CHORD SEQUENCE PATTERNS IN OWL

MUSIC REPRESENTATION AND REASONING
WITH DESCRIPTION LOGICS FOR THE SEMANTIC WEB

JENS SEBASTIAN WISSMANN

A THESIS SUBMITTED FOR THE DEGREE
DOCTOR OF PHILOSOPHY

LONDON, JULY 2012

School of Informatics
CITY UNIVERSITY LONDON

Contents

List of Figures viii
List of Tables ix
List of Definitions xi
Acknowledgements xiii
Declaration xv
Abstract xvii

1 Introduction 1
1.1 Motivation 1
1.2 Research Questions and Objectives 2
1.3 Method 3
1.4 Scope 4
1.5 Outline 5

I Foundations and Literature Review 7

2 Knowledge Representation in the Semantic Web 11
2.1 Knowledge Representation and Reasoning 11
2.1.1 Scope of Knowledge Representation 11
2.1.2 Ontologies 12
2.2 Building blocks of the Semantic Web 12
2.3 Resource Description Framework RDF 13
2.3.1 Serialisation 14
2.3.2 RDF Semantics 15
2.4 Linked data 16
2.5 Summary 18

3 Description Logics 19
3.1 Description Logic Syntax 19
3.2 Standard Reasoning Services 23
3.3 Characteristics of Description Logics 24
3.4 Tableaux Calculi 26
3.5 Summary 26

4 OWL Web Ontology Language 29
4.1 Language Profiles 29
4.2 Modelling in the context of open world assumption . . . 31
4.3 Extensions to OWL 31
4.3.1 Rules 32

4.3.2 Graph-shaped descriptions 35
4.4 Non-standard Reasoning Services 36
4.4.1 Module Extraction 36
4.4.2 Justifications 37
4.4.3 Reductions 37
4.4.4 Learning DL concepts 38
4.5 Querying 38
4.5.1 Relational Queries 39
4.6 Summary 40

5 Representation of Sequences 41
5.1 Sequence Representation in RDF 41
5.1.1 *rdf:List* and *rdf:Seq* 41
5.1.2 N-ary Predicates 42
5.2 Suffix Trees 42
5.3 OWLList 43
5.4 Ordered List Ontology 46
5.5 Representation of Time 47
5.6 Summary 49

6 Music representation 51
6.1 OMRAS2 Music Ontology 52
6.2 Harmony OWL 52
6.3 Chord Symbols and Chord Sequences 54
6.4 Harmonic Analysis 54
6.5 Harmonic Structure 55
6.6 Higher Structural Levels 55
6.7 Grammars for Automatic Harmonic Analysis 56
6.8 Summary 57

II Modelling 59

7 Musical Entities and Simultaneities 63
7.1 Notes 63
7.1.1 Pitch and Pitch Class 64
7.1.2 Pitch Classes 65
7.2 Pitch Type Relationships 66
7.3 Intervals 67
7.3.1 Expressible pitch range 67
7.3.2 Minor, Diminished, and Augmented Intervals . . 68
7.4 Simultaneities 68
7.5 Chords 68
7.5.1 Chord Subsumption 69
7.6 Summary 69

8 Sequential Patterns 71
8.1 Modelling sequences in OWL 71
8.2 Approach to defining $\mathcal{SEQ}$ constructs 73
8.3 Sequence Pattern 73

8.4 Sequence Instances 74
8.5 Sequence construction and Pattern Labelling 76
8.6 Separation of sequential structure and containment . . . 77
8.7 Repetition 78
8.8 Number Restrictions 79
8.8.1 Number Restriction of Successive γ-Components 79
8.8.2 Number Restriction of Distributed γ-Components 80
8.8.3 Number Restrictions in α-pattern 80
8.9 Sequence Definition 80
8.10 Pattern Subsumption 81
8.11 Concatenation of Sequential Patterns 86
8.12 Summary 88

9 Examples of combining SEQ and MEO 91
9.1 Musical Forms 91
9.2 Conceptual Graphs 92
9.3 Summary 95

10 Grammars as SEQ Patterns 97
10.1 Formal Grammars 97
10.2 Grammars for Chord Sequences 100
10.3 Desirable transformation properties 101
10.4 Transformations of Productions 101
10.4.1 Variable-free productions 102
10.4.2 Non-recursive productions 102
10.5 Algorithm 105
10.6 Transformability 108
10.7 Optimisation of concept size 109
10.8 Summary 109

11 API for MEO/SEQ 111
11.1 Summary 113

III Evaluation and Applications 115

12 Profiling of SEQ Retrieval and Classification 119
12.1 Theoretical Considerations 119
12.1.1 Worst-Case Complexity 119
12.1.2 Effects of set-based DL Semantics 120
12.2 Performance tests 120
12.2.1 Performance of Instance Retrieval 121
12.3 Optimisations 122
12.3.1 Precomputing assertions and reasoning w.r.t. a simplified the TBox 122
12.4 Summary 123

13 Data Integration 125
13.1 Structural Integration 126
13.1.1 9GDB Corpus 126
13.2 Semantic Integration 126

13.2.1 Isophonics Beatles Corpus 127
13.2.2 A binding ontology for OMRAS 2 Chord Ontology 127
13.2.3 Representation of Chord Ontology Four-Chord Patterns 128
13.3 Summary 130

14 Subsumption of Discovered Feature Patterns 131
14.1 Representation of Feature Set Patterns 131
14.2 Translation of Feature Set Patterns 132
14.3 Maximally General Distinctive Chord Patterns 132
14.4 Subsumption of Distinctive Pattern 133
14.5 Summary 136

15 Harmonic Analysis with Grammars 137
15.1 Hybrid harmonic analysis 137
15.1.1 Implementation 138
15.2 Harmonic analysis using $\mathcal{SEQ}$ grammar patterns 140
15.2.1 Musical background 140
15.2.2 $\mathcal{SEQ}$ and grammar representation 141
15.2.3 Association of the bVII7 chord with the Bebop style 142
15.2.4 Role of the bVII7 chord: Formalisation 144
15.2.5 Role of the bVII7 chord: Results 145
15.3 Summary 148

16 Federated queries 151
16.1 Experimental setup 151
16.1.1 Harmonic Patterns 152
16.1.2 Query 153
16.1.3 Data quality issues 153
16.1.4 Reasoning Support 155
16.2 Results 155
16.3 Summary 156

IV Conclusion 159

17 Summary and Discussion 161

18 Future Work and Reflections 167

V Appendix 169

OWL Constructs and Semantics 171

Namespaces and Expressivity of Ontologies 175

References 177

Index 187

List of Figures

2.1 Overview of Semantic Web standards 13
2.2 An example of an RDF graph 14
2.3 Linked Open Data 17

3.1 Architecture of a simple DL system 21

4.1 Relationships between DLs and other KR formalisms . . 32
4.2 A conceptual model that requires graph relationships . . 36

5.1 Suffix tree representation of a string 43
5.2 Example of a pattern expressed as OWLList 43
5.3 Allen's Interval Relationships 49

6.1 Brief overview of Music Representations 51
6.2 Example Harmonic Analysis by a DCG 57

7.1 Accidentals 64
7.2 Subsumption lattice for pitches 66
7.3 Subsumptions between interval relations 67
7.4 Pitch range of a piano. 67
7.5 Chromatic and diatonic half step 68
7.6 Pattern for representing simultaneities 68
7.7 Chord Extensions 69

8.1 Structure of an example sequential pattern 72
8.2 Effect of different starting constructs on type-classifications 77

9.1 Chart of tonal progressions 91
9.2 Conceptual Graph for two voices singing in harmony . . 93
9.3 Rule for the harmonious resolution of a dissonance . . . 94

10.1 Relationship between grammars and $\mathcal{SEQ}$ patterns . . . 108

12.1 Instance retrieval times 121
12.2 Instance retrieval times 122

13.1 Overview of data sets and ontologies. 125
13.2 Mapping from OMRAS2 Music Ontology to $\mathcal{SEQ}$ 128

14.1 Sketch of the pattern space of MGDPs 133

14.2 Example patterns and instances found with viewpoint learning 134
14.3 Subsumption of discovered sequence patterns 135

15.1 Example Harmonic Analysis by a DCG 138
15.2 Example Annotation of the cadence Dm7, G7, Cj 139

16.1 Setup of federated query infrastructure 152
16.2 Average pattern count by year 156
16.3 Average pattern count by artist 156
16.4 Average pattern count by album 157

18.1 Model Exploration Cycle 167

List of Tables

3.1 Overview of common DLs and their constructs 23
3.2 Tableau expansion rules for $\mathcal{ALC}$ 26

5.1 Axiomatisation of the OWLList concept 44
5.2 Summary of OWLList constructs 45
5.3 Ordered List Ontology 46

6.1 Summary of the OMRAS2 Chord Ontology in DL notation 53
6.2 Example OMRAS2 Chord Description in DL notation . 53

8.1 Summary of $\mathcal{SEQ}$ constructs 89

10.1 Notational conventions for types of grammatical symbols 98

12.1 Complexity of satisfiability checking in OWL fragments 119

13.1 Number of chord sequences per genre and subgenre. . . . 126
13.2 Beginning of “How Insensitive” in the 9GDB format . . . 126
13.3 EBNF grammar for 9GDB chord sequences 127
13.4 Extensions in the 9GDB corpus 127

14.1 Example of a instance sequence for viewpoint learning . 131

15.1 Frequencies of A: bVII7 chords, B: [bVII7] $\triangleright$ [bIII] progressions, and C: IVm chords in 4 sub-corpora of the 9GDB. 143
15.2 Matching frequencies of grammars a) b) c) variants A and B in 4 sub-corpora of the 9GDB. 147
15.3 Matching frequencies of grammars a) b) c) restricted to IVm preceding bVII7 in 4 sub-corpora of the 9GDB. . . 148
15.4 Matching frequencies of grammars c) restricted to IIm preceding bVII7 in 4 sub-corpora of the 9GDB. 148

16.1 Root movement patterns 153
16.2 Federated query in SPARQL 154

18.1 DL axioms 171
18.2 Common Description Logics and their constructs 171
18.3 Common Description Logics (continued) 172
18.4 OWL variants 172

List of Definitions

1 RDF Abstract Syntax 14
2 Syntax of $\mathcal{ALC}$ concepts 19
3 Semantics of $\mathcal{ALC}$ 21
4 Concept Closure 31
5 Disjoint Union 31
6 Property Closure 31
7 Identity Role 34
8 Restricted Role Subsumption 34
9 Concept Product (Rudolph et al., 2008) 35
10 Module 36
11 Justification 37
12 Redundancy 37
13 Least Common Subsumer 38
14 Semantics of basic SPARQL 39
15 Content Restriction 73
16 Succession 74
17 Initial and Terminal Component 74
18 Successor Assertion 74
19 Content Assertion 75
20 Content Properties Assertion 75
21 Sequence Assertion 75
22 One or more C 78
23 None or more C 78
24 Successive γ Minimal Number Restriction 79
25 Successive γ Exact Number Restriction 79
26 Successive γ Maximal Number Restriction 79
27 Following γ Minimal Number Restriction 80
28 Following γ Exact Number Restriction 80
29 Following γ Maximal Number Restriction 80
30 Exact Distributed Cardinality in α 80
31 Maximal Distributed Cardinality in α 80
32 α-subsumption 84
33 γ-subsumption 84
34 Inverse construction 84
35 Disjunctive Normal Form 87
36 Sequence Concatenation 88
37 Concept Product Subsumption 95
38 Unrestricted Grammar (Type 0) 97
39 Derivation 98

40 Context-sensitive Grammar (Type 1) 99
41 Context-free Grammar (Type 2) 99
42 Regular Grammar (Type 3) 99
43 Linear Grammars 99
44 Safe Concatenability 103
45 OWL2 Direct Semantics 173

Acknowledgements

First and foremost, I like to thank my supervisor Dr. Tillman Weyde who from the beginning encouraged and guided my journey into Music Informatics. I am infinitely grateful for his sustained support, encouragement and considerable help throughout in the process of this research without which this thesis would not have been possible.

I'm deeply grateful to Prof. Darrell Conklin whose faculty to quickly understand my ideas and abstract the essential concepts has been impressive. His advise and insightful comments were invaluable in focusing this research and building the foundations which allowed me to progress this work.

I like to thank Prof. David De Roure and Dr. Andrew MacFarlane for their thorough review of this work and the many suggestions that helped to improve this thesis significantly.

I thank City University for supporting my work with a scholarship during my time in London.

I wish to thank all former colleagues from City University, and in particular from the Music Informatics research group, Mathieu Bergeron, Kerstin Neubarth, Aline Honingh and Ruben Hillewaere, for inspiring thought exchanges in work sessions, reading group meetings and retreats. Additional thanks to Mathieu for sharing and reflecting my visions and for being a faithful companion in the London adventure.

I wish to thank all colleagues and former colleagues at "Semantic Karlsruhe" for providing such an inspiring and enjoyable atmosphere. The unique environment that made this possible has largely been shaped and maintained by the efforts of Prof. Dr. Rudi Studer on the large scale and within FZI by Dr. Andreas Abecker, PD Dr. Catherina Burghart and Dr. Valentin Zacharias. I am sincerely grateful to them. I like to thank Heiko Haller for introducing me to this environment and for continuous inspiration in bridging the semantic gap. Particular, I'd like to thank the Theseus team, Veli Bicer, Jürgen Bock, Eugenie Giesbrecht, Stephan Grimm, Joachim Kleb, Michael Schneider and Tuvshintur Tserendorj for stimulating discussions and collaborations. Thanks also to Sebastian Rudolph for the many little tricks that helped me to put logic into practise.

I'm deeply touched by the continuous support of friends and family, without whom I could not have balanced the challenges of private and professional duties. I would especially like to thank Nan for her constant love, support, understanding for me and care for our children, even when I had to work overnight, on week-ends or on holidays. Finally, I'd like to thank Clara and Ida simply for being there and for reminding me that there is something else to live for…

Declaration

Abstract

This thesis addresses the representation of and reasoning on musical knowledge in the Semantic Web. The Semantic Web is an evolving extension of the World Wide Web that aims at describing information that is distributed on the web in a machine-processable form. Existing approaches to modelling musical knowledge in the context of the Semantic Web have focused on metadata. The description of musical content and reasoning as well as integration of content descriptions and metadata are yet open challenges. This thesis discusses the possibilities of representing musical knowledge in the Web Ontology Language (OWL) focusing on chord sequence representation and presents and evaluates a newly developed solution.

The solution consists of two main components. Ontological modelling patterns for musical entities such as notes and chords are introduced in the ($\mathcal{MEO}$) ontology. A sequence pattern language and ontology ($\mathcal{SEQ}$) has been developed that can express patterns in a form resembling regular expressions. As $\mathcal{MEO}$ and $\mathcal{SEQ}$ patterns both rewrite to OWL they can be combined freely. Reasoning tasks such as instance classification, retrieval and pattern subsumption are then executable by standard Semantic Web reasoners. The expressiveness of $\mathcal{SEQ}$ has been studied, in particular in relation to grammars.

The complexity of reasoning on $\mathcal{SEQ}$ patterns has been studied theoretically and empirically, and optimisation methods have been developed. There is still great potential for improvement if specific reasoning algorithms were developed to exploit the sequential structure, but the development of such algorithms is outside the scope of this thesis.

$\mathcal{MEO}$ and $\mathcal{SEQ}$ have also been evaluated in several musicological scenarios. It is shown how patterns that are characteristic of musical styles can be expressed and chord sequence data can be classified, demonstrating the use of the language in web retrieval and as integration layer for different chord patterns and corpora. Furthermore, possibilities of using $\mathcal{SEQ}$ patterns for harmonic analysis are explored using grammars for harmony; both a hybrid system and a translation of limited context-free grammars into $\mathcal{SEQ}$ patterns have been developed. Finally, a distributed scenario is evaluated where $\mathcal{SEQ}$ and $\mathcal{MEO}$ are used in connection with DBpedia, following the Linked Data approach. The results show that applications are already possible and will benefit in the future from improved quality and compatibility of data sources as the Semantic Web evolves.

1 *Introduction*

1.1 *Motivation*

This thesis addresses the representation of and reasoning on musical knowledge in the Semantic Web. The Semantic Web is an evolving extension of the World Wide Web (the *Web*) that aims at describing information on the web in machine-processable form (Berners-Lee et al., 2001). Formal ontologies play a key role in this effort. They are used to describe concepts of a domain and their relationships and allow inferences on them. In the multimedia domain, ontology-based semantic annotations enable many new applications such as content-based querying and automatic media management (Stamou and Kollias, 2005). However, the application of semantic technologies to the musical domain is still in its beginnings and there are many non-trivial challenges (Celma et al., 2006). Computational music representations can roughly be categorised into audio representations (e.g. waveforms, MP3), symbolic abstractions (e.g. score models, MIDI) and metadata (e.g. editorial information). Several efforts have addressed the representation of audio descriptors (Hunter, 2001; Tummarello et al., 2004) and of musical metadata (Raimond et al., 2007; Raimond, 2009). However, the work on symbolic music representation and musical structures is still in its early stages (Ibbotson, 2009).

This thesis explores an approach to representation of and reasoning on musical structure based on the Web Ontology Language OWL using Description Logic. An ontology for musical entities, $\mathcal{MEO}$, and an ontology for sequential structures, $\mathcal{SEQ}$, have been developed for representing *chord sequence patterns*. In this way, chord sequence data can be published on the Semantic Web. Further, Description Logic reasoners can use the logical relationships between musical entities that are encoded in the ontologies to infer additional knowledge of musicological interest such as harmonic information about chords sequences.

This representation enables many interesting applications using standard Semantic Web technologies with some extensions and specific tools. For example, chord sequence patterns that can be learnt from corpora using machine learning techniques (cf. Conklin, 2010), can be modelled and used in Web retrieval, e.g. for style-based queries. Grammars for harmony can be modelled, within limits, (cf. Weyde, 1994; Weyde and Wissmann, 2007) for powerful queries and even to perform automatic harmonic analysis. The more recent development of Linked Data (cf. Bizer et al., 2009) can also be applied to music, which has been explored by linking corpora to data from the Web.

Musicologists are often interested in questions such as ”In which pieces does this progression occur?”, ”Does this pattern occur in pieces of a specific style?”, ”Which progressions have the Beatles used in different creative periods?”, ”Is this pattern a subtype a more general pattern?”, ”Which chord variants does Charlie Parker use in his II V I progressions?”. Answering these questions requires a language to express patterns, means for pattern-based analysis and retrieval from a corpus or corpora, means to relate pattern and metadata as well as means to compare patterns. The techniques developed in this work provide these means for the case of chord sequences. The Semantic Web approach offers further possibilities. Musicologists are often faced with the laborious task to collect and prepare the data necessary for their experiments. Using the Semantic Web platform potentially eases this burden as researcher can access and contribute to a shared pool of chord sequences, patterns and metadata. The growth of music data in the web of linked data makes this scenario not unlikely. $\mathcal{MEO}/\mathcal{SEQ}$ can act as a shared ontology that can be combined together with already available music metadata ontologies in this scenario.

Additionally, the techniques for expressing and evaluating sequential patterns can be of use for knowledge engineers in other domains such as Bioinformatics.

1.2 Research Questions and Objectives

The Semantic Web comprises a set of currently quickly evolving standards and practices, among which the Description Logic based OWL is the main standard for knowledge representation and reasoning. Yet there is little work on whether and how this technology can be used to represent musical knowledge and to solve musical queries using Description Logic inferences. The research question underlying this thesis is:

> How can musical knowledge be represented in the Web Ontology Language and how can queries and reasoning be applied to musical and musicological tasks?

Answering this question requires a careful analysis of what musical knowledge is and what the requirements for a music information system are. We focus here on *chord sequences* as a starting point, as these are a common abstraction of musical structure. As OWL does not provide native constructs for representing temporal or sequential order, a necessary sub-goal is the representation of order of musical elements. Further, the representation should allow expressing different levels of musical abstractions such as harmony and notes.

The representation needs to support relevant reasoning tasks, such as classification and subsumption. In this context the possibilities and limits of expressing complex patterns are of importance, for example in comparison to grammars, which have been used often for harmony.

From the perspective of distributed music information systems it is also interesting to interlink structural information with metadata (e.g.

"Chord pattern C in piece P_1 was originally conceived by artist A in piece P_2."), and the system should run efficiently to support reasonable sizes of datasets and patterns.

Based on these considerations the following objectives will be investigated in this thesis:

1. To provide an overview of the state of the art in music representation on the Semantic Web.

2. To develop ontologies that allow expressing chord sequence patterns and that support musicological queries and inferences on the Semantic Web.

   (a) To define constructs for musical entities that allow defining chords and capturing their subsumption relationships.

   (b) To define constructs for sequential relationships that can capture the subsumption relationships.

   (c) To define constructs to express harmonic grammatical structures and techniques to enable automatic harmonic analysis

   (d) To develop techniques for linking structural and metadata descriptions.

   (e) To develop an API to support software development based on results 2a–2d.

3. To demonstrate how the developed ontologies allow integration of different conceptualizations of corpora and description.

4. To demonstrate how harmonic analysis can be conducted automatically with Semantic Web technology.

5. To demonstrate how the developed ontologies in combination with Semantic Web techniques enable new query possibilities for musicologists, especially with respect to federated queries over multiple data sources and different kinds of musical information.

1.3 Method

The main approach employed in this thesis is the logical modelling of musical structures, especially chord sequence patterns, in OWL-DL. The developed model is applied to a set of modelling and reasoning applications to provide a proof of concept and empirical results on their feasibility.

First, the sequential pattern language $\mathcal{SEQ}$ is developed using a formal modelling approach. The modelling is based on the Description Logic in OWL DL and follows the model of regular expressions. This is motivated by the finding that regular expressions are a popular means of computational music analysis, e.g. in the Humdrum toolkit (Huron, 1995). New modelling constructs are introduced one by one, analogous to those found in regular expressions. By staying within the means of OWL DL decidability of satisfiability checking is guaranteed.

It follows that reasoning support for the formalism is available. This includes for example instance classification that is useful for retrieval and subsumption checking which is useful for analytical purposes. Parts of the results have been published in Wissmann et al. (2010). A music object language ($\mathcal{MEO}$) is also provided.

Second, musical domain knowledge is modelled focusing on chords and chord sequences. By establishing translations from other representations it is established that $\mathcal{SEQ}$ can express characteristic musical patterns with the added benefit of reusing the patterns in Semantic Web queries. One task has been the translation of viewpoint representation for distinctive chord patterns (Conklin, 2010) and for Chord Ontology patterns (Mauch et al., 2007; Anglade and Dixon, 2008). As the translations lead to different conceptualisations of the respective approaches in $\mathcal{SEQ}$ ontologies, the question arises of whether these conceptualisations can be combined to become inter-operable. An initial *binding ontology* has been discussed that describes mappings between central concepts in the ontologies. It has been shown that certain mappings cannot be directly expressed in OWL.

Third, an investigation into using grammars with $\mathcal{SEQ}/\mathcal{MEO}$ has been conducted. Grammars are a powerful formalism that has frequently been used to model musical chord progressions. However, the limited expressiveness of OWL-DL does not allow a full modelling of context-free grammars. The limits of modelling grammars in $\mathcal{SEQ}$ have been investigated and a translation of a grammar into a $\mathcal{SEQ}$ pattern within these limits has been developed and tested on a grammar for Jazz chord progressions (Weyde, 1994). The alternative approaches of a hybrid system for using grammars has also been explored (Weyde and Wissmann, 2007).

Fourth, APIs and tools for working with $\mathcal{SEQ}/\mathcal{MEO}$ knowledge bases, queries and grammars have been implemented in Java. These tools and the ontologies have been used in several application scenarios.

1.4 Scope

Musically, the focus of the presented work is on representing chords and their sequential progression. The discussed sequential patterns can capture harmonic structures. Rhythm and melody are mostly omitted. It is shown how automatic classification of patterns that are characteristic for musical genres is possible with Semantic Web reasoning techniques and how these can be used in query scenarios.

Ontologically, the patterns are expressed using the Web Ontology Language OWL-DL, a decidable fragment of first order logic. It is not intended to move beyond the scope of this representation language in this work, i.e. by devising extensions of the underlying Description Logic.

Technically, this work builds upon the OWL API Java framework for processing of OWL.

1.5 Outline

In Part I the foundations of knowledge representation with Semantic Web technologies and Description Logics are introduced and topics in music representation are reviewed.

Part II of this thesis describes the main modelling work. First, in chapter 7, modelling constructs for musical entities such as notes and chords are described. As musical patterns have a sequential character, in chapter 8 representational means for sequential patterns in OWL are provided. Here the focus is on developing a formalism that resembles regular expressions in expressivity and syntax. Special attention is given to issues of pattern subsumption (section 8.10) and concatenation (section 8.11). Grammar formalisms are a well-known means of modelling harmonic structure. In chapter 10 we describe a transformation procedure from grammars into $\mathcal{SEQ}$ patterns. Expressivity and limitations of the translation are discussed. Part II is concluded by chapter 11, which introduces an API for $\mathcal{MEO}/\mathcal{SEQ}$.

In Part III the complexity of reasoning on $\mathcal{SEQ}$ is addressed theoretically and empirically with current OWL reasoners. The use of $\mathcal{MEO}/\mathcal{SEQ}$ is evaluated for representing patterns learnt from data and studying their subsumption relations. Another application is the use of grammars, where a hybrid and a pure Semantic Web approach are tested. Also, a Linked Data scenario has been realised linking the Beatles chord data to metadata from DBpedia.

Finally, Part IV discusses the overall results, directions for future work and reflections of the author.

Note. The author is referred to as 'we' in the thesis, but this does not imply collaborations. All work presented in this thesis has been done by myself, except where stated otherwise.

Part I

Foundations and Literature Review

In this part necessary technical foundations and relevant literature in the areas of Semantic Web and music knowledge representation are presented.

At the beginning, the Semantic Web approach to knowledge representation is introduced (cf. chapter 2) focusing on the representation of knowledge graphs in RDF and their application in creating linked data on the world wide web. Ontologies are seen as a key ingredient in this approach that allows logical description of knowledge and enables reasoning services. The foundations of ontological modelling in Description Logics (DLs) are given in the following (cf. chapter 3) and the web ontology language OWL that is based on DLs is presented (cf. chapter 4). These means of modelling and reasoning will be applied in the design of ontologies for musical structures in Part II.

As the Semantic Web approach is geared towards the description of static knowledge it does not directly provide means to represent temporal or sequential structures that naturally arise in music representation. In chapter 5 several existing approaches are surveyed and discussed. Notably, OWLList is presented in detail as it motivates the $\mathcal{SEQ}$ approach in chapter 8.

Finally, chapter 6 presents the musicological perspective. A brief general overview on types of musical knowledge is given. Afterwards, the state of the art of music information with Semantic Web methods is surveyed. The existing ontologies are mostly orthogonal to the work in this thesis as they either focus on non-structural aspects of music such as editorial metadata or do not focus on support for musicological reasoning tasks. Yet, the common expression of these ontologies and the later developed $\mathcal{MEO}$ and $\mathcal{SEQ}$ ontologies as RDF allows interoperability via schema integration as well as federated queries over different kinds of musical knowledge (cf. Part III). Additionally, approaches to automatic musical analysis tasks are discussed such as harmonic analysis using grammars. These serve as a yardstick for providing similar expressive means and reasoning capabilities in $\mathcal{MEO}$ and $\mathcal{SEQ}$ (cf. chapter 10 and chapter 15).

2 Knowledge Representation in the Semantic Web

The work presented in this thesis relates to a number of subject areas, which will be reviewed in this chapter. First an overview of knowledge representation techniques in the Semantic Web is given. Then approaches to representing temporal and sequential knowledge with these techniques are discussed, and existing approaches to music representation are described.

2.1 Knowledge Representation and Reasoning

Knowledge representation (KR) and reasoning is an area of artificial intelligence that has the fundamental goal of representing knowledge in a manner that facilitates automatic inferencing.

2.1.1 Scope of Knowledge Representation

Davis et al. (1993) outlines that computational knowledge representation can be best understood in terms of five important and distinctive roles that a representation plays, each of which places different and, at times, conflicting demands on the properties a representation should have.

Surrogate A knowledge representation acts a *surrogate*, i.e. a substitute for the thing itself that is to be represented, used to enable an entity to determine consequences by thinking rather than acting, i.e. by reasoning about the world rather than taking action in it.

Ontological Commitment Second, KR is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?

Theory of Intelligent Reasoning Third, KRs can be seen as (fragmentary) theories of intelligent reasoning, expressed in terms of three components: the representation's fundamental conception of intelligent reasoning, the set of inferences the representation sanctions, and the set of inferences it recommends.

Medium of Efficient Computation Fourth, KR is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organising information so as to facilitate making the recommended inferences.

Medium of Human Expression Fifth, KR is a medium of human expression, i.e., a language in which we say things about the world.

The requirements of these applications can diverge. Therefore, today there is no single unified knowledge representation language or mechanism but many languages with individual trade-offs. This is also the case in music knowledge representation. For example, Mathematical Music Theory (e.g. Mazzola, 2003) focuses on ontological commitment and partially aims at providing a medium for human expression. Cognitive models such as the cognitive model for rhythm and melody segmentation by Weyde (2005) act as surrogates that are used to formulate and test hypotheses about the real world phenomena, in this case music perception.

2.1.2 Ontologies

The word *ontology*[1] originally refers to a subfield of philosophy or metaphysics that studies the nature of being. More concretely, this often involves the question of what entities can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences. The outcome of such ontological research, i.e. a description of the world is also called an ontology.

[1] from Greek: *Οντολογια* "study of being"

In Computer Science and Information Science, *representations* of the world are needed in many applications, e.g. for description of the knowledge of an artificial agent. Here, the concept of an *ontology* has been adapted in a more pragmatic sense, defining ontologies as formal descriptions of some entities of a domain of interest that are useful for reasoning tasks such as inference of new knowledge or checking the concepts for consistency.

The modern usage was coined by **?** who defined the notion of an ontology as an "explicit specification of a conceptualisation". Borst (1997) gives an extended definition of an ontology as a "formal specification of a shared conceptualisation". This definition additionally required that the conceptualisation should express a *shared* view. Also, such conceptualisation should be expressed in a formal (i.e. machine readable) format. Studer et al. (1998) merged these two definitions stating that "an ontology is a *formal*, *explicit* specification of a *shared* conceptualisation".

2.2 Building blocks of the Semantic Web

The Semantic Web is an ongoing activity to provide a common framework to describe share and reuse data on the World Wide Web. It comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. The development is coordinated by the World Wide Web consortium (W3C).[2] The W3C has recommended several standards that serve as building blocks for the Semantic Web. They are often depicted as a stack of technologies (see Figure 2.1) that now is habitually referred to as "the layer cake".

[2] `http://www.w3.org/2001/sw/`

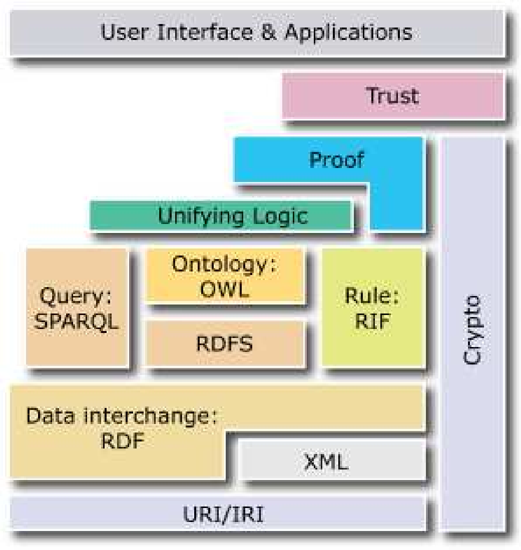


Figure 2.1: The latest "layercake" diagram of Semantic Web standards, retrieved from `http://www.w3.org/2001/sw/` 30.09.2008

The bottom layers can be seen as syntactical layers that standardise data structures and referencing using Universal Resource Identifiers (URIs) and the data interchange format XML (eXtensible Markup Language).

The next layer adds a way of making statements about resources (RDF) in a graph-like format. Such knowledge graphs can be given a formal semantics using the RDF-Schema language (RDFS) or the more expressive Web Ontology Language (OWL) allowing for more intelligent machine-processing.

Further layers that are still in the process of development add rule reasoning (SWRL, RIF). The work on Logic and Proof layers is still heavily under investigation. This is also the case for the Trust layer, which is supposed to involve human judgement to estimate the quality of knowledge distributed on the Semantic Web.

In the following, we will focus on an introduction to the standards RDF (cf. Section 2.3) and OWL (cf. Section 4). XML and XML-Schema play a minor role for the task of semantic interpretation that is discussed in this work. It will, however, be relevant at some occasions, as currently most of the relevant multimedia representation standards (e.g. MPEG-7 or MPEG-SMR) are described as structured documents using XML-Schema rather than described with semantic descriptions. A basic knowledge of XML and XML-Schema will be assumed. For further reference see the XML specifications[3] or one of the many good introductions available on the web.[4]

[3] `http://www.w3.org/XML/`

[4] e.g. `http://www.w3schools.com/xml/`

2.3 Resource Description Framework RDF

The Resource Description Framework RDF is the foundation for knowledge representation within the Semantic Web.[5] Following the tradition of Semantic Networks, RDF describes knowledge as graph structures.

[5] RDF is specified in a series of documents accessible via `http://www.w3.org/RDF/`.

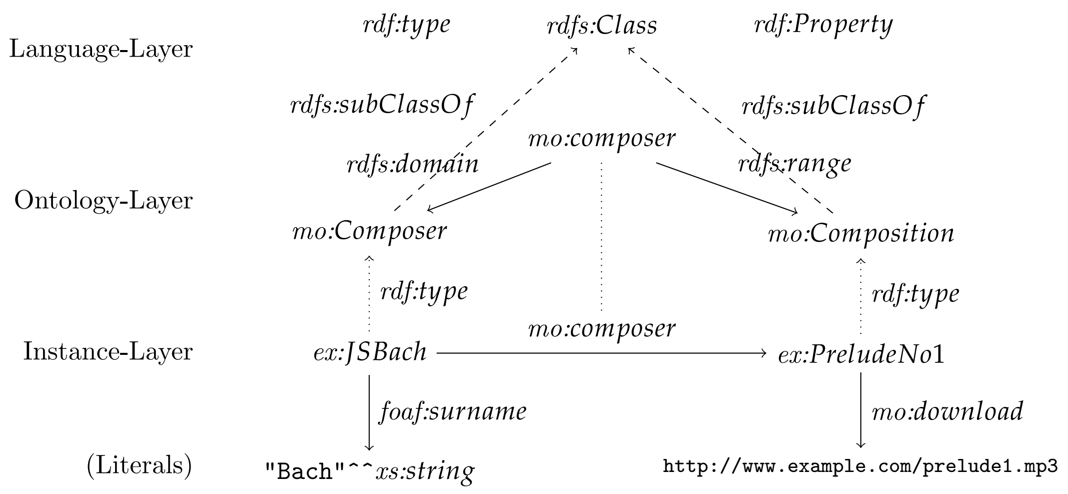


Figure 2.2: An example of an RDF graph

Definition 1 (RDF Abstract Syntax). *Knowledge is represented using* RDF Graphs*, sets of* statements *in the form of* triples *of information resources. An RDF triple is a tuple*

$$(s, p, o) \in (\mathbb{U} \cup \mathbb{B}) \times \mathbb{U} \times (\mathbb{U} \cup \mathbb{B} \cup \mathbb{L})$$

where $\mathbb{U}$ *is a set of explicit identifiers denoted using URIs (cf. Network Working Group, 2005),* $\mathbb{B}$ *is a set of anonymous identifiers called* blank nodes*, and* $\mathbb{L}$ *a set of* literals*. The components* s*,* p*, and* o *of an RDF triple are conventionally referred to as* subject*,* predicate *and* object*.*

An RDF-graph $\mathcal{G}$ *is a set of triples*

$$\mathcal{G} = \{t_1, \ldots, t_n \mid t_i \in \mathcal{T}\}$$

where $\mathcal{T}$ *is the set of all possible triples.*

Figure 2.2 shows an example of an RDF graph.

2.3.1 Serialisation

RDF can be serialised in different formats. Common formats are:

RDF/XML For web delivery an XML serialisation (*RDF/XML*) is commonly used, as this format also can be embedded in other XML based documents such as web pages (XHTML). A popular example is the description of newsfeeds using RSS, that combine XHTML-styled text with RDF metadata such as publication date or authorship.

Turtle For documentation purposes, a more easily readable plain text serialisation called *Turtle* is widely used.

Some technical concepts that appear in most syntaxes are

URI. Identifiers are expressed as URIs. Full URIs are often enclosed in angle brackets, i.e. ⟨`http://...`⟩.

QNames. URIs are where the namespace-prefix ns is defined in an @prefix annotation.

either given full within brackets (`http://...`) or as QNAME of the form *ns:name*,

Blank node identifiers use an underscore as prefix (*_:id*).

Literals are given in quotes. There exist a number of syntactic shortcuts, for example ";" to introduce another predicate and object for the same subject. URIs are normally understood as being composed of a namespace and name (`QNAME`), written *ns:name*.

Example 2.1 (Metadata in RDF Turtle) *The following RDF turtle fragment describes meta data on Bach's Kunst der Fuge and a performance of it:*

```
@prefix dc:   <http://purl.org/dc/elements/1.1/>.
@prefix mo:   <http://purl.org/ontology/mo/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix inst: <http://instrument.org/>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.

ex:kunstderfuge
        a        mo:Score;
        dc:title "Die Kunst der Fuge, composer's score";
        mo:published_as ex:publication1.
ex:publication1 dc:date "1751"^^xsd:year.
ex:emersonperf a mo:Performance;
        mo:performer  ex:emersonquartet;
        mo:instrument [ inst:violin, inst:violin,
                        inst:viola, inst:cello ];
        mo:performance_of bach:kunstderfuge.
```

(modified example from `http://www.musicontology.com/`*)*

2.3.2 RDF Semantics

RDF Schema (RDFS)[6] provides vocabulary to specify RDF vocabularies. This can be used for example to specify domain and range of relationships, or to specify subsumption relationships (see language-layer in Figure 2.2). The semantics of RDFS is defined through a set of axiomatic triples and entailment rules (Hayes, 2004). They determine the full set of valid inferences from an RDF graph. Even though the concrete logical apparatus is not trivial, the entailment rules can be intuitively explained as follows. Each rule has a set of premises that conjunctively define its body. The premises represent "extended" RDF statements, where variables can occupy any of the three possible positions in the triple (that of a subject, of a predicate, or of an object).

[6] W3C Recommendation: `http://www.w3.org/TR/rdf-schema/`

The head of the rule comprises of one or more consequences, each of which represents in its turn an RDF statement. The consequences may not contain free variables, i.e. such that are not used within the body of the rule. The consequences may contain blank nodes.

Example 2.2 (Axiomatisation of subclass in RDFS) *In the RDFS semantics the following rule ("rdfs9") propagates superclass membership to a resource:*

```
u rdfs:subClassOf x
v rdf:type u          => v rdf:type x
```

RDFS entailment is the only reasoning task supported by the RDFS semantics. It is the process of entailment of one (target) RDF graph from another (source) RDF graph by means of the set of rules that define the semantics of RDFS. In other words, it is the process of applying the entailment rules to RDF graphs in order to infer new triples, which can be regarded as a logical consequence of the initial (source) graph. The newly inferred triples can be denoted as *inferred closure* of the source graph. Determining whether a specific statement can be inferred from an RDF graph, therefore means to check if it is a member of its inferred closure.

2.4 Linked data

In recent years the amount of data available on the web in RDF format has been continuously increasing. For example, as for November 2008 the Semantic search engine Sindice[7] indexes over 10 billion triples. Though the quantity does not necessarily say something about the quality of the information, it is interesting as research is challenged to deal with such amounts of data[8], for example in the design of reasoning methods.

