of harmonic intervals in Bach chorales, Mozart Symphony no.40 and Chopin piano pieces. Only intervals occurring over at least 1% of `st` temporal relations are considered

F.2 Queries

(S1, D1)	(S2, D2)	Chorales	no.40	Chopin	Folk songs
(L, +)	(S, −)	19.48%	7.67%	5.31%	12.65%
(L, −)	(L, +)	18.92%	13.18%	17.10%	16.16%
(L, −)	(S, +)	16.88%	6.97%	4.65%	7.64%

Table F.1: The three most frequent melodic sequences that begin with a step and their frame-conditioned probabilities

Chord	Chorales	no.40	Chopin
Minor	2.32%	0.17%	0.14%
P5, P4, m3	32.66%	18.53%	17.37
m3, M3, P4	23.80%	11.53%	15.62%
P1, P5, m6	15.96%	23.63%	16.48%

Table F.2: Minor chord with the root as the lowest note and its frame-conditioned probability with respect to all block chords. The following lines show the three most frequent permutations and their frame-conditioned probabilities with respect to other possible permutations

Embellishment	Approach			Resolution		Chorales	no.40	Chopin
	S1	D1	A	S2	D2			
Upper neighbor	S	+	F	S	−	3.00%	1.80%	0.53%
Lower neighbor	S	−	F	S	+	5.53%	1.39%	0.58%
Acc. upper neighbor	S	+	T	S	−	2.16%	0.88%	0.21%
Acc. lower neighbor	S	−	T	S	+	1.52%	0.96%	0.27%
Passing	S	(= D2)	F	S	(= D1)	30.75%	3.06%	1.11%
Acc. passing	S	(= D2)	T	S	(= D1)	3.99%	1.49%	0.49%
Anticipation				F	=	10.63%	11.47%	9.54%
Retardation			T	S	+	1.84%	1.78%	1.56%
Escape	S		F	L		4.75%	3.91%	3.85%
Appoggiatura	L		F	S		0.79%	1.45%	1.21%
Repetition		=	F			4.24%	11.74%	11.76%
Arpeggio	L		F			7.54%	18.97%	18.38%
Unclassified						23.27%	41.08%	50.51%