[7] see `http://sindice.com/`

[8] see `http://challenge.semanticweb.org/`

The Linking Open Data project (Bizer et al., 2009) hosted by the W3C Semantic Web Education and Outreach group aims at publishing a wide range of datasets on the Semantic Web, and creating links between them. Figure 2.3 depicts the currently available datasets. Notably, there is a large set of music related metadata (encircled area). An advantage of the Semantic Web linking approach is the possibility of expressing queries over data sets from different domains, such as song and artist information together with Wikipedia knowledge (DBpedia) or geographical information (GeoNames). We will later in Section 6.1 discuss the the OMRAS2 music ontology (Raimond et al., 2007) that plays a key role in this effort.

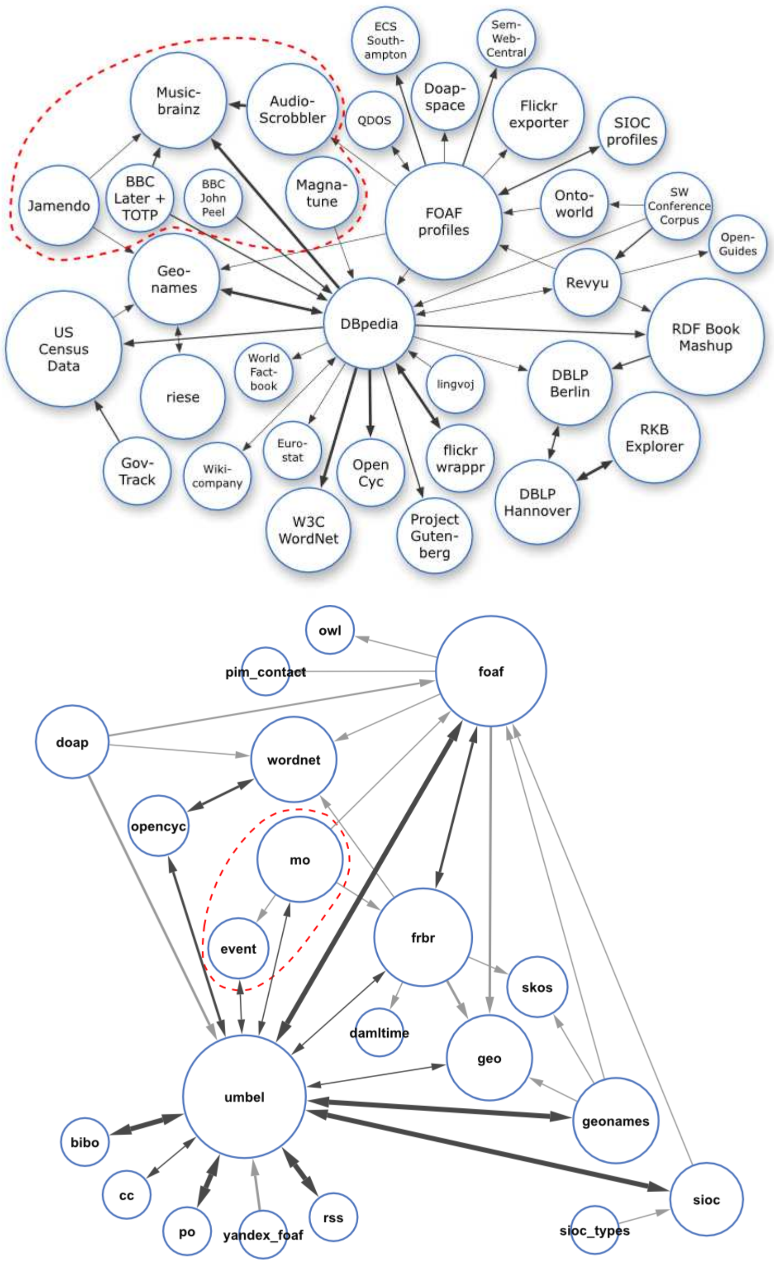


Figure 2.3: Data sets published in the Linking Open Data community project (top) and their underlying ontologies (bottom). Music related data sets and ontologies are encircled with a red-dashed line. Retrieved from `http://linkeddata.org/` and `http://www.umbel.org/lod_constellation.html` 2008-03-31

2.5 Summary

The W3C has developed standards for knowledge representation on the Semantic Web. The resource description framework (RDF) plays a central role as it allows expressing knowledge as graphs of relational data that can be published and interlinked on the web. The web ontology language OWL provides a way of adding logical descriptions to RDF data and enables reasoning on the data. The continuously growing web of linked data (LOD) in RDF already contains a large set of editorial data about music.

In this thesis additional primitives for musical objects and structures are developed that can be used to express chord sequences and chord sequence patterns in OWL, deploy them as RDF and perform musicological queries on them (cf. chapter 16). Notably, the RDF and LOD based approach has the benefit of enabling seamless combination of new and existing information in queries. RDF notation will mainly be used in the context of web deployment and web queries. OWL notation will be used for knowledge modelling as it more directly reflects the logical relationships and can be serialized to RDF.

The following chapters will focus on OWL and its theoretical foundations in Description Logics as these form the basis for the formal description of music knowledge developed in Part II.

3 Description Logics

Description logics (DLs) are a family of knowledge representation languages, which can be used to represent the terminological knowledge of an application domain in a structured and formally well understood way. DLs have been applied in many domains, such as medical informatics, software engineering, configuration of technical systems, natural language processing, databases, and web-based information systems.[1] The name Description Logic refers, on the one hand, to *concept descriptions* used to describe a domain, and on the other hand, to the *logic-based semantics* of DLs. They originated from research in semantic networks (Quillian, 1967) and frame systems (Minsky, 1981), and efforts to give these a formal basis (Woods, 1975; Brachman, 1977; Hayes, 1977, 1979).

[1] For details on these and other applications, see Baader et al. (2003, Part III).

3.1 Description Logic Syntax

Description Logics formalise the vocabulary of a domain in terms of *individuals*, *concepts*, and *roles*. They provide logical constructs that can be used to formally describe the relationship between the vocabulary. The "Handbook of Description Logics" (Baader et al., 2003) discusses several Description Logics that provide different sets of constructs. Many DLs such as the Web Ontology language OWL DL take the basic Attribute Language with Complements ($\mathcal{ALC}$) that was introduced by Schmidt-Schauß and Smolka (1991) as starting point and extend it with further constructs. We will now discuss $\mathcal{ALC}$ as a representative.

Concepts in $\mathcal{ALC}$ have the following form:

$$C, D \rightarrow \underbrace{A \mid \top \mid \bot}_{\substack{\textit{atomic}\\ \textit{concepts}}} \mid \underbrace{C \sqcap D \mid C \sqcup D \mid \neg C}_{\substack{\textit{boolean}\\ \textit{constructs}}} \mid \underbrace{\forall R.C \mid \exists R.C}_{\substack{\textit{role}\\ \textit{restrictions}}}$$

More formally, concepts are recursively defined as follows.

Definition 2 (Syntax of $\mathcal{ALC}$ concepts). *Let N_I, N_C and N_R be disjunct sets of* individual names, concept names *and* role names. *Then the set of $\mathcal{ALC}$-*concept descriptions *is defined inductively as follows:*

1. *Each atomic concept name $A \in N_C$ is an $\mathcal{ALC}$-concept description.*
2. *The* most general concept $\top$ *and the* unsatisfiable concept $\bot$ *are $\mathcal{ALC}$-concept descriptions.*

3. *If C, D are $\mathcal{ALC}$-concept descriptions, then the concept conjunction $C \sqcap D$, the concept disjunction $C \sqcup D$ and the concept negation $\neg C$ are also $\mathcal{ALC}$-concept descriptions.*

4. *If C is an $\mathcal{ALC}$-concept description and $R \in N_R$, then the existential role restriction $\exists R.C$ and universal role restriction $\forall R.C$ are also $\mathcal{ALC}$-concept descriptions.*

Concepts can be used to build definitions or state facts of the form

$$A \equiv C \qquad \text{(concept definition)}$$
$$C \sqsubseteq D \qquad \text{(concept inclusion)}$$
$$a \in C \qquad \text{(concept assertion)}$$
$$R(a, b) \qquad \text{(role assertion)}$$

In Appendix V a comprehensive list of further constructs that are available in the more expressive Description Logic OWL is given.

As a simple example, consider the following descriptions:

$$\texttt{Musician} \equiv \exists\texttt{performed.Music} \sqcup \exists\texttt{wrote.Music}$$
$$\texttt{Composer} \equiv \exists\texttt{wrote.Music}$$

These axioms define a musician as someone who performed or wrote music, and a composer as someone who wrote music. Boolean constructs and property restrictions are used to form expressions. As DLs have first order logic semantics we can think of these as complement, intersection and union of sets (of instances). Quantification on roles allows to define properties of the concept, e.g. that for an instance that is a `Composer` there exists a property `wrote` with the range `Music`.

OWL Reasoners provide certain standard reasoning services. For example, by *subsumption reasoning* a reasoner can infer that all composers are necessarily musicians ($\texttt{Composer} \sqsubseteq \texttt{Musician}$). In fact all subsumption problems in DLs are decidable, i.e. we can do this for any two concept descriptions. So the main challenge is to capture the interesting aspects of a terminology as DL axioms, whereas the reasoning is done automatically.

A further reasoner task is classification. Consider the facts

$$\texttt{wrote(mozart,magic\_flute)},$$
$$\texttt{wrote(shakespeare,hamlet)},$$
$$\texttt{magic\_flute} \in \texttt{Music},$$
$$\texttt{shakespeare} \in \forall\texttt{wrote.Literature}$$

Here, for example, Mozart will be classified as composer and musician. Shakespeare will not be classified as musician as he just wrote literature.

Knowledge Base. In DL systems information is stored in a knowledge base. It can be seen as a logical theory. The knowledge base is usually divided into into a *terminological* part (TBox) and *assertional* part (ABox) (see Figure 3.1). The TBox describes the terminology by relating concepts and roles. The TBox contains all the concept and role definitions, and also contains all the axioms of our logical theory (e.g. “A father is a Man with a Child”). The axioms of a TBox can be divided into definitions that can be used to state that a concept C is equivalent to another concept D ($C \equiv D$), and subsumptions that can be used to state that a concept C is a subclass of the concept D ($C \sqsubseteq D$). The ABox contains all the assertions (also called facts) of the logic theory. An assertion is used to express a property of an individual of the domain (for example $\mathit{Composer}(\mathit{Bach})$). An assertion can also assert a role, e.g. $\mathit{hasComposed}(\mathit{Bach}, \mathit{PreludeNo1})$.

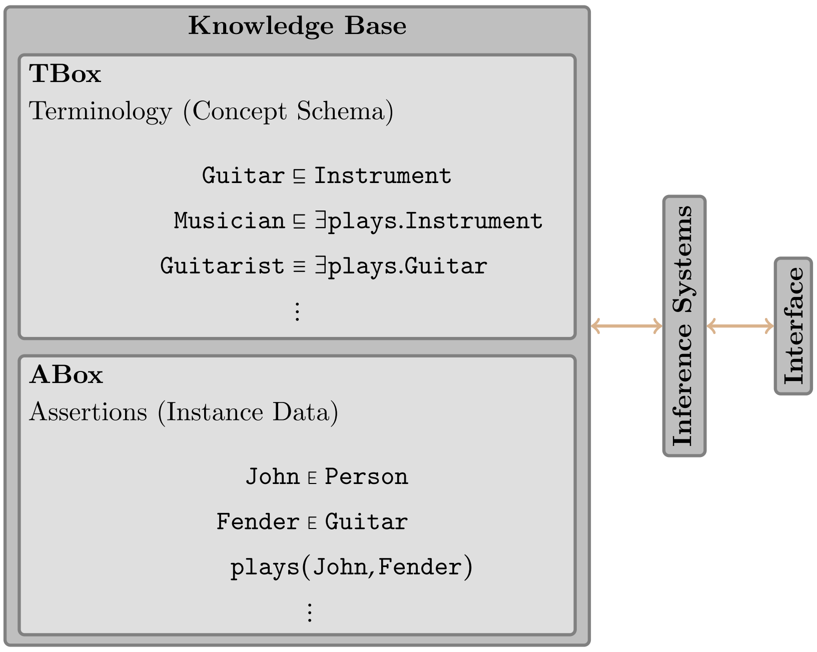


Figure 3.1: Architecture of a simple DL system

Definition 3 (Semantics of $\mathcal{ALC}$). *The semantics of $\mathcal{ALC}$ is given by an interpretations $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is a non-empty set, every concept* $\mathtt{A}$ *is mapped by the interpretation function $\cdot^{\mathcal{I}}$ to a set $\mathtt{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every role name R is mapped to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This function is extended to arbitrary concepts:*

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \qquad \text{(most general concept)}$$
$$\bot^{\mathcal{I}} = \emptyset \qquad \text{(unsatisfiable concept)}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad \text{(conjunction)}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \qquad \text{(disjunction)}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \qquad \text{(negation)}$$
$$(\exists R.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \qquad \text{(existential quantification)}$$
$$(\forall R.C)^{\mathcal{I}} = \{a \mid \forall b.(a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\} \qquad \text{(universal quantification)}$$

The semantics of the interpretation function is extended to axioms such that

$$\begin{aligned} \big(A \equiv C\big)^{\mathcal{I}} &\ \textit{if}\ A^{\mathcal{I}} = C^{\mathcal{I}} && \text{(concept definition)}\\ \big(C \sqsubseteq D\big)^{\mathcal{I}} &\ \textit{if}\ C^{\mathcal{I}} \subseteq D^{\mathcal{I}} && \text{(concept inclusion)}\\ \big(a \in C\big)^{\mathcal{I}} &\ \textit{if}\ a^{\mathcal{I}} \in C^{\mathcal{I}} && \text{(concept assertion)}\\ \big(R(a,b)\big)^{\mathcal{I}} &\ \textit{if}\ (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} && \text{(role assertion)} \end{aligned}$$

Let $\mathcal{KB} = (\mathcal{T}, \mathcal{A})$ be a knowledge base. $\mathcal{I}$ is a model of the knowledge base iff $\mathcal{I}$ is a model of the TBox and $\mathcal{I}$ is a model of the ABox, i.e.

$$\mathcal{I} \models \mathcal{K}\ \textit{iff}\ \mathcal{I} \models \mathcal{T}\ \textit{and}\ \mathcal{I} \models \mathcal{A}$$

where the interpretation of the TBox is defined as

$$\begin{aligned} &\mathcal{I} \models C \sqsubseteq D && \textit{iff}\ \ C^{\mathcal{I}} \subseteq D^{\mathcal{I}}\\ &\mathcal{I} \models \mathcal{T} && \textit{iff}\ \ \mathcal{I} \models \alpha\ \textit{for every axiom}\ \alpha \in \mathcal{T} \end{aligned}$$

and the interpretation of the ABox is defined as

$$\begin{aligned} &\mathcal{I} \models a \in C && \textit{iff}\ \ a^{\mathcal{I}} \in C^{\mathcal{I}}\\ &\mathcal{I} \models (a,b) \in R && \textit{iff}\ \ (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}\\ &\mathcal{I} \models \mathcal{A} && \textit{iff}\ \ \mathcal{I} \models \alpha\ \textit{for every axiom}\ \alpha \in \mathcal{A} \end{aligned}$$

DLs such as $\mathcal{ALC}$ can be seen as decidable subsets of first-order logic (FOL). Though FOL could (and is) also be used directly as means of knowledge representation, not all FOL reasoning task are decidable which is an obstacle for the implementation of automatic reasoners. The development of DLs has thus started from using decidable subsets of FOL and has been driven by the desire to push the expressivity bounds of these knowledge representation formalisms while still maintaining *decidability* and *implementability*. DLs have been continuously extended to include more expressive constructs while still guaranteeing the decidability of the language. DLs are named according to the constructs they provide to describe entities (see Table 3.1).

Entailment. Inference with OWL ontologies builds on the notion of logical consequence, and we write $\mathcal{O} \models \alpha$ to mean that an ontology $\mathcal{O}$ logically *entails* an axiom α, and $\mathcal{O} \models \{\alpha_1, \ldots, \alpha_n\}$ to express entailment of several axioms.

Signature. The *signature* of an ontology $\mathcal{O}$, denoted by $\Sigma(\mathcal{O})$, is the set of all individual, concept and role names occurring in $\mathcal{O}$, and thus, its vocabulary.

Deductive Closure. The *deductive closure* of an ontology $\mathcal{O}$, denoted $\langle\mathcal{O}\rangle$, is the set $\{\alpha \mid \mathcal{O} \models \alpha\}$ of all DL axioms α that are logical consequences of $\mathcal{O}$.

DLs	Constructors		Axioms [**Ax**]		
	Con	**Rol**	TBox	ABox	RBox
$\mathcal{EL}$	$\top$, A, $C_1 \sqcap C_2$, $\exists R.C$	R	$A \equiv C$, $C_1 \sqsubseteq C_2$	$a \in C$, $R(a,b)$	
$\mathcal{ALC}$	-||-, $\neg C$	-||-	-||-	-||-	
$\mathcal{S}$	-||-	-||-	-||-	-||-	`tra` R
+ $\mathcal{I}$		R^-			
+ $\mathcal{H}$					$R_1 \sqsubseteq R_2$
+ $\mathcal{F}$					`fun` R
+ $\mathcal{N}$	$\exists^{\geq n} S$				
+ $\mathcal{Q}$	$\exists^{\geq n} S.C$				
+ $\mathcal{O}$	$\{i\}$				
+ $\mathcal{R}$	$\exists R.$**Self**	$\neg R$ **U**			$R \circ S \dot{\sqsubseteq} R$ $S \circ R \dot{\sqsubseteq} R$ `ref` R `irr` R `asy` R **Disj**(R,S)

Table 3.1: Overview of common DLs and their constructs (adapted from Horrocks and Sattler (2005) and extended with constructs from Horrocks et al. (2006))

3.2 Standard Reasoning Services

A knowledge representation system based on DLs is able to perform specific kinds of reasoning. The purpose of a knowledge representation system goes beyond storing concept definitions and assertions. A DL knowledge base has semantics that makes it equivalent to a set of axioms in first-order predicate logic. Thus, like any other set of axioms, it contains *implicit knowledge* that can be made explicit through inferences. For this purpose, several existing DL reasoners can be used such as Pellet (Sirin et al., 2007), FaCT++ (Tsarkov and Horrocks, 2006), and RACER (Haarslev and Möller, 2003). They are typically based on tableau calculi (Baader et al., 2003, Chapter 2). DL reasoning services can be distinguished between services for the TBox and services for the ABox.

For a TBox T standard reasoning services are:

Satisfiability. This task verifies that a concept C is *satisfiable* with respect to T, i.e. there exists a model $\mathcal{I}$ of T such that $C^{\mathcal{I}}$ is nonempty. In this case we say also that $\mathcal{I}$ is a model of C.

Subsumption. This task verfies that a concept C is *subsumed* by a concept D with respect to T, i.e. $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of T. In this case we write $C \sqsubseteq_T D$ or $T \models C \sqsubseteq D$.

Equivalence. This task verifies that two concepts C and D are *equivalent* with respect to T, i.e. $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of T . In this case we write $C \equiv_T D$ or $T \models C \equiv D$.

Disjointness. This task verifies that two concepts C and D are *disjoint* with respect to T, i.e. $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \varnothing$ for every model $\mathcal{I}$ of T.

For an ABox A standard reasoning services are:

Consistency. This task verifies that an ABox is consistent with respect to a given TBox.

Instance Checking. This tasks verifies if a given individual x belongs to a particular concept C, i.e. $A \vDash C(x)$.

Instance Retrieval. This tasks returns the *extension* of a given concept C, that is, the set of individuals belonging to C.

3.3 Characteristics of Description Logics

In the design of DLs and OWL choices and trade-offs have been made that define the scope of the language. In the following we summarise a few fundamental notions.

Decidability. A logical language is called decidable if there is an effective procedure that given a theory can check the validity of an arbitrary formula. Famously, Gödel proved that there is no such procedure for FOL. FOL reasoners therefore cannot guarantee to find an answer to a given problem and heavily depend on heuristics. The expressivity of most Description Logics and especially OWL DL are restricted such that decidability is guaranteed and reasoning procedure can always find an answer.

Soundness and Completeness. *Consequence* is a central concept of logical languages. It describes that a logical statement follows from other sentences in the language. Two kinds of consequence relationships are distinguished. *Deductive consequence*, denoted $\Gamma \vdash S$, indicates that a sentence S can be deduced from a (possibly empty) class of sentences Γ of the given language. *Semantic consequence*, denoted $\Gamma \vDash S$, means that S is true in every model of Γ. Soundness means that all deducible sentences are also valid with respect to the semantics. Completeness means that consequences that follow from the semantic can be deduced.

$$\begin{tikzcd} \Gamma \vdash S \arrow[d, Rightarrow, shift left=2, "completeness"] \\ \Gamma \vDash S \arrow[u, Rightarrow, shift left=2, "soundness"] \end{tikzcd}$$

Monotonicity. Like FOL, Description Logics are monotonic. Monotonicity means that adding new information to a knowledge base never falsifies a previous conclusion.

Open World Assumption. Most Description Logics and in particular OWL use the *open world assumption* (OWA), i.e. the assumption that the truth-value of a statement is independent of whether or not it is known by any observer to be true. It is the opposite of the *closed world assumption* (CWA) which holds that any statement that is not known to be true is false. For example, consider the knowledge base containing the single statement "Bach is a composer.". If we ask whether "Beethoven is a composer", OWA based reasoning procedures will answer "unknown" as failure to derive a fact does not imply the opposite. On the hand CWA based reasoning procedures (as common in database languages such as SQL) will answer answer "no". Due to the non-monotonic nature of the closed-world assumption and the ambiguities about what closing should actually mean, in Description Logic inference systems usually there is no support for the closed-world assumption.

Unique Name Assumption. Standard DLs and OWL do not assume unique names, i.e. two individual names could signify the same individual if not stated otherwise.

Tree Model Property. Like modal logics, most DLs enjoy some form of tree model property (Grädel, 2001; Vardi, 1997), that is, every satisfiable concept or knowledge base has a model whose relational structure is tree shaped. This is not to say that it does not have other, non-tree shaped models, yet it tells us that we can concentrate on tree-shaped ones in algorithms to decide satisfiability. In contrast, function-free Horn rules are decidable because they lack existential quantification (and thus full negation) and therefore can be said to have a (very) small model property: a satisfiable set of such rules has a model whose elements all occur explicitly (i.e., as individual names) in the rule set. The relational structure of models, however, cannot be restricted.

Restriction vs. Constraint Semantics. The Description Logics used in the present research model knowledge in terms of restrictions. Restrictions are very similar to constraints as both terms refer to some aspects of a property, such as the cardinality or the range of the property. However, their logical behaviour is different. As constraint systems have found application in music informatics research (see Anders, 2007) it is useful to discuss the difference of the two methods.

From a logical point of view, a restriction applied to a class definition can be seen as a first-order formula. Thus, a restriction restricts the number of models of a logical theory. Restrictions are thus an integral part of the logical theory. For an ontology $\mathcal{O}$ and a restriction r, let M be the set of models of $\mathcal{O}$ and $M^{\mathcal{O} \cup r}$ be the set of models of $\mathcal{O} \cup r$, then $M^{\mathcal{O}} \supseteq M^{\mathcal{O} \cup r}$. Let $\langle \mathcal{O} \rangle$ be the set of consequences of $\mathcal{O}$ and $\langle \mathcal{O} \cup r \rangle$ be the set of consequences of $\mathcal{O} \cup r$, then $\langle \mathcal{O} \rangle \subseteq \langle \mathcal{O} \cup r \rangle$. Thus, restrictions allow to *infer* additional information, because they increase the number of consequences.

Constraints specify conditions which may not be violated by an interpretation of the logical theory. Constraints are not part of the logical theory and thus they do not increase the number of consequences of the theory. For ontology $\mathcal{O}$ and constraint c, if $\mathcal{O} \not\models c$ we say that the constraint is violated. However, since we don't see c as part of the theory, it does not affect the set of consequences. In summary, constraints do not allow to infer additional information, instead, they can be used to check the knowledge base with respect to certain conditions.

3.4 Tableaux Calculi

One prominent reasoning method to compute concept subsumption for DLs are semantic tableaux. Tableaux algorithms build models for concept descriptions. Subsumption can be tested by checking the (un)satisfiability of the negated concept descriptions. E.g. $C \sqsubseteq D$ is satisfiable iff $C \sqcap \neg D$ is unsatisfiable. To check this a tableaux algorithm tries to build a tree-like model for the input concept by applying the rules given in Table 3.2.

Note that the reasoning approach is based on set-theoretic operations. Therefore, though we can express and reason about sequential patterns, it cannot be expected that this order is taken into account when the tableaux is expanded.

Table 3.2: Tableau expansion rules for $\mathcal{ALC}$

$\sqcap$-rule:	if	1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and
		2. $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$
	then	set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if	1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \varnothing$
	then	set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$
$\exists$-rule:	if	1. $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and
		2. x has no R-successor y with $C \in \mathcal{L}(y)$,
	then	create a new node y
		with $\mathcal{L}(\langle x, y\rangle) = \{R\}$ and $\mathcal{L}(y) = \{C\}$
$\forall$-rule:	if	1. $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and
		2. there is an R-successor y of x with $C \notin \mathcal{L}(y)$,
	then	set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\sqsubseteq$-rule:	if	1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x is not blocked, and
		2. $C_2 \sqcup \dot{\neg} C_1 \notin \mathcal{L}(x)$
	then	set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \dot{\neg} C_1\}$

3.5 Summary

This chapter introduced Description Logics. First, in section 3.1, syntax and modelling means of DLs were described. Afterwards, in section 3.2, several inference services such as concept subsumption and instance classification that can be computed by DL reasoners were presented. Modelling and reasoning means are used in Part II to design modelling primitives for musical knowledge and musicological inferences.

Section 3.3 described characteristics of DLs that determine their modelling style, computational properties and (limits of) expressivity. A tableaux algorithm was presented in section 3.4 as the most common reasoning paradigm for DLs. The characteristics of DLs and DL reasoning mechanisms are introduced as reference for the later discussion and evaluation.

For simplicity this chapter focused on the basic Description Logic $\mathcal{ALC}$. The next chapter will present OWL (DL), an extension of $\mathcal{ALC}$ that is used in the Semantic Web to express knowledge in the form of web ontologies.

4 OWL Web Ontology Language

OWL is the second development of a standard for semantic metadata by the W3C. RDFS was the first W3C approach for modelling ontologies and has been widely adopted in the community of practice. However, concerns have been raised as to what extent it is a good basis for more sophisticated reasoning tasks. It has been shown that reasoning in RDFS is in some cases undecidable (ter Horst, 2005), and thus modifications, extensions and alternatives have been proposed as the basis for the further development of automatic reasoning within the Semantic Web.

Currently, attention has focused on using *Description Logics* (DLs) as a basis for ontological modelling. They have been used as formal basis for the Web Ontology Language (OWL), the currently recommended modelling standard within the Semantic Web.

This has led to very expressive DLs such as $\mathcal{SHOIN}$, the logic underlying the Web Ontology Language (OWL), $\mathcal{SHOIQ}$, and more recently $\mathcal{SROIQ}$ (Horrocks et al., 2006) which is the basis for the standardisation of OWL2. On the other hand, more light-weight DLs for which most common reasoning problems can be implemented in (sub)polynomial time have also been sought, leading, e.g., to the tractable DL $\mathcal{EL}^{++}$ (Baader et al., 2005).

4.1 Language Profiles

Often applications vary in their reasoning requirements. For example, in a bibliographic search scenarios speed of query answering might be more important than expressivity while in medical diagnosis one is often interested in accuracy and thus expressive modelling while reasoning performance is secondary. The OWL standard therefore differentiates between sub-languages called *profiles* that make different trade-offs regarding performance, expressivity and decidability.

The current OWL 1.0 language specification purposefully defines three different OWL sub-languages: OWL-Lite, OWL-DL, and OWL-Full. Each of the sub-languages has different usage limitations and has different expressive power.

OWL Lite is based on the model theoretic semantics of the DL $\mathcal{SHIF}(D)$ and a decidable subset of RDFS. It is a decidable and traceable subset of FOL.

OWL DL extends OWL Lite with the expressivity of the DL $\mathcal{SHOIN}_{(D)}$, thus becoming a larger but still decidable fragment of FOL, trading more expressivity for less tractability. The provided constructs are:

$$\{\underbrace{\mathtt{A}, \top, \mathtt{R}, \mathtt{R}^{+}, \mathtt{C} \sqcap \mathtt{D}, \mathtt{C} \sqcup \mathtt{D}, \neg \mathtt{C}, \exists \mathtt{R}.\mathtt{C}, \forall \mathtt{R}.\mathtt{C}}_{\mathcal{S}}, \underbrace{\mathtt{R} \sqsubseteq \mathtt{S}}_{\mathcal{H}}, \underbrace{\{i_1, \dots, i_n\}}_{\mathcal{O}}, \underbrace{\mathtt{R}^{-1}}_{\mathcal{I}}, \underbrace{\exists^{\geq n}\mathtt{R}, \exists^{\leq n}\mathtt{R}}_{\mathcal{N}}\}$$

OWL Full extends full RDFS and includes OWL DL. It is however not a DL any more but based on the axiomatic semantics of RDFS leading to an undecidable subset of FOL.

OWL 2 defines three different profiles:

OWL 2 EL enables polynomial time algorithms for all the standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed, and where expressive power can be traded for performance guarantees.

OWL 2 QL enables conjunctive queries to be answered in LogSpace (more precisely, AC0) using standard relational database technology; it is particularly suitable for applications where relatively lightweight ontologies are used to organise large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g., SQL).

OWL 2 RL enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organise large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.

OWL2 DL is based on the logic $\mathcal{SROIQ}_{(D^{+})}$ which is all of OWL DL plus qualified number restrictions, local reflexivity for simple properties; reflexive, irreflexive, and anti-symmetric flags for simple properties; disjointness of simple properties; and regular property inclusion axioms:

$$\underbrace{\mathtt{R} \sqsubseteq \mathtt{S}, \mathtt{R} \circ \mathtt{S}, \mathbf{U}, \exists \mathtt{R}.\mathbf{Self}}_{\mathcal{R}}, \underbrace{\exists^{\leq n}\mathtt{R}.\mathtt{C}, \exists^{\geq n}\mathtt{R}.\mathtt{C}}_{\mathcal{Q}}$$

Also, OWL is integrated with other web standards.

XML Datatypes. OWL allows ontologies to use concrete domains using simple built-in datatypes from XML Schema. XML Schema has a rich set of basic data types[1] including various numeric types, strings, and date-time types. It also has several mechanisms for creating new types out of the base types, e.g. defining an integer interval as a new data type that extends the built-in integer type.

[1] see `http://www.w3.org/TR/xmlschema-2/#built-in-datatypes`

The Semantics of the DL variants of OWL is defined using interpretation functions in a $\mathcal{ALC}$. Details can be found in Table V where we give a short summary of standard OWL 2 semantics based on Motik et al. (2009c).

4.2 Modelling in the context of open world assumption

In the context of the open world assumption (cf. paragraph 3.3) lack of knowledge of a fact does not immediately imply knowledge of the negation of a fact. One common pattern to restrict the possible models as needed are *closure axioms* (as e.g. discussed in Rector et al., 2004) that close off the possibility of further additions for a given property.

In the following we will introduce notation for common patterns to avoid unnecessary verbose axiomatisation in the later discussion.

Definition 4 (Concept Closure). *A* concept closure *expresses that a concept C is always one of the concepts D_i*

$$C \sqsubseteq \bigsqcup D_i$$

Concept union is based on classical disjunction and therefore not exclusive. For example, the genre definition `Genre` $\sqsubseteq$ `Pop` $\sqcup$ `Rock` $\sqcup$ `Classic` would also allow genres that are `Pop` and `Classic` at the same time. One common way to avoid this is the use of *disjoint union* in the concept definition to locally forbid concept intersection.

Definition 5 (Disjoint Union). *The (pairwise)* disjoint union *of a set of concepts $C_1, \dots, C_n$ is the concept expression*

$$\biguplus C_1, \dots, C_n := \bigsqcup_{1 \le i \le n} \left(C_i \sqcap \bigsqcap_{j \ne i} \neg C_j \right)$$

Using this construct we can redefine `Genre` concept as `Genre` $\sqsubseteq$ $\biguplus$`Pop,Rock,Classic`. This definition disallows any intersection of `Pop`, `Rock` and `Classic` within the context of `Genre`. A further possibility is to disallow such intersections globally by asserting *class disjointness axioms.*

Closure axioms can be used to express a finite choice.

Definition 6 (Property Closure).

$$C \sqsubseteq \forall R. \bigsqcup D_i$$

4.3 Extensions to OWL

The challenge in the design of a DL is to offer a combination of constructs that is expressive enough to capture interesting aspects of a given domain while still being decidable. In modelling musical knowledge sometimes the expressiveness of OWL is not enough to capture some musical notions. Particularly challenging is the limitation of most DLs that concepts can only describe tree-shaped models. DLs and OWL allow only very limited forms of graph relationships as the addition of constructs to describe graph relationships easily lead to undecidability. Yet, as musical objects are highly structured we will often want to express graph relationships. In the following therefore we will discuss the possibilities of OWL2 and then discuss extensions of OWL that have been found useful in the context of this work to provide additional constructs for graph relationships.

4.3.1 Rules

Rule-based formalisms as found in logic programming (e.g. Prolog) or deductive databases (e.g. Datalog) are popular paradigm of knowledge representation. In our context, rules are interesting because rule systems have also been used for music analysis and generation (e.g. Schaffer and McGee, 1998). Like for DLs, there is a large body of work on expressivity and complexity of rule languages (Dantsin et al., 2001). Many decidable and tractable formalisms are known.

The integration of DL formalism with rule formalisms is an ongoing project within the Semantic Web. Several proposals have been made, though not standardized. An idea underlying many of these approaches is that as OWL has a first-order semantics it is straightforward to combine OWL with first-order Horn-logic rules (Horrocks and Patel-Schneider, 2004).

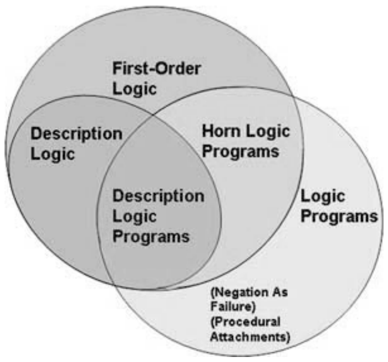


Figure 4.1: Relationships between DLs and other KR formalisms

The Semantic Web Rule Language SWRL[2] is a proposed rule extension to OWL that follows the union approach. SWRL rules are implications of conjunctions with unary predicates, binary predicates and constants corresponding to OWL concepts, role names and individuals. SWRL further has a library of built-in predicates and allows user-defined predicates, e.g. to allow arithmetic tests:

$$\begin{aligned} &\mathit{Piece}(?p) \land \mathit{composed}(?p, ?year) \land \mathit{swrlb{:}greaterThan}(?year, 1600) \\ &\qquad \land \mathit{swrlb{:}lesserThan}(?year, 1760) \\ &\qquad \Rightarrow \mathit{BaroquePiece}(?p) \end{aligned}$$

[2] http://www.w3.org/Submission/SWRL/

As the combination is straightforward, many of the current DL reasoners also support SWRL. However, reasoning becomes undecidable for the combination of OWL and SWRL. Therefore more restricted rule languages have been investigated. DL-safe rules (Motik et al., 2005) restrict the applicability of rules to a finite set of named individuals to retain decidability. Another approach is to identify the Horn-logic rules directly expressible in OWL. This has first been done by Grosof et al. (2003) for OWL-DL (i.e. $\mathcal{SHOIN}$) and this fragment has been called *Description Logic Programs* (DLP).

Recently, Krötzsch et al. (2008) and Gasse et al. (2008) characterised a large decidable fragment of SWRL called *DL rules* that can indirectly be expressed in OWL 2 (i.e. $\mathcal{SROIQ}$). The characterisation is based on the observation that the rules can be expressed in OWL 2 iff they have *tree-like* interdependencies of variables can be expressed. For example the conjunct

$$worksAt(x,y) \land University(y) \land supervises(x,z) \land PhDStudent(z)$$

that describes all people working at a university and supervising some PhD student can be expressed as the DL concept

$$\exists worksAt.University \sqcap \exists supervises.PhDStudent$$

Here variables form the nodes of a tree with root x, where edges are given by binary predicates. Intuitively, DL rules are exactly those SWRL rules, where premises (rule bodies) consist of one or more of such tree-shaped structures. One could, for example, formulate the following rule:

$$\begin{aligned} \forall x \forall y \forall z \quad & worksAt(x,y) \quad \land \quad University(y) \\ \land \quad & supervises(x,z) \quad \land \quad PhDStudent(z) \quad \rightarrow \quad profOf(x,z) \end{aligned}$$

For other rules the rewriting to DL axioms is less direct, but gives rise to interesting modelling patterns.

Example 4.1 *The fact that a woman that has a child is the mother of this child can be expressed by the rule:*

$$\forall x,y.Woman(x) \land hasChild(x,y) \rightarrow motherOf(x,y)$$

In order to express a binary property in the conclusion role inclusion axioms have to be used. The inclusion axiom $hasChild \sqsubseteq motherOf$ captures part of the rule but is still too general as it does not only hold for mothers but all individuals that have a child. OWL 2 does not provide direct means to restrict domain or range in role inclusion axioms. Krötzsch et al. (2008) noted that the role chains ($R_1 \circ R_2 \sqsubseteq S$) and local reflexivity restrictions ($\exists R.\mathbf{Self}$) as expressible in OWL 2 can be used to simulate such restrictions. For example, we can also produce above conclusion using the rules

$$\forall x.\texttt{Woman}(x) \leftrightarrow \texttt{women}(x,x)$$
$$\forall x,y.\texttt{woman}(x,x) \land \texttt{hasChild}(x,y) \rightarrow \texttt{motherOf}(x,y)$$

using an additional reflexive predicate woman*. Given this form a direct translation to OWL 2 axioms can be given:*

$$\begin{aligned} \texttt{Woman} &\equiv \exists\texttt{woman}.\textbf{Self} \\ \texttt{woman} \circ \texttt{hasChild} &\sqsubseteq \texttt{motherOf} \end{aligned}$$

Note that restrictions on the range of a property can be expressed in an analogous manner. Krötzsch et al. (2008) give other useful examples, but as their main aim is to investigate the expressivity of OWL2 they do turn it into a modelling pattern with additional new syntax.

Role restrictions are also known in other DLs. Baader et al. (2003, p.105) denote a domain restriction by $R|^C$ with its semantics defined as

$$\left(R|^C\right)^{\mathcal{I}} = \left\{(x,y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\right\}$$

Range restriction $R|_D$, and, domain and range restrictions $R|^C_D$ are defined accordingly. The combination of this general form with OWL2 is undecidable. For convenience, we will use this syntax for the modelling in this thesis and understand it under the semantics defined by the rewritings of Krötzsch et al. (2008).

Definition 7 (Identity Role). *Let C be a concept. The* identity role *of C is a role with role name $\texttt{id}_C$ interpreted w.r.t. $C \equiv \exists\texttt{id}_C.\textbf{Self}$.*

Definition 8 (Restricted Role Subsumption). *Let C be a concept, R be a role or role chain $R_1, \ldots, R_n$, and S a role, then*

$$\begin{aligned} R\big|^C \sqsubseteq S &= \quad \texttt{id}_C \circ R \sqsubseteq S \\ R\big|_C \sqsubseteq S &= \quad R \circ \texttt{id}_C \sqsubseteq S \\ R\big|^C_D \sqsubseteq S &= \texttt{id}_C \circ R \circ \texttt{id}_D \sqsubseteq S \end{aligned}$$

Further interesting constructs can be simulated using restricted role subsumptions together with the universal role $\mathbf{U}$ that holds between all individuals of the domain (cf. Rudolph et al., 2008). By restricting the universal role to $\mathbf{U}|^C_D$ we can describe a role that exists between all instances of C and all instances of D. For example the fact that all mice are bigger than elephants can be expressed as

$$\mathbf{U}\big|^{\texttt{Mouse}}_{\texttt{Elephant}} \sqsubseteq \texttt{biggerThan}$$

which according to Rudolph et al. (cf. 2008) captures the notation of a *concept product* and which they thus denote:

$$\texttt{Mouse} \times \texttt{Elephant} \sqsubseteq \texttt{biggerThan}$$

Note that the underlying DL structure is a role chain and that role chains are just allowed on the left-hand side of a role inclusion.

To simulate a concept product on the right side, we can The idea is to define domain and range for an inferred anonymous relation.

Definition 9 (Concept Product (Rudolph et al., 2008)).

$$C \times D \sqsubseteq R = \mathtt{id}_C \circ \mathbf{U} \circ \mathtt{id}_D \sqsubseteq R$$

How about concept products on the right-hand side of a role subsumption? In analogy to set operations, the expression $R \sqsubseteq C \times D$ could be used denote that R is a relation between instances of type C and D. We can directly express this in OWL by using a domain and a range restriction.

4.3.2 Graph-shaped descriptions

Motik et al. (2009b) recognized limitations of the modelling means of DLs in the medical domain:

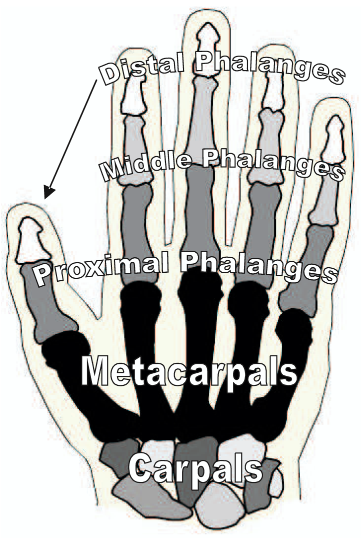
> "the expressivity [of DLs] is often insufficient to accurately describe structured object — objects whose parts are interconnected in arbitrary, rather than tree-like ways. DL knowledge bases describing *structured objects* are therefore usually underconstrained, which precludes the entailment of certain consequences and causes performance problems during reasoning."

To address this problem, they proposed an extension of DL languages with *description graphs* — a knowledge modelling construct that can accurately describe objects with parts connected in arbitrary ways.

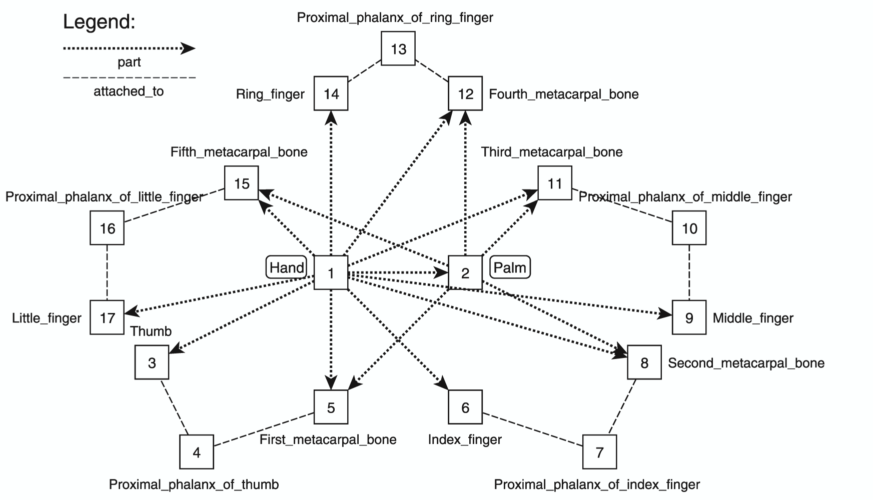
Consider the description of the anatomical structure of a hand depicted in Figure 4.2. This structure can be straightforwardly encoded in a DL ABox, but in this case would just describe one instance while we are interested in giving a general concept description that can be used to classify and match instances. A schema-level interpretation of the graph can be given by treating vertices as concepts and arrows as participation constraints between the concepts. For example, vertices 1 and 6 would correspond to concepts Hand and Index finger, whose instances would be all hands and all index fingers, respectively. Furthermore, the arrow from 1 to 6 would be interpreted as a statement that each hand has an index finger as its part:

$$\begin{aligned}
\mathtt{Hand} &\sqsubseteq \exists^{=1}\mathtt{part.Index\_finger} \\
\mathtt{Index\_finger} &\sqsubseteq \exists^{=1}\mathtt{part.Distal\_phalanx\_oif} \\
\mathtt{Index\_finger} &\sqsubseteq \exists^{=1}\mathtt{part.Middle\_phalanx\_oif} && \Leftarrow \\
\mathtt{Index\_finger} &\sqsubseteq \exists^{=1}\mathtt{part.Proximal\_phalanx\_oif} \\
\mathtt{Distal\_phalanx\_oif} &\sqsubseteq \exists^{=1}\mathtt{attached\_to.Middle\_phalanx\_oif} && \Leftarrow \\
\mathtt{Middle\_phalanx\_oif} &\sqsubseteq \exists^{=1}\mathtt{attached\_to.Distal\_phalanx\_oif} \\
\mathtt{Middle\_phalanx\_oif} &\sqsubseteq \exists^{=1}\mathtt{attached\_to.Proximal\_phalanx\_oif} \\
\mathtt{Proximal\_phalanx\_oif} &\sqsubseteq \exists^{=1}\mathtt{attached\_to.Middle\_phalanx\_oif}
\end{aligned}$$

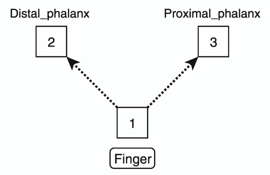
The two marked axioms imply the existence of middle phalanxes of the index finger, but this axiomatisation does not capture the fact



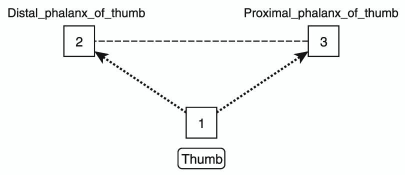
(a) Anatomy of the Hand



(b) Model of the Hand (G_{hand})



(c) Model of a Finger (G_{finger})



(d) Model of the Thumb (G_{thumb})

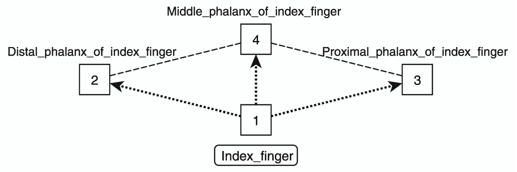

Middle_phalanx_of_index_finger
Distal_phalanx_of_index_finger
4
Proximal_phalanx_of_index_finger
2
3
1
Index_finger
(e) Model of the Index Finger (G_{index_finger})

Figure 4.2: Anatomy of a hand as an example for a conceptual model that requires graph relationships (from Motik et al., 2009b)

that, for any given index finger, these two middle phalanges must be the same object. In fact, OWL(2) offers no means to express this kind of relationships. Therefore Motik et al. (2009b) introduce *description graphs* as an extension to OWL in order to express additional graph-like relationships in DL concepts. The formalism introduces the modelling constructs *graph specialisation* and *graph alignment* to capture these relationships.

4.4 Non-standard Reasoning Services

Additional reasoning services have been developed that are commonly referred to as “non-standard” reasoning services.

4.4.1 Module Extraction

Techniques of module extraction are used to obtain a fragment of an ontology that is semantically relevant for entailments over a given signature. Despite containing only a subset of the original ontology’s axioms, a *module* preserves all entailments with regard to this signature. We adapt the definition of a module based on the notion of conservative extension from Cuenca Grau et al. (2008).

Definition 10 (Module). *Let $\mathcal{O}$ and $\mathcal{O}_m$ be ontologies with $\mathcal{O}_m \subseteq \mathcal{O}$ and Σ be a signature. Then, $\mathcal{O}_m$ is a* module for Σ in $\mathcal{O}$ *if for every* *ontology $\mathcal{O}'$ with $\Sigma(\mathcal{O}') \cap \Sigma(\mathcal{O}) \subseteq \Sigma$, we have that $\mathcal{O}' \cup \mathcal{O} \vDash \alpha$ if and only if $\mathcal{O}' \cup \mathcal{O}_m \vDash \alpha$ for any axiom α with $\Sigma(\alpha) \subseteq \Sigma$.*

4.4.2 Justifications

Justifications are a means to provide explanations for given entailments. For example, we might be interested why exactly a piece does (or does not) fit a blues pattern such as the twelve-bar blues. Assuming that the chord sequence of the piece is represented as a DL instance `s` and that we can capture the blues pattern by a DL concept `P`, fit is defined by an instance-of relationship $\mathtt{s} \in \mathtt{P}$. A justification is then the minimal set of all axioms in the ontology that make this relationship true (or false).

A (regular) justification is a minimal subset of an ontology that supports a given entailment, captured by the following definition adapted from Kalyanpur et al. (2008):

Definition 11 (Justification). *For an ontology $\mathcal{O}$ and an axiom α with $\mathcal{O} \vDash \alpha$, a set $\mathcal{O}^{\mathcal{J}} \subseteq \mathcal{O}$ of axioms is a* justification for α in $\mathcal{O}$ *if*

$$\mathcal{O}^{\mathcal{J}} \vDash \alpha \textit{ and} \qquad \text{(entailment)}$$

$$\nexists\, (\mathcal{O}^{\mathcal{J}'} \subseteq \mathcal{O}).(\mathcal{O}^{\mathcal{J}'} \subset \mathcal{O}^{\mathcal{J}}) \wedge (\mathcal{O}^{\mathcal{J}'} \vDash \alpha) \qquad \text{(minimality)}$$

So called *laconic justifications* extend the notion to more fine-grained explanations and were introduced in Horridge et al. (2008). They are justifications whose axioms are maximally weak and do not contain any superfluous parts irrelevant for the respective entailment to hold. The core idea for computing laconic justifications is to first perform a structural transformation which flattens complex axioms into several simpler ones.

4.4.3 Reductions

For an ontology it is often desirable to derive a minimal version that does not contain any redundancy in its axioms but has the same "meaning" semantically. An ontology contains redundancy in terms of expressed axioms if any of the axioms it contains is entailed by other axioms contained. The removal of such a redundant axiom would preserve the deductive closure of the ontology due to this entailment. This is captured by the following definition.

Definition 12 (Redundancy). *An ontology $\mathcal{O}$ is* redundant *if it contains an axiom α such that $\mathcal{O} \setminus \{\alpha\} \vDash \alpha$. An ontology $\hat{\mathcal{O}} \subseteq \mathcal{O}$ is a* reduction of $\mathcal{O}$ *if*

$$\nexists\, \alpha \in \hat{\mathcal{O}} : \hat{\mathcal{O}} \setminus \{\alpha\} \vDash \alpha \textit{ and} \qquad \text{(irredundancy)}$$

$$\langle \hat{\mathcal{O}} \rangle = \langle \mathcal{O} \rangle \qquad \text{(semantic equivalence)}$$

Grimm and Wissmann (2011) defined algorithms to efficiently compute all possible reductions of an ontology.

4.4.4 Learning DL concepts

Least Common Subsumer Though several tasks can be achieved with above reasoning services, all knowledge that is inferred and classified is already implicit in the logical model and does not add new knowledge in terms of classes or properties. Additional (sometimes called *non-standard*) reasoning services for DLs exist that can be used to learn new concepts form given data and concepts. Here we are interested in computing a concept L that generalises some other concepts $C_1, \dots, C_n$, i.e. that is a *common subsumer* of them. Not all common subsumers are interesting though. For example $\top$ is a trivial common subsumer as it subsumes any DL concept. Normally, there are more specific descriptions just generalize as much as to capture the commonalities of $C_1, \dots, C_n$. The notion of a least common subsumer (lcs) (Baader and Küsters, 2006; Baader et al., 2007) captures this idea. Intuitively, an lcs L generalises all concepts $C_1, \dots, C_n$ while there is no other generalisation L' that is more specific then L.

Definition 13 (Least Common Subsumer). *A concept description* L *is a least common subsumer (lcs) of concept descriptions* $C_1, \dots, C_n$ *towards a TBox* $\mathcal{T}$ *iff it satisfies the following two conditions:*

$$\begin{aligned} \forall i \in \{1, \dots, n\} : \quad & \mathcal{T} \models C_i \sqsubseteq L \quad \textit{and} \\ \forall L' : \ \textit{if} \ \forall i \in \{1, \dots, n\} : \quad & \mathcal{T} \models C_i \sqsubseteq L' \ \textit{and} \ L' \neq L \ \textit{then} \ \mathcal{T} \not\models L' \sqsubseteq L \end{aligned}$$

Least common subsumers have been used to support the bottom-up construction of DL knowledge bases. Other interesting subsumers such as partial and informative subsumers have been discussed in the literature (Colucci et al., 2008). Programmatic support for DL learning tasks has for example been implemented in the DL-Learner API[3].

[3] `http://aksw.org/Projects/DLLearner`

4.5 Querying

The query languages for Semantic Web ontologies can be classified under two categories:

1. RDF-based and
2. DL-based query languages.

RDF-based query languages, such as the W3C recommendation SPARQL[4], are based on the notion of RDF triple patterns and their semantics is based on matching triples with RDF graphs. RDF-based query languages offer flexibility similar to SQL but offers no guarantees to the consistency or well-formedness of results with respect to OWL.

[4] `http://www.w3.org/TR/rdf-sparql-query/`

DL-based query languages such as the queries of DIG protocol (Bechhofer et al., 2003), on the other hand, have well-defined semantics based on the DL model theory. While they make explicit guarantees to the consistency of the results they are more limited in the possible query constructs.

4.5.1 Relational Queries

The RDF query language SPARQL[5] allows to query RDF data by means of graph pattern matching. Graph patterns are expressed using turtle syntax and can be used in query statement similar to SQL such as the select-statement to retrieve information form RDF graphs:

[5] W3C recommendation: `http://www.w3.org/TR/rdf-sparql-query/`

Example 4.2 (SPARQL Select Query) *A list of Beatles albums is retrieved using a SELECT query :*

```
PREFIX mo:  <http://purl.org/ontology/mo/>
PREFIX dbp: <http://dbpedia.org/resource/>
SELECT  ?album
WHERE { ?album a mo:Record ;
               mo:releaseType mo:album ;
               dc:creator     dbp:the_beatles . }
ORDER BY ?album
```

Intuitively, SPARQL queries are RDF graphs where variables can occur in triples. A query is then comparing the triples with a data graph and binding the variables if the pattern matches. More formally, de Bruijn et al. (2005) defines the semantics of a query as follows :

Definition 14 (Semantics of basic SPARQL).
A SPARQL query basic graph pattern to an RDF graph $\mathcal{S}$ is a (possibly ground) RDF graph $\mathcal{Q}_{\mathsf{x}}$ where, in addition to URIs and bnodes, variables are allowed; the elements in the set x (possibly empty) of n variables of a query are called distinguished variables, and the bnodes play the role of non-distinguished variables. The answer set of $\mathcal{Q}_{\mathsf{x}}$ is the set of all substitutions of the distinguished variables with some arbitrary URI from $\mathcal{RDF}_{\mathrm{U}}$ (the set of all URIs and canonical literals) such that for each substitution the instantiated query is entailed by S, i.e.,

$$\{\langle c_1 \dots c_n \rangle \in (\mathcal{RDF}_U)^n \mid \mathcal{S} \vdash \mathcal{Q}_{[x_1 \mapsto c_1, \dots, x_n \mapsto c_n]}\}.$$

From an expressiveness point of view SPARQL is equivalent to Relational Algebra as shown by Angles and Gutierrez (2008). On the one hand this is of advantage for knowledge engineers as it provides them with powerful means to express interesting queries. On the other hand the language cannot guarantee decidability or soundness of the returned results. This can lead to problems for automatic reasoning and quality management.

4.6 Summary

The web ontology language OWL is a major standard for describing the formal meaning of vocabulary used in RDF. When using OWL to support reasoning on the web an increase of expressivity when describing knowledge models comes often at the cost of higher complexity. Language profiles of OWL help to make trade-offs by offering pre-defined subsets of OWL vocabulary with well-defined complexity (cf. section 4.1). In the context of this thesis we focus on the DL profile as it the most expressive OWL profile that still offers decidable reasoning.

OWL has certain characteristics as a modelling language that influence style and limitations of modelling in OWL. In section 4.2 some modelling constructs are discussed that become relevant in the context of the open world assumption. Section 4.3 discusses limitations of OWL in comparison to rule reasoning and graph shaped models and ways to deal with these within OWL and via extensions of OWL.

Standard reasoning tasks supported by OWL reasoners are consistency, satisfiability and subsumption checking. Next to these, further "non-standard" reasoning tasks have been developed that are potentially interesting for musicological queries and for organisation of a musical knowledge base. Section 4.4 discusses module extraction, justifications and reductions. In this thesis they are mainly used as means of optimization (e.g. section 12.3).

A major use of a knowledge basis is querying. Query paradigms for OWL can be distinguished between RDF-based and DL-based (cf. section 4.5). The RDF-based approach as found in SPARQL offers expressive relation queries on RDF graphs. This approach is primarily syntactic and formal properties such as consistency are not guaranteed though a reasoning engine can be used to provide additional inferences. The DL-based approach directly uses reasoning and makes formal guarantees whereas the query expressivity is restricted.

5 Representation of Sequences

"The world to be modelled is full of sequences" (Drummond et al., 2006b). Due to the temporal nature of music, ordered sequences are a natural and ubiquitous data type in the models of musical data such as note sequences and chord progressions.

In first order logic, sequences are often modelled as n-ary predicates. Currently, only the highly expressive DLs of the $\mathcal{DLR}$ family (Lutz et al., 1999) have native support for *n-tuples* and *recursive fixed-point structures*, from which one can build lists, trees, etc. In contrast most common DLs and OWL are subsets of FOL that are restricted to two or three variables.

Being restricted to these means, modelling of sequences as *linked structures* remains as an option. We are aware of three such approaches. The first is the vocabulary provided in RDF itself. Second, Hirsh and Kudenko (1997) described a DL pattern based on *suffix trees*. And third, Drummond et al. (2006b) used a *linked list* pattern in OWL. We will describe these in the following.

5.1 Sequence Representation in RDF

5.1.1 rdf:List and rdf:Seq

RDF provides two constructs for list, *rdf:List* and *rdf:Seq*. An *rdf:List* is essentially a *linked list*, constructible using *rdf:first* (referring to list head), *rdf:rest* (referring list tail) and *rdf:nil* (denoting list end). A list with three elements *ex:a*, *ex:b*, *ex:c* could for example be written as the statements

$$
\begin{array}{ccl}
\_{:}1 & \textit{rdf:first} & \textit{ex:a}\ . \\
\_{:}1 & \textit{rdf:rest} & \_{:}2\ . \\
\_{:}2 & \textit{rdf:first} & \textit{ex:b}. \\
\_{:}2 & \textit{rdf:rest} & \_{:}3\ . \\
\_{:}3 & \textit{rdf:first} & \textit{ex:c}. \\
\_{:}3 & \textit{rdf:rest} & \textit{rdf:nil}.
\end{array}
$$

The notation "_:... " indicates anonymous identifers and their name have no additional meaning.In Turtle syntax (as used in SPARQL) this statement could therefore be abbreviated

$$[\textit{ex:a}\ \textit{ex:b}\ \textit{ex:c}]$$

The axiomatisation of RDF provides an infinite set of properties

$$\textit{rdf:\_i} \text{ with } i \in \mathbb{N}$$

that is used *rdf:Seq* to express order. Using this pattern, the "abc" could also be expressed

$$\begin{array}{lcl} \textit{\_:aSeq} & \textit{rdf:type} & \textit{rdf:Seq}\ . \\ \textit{\_:aSeq} & \textit{rdf:\_1} & \textit{ex:a}\ . \\ \textit{\_:aSeq} & \textit{rdf:\_2} & \textit{ex:b}\ . \\ \textit{\_:aSeq} & \textit{rdf:\_3} & \textit{ex:c}\ . \end{array}$$

However, the usage scope of these RDF patterns is very limited. For technical reasons *rdf:List* cannot be used in OWL. It is used for the RDF-serialization of OWL constructs and is therefore disallowed as role name[1]. The *rdf:Seq* construct is not disallowed but its logical semantics cannot be accessed within OWL. Further, this construct does not offer means to define the semantics of the structures, e.g. defining a "list of notes".

[1] see `http://www.w3.org/TR/owl-semantics/mapping.html#rdf_List_mapping`

5.1.2 N-ary Predicates

RDF data and ontologies are represented as triples. While this uniform representation exhibits some advantages with respect to applying means of graph processing, modelling and visualisation, many data structures can not be directly but have to be broken down into triples.

While unary and binary relationships can be directly expressed using triples, modelling of containment structures or n-ary relationships[2] is more indirect, as is sequence representation.

[2] see `http://www.w3.org/TR/swbp-n-aryRelations/`

5.2 Suffix Trees

Hirsh and Kudenko (1997) describe how strings and string pattern can be rewritten as *suffix trees* and represented in a DL. Figure 5.1 shows an example pattern that matches the string "RFFB". Every suffix of the string corresponds to a path from the root of the tree to some leaf, and each path from the root to some leaf corresponds to a suffix of the string. Substrings can be matched by using subtrees.

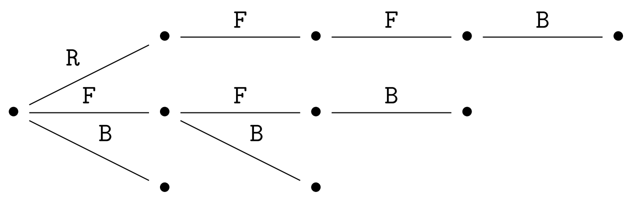


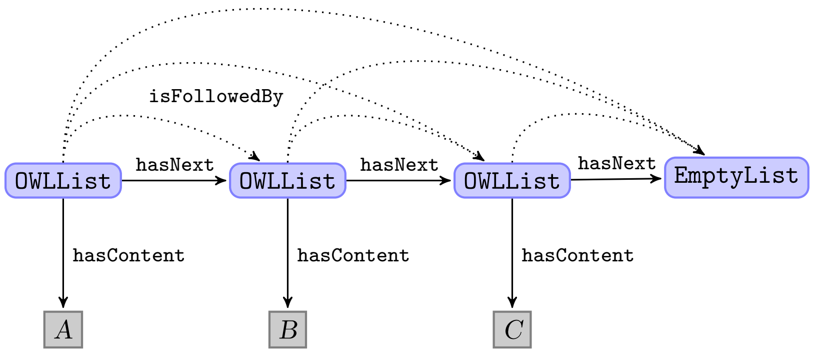
Figure 5.1: Suffix tree representation of the string “RFFB”

$$
\begin{aligned}
&\mathtt{NODE} \sqcap \forall \mathtt{R}.(\mathtt{NODE} \sqcap \forall \mathtt{F}.(\mathtt{NODE} \sqcap \forall \mathtt{F}.(\mathtt{NODE} \sqcap \forall \mathtt{B}.\mathtt{NODE})))\\
&\quad \sqcap \forall \mathtt{F}.(\mathtt{NODE} \sqcap \forall \mathtt{F}.(\mathtt{NODE} \sqcap \forall \mathtt{B}.\mathtt{NODE})\\
&\qquad\qquad \sqcap \forall \mathtt{B}.(\mathtt{NODE})\\
&\quad \sqcap \forall \mathtt{B}.\mathtt{NODE})
\end{aligned}
$$

5.3 OWLList

Drummond et al. (2006b,a) proposed an OWL pattern for lists in OWL that allows for instance classification and subsumption reasoning. It is based on the idea of a *linked list* representation. The scaffold for this representation are the roles `hasNext`, `isFollowedBy` and `hasContents`.

Figure 5.2: Example of a pattern expressed as OWLList



$$
\begin{aligned}
\mathtt{ABCList} \equiv\, & \mathtt{OWLList} \sqcap \exists \mathtt{hasContents}.A \sqcap \exists \mathtt{hasNext}.(\\
& \mathtt{OWLList} \sqcap \exists \mathtt{hasContents}.B \sqcap \exists \mathtt{hasNext}.(\\
& \mathtt{OWLList} \sqcap \exists \mathtt{hasContents}.C \sqcap \exists \mathtt{hasNext}.\\
& \mathtt{EmptyList}))
\end{aligned}
$$

Figure 5.2 illustrates how a “ABC list” pattern can be defined, i.e. a pattern for a list with exactly three subsequent components where the first component is restricted to be of class A, the next to be of class B and the final component of class C. While the pattern explicitly used the `hasNext` relationship, the `isFollowedBy` relationship is inferred.

A class for a ABC list, i.e. a list with just three elements where the first element is restricted to be of class A, the next to be of class B and the final element of class C, can be defined as follows:

Table 5.1: Axiomatisation of the OWLList concept

$$\texttt{OWLList} \equiv \forall \texttt{isFollowedBy.OWLList} \tag{5.1}$$
$$\texttt{hasNext} \in \texttt{fun} \tag{5.2}$$
$$\texttt{hasNext} \sqsubseteq \texttt{isFollowedBy} \tag{5.3}$$
$$\texttt{isFollowedBy} \in \texttt{tra} \tag{5.4}$$
$$\texttt{isFollowedBy} \sqsubseteq \texttt{OWLList} \times \texttt{OWLList} \tag{5.5}$$
$$\texttt{EmptyList} \equiv \texttt{OWLList} \sqcap \neg\exists \texttt{isFollowedBy.}\top \sqcap \exists^{\leq 0}\texttt{hasContents.}\top \tag{5.6}$$
$$\texttt{hasContents} \in \texttt{fun} \tag{5.7}$$
$$\texttt{hasContents} \sqsubseteq \texttt{OWLList} \times \top \tag{5.8}$$

The `EmptyList` construct acts as *terminal symbol*. Leaving out the terminal symbol, it is also possible to define list that starts with a defined sequence (here ABC) and is followed by arbitrary elements. Using different DL constructs, this definition can be used to define more complex expressions such as given in Table 5.2.

The patterns can be used for classification tasks. Given a list of patterns described using `OWLList` and instances, e.g. $\texttt{hasNext}(a, b)$ and $\texttt{hasNext}(b, c)$, a DL reasoner can determine which classes subsume the instances. Further, the subsumption hierarchy of (explicitly defined) list patterns can be automatically inferred, e.g given list patterns for "AB. . . ", "ABC" and "ABC. . . ", the reasoner would infer "AB. . . " *subsumes* "ABC. . . " *subsumes* "ABC". Such reasoning support gets more interesting for more expressive patterns, e.g. if content is defined in terms of its properties (see pattern 8, table 5.2).

As the direct DL notation of complex OWLList patterns like in the above example is hard to read, Drummond et.al. illustrated their examples by a simplified notation that resembles regular expressions. Unfortunately, this notation has not been formalised.

We will later use OWLList as a starting point to represent chord sequence patterns (cf. section 14). Some representational issues regarding instance labelling and chord subsumption are discussed, eventually leading to an extended sequence language that we call $\mathcal{SEQ}$. Further, continuing the idea of providing a readable syntax that resembles regular expressions, a succint yet formal notation that integrates with DL syntax (cf. chapter 8) and an API (cf. chapter 11) is developed.

Suffix trees and *OWLLists* both demonstrate the use of DL for pattern based classification. Such a classification is provided as a standard reasoning service by DL reasoners. Hirsh and Kudenko (1997) further state the possibility of learning suffix tree patterns using the least common subsumer algorithm described by (Cohen et al., 1992). In how far such services are possible for the more expressive OWLList pattern has not been studied, yet. This could for example be useful for discovery of common patterns in chord sequences.

	informal notation	meaning	examples
1	(A, B, C)	Exactly ABC (terminated)	abc
2	(A^*)	A list consisting only of As	aaa, aa (or empty)
3	$(A, B, C, \ldots)$	Starting with ABC (nonterminated)	abc, abcx
4	$(\ldots, A, B, C)$	Ending With ABC (terminated)	abc, xabc
5	$(\ldots, A, B, C, \ldots)$	Containing ABC	abc, xabc, xabcx, abcx
6	$(A^+, B, \ldots)$	One or more As followed by B	ab, aaab
7	$([A\|B\|C], B, C)$	A or B or C, followed by B then C	abc, bbc, cbc
8	$(\mathit{hasProp\ some}\ X, B, C)$	Restriction followed by B then C	Any abc where a hasProp x
9	$\mathit{not}(A, B, C, \ldots)$	Not starting ABC	cbaxx
10	$((A, B, C, \ldots), (D, E, F, \ldots))$	Starting ABC, followed by anything, followed by DEF, followed by anything	abcdef, abcxxdefx
11	$(A, B, C, \ldots)$ and (. . . , A, B)	Starting ABC, and ending AB	abcab, abcxxab
12	()	Empty list	(nil)

Table 5.2: Informal summary of OWLList expressivity (adapted from Drummond et al. (2006b))

5.4 Ordered List Ontology

The Ordered List Ontology (OLO) Specification[3] provides basic concepts and properties for persisting ordered lists as RDF graphs. It provides vocabulary to describe sequential order and indexing elements of the list. However, it is weakly axiomatised in the sense that neither order and indexing nor their interaction is captured in OWL but only the existence of order-related properties is defined as shown in Table 5.3.

[3] http://purl.org/ontology/olo/orderedlistontology.html

Table 5.3: Summary of the Ordered List Ontology in DL notation (v0.72, 2010-07-23)

$$\begin{aligned} \texttt{OrderedList} \sqsubseteq {} & \exists^{=1}\texttt{index} \sqcap \exists^{\leq 1}\texttt{next} \sqcap \exists^{\leq 1}\texttt{previous} \\ & \sqcap \exists^{\geq 1}\texttt{item} \sqcap \exists^{\leq 1}\texttt{ordered\_list} \end{aligned} \tag{5.9}$$

$$\texttt{Slot} \sqsubseteq \exists^{=1}\texttt{length} \tag{5.10}$$

Example 5.1 (LOD using OLO)

```
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix dc:      <http://purl.org/dc/elements/1.1/> .
@prefix olo:     <http://purl.org/ontology/olo/core#> .
@prefix mo:      <http://purl.org/ontology/mo/> .
@prefix ex:      <http://example.org/> .

ex:FunkyPlaylist a olo:OrderedList ;
   dc:title "Funky Playlist" ;
   dc:description "A playlist full of funky legends" ;
   dc:creator <http://foaf.me/zazi#me> ;
      olo:length 2 ;
      olo:slot [
         olo:index 1 ;
         olo:item ex:SexMachine
      ] ;
      olo:slot [
         olo:index 2 ;
         olo:item ex:GoodFoot
      ] .

ex:SexMachine a mo:Track ;
   dc:title "Sex Machine" ;
   dc:creator dbp:James_Brown .

ex:GoodFoot a mo:Track ;
   dc:title "Good Foot" .
```

5.5 *Representation of Time*

Time plays an important role in music. In the context of knowledge representation several theories of time have been developed. Several choices of temporal primitives have been discussed: intervals (Allen, 1984), points (McDermott, 1982) or events (Kowalski and Sergot, 1989). The choice of primitives emphasises the dimension of time considered as fundamental in the problem solving context, but have been proved equivalent in terms of expressiveness (Tsang, 1987).

Efforts have been made to express temporal models as OWL ontologies. However, these theories cannot directly be expressed in OWL DL as they are undecidable and the expressivity of OWL DL is restricted to guarantee decidability. Thus many interesting temporal reasoning tasks are not supported by standard OWL reasoners. Language extensions that include restricted forms of temporal reasoning have been proposed. More pragmatically, some ontologies provide vocabulary for temporal relationships without fully capturing their semantics.

Time as domain. Time can also been treated as a domain within a given DL. Allen's Interval Algebra (Allen, 1984) is a calculus for temporal reasoning that defines 13 base relations to capture the possible relations between two intervals such as "X begins before Y" or "X overlaps with Y". Allen's temporal relationships have been adapted in the DOLCE top ontology, and ported to OWL-DL ($\mathcal{SHI}$)[4].

[4] `http://www.loa-cnr.it/ontologies/TemporalRelations.owl`

OWL-Time. The OWL-Time ontology[5] was initially developed to represent the temporal content of Web Pages and the temporal properties of Web Services, but is currently also used as basis for the OMRAS2 Music Ontology (see section 6.1). The ontology is built around the class *TemporalEntity* and the relations describing its individuals. Temporal entities may be of two types: *Instant* (point-like moments in time) and *Interval* (time descriptions having a duration, represented as ordered pairs of Instant individuals). Additionally, the ontology contains classes meant to describe several other temporal concepts, such as duration (DurationDescription), dates and times (DateTimeDescription), temporal units (TemporalUnit) and alternative representations for the days of the week (DayOfWeek).

[5] `http://www.w3.org/2001/sw/BestPractices/OEP/Time-Ontology`, W3C working draft, as of September 27th, 2006

The main relations describing individuals of type Instant are the functional properties begins (the beginning, same as the end in case of an instant) and ends (the end, same as the beginning in case of an instant). The range of the properties are objects of type InstantThing. The actual time belonging to these individuals can be expressed as of the internal type CalendarClockDescription or as xsd:dateTime of XML Schema. Intervals exhibit, amongst others, the same properties as instants, with the same flexibility in the representation of the actual date and/or time. Additionally, the property inside may be defined in the case of individuals of type Interval, and describing a relation between an Instant and an Interval, equivalent to asserting that some individual of type Instant is within the bounds of the individual of type Interval.

In the context of designing a temporal language, the expressiveness offered by OWL-Time addresses mainly the issue of the representation of time in its qualitative nature. However, as Milea et al. (2008) point out, due to a lack of representational power of OWL-DL the reasoning possibilities remain limited. For example, it cannot be represented in OWL DL that the left bound of an interval should always be smaller than its right bound. The semantic limitation regarding temporal aspects translates to little or no reasoning support in such contexts.

An approach presented in Welty and Fikes (2006) incorporates perdurants through the use of timeslices and fluents in OWL-DL. Perdurants are those entities for which only a part exists if we look at them at any given snapshot in time. When we freeze time we can only see a part of the perdurant. Taking 'running' as an example, one can observe that if we freeze time then we only see a part of the running, without any previous knowledge one might not even be able to determine the actual process as being a process of running. The fundamental building blocks of the OWL-DL representation are timeslices and fluents. Timeslices represent the temporal parts of a specific entity at clear moments in time and the concept itself is then defined as all of its timeslices. Fluents are nothing more than properties that hold at a specific moment in time, may this time be an instant (point-like representation of time) or an interval.

The approach in Welty and Fikes (2006) comes to address a relevant issue, namely the representation of change in OWL ontologies. However, as in the case of OWL-Time, relying solely on OWL-DL for such a goal limits in several ways the power of the approach. The semantic limitation regarding temporal aspect translates to little to no reasoning support in such contexts.

Temporal Extensions of DLs. The work on temporal DLs has focused on combining DLs with well known temporal logics such as linear temporal logic (LTL).

For surveys on temporal extensions of temporal DLs see Artale and Franconi (2001); Lutz et al. (2008).

As we have seen, there exist different notions of time. In the context of DLs mainly *point-based* and *interval-based* notions of time have been investigated. The approaches to adding a notion of time to a DL can further be differentiated into an *implicit* and an *explicit* approach. In the former, temporal information is only implicit in the language. In the later, an explicit notion of time is adopted.

Explicit representations of time in DLs take either an *external* or *internal* point of view. In the external view the same individual can have different snapshots in different moments describing various states of the individual at these times. In the internal point of view different states of an individual are seen as different individual components.

In summary, temporal reasoning based on numerical representations is currently difficult with DLs in the Semantic Web and has mainly been discussed in the literature in terms of extensions of DLs.

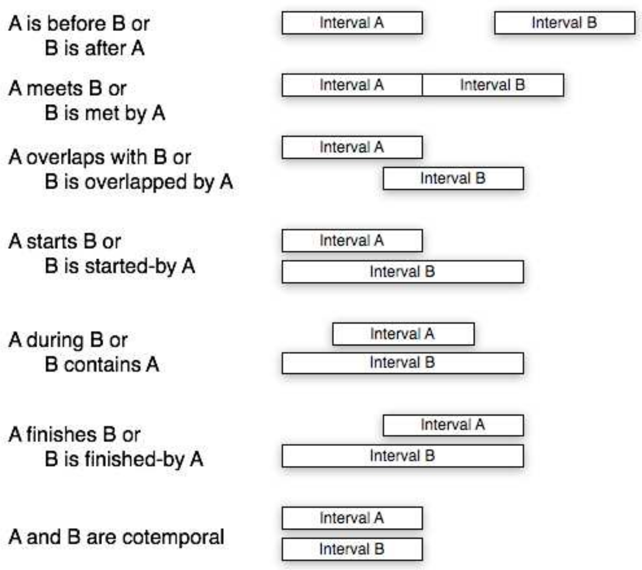


Figure 5.3: 13 possible relationships between two intervals (Allen, 1984)

5.6 *Summary*

RDF provides vocabulary to describe sequences (cf. section 5.1). This lacks however means to specify sequential patterns that can describe the content of a sequence and can constrain its structure. Further technical issues exist that make it incompatible with OWL. Several alternatives have been proposed that reflect different modelling approaches. OWLList uses a successor-role representation for sequences and allows to define sequential patterns that resemble regular expressions (cf. section 5.3). Another approach is to label the roles of a suffix tree with a string (cf. section 5.2). This representation allows for substring matching. In the context of linked web data (LOD) ontologies have been provided that contain vocabulary for expressing sequential relationships while not supporting formal reasoning (cf. section 5.4). Reasoning based on numerical representations is currently difficult with DLs in the Semantic Web though several temporal extensions for DLs have been discussed in the literature (cf. section 5.5.

Of the surveyed approaches OWLList and suffix trees are most interesting for the design of sequential patterns as they provide high expressivity and have reasoning support. Suffix trees have limited means to describe the elements of a sequence in comparison to OWLList which is a limitation for the use case of combining sequential abstraction with abstractions of musical entities such as chords. In chapter 8 the new sequential pattern language $\mathcal{SEQ}$ will be specified that addresses shortcomings of OWLList and further extends its expressivity.

6 Music representation

In music applications and research several forms of representations are used that deal with different aspects of music. The representation can be roughly categorised as *audio representations*, *symbolic representations* and *metadata representations* (see Figure 6.1).

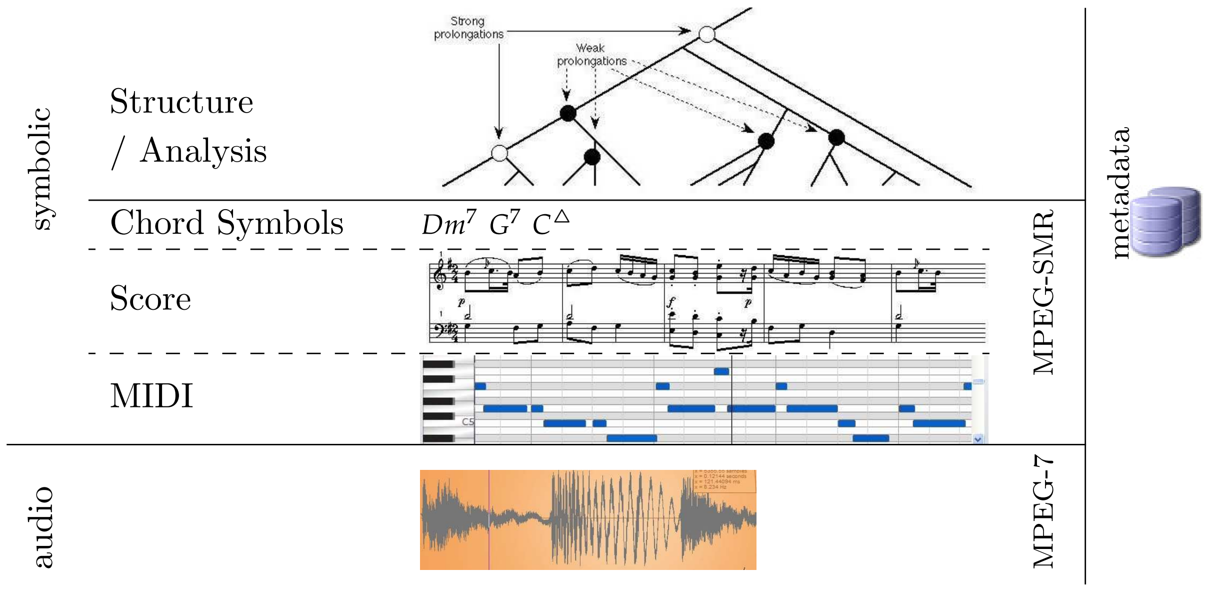


Figure 6.1: Brief overview of Music Representations

Audio. Audio representations such as wave format or MP3 represent sound waves more or less directly in numerical stream formats.