Table F.3: Polyphonic embellishments and their frame-conditioned probabilities

F.3 Musical excerpts for Chapter 5

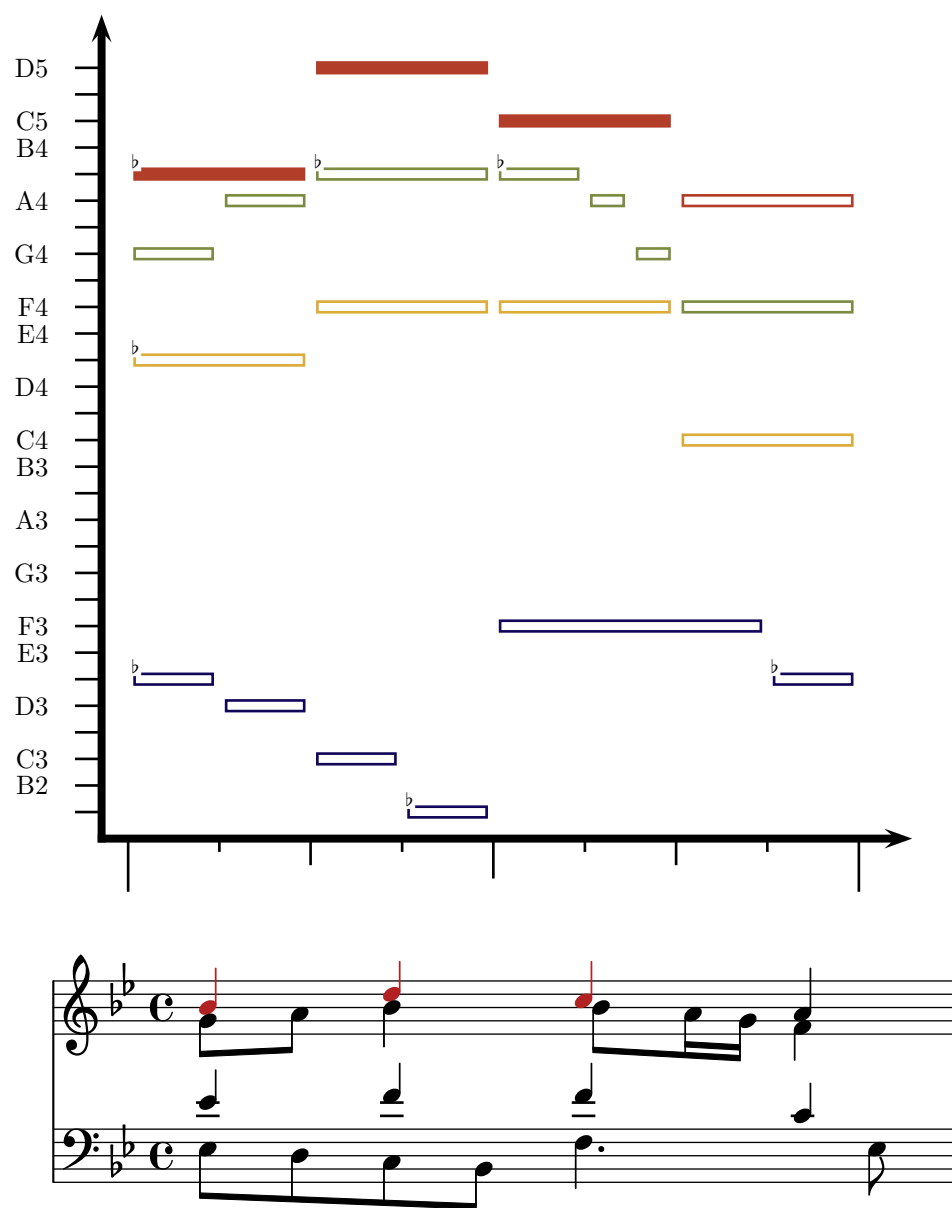


Figure F.3: BWV 273 (bar 5) with a highlighted compensated leap pattern

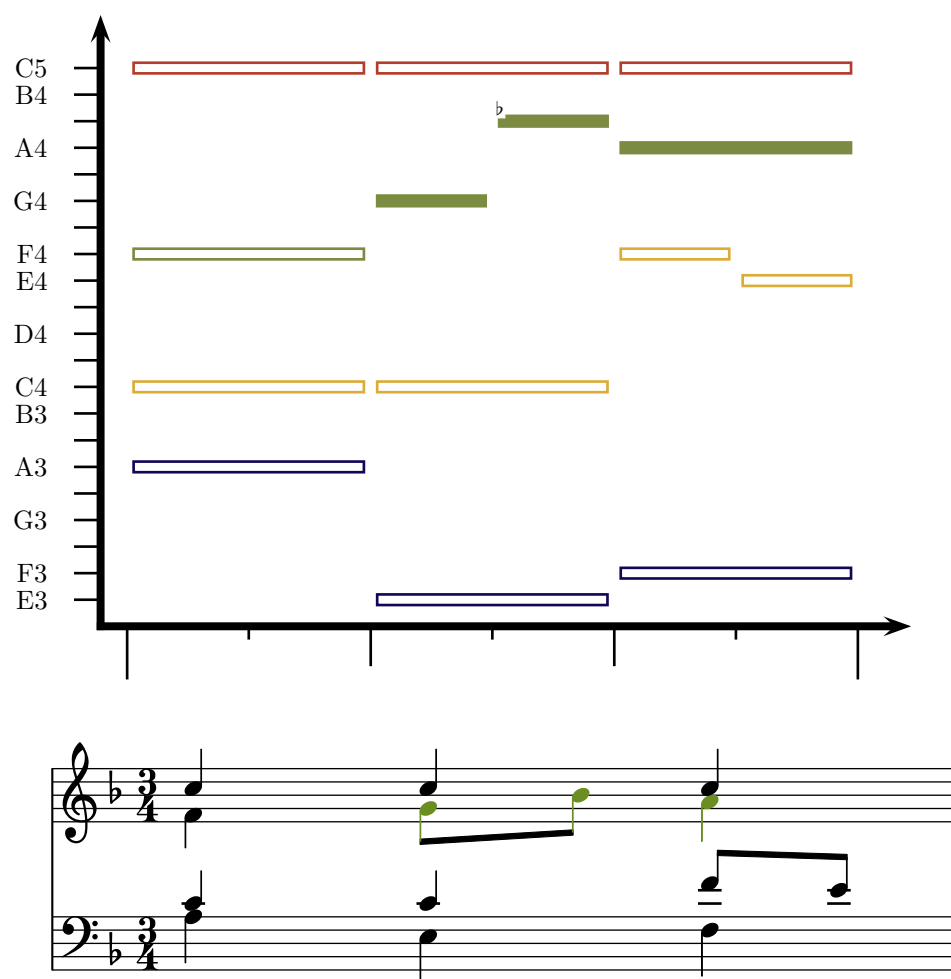


Figure F.4: BWV 349 (bar 13) with a highlighted compensated leap pattern

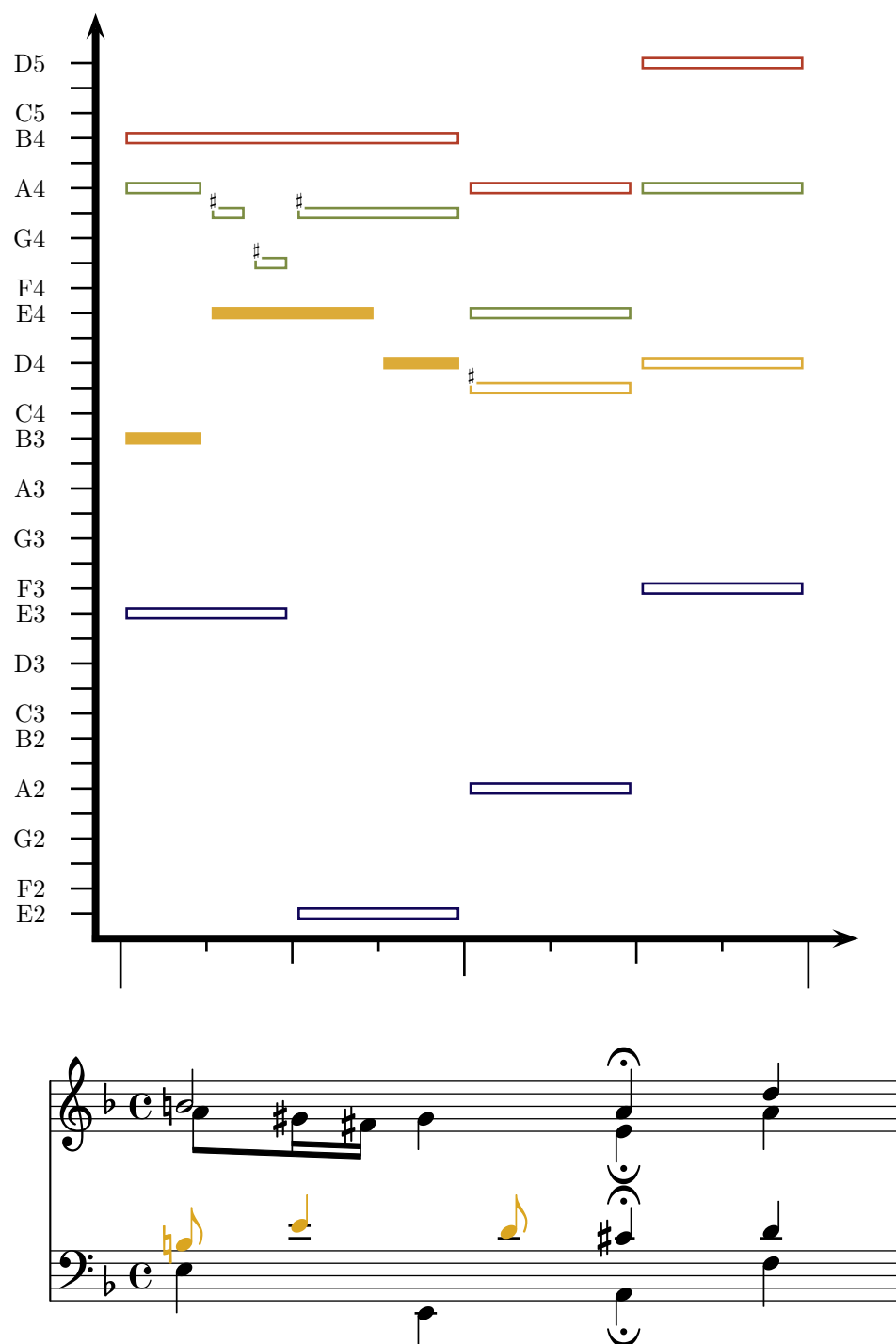


Figure F.5: BWV 297 (bar 3) with a highlighted compensated leap pattern

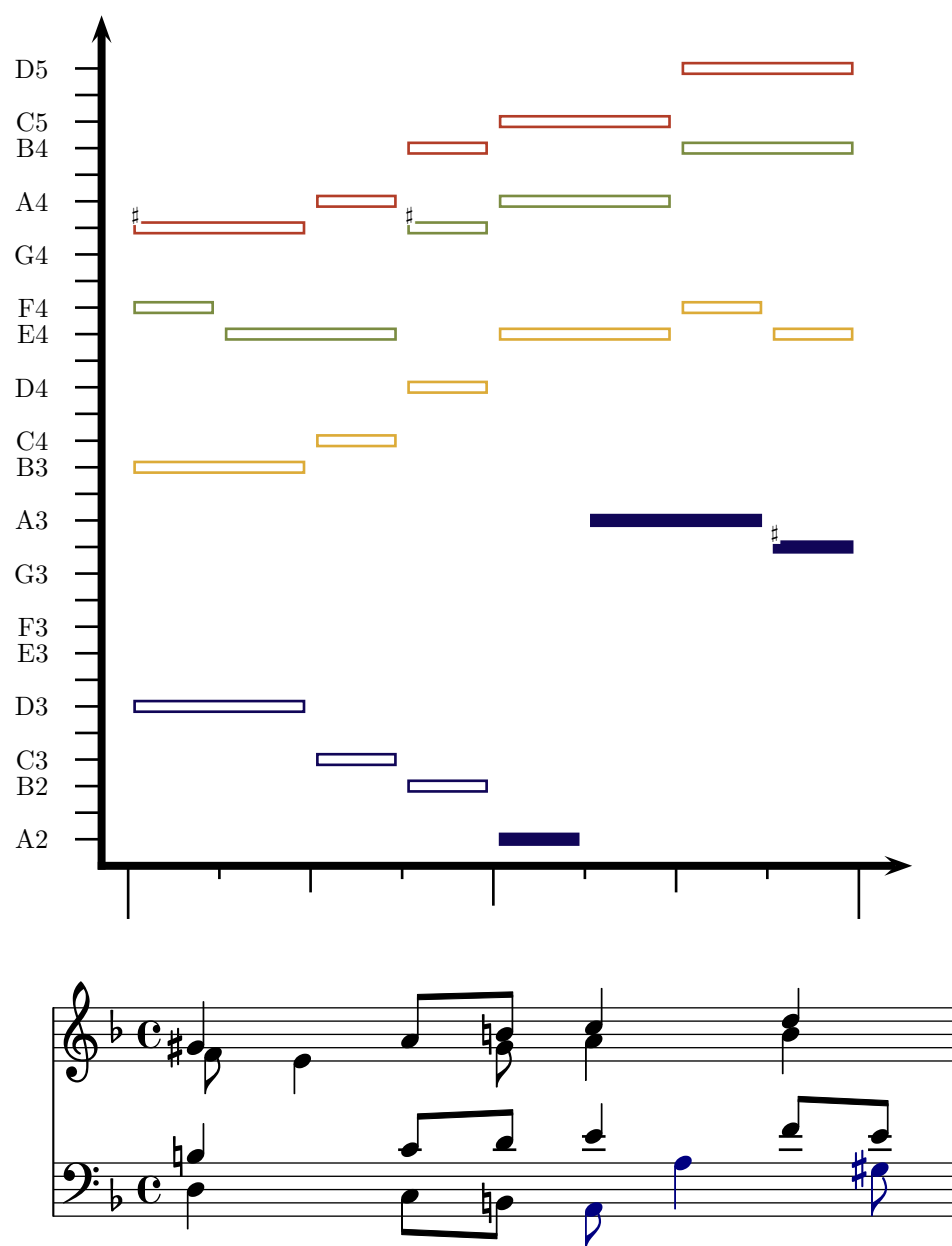


Figure F.6: BWV 277 (bar 1) with a highlighted compensated leap pattern

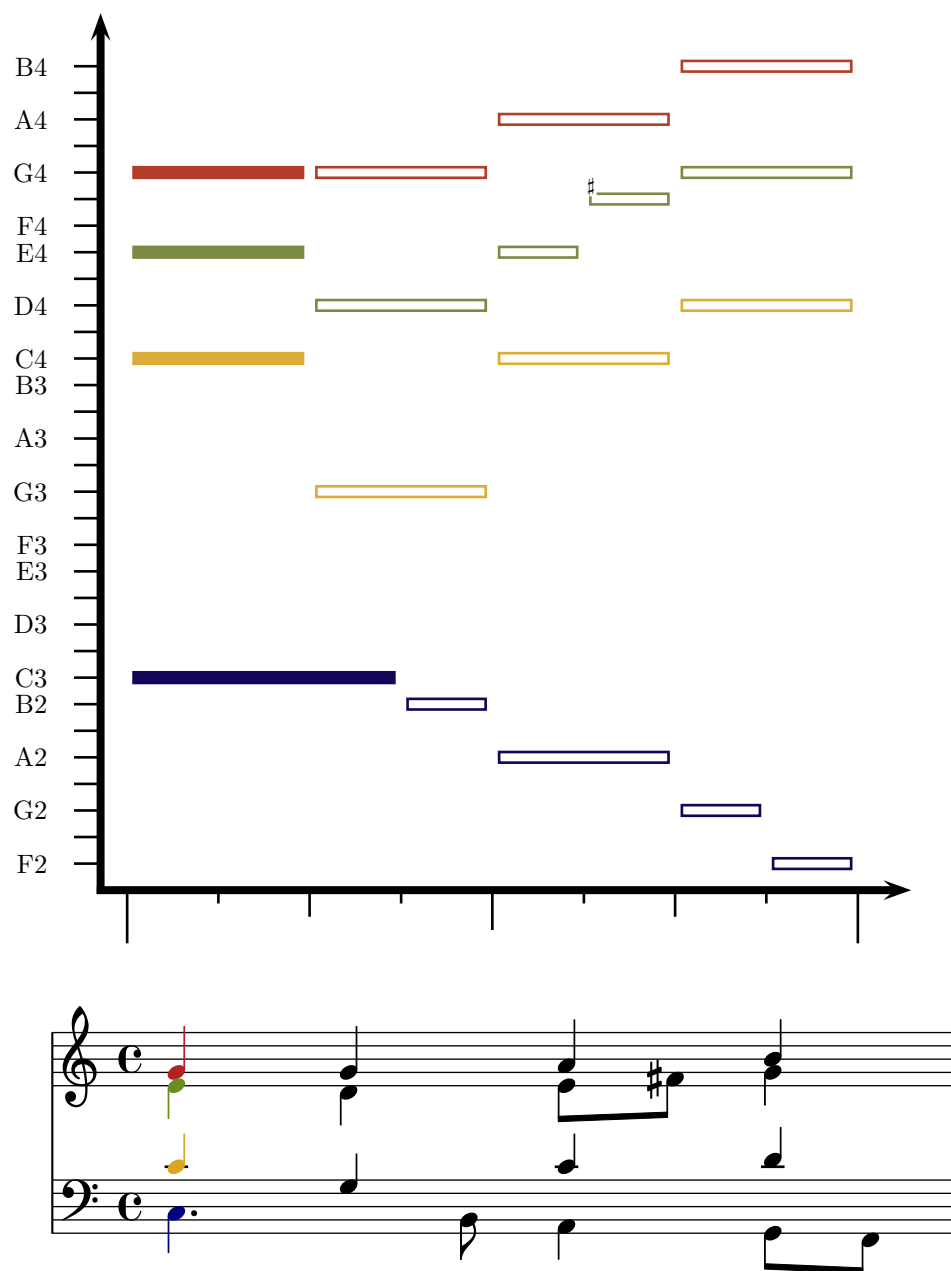


Figure F.7: BWV 292 (bar 1) with a highlighted I/i block chord pattern

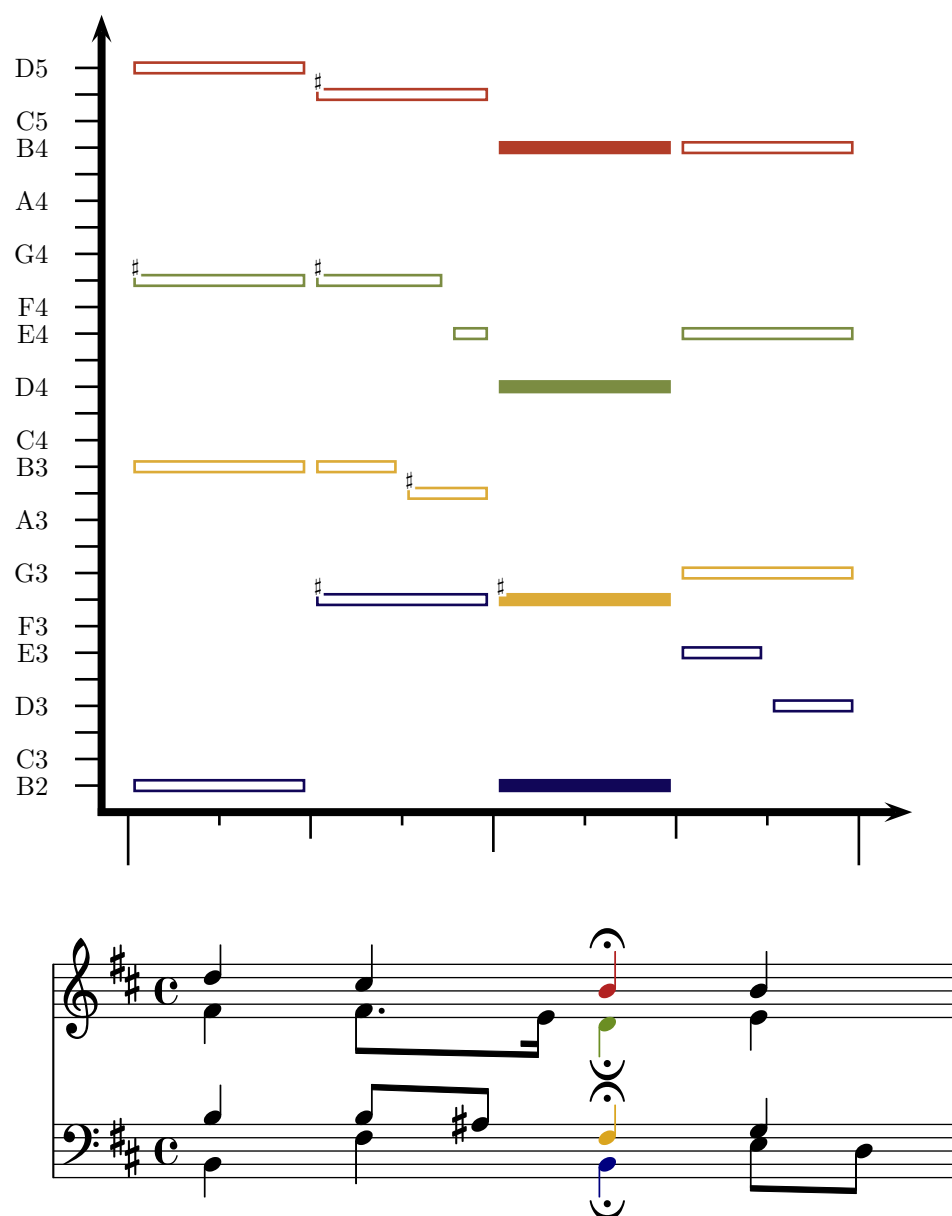


Figure F.8: BWV 335 (bar 4) with a highlighted I/i block chord pattern

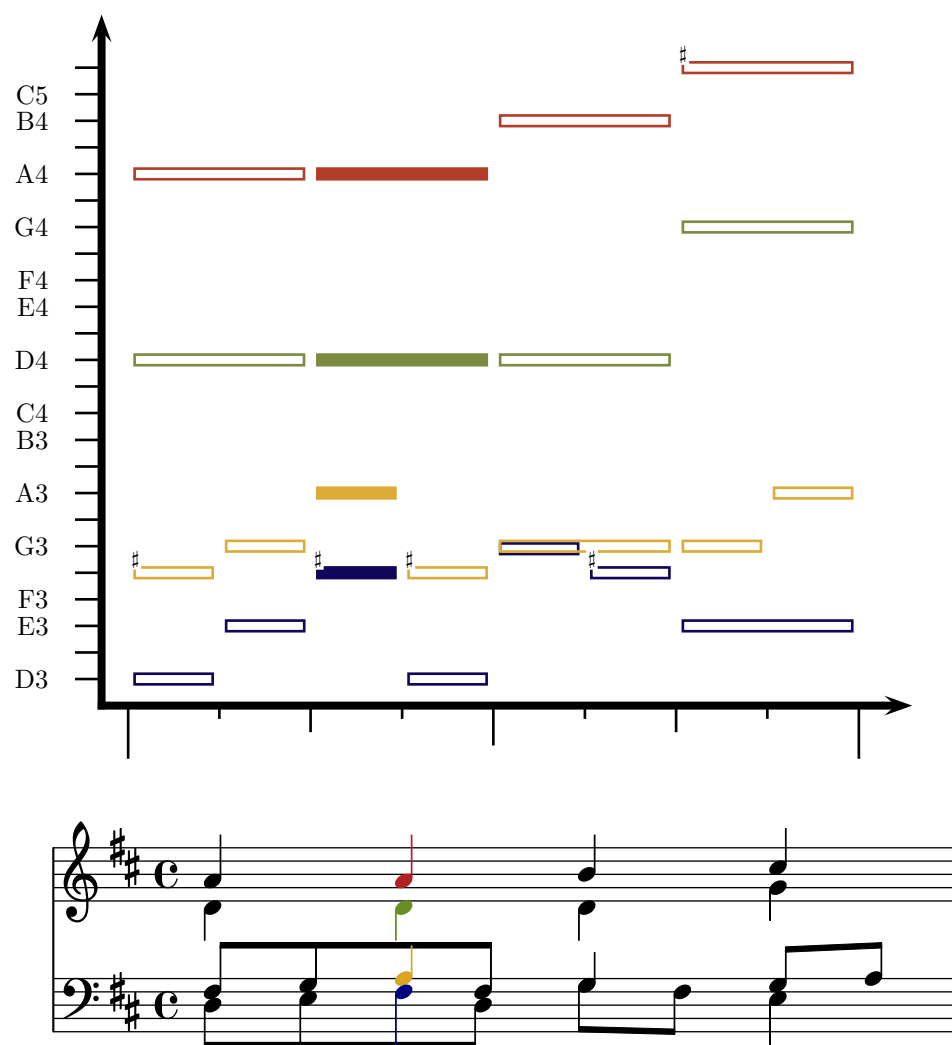


Figure F.9: BWV 415 (bar 1) with a highlighted I/i block chord pattern

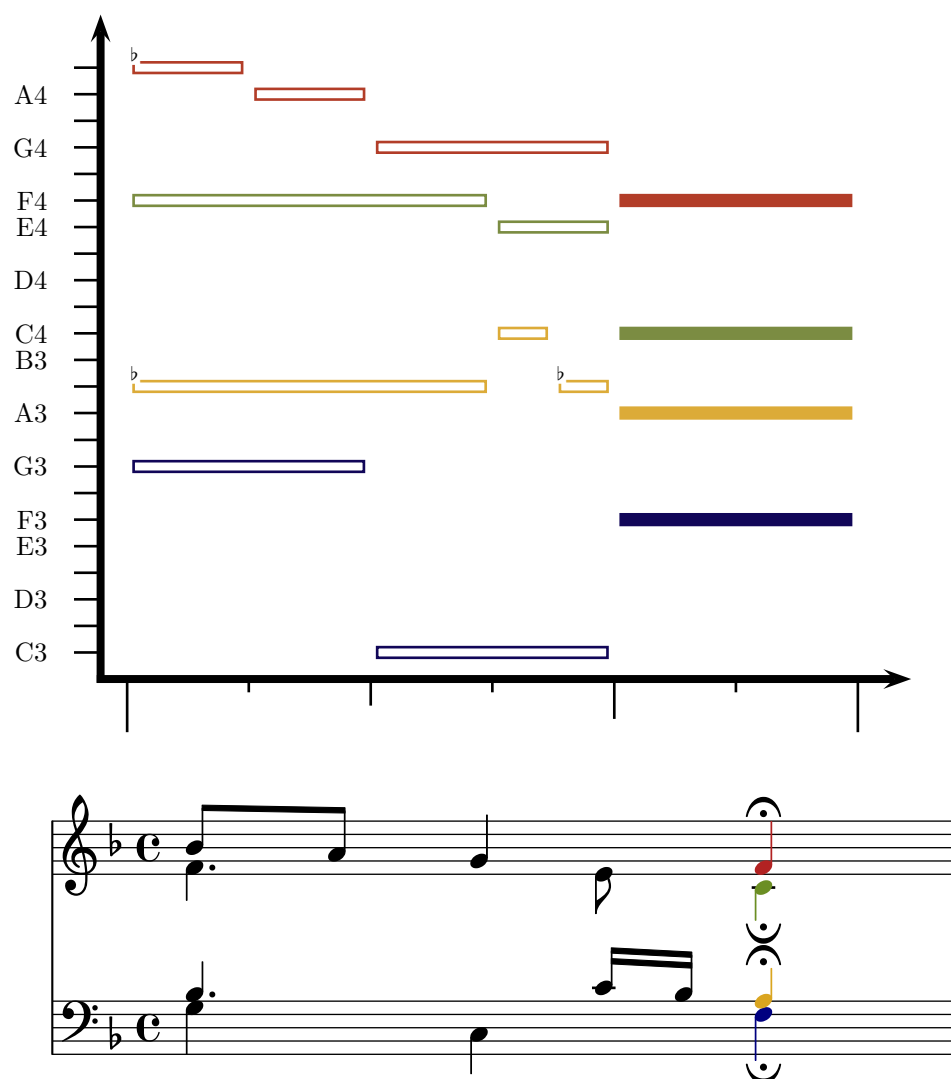


Figure F.10: BWV 438 (bar 8) with a highlighted I/i block chord pattern

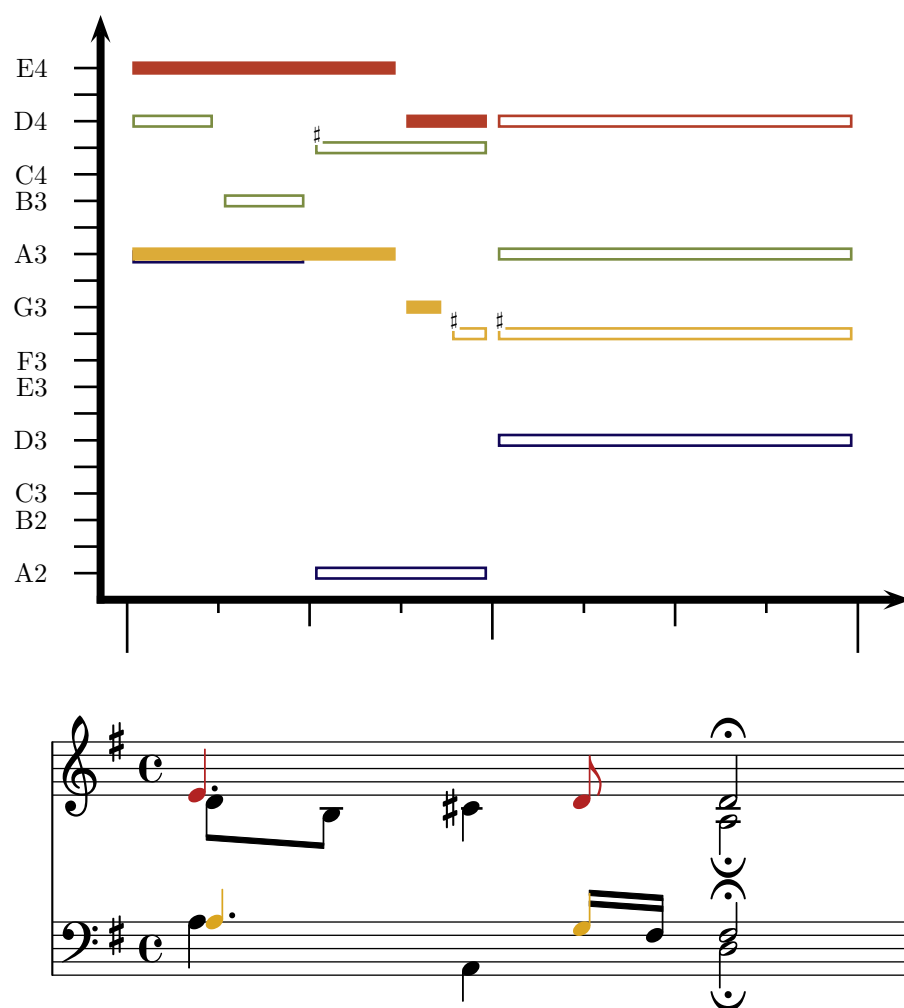


Figure F.11: BWV 263 (bar 6) with a highlighted parallel fifth pattern

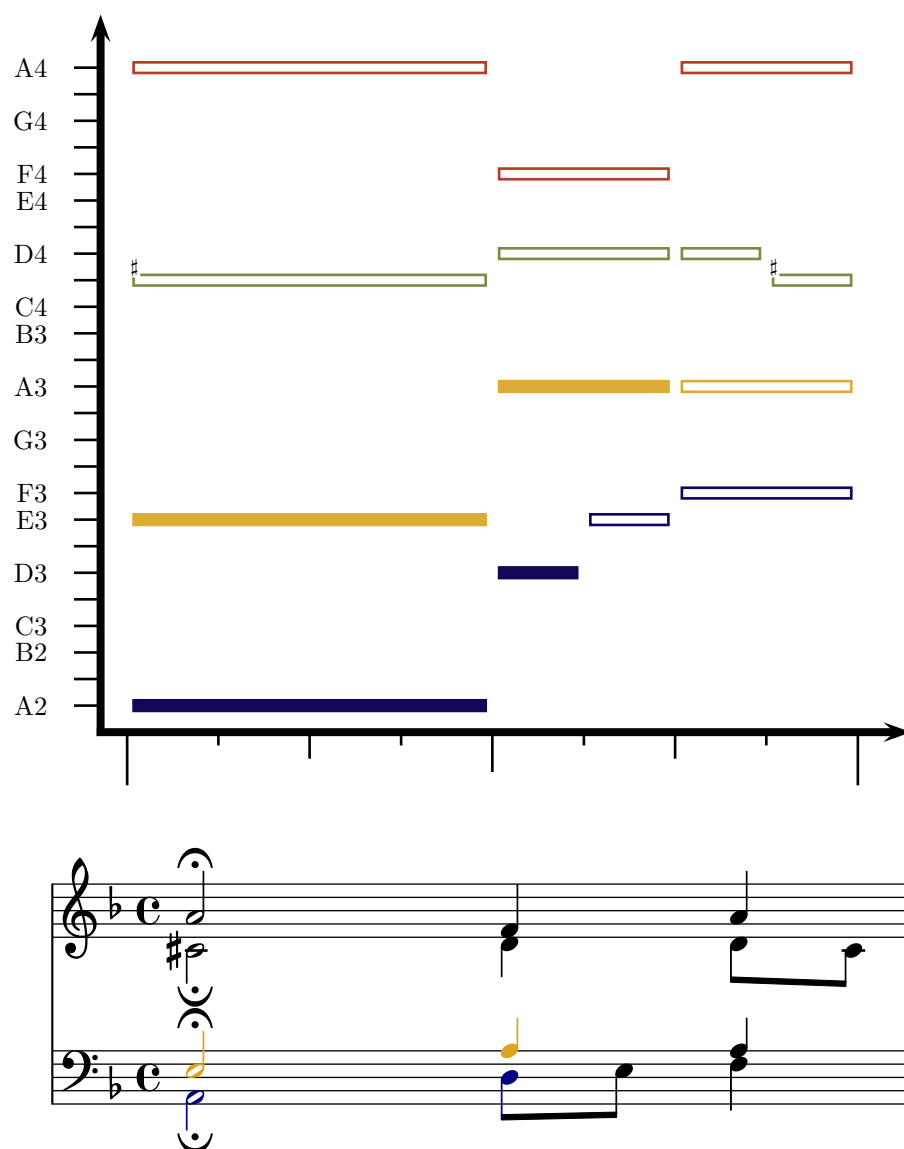


Figure F.12: BWV 301 (bar 3) with a highlighted parallel fifth pattern

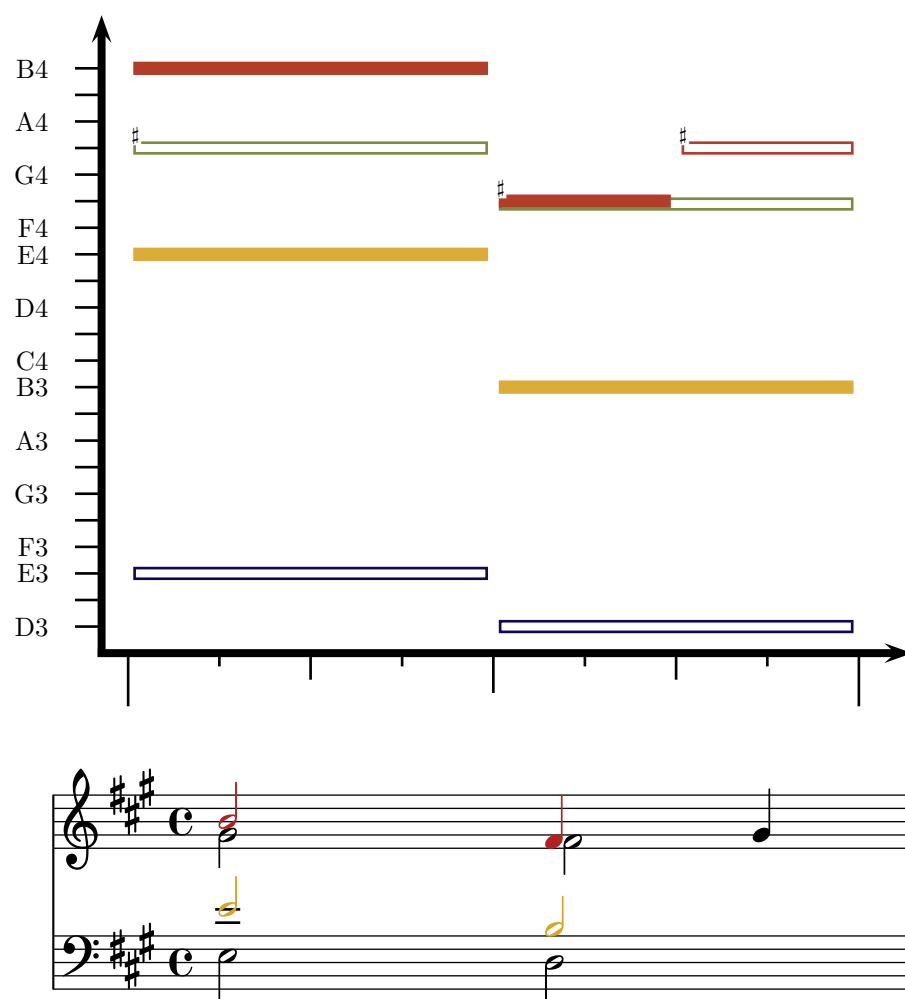


Figure F.13: BWV 323 (bar 8) with a highlighted parallel fifth pattern

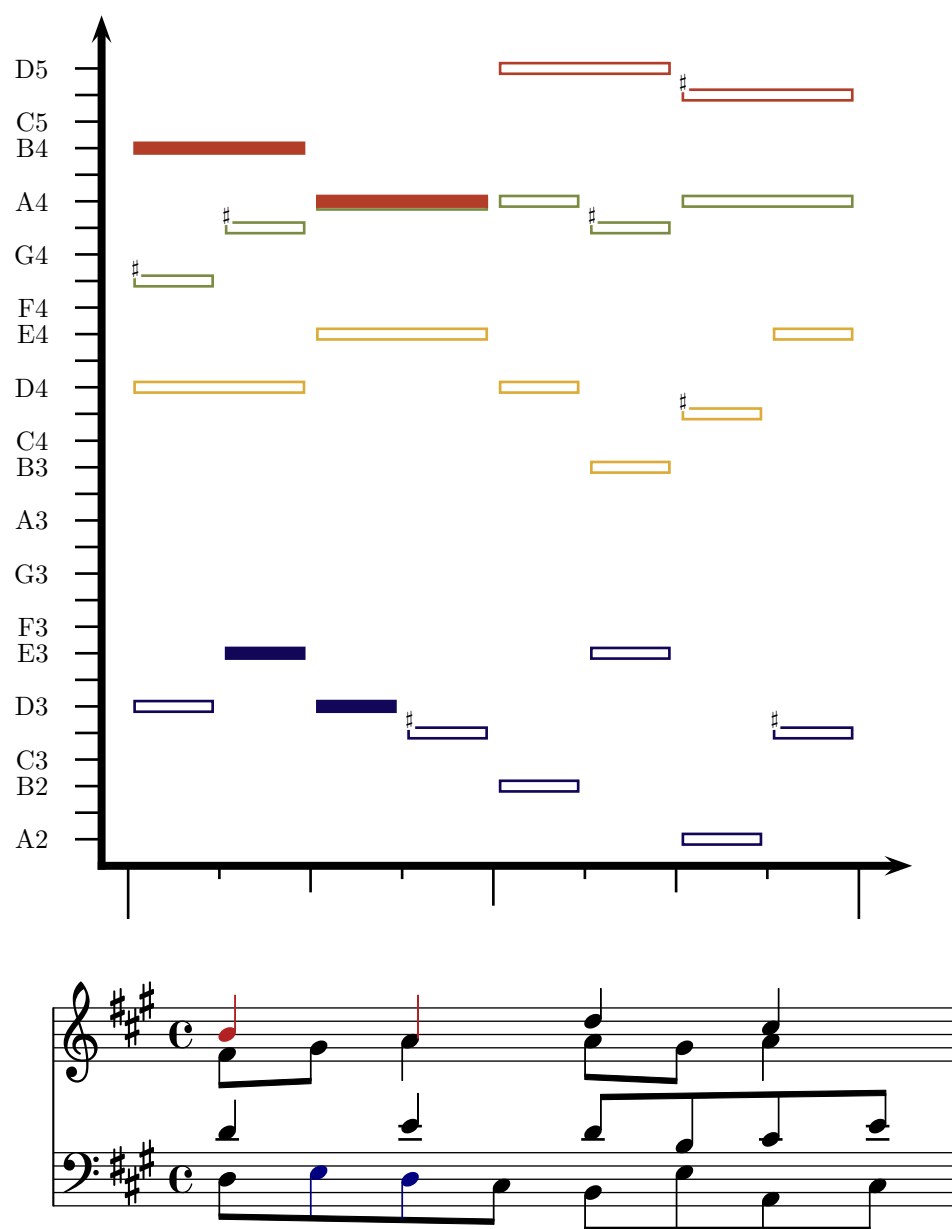


Figure F.14: BWV 355 (bar 15) with a highlighted parallel fifth pattern