Symbolic representation. Symbolic forms encode music in terms of the discrete objects of action or perception, most notably common western music notation and MIDI. In music notation the main objects are notes, which contain pitch and metrical timing information[1]. Metrical timing is expressed relative to a grid of beats and bars, which can be performed with varying tempo, rater then absolute time. The MIDI format is more closely related to performance, as is was originally designed to control a synthesisers with external keyboard[2]. In addition to note information, there are lyrics and harmony symbols which commonly appear in symbolic representations, providing information for singers and players.

[1] For a more information see e.g. Gehrkens (1914).

[2] For the technical specification see International MIDI Association (1983).

Metadata. Information that are not directly part of the music, but related, such as the composer's name, the date of composition, copyright information etc. are generally referred to as metadata. The distinction between data and metadata is not always clear and can depend on the use case.

6.1 OMRAS2 Music Ontology

The OMRAS2 Music Ontology project[3] develops a formal framework for dealing with music-related information on the Semantic Web, including "editorial, cultural and acoustic" information (Raimond et al., 2007). The ontology is intended to act as "a grounding for more domain-specific knowledge representation". The OMRAS2 Music Ontology Specification aims at providing main concepts and properties for describing music (i.e. artists, albums, tracks, but also performances, arrangements, etc.) on the Semantic Web.

[3] `http://www.musicontology.com/`

The music ontology is based on OWLTime (see section 5.5). The timeline concept of OWLTime has been extended to been able to express multiple timelines and relationships between them.

The vocabulary of Music Ontology[4] is clustered around the three categories:

[4] `http://purl.org/ontology/mo/`

Level 1 aims at providing a vocabulary for simple editorial information (tracks/artists/releases, etc.)

Level 2 aims at providing a vocabulary for expressing the music creation workflow (composition, arrangement, performance, recording, etc.)

Level 3 aims at providing a vocabulary for complex event decomposition, to express, for example, what happened during a particular performance, what is the melody line of a particular work, etc.

The core specification of the Music Ontology does not provide vocabulary for symbolic music representations. However, different drafts exist that describe symbolic music concepts, namely the chord ontology[5] and tonality ontology[6].

[5] see `http://motools.sourceforge.net/chord_draft_1/chord.html`

[6] see `http://motools.sourceforge.net/doc/tonality.html`

As concrete chords (e.g. Cmaj7) are modelled as individuals, this definition cannot directly be used for classification tasks such as chord subsumption.

A further effort in the direction of symbolic music representation has been made with the OMRAS2 chord ontology draft.

6.2 Harmony OWL

Ibbotson (2009) developed *Harmony OWL*, an ontology that provides vocabulary for the use of harmonic annotation in a real-time event processing system. In this ontology, musical note, chord and key objects together with their relationships are described.

Harmony OWL has not been adopted widely yet. The representation and reasoning capabilities are not well explored. Ibbotson (2009,

TBox:

concepts

Chord, *Event*, *Interval*, *Modifier*, *Note*

$$\begin{array}{rcl} \mathit{ChordEvent} & \sqsubseteq & \mathit{Event} \\ \mathit{Natural} & \sqsubseteq & \mathit{Note} \\ \mathit{ScaleInterval} & \sqsubseteq & \mathit{Interval} \\ \mathit{SemitoneInterval} & \sqsubseteq & \mathit{Interval} \end{array}$$

roles

$$\begin{array}{r} \mathit{degree} \sqsubseteq \top \times \top \\ \mathit{interval} \sqsubseteq \mathit{Chord} \times \mathit{Interval} \\ \mathit{without\_interval} \sqsubseteq \mathit{Chord} \times \mathit{ScaleInterval} \\ \mathit{root} \sqsubseteq \mathit{Chord} \times \mathit{Note} \\ \mathit{chord} \sqsubseteq \mathit{ChordEvent} \times \mathit{Chord} \\ \mathit{degree} \sqsubseteq \mathit{ScaleInterval} \times \mathbb{N} \\ \mathit{modifier} \sqsubseteq \mathit{Note} \sqcap \mathit{ScaleInterval} \times \mathit{Modifier} \\ \mathit{natural} \sqsubseteq \mathit{Note} \times \mathit{Natural} \\ \mathit{semitone\_interval} \sqsubseteq \mathit{SemitoneInterval} \times \mathbb{N} \end{array}$$

ABox:

$\mathit{flat}, \mathit{sharp}, \mathit{doubleflat}, \mathit{doublesharp} : \mathit{Modifier}$

$$\left.\begin{array}{l} Cb \in \mathit{Note},\ \mathit{modifier}(Cb, \mathit{flat}),\ \mathit{natural}(Cb, C) \\ C \in \mathit{Natural} \\ Cs \in \mathit{Note},\ \mathit{modifier}(Cs, \mathit{sharp}),\ \mathit{natural}(Cs, C) \\ \vdots \\ Bb \in \mathit{Note},\ \mathit{modifier}(Bb, \mathit{flat}),\ \mathit{natural}(Bb, C) \\ B \in \mathit{Natural} \\ Bs \in \mathit{Note},\ \mathit{modifier}(Bs, \mathit{sharp}),\ \mathit{natural}(Bs, C) \end{array}\right\} \text{pitch definitions}$$

Table 6.1: Summary of the OMRAS2 Chord Ontology in DL notation

ABox excerpt:

Ds:min7(*b3,9)/5 : *Chord*

/* root description */

root(Ds:min7(*b3,9)/5, *_root*1)	*_root*1 $\in$ *Note*	
	modifier(*root*1, *sharp*)	
	natural(*root*1, *note*/*D*)	
bass $\in$ *ScaleInterval* *degree*(*bass*, 5)		
base_chord(Ds:min7(*b3,9)/5, min7)		
without_interval(Ds:min7(*b3,9)/5, _wout)		
	_wout $\in$ *ScaleInterval*	*degree*(_wout, 3)
		modifier(_wout, flat)
interval(Ds:min7(*b3,9)/5, _int1)	*_int* $\in$ *ScaleInterval*	*degree*(_int1, 1)
interval(Ds:min7(*b3,9)/5, _int5)	*_int*5 $\in$ *ScaleInterval*	*degree*(_int5, 5)
interval(Ds:min7(*b3,9)/5, _int7)	*_int*7 $\in$ *ScaleInterval*	*degree*(_int7, 7)
	modifier(_int7, flat)	
interval(Ds:min7(*b3,9)/5, _int9)	*_int*9 $\in$ *ScaleInterval*	*degree*(_int9, 9)

Table 6.2: Example Chord Description in DL notation (translated from example in RDF/Turtle notation from `http://motools.sourceforge.net/chord_draft_1/chord.html`)

p.115) suggests to explore “longer-lasting” chord structures, which is supported by the abstraction techniques developed in this work.

6.3 Chord Symbols and Chord Sequences

Chords symbols can be found in a broad range of data on the web, e.g. song collections in text format (alternating lines of text and chord symbols), music notation with chord symbols, guitar tablatures, and there is even a specific markup language for adding chord symbols to lyrics, called ChordML (Frederico, 2002). There is no formal standard for symbols and their meaning in music theory, but there are some widely accepted conventions in mainstream music theory, such as using scale degrees and functions.

At the lowest level, chord symbols classify the notes sounding simultaneously or within a short time. Chords symbols describe the *root* and *type* (or mode) of chords and alterations or extensions of the basic structure of these types. Although these notes may be arranged differently with regards to their distribution over voices and octaves, this specification is sufficient to ensure that it fits with a given melody. This level of description is used in Jazz and Pop music (e.g. in lead-sheets and guitar songbooks).

A typical example of a chord symbol is $Am7$, which means that the chord has the root A, is of type minor (that defines it contains the pitches A, C, and E) and a 7th is added (the note G).

Chord sequences are a widely used notation for harmonic progressions. (Harte et al., 2005) proposed a text representation for musical chord symbols that is closely resembles the notation used by (jazz) musicians for musically trained individuals to write and understand, yet highly structured and unambiguous to parse with computer programs.

6.4 Harmonic Analysis

Harmony is a major topic in music analysis. Harmonic analysis is taught to musicology students, and it has been the subject of computer based research (e.g. Temperley (1997), Hiraga (1989)). However, computer music software and standard data formats provide little means for representing harmonic structure of music. The interest in the semantic annotation of music, especially automatic annotation, has in recent years mainly focused on music information retrieval based on audio-related features and the classification of genres, styles, or artists. The musical content and structure as it is represented in traditional musical notation and analysis is rarely dealt with.

The harmonic structure of music is important to both music theory and practice, as chord symbols are used by musicians in jazz and pop music, they appear in figured bass in baroque music, and they are used in music theory for all styles.

In music theory, an analysis of the harmonic structure of a piece is started by subsuming simultaneous or successive notes to chords, chords to chord classes, chords classes to functions and functions to progression patterns. This structure provides condensed information on the harmonic characteristics of a piece, helping musicians to understand, memorise, and play the music. Therefore, the creation of these kinds of annotation is very useful for applications in musicology, music education, and music information retrieval.

Music theorists often speak of the 'rules' of harmony, e.g. Fux' gradus (Fux, 1967), or preference rules as used by Lerdahl and Jackendoff (Lerdahl and Jackendoff, 1983) or Temperley (Temperley, 2001). Although these rules are not formally specified (the works mentioned before are insofar exceptions as they do formalise their rules to some degree), it is desirable for a musical knowledge representation to represent these rules. In music as a form of art rules can be broken, but they can still give a description of relevant harmonic structures at least within a given style. Appropriately formalised, rules can be specified to a degree where they can be used as the basis for automatic music analysis to generate harmonic structural descriptions. Annotating music manually can be tedious, therefore a method for automatic annotation can support a musical expert in creating a machine readable representation of musical knowledge, and it helps understand the structure of music better.

6.5 Harmonic Structure

Harmony in general describes the sounding of several notes simultaneously. For analytical purposes, the harmonic dimension of music is represented by symbols that abstract from the actual notes to classes of chords. This abstraction occurs on the pitch dimension, as octave registers are mostly not taken into account, and on the time dimension, as the harmonic symbols are normally assigned to measures or half measures, which may have notes arranged differently over time. The first abstraction is to classify a set of notes or a time period by assigning a chord symbol, which we assume to be given. The chord symbols are often provided in scores for Jazz or Pop Music. The higher levels put these symbols into a structural context. In tonal music, this context consists of progression patterns and cadences, which are modelled in the rule system we describe here.

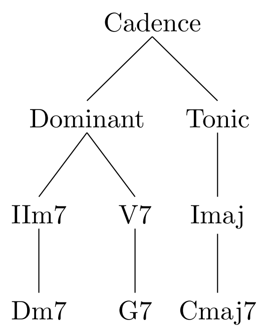
6.6 Higher Structural Levels

Based on the low level chord symbols, structural patterns have been identified, that describe the relationships between different harmonies in a given context. The first level of these are scale degrees, that put a chord in relation to a key, respectively its root note. E.g. *IIm7* describes a chord on the second degree of a scale, again in minor with an added 7th. The chord on the first degree (i.e. the root note) is called the tonic.

In a piece of music, the chord progression usually departs from the tonic and returns to it several times. The departing and returning to the tonic is called a *cadence* according to (Riemann, 1877). Although music theorists use the term cadence in different ways, it will be used only with this meaning here. Within a cadence, chords are classified as having certain functions. The tonic is the most stable function, while the dominant (mainly the V7 chord) implies a sense of tension leading to the tonic.

A harmonic analysis starts with the marked up chords symbols (which are often given on metrical units such as whole or half bars) and interprets them as a sequence of cadences. By doing that, a key is implied. This key is often stable but in many pieces of music it will change at some point in a modulation. A modulation is created by chords that cannot be interpreted in the current key and imply therefore a change of key that is often designed in a way that the key is ambiguous over some chords.

A simple typical example of a cadence in jazz is *Dm7 – G7 – Cj*. The *m7* after the *D* indicates a minor triad with a minor 7th added. The *j* after the C indicates a major triad with a major seventh to be added, which makes it a typical tonic chord. In Jazz, the first *m7* chord is in this constellation seen as closely coupled with the following dominant chord as a dominant function to the key a fifth below the dominant chord (Jungbluth, 1981; Haerle, 1980). This chord sequence is therefore normally (depending on the context) analysed as a Cadence in C major, consisting of the degrees II, V, and I. The analysis can be represented as a tree in this form:



6.7 Grammars for Automatic Harmonic Analysis

While traditional music theory treats harmony on an intuitive level, some efforts have been made to develop more formal models of harmony. The analysis can even be created automatically, when a sufficient set of rules is defined. This can be done using a formal grammar, that defines, how symbols on different levels can be related. They have proven useful for describing natural and formal languages.

Grammars have been used in various musical contexts, see for instance Eberlein (1992), Sundberg and Lindblom (1976), and Roads (1987). The grammar implemented here is a subset of that defined by Weyde (1994), which extends the works by Steedman (1984) and Ulrich (1977) on grammars for jazz harmony. The main difference to Ulrich's work is that his system tries to find regions within a piece, where one scale can be used. This is not necessarily identical to the cadences within one key. The grammar introduced by Steedman does introduce more harmonic structure, but it mixes harmony with metrical structure, and it is not strictly context-free, which makes the processing potentially more demanding.

The grammar by Weyde (1994) includes IIm-V patterns, substitutions, and recursive circle-of-fifth cadences. It has variables that regulate the use of style-specific features, such as the use of the bVII7 in BeBop, cf. Potter (1989), or the use of dominant-7 chords on the tonic in Blues.

As there are patterns, which occur with variations and extensions, it is necessary to include additional levels, which may seem redundant here. E.g. a dominant can be repeated (with its dependant IIm7), so that an intermediate level has to be included that we called a dominant range.

In a basic grammar notation, a rule for simple cadences consisting of dominant and tonic could look like this:

```
Cadence       -> DominantRange TonicRange
TonicRange    -> Tonic
DominantRange -> Dominant
DominantRange -> Dominant Dominant
```

The third rule uses two dominant symbols which may relate to different realisations of the dominant function, e.g. using a *D*7 and a *D*79. To make sure that the chords in the Cadence actually relate to the same key, information about the root needs to be added:

```
Cadence(Root) -> DominantRange(Root) TonicRange(Root)
DominantRange -> Dominant(Root) Dominant(Root)
DominantRange -> Dominant(fifth(Root)) Dominant(Root)
```

The second rule now describes a sequence of two dominants, while the third contains the dominant to the dominant, which is a fifth higher.

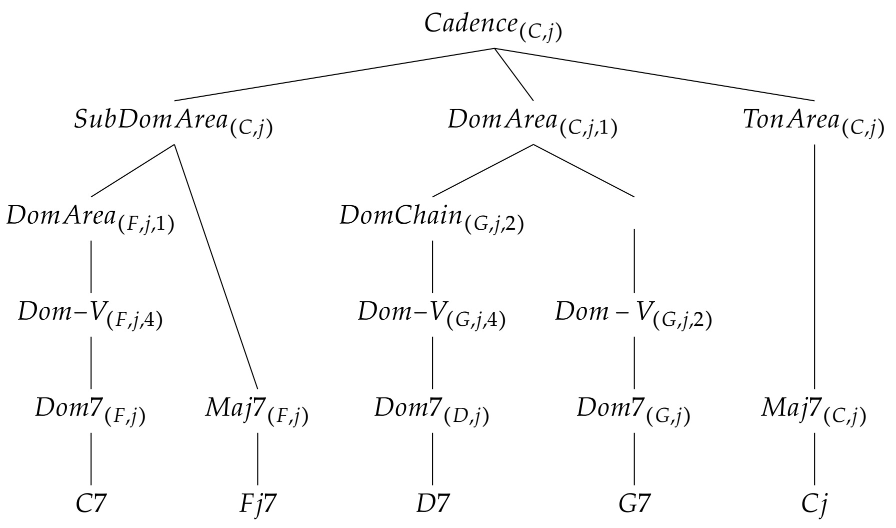


Figure 6.2: Example Harmonic Analysis by a DCG

6.8 Summary

This chapter presented approaches to music representation and reasoning. First, a brief overview of music knowledge representation was given. Notably, the focus on different aspects of music such as score, audio or metadata often results in specialised representations that are not necessarily interoperable. Chord sequences are interesting as they are at the intersection of several approaches and have been used for example as generalisation of score representations and audio streams, as well as building block of harmonic models.

Afterwards, the state of the art of music representation on the Semantic Web was discussed. The OMRAS2 Music Ontology provides an ontology for editorial metadata in RDF/OWL and has been largely adopted in the linked web data community for publishing and querying metadata (cf. section 6.1). In this context symbolic music representation has been addressed with the OMRAS2 Chord Ontology that provides a large catalogue of decomposed chord symbols. Harmony OWL (cf. section 6.2) provides a description of keys, chords and simple temporal relationships for use as datatypes in a distributed event processing system. Both the OMRAS2 Chord Ontology and Harmony OWL do not formally describe the logical relationships between the represented entities and thus have limited support for automated reasoning tasks such as chord classification. The approach taken in Part II is orthogonal to the existing ontologies as it focuses on modelling of structural patterns and reasoning support in OWL. Yet, a major benefit of Semantic Web technologies is the possibility to reuse and integrate existing data and schemata. Schema integration with OMRAS2 Chord Ontology and reuse of musical metadata in federated queries is shown in chapter 13 and chapter 16.

Harmonic representation and analysis are major areas of musicological research and several techniques to automate analytical tasks have been developed in the symbolic AI community (cf. sections 6.4–6.6). Especially grammars have been successfully used to describe and infer the harmonic structure of chord sequences (cf. section 6.7). The methods presented in these sections are used as yardstick for developing the expressive means of $\mathcal{SEQ}$ leading to a formalisation of grammars in $\mathcal{SEQ}$ (cf. section 10.5). In chapter 15, they will be applied to analyse jazz chord sequences.

Part II

Modelling

This part introduces the original contributions in terms of concepts, ontologies and algorithms for working with chord sequence patterns in the Semantic Web.

The first chapters develops musical and sequential constructs in a modular fashion independent of each other. In chapter 7 the $\mathcal{MEO}$ ontology is developed that models basic musical entities such as pitches, notes, intervals and chords. In chapter 8 the $\mathcal{SEQ}$ ontology is developed which provides expressive means for modelling sequential patterns resembling regular expressions. Notably, both $\mathcal{MEO}$ and $\mathcal{SEQ}$ allow the comparison of entities and patterns in terms of subsumption relationships. In chapter 9 the two modelling aspects are combined and examples are presented of how $\mathcal{MEO}$ and $\mathcal{SEQ}$ can be used together to describe chord sequence patterns such as the twelve bar blues.

To support the modelling of harmonic grammars, chapter 10 presents an algorithm to rewrite grammars to $\mathcal{SEQ}$ patterns.

The part finishes with a description of parts of the implementation of an API for $\mathcal{MEO}$ and $\mathcal{SEQ}$ in chapter 11.

The developed model will be applied to a set of musicological applications in Part III.

7 *Musical Entities and Simultaneities*

In this chapter basic musical concepts are formalised in OWL. We focus on elementary concepts such as pitches, notes and intervals that are used as building blocks to define chords. In later chapters these definitions are used together with sequential patterns to define chord sequence patterns.

Note on Ontological Commitment. The goal of our formalisation is to support web retrieval and analysis of chord sequences using OWL reasoning, for example, by providing small inferences such as translating MIDI to score pitches, chords subsumption relationships between chords or checking if concepts and sequences are well defined. The development of a complete catalogue of musical terminology or of authoritative formal definitions for musicological concepts is outside the scope of this thesis.

7.1 Notes

A note is a sign used in musical notation to represent the relative duration and pitch of a sound. This representation is mainly used by musicians and is a central concept of music theory. Nattiez (1990, p. 81), for example, calls notes the "atoms" of much Western music: discretisations of musical phenomena that facilitate performance, comprehension, and analysis.

The qualities of a note unfold in time. In musical discourse it is common to distinguish between perceptional qualities of a note such as pitch and temporal qualities such as duration. We follow this distinctions and conceptualise a note as being constituted of pitch properties and temporal properties. We may express this in a DL as

$$\begin{aligned}\texttt{Note} \equiv\ & \exists^{=1}\texttt{pitch.Pitch}\\ & \sqcap \exists^{=1}\texttt{duration.Duration}\end{aligned}$$

Several conventions to label pitches exist that reflect different aspects of the underlying phenomenon. Score notes are used as notational abstraction in classical sheet music. In technical settings MIDI pitch encoding is frequently used. We can thus specialise the `Note` concept to include these different forms of description:

$$\begin{aligned}\texttt{MIDINote} &\sqsubseteq \texttt{Note}\\ \texttt{ScoreNote} &\sqsubseteq \texttt{Note}\end{aligned}$$

This definition allows for existence of other note types. Pitches are physical phenomena that are characterised by their frequency. So in settings of acoustic studies or audio processing they are often expressed in Hertz (Hz). For example, we can add a class for the physical properties of notes

$$\begin{aligned}&\texttt{PhysicalNote} \sqsubseteq \texttt{Note}\\ &\texttt{PhysicalNote} \sqsubseteq \exists \texttt{frequency}.\mathbb{R}\end{aligned}$$

In order to restrict the allowed types closure axioms can be used

$$\texttt{Note} \equiv \texttt{MIDINote} \sqcup \texttt{ScoreNote} \tag{c1}$$
$$\texttt{Note} \equiv \texttt{MIDINote} \uplus \texttt{ScoreNote} \tag{c2}$$

Both axioms define that a note has to be a MIDI or score note, but axiom (c1) still allows instances that are of both types at the same time while axiom (c2) is more restrictive. Which commitment is made depends on the application.

Pitch and duration are basic properties of notes.

$$\begin{aligned}\texttt{MIDINote} \equiv\ &\exists \texttt{pitch}.\texttt{MP}\\ &\sqcap \exists \texttt{hasDuration}.\texttt{Tiks}\end{aligned}$$

$$\begin{aligned}\texttt{ScoreNote} \equiv\ &\exists \texttt{pitch}.\texttt{SP}\\ &\sqcap \exists \texttt{hasMetricDuration}.\mathbb{Q}^+\end{aligned}$$

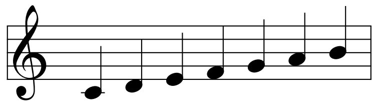
7.1.1 Pitch and Pitch Class

Pitches represent the perceived fundamental frequencies of a sound. A listener assigns perceived tones to relative positions on a musical scale based primarily on the frequency of vibration.

The MIDI specification assigns a number $0\ldots127$ for every possible note pitch (C, $C^\sharp$, D etc.), and this number is included in the message.

$$\texttt{MP} \equiv \exists \texttt{pitchNumber}.\{0\ldots127\}$$

In common score notation, seven pitch names are distinguished that correspond to a C major scale.



score pitch:	c	d	e	f	g	a	b
MIDI pitch:	0	2	4	5	7	9	11

$$\begin{aligned}&\texttt{Natural} \sqsubseteq \texttt{Pitch}\\ &\{c,d,e,f,g,a,b\} \in \texttt{Natural}\end{aligned}$$

In common notation, the modifiers sharp ($\sharp$), flat ($\flat$), and natural ($\natural$) indicate a change of the pitch. A sharp symbol raises the pitch (of a natural note) by one half step; a flat symbol lowers it by one half step.

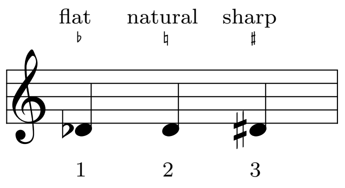


Figure 7.1: Accidentals

$$\begin{aligned}\texttt{SP} &\equiv \exists\texttt{base}.\{c,d,e,f,g,a,b\}\\ &\sqcap \exists\texttt{accidental}.\mathbb{N}\\ &\sqcap \exists\texttt{octave}.\{0\ldots 8\}\end{aligned}$$

7.1.2 Pitch Classes

A pitch class is a set of all pitches that are a whole number of octaves apart, e.g. the score pitch class C consists of the Cs in all octaves. As we can view a pitch class as an abstraction of a pitch w.r.t. octave information we can define it as follows:

$$\begin{aligned}\texttt{SPC} &\equiv \exists\texttt{base}.\{c,d,e,f,g,a,b\}\\ &\sqcap \exists\texttt{accidental}.\mathbb{N}\end{aligned}$$

From the definitions it follows that pitches are subsumed by pitch classes, i.e. it is satisfied that $\texttt{SP} \sqsubseteq \texttt{SPC}$.

Analogously we can define a MIDI pitch class

$$\texttt{MPC} \equiv \exists\texttt{pitchClassNumber}.\{0\ldots 12\}$$

Here, the subsumption between pitch and pitch class do not directly follow as the arithmetic relationship between the two classes is not captured yet. Ideally, we want to express these relationships using arithmetic operations, as a pitch class can be calculated from a pitch by division modulo 12. However, in many cases we can reduce the problem by focusing on arithmetic equations with finite solutions that can be enumerated algorithmically. In this case, the solutions of $p \bmod 12$ are restricted by the finite number of MIDI pitches. This suggests to enumerate the possible cases.

In the following, we will use a template notation with embedded arithmetic expressions in quill brackets '$\{\ldots\}$' and that is used to generate OWL axioms. The template consists of a header that introduces variables. Each variable is required to be defined on a finite domain. The template body consists of a scaffold for an OWL axiom. Axioms are generated by replacing the expressions in quill brackets with their solutions for all possible variable bindings.

The relationship between MIDI pitches and pitch classes can be expressed as

$$\begin{aligned}&\forall p \in [0\ldots 127]:\\ &\qquad \exists\texttt{pitchNumber}.\{p\} \sqsubseteq \exists\texttt{pitchClassNumber}.\{p \mod 12\}\end{aligned}$$

This template generates a set of OWL axioms:

$$\begin{aligned}
\exists\texttt{pitchNumber}.0 &\sqsubseteq \exists\texttt{pitchClassNumber}.0\\
\exists\texttt{pitchNumber}.1 &\sqsubseteq \exists\texttt{pitchClassNumber}.1\\
&\vdots\\
\exists\texttt{pitchNumber}.12 &\sqsubseteq \exists\texttt{pitchClassNumber}.0\\
\exists\texttt{pitchNumber}.13 &\sqsubseteq \exists\texttt{pitchClassNumber}.1\\
&\vdots\\
\exists\texttt{pitchNumber}.127 &\sqsubseteq \exists\texttt{pitchClassNumber}.7
\end{aligned}$$

7.2 Pitch Type Relationships

Pitch types can be ordered according to their generality. In both the enharmonic and non-enharmonic case pitch classes can be seen as generalisations of pitches that abstract the octave information. Our formalisation supports this relationship as `MP` $\sqsubseteq$ `MPC` and `SP` $\sqsubseteq$ `SPC` follow from the definitions of the concept.

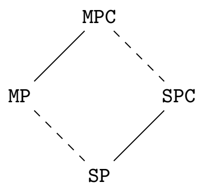


Figure 7.2: Subsumption lattice for pitches (assuming enharmonic equivalence)

Enharmonic Equivalence. In modern music and notation, an enharmonic equivalent is a note (enharmonic tone), interval (enharmonic interval), or key signature which is equivalent to some other note, interval, or key signature, but ”spelled”, or named, differently.

Given a `SPC`, the corresponding `MPC` can be inferred. This can be achieved by adding mapping axioms from enharmonic to numeric representation:

$$\forall nat \in \texttt{Natural}^{\mathcal{I}}, \forall alt \in \{-\epsilon \ldots \epsilon\} :$$
$$\exists diatonic.\{\{nat\}\} \sqcap \exists alteration.\{alt\} \sqsubseteq \exists midiValue.\{\{mpval(nat) - alt\}\}$$

where nat is a natural pitch alt is the number of accidentals within an ϵ-neighbourhood (two will be sufficient for most classical pieces), and $mpval$ is the following map from naturals to pitch class numbers:

$$\begin{aligned}
mpval : &\texttt{Natural} \to \mathbb{N}\\
c &\mapsto 0\\
d &\mapsto 2\\
e &\mapsto 4\\
f &\mapsto 5\\
g &\mapsto 7\\
a &\mapsto 9\\
b &\mapsto 11
\end{aligned}$$

The distribution of values reflects that natural pitches are associated with the ionic tone scale.

Mathematical representations of pitch conversions often don’t restrict the above and we thus cannot express an infinite axiomatisation. However, in common score notation we rarely find accidentals with higher cardinality than double sharps or double flats. It is thus reasonable to limit the possible values for accidentals to ±2 (or ±3), and generate the respective $12 \cdot 5$ (or $12 \cdot 7$) enharmonic equivalence axioms.

7.3 Intervals

In music theory, the term interval describes the relationship between the pitches of two notes. Intervals may be described as:

- *vertical* (or harmonic) if the two notes sound simultaneously, or
- *horizontal* (or melodic, linear), if the notes sound successively.

The *semitone* is smallest interval in Western music.

Corresponding to the classification of different types of pitches (cf. section 7.2), we can distinguish four interval types:

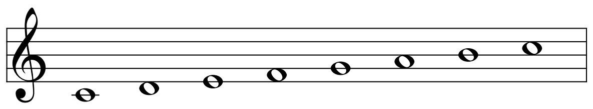
$$\begin{aligned}
\texttt{mpcInt} &\sqsubseteq \texttt{MPC} \times \texttt{MPC}\\
\texttt{mpInt} &\sqsubseteq \texttt{MP} \times \texttt{MP}\\
\texttt{spcInt} &\sqsubseteq \texttt{SPC} \times \texttt{SPC}\\
\texttt{spInt} &\sqsubseteq \texttt{SP} \times \texttt{SP}
\end{aligned}$$

$$\begin{tikzcd}
 & \texttt{mpcInt} \arrow[dl, dash] \arrow[dr, dash, dashed] & \\
\texttt{mpInt} \arrow[dr, dash, dashed] & & \texttt{spcInt} \arrow[dl, dash] \\
 & \texttt{spInt} &
\end{tikzcd}$$

Figure 7.3: Subsumptions between interval relations

Further we can construct a corresponding property subsumption lattice:

The *quality* of an interval is determined by comparison with the intervals found in the *major scale.* When compared with a major scale, a whole step is found to be equal to the distance from the first to the second note, giving us yet another name for it: the *major second.*



Interval relationships between MIDI pitches have the form

$$\begin{aligned}
&\forall n, m \in \{0 \dots 127\}:\\
&\qquad \texttt{MPC}\{n\} \times \texttt{MPC}\{m\} \sqsubseteq \texttt{mpcInt}\{(m-n) \bmod 12\}
\end{aligned}$$

giving a finite set of 12^2 axioms.

7.3.1 Expressible pitch range

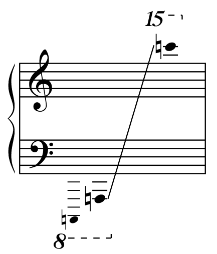


Figure 7.4: Pitch range of a piano.

For our approach it is practical to restrict ourselves to a finite amount of pitches that can be formalised in a finite set of DL axioms. The restriction which pitches are chosen has a certain degree of arbitrariness but can be based on common music practices on and perceptual limits of human pitch recognition. Reasonable choices are for example the 88 pitches A_0 to C_8 (cf. subsection 7.3.1) that form the range of a modern piano. Another choice would be the range of MIDI pitches C_{-1} to G_9 $(1 \dots 127)$ though the additional pitches can be seen as technical artifacts of the seven bit encoding that will hardly be used in performances. The human ear can roughly perceive pitches between $20Hz$ and $20kHz$.

7.3.2 Minor, Diminished, and Augmented Intervals

Consider the half step interval as an example. A *chromatic half step* is a half step written as the same note twice with different accidentals (i.e. $G - G^\sharp$) while a *diatonic half step* is a half step that uses two different note names (i.e. $G^\sharp - A$).

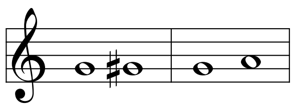


Figure 7.5: Chromatic and diatonic half step

To formalise this situation, we first assert an `equalNatural` role between pitches with the same natural. For each pitch label $l \in \{c, d, e, f, g, a, b\}$ we define a mapping axiom

$$\exists\texttt{hasNatural}.\{\{l\}\} \times \exists\texttt{hasNatural}\{\{l\}\} \sqsubseteq \texttt{equalNatural}$$

$$\texttt{equalNatural} \in \texttt{sym}$$
$$\texttt{equalNatural} \sqsubseteq \texttt{SPC} \times \texttt{SPC}$$

Further inequality is asserted using a role negation:

$$\neg\texttt{equalNatural} \equiv \texttt{unequalNatural}$$

We then can use boolean role conjunctions to infer the harmonic interpretation of a half step (i.e. `spcInt1`):

$$\begin{aligned} \texttt{equalNatural} \sqcap \texttt{spcInt1} &\sqsubseteq \texttt{chromaticHalfStep} \\ \texttt{unequalNatural} \sqcap \texttt{spcInt1} &\sqsubseteq \texttt{diatonicHalfStep} \end{aligned}$$

This definition can analogously be defined for the remaining interval classes.

7.4 Simultaneities

Simultaneities are musical events that occur at the same time. For example a chord can be described as a simultaneity of pitches, though not all pitch simultaneities are necessarily chords.

A simultaneity can be formalised using a container/containment pattern. We define a concept `Simultaneity` , roles `contains` and its inverse `containedIn` to capture containment, and a role `sibling` to hold between neighbour elements in a simultaneity.

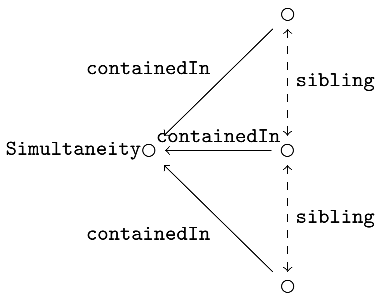


Figure 7.6:
Pattern for representing simultaneities

$$\texttt{Simultaneity} \equiv \exists\texttt{containedIn}.\texttt{Note}$$
$$\texttt{contains} \sqsubseteq \texttt{Simultaneity} \times \texttt{Note}$$
$$\texttt{contains} \equiv \texttt{containedIn}^-$$
$$\texttt{contains} \circ \texttt{containedIn} \sqsubseteq \texttt{sibling} \qquad \text{(sibling)}$$
$$\texttt{sibling} \circ \texttt{containedIn} \sqsubseteq \texttt{containedIn} \qquad \text{(containment)}$$

Axiom (sibling) assert the existence of simultaneity relationships between individuals that are contained in the same structure.

7.5 Chords

A chord in music is a set of notes that is heard as if sounding simultaneously, i.e.

$$\texttt{Chord} \sqsubseteq \texttt{Simultaneity},$$

and has certain characteristics.

The most frequently encountered chords in theory and music are triads, so called because they consist of three distinct notes. Further notes may be added to give seventh chords, extended chords, or added tone chords. The most common chords are the major and minor triads and then the augmented and diminished triads. They constitute the basic *chordal quality.*

$$\texttt{Chord} \equiv \exists\texttt{triad}.\{\texttt{maj},\texttt{min},\texttt{aug},\texttt{dim}\}$$

Tonal centricity is an ubiquitous phenomenon in Western music. Chords invoke the perception of having a tonal centre called the *root* note of the chord. In musical training students learn verbal labelling of the root note as a basic skill. The *root* of a chord is the note or pitch upon which the chord is tonally centred. Note that the root note is not not necessarily the lowest note of a chord. The lowest note of a chord is called the *bass.*

7.5.1 Chord Subsumption

An interesting aspect of chords in music analysis is the question as to whether a given chord is subsumed by another chord. This means that it has the same root and contains the same notes added to the root. The advantage of modelling a chord as a simultaneity is that the notes contained in the chord are explicitly modelled, not just the symbol used to describe it.

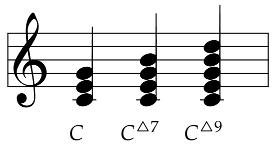


Figure 7.7: Chord Extensions

The music example in Figure 7.7 illustrates the relevance of describing relationships between chord extension. The second chord is clearly a superset of the first, therefore the first concept should subsume the second. The third chord is a superset of the second, and therefore the second and first concept should subsume the third. Using the OMRAS 2 Chord Ontology, only the first subsumption would be detected by a reasoner, while the third chord symbol does not contain the 7 which is part of the second chord. In $\mathcal{MEO}$ both subsumptions would be detected, because the notes are compared rather than the symbols. This is an important feature for harmonic analysis in Jazz, where $C^{\triangle 7}$ and $C^{\triangle 9}$ are both treated as acceptable realisations of a major chord.

7.6 Summary

This chapter described the OWL ontology $\mathcal{MEO}$ for musical entities. Basic musical entities — pitches, notes, intervals and chords — have been modelled. Notably, subsumption can be used to describe and deduce musical relationships such as assigning MIDI pitch classes to score pitches, or organising chord variants according to their extensions.

The next chapter will describe $\mathcal{SEQ}$ constructs — means to describe sequential structures. chapter 9 introduces examples of how $\mathcal{MEO}$ and $\mathcal{SEQ}$ can be combined to describe chord sequence patterns such as the twelve bar blues. Further, Part III uses $\mathcal{MEO}$ in various application scenarios such as harmonic analysis and web queries.

8 Sequential Patterns

This chapter introduces modelling constructs for sequential patterns in OWL, we call $\mathcal{SEQ}$. Constructs that resemble regular expression are introduced. Further, the possibilities to use subsumption reasoning to compare and classify sequential patterns are investigated. Parts of these results have also been published in Wissmann et al. (2010).

8.1 Modelling sequences in OWL

The wish to model sequences arises naturally in the music domain, given its temporal nature. Unfortunately, there are no native constructs within OWL DL to express sequence patterns. Drummond et al. (2006b) proposed to use a *linked list* approach. We extended this approach and developed $\mathcal{SEQ}$, an ontological representation of sequence patterns.

In the following we describe the axiomatisation of basic $\mathcal{SEQ}$ patterns and give examples. The axiomatisation of the linked list structure follows the ideas of Drummond et al. (2006b). One difference is that we introduce an initial component because this is crucial for the behaviour of pattern subsumption and for the creation of more complex pattern constructs. Further we introduce a notation to express sequences in a more intuitive (yet formal) way.

The core structure of a $\mathcal{SEQ}$ pattern is similar to a linked list. Figure 8.1 shows an example. Components of patterns are linked by solid dots. Each component can be associated with linking and content properties: Linking is expressed by using the *functional* property *seq:*`hasNext` (solid arrow) that connects a component to its immediate successor or by using the *transitive* property *seq:*`followedBy` (dashed arrow) that connects a component to all following components. A *pattern* is characterised by restricting these properties. As the subproperty relationship *seq:*`hasNext` $\sqsubseteq$ *seq:*`followedBy` is asserted for $\mathcal{SEQ}$ patterns, *seq:*`followedBy` relationships are are implicitly defined between all connected components (dotted arrows).

The property *seq:*`hasContent` can be used to describe the content of a pattern component. Finally, we introduce an intital component (α) with no precursor and no content and a final component (ω) with no successor and no content (see following section for definitions). A sequence pattern $\mathtt{SP_1}$ that describes sequences that consist of "some instances of `W`, then `X`, then `Y`, then followed by `Z`" (as shown in Fig-

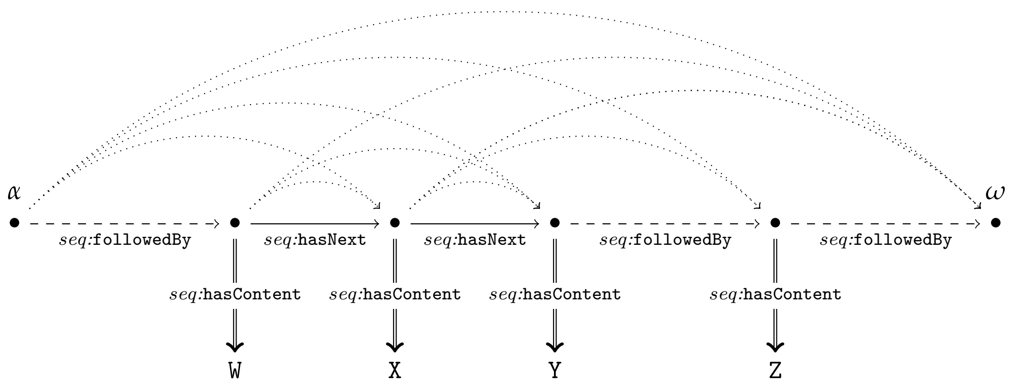


Figure 8.1: Structure of an example sequential pattern

ure 8.1) can be described by the DL concept

$$\begin{aligned} \mathtt{SP_1} \equiv\ & \sqcap\ \exists \mathit{seq}{:}\mathtt{followedBy}.(\exists \mathit{seq}{:}\mathtt{hasContent.W} \\ & \sqcap\ \quad \exists \mathit{seq}{:}\mathtt{hasNext}.(\exists \mathit{seq}{:}\mathtt{hasContent.X} \\ & \sqcap\ \quad \exists \mathit{seq}{:}\mathtt{hasNext}.(\exists \mathit{seq}{:}\mathtt{hasContent.Y} \\ & \sqcap\ \exists \mathit{seq}{:}\mathtt{followedBy}.(\exists \mathit{seq}{:}\mathtt{hasContent.Z} \\ & \sqcap\ \exists \mathit{seq}{:}\mathtt{followedBy}.\omega)))) \end{aligned}$$

For simplification, we can state this expression equivalently in $\mathcal{SEQ}$ notation as

$$\mathtt{SP_1} \equiv [\mathtt{W}] \triangleright [\mathtt{X}] \triangleright [\mathtt{Y}] \cdots [\mathtt{Z}]$$

with α and ω not explicitly stated and arrows, dots and square brackets capturing the details of the succeeds, follows and content restrictions. Consider, the pattern

$$\mathtt{SP_2} \equiv [\mathtt{W}] \triangleright [\mathtt{X}] \triangleright [\mathtt{Y}] \triangleright [\mathtt{Z}]$$

The difference with `SP`$_1$ here is that `Y` has to be directly followed by `Z`. Intuitively we expect that `SP`$_2$ is more specialised than `SP`$_1$ and all instances of `SP`$_2$ will also be instances of `SP`$_1$. As we have formalised $\mathcal{SEQ}$ patterns as DL concepts, we can directly use the machinery for computing DL concept subsumption to automatically compute pattern subsumption. In this case a standard DL reasoner will infer the subsumption relationship $\mathtt{SP_2} \sqsubseteq \mathtt{SP_1}$ (taking into consideration that *seq:*`hasNext` is a subproperty of *seq:*`followedBy`).

The possibilities of subsumption reasoning get more interesting when we use concept *expressions* (such as we have done in the `Musician` example on page 20) rather than simple concept names. For example we could define a chord by its properties, e.g.

$$\begin{bmatrix} & \exists & \mathtt{root} & . & \mathtt{C} \\ \sqcap & \exists & \mathtt{triad} & . & \mathtt{Maj} \\ \sqcap & \exists & \mathtt{seventh} & . & \mathtt{b7} \end{bmatrix}$$

where the pattern characterises a chord by the properties `root`, `triad` and `seventh`. Given another more general pattern that for example only restricts root and triad a reasoner could infer a subsumption relationship such as

$$\begin{bmatrix} \exists & \texttt{root} . \texttt{C} \\ \sqcap \exists & \texttt{triad} . \texttt{Maj} \\ \sqcap \exists & \texttt{seventh} . \texttt{b7} \end{bmatrix} \sqsubseteq \begin{bmatrix} \exists & \texttt{root} . \texttt{C} \\ \sqcap \exists & \texttt{triad} . \texttt{Maj} \end{bmatrix}$$

In the work described in the following section we restrict ourselves to patterns that describe their content as a conjunction of features (functional properties) as we can discover patterns of this form automatically using the pattern discovery method by Conklin (2010). Note, that in principle it is also possible to make use of further DL operators when defining patterns. For example, the pattern

$$\begin{bmatrix} \exists & \texttt{root} . \neg(\texttt{F} \sqcup \texttt{G}) \\ \sqcap \exists & \texttt{triad} . \texttt{Maj} \end{bmatrix}$$

matches major chords that have a root other then F or G, and given our previous example pattern would give rise to the subsumption relationship:

$$\begin{bmatrix} \exists & \texttt{root} . \texttt{C} \\ \sqcap \exists & \texttt{triad} . \texttt{Maj} \\ \sqcap \exists & \texttt{seventh} . \texttt{b7} \end{bmatrix} \sqsubseteq \begin{bmatrix} \exists & \texttt{root} . \neg(\texttt{F} \sqcup \texttt{G}) \\ \sqcap \exists & \texttt{triad} . \texttt{Maj} \end{bmatrix}$$

Naturally the question arises how such patterns can be created in practise. As manual modelling is often costly and time consuming, it is interesting to investigate methods for automatic pattern creation. In the following section we will outline the relationship of the $\mathcal{SEQ}$ formalism to the established viewpoint approach to automatic pattern discovery.

8.2 Approach to defining $\mathcal{SEQ}$ constructs

Concepts, roles and individuals are the modelling primitives of Description Logics. $\mathcal{SEQ}$ constructs translate into one of these types and additionally to a set of axioms that further describe these. We use the notation '$:=_{\mathcal{KB}}$' for the translation from $\mathcal{SEQ}$ constructs to DL entities, where $\mathcal{KB}$ is a knowledge base that the translation is operating on. For some $\mathcal{SEQ}$ constructs the translation requires additional axioms to be included into the knowledge base. We then state that some axioms α_i are included into a knowledge base $\mathcal{KB}$ by writing $\mathcal{KB} \supseteq \{\alpha_i\}$.

8.3 Sequence Pattern

In the following we define syntax and semantics of the $\mathcal{SEQ}$ constructs that were informally introduced in the previous section.

Definition 15 (Content Restriction). *A* content restriction $[C]$ *is defined as a concept expression*

$$[C] :=_{\mathcal{KB}} \exists \mathit{seq}{:}\texttt{hasContent}.C$$

with $\mathcal{KB} \supseteq \{\textit{\textbf{fun}}\ \textit{seq}{:}\texttt{hasContent}\}$.

Definition 16 (Succession). *For two concepts* C *and* D *a* direct successor restriction *is defined as concept expression*

$$C \triangleright D :=_{\mathcal{KB}} C \sqcap \exists \textit{seq}{:}\texttt{hasNext}.D$$

and a general successor restriction *is defined as*

$$C \cdots D :=_{\mathcal{KB}} C \sqcap \exists \textit{seq}{:}\texttt{followedBy}.D$$

Analogously we define corresponding precursor restrictions using inverse role as

$$C \triangleright^{-} D :=_{\mathcal{KB}} C \sqcap \exists \textit{seq}{:}\texttt{hasNext}^{-}.D$$
$$C \cdots^{-} D :=_{\mathcal{KB}} C \sqcap \exists \textit{seq}{:}\texttt{followedBy}^{-}.D$$

and successor negation restriction *using concept negation as*

$$C \ntriangleright D :=_{\mathcal{KB}} C \sqcap \neg\exists \textit{seq}{:}\texttt{hasNext}.D$$
$$C \cdot\!/\!\cdot D :=_{\mathcal{KB}} C \sqcap \neg\exists \textit{seq}{:}\texttt{followedBy}.D$$

We use the symbol $\odot \in \{\triangleright, \cdots\}$ as a notational shortcut in definitions that are defined for each type of succession but do not differ otherwise.

Definition 17 (Initial and Terminal Component). *A sequence is enclosed by an initial component* α *and a terminal component* ω*, defined as the concepts:*

$$\alpha :=_{\mathcal{KB}} \neg\exists \textit{seq}{:}\texttt{followedBy}^{-}.\top \sqcap \exists^{\leq 0} \textit{seq}{:}\texttt{hasContent}.\top$$
$$\omega :=_{\mathcal{KB}} \neg\exists \textit{seq}{:}\texttt{followedBy}.\top \sqcap \exists^{\leq 0} \textit{seq}{:}\texttt{hasContent}.\top$$

8.4 Sequence Instances

Definition 18 (Successor Assertion). *A (direct)* successor assertion *is an individual*

$$c_1 \blacktriangleright \ldots \blacktriangleright c_n :=_{\mathcal{KB}} c_1$$

together with a set of assertions

$$\mathcal{KB} \subseteq \{\textit{seq}{:}\texttt{hasNext}(c_i, c_{i+1}) \mid 1 \leq i < n\}$$

Note that we can omit additional type assertions to state that c_1 and c_2 are instances of *seq:*`Component` here as their type is implicitly defined by assertions of domain and range of the *seq:*`hasNext` relationship.

Definition 19 (Content Assertion). *A* content assertion *is a fresh anonymous individual*

$$[t] \coloneqq_{\mathcal{KB}} \_{:}c$$

for which is asserted that

$$\{seq{:}\mathtt{hasContent}(\_{:}c, t)\} \subseteq \mathcal{KB}$$

Note that by definition of *seq:*`hasContent` this implies also the assertion that $\_{:}c \in seq{:}\mathtt{Component}$.

Some knowledge representation formalism provide means to specify an object by a list of types and features. One example is *frames* (Minsky, 1981). The following notation uses a frame-like style to express properties of the content of a component.

Definition 20 (Content Properties Assertion). *A* content properties assertion *is a content assertionl*

$$\begin{bmatrix} & \in & C_1 \\ & & \vdots \\ & \in & C_n \\ R_1 & . & a_1 \\ \vdots & & \vdots \\ R_m & . & a_m \end{bmatrix} \coloneqq_{\mathcal{KB}} [\_{:}t]$$

where the content $\_{:}t$ *is described by*

$$\mathcal{KB} \supseteq \{\_{:}t \in \bigsqcap_{i=1}^{n} C_i\} \cup \bigcup_{i=1}^{m} \{R_i(\_{:}t, a_i)\}$$

We introduced the concepts α and ω as markers for the initial and terminal component. These can be used to "close" a sequence by asserting individuals of type α and ω at the beginning and end of the list. We incorporate this in our definition of a sequence instance:

Definition 21 (Sequence Assertion). *A* sequence assertion *between individuals* $c_1, \ldots, c_n$ *is defined as*

$$\ulcorner c_1 \blacktriangleright \ldots \blacktriangleright c_n \urcorner \coloneqq_{\mathcal{KB}} \_{:}a \blacktriangleright c_1 \blacktriangleright \ldots \blacktriangleright c_n \blacktriangleright \_{:}o$$

together with the assertions $\_{:}a \in \alpha$ *and* $\_{:}o \in \omega$*.*

We are interested in being able to *refer* to a sequence as a whole in order to make statements about a sequence such as adding editorial metadata or relating it to another sequence. In OWL DL we cannot directly refer to a structure as would be possible in a higher order logic. Rather we are restricted to single individuals and roles between these. Therefore we have to provide an individual that represents a sequence and then either explicitly relate this individual to every component contained in the sequence are point somewhere into the sequence and provide means to compute the contained elements by following role paths. Note that these requirements are already met by our definition of α as it is related to every component of the sequence via a *seq:*`followedBy` role — either explicitly given or inferable.

We can now assert additional statements about the sequence by referring to the representative individual. For example, we can give the sequence an explicit name:

Example 8.1 (Naming a Sequence) *Given above definition we can assert a name for a sequence by writing:*

$$n \coloneqq_{\mathcal{KB}} \ulcorner [c_1] \blacktriangleright \ldots \blacktriangleright [c_{\mathtt{n}}] \urcorner$$

The sequence closure assertion $\ulcorner [\mathrm{c}_1] \blacktriangleright \ldots \blacktriangleright [\mathrm{c}_{\mathtt{n}}] \urcorner$ *is translated to an anonymous individual* $\_{:}\mathrm{a}$ *with the additional axioms that define structure and content. The expression then becomes an equality statement between the named and the anonymous individual, i.e.* $\mathrm{n} = \_{:}\mathrm{a}$.

Further, we can assert a pattern to a sequence:

Example 8.2 (Assigning a pattern) *A* sequence type assertion

$$\ulcorner c_1 \blacktriangleright \ldots \blacktriangleright c_n \urcorner \sqsubseteq C$$

denotes a sequence closure assertion together with an individual equality assertion on its initial component, i.e.

8.5 *Sequence construction and Pattern Labelling*

Finding occurrences of a pattern in a list is a typical query use case. Such a use case requires to relate a pattern either to the whole list or to substructures such as segments or components. Examining the DL concept `OWLList`, we find that it does not correspond to a list in its totality. Rather it describes a component in a linked list pattern. $\mathcal{SEQ}$ uses the concept name *Component* instead of OWLList to clarify this and after a discussion of some preliminaries a concept *Sequence* that identifies a single individual as "being" the list will be introduced.

As patterns are DL concept, i.e. unary FOL predicates, they assign the property of being instances of the pattern concept to single individuals. In the examples by Drummond et al. (2006b), the relationship of this assignment does not always correspond to the intended meaning. Figure 8.2 illustrates the possible effect of different starting constructs on list classification.

The pattern $P1 \equiv [\mathtt{A}] \triangleright [\mathtt{B}]$ is a simple pattern construct that Drummond et al. (2006b) paraphrases as "starts with". In the example

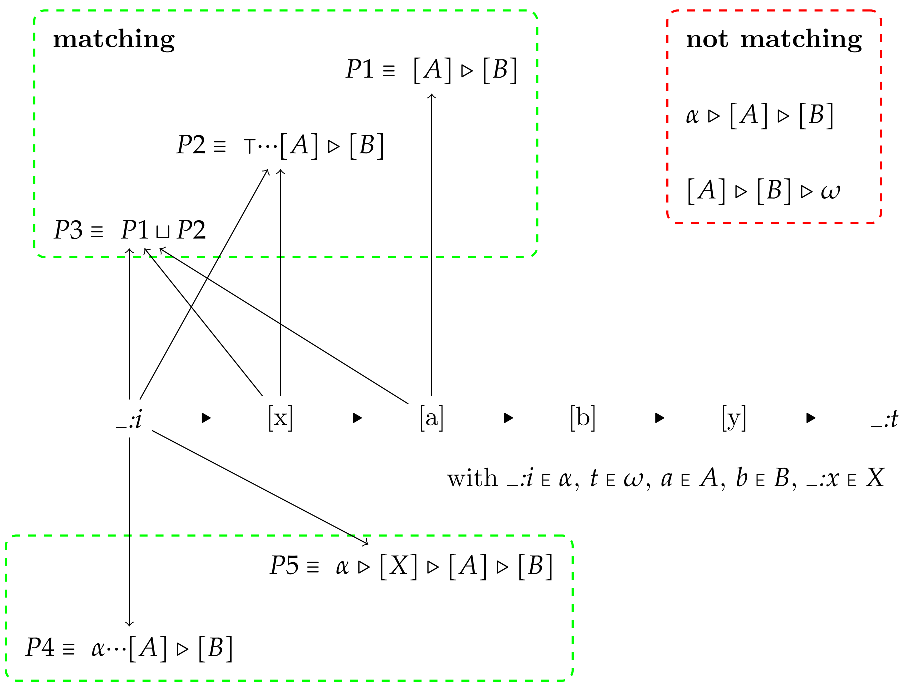

matching
$P1 \equiv [A] \triangleright [B]$
$P2 \equiv \top \cdots [A] \triangleright [B]$
$P3 \equiv P1 \sqcup P2$
not matching
$\alpha \triangleright [A] \triangleright [B]$
$[A] \triangleright [B] \triangleright \omega$
$\_:i$ ▸ [x] ▸ [a] ▸ [b] ▸ [y] ▸ $\_:t$
with $\_:i \in \alpha$, $t \in \omega$, $a \in A$, $b \in B$, $\_:x \in X$
$P5 \equiv \alpha \triangleright [X] \triangleright [A] \triangleright [B]$
$P4 \equiv \alpha \cdots [A] \triangleright [B]$

Figure 8.2: Examples for the effect of different starting constructs on type-classifications. Arrows denote type-relationships.

sequence instance it is just assigned to the single component $[a]$. Classifying or *labelling* the first component instance in this way seems to capture the intended meaning. We call this a γ-labelling. $P2$ labels "something followed by A B". We call the resulting labelling a *pre*-labelling as it labels all concepts that precede the first named component (here [A]). $P3$ is an extension of $P2$ that also includes the named component. Drummond et al. (2006b) discusses this type of construct (in $L5$) as expressing the meaning of list containment. This formalisation seems not intuitive to us. In particular, one would expect a query for pattern-in-list-occurrence to return a single reference to a list rather then several references to components. We address this problem in $\mathcal{SEQ}$ by adding a starting construct α that behaves symmetrically to the ω construct in that it disallows content and preceding components. In our example, α is such a start construct. We can see that given such individual that is guaranteed to proceed all other components, we can now express patterns as $P4$ and $P5$ that exactly label this component. Moreover, $P5$ now captures the notion of containment. Queries for containment would return just this component. We call this an α-labelling.

8.6 Separation of sequential structure and containment

The `hasContent` role separates instances that represent the sequential structure and instances that represent content. To illustrate its necessity, consider the pattern type relationships

$$(a \blacktriangleright b) \in (A \triangleright B)$$

and

$$([a] \blacktriangleright [b]) \in ([A] \triangleright [B])$$

where $a \in A$ and $b \in B$. Both patterns are satisfiable. Yet, they differ in the degree of separation between sequence structure and content domain. In the first case we *implicitly specialised* the type of a and b as the succession relationship is defined between components. Hence, it follows that

$$a \in (A \sqcap \mathit{seq}{:}\texttt{Component})$$
$$b \in (B \sqcap \mathit{seq}{:}\texttt{Component})$$

This style of definition adds additional inferences to the content domain which may be unwanted. For example, a sequence $a \blacktriangleright b \blacktriangleright a$ that contains the same individuals at different positions in the sequence will lead to an inconsistency with the acyclic defined `Sequence` concept, whereas a sequence $[\texttt{a}] \blacktriangleright [\texttt{b}] \blacktriangleright [\texttt{a}]$ may contain same individuals as content while the three distinct anonymous individuals that represent the components ensure the acyclic order.

8.7 Repetition

Definition 22 (One or more C).

$$C^{+} \triangleright D \coloneqq_{\mathcal{KB}} \_{:}R \ \mathit{w.r.t.}\ \mathcal{KB} \supseteq \left\{\_{:}R \equiv C \triangleright (D \sqcup \_{:}R)\right\}$$
$$C^{+} \cdots D \coloneqq_{\mathcal{KB}} \_{:}R \ \mathit{w.r.t.}\ \mathcal{KB} \supseteq \left\{\_{:}R \equiv C \cdots (D \sqcup \_{:}R)\right\}$$

Definition 23 (None or more C).

$$C^{*} \triangleright D \coloneqq_{\mathcal{KB}} D \sqcup (C^{+} \triangleright D)$$
$$C^{*} \cdots D \coloneqq_{\mathcal{KB}} D \sqcup (C^{+} \cdots D)$$

Drummond et al. (2006b) gave examples of how to model repetitions in OWLList and informally introduced star (*) and plus ($^+$) notation repetitive pattern. They first discussed a list of just A's (A^*) that they formalised $L2 \equiv [A] \overset{\forall}{\cdots} [A]$. As obvious in the $\mathcal{SEQ}$ notation, the meaning of this definition is different from a regular expressions, as it actually does not allow an empty list. Further, it is not composable, for example in order to define a pattern of As and then a B as the universal quantifier would disallow a B to follow. $\mathcal{SEQ}$ therefore models star-repetition differently based on plus-repetition.

Drummond et al. (2006b) show how a *recursive* definition can be used to construct a list of “As then B”: $L6 \equiv [A] \triangleright ([B] \sqcup L6)$. This is a nice definition though it leaves open the question how to deal with compositional cases. For the example of the definition of a list of “As then Bs” would require two recursions. The main idea to enable complex composition is to generate new recursive axioms for each plus or star construct. A star pattern is defined based on the observation that all star instances can also be classified either by a corresponding plus pattern or a pattern that omits the star component.

8.8 Number Restrictions

Number restrictions are useful patterns for sequence definition. For example, we are interested in expressing to describe the *minimal*, *maximal* or *exact* number of occurrences of a kind of component such as a certain kind of chords. Further, we distinguish restrictions about the number of *successive components* (e.g. "minimally three successive major chords") or about the number *distributed components.* (e.g. "minimally three major chords distributed in the whole sequence").

Number restrictions for subsequent (C^n) or distributed components ($C^n_{\dots}$) are modelled by writing them out as n components connected by *seq:*`hasNext` and *seq:*`followedBy` relationships respectively.

8.8.1 Number Restriction of Successive γ-Components

To restrict the minimal number of successive components of a certain type it is sufficient to state that the component type occurs n-times successively and not impose any further restrictions about what follows

Definition 24 (Successive γ Minimal Number Restriction).

$$C^{\geq 1} :=_{\mathcal{KB}} C$$
$$C^{\geq n} :=_{\mathcal{KB}} C \triangleright C^{\geq n-1} \; \textit{for } n \in \mathbb{N}, n \geq 2$$

We can specialise this definition to express a restriction of the *exact* number of successive components by explicitly disallowing a component of the same type after a certain number of successive components.

Definition 25 (Successive γ Exact Number Restriction).

$$C^{=1} := C \ntriangleright C$$
$$C^{=n} := C \triangleright C^{=n-1} \; \textit{for } n \in \mathbb{N}, n \geq 2$$

Finally we can describe a maximal number restriction by noting that it describes the possibility of a number of different exact-length sequences. We combine the possibilities using disjunction.

Definition 26 (Successive γ Maximal Number Restriction).

$$C^{\leq 1} :=_{\mathcal{KB}} C^{=1}$$
$$C^{\leq n} :=_{\mathcal{KB}} C^{=n} \sqcup C^{\leq n-1} \; \textit{for } n \in \mathbb{N}, n \geq 2$$

8.8.2 Number Restriction of Distributed γ-Components

Restrictions of the minimal number of following γ-components of a certain type can be defined analogously to the definition of successive γ number restrictions by replacing *seq:*`hasNext` with *seq:*`followedBy`.

Definition 27 (Following γ Minimal Number Restriction).

$$\begin{aligned} C^{\geq 1} &:=_{\mathcal{KB}} C \\ C^{\geq n} &:=_{\mathcal{KB}} C \cdots C^{\geq n-1} \; \mathit{for}\ n \in \mathbb{N}, n \geq 2 \end{aligned}$$

Definition 28 (Following γ Exact Number Restriction).

$$\begin{aligned} C^{=1} &:=_{\mathcal{KB}} C \cdot\!/\!\cdot C \\ C^{=n} &:=_{\mathcal{KB}} C \cdots C^{=n-1} \; \mathit{for}\ n \in \mathbb{N}, n \geq 2 \end{aligned}$$

Definition 29 (Following γ Maximal Number Restriction).

$$\begin{aligned} C^{\leq 1} &= C^{=1} \\ C^{\leq n} &= C^{=n} \sqcup C^{\leq n-1} \; \mathit{for}\ n \in \mathbb{N}, n \geq 2 \end{aligned}$$

8.8.3 Number Restrictions in α-pattern

"Global" number restrictions, i.e. restrictions of the number of occurrences of a component type throughout the whole sequence, can be expressed as restricted α-pattern.

Definition 30 (Exact Distributed Cardinality in α).

$$\begin{aligned} \alpha \triangleright C^{=n} &:=_{\mathcal{KB}} \alpha \cdots C^{n}_{\dots} \sqcap \neg \cdots C^{n}_{\dots} \; \mathit{for}\ n \in \mathbb{N}, n \geq 1 \\ \alpha \triangleright C^{=0} &:=_{\mathcal{KB}} \alpha \triangleright \neg \cdots C \end{aligned}$$

Definition 31 (Maximal Distributed Cardinality in α).

$$\begin{aligned} \alpha \triangleright C^{\leq n} &:=_{\mathcal{KB}} C^{=n} \sqcup \alpha \triangleright C^{\leq n-1} \; \mathit{for}\ n \in \mathbb{N}, n \geq 1 \\ \alpha \triangleright C^{\leq 0} &:=_{\mathcal{KB}} \alpha \triangleright C^{=0} \end{aligned}$$

8.9 Sequence Definition

Having focused on elements to specify sequential structure up to now, we can now attempt to formalise the concept of sequence. A sequence should be specified in a way that enables:

Referring to a sequence. It should be possible to refer to a sequence using a unique reference or name.

Referring to the content of the sequence. Given the sequence name it should be possible to retrieve its elements.

Guaranteeing integrity of sequential structure. The definition should capture well-formedness constraints of sequential structure and thus enable verification.

In order to reference to a sequence as a whole, it sufficient to provide a concept *seq:*`Sequence` that is independent of other domain concepts, especially components and contents. With respect to sequential structure we introduced concepts for beginning (α) and end (ω) of a sequence. These concepts can be related in an expression $\alpha\cdots\omega$ that defines boundedness of a sequence. These ideas can be merged into a single definition:

$$\textit{seq:}\texttt{Sequence} \equiv \alpha\cdots\omega$$

In this case the individual of type α would be the identifier of a sequence as a whole iff it is additionally followed by an end. This definition still allows *underspecified* ABox instances where *seq:*`followedBy` relationships are used instead of *seq:*`hasNext` as, for example, in the ABox, such as $\alpha \blacktriangleright [a]\cdots[b] \blacktriangleright \omega$. Further, the type of the components is not specified, e.g. there can be sequences with components without content.

Additional restrictions can be imposed to define stronger notions of valid sequential structure. For example, the following definition enforces the existence of direct successors and of contents:

$$\textit{seq:}\texttt{Sequence} \equiv \alpha \triangleright \gamma^* \triangleright \omega \quad \text{with } \mathcal{KB} \supseteq \{\gamma \equiv [\top]\}$$

8.10 Pattern Subsumption

Sequence pattern can be organised according to their degree of generality. Conklin (1994) uses the notion of "contains"-semantics of pattern subsumption for *multiple viewpoint* patterns, where a pattern P is subsumed by a pattern Q iff all sequences described by P are also described by Q. As an example, consider the subsumptions of String patterns defined by a substring matcher for 'ab' and a wildcard '⋆' as illustrated in the following subsumption lattice:

$$\begin{tikzcd}
 & \star \arrow[d, dash] & \\
 & \star ab \star \arrow[dl, dash] \arrow[dr, dash] & \\
\star ab \arrow[dr, dash] & & ab\star \arrow[dl, dash] \\
 & ab &
\end{tikzcd}$$

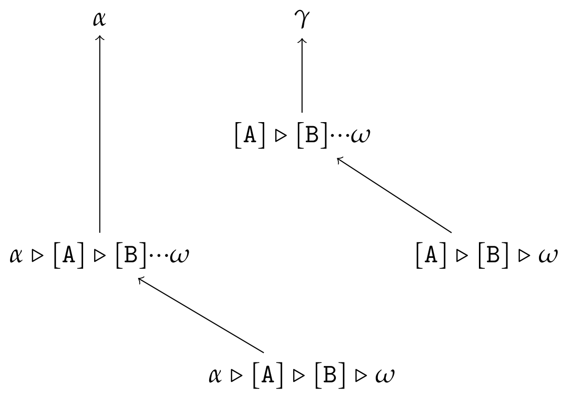
As we have defined sequential patterns as DL concepts, we are also able classify them w.r.t. general DL concept subsumption ($C \sqsubseteq D$). Concept subsumption captures above notion of "contains" subsumption to some extent. Considering the examples

$$\begin{array}{lllll}
\mathcal{KB} \models & \mathtt{[A]} \triangleright \mathtt{[B]} \triangleright \omega & \sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]} \\
\mathcal{KB} \models & \mathtt{[A]} \triangleright \mathtt{[B]} & \not\sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]} \triangleright \omega \\
\mathcal{KB} \models & \mathtt{[A]} \triangleright \mathtt{[B]} \triangleright \mathtt{[X]} & \sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]} \\
\mathcal{KB} \models & \mathtt{[A]} \triangleright \mathtt{[X]} & \not\sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]}
\end{array}$$

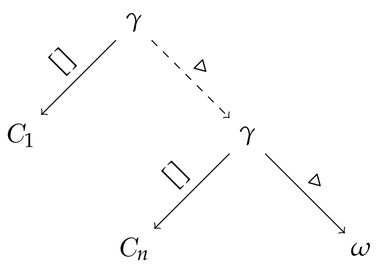
we find that we can specialise a pattern by appending it while altering some of the pattern components breaks the subsumption. Note that all of these examples start with the same type of component, in this case [A]. Unfortunately, when we use patterns that start with different component types we find a different subsumption behaviour:

$$\begin{array}{lrll}
\mathcal{KB} \models & \alpha \triangleright \mathtt{[A]} \triangleright \mathtt{[B]} & \not\sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]} \\
\mathcal{KB} \models & \mathtt{[X]} \triangleright \mathtt{[A]} \triangleright \mathtt{[B]} & \not\sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]} \\
\mathcal{KB} \models & \ldots \mathtt{[A]} \triangleright \mathtt{[B]} & \not\sqsubseteq & \mathtt{[A]} \triangleright \mathtt{[B]}
\end{array}$$

From the point of view of sequence subsumption we would have expected a symmetric subsumption behaviour when we append to the left or to the right. Similar problems occur when we express our initial example with $\mathcal{SEQ}$-expressions as follows:


$$\begin{tikzcd}
\alpha & & \gamma & \\
& & \mathtt{[A]} \triangleright \mathtt{[B]} \cdots \omega \arrow[u] & \\
\alpha \triangleright \mathtt{[A]} \triangleright \mathtt{[B]} \cdots \omega \arrow[uu] & & & \mathtt{[A]} \triangleright \mathtt{[B]} \triangleright \omega \arrow[ul] \\
& \alpha \triangleright \mathtt{[A]} \triangleright \mathtt{[B]} \triangleright \omega \arrow[ul] & &
\end{tikzcd}$$

The root of the problem becomes apparent if we recall that from the point of view of DL subsumption reasoning we compare *tree*-shaped patterns. For example, a pattern $[C_1] \triangleright [C_2] \triangleright \ldots \triangleright [C_n] \triangleright \omega$ corresponds to a tree pattern


$$\begin{tikzcd}
& \gamma \arrow[dl, "{[\,]}"'] \arrow[dr, dashed, "\triangleleft"] & & \\
C_1 & & \gamma \arrow[dl, "{[\,]}"'] \arrow[dr, "\triangleleft"] & \\
& C_n & & \omega
\end{tikzcd}$$

Observe, that appending to the right of a sequence corresponds to specialising the root concept $[C_1]$ which is reflected in the DL subsumption just in the way we want for sequence subsumption, but that we change the root when we append to the left of the sequence, thus leading to the unsatisfiable subsumptions in the examples.

One strategy to establish a symmetric subsumption behaviour when appending left or right is to *"fix the root"*. One idea is based on the observation that in the case of α-patterns there are (by definition of α) no preceding components, and that α therefore is a candidate for a common pattern-root:

$$\begin{tikzcd}
\alpha \arrow[dr, "\vartriangleleft"] & & & \\
& \gamma \arrow[dl, "{[]}"'] \arrow[dr, dashed, "\vartriangleleft"] & & \\
C_1 & & \gamma \arrow[dl, "{[]}"'] \arrow[dr, "\vartriangleleft"] & \\
& C_n & & \omega
\end{tikzcd}$$

Naturally, we do not want to restrict our selves to compare the component directly after the α and thus use *seq:*`hasNext` and *seq:*`followedBy` as appropriate. For every γ-pattern Γ we can define an α-pattern $\alpha\cdots\Gamma$ that states that the Γ pattern is occurring at some place in the sequence. By translating all γ-patterns in our example into α-patterns we can thus capture the intended subsumption behaviour

$$\begin{array}{llll}
\mathcal{KB}_{\mathcal{SEQ}} \models & \alpha\cdots & [\texttt{A}] \vartriangleright [\texttt{B}] \vartriangleright [\texttt{X}] & \sqsubseteq \ \alpha\cdots[\texttt{A}] \vartriangleright [\texttt{B}] \\
\mathcal{KB}_{\mathcal{SEQ}} \models & \alpha\cdots[\texttt{X}] \vartriangleright & [\texttt{A}] \vartriangleright [\texttt{B}] & \sqsubseteq \ \alpha\cdots[\texttt{A}] \vartriangleright [\texttt{B}]
\end{array}$$

Depicted as subsumption lattice:

$$\begin{tikzcd}
& \alpha \arrow[d, dash] & \\
& \alpha\cdots[\texttt{A}] \vartriangleright [\texttt{B}]\cdots\omega \arrow[dl, dash] \arrow[dr, dash] & \\
\alpha \vartriangleright [\texttt{A}] \vartriangleright [\texttt{B}]\cdots\omega \arrow[dr, dash] & & \alpha\cdots[\texttt{A}] \vartriangleright [\texttt{B}] \vartriangleright \omega \arrow[dl, dash] \\
& \alpha \vartriangleright [\texttt{A}] \vartriangleright [\texttt{B}] \vartriangleright \omega &
\end{tikzcd}$$

We have thus demonstrated a way to express "contains"-semantics based on α-patterns (eventually combined with a *seq:*`followedBy` restriction).

Definition 32 (α-subsumption). *We call a concept subsumption an* α-subsumption, *denoted $C \sqsubseteq_\alpha D$ iff $C \sqsubseteq D$ and $C, D \sqsubseteq \alpha$.*

Definition 33 (γ-subsumption). *We call a concept subsumption a* γ-subsumption, *denoted $C \sqsubseteq_\gamma D$ iff $C \sqsubseteq D$ and $C, D \sqsubseteq \gamma$.*

Note that in α-patterns the reference to the matched content is less obvious than in γ-patterns. For example, a pattern [A] $\rhd$ [B] would directly refer to an individual [A], while a pattern $\alpha \cdots$[A] $\rhd$ [B] would not. Sometimes this is not satisfying as we want to maintain the relationship between a pattern and the place it occurs *inside* the sequence. Thus, a pattern definition that match the γ-individual and does not change subsumption behaviour when left-appending is required. We can achieve this behaviour by constructing a concept using *inverse roles*. For example, consider the pattern

$$\alpha \rhd [\mathtt{C_1}] \rhd \ldots \rhd [\mathtt{C_i}] \rhd \ldots \rhd [\mathtt{C_n}] \rhd \omega$$

that has an α-component as root and describes all elements as successors. We can describe a pattern that would match the same instances but has the γ-component with content C_i by defining all precursors using inverse roles

$$\begin{aligned} [\mathtt{C_i}] \rhd \ \ \ldots \rhd \ \ & [\mathtt{C_n}] \rhd \omega \\ \sqcap \rhd^- \ldots \rhd^- \ & [\mathtt{C_1}] \rhd^- \alpha \end{aligned}$$

which we can depict as follows:

$$\begin{tikzcd}
 & & & \gamma \arrow[dl, "\rhd^-"'] \arrow[d, "{[]}"] \arrow[dr, "\rhd"] & & & \\
 & & \gamma \arrow[dl, dashed, "\rhd^-"'] & C_i & \gamma \arrow[dr, dashed, "\rhd"] & & \\
 & \gamma \arrow[dl, "\rhd^-"'] \arrow[dr, "{[]}"] & & & & \gamma \arrow[dl, "{[]}"'] \arrow[dr, "\rhd"] & \\
\alpha & & C_1 & & C_n & & \omega
\end{tikzcd}$$

For convenience we introduce additional syntax:

Definition 34 (Inverse construction). *Let $C_0, \ldots, C_n$ be components. Then an* inverse construction *is defined as*

$$\underleftarrow{C_n \odot_n \ldots C_1 \odot_1 C_0} \coloneqq_{\mathcal{KB}} C_0 \odot_1^- C_1 \odot_2^- \ldots \odot_n^- C_n$$

where $\odot_i \in \{\rhd, \cdots\}$.

Using γ-patterns with inverse constructions we can now express our initial example such that a symmetric subsumption for γ-patterns can be found:

$$\begin{tikzcd}
 & \gamma \arrow[d, no head] & \\
 & \underleftarrow{\alpha}\cdots\texttt{[A]} \rhd \texttt{[B]}\cdots\omega \arrow[dl, no head] \arrow[dr, no head] & \\
\underleftarrow{\alpha} \rhd \texttt{[A]} \rhd \texttt{[B]}\cdots\omega \arrow[dr, no head] & & \underleftarrow{\alpha}\cdots\texttt{[A]} \rhd \texttt{[B]} \rhd \omega \arrow[dl, no head] \\
 & \underleftarrow{\alpha} \rhd \texttt{[A]} \rhd \texttt{[B]} \rhd \omega &
\end{tikzcd}$$

Example 8.3 (Classification using Inverse Construction)
Inverse constructions can be helpful to define the position in a pattern. The patterns $[C] \rhd [D] \rhd [E]$, $\underleftarrow{[C] \rhd}\ [D] \rhd [E]$ *and* $\underleftarrow{[C] \rhd [D] \rhd}\ [E]$ *all impose the same restrictions with regard to type and succession of the contents of a sequence but will match different individuals within a sequence:*

$$\begin{tikzcd}
{[y]} & \\
{[e]} \arrow[u] & \in \underleftarrow{[C] \rhd [D] \rhd} [E] \\
{[d]} \arrow[u] & \in \underleftarrow{[C] \rhd} [D] \rhd [E] \\
{[c]} \arrow[u] & \in [C] \rhd [D] \rhd [E] \\
{[x]} \arrow[u] &
\end{tikzcd}$$

Example 8.4 (Subsumption of progression patterns)
Musical progression often develop towards a stable state such as a tonic chord. In this situation the use inverse construction to capture musically interesting subsumptions. For example, given the circle progressions

$$\texttt{P1} \equiv \underleftarrow{\texttt{[VIm]} \rhd \texttt{[IIm9]} \rhd \texttt{[V9}^{\sharp}\texttt{13}^{\flat}\texttt{]}} \rhd \texttt{[I]}$$

$$\texttt{P2} \equiv \underleftarrow{\texttt{[IIm]} \rhd \texttt{[V]}} \rhd \texttt{[I]}$$

and some background knowledge about chord relationships it con be inferred that P2 $\sqsubseteq$ P1*. The subsumption would not be inferred without inverse constructs here as the patterns would be compared left to right beginning with to the first component.*

8.11 Concatenation of Sequential Patterns

Motivation. Concatenation of sequential patterns is the combination of two arbitrary predefined patterns to a new composed pattern. The previously defined sequential constructs ($\triangleright$, $\cdots$) behave as a head-tail concatenation that does not exhibit the desired semantics if the first pattern is of length greater than 1.

To solve this we introduce a new concatenation construct $C \overset{\triangleright}{+\!\!+} D$ together with an algorithm that (1) introspects the expression C, (2) identifies the subconcepts in C that represent the terminal γ-components and (3) appends the expression D to them.