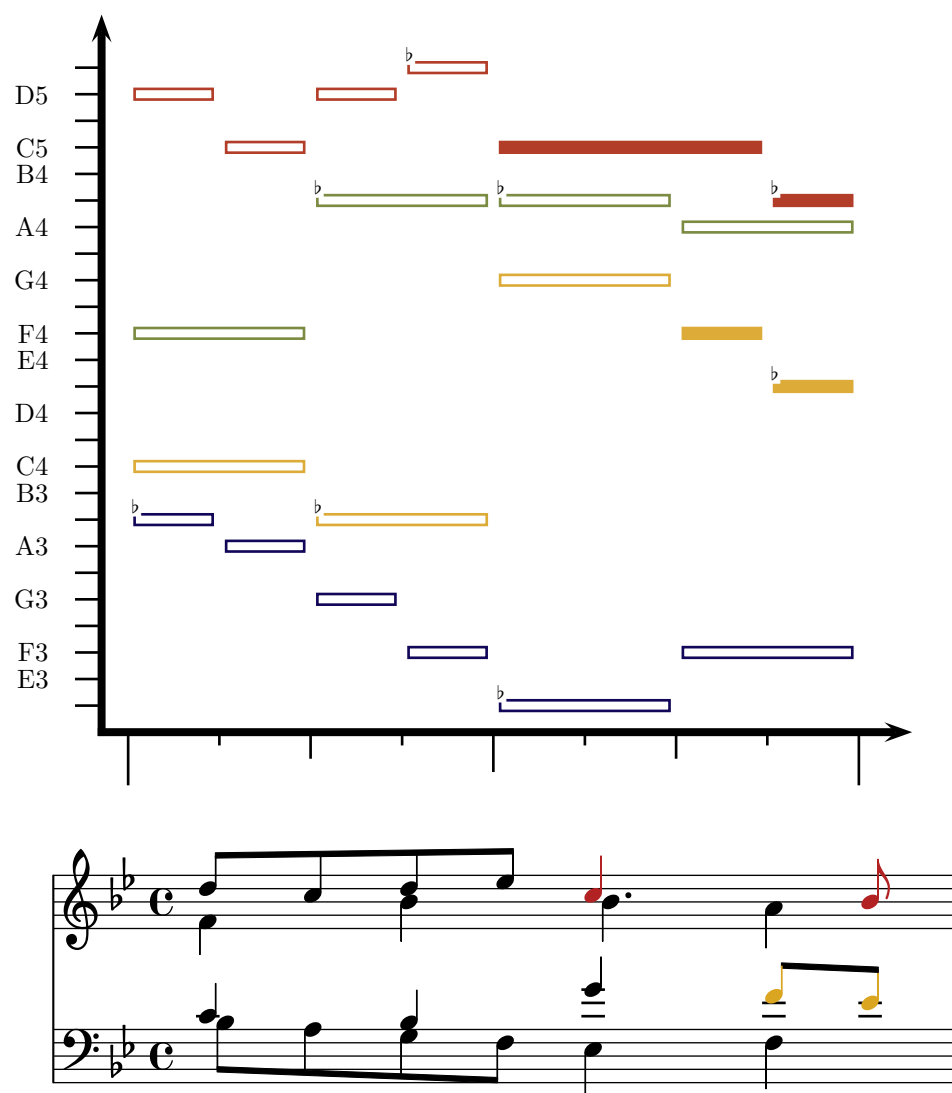


Figure F.15: BWV 361 (bar 12) with a highlighted parallel fifth pattern

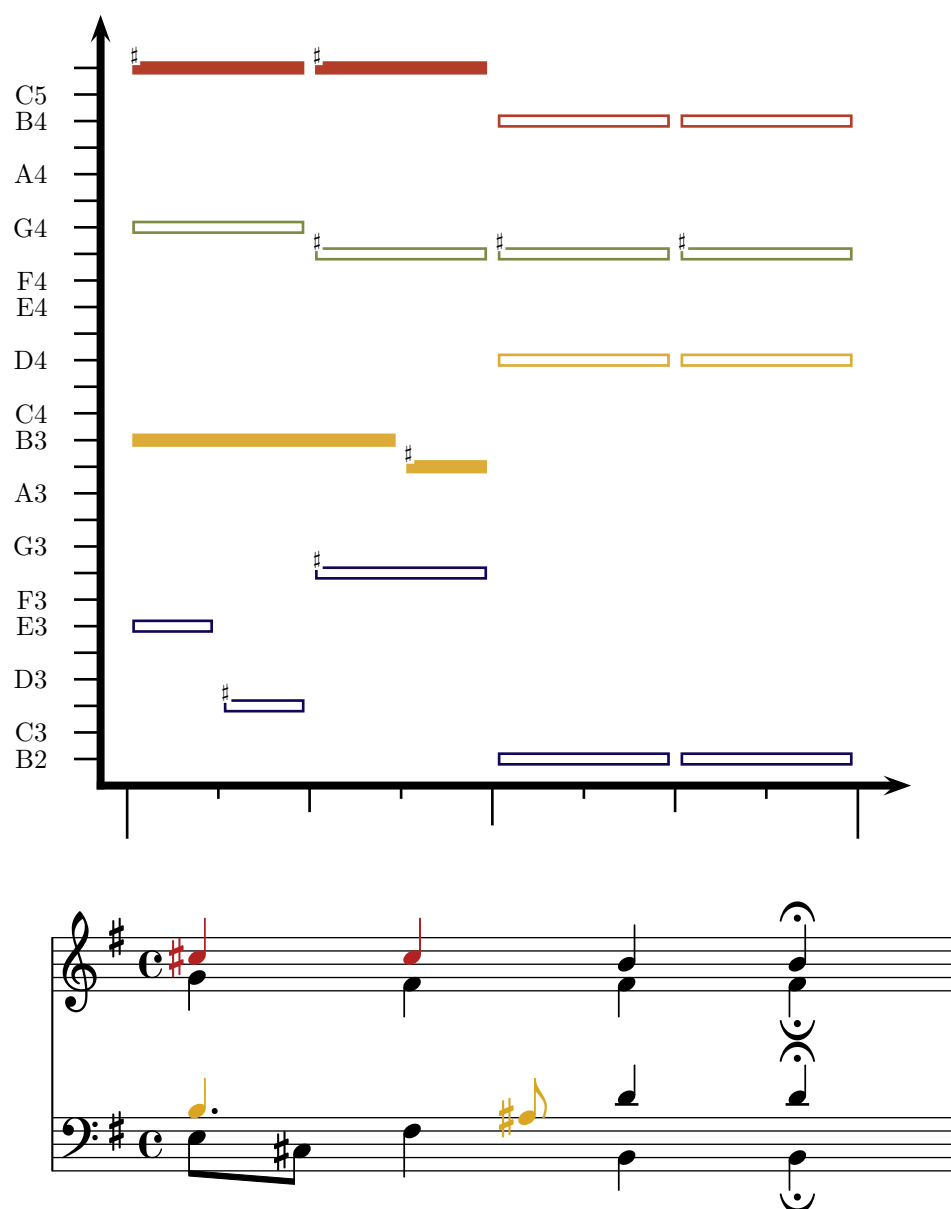


Figure F.16: BWV 259 (bar 8) with a highlighted suspension pattern

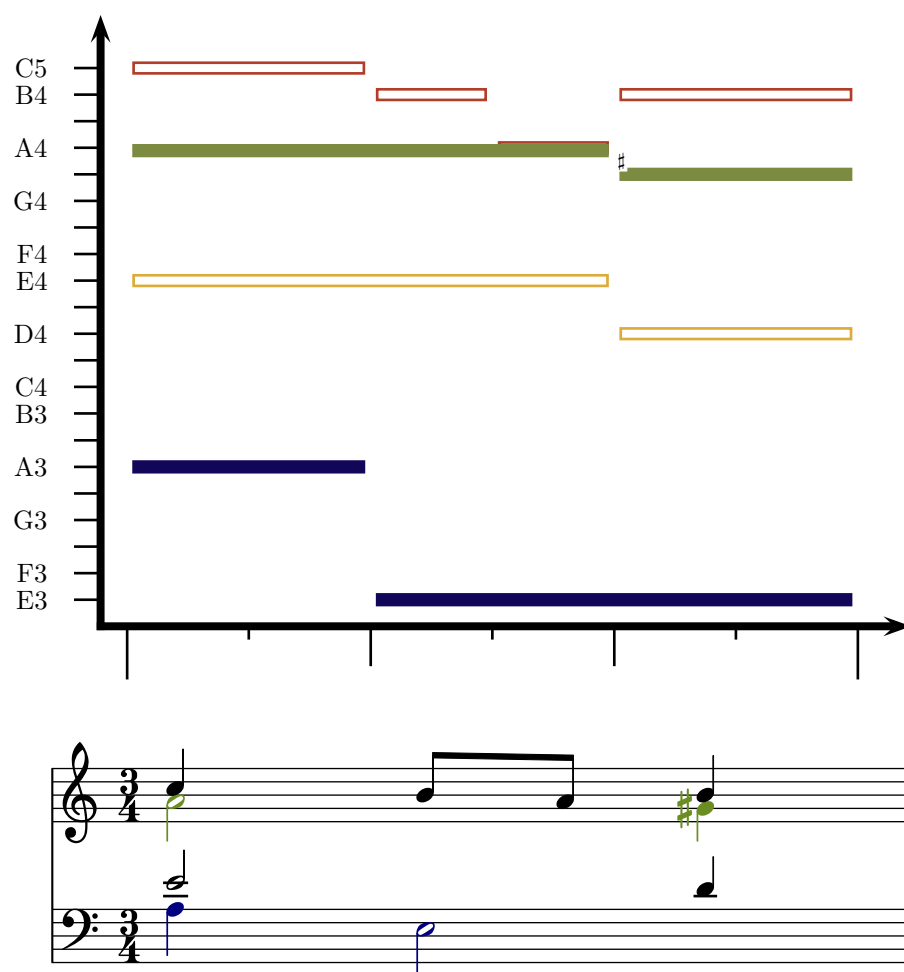


Figure F.17: BWV 390 (bar 12) with a highlighted suspension pattern

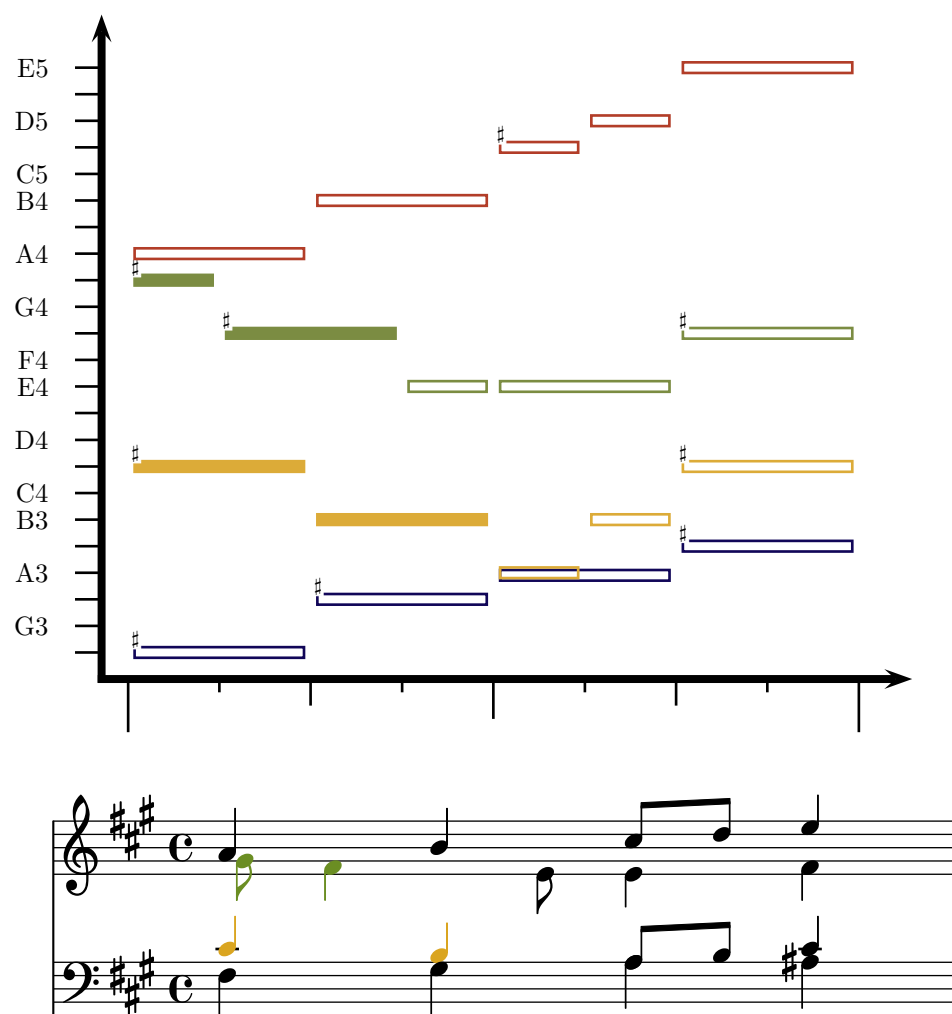


Figure F.18: BWV 393 (bar 7) with a highlighted suspension pattern

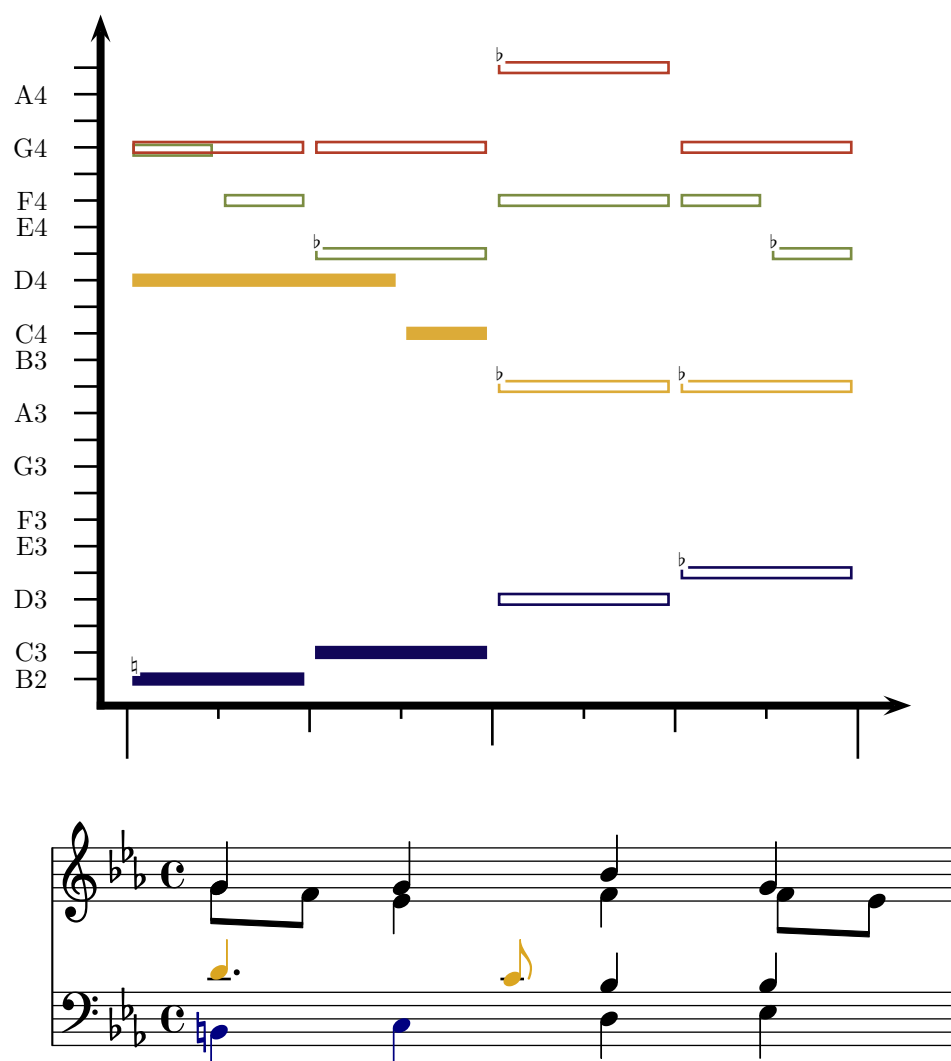


Figure F.19: BWV 285 (bar 1) with a highlighted suspension pattern

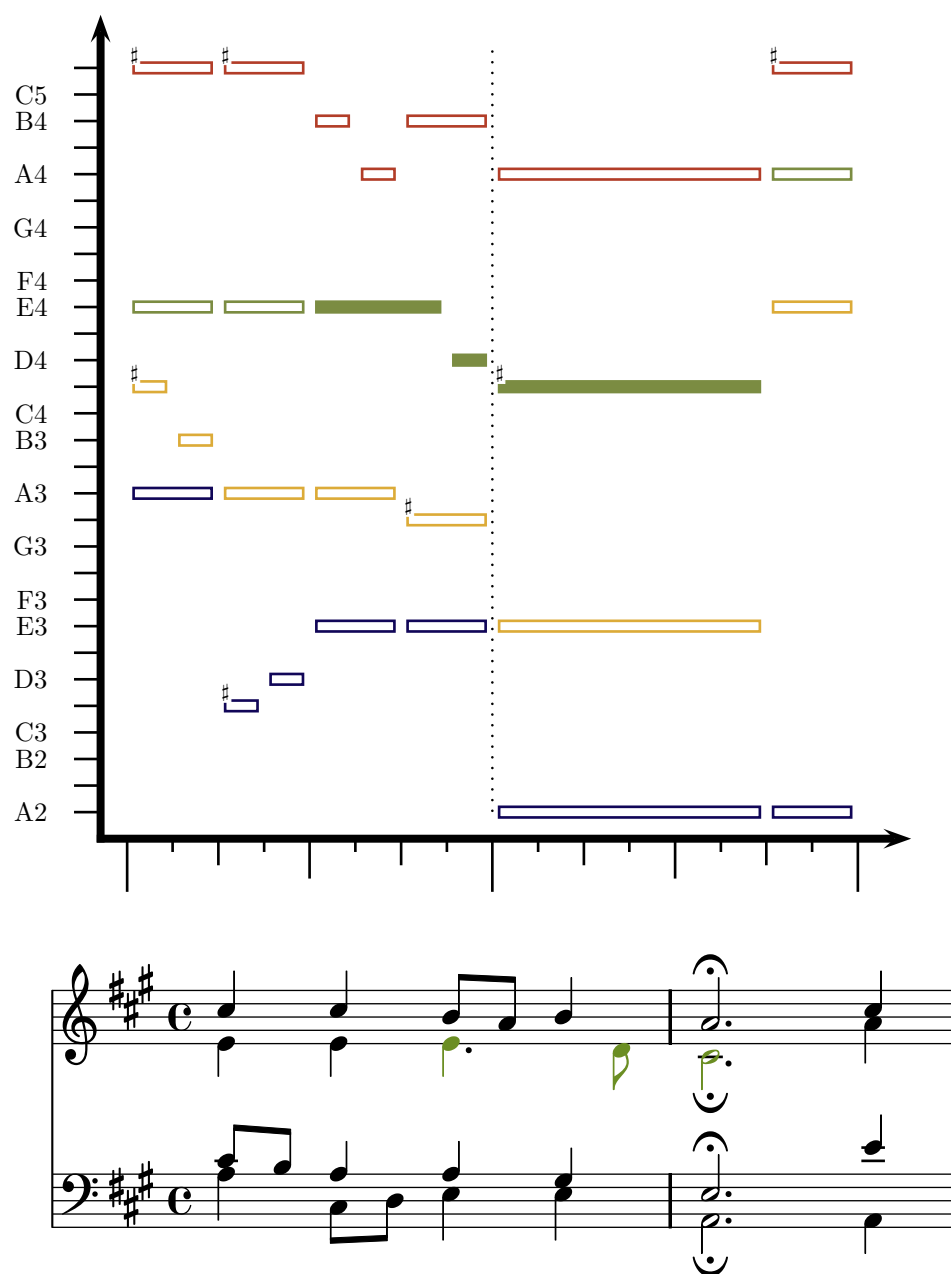


Figure F.20: BWV 253 (bar 4) with a highlighted passing tone pattern

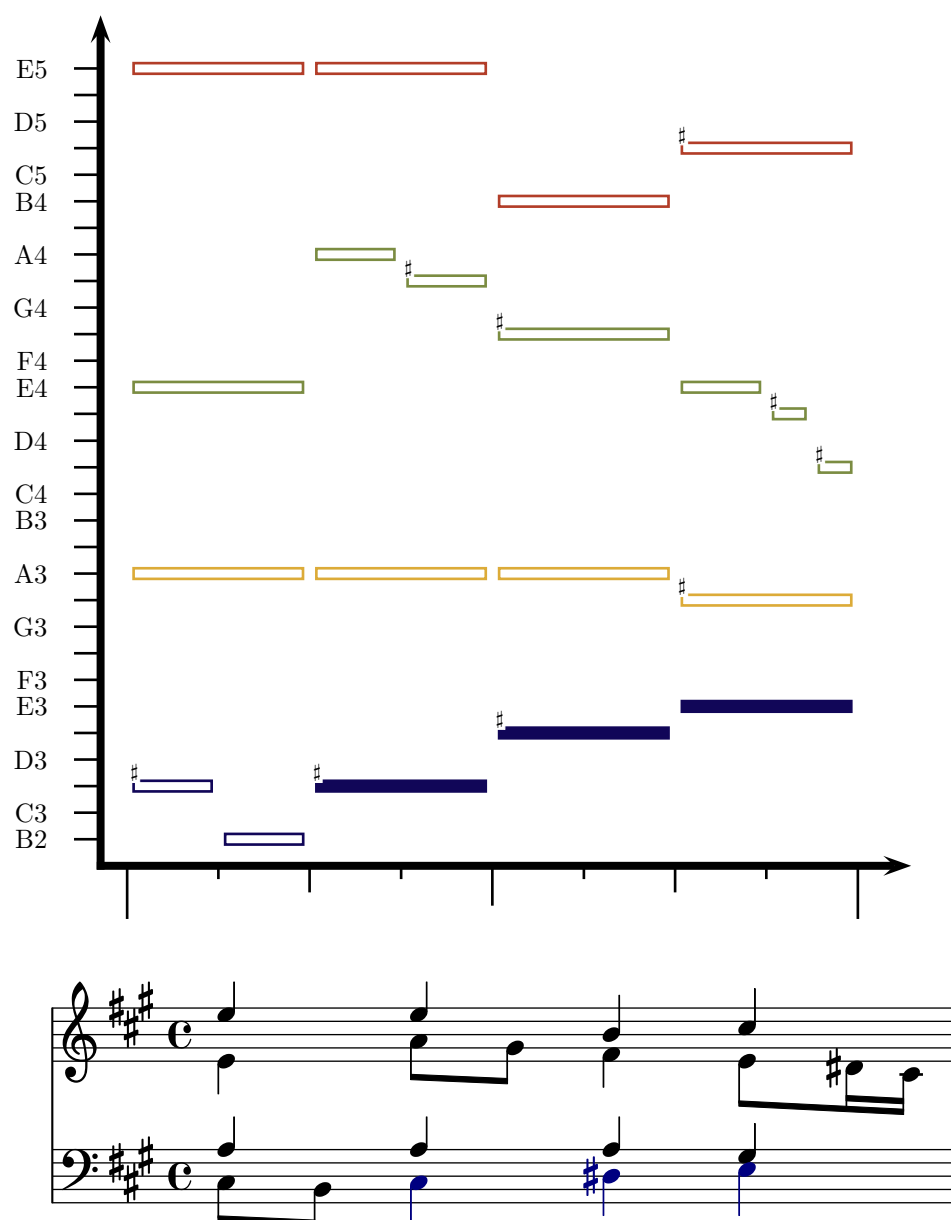


Figure F.21: BWV 395 (bar 9) with a highlighted passing tone pattern

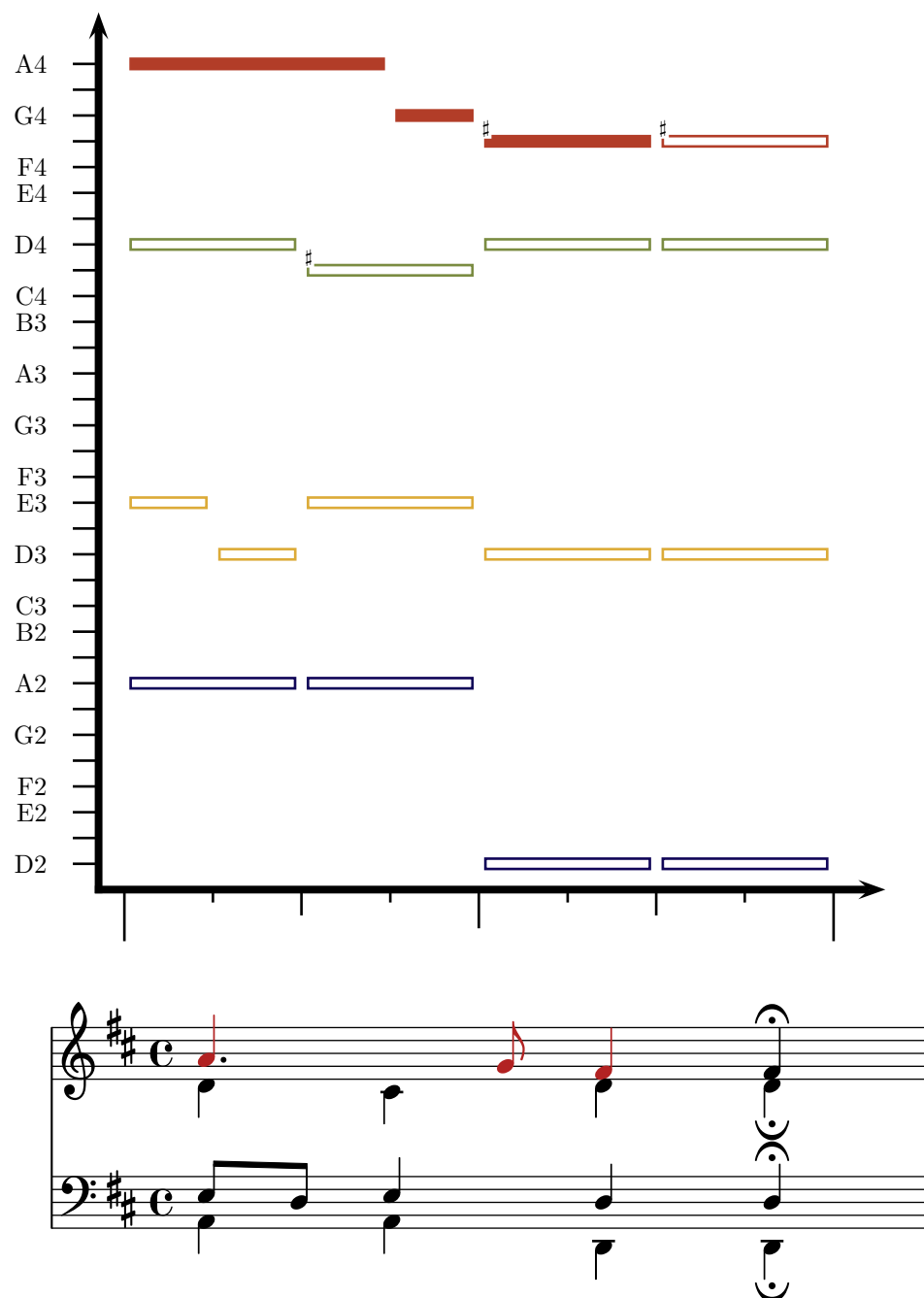


Figure F.22: BWV 262 (bar 2) with a highlighted passing tone pattern

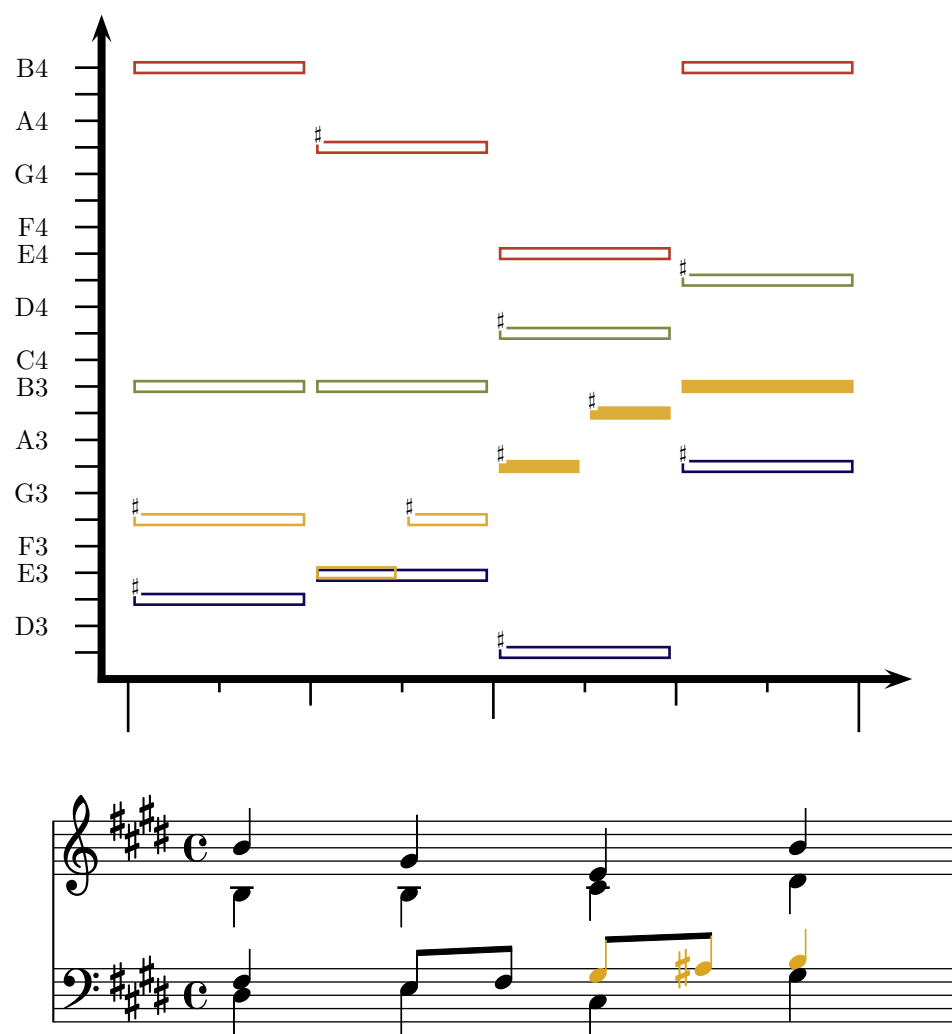


Figure F.23: BWV 436 (bar 1) with a highlighted passing tone pattern

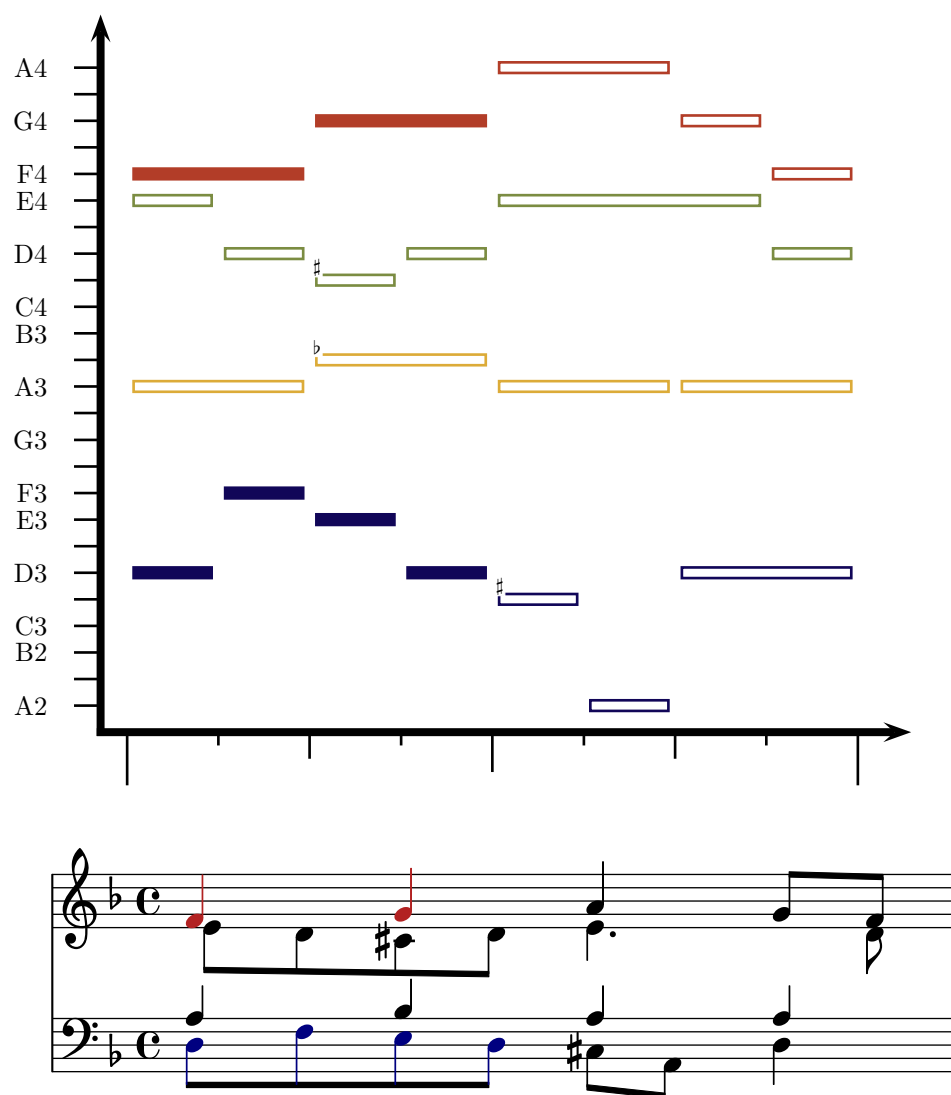


Figure F.24: BWV 277 (bar 3) with a highlighted two-voice counterpoint pattern

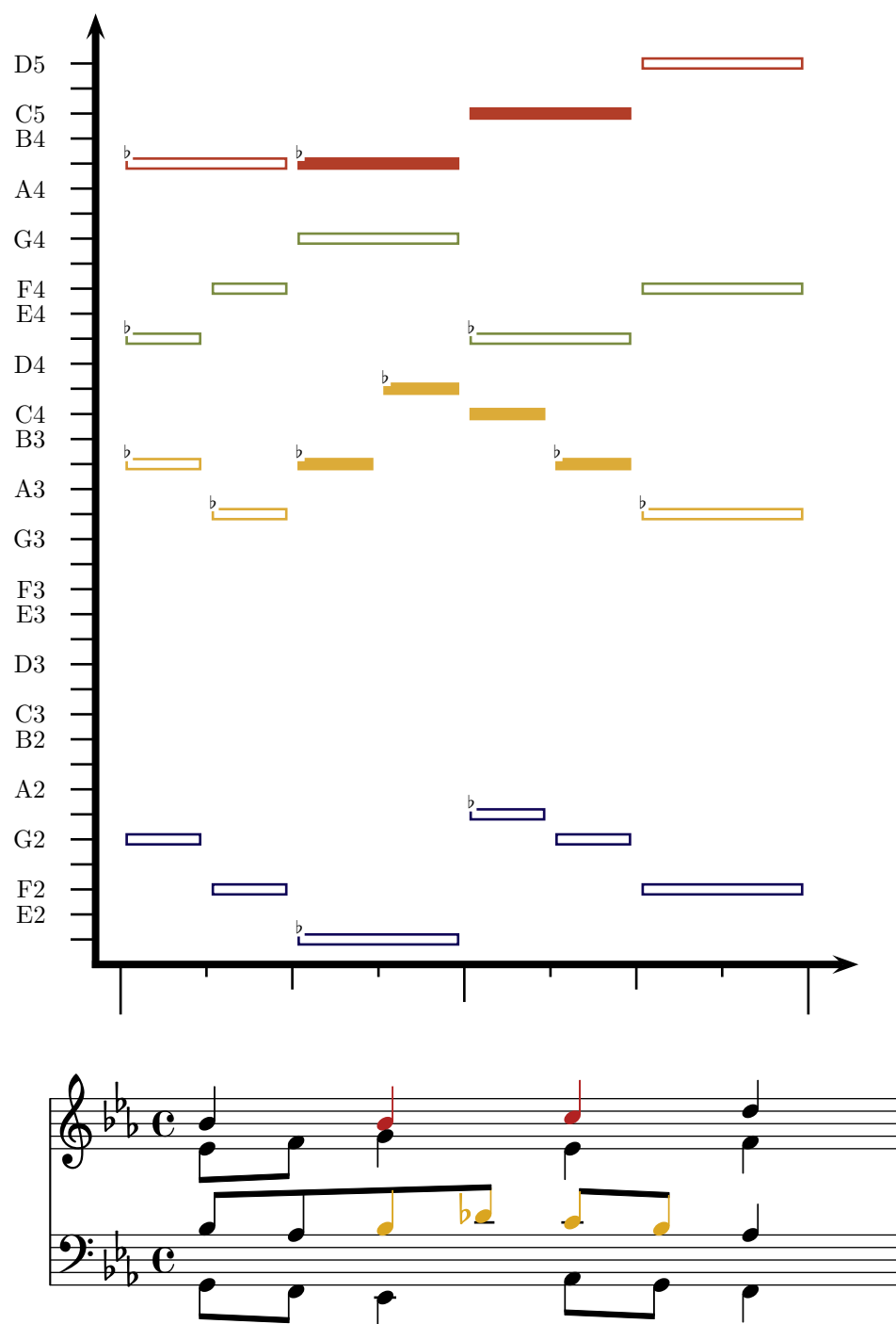


Figure F.25: BWV 285 (bar 9) with a highlighted two-voice counterpoint pattern

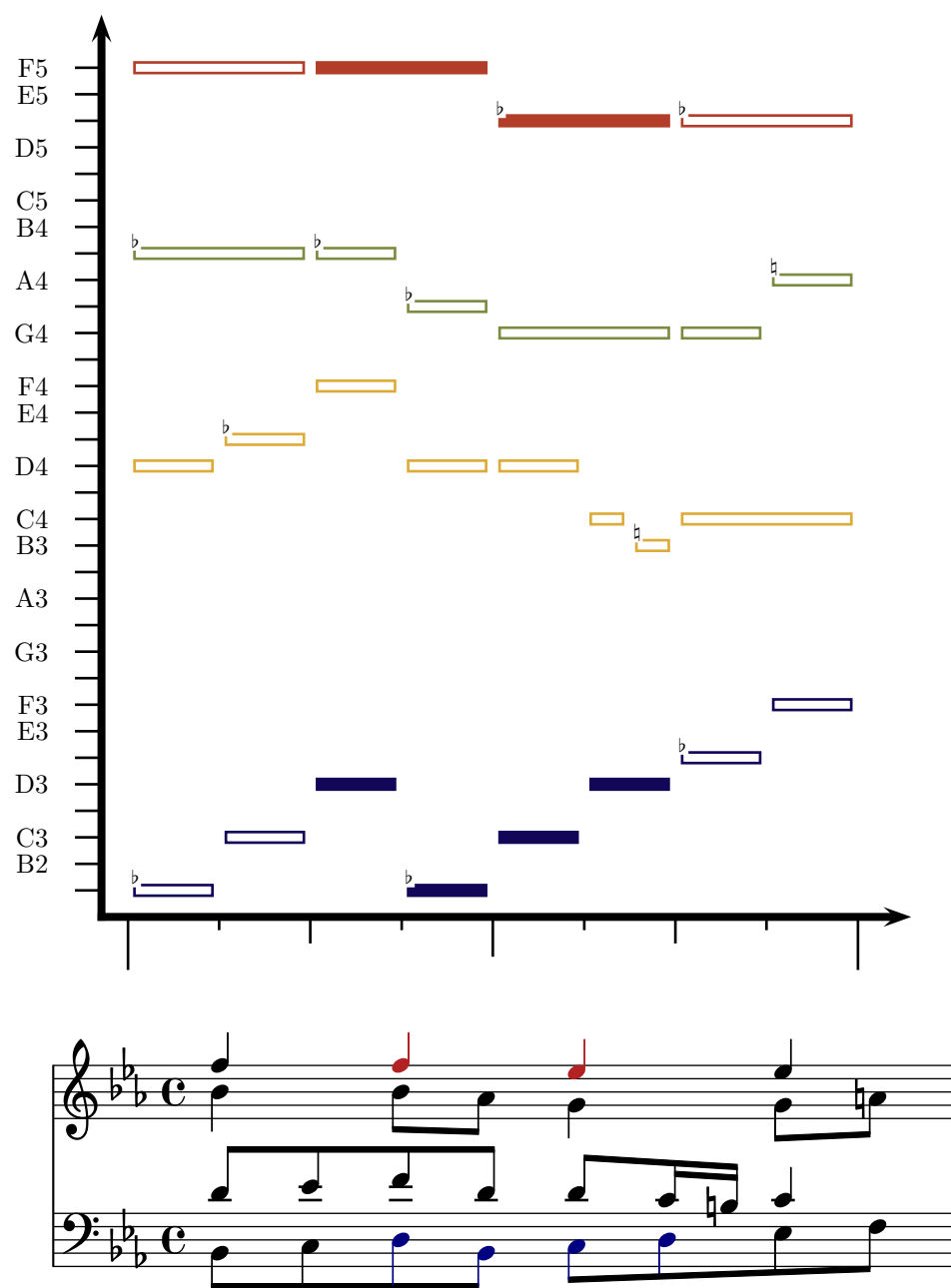


Figure F.26: BWV 380 (bar 3) with a highlighted two-voice counterpoint pattern

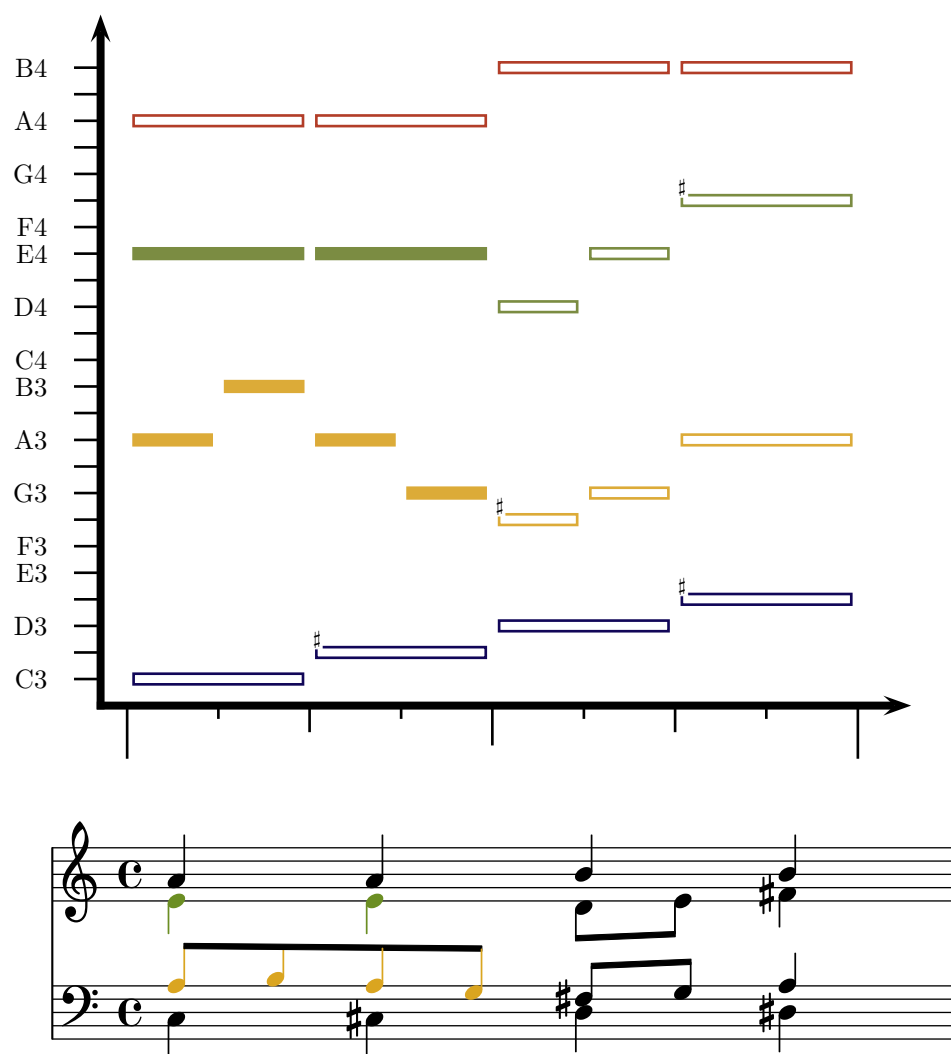


Figure F.27: BWV 418 (bar 7) with a highlighted two-voice counterpoint pattern

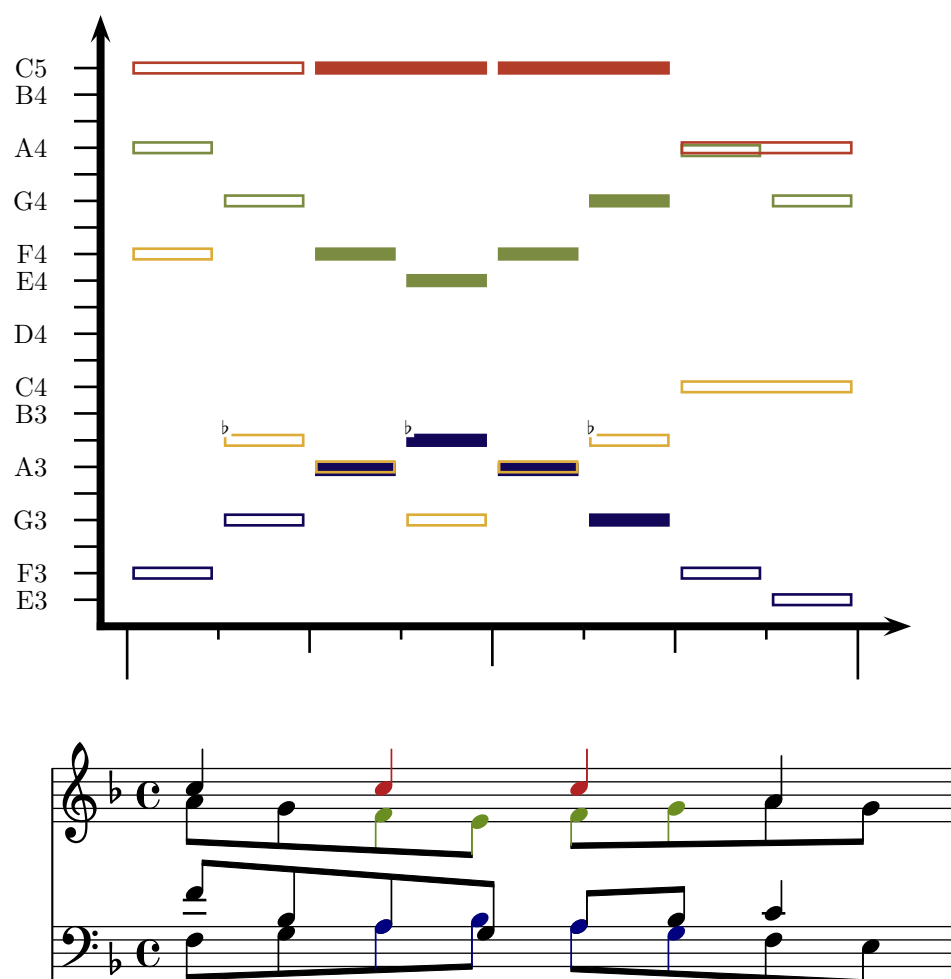


Figure F.28: BWV 374 (bar 7) with a highlighted two-voice counterpoint pattern

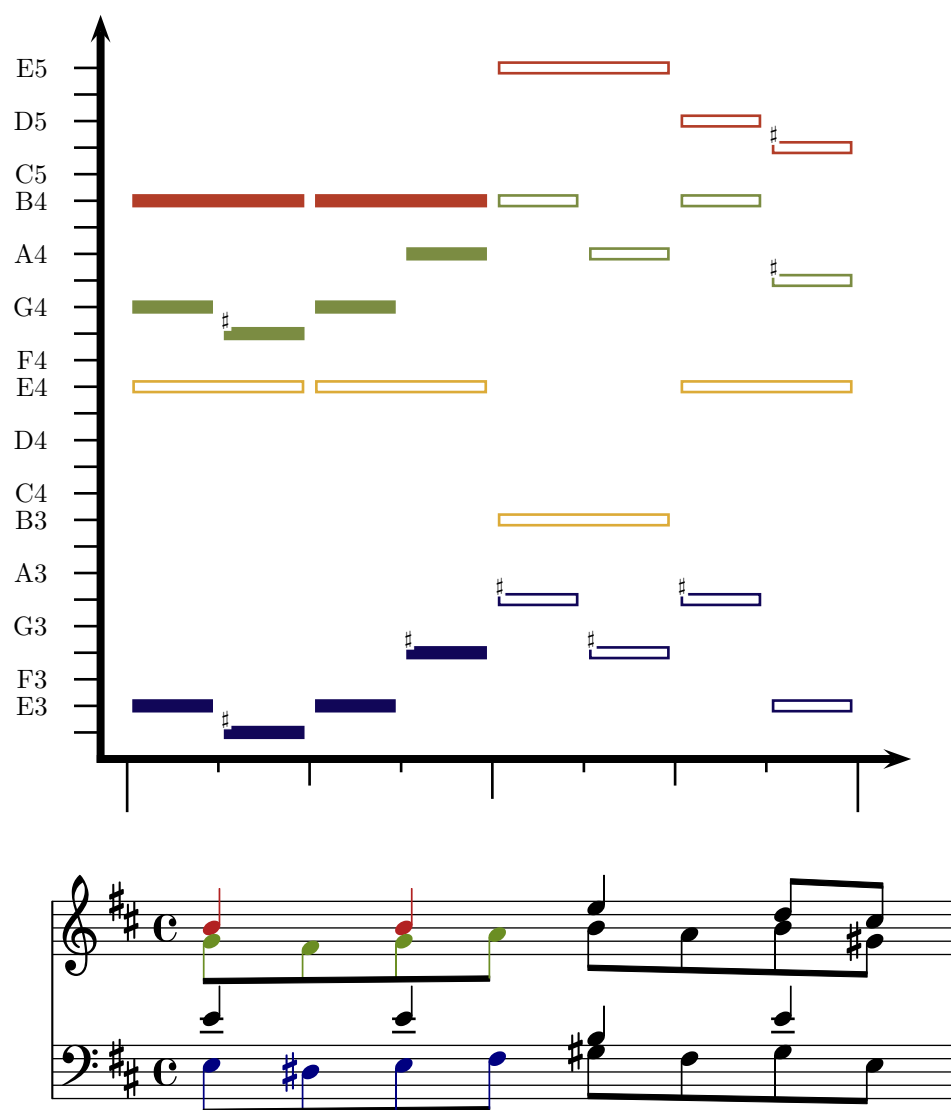


Figure F.29: BWV 398 (bar 11) with a highlighted two-voice counterpoint pattern

F.4 Musical excerpts for Chapter 6

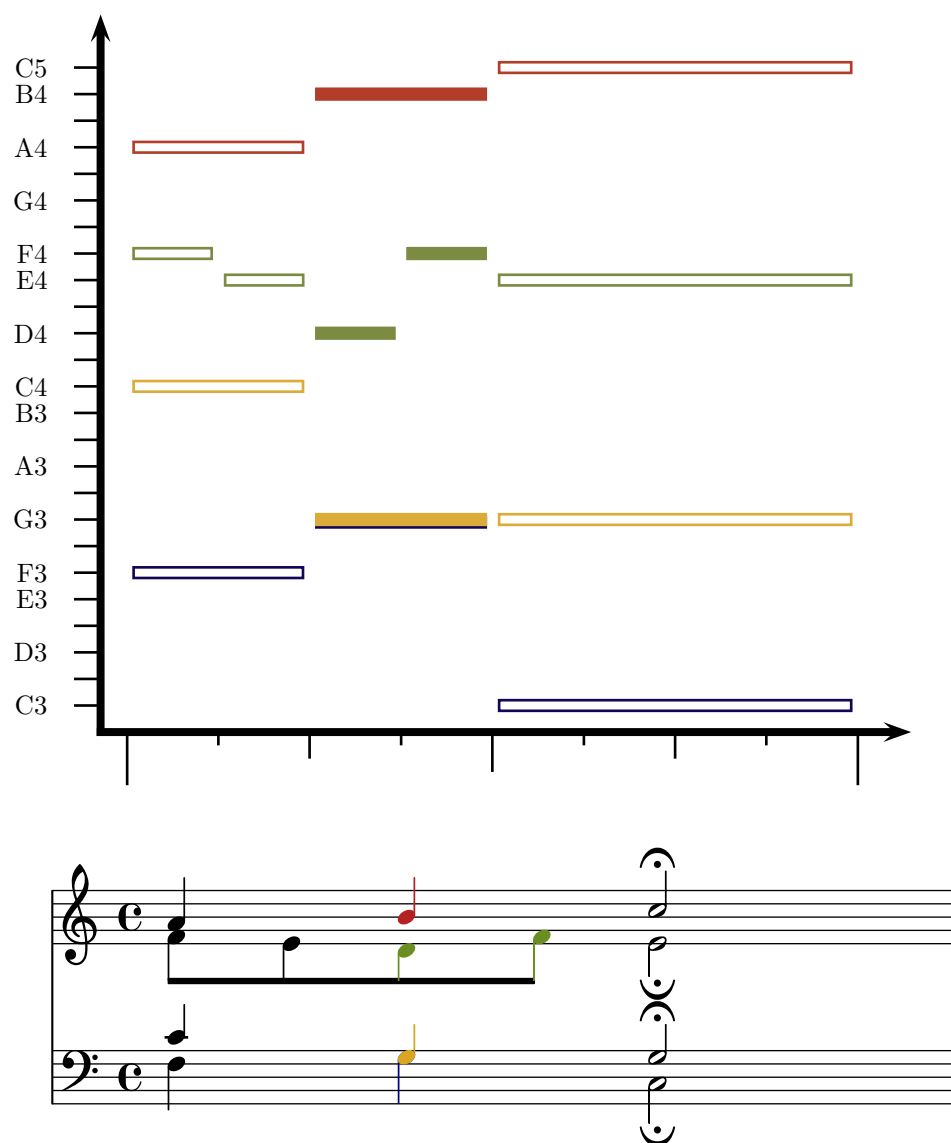


Figure F.30: BWV 284 (bar 15) with a highlighted dislocated chord pattern

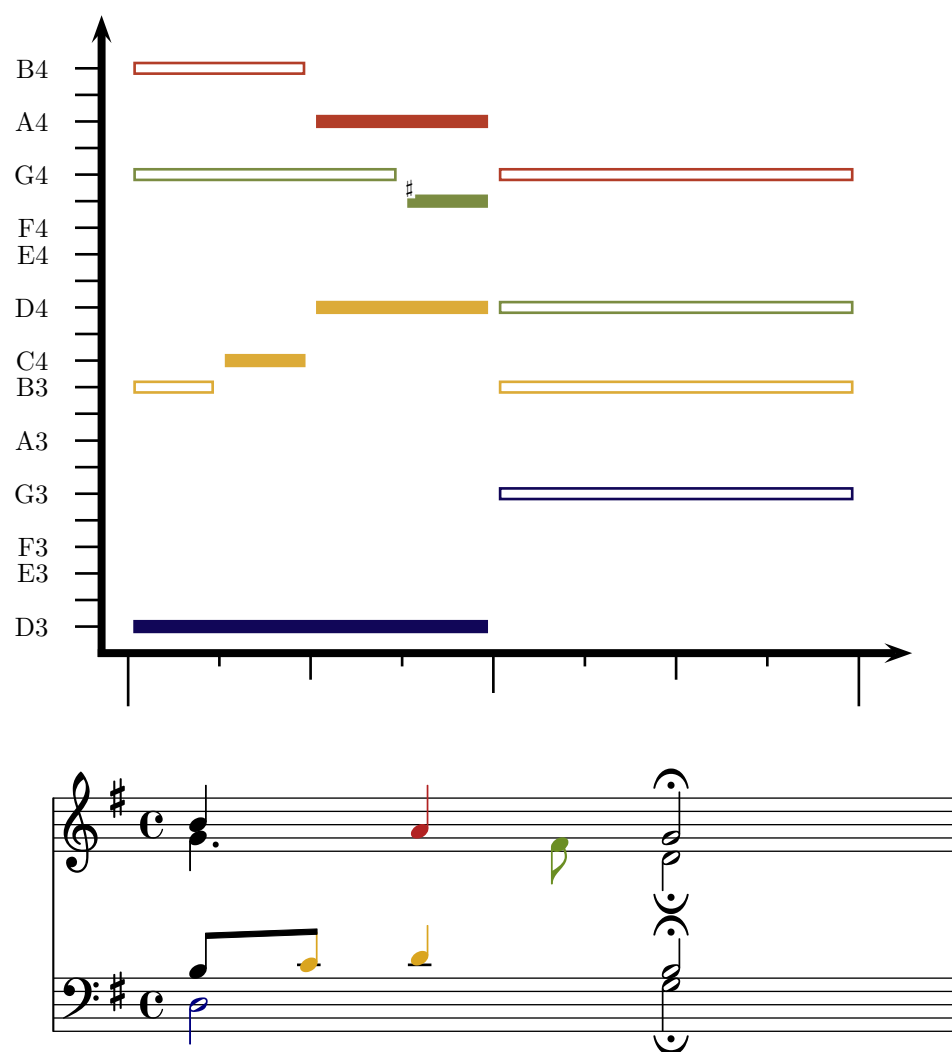


Figure F.31: BWV 318 (bar 13) with a highlighted dislocated chord pattern

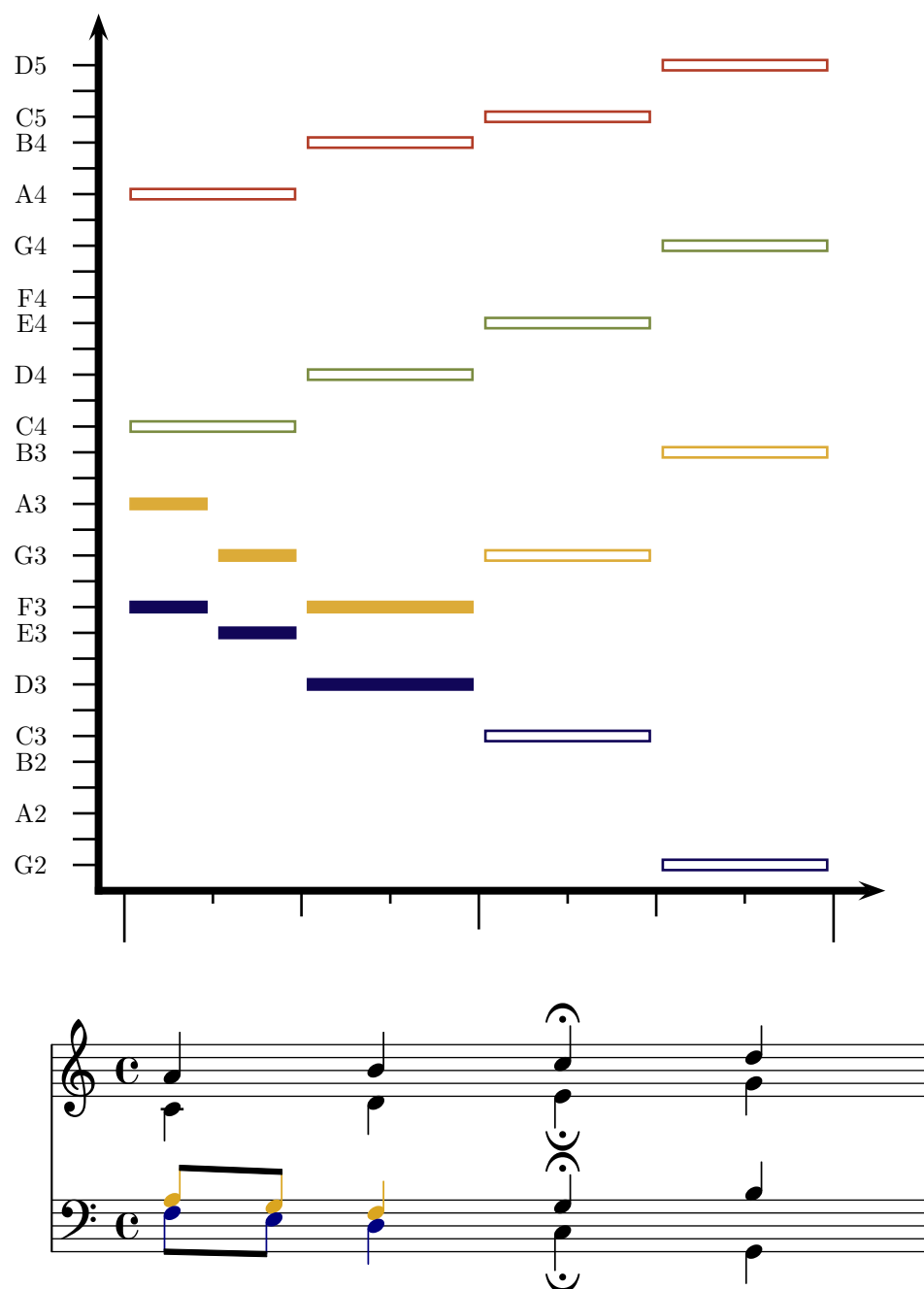


Figure F.32: BWV 255 (bar 2) with a highlighted layered passing tones pattern

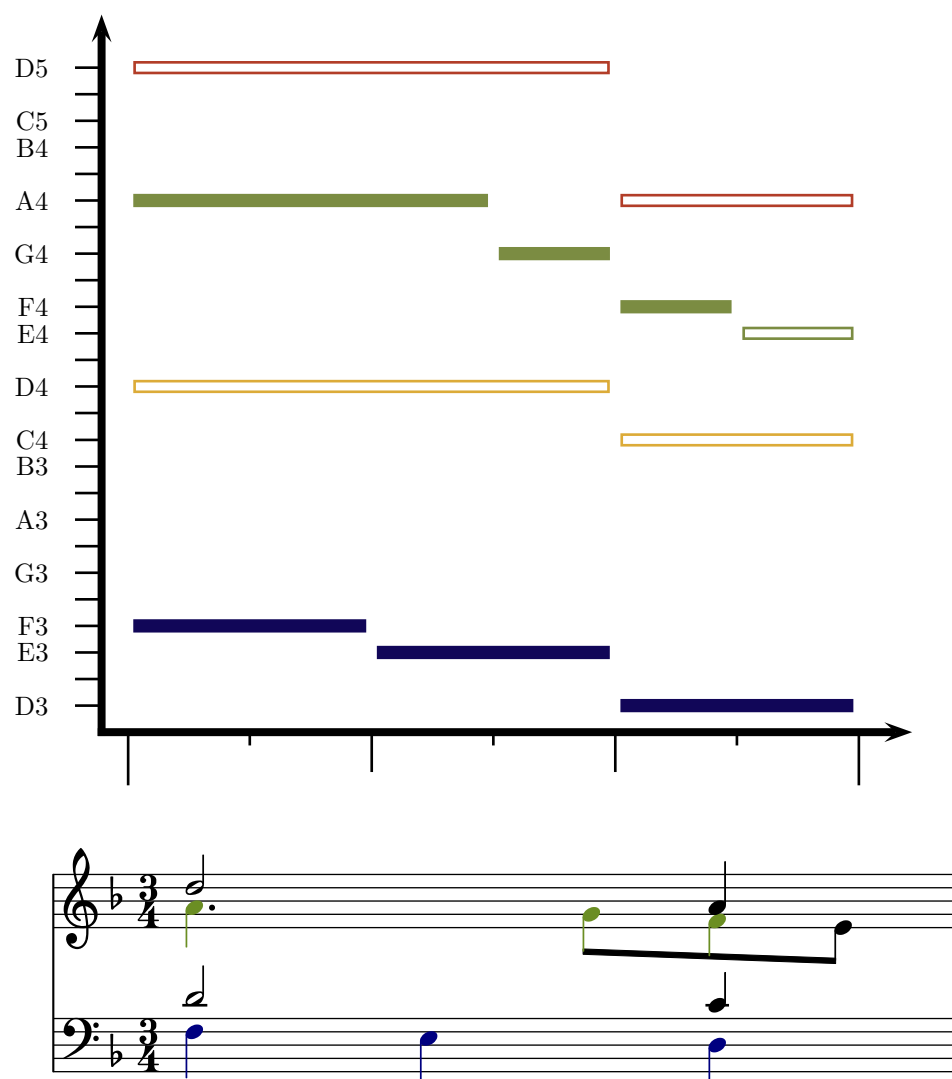


Figure F.33: BWV 320 (bar 19) with a highlighted layered passing tones pattern

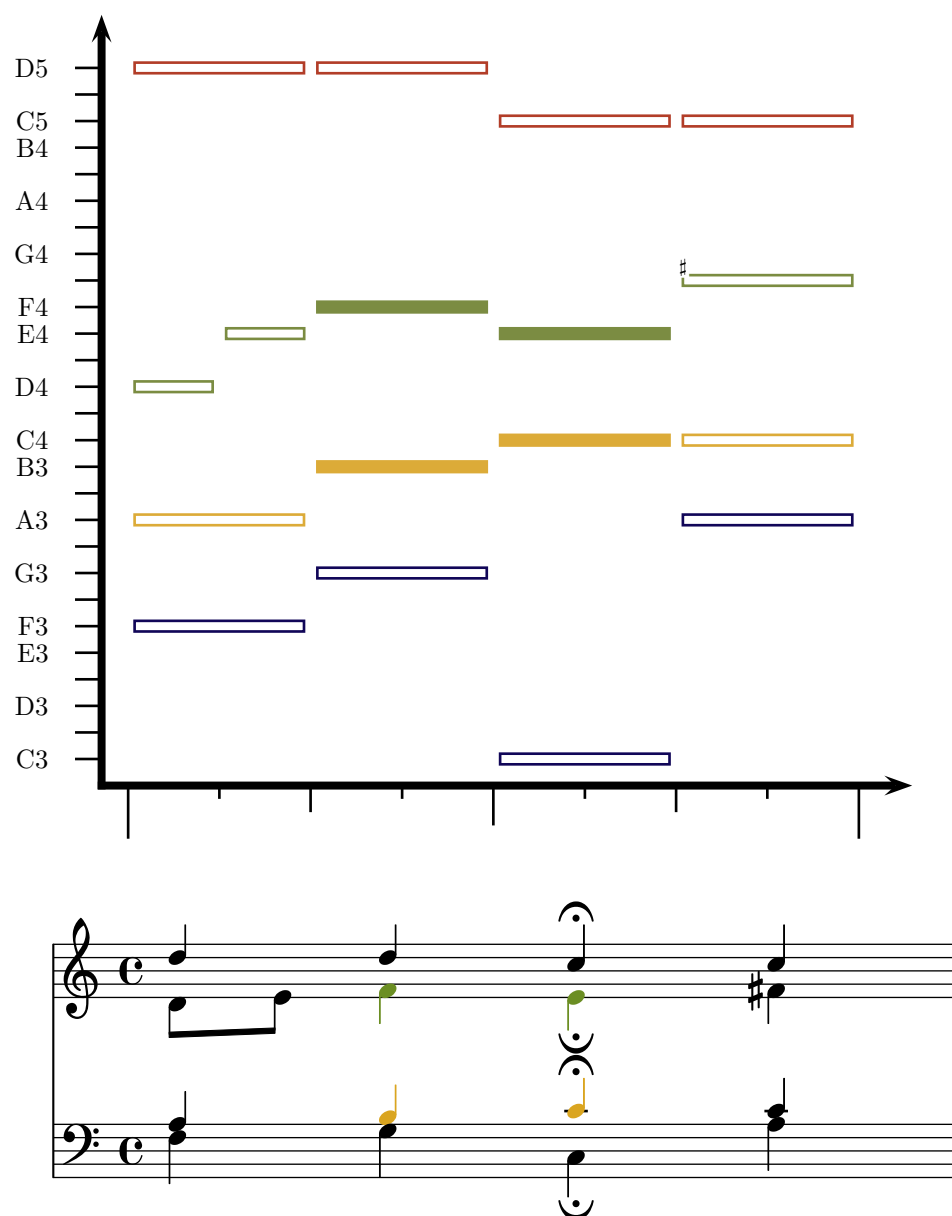


Figure F.34: BWV 257 (bar 2) with a highlighted tritone resolution pattern

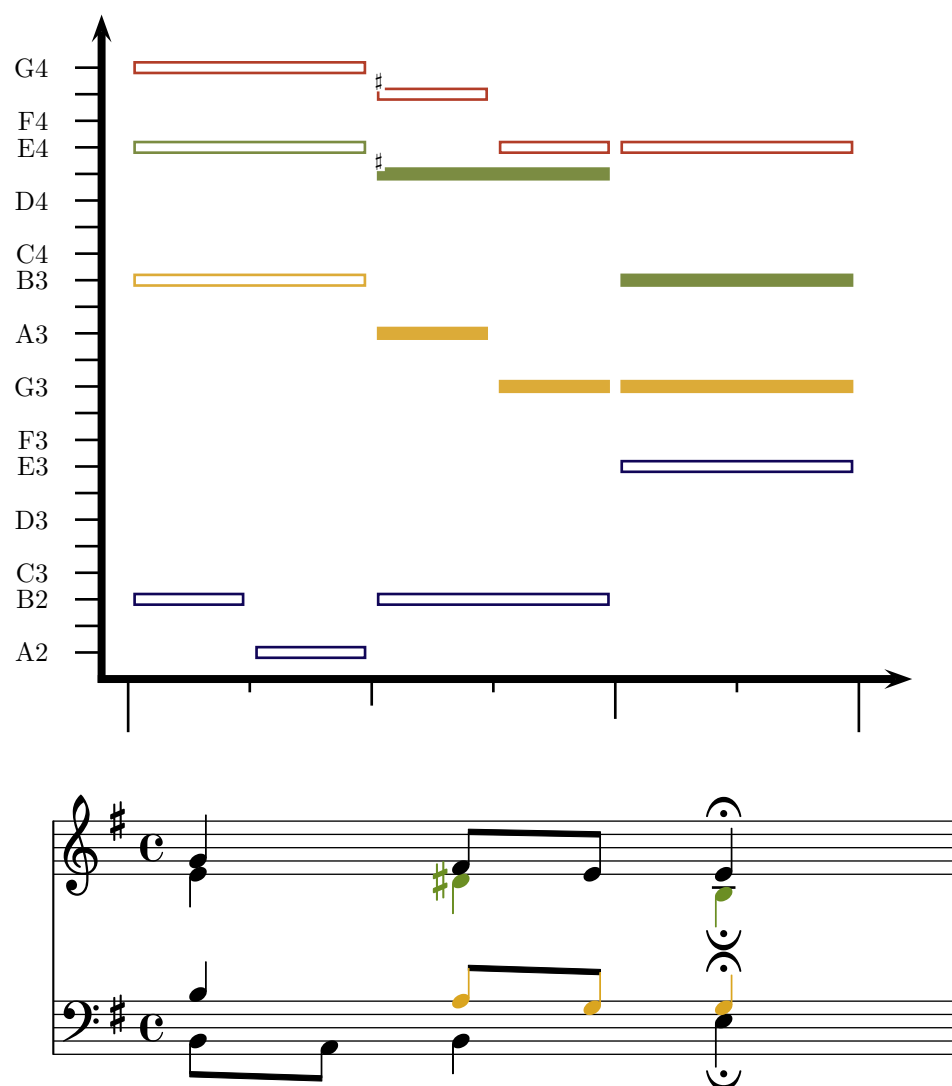


Figure F.35: BWV 315 (bar 13) with a highlighted tritone resolution pattern

F.5 Musical excerpts for Chapter 7

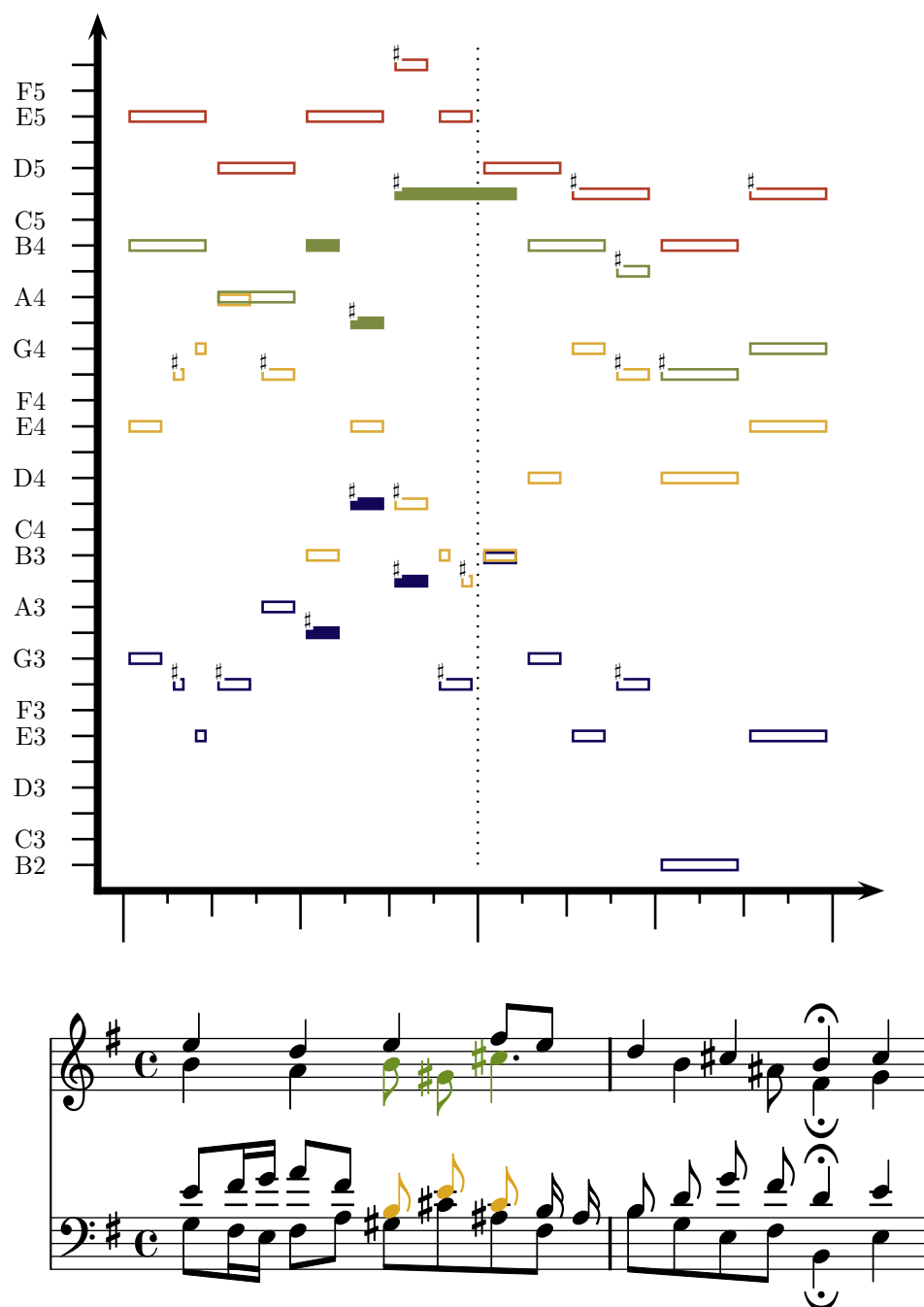


Figure F.36: BWV 278 (bar 8) with a highlighted rising fourths canon pattern

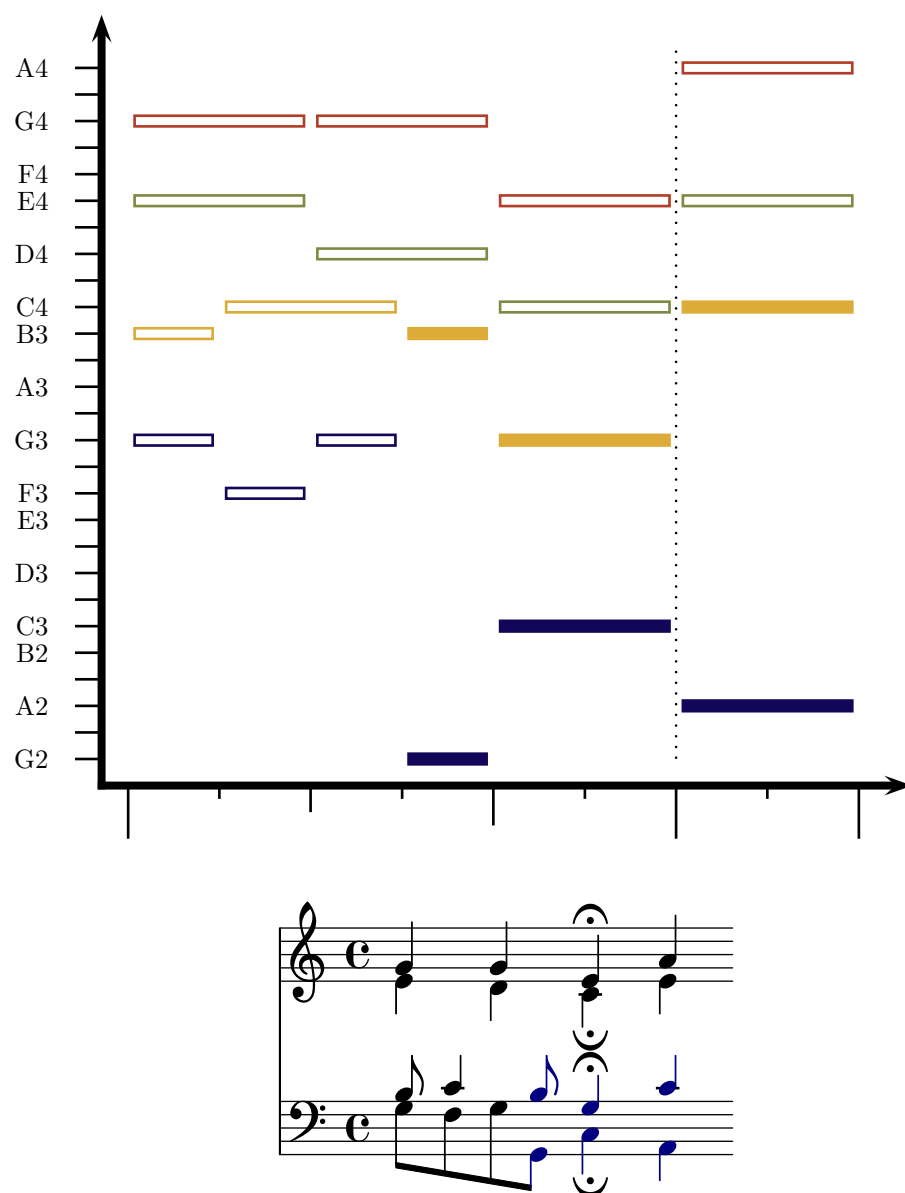


Figure F.37: BWV 328 (bar 43) with a highlighted rising fourths canon pattern

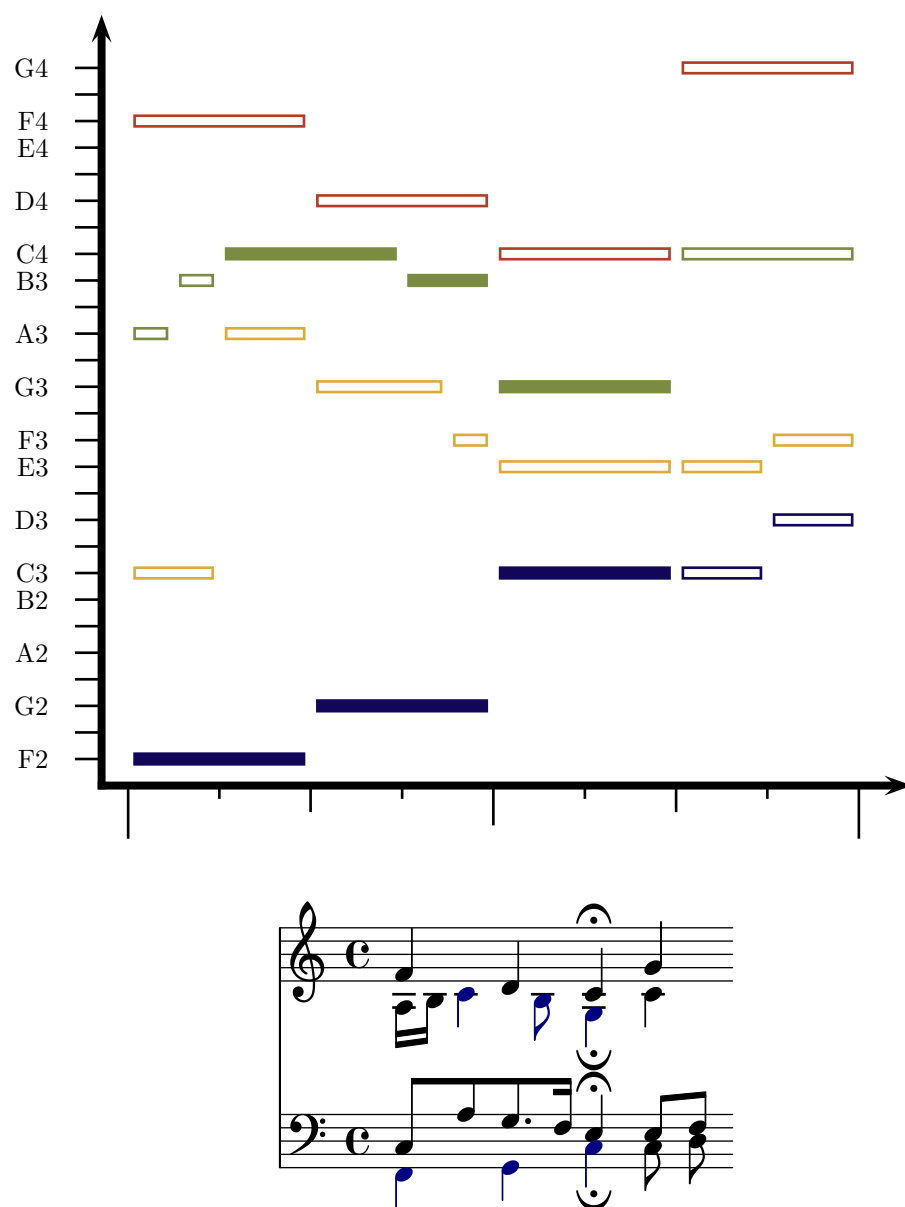


Figure F.38: BWV 328 (bar 33) with a highlighted two-voice module

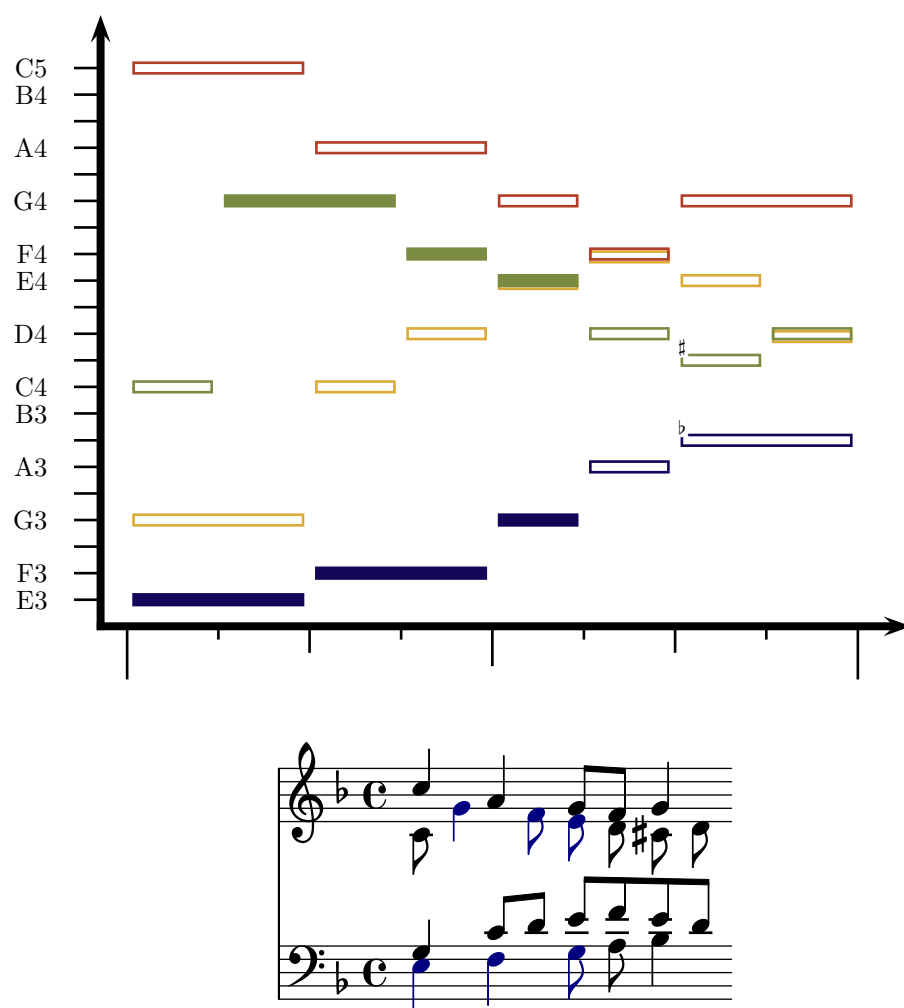


Figure F.39: BWV 382 (bar 9) with a highlighted two-voice module

Appendix G