Remark 8.1 (Blending effect) *The succession constructs* $\triangleright$ *and* $\cdots$ *do not capture concatenation. For example, given two defined patterns*

$$P \equiv P_1 \triangleright \ldots \triangleright P_m$$
$$Q \equiv Q_1 \triangleright \ldots \triangleright Q_n$$

we would like to define a new pattern by concatenating P and Q. Using succession $P \triangleright Q$ it holds that

$$P \triangleright Q \equiv P \triangleright Q_1 \triangleright \ldots \triangleright Q_n$$

A correct concatenation behaviour would now be achieved if $P \triangleright Q$ would imply a restriction of P_m to $\exists seq{:}\texttt{hasNext}.Q$. However,

$$P \triangleright Q \not\equiv P_1 \triangleright \ldots \triangleright P_m \triangleright Q_1 \triangleright \ldots \triangleright Q_n$$

as the pattern defines a restriction relative to the root concept of P, i.e. on P_1.

$$P \triangleright Q \equiv P_1 \triangleright (P_2 \sqcap Q_1) \triangleright \ldots \triangleright (P_m \sqcap Q_{n-1}) \triangleright Q_n$$

In order to express a concatenation, we need to restrict the $\triangleright$-role of the last *component of P to the range Q. Therefore we define the semantics the concatenation construct $C \overset{\triangleright}{+\!\!+} D$ using a recursive algorithm that identifies the last components.*

Outline of the Concatenation Process. A concept that represents a concatenation $C \overset{\triangleright}{+\!\!+} D$ is algorithmically constructed. The algorithm

The proposed concatenation process for a construct $C \overset{\triangleright}{+\!\!+} D$ is that the it is a specialised concept $\tau_D(C)$ derived

1. the concept C is traversed along $\triangleright/\cdots$paths.
2. The traversal simply copies the concept until it reaches the last γ-components on the $\triangleright/\cdots$paths.
3. These last γ-components $\Gamma_1, \ldots, \Gamma_n$ are replaced by a specialised by append of C to $\Gamma_i \triangleright D$.

During the process bookkeeping of traversed concepts has to be done in order to define blocking conditions in case of *cyclic* concepts such as C^+.

A normal form for path traversal. In order to identify the last components the algorithm traverses the sequence pattern C along the sequential role restrictions ($\triangleright/\cdots$). There can be more than one possible path and many last components. For example, the concept $C \equiv C_1 \triangleright (C_2 \sqcup C_2')$ defines a choice of possible successors using disjunction. Hence, the algorithm needs to consider different paths. Further, care has to be taken regarding conjunctions of sequence role restrictions. Consider the pattern $C \equiv (C_1 \triangleright C_2 \triangleright C_3) \sqcap (C_1' \triangleright C_2')$. In this example we can see that C_3 is a last component while C_2' is not a last component as it is conjunct to C_2 and thus followed by C_3. If we restate the concept as $C \equiv (C_1 \sqcap C_1') \triangleright (C_2 \sqcap C_2') \triangleright C_3$ the interrelationship between sequence constructs and intersection becomes more structurally accessible and we can directly follow the *seq:*`hasNext`.

We therefore introduce a *normal form* which allows to traverse additive pathes as one path while branching at disjunctions.

Definition 35 (Disjunctive Normal Form). *A statement is in* disjunctive normal form (DNF) *if it is a disjunction consisting of one or more disjuncts, each of which is a conjunction of possibly negated concept names or role restrictions, i.e. has the form*

$$\textsc{dnf}(C) = \bigsqcup_i \bigsqcap_j (\neg) C_{ij}$$

A concept C can be transformed into $\textsc{dnf}(C)$ *by applying the following procedure:*

1. *Distribute negation over any conjunctions or disjunctions according to deMorgan's laws*

$$\neg(C \sqcap D) \rightsquigarrow \neg C \sqcup \neg D$$
$$\neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D$$

   until negation is only applied to atomic concepts. Remove double negations where these arise.

2. *Let* $\textsc{disjuncts}(C) := \{C_i \mid C_1 \sqcup C_2 \sqcup \ldots \sqcup C_m\}$ *denote the set of disjuncts within a concept expression C. The disjunctive normal form is computed by exhaustively applying the following rules:*

$$\textsc{dnf}(C \sqcup D) \rightsquigarrow \textsc{dnf}(C) \sqcup \textsc{dnf}(D)$$
$$\textsc{dnf}(C \sqcap D) \rightsquigarrow \bigsqcup_{\substack{C_i \in \textsc{disjuncts}(\textsc{dnf}(C)) \\ D_j \in \textsc{disjuncts}(\textsc{dnf}(D))}} C_i \sqcap D_j$$

3. *Remove duplicates of atomic concepts and negated atomic concepts in an "inner" conjunction. Finally if the exact same "inner" conjunction appears more than once in the "outer" disjunction, that duplication can be eliminated by removing one copy.*
4. *Finally, simplify the $\mathcal{SEQ}$ property restrictions by exhaustively applying the following rules:*

$$\exists seq{:}\texttt{hasNext}.C \sqcap \exists seq{:}\texttt{hasNext}.D \rightsquigarrow \exists seq{:}\texttt{hasNext}.C \sqcap D$$
$$\exists seq{:}\texttt{followedBy}.C \sqcap \exists seq{:}\texttt{followedBy}.D \rightsquigarrow \exists seq{:}\texttt{followedBy}.C \sqcap D$$

Definition 36 (Sequence Concatenation). *The concatenation of two concepts C and D, denoted $C \overset{\triangleright}{+\!\!+} D$, is defined as*

$$C \overset{\triangleright}{+\!\!+} D := \tau_\sigma(\textsc{dnf}(C))$$

where σ is a specialisation function $\sigma := \Gamma \mapsto \Gamma \triangleright D$ and the transformation τ_σ is recursively defined as follows:

$$\tau_\sigma(C_1 \sqcup \ldots \sqcup C_n) = \tau_\sigma(C_1) \sqcup \ldots \sqcup \tau_\sigma(C_n)$$

$$\tau_\sigma(C_1 \sqcap \ldots \sqcap C_n) = \begin{cases} \textit{if } |\mathcal{C}^{\cdots}| > 0 & \displaystyle\bigsqcap_{C_i \in \mathcal{C}^{\cdots}} \tau_\sigma(C_i) \sqcap \bigsqcap_{C_j \in \mathcal{C} \setminus \mathcal{C}^{\cdots}} C_j \\ \textit{else if } |\mathcal{C}^{\triangleright}| > 0 & \displaystyle\bigsqcap_{C_i \in \mathcal{C}^{\triangleright}} \tau_\sigma(C_i) \sqcap \bigsqcap_{C_j \in \mathcal{C} \setminus \mathcal{C}^{\triangleright}} C_j \\ \textit{otherwise} & \sigma(C_1 \sqcap \ldots \sqcap C_n) \end{cases}$$

$$\begin{aligned} \textit{where } \mathcal{C} &:= \{C_1, \ldots, C_n\}, \\ \mathcal{C}^{\cdots} &:= \{C_i \mid C_i \in \mathcal{C} \textit{ and } C_i \textit{ is a } \triangleright\textit{-expression}\}, \\ \mathcal{C}^{\triangleright} &:= \{C_i \mid C_i \in \mathcal{C} \textit{ and } C_i \textit{ is a } \cdots\textit{-expression}\} \end{aligned}$$

$$\tau_\sigma(\mathtt{q}R.C) = \tau_\sigma(\textsc{dnf}(C))$$

A concatenation $C \overset{\cdots}{+\!\!+} D$ that appends a component via a seq:`followedBy` *restriction can be computed analogously using a specialisation function $\sigma = \Gamma \mapsto \Gamma \cdots D$.*

Expansion behaviour. The concatenation operator identify the leaves of the concept tree of its first operand and specialises with a succession restriction to the second operand. Therefore the second operand is duplicated as often as there are leaves.

8.12 Summary

This chapter described the $\mathcal{SEQ}$ representation for chord sequence patterns in OWL-DL based on the OWLList representation by Drummond et al. (2006b). Notation and expressivity are similar to regular expressions, and allow the expression of different levels of abstraction with respect to subsumption of chords. Several reasoning services on such a representation can be implemented using OWL reasoners. In a web retrieval scenario, for example, instance checking can be used to find chord sequences that match or contain a search pattern (cf. chapter 16). Subsumption checking can be used to compare pattern inclusion (cf. section 14.3).

Summarising, we have defined the following constructs:

Table 8.1: Summary of $\mathcal{SEQ}$ constructs

Notation	Meaning
$C \triangleright D$	C then directly D (head-tail composition)
$C \cdots D$	C followed by D (head-tail composition)
$C \overset{\triangleright}{+\!\!+} D$	C then directly D (concatenation)
$C \overset{\cdots}{+\!\!+} D$	C followed by D (concatenation)
C^{+}	one or more C
C^{*}	none or more C
$C^{\leq n}$	maximally n C
$C^{\geq n}$	minimally n C
$C^{=n}$	exactly n C
α	initial component
γ	component with content
ω	terminal component

Care has to be taken regarding the intended *labelling* and *subsumption* behaviour of sequence patterns. In order to avoid non-symmetric subsumption behaviour, the notions of *α-patterns* and *γ-patterns* have been introduced, which by restricting general DL subsumption to

$$C \sqsubseteq_{\alpha} D \text{ and } C \sqsubseteq_{\gamma} D$$

guarantee well-behaved subsumption.

9 Examples of combining $\mathcal{SEQ}$ and $\mathcal{MEO}$

With the possibilities to represent musical entities and sequential structures we can now model chord sequence patterns.

9.1 Musical Forms

We can now define musical forms as pattern. For example a 12-bar blues

C7			
F7		C7	
G7	F7	C7	C7

could be represented

$$\begin{array}{lllllll} \texttt{TwelveBarBlues} & \equiv & \alpha & \triangleright [\texttt{C7}]^4 & & & \\ & & & \triangleright [\texttt{F7}]^2 & & \triangleright [\texttt{C7}]^2 & \\ & & & \triangleright [\texttt{G7}] & \triangleright [\texttt{F7}] & \triangleright [\texttt{C7}] & \triangleright [\texttt{C7}] \triangleright \omega \end{array}$$

While the blues example is rigid regarding the number of successive components, we can also define musical patterns in a more flexible way. For example, Kostka and Payne (2003) describe a simple and interesting way to model diatonic progressions using a diagrammatic representation resembling a state space. Figure 9.1 shows this representation. A valid progression can be generated by starting at any given point in the diagram and following the arrows until reaching a desired state.

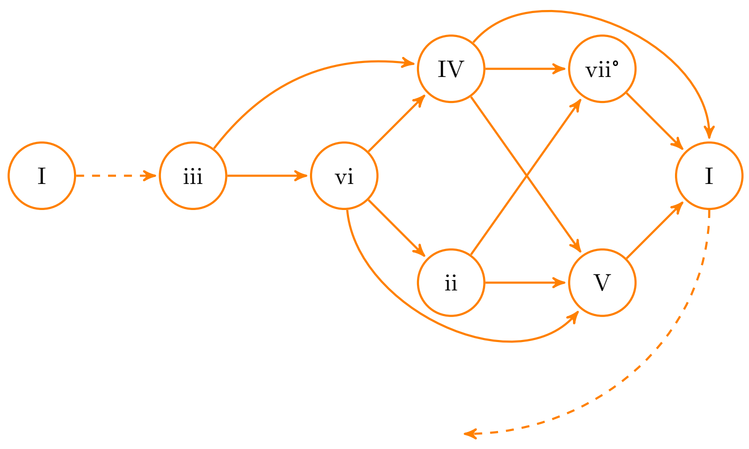


Figure 9.1: Chart of tonal progressions (drawn from Kostka and Payne, 2003)

We can restate the transitions in the diagram as a $\mathcal{SEQ}$ concept

$$\texttt{KP\_Progression} \equiv \alpha \rhd \texttt{Transition}^{+} \rhd \omega$$

where the *Transition* concept is defined by interpreting the node labels as class names and the edges as $\rhd$-restrictions:

$$\begin{aligned}\texttt{Transition} \equiv\; & [\texttt{I}] \rhd [\texttt{iii}] \\ \sqcup\; & [\texttt{ii}] \rhd [\texttt{V} \sqcup \texttt{vii}^{\circ}] \\ \sqcup\; & [\texttt{iii}] \rhd [\texttt{vi} \sqcup \texttt{IV}] \\ \sqcup\; & [\texttt{IV}] \rhd [\texttt{I} \sqcup \texttt{V} \sqcup \texttt{vii}^{\circ}] \\ \sqcup\; & [\texttt{V}] \rhd [\texttt{I}] \\ \sqcup\; & [\texttt{vi}] \rhd [\texttt{IV} \sqcup \texttt{ii} \sqcup \texttt{V}] \\ \sqcup\; & [\texttt{vii}^{\circ}] \rhd [\texttt{I}]\end{aligned}$$

For both examples DL reasoning can then be used to identify chord sequences that satisfy the harmonic pattern or to validate if a sequence agrees with a musical form.

9.2 Conceptual Graphs

Conceptual graphs are notation for logic introduced by John F. Sowa, inspired by the existential graphs of Charles Sanders Peirce and the semantic networks. They were first used to represent the conceptual schemas in database systems and later applied to a wide range of topics in artificial intelligence, computer science, and cognitive science (Sowa, 1984). Conceptual graphs have been standardised as interchange format within the Common Logic ISO-standard (ISO/IEC 24707).

Figure 9.2 and Figure 9.3 are examples that represent musical score and succession rules using the graphical formalism.

Conceptual graphs can be translated to closed first order formulae. As full first order logic can be expressed, it follows that subsumption checking is undecidable in the general case. On the other hand work has been done on identifying decidable subsets and especially such subsets that coincide with Description Logics. Baader et al. (1998) concludes that while shared fragments of the underlying first order semantics can be found, no “natural” fragments that allow direct translation of the graphical notation to Description Logic syntax are obvious. However, the given examples can be expressed using $\mathcal{MEO}/\mathcal{SEQ}$ as follows.

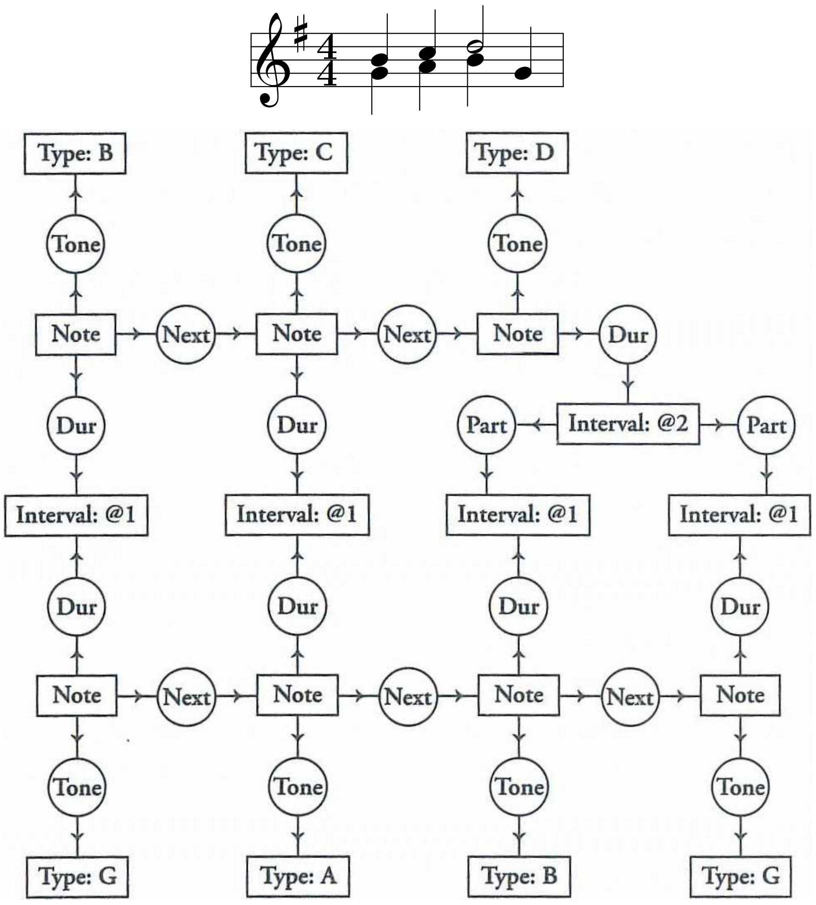


Figure 9.2: Conceptual Graph for two voices singing in harmony (Sowa, 1984, p. 37)

The CG in Figure 9.2 corresponds to the FOL formular:

$$
\begin{aligned}
&(\exists x_1, x_2, x_3, x_4, x_5, x_6, x_7 : Note)\\
&\ (\exists y_1, y_2, y_3, y_4, y_5 : Interval)\\
&\ (tone(x_1, B) \wedge dur(x_1, y_1) \wedge hasAmount(y_1, 1beat) \wedge next(x_1, x_2) \wedge\\
&\ \ tone(x_2, C) \wedge dur(x_2, y_2) \wedge hasAmount(y_2, 1beat) \wedge next(x_2, x_3) \wedge\\
&\ \ tone(x_3, D) \wedge dur(x_3, y_3) \wedge hasAmount(y_3, 2beat) \wedge\\
&\ \ tone(x_4, G) \wedge dur(x_4, y_1) \wedge next(x_4, x_5) \wedge\\
&\ \ tone(x_5, A) \wedge dur(x_5, y_2) \wedge next(x_5, x_6) \wedge\\
&\ \ tone(x_6, B) \wedge dur(x_6, y_4) \wedge hasAmount(y_4, 1beat) \wedge next(x_6, x_7) \wedge\\
&\ \ tone(x_7, G) \wedge dur(x_7, y_5) \wedge hasAmount(y_5, 1beat) \wedge\\
&\ \ part(y_3, y_4) \wedge part(y_3, y_5))
\end{aligned}
$$

As only binary relationships are used we can translate this as a set of ABox assertions where predicates (tone, dur, ...) are understood as functional roles and constants (B, 1beat, ...) as individuals. Existentially quantified variables can directly be expressed in the ABox as blank nodes.

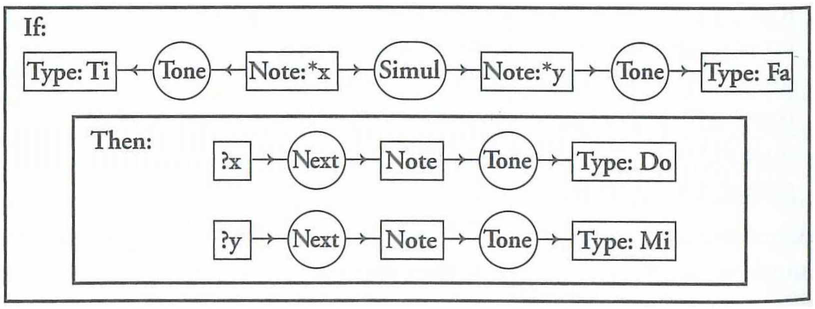


Figure 9.3: Rule for the harmonious resolution of a dissonance (Sowa, 1984, p. 38)

The CG in Figure 9.3 describes a rule for the harmonious resolution of a dissonance. Sowa (1984, p. 38) translates this to the following FOL formula:

$$
\begin{aligned}
&(\forall x, y : Note)((simul(x, y) \wedge tone(x, Ti) \wedge tone(y, Fa) \implies\\
&\ (\exists z, w : Note)(next(x, z) \wedge tone(z, Do) \wedge next(y, w) \wedge tone(w, Mi)))
\end{aligned}
$$

This formula can be understood as a subsumption relationship on a simultaneity of two notes. DL role subsumption of the form $R \sqsubseteq S$ is not sufficient as we additionally need to capture the conditions on x and y in the antecedent as well as propagate the variable structure in the consequent. The use of role chains to capture the information in the antecedent and domain/range axioms to capture the information in the consequent can fill this gap. We introduce an additional modelling pattern for this purpose:

Definition 37 (Concept Product Subsumption). *A concept product subsumption along a role R expresses that any two instances $c \in C$, $d \in D$ which are related by a role R are also of types C' and D' resp. We denote this*

$$C \underset{R}{\times} D \sqsubseteq C' \underset{R}{\times} D'$$

which is axiomatised as

$$\begin{aligned} id_C \circ R \circ id_D &\sqsubseteq S \\ domain\ S &= C' \\ range\ S &= D' \end{aligned}$$

where S is a fresh role name.

Finally, we can capture the resolution rule as follows:

$$[\exists \texttt{tone.Ti}] \underset{\texttt{sim}}{\times} [\exists \texttt{tone.Fa}] \sqsubseteq (\triangleright [\exists \texttt{tone.Do}]) \underset{\texttt{sim}}{\times} (\triangleright [\exists \texttt{tone.Mi}])$$

9.3 Summary

$\mathcal{MEO}$ and $\mathcal{SEQ}$ can be combined in various ways to express chord sequence patterns. This chapter discussed three examples. First, a blues form was expressed using a regular expression like approach. Second, we showed how to model diatonic progressions by expressing state transitions in a pattern. A third example showed how a small score excerpt can be represented together with a rule that described harmonious resolution of a dissonance.

10 Grammars as $\mathcal{SEQ}$ Patterns

Grammars have long been a part of the toolbox of computational musicologists. Already in the 19th century music theorists like Riemann (1877) started to develop syntactic descriptions of music. In the second half of the 20th century, the paradigm shift by generative grammars in linguistics (see e.g. Chomsky, 1956) sparked a strong interest in systematic musicology too (see e.g. Roads, 1987). They have been used as a means for generating musical structures — such as chord sequences — using substitution rules as well as for creating musical analyses by parse trees. Therefore it is desirable to express grammars in $\mathcal{SEQ}$, in order to make use of this work in a Semantic Web context. The limited expressiveness of Description Logics make it difficult to formulate grammars directly in OWL or $\mathcal{SEQ}$. In fact, for context-free grammars a full translation is not possible. This is evident from the fact that it is not generally decidable if two CFGs generate the same language whereas DLs are restricted in expressivity such that concept equality always remains decidable. After the following brief introduction of formal grammars we will discuss possible approaches of using grammars in the Semantic Web with $\mathcal{SEQ}$ with restrictions and partial transformations of the grammar and the parsing process. An algorithm for partial transformation is developed and the scope and limitations of this approach are then discussed.

10.1 Formal Grammars

This section introduces definitions and notations on formal grammar that will be used in the following sections. For more details on grammars and their properties, the reader is referred to Hopcroft et al. (2003).

A grammar is a formal device that uses production rules to generate strings thus describing the language of possible strings. Noam Chomsky (Chomsky, 1956, 1957) first proposed the notion of (generative) grammars and classified them into a hierarchy of incremental specialisations (restrictions of the allowed forms of production rules), with type 0 being the most general (unrestricted) form.

Definition 38 (Unrestricted Grammar (Type 0)).
A grammar G is a tuple (N, Σ, P, S) consisting of

– *a finite set N of* nonterminal *symbols,*

- *a finite set* Σ *of* terminal *symbols (with* $N \cap \Sigma = \varnothing$*),*
- *a finite set* P *of* production rules*, each rule of the form* $(\Sigma \cup N)^* \to (\Sigma \cup N)^*$ *where* * *is the Kleene star operator, and*
- *a distinguished start symbol* $S \in N$*.*

Each production rule maps from one string of symbols to another, where the first string (the *head*) contains at least one nonterminal symbol.

	non-terminal	terminal	either
single	$A, B \in N$	$a, b \in \Sigma$	$X, Y \in \Sigma \cup N$
sequence	$U, V \in N^*$	$u, v \in \Sigma^*$	$\alpha, \beta \in (\Sigma \cup N)^*$

Table 10.1: Notational conventions for types of grammatical symbols

Notation and Terminology. We follow the usual stylistic conventions to denote kinds of symbols and symbol sequences as shown in Table 10.1. Any production of the form $A \to \alpha$ will be said to be an *A-production*, and α will be said to be an *expansion* of A. The empty string is denoted by the symbol ϵ. Productions of the form $A \to \ldots \mid \epsilon \mid \ldots$, that generate the empty string, are called *ϵ-productions*.

Derivation and generated language. Grammars define a set of strings, i.e. the *language* of a grammar, that can be derived from the production rules.

Definition 39 (Derivation). *For a grammar G, one step derivation is defined as a binary relation on strings* $\Rightarrow \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ *such that* $\alpha \Rightarrow \alpha'$ *iff*

$$\exists \gamma_1, \gamma_2 \in (\Sigma \cup N)^*, \exists \beta \to \beta' \in P : (\alpha = \gamma_1 \beta \gamma_2) \wedge (\alpha' = \gamma_1 \beta' \gamma_2).$$

By $\overset{*}{\Rightarrow}$ *we denote the transitive reflexive closure of* $\Rightarrow$ *, i.e. the derivation in zero or more steps. A grammar is called* cyclic *if there is a derivation of the form* $A \Rightarrow \ldots \Rightarrow A$ *for any nonterminal* A*.*

Types of grammars. Unrestricted grammars can exhibit undesirable computational properties. For example, the decision problem of whether a given string belongs to the language of some unrestricted grammar is in general undecidable. Therefore several restricted forms of grammars have been defined, such as context-sensitive (type 1) grammars (CSGs).

Definition 40 (Context-sensitive Grammar (Type 1)). *A context-sensitive grammar G is a grammar where the set P is restricted to production rules of the form*

$$(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$$

Context-free (type 2) grammars (CFGs) allow the construction of efficient parsing algorithms and are the basis for most practical applications of grammars.

Definition 41 (Context-free Grammar (Type 2)). *A context-free grammar G is a grammar where the set P is restricted to production rules of the form*

$$N \rightarrow (\Sigma \cup N)^*$$

A context-free grammar G is *self-embedding* (SE), if there exists a derivation $A \overset{*}{\rightarrow} \alpha A \beta$, with both α and β non-empty. G is *non-self-embedding* (NSE) if it is not self-embedding. By a result of Chomsky (1959), any NSE grammar generates a regular language.

Definition 42 (Regular Grammar (Type 3)). *A regular grammar is a context-free grammar where the set P is restricted to production rules of one the following forms:*

$$N \rightarrow \Sigma N \cup \Sigma \qquad \textit{(right regular)}$$
$$N \rightarrow N \Sigma \cup \Sigma \qquad \textit{(left regular)}$$

Apart from the types distinguished by Chomsky, further kinds of grammars have been introduced in the literature. One distinction that will be used in the following are linear grammars that are between type 2 and 3 in the Chomsky hierarchy, i.e. specialise context-free and generalise regular grammars.

Definition 43 (Linear Grammars). *A* linear grammar *is a context-free grammar where the set P is restricted to production rules with at most one terminal in the expansion, i.e. of the form*

$$N \rightarrow \Sigma^* N \Sigma^* \cup \Sigma$$

Special cases of linear grammars are left *and* right linear *grammars that generalise regular grammars to allow sequences of terminal symbols:*

$$N \rightarrow \Sigma^* N \cup \Sigma^* \qquad \textit{(right linear)}$$
$$N \rightarrow N \Sigma^* \cup \Sigma^* \qquad \textit{(left linear)}$$

10.2 Grammars for Chord Sequences

As mentioned earlier, grammars have been applied in music analysis and generation. With respect to chord sequences, there has been the use of grammars for harmonic analysis (see Sundberg and Lindblom, 1976; Ulrich, 1977; Baroni and Callegari, 1984; Steedman, 1984; Giomi and Ligabue, 1991; Weyde, 1995; Pachet, 1997) and also for generation of new sequences in improvisatory and interactive systems (Chemilier, 2001; Chemillier, 2004).

These grammars typically describe a sequence of chords as one or more cadences, which are structured relative to one or possibly several keys. A cadence of minimal size could be a sequence of two chords like `G7 Cj`, where upper case letters represent major chords, and the `7` and `j` represent seventh and major chord types respectively. This notation is found in so-called 'lead-sheets' which are commonly used in Jazz.

Here is a small example of a grammar to parse such chord sequences:

```
Cadence -> D T
D -> V7
T -> Ij
```

These rules describe the structure of a simple cadence consisting of tonic and dominant parts, which are realised by a seventh chord on degree V and a major chord on degree I.

Depending on the key there may be rules like the following for the key of C:

```
Ij -> Cj
V7 -> G7
```

These rules allow the generation and analysis of sequences of chord symbols as in the example above. The last rules illustrate one problem of this approach: to extend this approach to other keys it is necessary to repeat the last two rules and also the previous ones to avoid the unintended mixing of keys in one cadence. For this reason, grammars in practical applications are often extended with facilities like the use of variables or logical conditions to make the notation more expressive. Typically, grammars are implemented in logic-oriented programming languages like Prolog or Lisp, and use the facilities of these languages.

When applied for analysis, it is interesting, which rules were applied where. This information is represented in a parse-tree, which can be produced as a by-product of the parsing process. For the generation of sequences, it is more interesting to control the generation process by limiting the potentially huge number of options given by the grammar. In the context of the Semantic Web, the analysis case is more prevalent, as the focus is on retrieving existing data.

Transforming a grammar into a $\mathcal{SEQ}$ pattern could enable the use of established grammars on data sets in the Semantic Web without the need for downloading and further preprocessing of data.

10.3 Desirable transformation properties

For the use of grammars in $\mathcal{SEQ}$, especially in musicological settings, the following features are desirable for our transformation of grammars to $\mathcal{SEQ}$ patterns:

- *Preservation of language.* A transformed $\mathcal{SEQ}$ pattern should describe the same set of strings as the original grammar.
- *Expressivity.* The transformation should work on a large set of grammars, ideally all Context Free Grammars.
- *Provision of parse tree information.* The most direct applications of formal grammars are the generation of sequences by expanding production rules and the checking of a given sequence for being described by the language. Parse-trees are byproducts of generation or checking that capture the application of production rules. They are interesting as the hierarchical structuring of the tree is often used to capture a meaningful structuring of the sequence where nonterminals are labels of the structural units.
- *Additional processing features.* In practical applications, grammars are often extended with processing features, such as constraint checks or arithmetic operations on variables during parsing or production to control application of rules.

When converting a grammar into a DL pattern ideally the DL concept should describe the set of possible sequences but also capture the information of a parse tree.

10.4 Transformations of Productions

In the following we define a transformation procedure from grammars into $\mathcal{SEQ}$ patterns. First we discuss how different kinds of production rules can be transformed and which properties are preserved. In section 10.5 these transformation steps are integrated into an algorithm.

Our initial intention is to directly transform context free rules of the form

$$A \rightarrow X_1 \dots X_n$$

to DL concepts. Ideally, we want to capture the sequential structure as well as the edge structure of the parse tree. We could separately formalise these two aspects as concept definitions

$$A \equiv X_1 \rhd \dots \rhd X_n \qquad \text{(succession)}$$

$$A \equiv \exists \mathtt{expansion_1.X_1} \sqcap \dots \sqcap \exists \mathtt{expansion_n.X_n} \qquad \text{(tree shape)}$$

where the first definition captures the sequential succession and the second models the parse tree edges as `expansion` roles. However, the tree model restriction and lack of variables in OWL (cf. subsection 4.3.2) do not allow us to equate the corresponding X_i that occur along *seq:*`hasNext` and `expansion` pathes.

Therefore, we weaken our requirements and only use (succession). This definition still allows to retain some relationship between non-terminal and first component in the expansion via subsumption.

We introduce the transformation starting with simple forms of production rules without non-terminals, then discuss substitution and recursion.

Remark 10.1 *We assume that grammatical symbols directly correspond to a concept name or concept expression, i.e.* $\Sigma \cup N \subseteq \mathbf{C}$ *where* $\mathbf{C}$ *is the set of possible concept expressions. The idea is to be able to perform a stepwise transformation of parts of the grammar into concept expressions.*

10.4.1 Variable-free productions

For a sequence of terminal symbols $a_1 \dots a_n \in \Sigma^*$ we can form a concept expression $a_1 \rhd \dots \rhd a_n$ that captures the succession of the symbols. We can transform a variable-free production of the form

$$A \to a_1 \dots a_n$$

that describe a sequence without substitution into a concept of the form

$$A \equiv a_1 \rhd \dots \rhd a_n$$

We use equivalence as it encompasses the possibility to assign or check

1. the *sequential structure of a type* ($A \sqsubseteq a_1 \rhd \dots \rhd a_n$) as well as
2. the *type of a sequential structure* ($a_1 \rhd \dots \rhd a_n \sqsubseteq A$).

Note that in this definition pattern type assignment and checking are relative to the first symbol a_1 (cf. discussion in section 8.5), i.e. in contrast to a parse tree where the relationship between head symbol and every expanded symbol is retained we only retain the relationship to the first expanded symbol explicitly. Further, in section 8.11 we showed the limitations of the $\rhd$-operator if the first operand is itself a sequential pattern and introduced the $\overset{\rhd}{+\!\!+}$-operator as a remedy. Therefore we modify the transformation to use concatenation:

$$A \equiv a_1 \overset{\rhd}{+\!\!+} \dots \overset{\rhd}{+\!\!+} a_n$$

10.4.2 Non-recursive productions

Let us consider general non-recursive context-free productions, i.e. productions of the form

$$A \to X_1 \dots X_n$$

where $A \overset{*}{\not\Rightarrow} \alpha A \beta$. The approach we have taken for final productions would suggest a transformation to a concept

$$A \equiv X_1 \overset{\rhd}{+\!\!+} \dots \overset{\rhd}{+\!\!+} X_n$$

The concatenation operator is defined on sequential patterns. If we substitute a variable by a pattern we achieve the desired result, e.g. assuming X_1 to be a variable replace X_1 by $a \triangleright b$. Yet, the problems discussed in section 8.11 still arises if we transform the variable to a concept name and express the sequential structure of X_1 in a further definition, e.g. $X_1 \equiv a \triangleright b$. In this example, the concatenation algorithm will just append the names, giving $A \equiv X_1 \triangleright X_2 \ldots \triangleright X_n$ and blending effects can occur, in this case $b \sqcap X_2$. Notably, for the last operand in a concatenation no blending effects can occur such that X_n can be safely concatenated and the X_n-productions can be transformed into separate axioms. This motivates the following definition:

Definition 44 (Safe Concatenability). *An expansion is called* safely concatenable *if it is of the form* $\Sigma^*(\Sigma \cup N)$.

A safely concatenable string $a_1 \ldots a_n X$ can be directly transformed to a $\mathcal{SEQ}$ concept $a_1 \overset{\triangleright}{+\!\!+} \ldots \overset{\triangleright}{+\!\!+} a_n \overset{\triangleright}{+\!\!+} X$.

In productions that are not safely concatenable, variables at non-final position can be treated by replacing the variable by its expansions. This is partial materialisation of the grammar. Note, that in some cases productions that are not safely concatenable can be transformed into safely concatenable productions. For example, in the next section we will transform left recursive (non-safe) into right recursive (safe) grammars. We have not fully characterised which non-safe productions can be rewritten to safely concatenable expansions. This suggests a direction for optimisation that we leave for future work.

For one variable A there can be many A-productions. Separate A-productions are interpreted as *disjunctions* that describe options for substitution. Separate definitions in a Description Logic on the other hand are understood as *conjunctions* that together restrict the possible models for a concept. Therefore, our transformation has to explicitly combine the expansions in disjunctive concept expression. To achieve this, we first normalise the grammar and then add disjunctions. Every context-free grammar can be rewritten to Chomsky normal form where each production is of the form $A \to \alpha_1 \mid \ldots \mid \alpha_n$. Assuming we have transformed all α_i into concepts, we can transform the expansion $\alpha_1 \mid \ldots \mid \alpha_n$ into disjunction $\alpha_1 \sqcup \ldots \sqcup \alpha_n$.

Repetitive structures naturally arises in music, for example in cadential structure. Grammars can express repetition by means of recursion. OWL2 allows recursive concept definitions as we have seen in the definition of C^+ repetition in $\mathcal{SEQ}$ and we can use this to transform recursive productions.

In the following we will see that right recursive grammars can directly converted to $\mathcal{SEQ}$. Due to the restriction of safe concatenability, grammars with a recursive symbol in the centre of an expansion can not be transformed. For left recursive grammars we can circumvent the restrictions as they can be transformed into right recursive grammars.

We can differentiate between *left-recursive* and *right-recursive* production rules, for example,

$$A \to a\,A \mid b \text{ describes sequences } a^*b \text{ right-recursively,}$$
$$A \to A\,a \mid b \text{ describes sequences } a^*b \text{ left-recursively.}$$

The production rules in these example are *direct* recursive rules, i.e. rules where the recursive head symbol also occurs in the rule expansion. In the case of *indirect* recursion a recursive head symbol reoccurs in another expansion recursive nonterminal. For nonterminals $A_0, A_1, \ldots, A_n$, indirect left recursion can be defined as being of the form:

$$\begin{aligned} A_0 &\to A_1\alpha_1 \mid \ldots \\ A_1 &\to A_2\alpha_2 \mid \ldots \\ &\ldots \\ A_n &\to A_0\alpha_{n+1} \mid \ldots \end{aligned}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_n$ are sequences of nonterminals and terminals.

We can define recursive concept expressions in OWL. In section 8.7 we used recursive concepts to define repetition operators such as $C^+ \triangleright D$ that are transformed into a fresh concept R and an axiom $R \equiv C \triangleright (D \sqcup R)$. Here D can be an arbitrary sequential concepts. Left and center recursion are not directly expressible. Though we can place the recursive concept at other positions in a DL definition this will lead to blending of sequential definitions rather than embedding, for example such as in the definition $R \equiv C \triangleright (D \sqcup R) \triangleright E$. Essentially we cannot append to a recursive concept.

Thus this definition allows expressing right recursion. Left and centre recursion is not in general possible.

Our approach is to directly transform right recursive grammars to $\mathcal{SEQ}$ expressions. While left and centre recursive grammars can not be directly transformed they can in some cases be translated into a weakly equivalent right recursive form that suits our transformation procedure. Weakly in this context means that the set of described sequences is preserved while the variables and production rules change. This usually does not involve removal of variables but introduction of new variables.

Removing Left Recursion. The requirement of safe concatenability restricted the scope of our $\mathcal{SEQ}$ transformation to right recursive grammars. Moore (2000) describes how every left recursive CFG can be transformed into a (weakly) equivalent right-recursive CFG using Algorithm 1. By applying Moore's transformation as a preprocessing step we can extend the scope of our transformation to left recursive grammars. Algorihm 2 further allows to remove indirect left recursion in grammars that are non-cyclic and contain no ϵ-productions.

Moore (2000) describe further improvements to these algorithms that can reduce the number of newly introduced productions. For our purpose it is sufficient to note that the resulting productions have the form for which we previously described steps for conversion into DL

Algorithm 1: removeDirectLR : $2^{Rule} \rightarrow 2^{Rule}$ Removing direct left-recursions (adapted from Moore, 2000)

```
Require: A non-cyclic grammar $G$ with no $\epsilon$-productions.
Ensure: A grammar with no direct left recursions.
  for each rule of the form $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$ do
    replace the $A$-production by the production
    $A \rightarrow \beta_1 A' \mid \ldots \mid \beta_m A'$
    and create a new nonterminal
    $A' \rightarrow \epsilon \mid \alpha_1 A' \mid \ldots \mid \alpha_n A'$
  end for
```

Algorithm 2: removeIndirectLR : $2^{Rule} \rightarrow 2^{Rule}$ Removing indirect left-recursions (adapted from Moore, 2000)

```
Require: A non-cyclic grammar $G$ with no $\epsilon$-productions, with non-
  terminals $A_1, \ldots A_n$.
Ensure: A grammar with no indirect left recursions.
  for i = 1 to n do
    for j = 1 to i - 1 do
      let the current $A_j$-production be
      $A_j \rightarrow \alpha_1 \mid \ldots \mid \alpha_k$
      replace each production $A_i \rightarrow A_j\beta$ by
      $A_i \rightarrow \alpha_1\beta \mid \ldots \mid \alpha_k\beta$
    end for
    remove direct left recursion for $A_i$
  end for
```

concepts. Thus with the described methods left-recursive concepts can be rewritten into $\mathcal{SEQ}$ patterns as well.