Accompanying CD

Track	Piece	Excerpt	Pattern	Figure	Page
1	BWV 323	bars 1-5, tenor		1.1	3
				2.1	15
				2.2	16
				2.7 a)c)	27
2	BWV 323			1.2	9
3	BWV 323	bars 1-2		2.3	17
4	BWV 323	bars 3-5		2.5	19
5	BWV 323	bars 1-5	Transposed	2.7a)d)	27
6	BWV 323	bars 1-5	Transformed	2.7b)e)	27
7	BWV 323	bar 1		2.8a)c)	29
8	BWV 323	bar 1	Transposed	2.8a)d)	29
9	BWV 323	bar 1	Transformed	2.8b)e)	29
10	BWV 323	bar 8	Parallel fifth	2.9	31
11	BWV 323	bar 10	Suspension	2.10	32
12	BWV 304	bar 21		4.1	60
				5.2	80
				5.4	83

Table G.1: Accompanying CD, tracks 1-12

Track	Piece	Excerpt	Pattern	Figure	Page
13	BWV 273	bar 5	Compensated leap	5.6a)	91
				F.3	173
14	BWV 349	bar 12	Compensated leap	5.6b)	91
				F.4	174
15	BWV 297	bar 3	Compensated leap	5.6c)	91
				F.5	175
16	BWV 277	bar 1	Compensated leap	5.6d)	91
				F.6	176
17	BWV 292	bar 1	Block chord	5.7a)	93
				F.7	177
18	BWV 335	bar 4	Block chord	5.7b)	93
				F.8	178
19	BWV 415	bar 1	Block chord	5.7c)	93
				F.9	179
20	BWV 438	bar 8	Block chord	5.7d)	93
				F.10	180
21	BWV 263	bar 6	Parallel fifth	5.8a)	96
				F.11	181
22	BWV 301	bar 3	Parallel fifth	5.8b)	96
				F.12	182
23	BWV 323	bar 8	Parallel fifth	5.8c)	96
				F.13	183
24	BWV 355	bar 15	Parallel fifth	5.8d)	96
				F.14	184
25	BWV 361	bar 12	Parallel fifth	5.8e)	96
				F.15	185

Table G.2: Accompanying CD, tracks 13-25

Track	Piece	Excerpt	Pattern	Figure	Page
26	BWV 259	bar 8	Suspension	5.9a)	98
				F.16	186
27	BWV 390	bar 12	Suspension	5.9b)	98
				F.17	187
28	BWV 393	bar 7	Suspension	5.9c)	98