Self-Embedding grammars Finally, we consider recursion symbols in other places than left and right corner of a production. The corresponding subtypes of context-free grammars are called self-embedding grammars. Though it would be desirable to reduce the problem of representing self-embedding grammars in $\mathcal{SEQ}$ by reducing them to regular grammars this is in general not possible. **?** discuss and present results on transformations of self-embedding to regular grammars. In general, it is undecidable if an arbitrary context-free grammar has a regular solution. Nevertheless, there are self-embedding grammars that can be transformed and **?** have characterised some transformable sublanguages. For example, languages with self-embeddedness terms of the form $X\alpha X$ and $\gamma X\gamma$ can still be reduced to regular languages.

10.5 Algorithm

Algorithm 10.5 realises the transformation of a grammar to a $\mathcal{SEQ}$ combining the transformations steps that have been developed in the previous section.

Example 10.1 illustrates how the algorithm proceeds. In a first step, the productions are rewritten to Chomsky normal form. The algorithm only excepts left or right recursive grammars as input. Left recursive grammars are transformed into right recursive grammars.

Within the main loop the algorithm proceeds to transform expan-

Algorithm 3: Rewriting of production rules to $\mathcal{SEQ}$ expressions.
$\mathsf{transform} : 2^{Rule} \rightarrow 2^{DLAxiom}$

```
Require: a set of context-free rules $P$ without self-embedding
Ensure: a set of $\mathcal{SEQ}$ patterns $P'$
  // normalise rules to Chomsky normal form $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$        (1)
  $P := \{(A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n) \mid \{A \rightarrow \alpha_i\} \subseteq 2^P\}$
  // remove left-recursion using Algorithm 2
  $P := \mathsf{removeIndirectLR}(P)$
  // transform expansions into concepts                                   (2)
  repeat
    // concatenate safely concatenable expansions                         (a)
    for each production $p \in P$ of the form $A \rightarrow a_1 \ldots a_n X$ do
      $P := (P \setminus p) \cup \{A \rightarrow a_1 \overset{\triangleright}{+\!\!+} \ldots \overset{\triangleright}{+\!\!+} a_n \overset{\triangleright}{+\!\!+} X\}$
    end for
    // form concept unions for single-terminal expansions:               (b)
    for each pair of single terminals $a$, $b$
                in rules of the form $A \rightarrow a \mid b \mid \alpha_1 \mid \ldots \mid \alpha_n$ do
      $P := (P \setminus p) \cup \{A \rightarrow a \sqcup b \mid \alpha_1 \mid \ldots \mid \alpha_n\}$
    end for
    // non-right-corner variables are substituted if                      (c)
    // there is an expansion with only one terminal symbol
    for each production of the form $B \rightarrow a$ do
      for each production of the form $A \rightarrow \alpha B \beta X | \gamma_1 \mid \ldots \mid \gamma_n$ do
        replace the $A$-production by $A \rightarrow \alpha a \beta X | \gamma_1 \mid \ldots \mid \gamma_n$
      end for
    end for
  until all production are of the form $A \rightarrow a$
  // create concept definitions                                           (3)
  $P' := \{A \equiv a \mid A \rightarrow a \in P\}$
```

sion to concepts. By definition grammatical symbols directly correspond to a concept names or concept expression ($\Sigma \cup N \subseteq \mathbf{C}$). Therefore a production of the form $A \rightarrow a$ can be directly rewritten to a concept definition. The rewriting process in the main loop finishes if all productions have this form. Within the loop expansions are continuously reduced until the expansion has only one (DL) concept by the following applying rewriting patterns: (a) Safely concatenable expansions are concatenated. (b) in rules of the form $A \rightarrow a \mid b \mid \ldots$ concept unions $a \sqcup b$ are formed for the single terminal expansions a and b. (c) Non-right-corner variables are substituted if there is an expansion with only one terminal symbol.

Example 10.1 (Transformation of a non-recursive grammar)

A set of production rules

$$\begin{array}{lll} A \rightarrow B\ C & B \rightarrow a & C \rightarrow d \\ & B \rightarrow b\ c & C \rightarrow e\ f \end{array}$$

can be rewritten to $\mathcal{SEQ}$ concepts

$$\begin{array}{l} A \equiv (a \triangleright C) \sqcup (b \triangleright c \triangleright C) \\ C \equiv d \sqcup (e \triangleright f) \end{array}$$

by applying the following transformation steps:

1. *Production rules are normalised.*
   $A \rightarrow B\ C, \quad B \rightarrow a \mid b\ c, \quad C \rightarrow d \mid e\ f$

2. *Apply the following steps until all expansions are of the form $N \rightarrow \Sigma$:*

   (a) *Safely concatenable expansions are transformed to concepts:*

   i. $A \rightarrow B\ C, \quad B \rightarrow a \mid (b \overset{\triangleright}{+\!\!+} c), \quad C \rightarrow d \mid (e \overset{\triangleright}{+\!\!+} f)$

   ii. $A \rightarrow B\ C, \quad B \rightarrow a \mid (b \triangleright c), \quad C \rightarrow d \mid (e \triangleright f)$

   (b) *Concept unions are formed for choices of single-terminal expansions:*
   $A \rightarrow B\ C, \quad B \rightarrow a \sqcup (b \triangleright c), \quad C \rightarrow d \sqcup (e \triangleright f)$

   (c) *Right-hand side non-right-corner variables are substituted if their expansion is a single terminal.*
   In this example only B is non-right-corner :
   $A \rightarrow (a \sqcup (b \triangleright c))\ C \qquad C \rightarrow d \sqcup (e \triangleright f)$

   Second iteration.

   (a) *The A-production is now safely concatenable as well:*
   $A \rightarrow (a \sqcup (b \triangleright c)) \overset{\triangleright}{+\!\!+} C \qquad C \rightarrow d \sqcup (e \triangleright f)$
   $A \rightarrow (a \triangleright C) \sqcup (b \triangleright c \triangleright C) \qquad C \rightarrow d \sqcup (e \triangleright f)$

   All expansions are now of the form $N \rightarrow \Sigma$.

3. Finally concept definitions are formed:

$A \equiv (a \triangleright C) \sqcup (b \triangleright c \triangleright C) \qquad C \equiv d \sqcup (e \triangleright f)$

This example also shows that the transformation performs a partial materialisation. In this case B is substituted and C is retained.

10.6 Transformability

Our transformation focused on the types of productions that occur in context-free grammars as well as on restrictions of these. Figure 10.1 sketches the relationship of grammar formalisms and $\mathcal{SEQ}$ patterns.

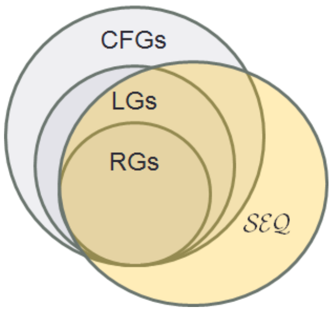


Figure 10.1: Relationship between grammars and $\mathcal{SEQ}$ patterns with respect to describable sequences

Our method can transform all *finite* context-free grammars to $\mathcal{SEQ}$ patterns. Our current algorithm transforms only safely concatenable productions directly while other forms of productions lead to a partial materialisation of (parts of) sequences.

Regarding *infinite* grammars we find that right recursive grammars can directly be transformed to $\mathcal{SEQ}$ patterns. Generally, left recursive context-free grammars can be transformed into (weakly equivalent) right recursive grammars (Moore, 2000). Therefore our transformation fully covers the types of productions that occur in *regular grammars*. Further, the special cases of *left and right linear grammars* can be transformed. Self-embedding grammars can not generally be expressed in our approach as they are not generally reducible to a regular grammar. Furthermore, it is not decidable if a self-embedding grammar is reducible. This is however possible in special cases, some of which have been characterised by **?**. We have not incorporated these special cases in our algorithm but this is an option for future work if it is interesting in the context of our application.

Other forms of grammars. OWL2-DL supports expressive means that are not directly available in CFGs such as boolean constructs or means for underspecification. The algorithm allows to mix DL concepts and grammar syntax.

Future research could be investigating the relationship between $\mathcal{SEQ}$ and other forms of grammars. While CFGs only support disjunction as logical operation, recently developed grammar formalisms such as conjunctive grammars (Okhotin, 2001) and boolean grammars (Okhotin, 2003) include further logical operators.

10.7 Optimisation of concept size

The number of expansions that are materialised can be reduced by introducing new variables.

A concatenation $C \overset{\triangleright}{+\!\!+} D$ restricts the leaves of the concept-tree of its first operand C to be succeeded by its second operand D. In this process the concept D is duplicated as often as there are leaves of C.

Note that for D it is sufficient to provide the root of the succeeding sequential structure. If D is a complex expression, we can always introduce a fresh concept name $D' \equiv D$. In this case just the name D' is duplicated whereas the potential verbose D still remains non-redundant as part of a single definition axiom. This suggests a way to optimise the transformation with respect to redundancy of concept parts by introducing new concept names. As less productions are materialised the resulting knowledge base will have many interrelated shorter definitions instead of few very long independent definitions. We can expect that higher interrelation is beneficial for classification times as results can be reused.

The current algorithm does not include this optimisation, but could be extended. One option is to introduce new concept definitions directly when concatenating. Note, that right-*regular* grammars by definition have the desired form. This suggest to transform a given set of productions if possible to a right-regular productions in a preprocessing step.

10.8 Summary

In this chapter an algorithm to rewrite grammars to $\mathcal{SEQ}$ patterns was developed with the focus on context-free grammars as these are popular means to express harmonic analysis. Although context-free grammars are not generally translatable to DLs, a partial transformation has been described in section 10.4 that allows instance classification with $\mathcal{SEQ}$ patterns for many types of production rules as characterized in section 10.6. Notably, right-recursive linear grammars can directly be transformed to $\mathcal{SEQ}$ patterns. Left-recursive grammars can be transformed to right-recursive grammars in a preprocessing step. Additional types of production rules can be treated by partial materialisation.

An application of this formalism to a musicological problem is presented in chapter 15.

11 API for $\mathcal{MEO}/\mathcal{SEQ}$

We implemented an API for $\mathcal{MEO}/\mathcal{SEQ}$ based on the OWL API[1], a Java API and reference implementation for creating, manipulating and serialising OWL ontologies that is supported by most standard reasoners. We added object representations for $\mathcal{MEO}/\mathcal{SEQ}$ constructs along with algorithms to transform them into OWL expressions as described in the previous chapters. The implementation is provided online.[2]

[1] `http://owlapi.sourceforge.net/`

[2] `http://www.j3w7.de/phd/`

As complexity depends on the choice of constructs we can control worst-case complexity by restricting use to a selected subset of concepts with desirable properties.

One approach taken by many ontology providers is to offer special fragments of an ontology. For example, the upper-ontology DOLCE has, next to the fully axiomatised standard version, several "light" versions (e.g, DOLCE-Lite-Plus) with better reasoning behaviour. However, this approach is not flexible as a user does not have fine-grained control over which combination of constructs are needed.

We propose an alternative approach that generates the required ontology at modelling time. The API is aware of the dependencies between constructs and background knowledge and dependencies are collected during concept construction. For example, the construct $C \triangleright D$ is translated to $C \sqcap \exists \mathit{seq}{:}\texttt{hasNext}.D$ with the additional axioms $\{\texttt{fun hasNext}\}$. When reusing this concept in other constructs the additional axioms are propagated. In this way all the needed axioms are aggregated during the process of concept and knowledge base construction. Moreover, axioms that are not relevant are not included. This is important as many DL constructs such as disjunction or transitive roles raise the reasoning complexity significantly.

Within the class *OWLDataFactoryImpl* the OWL API provides methods to compose complex concepts from simple concepts and axioms from concepts are provided. In our implementation we augment these "make" methods such that they propagate a set of additional axioms. As there are around 200 methods, a manual extension of each method would be laborious and hard to maintain if the OWL API is extended in future. However, the augmentation can be expressed in a uniform way as a transformation of simple make functions into augmented make functions. In contrast to functional programming languages, Java does not directly provide means to transform functions. However, libraries exists provide functional programming concepts within Java to achieve composition-oriented development. We used the library functionaljava[3] for this purpose and wrapped all OWL API into *Function* objects. For conciseness of presentation we use usual mathematical notation for function composition in the following. Further we focus on a small set of methods and abstract technical details of Java implementation. Here are some examples of the “make” methods in functional form:

[3] `http://functionaljava.org/`

$$\begin{array}{llll}
make_{\neg}: & Concept & \rightarrow & Concept \\
 & C & \mapsto & (\neg C) \\
make_{\sqcap}: & Concept \times Concept & \rightarrow & Concept \\
 & C, \quad D & \mapsto & (C \sqcap D) \\
make_{\exists}: & Role \times Concept & \rightarrow & Concept \\
 & R, \quad C & \mapsto & (\exists R.C) \\
make_{\equiv}: & Concept \times Concept & \rightarrow & Axiom \\
 & C, \quad D & \mapsto & (C \equiv D)
\end{array}$$

Attachment of background knowledge. $\mathcal{SEQ}$ syntax is embedded in DL syntax. $\mathcal{SEQ}$ constructs are translated to DL entities with additional axiomatisation. We extended the *make* functions to carry around

$$\begin{array}{llll}
augment(make\_\sqcap): & Concept^{+} \times Concept^{+} & \rightarrow & Concept^{+} \\
 & (C, axioms_1),\ (D, axioms_2) & \mapsto & (C \sqcap D, axioms_1 \cup axioms_2)
\end{array}$$

We refer to concepts, roles and individuals collectively as entities. We define the type defined by the set

$$Entity := Concept \cup Role \cup Individual$$

We define a new type $Entity^{+} := Entity \times 2^{Axiom}$ that represents an entity that is augmented with a set of axioms. A plain entity can then be axiomatised using functions

$$\begin{array}{llll}
augment: & Entity & \rightarrow & Entity^{+} \\
 & e & \mapsto & (e, \varnothing) \\
augment: & Entity \times 2^{Axiom} & \rightarrow & Entity^{+} \\
 & e \quad axioms & \mapsto & (e, axioms)
\end{array}$$

Entities and axiomatisation can be recovered using the projection functions π_1 (get the entity) and π_2 (get the axioms) that return the first resp. second component of the product.

The predefined *make* functions of the OWL API have to be extended to operate on *augmented* entities. The construct composition of the new augment function remains as in the plain *make* functions. In the unary augmented function axioms are merely propagated. In the binary augmented function axioms are joined.

$$\begin{aligned}
\mathit{augment}: \ & (\mathit{Entity} \to \mathit{Entity}) && \to (\mathit{Entity}^+ \to \mathit{Entity}^+) \\
& \mathit{make} && \mapsto (\mathit{make} \circ \pi_1) \times \pi_2 \\
\mathit{augment}: \ & (\mathit{Entity} \times \mathit{Entity} \to \mathit{Entity}) && \to (\mathit{Entity}^+ \times \mathit{Entity}^+ \to \mathit{Entity}^+) \\
& \mathit{make} && \mapsto (\mathit{make} \circ (\pi_1 \times \pi_1)) \times (\cup \circ (\pi_2 \times \pi_2))
\end{aligned}$$

11.1 Summary

This chapter described parts of the implementation of $\mathcal{MEO}$ and $\mathcal{SEQ}$ focusing on the mechanism of pattern composition. In contrast to providing a fixed ontology with axioms for $\mathcal{MEO}$ and $\mathcal{SEQ}$ constructs, the API allows to flexibly gather axioms during pattern creation, rewriting and composition. This way, a minimal ontology can be generated for given patterns that potentially avoid unnecessary complexity of axiomatisation and incompatible axiom types.

Part III

Evaluation and Applications

In this part, $\mathcal{MEO}/\mathcal{SEQ}$ are evaluated with respect to reasoning performance and applied in case studies for music analysis and music information retrieval.

First, the complexity of sequence classification and retrieval using OWL reasoning is discussed based on theoretical considerations on the impact of the used modelling constructs on worst-case complexity and based on performance experiments with generated pattern and sequences (cf. chapter 12).

Then, the integration of existing musical data sets with $\mathcal{MEO}/\mathcal{SEQ}$ is described (cf. chapter 13). Semantic integration using OWL is achieved by developing a binding ontology between the OMRAS2 Chord Ontology and $\mathcal{SEQ}$. The 9GDB corpus of chord sequences is translated into $\mathcal{MEO}/\mathcal{SEQ}$ format and mappings between the Isophonics corpus and DBpedia are provided.

In chapter 14 we show how patterns that have been discovered using machine learning techniques (Conklin, 2010; Anglade and Dixon, 2008) can be represented as $\mathcal{SEQ}$ patterns. The patterns are then organised by computing their subsumption relationships using OWL reasoning.

In chapter 15 we discuss the possibility of harmonic analysis of music on the web. We consider grammars as means to create such analysis. can be described and computed using grammars. Initially, we discuss a hybrid approach that produces a harmonic annotation in RDF by means a Prolog DCG grammar. Then we take a purely Semantic Web based approach based on the translation of a grammar into $\mathcal{SEQ}$ patterns and the use OWL classification.

Finally, in chapter 16, we show web retrieval of musical information using federated SPARQL queries . Especially we demonstrate the combination of different kinds of musical information, i.e. cadential patterns and metadata.

12 Profiling of $\mathcal{SEQ}$ Retrieval and Classification

An important aspect of knowledge representation and reasoning technologies are requirements in terms of computation time and memory, which determine limits of applications (e.g. length of patterns) and environments where the technology can be used (e.g. mobile devices). This section addresses the complexity of typical tasks performed on the $\mathcal{SEQ}$ representation.

12.1 Theoretical Considerations

12.1.1 Worst-Case Complexity

The complexity of Description Logics is well studied. Usually the worst case complexity of reasoning problems in a language is discussed, but running times may be better when they are evaluated empirically for a given knowledge base and reasoner.

The complexity of DLs differs with the combination of constructs used (cf. Baader et al., 2003). Initially, studies about the complexity of reasoning problems in DLs were mainly focused on polynomial-time versus intractable problems in order to guarantee timely answers. However, once very expressive DLs with exponential-time reasoning problems were implemented, it was found that knowledge bases of realistic size could still be processed in reasonable time (Horrocks, 1998).

In order to allow knowledge engineers to find the appropriate trade-off for their application, different language fragments of OWL have been defined. OWL 2 DL is currently the most expressive logic while OWL 2 EL, RL and QL are less expressive subsets of DL but show better complexities. For satisfiability checking within the standardised OWL fragments we find the following worst-case complexities (cf. Motik et al., 2009a):

OWL 1 DL	($\mathcal{SHOIN}$)	is NExpTime-complete
OWL 2 DL	($\mathcal{SROIQ}$)	is 2NExpTime-complete
OWL 2 EL	($\mathcal{EL}^{++}$)	is PTIME-complete
OWL 2 RL	($\mathcal{DLP}$)	is PTIME-complete
OWL 2 QL	(DL-Lite)	is in AC0 w.r.t. size of data
OWL 2 Full		is semi-decidable

Table 12.1: Complexity of satisfiability checking in OWL fragments

The worst-case complexity of reasoning on a given knowledge base with $\mathcal{MEO}/\mathcal{SEQ}$ can be determined by identifying the OWL language constructs that occur in the knowledge base.

The basic sequential construct is direct succession that by definition uses the following constructs:

$$\{\triangleright\} \xrightarrow{\textit{depends on}} \{C \sqcap D \quad \exists R.C \quad \texttt{fun}\ \}$$

These constructs are contained in all lightweight fragments (EL, RL and QL) and can we can thus satisfiability checking can be guaranteed PTIME complexity. The use of *seq:*`followedBy` introduces the following dependencies:

$$\{\cdots\} \xrightarrow{\textit{depends on}} \{\triangleright \quad R \sqsubseteq S \quad \texttt{tra}\}$$

This forces use to move to the DL variants of OWL in order to express role subsumption (in $\mathcal{H}/\mathcal{R}$) and transitivity (in $\mathcal{S}$) and exponential complexities may arise. The omission of *seq:*`followedBy` in order to keep reasoning tractable is difficult as it is not only used to define more expressive sequential patterns but also to express sequential closure (beginning and end) that is relevant in the context of the open world assumption:

$$\alpha/\omega \xrightarrow{\textit{depends on}} \{\cdots \quad \exists^{\leq n} R.C \quad \neg C\}$$

We therefore need a language that allows negation and qualified number restrictions ($\mathcal{Q}$). Again only the OWL-DL fragment offers this.

12.1.2 Effects of set-based DL Semantics

A Description Logic knowledge base is defined as a *set* of axioms. There is no prescribed ordering mechanism in the OWL specification. Accordingly, typical definitions of reasoning procedures such as tableaux algorithms for DLs (cp. section 3.4) do not consider order. Reasoners might implement indexing and ordering mechanisms on the knowledge base for optimisation purposes.

In a procedural implementation the evaluation of a regular expression is typically an iterative process that starts from the beginning of a sequence and will take longer if the matching part occurs later in the sequence. Given the set-based specification of DL, the reasoning algorithm can start at an any component and the impact of matching position and sequence length will depend on the behaviour of the given reasoner implementation. Therefore an empirical evaluation is necessary to assess the actual behaviour of OWL reasoners.

12.2 Performance tests

While the worst-case behaviour of the developed techniques can be characterised theoretically, empirical tests are necessary to observe the actual behaviour of reasoners. In the following we describe profiling experiments for sequence classification and retrieval based on generated abstract patterns and data.

Technical setup. The experiments were implemented using the Java OWL-API v3.2.2[1], conducted using the reasoner FACT^{++} 1.4.1[2] and

[1] `http://owlapi.sourceforge.net/`

[2] `http://owl.man.ac.uk/factplusplus/`

performed on a notebook computer with 1GB RAM allocated to the reasoner.

12.2.1 Performance of Instance Retrieval

Experiment 1. The knowledge base consisted of single sequences varying in length n. The components of a sequence assert to be of type A except for one component that was of type M and was supposed to be the matching position of the pattern. We use [M] as pattern and varied the position *pos* of the component of type M in the sequence with the intention to measure how the position affects the retrieval time. I.e. a test sequence had the form:

$$\ulcorner\!\_{:}c_1 \blacktriangleright \ldots \blacktriangleright \_{:}c_n\!\lrcorner \quad \text{with } \_{:}c_i \in \begin{cases} \mathtt{M} & \text{if } i = pos \\ \mathtt{A} & \text{otherwise} \end{cases}$$

The results in Figure 12.1 show that while the length of the classified sequences ($n \in \{100, 200, 300\}$) had an impact on the retrieval time, no effect for position has been found. This indicates that the reasoner does not take sequential order into account when retrieval as was expected from the set-based definition of DL semantics and reasoning (cp. subsection 12.1.2).

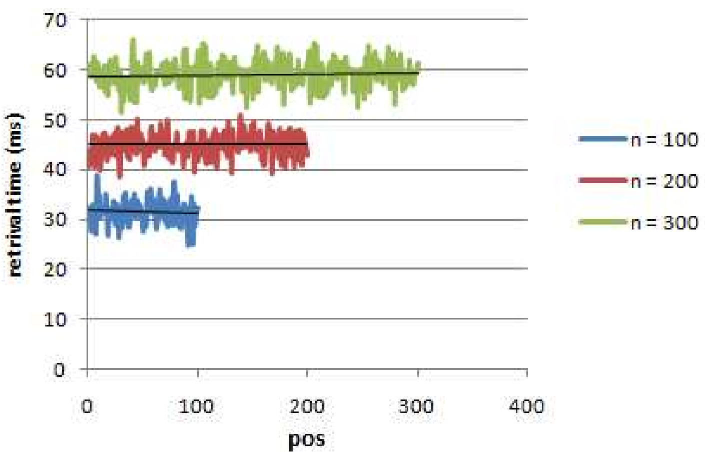


Figure 12.1: Time needed to retrieve instances of a pattern [M] from sequences of varying length n and varying matching position *pos* of [M]. Retrieval times rise with length of sequence but stay constant relative to matching position.

Experiment 2. A further experiment investigated the effect of the pattern length. We modified the initial setup such that we used matching pattern $[\mathtt{M}]^{len}$ of varying length *len* and sequences contains matching parts of varying length, i.e.

$$\ulcorner\!\_{:}c_1 \blacktriangleright \ldots \blacktriangleright \_{:}c_n\!\lrcorner \quad \text{with } \_{:}c_i \in \begin{cases} \mathtt{M} & \text{if } i \in [pos \ldots pos + len] \\ \mathtt{A} & \text{otherwise} \end{cases}$$

The position *pos* was randomly chosen as pattern matches an instance component has no effect on the retrieval time.

The results in Figure 12.2 indicate a linear effect of pattern length on matching. With longer sequences the effect remains linear while the gradient increases. The dips result from memory management strategies of the reasoner and can be neglected.

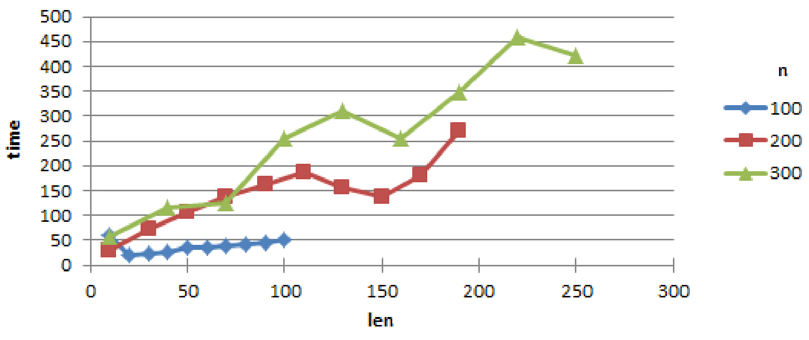


Figure 12.2: Time needed to retrieve instances of a pattern $\texttt{[M]}^{len}$ where the length *len* of the pattern as well as the length n of the sequences was varied.

12.3 Optimisations

We have developed two new approaches to enhance the reasoning performance, which are described in the following two sub-sections. These approaches have been implemented and successfully tested, but an empirical evaluation remains for future work.

12.3.1 Precomputing assertions and reasoning w.r.t. a simplified the TBox

Reasoning on a knowledge base that contains the full set of $\mathcal{MEO}/\mathcal{SEQ}$ axiomatisation and a large set of instance data can be highly costly. One major source of reasoning complexity is role transitivity. Removing all transitivity axioms from a TBox will improve reasoning performance but at the expense of loosing expressive means and relevant inferences. For example, we use the transitive *seq:*`followedBy` role in a pattern $\alpha \cdots P$ to retrieve the sequences in which P occurs. Note, however, that all instances of $\alpha \cdots P$ can be retrieved even if *seq:*`followedBy` is not transitive iff all *seq:*`followedBy` relationships between instances of α and P have been made explicit.

Following this considerations, one approach to optimize is thus to

1. divide the ABox into partitions that contain one or few sequences,
2. materialise implicit knowledge in these partitions, especially such knowledge that is derived by transitivity,
3. simplify the TBox by removing transitivity axioms and
4. recombine the simplified TBox with the expanded partitions to a new knowledge base.

More formally, consider a knowledge base $\mathcal{KB} := \mathcal{T} \cup \mathcal{A}$ where $\mathcal{T}$ is a TBox with $\mathcal{SEQ}$ and $\mathcal{MEO}$ and $\mathcal{A} := \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_n$ is an ABox that is partitioned into sets of sequence axiomatisations $\mathcal{S}_{(i)}$. A *simplified* knowledge base $\mathcal{KB}^-$ for $\mathcal{KB}$ is defined

$$\mathcal{KB}^- := \mathcal{T}^- \cup \mathcal{A}^+$$

where $\mathcal{T}^-$ is a simplified TBox

$$\mathcal{T}^- := \mathcal{T} \setminus \{\texttt{tra}\ R \mid R \in \Sigma(\mathcal{T})\}$$

in which for all roles R transitivity has been removed and $\mathcal{A}^+$ is an expanded ABox

$$\mathcal{A}^+ := \mathcal{S}_1^+ \cup \ldots \cup \mathcal{S}_n^+$$

with expanded partitions $\mathcal{S}_{(i)}^+$. Different approaches can be taken to defining and computing these expansions. One approach is to materialise the deductive closure of each fragment with respect to the original TBox $\mathcal{T}$, i.e. $\mathcal{S}_{(i)}^+ := \langle \mathcal{T} \cup \mathcal{S}_{(i)} \rangle \setminus \langle \mathcal{T} \rangle$. Note that we assume the deductive closure to be finite.

Computing the deductive closure might expand the knowledge base considerably and has still its expenses to compute. While the restriction to reasoning over ABox fragments already improves the feasibility of reasoning, we achieve further optimisation by considering only *relevant* subsets $\mathcal{T}_{(i)} \subseteq \mathcal{T}$ for the computation of the deductive closure, i.e. $\mathcal{S}_{(i)}^+ := \langle \mathcal{T}_{(i)} \cup \mathcal{S}_{(i)} \rangle \setminus \mathcal{T}$. The partitioning of the ABox along sequence axiomatisations is possible as we assume that these assertions not logically interact which each other. The interaction of TBox axioms on the other hand might be neither obvious nor trivial and hand-picking of relevant axioms might be error-prone. A solution for this problem is automatic *module extraction* technique (e.g. Cuenca Grau et al., 2008), which can be used to select for a given ontology and signature all axioms from the ontology that are necessary to preserve all entailments that involve the entities of the signature.

$$\mathcal{S}_{(i)}^+ := \langle Module(\mathcal{T}, \Sigma_{(i)}) \rangle \cup \mathcal{S}_{(i)} \setminus \mathcal{T}$$

No new axioms are added during the process and by monotonicity of OWL reasoning the deductive closure (cf. section 3.1) of the resulting knowledge base is a subset of the original one, i.e. $\langle \mathcal{KB}^- \rangle \subseteq \langle \mathcal{KB} \rangle$.

12.4 Summary

In this chapter the performance of sequential reasoning in OWL was examined. As the worst-case reasoning complexity for an ontology depends on the constructs used it has bene discussed theoretically in section 12.1 based on the OWL constructs used in $\mathcal{SEQ}$ constructs. Only simple succession can be expressed in the lightweight fragments (EL, RL and QL) with PTIME complexity. Sequential closure (beginning and end) or the use of *seq:*`followedBy` introduces constructs that are only expressible in OWL-DL thus exponential worst-case complexities are possible. Given the set-based definition of DLs it can be further assumed that retrieval and classification times do not depend on the matching position of a pattern. The experiments in subsection 12.1.2 confirmed this assumption. Further, the experiments showed that the total number of sequential components increased retrieval and classification times regardless of whether the components belong to different sequences. The experiments suggest possibilities to optimisation by exploiting order information during reasoning.

While many optimisations require a modification of reasoning algorithms, section 12.3 introduced optimisations that can be carried out in preprocessing steps. Especially, performance can be increase by modularising the KB into partitions on which reasoning is performed separately. Further costly inferences such as computing transitive relationships can be precomputed and online reasoning tasks can then be performed with a simplified TBox on the materialised knowledge.

13 Data Integration

In the following chapters we demonstrate analysis and querying of chord sequences with $\mathcal{SEQ}$ and $\mathcal{MEO}$. Quality controlled chord data sets are available, yet for using $\mathcal{SEQ}$ and $\mathcal{MEO}$ with them questions of data integration have to be addressed at two levels:

Structural Integration Corpora should be represented in the same syntactic format in order to be queried with the same query language. For example, using RDF for data representation allows federated SPARQL queries on distributed data sets on the web.

Semantic Integration Ontologies assign meaning to syntactic structures. If different ontologies are used the relationship between their terminologies has to be described.

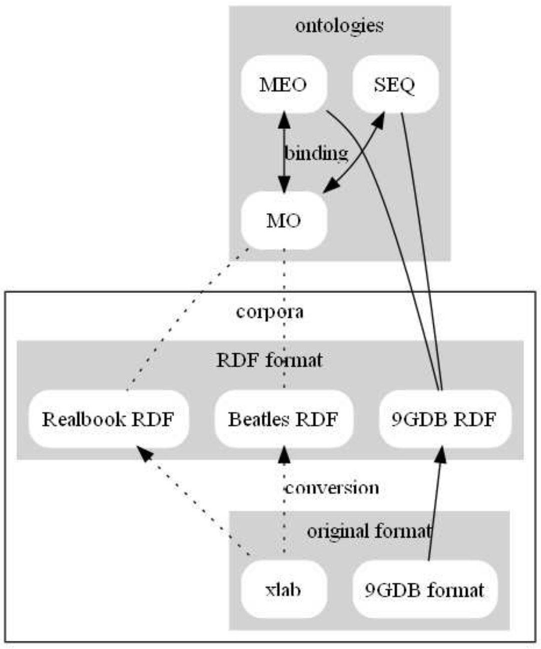


Figure 13.1: Overview of data sets and ontologies.

Figure 13.1 gives an overview of corpora and ontologies that have been integrated in order to allow retrieval and classification with $\mathcal{MEO}/\mathcal{SEQ}$ patterns. We used annotated Jazz and Beatles corpora as ground-truth data. Parts of these were given in textual formats (xlab, 9gdb) and had to be converted into RDF and described by $\mathcal{MEO}/\mathcal{SEQ}$ terminology. On the other hand corpora that were available in RDF format (Beatles) and described by music ontology needed to be integrated with $\mathcal{MEO}/\mathcal{SEQ}$ by providing bindings between the ontologies. We now describe some of the concrete steps.

13.1 Structural Integration

13.1.1 9GDB Corpus

The 9GDB (9 genres database)[1] provides a corpus of chord progressions from nine different genres taken from three domains: popular, jazz, and academic music. Categories have been defined with the help and advice of music experts who have also collaborated in the task of assigning metadata tags to the files and rejecting outliers in order to have a reliable ground truth (Pérez-Sancho et al., 2009). Popular music data have been separated into three sub-genres: pop, blues, and celtic (mainly Irish jigs and reels). For jazz, three styles have been established: a pre-bop class grouping swing, early Jazz, and Broadway tunes, bop standards, and bossa novas as representatives of latin jazz. Finally, academic music has been categorised according to historic periods: baroque, classicism, and romanticism.

[1] `http://grfia.dlsi.ua.es/cm/projects/prosemus/database.php`

Academic	235	Jazz	338	Popular	283
Baroque	56	Pre-bop	178	Blues	84
Classical	50	Bop	94	Pop	100
Romanticism	129	Bossa nova	66	Celtic	99

Table 13.1: Number of chord sequences per genre and subgenre.

Transformation to $\mathcal{MEO}/\mathcal{SEQ}$. The 9GDB corpus is represented in a textual format that has to be transformed to $\mathcal{MEO}/\mathcal{SEQ}$ data.

The sequences in the 9GDB corpus are given in four different variants spanned by the dimensions root-note/degree form and triad/full form as shown in Table 13.2.

Am	G#dim	Gm	F#dim	F	Bb	Bdim	E	Am	G#	Gm	F#dim	F	...
Am7	G#dim	Gm7	F#m7b5	Fmaj7	Bbmaj7	Bm7b5	E7	Am7	G#7	Gm7	F#m7b5	Fmaj7	...
Im	VII#dim	VIIm	VI#dim	VI	IIb	IIdim	V	Im	VII#	VIIm	VI#dim	VI	...
Im7	VII#dim	VIIm7	VI#m7b5	VImaj7	IIbmaj7	IIm7b5	V7	Im7	VII#7	VIIm7	VI#m7b5	VImaj7	...

Table 13.2: Beginning of "How Insensitive" by Antonio Carlos Jobim represented in the 9GDB format

To translate this representation into RDF, a parser was implemented based on the grammar given in Table 13.3. The parse tree is then mapped to a $\mathcal{MEO}$ description.

13.2 Semantic Integration

Knowledge engineers might follow different modelling patterns for the same use case. For example, OMRAS2 as well as $\mathcal{SEQ}$ provide constructs to order chords. We can use OWL axioms to capture the logical

```
<chordseq> := (<chord>' ')*
<chord>     := (<root>|<degree>)(<triad>|<>)<extension>*
<root>      := <natural><alteration>
<degree>    := <roman><alteration>
<natural>  := C|D|E|F|G|A|B
<roman>     := I|II|III|IV|V|VI|VII
<triad>     := dim|aug|sus|(b5)|(+5)|m(+5)|m
<alteration> := (b)*|(#)*
<extension>  := 4|6|7+|7|9|#9|#11|13|b5|alt|maj7|maj9|maj|whole
```

Table 13.3: EBNF grammar for 9GDB chord sequences

triad version:	dim	aug	sus	(b5)	(+5)	m(+5)
full version:	m7b5	7+|7alt|whole	11|4|13sus	7alt|whole	#5	m#5

Table 13.4: Corresponding triad and full variants of extensions as used in the 9GDB corpus

relationship between the corresponding constructs in the two ontologies and thus make them interoperable.

13.2.1 Isophonics Beatles Corpus

The Centre for Digital Music (C4DM) at Queen Mary, University of London, provides several ground truth song collections of the OMRAS2 Metadata Project 2009 (Mauch et al., 2009) on the Isophonics.net website.[2]

The Beatles corpus[3] contains 174 reference chord transcriptions.

For all songs audio-aligned reference chord and key annotations as well as annotations of musical structural segmentation are provided. Chords are represented following Harte et al. (2005) in an RDF variant using “The Music Ontology”. and “The Chord Ontology”[4]

[2] http://www.isophonics.net

[3] http://isophonics.net/content/reference-annotations-beatles

[4] http://motools.sourceforge.net/chord_draft_1/chord.html

13.2.2 A binding ontology for OMRAS 2 Chord Ontology

Within the OMRAS2 Music Ontology effort (Raimond and Sandler, 2008) The Chord Ontology has been developed[5] that recently gained increasing research interest and in which chord data sets have been made available. As discussed in section 6.1, the descriptions are not sufficiently formalised to query their internal structure with OWL reasoning. It is therefore interesting to use the chord ontology data together with $\mathcal{SEQ}$-patterns. One way would be to convert the sequences from a chord ontology ABox into a $\mathcal{SEQ}$ sequence assertions. Another approach is to reason over the data with a TBox of the form

$$\mathit{ChordOntology} \cup \mathit{BindingOntology} \cup \mathcal{SEQ}$$

where *BindingOntology* is a set of OWL axioms that describe how Chord Ontology concepts formally relate to $\mathcal{SEQ}$ concepts. The advantage of this approach is that the ABox does not need any alteration. Given this TBox an OWL reasoner will recognise the ABox individuals as instances of $\mathcal{SEQ}$ concepts so that $\mathcal{SEQ}$ patterns can classify the data.

[5] cf. http://motools.sourceforge.net/chord_draft_1/chord.html

One possible binding ontology can be defined using the axioms:

$$\begin{aligned} \mathit{chord}{:}\texttt{ChordEvent} &\sqsubseteq \mathit{seq}{:}\texttt{Component} \\ \mathit{chord}{:}\texttt{chord} &\sqsubseteq \mathit{seq}{:}\texttt{hasContent} \\ \mathit{time}{:}\texttt{time} \circ \mathit{time}{:}\texttt{intervalAfter} \circ \mathit{time}{:}\texttt{time}^{-} &\sqsubseteq \mathit{seq}{:}\texttt{hasNext} \end{aligned}$$

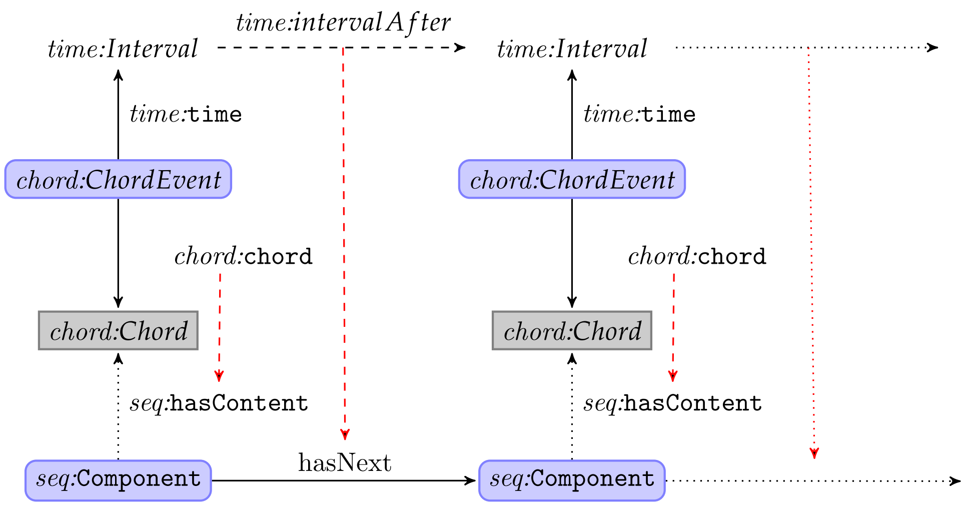


Figure 13.2: Mapping from OMRAS2 Music Ontology to $\mathcal{SEQ}$ Representation

The additional assertions are illustrated in figure 13.2. $\mathcal{SEQ}$ is treated as more general, i.e. definitions are provided to interpret ChordOntology concepts as $\mathcal{SEQ}$ concepts but not vice versa. The idea is to interpret a `ChordEvent` as a `Component`. The content roles then find a natural correspondence. The *seq:*`hasNext` relationship can be inferred from a role path between the `ChordEvent`s.