				F.18	188
29	BWV 285	bar 1	Suspension	5.9d)	98
				F.19	189
30	BWV 253	bar 4	Passing tone	5.10a)	100
				F.20	190
31	BWV 395	bar 9	Passing tone	5.10b)	100
				F.21	191
32	BWV 262	bar 2	Passing tone	5.10c)	100
				F.22	192
33	BWV 426	bar 1	Passing tone	5.10d)	100
				F.23	193
34	BWV 277	bar 3	Counterpoint	5.11a)	102
				F.24	194
35	BWV 285	bar 9	Counterpoint	5.11b)	102
				F.25	195
36	BWV 380	bar 3	Counterpoint	5.11c)	102
				F.26	196
37	BWV 418	bar 7	Counterpoint	5.11d)	102
				F.27	197

Table G.3: Accompanying CD, tracks 26-37

Track	Piece	Excerpt	Pattern	Figure	Page
38	BWV 374	bar 7	Three-voice counterpoint	5.12a)	102
				F.28	198
39	BWV 398	bar 11	Three-voice counterpoint	5.12b)	102
				F.29	199
40	BWV 285	bar 15	Dislocated chord	6.3a)	107
				F.30	200
41	BWV 318	bar 13	Dislocated chord	6.3b)	107
				F.31	201
42	BWV 255	bar 2	Layered passing tones	6.4a)	107
				F.32	202
43	BWV 320	bar 19	Layered passing tones	6.4b)	107
				F.33	203
44	BWV 257	bar 2	Tritone resolution	6.5a)	108
				F.34	204
45	BWV 315	bar 12	Tritone resolution	6.5b)	108
				F.35	205
46	Waltz Op.64 Nr.2	bars 9-10	Suspension	7.1	124
47	BWV 278	bar 8	Rising fourths cannon	7.12a)	141
				F.36	207
48	BWV 328	bar 43	Rising fourths cannon	7.12b)	141
				F.37	208
49	BWV 328	bar 33	Two-voice module	7.13a)	141
				F.38	209
50	BWV 382	bar 9	Two-voice module	7.13b)	141
				F.39	210

Table G.4: Accompanying CD, tracks 38-50

Bibliography

- C. Agon, G. Assayag, M. Laurson, and C. Rueda. Computer assisted composition at Ircam: Patchwork & Openmusic. *Computer Music Journal*, 23(3):59–72, 1999.
- J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–842, 1983.
- A. Anglade and S. Dixon. Characterisation of harmony with inductive logic programming. In *International Conference on Music Information Retrieval (ISMIR)*, pages 63–68, Philadelphia, USA, 2008.
- M. Balaban. The music structures approach in knowledge representation for music processing. *Computer Music Journal*, 20(2):96–111, 1996.
- B. Bel. Time and musical structures. *Interface, Journal of New Music Research*, 2-3 (19):107–135, 1990.
- M. Bergeron and D. Conklin. Structured Polyphonic Patterns. In *International Conference on Music Information Retrieval (ISMIR)*, pages 69–74, Philadelphia, USA, 2008.
- M. Bergeron and D. Conklin. Temporal patterns in polyphony. In E. Chew, A. Childs, and C.-H. Chuan, editors, *MCM 2009: International Conference on Mathematics and Computation in Music, Communications in Computer and Information Science*, volume 38, pages 32–42, New Haven, USA, 2009. Springer-Verlag.
- E. Cambouropoulos. Voice separation: Theoretical, perceptual and computational perspectives. In *Proceedings of the 9th International Conference on Music Perception and Cognition (ICMPC)*, 2006. Bologna, Italy.
- E. Cambouropoulos, T. Crawford, and C. Iliopoulos. Pattern processing in melodic sequences: challenges, caveats & prospects. In *AISB’99 Convention (Artificial Intelligence and Simulation of Behaviour)*, pages 9–21, 1999.

- E. Cambouropoulos, M. Crochemore, C. S. Iliopoulos, M. Mohamed, and M.-F. Sagot. A pattern extraction algorithm for abstract melodic representations that allow partial overlapping of intervallic categories. In *International Conference on Music Information Retrieval (ISMIR)*, pages 167–174, 2005. London, UK. 2005.
- M. Clausen, R. Engelbrecht, D. Meyer, and J. Schmitz. PROMS: A web-based tool for searching in polyphonic music. In *International Conference on Music Information Retrieval (ISMIR)*, 2000.
- D. Conklin. Mining for distinctive patterns in the first movement of brahms’s string quartet in c minor. In *MaMuX International Workshop on Computational Music Analysis*, IRCAM, Paris, France, 2008.
- D. Conklin. Discovery of distinctive patterns in music. *Journal of Intelligent Data Analysis, Special Issue on Machine Learning and Music*, 2010. to appear.
- D. Conklin. Representation and discovery of vertical patterns in music. In C. Anagnostopoulou, M. Ferrand, and A. Smaill, editors, *Music and Artificial Intelligence: Lecture Notes in Artificial Intelligence*, number 2445, pages 32–42. Springer Verlag, 2002.
- D. Conklin and C. Anagnostopoulou. Representation and discovery of multiple viewpoint patterns. In *Proceedings of the International Computer Music Conference*, pages 479–485. International Computer Music Association, 2001. Havana, Cuba.
- D. Conklin and M. Bergeron. Representation and discovery of feature set patterns in music. *Computer Music Journal*, 32(1):60–70, 2008.
- D. Conklin and M. Bergeron. Relational pattern discovery in two-voice counterpoint. In *International Conference on Music Information Retrieval (ISMIR)*, 2010. to appear.
- D. Conklin and I. H. Witten. Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24(1):51–73, 1995.
- N. Cook. *A Guide to Musical Analysis*. J.M. Dent & Sons Ltd, 1987.
- D. Cope. *Computers and Musical Style*. A-R Editions, Madison, Wisconsin, 1991.
- L. De Raedt. A perspective on inductive databases. *ACM SIGKDD Explorations Newsletter*, 4(2):69–77, December 2002.

- S. Doraisamy and S. M. Rüger. Robust polyphonic music retrieval with N-grams. *Journal Intelligent Information Systems*, 21(1):53–70, 2003.
- M. J. Dovey. A technique for regular expression style searching in polyphonic music. In *International Conference on Music Information Retrieval (ISMIR)*, pages 179–185, 2001. Bloomington, Indiana, USA. 2001.
- W. J. Dowling and D. L. Harwood. *Music Cognition*. Academic Press, 1986.
- J.S. Downie. Music information retrieval. In B. Cronin, editor, *Annual Review of Information Science and Technology*, pages 295–340. Information Today, 2003.
- S. Dubnov, G. Assayag, O. Lartillot, and G. Bejerano. Using machine-learning methods for musical style modeling. *IEEE Computer*, 36(10):73–80, 2003.
- G. Fitsioris and D. Conklin. Parallel successions of perfect fifths in the Bach chorales. In *Fourth Conference on Interdisciplinary Musicology*, 2008. Thessaloniki, Greece.
- J. J. Fux. *Gradus ad Parnassum*. W. W. Norton, 1965, orig. 1725. translated and edited by Alfred Mann.
- R. O. Gjerdingen. Partimento, que me veux-tu? *Journal of Music Theory*, 50(1): 85–135, 2007.
- M. Good. MusicXML: An internet-friendly format for sheet music. In *XML 2001 Conference Proceedings*, Orlando, FL, USA, 2001.
- P. Hanna and P. Ferraro. Polyphonic music retrieval by local edition of quotiented sequences. In *Fifth International Workshop on Content-Based Multimedia Indexing*, pages 61–68, Bordeaux, France, 2007.
- D. J. Hu and L. K. Saul. A probabilistic topic model for unsupervised learning of musical key-profiles. In *International Conference on Music Information Retrieval (ISMIR)*, pages 441–446, 2009.
- P. Hudak, T. Makucevich, S. Gadde, and B. Whong. Haskore music notation - an algebra of music. *Journal of Functional Programming*, 6(3):465–483, 1996.
- D. Huron. Music research using Humdrum: A user’s guide, 1999. Stanford, California: Center for Computer Assisted Research in the Humanities.
- D. Huron. Tone and voice: A derivation of the rules of voice-leading from perceptual principles. *Music Perception*, 19(1):1–64, 2001a.

- D. Huron. What is a musical feature? Forte's analysis of Brahms's Opus 51, No. 1, revisited. *Music Theory Online*, 7(4), 2001b.
- D. Huron. On the role of embellishment tones in the perceptual segregation of concurrent musical parts. *Empirical Musicology Review*, 2(4):123–139, 2007.
- S. Jan. Meme hunting with the humdrum toolkit: Principles, problems, and prospects. *Computer Music Journal*, 28(4):68–84, 2004.
- M. Khadkevich and M. Omologo. Use of hidden markov models and factored language models for automatic chord recognition. In *International Conference on Music Information Retrieval (ISMIR)*, pages 561–566, 2009.
- R. D. King, A. Srinivasan, and L. Dehaspe. Warmr: a data mining tool for chemical data. *Journal of Computer-Aided Molecular Design*, 15(2):173–181, 2001.
- S. Kramer. Relational learning vs. propositionalization: Investigations in inductive logic programming and propositional machine learning. *AI Communications*, 13(4):275–276, 2000.
- C. L. Krumhansl and E.J. Kessler. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review*, 89:334–368, 1982.
- O. Lartillot. A musical pattern discovery system founded on a modeling of listening strategies. *Computer Music Journal*, 28:53–67, September 2004.
- K. Lemström and J. Tarhio. Searching monophonic patterns within polyphonic sources. In *Proceedings of the RIAO Conference*, volume 2, pages 1261–1278, 2000. Paris, France. 2000.
- X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. *The Objective Caml system, Documentation and user's manual*, release 3.07 edition, 2003.
- A. Lubiw and L. Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *International Conference on Music Information Retrieval (ISMIR)*, pages 289–296, 2004. Barcelona, Spain. 2004.
- D. Müllensiefen M. T. Pearce and G. A. Wiggins. A comparison of statistical and rule-based models of melodic segmentation. In *International Conference on Music Information Retrieval (ISMIR)*, pages 89–94, 2008.

- S. T. Madsen and G. Widmer. Evolutionary search for musical parallelism. In *Applications of Evolutionary Computing, proceedings of the EvoWorkshops 2005*, Lecture Notes in Computer Science, pages 488–497. Springer Verlag, 2005. Lausanne, Switzerland, 2005.
- A. Marsden. *Representing Musical Time*. Swets & Zeitlinger, Lisse, The Netherlands, 2000.
- A. Marsden. Generative structural representation of tonal music. *Journal of New Music Research*, 34:409–428, 2005.
- D. Meredith. Point-set algorithms for pattern discovery and pattern matching in music. In Tim Crawford and Remco C. Veltkamp, editors, *Content-Based Retrieval*, number 06171 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
- B. Meudic and E. Staint-James. Automatic extraction of approximate repetitions in polyphonic MIDI files based on perceptive criteria. In Uffe Kock Wiil, editor, *Computer Music Modeling and Retrieval*, volume 2771 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2003.
- J.-J. Nattiez. *Music and Discourse: Toward a Semiology of Music*. Princeton University Press, 1990.
- S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997.
- F. Pachet. Computer analysis of jazz chord sequences: Is Solar a blues? In *Readings in Music and Artificial Intelligence*, pages 85–113. Harwood Academic Publishers, Amsterdam, Netherlands, 2000.
- B. Pardo and M. Sanghi. Polyphonic musical sequence alignment for database search. In *International Conference on Music Information Retrieval (ISMIR)*, pages 215–222, London, UK, 2005.
- J. Pickens and T. Crawford. Harmonic models for polyphonic music retrieval. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 430–437. ACM, 2002. McLean, Virginia, USA. 2002.
- W. Piston. *Harmony*. W. W. Norton & Company, 1941.
- W. Piston. *Counterpoint*. W. W. Norton & Company, 1949.

- U. Pompe, I. Kononenko, and T. Makše. An application of ILP in a musical database: Learning to compose the two-voice counterpoint. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programing*, pages 1–11, Heraklion, Crete, 1996.
- L. Price. *Dialogues of Alfred North Whitehead*. Greenwood Press Reprint, 1954.
- R. Ramirez. Inducing musical rules with ILP. In *International Conference on Logic Programming*, volume 2916 of *Lecture Notes in Computer Science*, pages 502–504. Springer, 2003.
- M. Rohrmeier and I. Cross. Statistical properties of harmony in bach’s chorales. In *Proceedings of the 10th International Conference on Music Perception and Cognition (ICMPC 2008)*, Sapporo, Japan, 2008.
- P.-Y. Rolland and J.-G. Ganascia. Pattern detection and discovery: The case of music data mining. In David J. Hand, Niall M. Adams, and Richard J. Bolton, editors, *Pattern Detection and Discovery*, volume 2447 of *Lecture Notes in Computer Science*, pages 190–198. Springer, 2002.
- C. A. Romming and E. Selfridge-Field. Algorithms for polyphonic music retrieval: the Hausdorff metric and geometric hashing. In *International Conference on Music Information Retrieval (ISMIR)*, pages 215–222, 2007. Vienna, Austria.
- C. Sapp. Online database of scores in the Humdrum file format. In *International Conference on Music Information Retrieval (ISMIR)*, pages 664–665, 2005. London, UK.
- H. Schaffrath. The essen folksong collection in kern format, 1995. Menlo Park, CA: Center for Computer Assisted Research in the Humanities.
- P. N. Schubert. Hidden forms in palestrina’s first book of four-voice motets. *Journal of the American Musicological Society*, 60(3):483–556, 2007.
- E. Selfridge-Field, editor. *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, Cambridge, MA, USA, 1997.
- W. M. Szeto and M. H. Wong. A graph-theoretical approach for pattern matching in post-tonal music analysis. *Journal of New Music Research*, 35(4):307–321, 2006.
- D. Temperley. *The Cognition of Basic Musical Structures*. MIT Press, 2001.

- G. Toussaint. Computational geometric aspects of rhythm, melody, and voice-leading. *Computational Geometry*, 43(1):2–22, January 2010.
- D. Tymoczko. Three conceptions of musical distance. In E. Chew, A. Childs, and C.-H. Chuan, editors, *MCM 2009: International Conference on Mathematics and Computation in Music, Communications in Computer and Information Science*, volume 38, pages 258–272, New Haven, USA, 2009. Springer-Verlag.
- R. Typke, F. Wiering, and R. C. Veltkamp. A search method for notated polyphonic music with pitch and tempo fluctuations. In *International Conference on Music Information Retrieval (ISMIR)*, pages 281–289, 2004. Barcelona, Spain.
- E. Ukkonen, K. Lemström, and V. Mäkinen. Geometric algorithms for transposition invariant content based music retrieval. In *International Conference on Music Information Retrieval (ISMIR)*, pages 193–199, 2003. Baltimore, Maryland, USA. 2003.
- P. E. Utgoff and P. B. Kirlin. Detecting motives and recurring patterns in polyphonic music. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 487–494, 2006. New Orleans, Louisiana, USA. 2006.
- R.W. Young. Terminology for logarithmic frequency units. *The Journal of the Acoustical Society of America*, 11(1):134–139, 1939.

Index

- abstraction, 1–3, 11, 14, 15, 22, 25, 30, 37, 43, 51–53, 58, 61, 62, 66, 71, 143, 144
- Bach chorales, 8, 10, 12, 37, 40, 41, 59, 75–81, 83–85, 88, 90–102, 107, 108, 141, 143
- bar, 15, 40, 41, 149
- beat, 16, 21, 22, 50, 125, 149
- Chopin, 12, 75, 77–79, 81, 84–86, 88, 90, 92, 94, 96–98, 124
- chord, 3, 6, 16, 20, 21, 23, 29, 78, 85, 86, 91–94, 106, 107, 133, 140, 150
 - major, 93, 94
 - minor, 93, 172
- component, v, 8, 43, 44, 48, 51–53, 58, 61–64, 66–69, 79, 81, 82, 84, 85, 88, 90, 91, 93, 103, 105, 107, 108, 110, 112, 115, 117, 125, 130, 158, 163, 164, 168
 - aligned, 64, 68–70, 108, 116, 117, 163, 164, 168
 - distinguished, 72
 - index, 63, 67–71, 105, 109–112, 115–118, 159, 161–164, 168
 - subsumption, 133
- consonance/dissonance, 10, 28, 30, 41, 42, 55, 76, 82, 97–99, 148, 151
- corpus, v, 8, 12, 49, 52, 54, 73–79, 81, 84, 85, 88, 90–92, 97, 127, 143
- data mining, v, 5, 7, 8, 11, 12, 34, 35, 39, 43, 49, 52, 54, 103, 104, 119, 131, 134, 135, 143, 144
- duration, 3, 16, 20–22, 26, 27, 40, 41, 52, 54–56, 63, 71, 75, 79, 80, 125, 149, 150, 152
- event, v, 5, 8, 22–25, 33, 34, 37, 38, 43, 46, 48, 51, 52, 54–56, 58, 59, 61–63, 66–72, 77–82, 88, 91, 120–122, 125–131, 144, 158
- feature, v, 8, 26, 58, 59
 - component feature, 61, 69, 72, 158
 - event feature, 59, 61, 69, 70, 72, 158
 - value, 26, 59, 61, 67, 158
- feature definition rule, v, 8, 45, 71, 89, 95, 97, 100, 101, 143
- feature name, 59, 61, 158
- feature set, 8, 26, 43, 123
- folk songs, 75, 77, 78, 81, 84, 85, 90
- frame query, 76, 78, 81, 82, 84, 85, 90, 91
- frame-conditioned probability, 76, 77, 79, 81, 84–86, 88, 90, 92, 94, 97, 98, 170–172
- instance, 1, 34, 43, 62, 66–69, 71–76, 78, 79, 106–108, 122–124, 161, 162
- interval, 9, 10, 26, 38, 54, 56, 59, 61, 71, 72, 75, 82, 84, 85, 89, 92–98, 147, 151, 153, 154, 156
 - compound, 28, 30, 82, 89, 148, 151

- non-compound, 28, 39, 89, 148, 151, 153, 154
- key, 75, 79, 82, 146, 151, 152
- layer, *see* structure
 - ends, 64, 69
 - starts, 63, 64, 70
- meter, 75, 79–81, 99, 148, 149, 152
- note, 3, 15, 149
- note symbol, 3, 8, 15, 150
- octave, 9, 28, 39, 53, 54, 82, 89, 94–96, 100, 125, 145, 149, 151, 153, 156
- offset, 23–25, 37, 38, 43, 59, 60, 71, 121, 149, 152
- onset, 23, 24, 37, 38, 43, 46, 48, 50, 55, 56, 59, 60, 63, 70, 71, 121, 149, 152
- operator
 - layering (“=”), 43, 62, 114–116, 123, 134, 159
 - onset modification (“–”), 37, 44, 62, 63, 68, 76, 88, 94, 115, 117, 121, 122, 126, 134, 159
 - sequencing (“;”), 43, 62, 64, 68, 69, 114–116, 118, 123, 134, 159
- parallel fifth, *see* pattern
- part, *see* voice
- pattern, 1, 62, 112, 159
 - block chord, 85, 86, 91–94, 133, 140, 172
 - compensated leap, 85, 88, 90, 91
 - counterpoint, 75, 85, 88, 99, 101, 102
 - dislocated chord, 106, 107
 - embellished tritone resolution, 108, 114
 - layered passing tones, 105–107, 113, 114
 - parallel fifth, 8–12, 14, 28, 30, 31, 34, 36–39, 43, 44, 46, 50, 51, 53, 85, 88, 94–97, 100, 104–106, 119, 120, 142, 144
 - pedal point, 125
 - suspension, 8–12, 14, 24, 28, 30, 32–34, 36, 37, 41, 42, 44, 46, 50–52, 76, 77, 85, 88, 97, 98, 106, 109, 110, 112, 113, 119, 120, 124, 127, 132, 139, 140, 142
- pattern discovery, *see* data mining
- pattern language, 11, 35, 39, 43, 48, 49, 51, 104–106, 108, 113
- pattern matching, 5, 6, 8, 11, 34, 36, 37, 40–43, 46, 49, 50, 52, 54–56, 58, 73, 161
- pattern subsumption, 134
- pitch, 3, 5, 9, 10, 15, 16, 20, 22, 25–28, 37, 39–41, 43, 46, 48, 52, 55, 56, 59, 61, 63, 71, 72, 75, 79, 80, 82, 89, 92, 94, 125, 145, 146, 149, 152
- polyphonic embellishments, 10, 99, 101, 172
- requirements (for a polyphonic pattern language), 12, 33, 34, 36, 57, 120
- root (of a chord), 93, 172
- salience, v, 6, 12, 75, 77, 81, 84–86, 88, 97, 143
- sequence, *see* structure
- source, 2, 11, 58, 78, 158
 - well-formed, 59
- strong note (counterpoint), 99, 156
- structure, 21, 22
 - free mixture, 21, 58, 64

- layer, v, 16, 20–22, 30, 36, 37, 40–44, 46, 48, 50, 52, 53, 56, 58, 62–64, 68, 73, 76, 85, 86, 88, 91, 92, 94, 97, 106, 108, 109, 112, 114–118, 130, 150
- layered sequences, 16, 21, 30, 64, 107, 108
- sequence, v, 7, 16, 21, 22, 58, 84, 88, 107, 108, 150
- sequence of layers, 16, 21, 30, 64, 104, 106, 108, 113, 115, 116, 118, 119
- subsumption, 7, 133
- suspension, *see* pattern
- Symphony no.40, 75, 77–79, 81, 84–86, 88, 90, 92, 94, 96–98
- temporal network, 105–107, 109, 112–115, 117–119
- temporal pattern, 104, 105, 143
- temporal relation
 - end together (**et**), 95, 99, 154
 - meet (**m**), 24, 33, 37, 62–64, 69, 84, 88, 105, 106, 108, 111, 117, 121, 142, 153, 159, 160
 - overlap (**ov**), 9, 30, 33, 38, 44, 62, 64, 78, 79, 88, 105, 106, 111, 112, 121, 142, 159
 - start together (**st**), 9, 24, 30, 33, 37, 38, 44, 62–64, 78, 79, 82, 84, 85, 91, 95, 99, 105, 106, 111, 112, 121, 142, 154, 159
 - start while (**sw**), 10, 24, 33, 36–38, 41, 44, 46, 62–64, 78, 79, 82, 88, 98, 99, 105, 106, 111, 112, 117, 121, 122, 142, 154, 159, 160
- texture
 - homophony, 14, 16, 18, 21, 24, 28, 29, 33, 52, 56, 120, 135, 150
 - monophony, v, 7, 8, 12, 14, 16, 20, 24, 26–28, 33, 45, 48, 50–52, 54, 56, 77, 150
 - polyphony, v, vi, 5–12, 14, 18, 24, 28, 34–37, 40, 45, 46, 48, 50, 51, 54–57, 75, 77, 79, 91, 94, 97, 103, 104, 120, 123, 125, 142, 150
 - time period, 22
 - time point, 22
 - token, 40, 109
 - transposition, 26–29, 56, 92, 150
 - variable
 - assignment, 67, 69, 72
 - value variable, 61, 67, 72, 90, 99, 122, 158
 - voice variable, 44, 61, 67, 72, 79, 99, 158
 - viewpoint, 7, 8, 26, 53, 123
 - voice, 6, 9, 16, 18, 30, 37, 39–41, 43, 44, 50, 53–56, 59, 61, 66, 67, 69, 77–79, 88, 91–94, 118, 121, 123, 124, 143, 144, 150
 - voice name, 43, 44, 59, 61, 67, 69, 72, 121, 158
 - weak note (counterpoint), 99, 156