Note that due to the discussed limited reasoning possibilities of the current ChordOntology and the underlying OWL-time ontology (cf. section 5.5), it is not guaranteed that all information is given or can be inferred. Especially, the *time:intervalAfter* relationship was missing in some of the data and cannot be inferred by OWL reasoners. We saturated the ABox with this relationship using SWRL-rules and Java Build-ins for number comparision in such cases.

13.2.3 Representation of Chord Ontology Four-Chord Patterns

Recently there has been an interest in learning patterns of four succeeding chords from the OMRAS2 Chord Ontology data. Mauch et al. (2007) used a statistical approach to describe. Anglade and Dixon (2008) used an inductive logic programming approach. Both used the same corpus and pattern representation. As the patterns themselves have not been expressed with Semantic Web techniques, they can not directly be reused to find pattern instances in a web retrieval scenario.

This issue can be resolved by expressing the patterns in $\mathcal{SEQ}$. The patterns can then directly be used to retrieve any $\mathcal{SEQ}$ instances such as the reinterpreted Chord Ontology instances described in the previous section.

The pattern representation used by Mauch et al. (2007) considers the chord type and the chromatic root movement. The pattern

$$maj \xrightarrow{+11} min \xrightarrow{+5} maj \xrightarrow{+5} min$$

that corresponds to the opening sequence of *Yesterday* is discussed as an example. The author sees it as an interesting example for how chord sequences for known idioms can help to find evidence for the songwriters' influences as it is unique within the Beatles Database, but occurs in 17 songs in the Real Book including the Charlie Parker standard *Confirmation* and *Like Someone in Love* by Johnny Burke.

Anglade and Dixon (2008) uses an inductive logic programming (ILP) approach. She uses the four chord pattern structure as above, but a different set of transitions. Notably, she uses different descriptors, namely root note progression, bass note progression, chord category progression, root interval progression, bass interval progression and degree progression, computed by Prolog predicates.

$$maj \xrightarrow{maj6th} min \xrightarrow{perf4th} min \xrightarrow{perf4th} dom$$

As the patterns correspond to four chord sequences of Chord Ontology chord symbols, we can treat them as *specialisation* of a general $\mathcal{SEQ}$ pattern

$$[chord{:}chord]^{=4}$$

As a first step, we could now specialise the *chord:base_chord* role, that is provided by the Chord Ontology and is filled by the chord type. As the Chord Ontology models chord types as individuals and not as concepts, we use the *nominal* constructor[6] (indicated by squared brackets) to specialise the pattern:

6 Provided by DLs that support nominals (the "$\mathcal{O}$" in the naming scheme. This is supported OWL-DL, but not OWL Lite.

$$\begin{aligned} &[\exists chord{:}base\_chord\{maj\}] \\ \triangleright\ &[\exists chord{:}base\_chord\{min\}] \\ \triangleright\ &[\exists chord{:}base\_chord\{maj\}] \\ \triangleright\ &[\exists chord{:}base\_chord\{min\}] \end{aligned}$$

Note that while musicologists normally would expect such a pattern to match chord types such as $maj7$ or $maj9$, such instances would not match here as the Chord ontology currently does not formalise a notation of *chord subsumption*. This is quite a surprising omission as this is an important musicological concept and as subsumption modelling is a primary use case of DLs. Further, Mauch et al. (2007) as well as Anglade and Dixon (2008) use notions of chord subsumption in their respective learning algorithms. However, as this knowledge is not described in an ontology, it is not available for a DL reasoner. Mauch et al. (2007) for example uses three sets of chord classes, representing three different levels of generality. A workaround could be to define classes such as

$$\begin{aligned} MajChord &\equiv \exists chord{:}base\_chord.\{maj, maj7, maj9, \dots\} \\ MinChord &\equiv \exists chord{:}base\_chord.\{min, min7, \dots\} \end{aligned}$$

and use these instead:

$$[\mathit{MajChord}] \triangleright [\mathit{MinChord}] \triangleright [\mathit{MajChord}] \triangleright [\mathit{MinChord}]$$

It is easy to see, that a more elaborated subsumption hierarchy is desirable. The final translation step is the modelling of transition roles. They could be modelled relative to the previous component as

$$[\mathit{MajChord}] \triangleright \begin{bmatrix} & \mathit{MinChord} \\ \sqcap & \mathit{root.maj6th} \end{bmatrix} \triangleright \begin{bmatrix} & \mathit{MinChord} \\ \sqcap & \mathit{root.perf4th} \end{bmatrix} \triangleright \begin{bmatrix} & \mathit{MinChord} \\ \sqcap & \mathit{root.perf4th} \end{bmatrix}$$

or as subrole of *hasNext*:

$$[\mathit{MajChord}] \underset{\mathit{maj6th}}{\triangleright} [\mathit{MinChord}] \underset{\mathit{perf4th}}{\triangleright} [\mathit{MajChord}] \underset{\mathit{perf4th}}{\triangleright} [\mathit{MinChord}]$$

In both cases, as most of the computed roles do not have a direct correspondence in the OMRAS2 Chord Ontology data, the instance data would need to be saturated or a method to automatically infer these roles would need to be devised.

13.3 Summary

Data sets and ontologies were described that will be used in the following chapters to demonstrate several applications. Structural and semantic interoperability of these was desirable in order to make full use of Semantic Web query and reasoning technologies.

Structural interoperability can be established by using RDF as common representation for all datasets. This enables, for example, federated queries as will be shown in chapter 16. The nine genre database (9GDB) is a rich source of harmonically annotated chord sequences given in a textual representation which was translated into $\mathcal{MEO}/\mathcal{SEQ}$-described RDF using our API.

Another concern that was addressed is semantic integration. The Isophonics Beatles corpus is another dataset that we used for musical queries. It is available as RDF but uses terminology from the OMRAS2 Music Ontology (MO). In order to enable $\mathcal{MEO}$/SEQ patterns to match these MO-described sequences a reasoner needs to have access to background knowledge of how entities in one ontology can be interpreted in terms of the other. Our approach is to capture the translations in OWL axioms in a separate ontology that can be provided to a reasoner on demand. Such a binding ontology was defined to interpret MO entities as $\mathcal{MEO}$/SEQ entities. Further, it was shown how Four-Chord Patterns (Mauch et al., 2007; Anglade and Dixon, 2008) can be represented with $\mathcal{MEO}/\mathcal{SEQ}$.

14 Subsumption of Discovered Feature Patterns

In the following we will show how existing pattern representations for chord sequences can be translated into $\mathcal{SEQ}$ patterns and how $\mathcal{SEQ}$ can be used to analyse chord pattern subsumption and classify sequence instances.

14.1 Representation of Feature Set Patterns

Although $\mathcal{SEQ}$ patterns can be specified in a top-down matter by a knowledge engineer, it is interesting to learn them from a corpus of music. Pattern discovery using *multiple viewpoints* is a machine learning approach for discovering patterns in sequential musical data (Conklin, 2006). It has mainly been used for discovery of patterns in melodies, but recently also for learning patterns in chord progressions (Conklin, 2008, 2010). Input and patterns are represented using a *feature set* representation (Conklin and Bergeron, 2008).

For a sequence of musical events $e_1 \ldots e_n$ (e.g. chords), viewpoints are computed. A viewpoint τ is a function from events to values in a specific range set. A feature f is defined as

$$\tau : v$$

where τ is a feature name and v a feature value. A *feature set* f then is a *conjunction* of features

$$\{\tau_1 : v_1, \ldots, \tau_n : v_n\}$$

and a pattern is a sequence of feature sets

$$f_1, \ldots, f_n$$

	events	Im7	IVm7	Vbm7b5	IV7	III#m7b5
features	degree	I	IV	I	IV	III#
	triad	m	m	m	M	minb5
	rootmvt	$\perp$	4n	5n	7+	7n
	function	I	IV	V	IV	III

Table 14.1: Example of a instance sequence for viewpoint learning

Table 14.1 shows an example of how a chord progression is represented as a sequence of feature sets. The viewpoints $degree$, $triad$ and $function$ directly relate to the chord symbol. Relationships between events are modelled as features that belong to a single event and have to be read as referring back to the previous event. The feature *rootmvt* : $4n$ for example expresses that the current root event is a fourth about the previous event. In the case of the first event e_1 features of this kind take the value $\perp$ as there is no previous event they could refer to. We use further viewpoints in later examples such as *meeus* that indicates harmonic function (*T*onic, *D*ominant or *S*ubdominant) as described by Meeus (2000), *kp* that indicates chord degree classes as described by (Kostka and Payne, 2003) and $ratio(dur)$ that indicates the duration of an event.

14.2 Translation of Feature Set Patterns

This representation can be translated into $\mathcal{SEQ}$ data and patterns using translation functions T as described in following. Each feature $\tau : v$ can be translated into a DL role restriction

$$T(\tau : v) = \exists \tau . v$$

where every viewpoint τ corresponds to a *functional* role τ and the value v is the filler that the role is restricted to. A corresponding DL axiom Funct(τ) has to be added to the TBox. A feature set f, i.e. a *conjunction* of features is than described by a DL concept intersection

$$T(f) = \exists \tau_1 . v_1 \sqcap \ldots \sqcap \exists \tau_n . v_n$$

A feature set pattern $\{f_1 \ldots f_n\}$ can then be expressed using *hasNext* relationships as

$$T(f_1, \ldots, f_n) = [T(f_1)] \triangleright \ldots \triangleright [T(f_n)]$$

In the following we will show examples of how genre-specific chord sequence patterns that have been learnt from chord sequences tagged with the genres *jazz*, *classic* and *pop* can be learnt from a corpus.

14.3 Maximally General Distinctive Chord Patterns

A *maximally general distinctive chord pattern* (MGDP) is a pattern that is distinctive and not subsumed by any other distinctive pattern. They are least likely to overfit the corpus and hence most likely to be useful for classification. To measure distinctiveness the likelihood ratio of a pattern P is employed. This is defined as

$$\Delta(P) \overset{def}{=} \frac{p(P|\oplus)}{p(P|\ominus)} = \frac{c^{\oplus}(P)}{p(P|\ominus) \times n^{\oplus}}$$

where $p(P|\oplus)$ is the probability of the pattern P in the corpus, $p(P|\ominus)$ is the probability of the pattern P in the anticorpus, $c^{\oplus}(P)$ is the count of the pattern in the corpus and $n^{\oplus}$ is the size of the corpus (Conklin, 2010).

Figure 14.2 illustrates three MGDP patterns. The were chosen from a much larger set as highly distinctive patterns found in the corpus of 338 jazz chord sequences used by Pérez-Sancho et al. (2008).

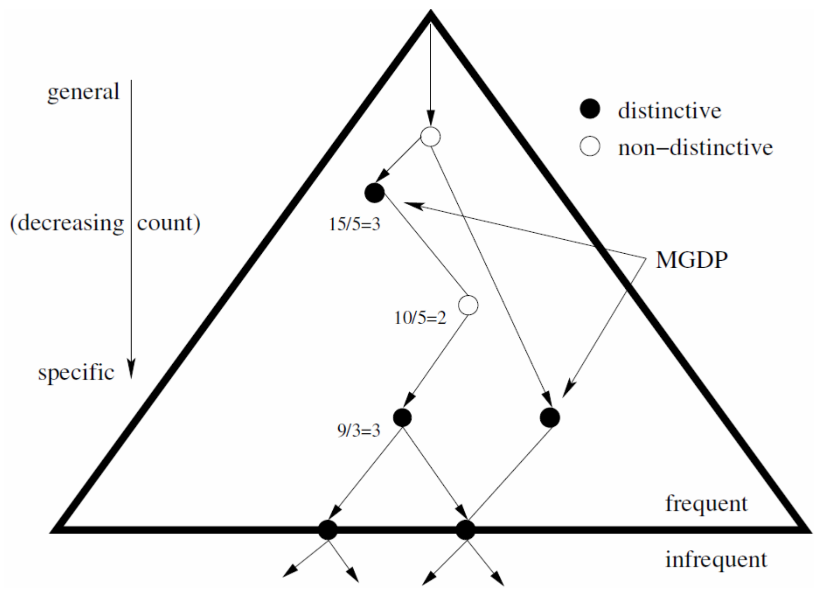


Figure 14.1: Regions of the pattern space for MGDPs, with examples of subsumption chains containing distinctive and non-distinctive patterns. (source: Conklin, 2010)

At the top of the figure are the three patterns after translation from viewpoint to $\mathcal{SEQ}$ format. The interest $\Delta(P)$ of the pattern is indicated: for example, the first pattern is overrepresented by a factor of 12.45. The length of the pattern is 2 and it occurs in 65 jazz sequences but only 8 sequences in the anticorpus. The pattern indicates a minor triad on degree III, followed by any triad on degree III (due to the fact that the *meuus* role indicates the T (tonic) chord transformation). Note that despite this high level of abstraction captured by this pattern, it remains highly distinctive in the corpus for the jazz genre.

In the bottom of Figure 14.2, instances of each of these patterns are represented as fully saturated feature set sequences.

14.4 Subsumption of Distinctive Pattern

After translating patterns from feature set representation to $\mathcal{SEQ}$ representation, subsumption relationships can automatically be inferred by a DL reasoner. Figure 14.3 illustrates a small fragment of a subsumption hierarchy of patterns, created from a larger set of many MGDP patterns. In this case the subsumption relationships were computed using the reasoner Pellet (Sirin et al., 2007). This figure has restricted the representation to five MGDP that appear on the right-hand side of the hierarchy. Some internal concepts have been constructed in $\mathcal{SEQ}$ and it can be seen how these capture commonalities between the MGDP thereby providing richer structure to a flat MGDP set. At the left-hand side of the figure are "primitive" features contained in single component patterns. Substantial structure can be seen. For example, *triad.min* can be seen to occur in four MGDPs and in addition in one internal $\mathcal{SEQ}$ pattern.

Pattern

$\Delta(P)$ =12.45 (2) (65 150) (8 25)

$$\left[\begin{array}{crcl} & \mathit{kp} & . & \mathit{III} \\ \sqcap & \mathit{basedegree} & . & \mathit{III} \\ \sqcap & \mathit{triad} & . & \mathit{min} \end{array}\right] \triangleright \left[\ \mathit{meeus} \,.\, T\ \right]$$

$\Delta(P)$ =9.04 (3) (59 107) (10 14)

$$\left[\begin{array}{crcl} & \mathit{meeus} & . & D \\ \sqcap & \mathit{kp} & . & \mathit{II/IV} \\ \sqcap & \mathit{triad} & . & \mathit{maj} \end{array}\right] \triangleright \left[\ \mathit{triad.min}\ \right] \triangleright \left[\begin{array}{crcl} & \mathit{kp} & . & \mathit{VI} \\ \sqcap & \mathit{basedegree} & . & \mathit{VI} \end{array}\right]$$

$\Delta(P)$ =7.66 (1) (60 146) (12 50)

$$\left[\begin{array}{crcl} & \mathit{meeus} & . & D \\ \sqcap & \mathit{kp} & . & \mathit{VI} \\ \sqcap & \mathit{degree} & . & \mathit{VIb} \\ \sqcap & \mathit{basedegree} & . & \mathit{VI} \\ \sqcap & \mathit{rootmvt} & . & 4n \end{array}\right]$$

Examples of matched instances

Look To The Sky

IIIm III♭ dim

$$\cdots \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & D \\ \mathit{kp} & . & \mathit{III} \\ \mathit{degree} & . & \mathit{III} \\ \mathit{basedegree} & . & \mathit{III} \\ \mathit{triad} & . & \mathit{min} \\ \mathit{rootmvt} & . & 4+ \end{array}\right] \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & T \\ \mathit{kp} & . & \mathit{III} \\ \mathit{degree} & . & \mathit{IIIb} \\ \mathit{basedegree} & . & \mathit{III} \\ \mathit{triad} & . & \mathit{dim} \\ \mathit{rootmvt} & . & 1n \end{array}\right] \blacktriangleright \cdots$$

Tangerine

I IV7 IIIm7 VI7

$$\cdots \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & D \\ \mathit{kp} & . & I \\ \mathit{degree} & . & I \\ \mathit{basedegree} & . & I \\ \mathit{triad} & . & \mathit{maj} \\ \mathit{rootmvt} & . & 4n \end{array}\right] \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & D \\ \mathit{kp} & . & \mathit{II/IV} \\ \mathit{degree} & . & \mathit{IV} \\ \mathit{basedegree} & . & \mathit{IV} \\ \mathit{triad} & . & \mathit{maj} \\ \mathit{rootmvt} & . & 4n \end{array}\right] \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & S \\ \mathit{kp} & . & \mathit{III} \\ \mathit{degree} & . & \mathit{III} \\ \mathit{basedegree} & . & \mathit{III} \\ \mathit{triad} & . & \mathit{min} \\ \mathit{rootmvt} & . & 7n \end{array}\right] \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & D \\ \mathit{kp} & . & \mathit{VI} \\ \mathit{degree} & . & \mathit{VI} \\ \mathit{basedegree} & . & \mathit{VI} \\ \mathit{triad} & . & \mathit{maj} \\ \mathit{rootmvt} & . & 4n \end{array}\right] \blacktriangleright \cdots$$

Quiet Now

III♭ maj VI♭ maj7

$$\cdots \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & D \\ \mathit{kp} & . & \mathit{III} \\ \mathit{degree} & . & \mathit{IIIb} \\ \mathit{basedegree} & . & \mathit{III} \\ \mathit{triad} & . & \mathit{maj} \\ \mathit{rootmvt} & . & 2n \end{array}\right] \blacktriangleright \left[\begin{array}{rcl} \mathit{meeus} & . & D \\ \mathit{kp} & . & \mathit{VI} \\ \mathit{degree} & . & \mathit{VIb} \\ \mathit{basedegree} & . & \mathit{VI} \\ \mathit{triad} & . & \mathit{maj} \\ \mathit{rootmvt} & . & 4n \end{array}\right] \blacktriangleright \cdots$$

corpus	:	338 chord sequences (jazz)
anticorpus	:	518 chord sequences (pop, classical)

Figure 14.2: Example patterns and instances found with viewpoint learning

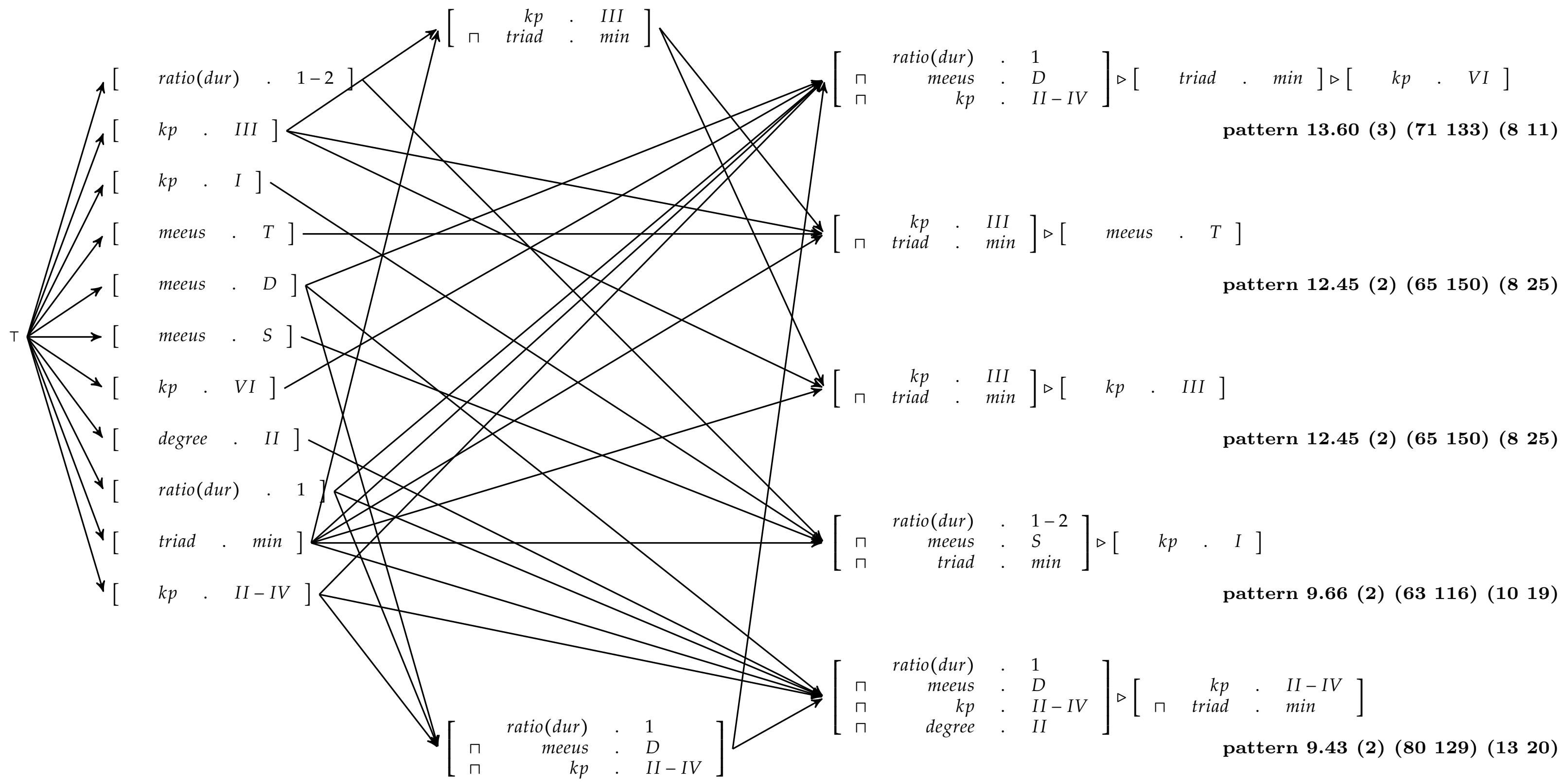

⊤
[ratio(dur) . 1 – 2]
[kp . III]
[kp . I]
[meeus . T]
[meeus . D]
[meeus . S]
[kp . VI]
[degree . II]
[ratio(dur) . 1]
[triad . min]
[kp . II – IV]
[kp . III ⊓ triad . min]
[ratio(dur) . 1 ⊓ meeus . D ⊓ kp . II – IV]
[ratio(dur) . 1 ⊓ meeus . D ⊓ kp . II – IV] ▷ [triad . min] ▷ [kp . VI]
pattern 13.60 (3) (71 133) (8 11)
[kp . III ⊓ triad . min] ▷ [meeus . T]
pattern 12.45 (2) (65 150) (8 25)
[kp . III ⊓ triad . min] ▷ [kp . III]
pattern 12.45 (2) (65 150) (8 25)
[ratio(dur) . 1 – 2 ⊓ meeus . S ⊓ triad . min] ▷ [kp . I]
pattern 9.66 (2) (63 116) (10 19)
[ratio(dur) . 1 ⊓ meeus . D ⊓ kp . II – IV ⊓ degree . II] ▷ [kp . II – IV ⊓ triad . min]
pattern 9.43 (2) (80 129) (13 20)

Figure 14.3: Subsumption of discovered sequence patterns

14.5 Summary

This chapter demonstrated the usage of $\mathcal{SEQ}$ to represent and analyse chord patterns that were discovered from a corpus using viewpoint learning. A DL reasoner can then use such patterns to classify instance data. Further, the patterns can be classified automatically in terms of their subsumption relationships as illustrated for distinctive patterns from Conklin (2010).

15 Harmonic Analysis with Grammars

In this chapter we address the harmonic analysis of jazz chord sequences using grammars with Semantic Web technology. We use the DCG grammar based analysis system by Weyde (1995) as our model and implement two approaches to use this in a Semantic Web context with $\mathcal{MEO}/\mathcal{SEQ}$.

The first approach is a hybrid one, where the analysis is realised with a Prolog DCG grammar and we show how sequences and annotations can then represented as $\mathcal{MEO}/\mathcal{SEQ}$.

The second approach is purely based on web standards, where a grammar is translated into a $\mathcal{SEQ}$ pattern using the method described in chapter 10 and the analysis is realised by classifying data based on these patterns using OWL reasoning. This has the advantage of allowing the grammar to be used directly within a Semantic Web context, which means potentially a greatly simplified application in a distributed data context.

15.1 Hybrid harmonic analysis

This section describes the work published in Weyde and Wissmann (2007). There, a hybrid system was presented that used the DCG grammar form Weyde (1994, 1995) implemented in Prolog for analysing and an OWL ontology for annotating the input sequence with the analytical results.

In this approach, a piece is understood as a sequence of cadences. The cadences contain functional 'ranges' that then refer to individual elements that identify the types of individual chords.

We used sequential concepts to model the upper levels of the analysis hierarchy in an ontology called JazzOWL or short *jowl*. The top-level sequence is `CadenceSequence`, which can contain `Cadence`s that can contain `FunctionRange`s. A sequence is modelled by using `OWLList` as superclass and by restricting the `isFollowedBy` property to the sequence type. Further the content of the sequence is defined by restriction:

$$\begin{aligned}
\texttt{CadenceSequence} &\equiv \alpha \triangleright [\texttt{Cadence}]^{+} \triangleright \omega \\
\texttt{Cadence} &\equiv [\texttt{FunctionRange}]^{+} \\
\texttt{FunctionRange} &\equiv [\texttt{Function}]^{+}
\end{aligned}$$

In Weyde (1995), a *generative grammar* is used to describe the harmonic structure of jazz chord sequences. This grammar is given as a

Prolog Definite Clause Grammar (DCG) that has been extended in three aspects: 1. Additional arithmetic predicates capture and propagate the root note relationships between branches. 2. The application of grammar rules are traced such that a parse tree is constructed. 3. The order of alternative rules reflects a musically preferred analysis. Additional predicates represent the root note, the mode and the distance in fifths from the root in a chain of dominants. A typical analysis of a cadence is shown graphically in figure 15.1.

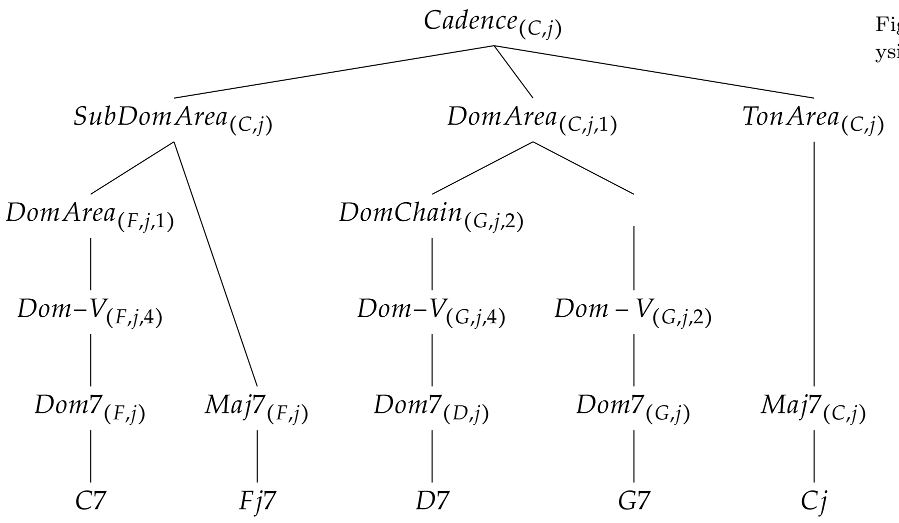


Figure 15.1: Example Harmonic Analysis by a DCG

15.1.1 Implementation

We have implemented the grammar rules and all necessary auxiliary rules in SWI Prolog.[1] A grammar implementation for Definite Clause Grammars is readily available, as with most Prolog systems, and it accepts grammar rules in a format very similar to the one described above. The Prolog inference directly provides a parsing facility. Also there are OWL and RDF packages, which we use to generate the RDF description when a piece is parsed.

[1] Cf. http://www.swi-prolog.org/

A typical rule for harmonic analysis now like this in the SWI/RDF implementation:

```
i_chord(Root,Tree,URI,[T1,T2,T3|Triples])
-->
d7_chord(Root, ChordTag, ChordURI, Triples),
{atom_concat('Blues I',ChordTag,Tree),
 rdf_db:rdf_bnode(URI),
 T1=[URI,'rdf:type','jowl:iim7'],
 T2=[URI,'jowl:root',Root],
 T3=[URI,'jowl:realizedAsChord',ChordURI]
```

Here the i_chord, which is used as part of the tonic range, is realised as a dominant 7th chord, which is typical for Blues style. This information is added to the textual output in Tree. The RDF information is created in T1, T2, and T3, which are all collected in the list passed as the last argument.

This implementation can be used to analyse sequences of Chord Symbols, and it outputs one or more analysis trees. When several analyses are possible the order in which the analysis trees are generated depends on the order of the rules.

A call for performing an analysis on a chord sequence looks like this:

```
analyse(['Dm7','G7','Cj'], Result, Root, Mode, URI, Triples).
```

Prolog then assigns to the variables possible values according to the rules, and *Triples* will contain the RDF description. The triples it produces have the graph structure shown in Figure 2.2.

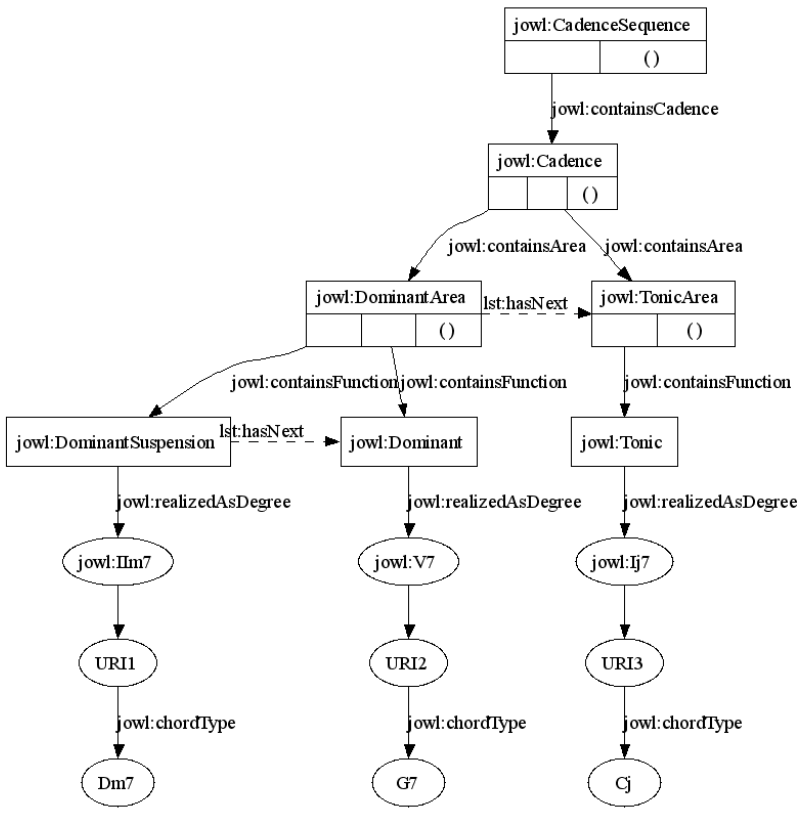


Figure 15.2: Example Annotation of the cadence Dm7, G7, Cj

Prolog can also be used to generate chord symbol sequences. It lists all possible sequences that the grammar allows, which may be a very large number. This is potentially interesting for composers looking for inspiration or for students needing exercise material, but was not attempted to evaluate in this context.

Discussion of JOWL JOWL uses OWL for annotations and the parse tree; however, the grammar itself is not defined in OWL but in the Prolog system. This approach is practical, because Prolog has been used frequently and successfully to implement grammars, while grammar representation OWL entails several issues as described in section 10.

The first issue in the work with grammars is the lack of support for sequential structures in OWL. This, we have addressed with *SEQ* and the OWLList pattern. The OWL ontology could be refined to include more specialised sequence types that model the grammar to some extent. By restricting the *lst:hasContents* and *lst:isFollowedBy* properties, we can restrict the elements of a list. Using this approach, it is also possible to define only parts of a sequence, for example stating that a sequence ends with the Tonic. However, the propagation of the root note from the bottom level (chord symbols) to the top levels (cadence sequence level) would require the use of variables, which are not part of Description Logics generally and OWL DL specifically.

Another, more desirable approach would be to represent the rules using web standards, too. Unfortunately rule languages for the Semantic web are still under development and currently available reasoning engines seem to support only special subsets of the OWL. The most likely candidate, the Semantic Web Rule Language (SWRL) (Horrocks et al., 2003), which combines OWL-DL and OWL-Lite with RuleML, lacks some features that we would require in order to translate our Prolog rules to SWRL: Most notably there is no notion of rule order. Rule order is a crucial element of our approach as we use it to model musically *preferred interpretations*. That is, the topmost rules are the preferred interpretations and are chosen first by Prolog's back-tracking algorithm. It has yet to be investigated whether other solutions such as modelling preference using weights are possible. Further, problems of list representation and reasoning seems to be inherited from OWL and need further investigation.

From a musical point of view, the rules are useful to represent properties of music theory concepts and to allow the analysis of pieces conforming to these concepts. For pieces that do not fit exactly into these concepts, annotations can be made by hand. To automate this process, it would be useful to have approximate matching or different degrees of belonging to a concept.

15.2 Harmonic analysis using $\mathcal{SEQ}$ grammar patterns

Music scholars are often concerned with identifying the properties of music that are typical for certain musical genres, styles or periods. Although human experts are good at recognising patterns, it is desirable to have a method of quantifying the relations between a chord progression and a genre, style or period. In the following, we will use the grammar representation in $\mathcal{SEQ}$ described in section 10 to perform an example of a formalising and testing musicological query and hypotheses.

15.2.1 Musical background

As example, we will here consider the case of the bVII7 chord as it was described by Potter (1989, p. 35):

> "bVII7 [...] is one chord which distinguishes bebop harmony from tonal harmony in Western art music and from pre-bebop jazz harmony as well."

This statement would lead us to expect more occurrences in Bebop pieces than in other styles of music. This kind of statement (chord sequence x is more typical for style s_1 than for style s_2) appears regularly in music harmony texts, but it is hard to quantify. Traditional methods of counting by hand are tedious and error prone. Semantic queries on web corpora can support researchers in gathering quantitative evidence or finding counterexamples. We demonstrate this by using queries on the genre-annotated 9GDB corpus to validate musicological statements made in Potter (1989).

Potter (1989) addresses the role of the bVII7 chord in Bebop jazz. The minor sub-dominant cadence in major keys is a type of cadence that has been debated by several music theorists over time. This is an example of this cadence type:

```
Cj7 Fj7 Fm6 Cj7
Ij  IVj IVm Ij
```

Typically the major sub-dominant is followed by a minor sub-dominant, followed by the tonic. This cadence can be seen as a variation of the subdominant, which just adds some coloration by temporarily moving to minor mode.

On the other hand, the IVm could be seen as a functionally different. The third of the minor sub-dominant, which is normally lead downwards to the fifth of the tonic, has leading note character. Therefore the minor sub-dominant can be seen as a substitute for the dominant. The notes contained in the Sm6 are the same as in a V7/sus4/b9 without root.

However, the following progression is also common:

```
Cj7 Fm7 Bb7/#11 Cj7
Ij7 IVj IVm Ij7
```

Here, the `Fm7 Bb7` can be seen as a typical `IIm V7` pattern, which has normally the function of a dominant. Jungbluth (1981) therefore views the bVII7 as a replacement for the V7. In contrast, Stanton (1982) classifies the bVII7 as a "Sub-Dminant Minor chord". Yet, Potter (1989), considering both options concludes "Ultimately the bVII7 ought to be viewed as something of a subdominant-dominant hybrid."

15.2.2 $\mathcal{SEQ}$ and grammar representation

There are two points made here, that we will address with $\mathcal{SEQ}$ and the grammar facility. The first is that the bVII7 chord is specific to bebop harmony. This is clearly verifiable querying for bVII chords in corpora containing different styles where it should appear mostly in Bebop pieces. We can also query for the related patterns mentioned (IVj-IVm, IVm-bVII7), using have the sequential information provided by $\mathcal{SEQ}$.

The second point is the harmonic function of the bVII7 chord, i.e. its role within the cadence. There is apparent disagreement about whether the bVII7 chord is a dominant, a subdominant or has a role different to both. We argue that by defining the contexts that would correspond to these different hypotheses as $\mathcal{SEQ}$ patterns and counting the matches in a relevant corpus, we can provide evidence for or against these hypothesis, possibly also differentiated by stylistic context.

To implement the approach described above, we have re-implemented a sub-set of the grammar defined by Weyde (1995) with the grammar approach presented earlier in section 10. We have created different variants of the context in which the bVII7 chord occur. These variants have been tested on the Alicante corpus on the subsets of Jazz/Bob, Jazz/Prebop, Jazz/Bossanova and Academic/Baroque to see the frequency of matches of the different grammar variants in the different styles.

The original grammar is recursive and infinite, because it is meant to analyse entire pieces of music, which would cause the problems in the conversion described above. Here, however, we do not need the whole power of the grammar, but we can limit ourselves to a finite sub-grammar, that defines only limited contexts of the bVII7 chord we are interested in. Therefore we do not need to specify limits to the recursive depth.

Here we can reduce the grammar substantially compared to the original, because we do not need to cover the whole sequence with the grammar. Instead we remove all rules from the grammar that are not necessary to match the desired patterns.

15.2.3 Association of the bVII7 chord with the Bebop style

We counted the number of bVII7 chords in the 9GDB sub-corpora for Jazz (Bop, Prebop, Bossanova) (see subsection 13.1.1) and in the Baroque corpus with $\mathcal{SEQ}$ queries. The latter was included as it was remarked by Jungbluth (1981), that the bVII7 chord can also appear as a local dominant to the bIII chord in a circle-of-fifth cadence, which is typical for baroque.

The results in table 15.1 show the query results for the bVII7 chords in section A. The first column shows the total number of pieces in the sub-corpus and the Alpha and Gamma matches are given in absolute and relative numbers to the size of the sub-corpus. Alpha matches indicate the number of pieces containing a match, while Gamma matches indicate the number of occurrences of bVII7 chords. Gamma/Pieces indicates the average number of bVII7 chords per piece, Gamma/Alpha indicates the average number of bVII7 chords in those pieces that contain at least one match.

The distribution of matches over the sub-corpora does not fully reflect the hypothesis that the bVII7 is specific to Bebop (row 'BOP'), as the number of both Alpha and Gamma matches is higher in PREBOP than BOP. The BOSSA sub-corpus shows a lower number of bVII7 chords than the BOP and PREBOP, which conforms to the expectation and the BAROQUE an even lower number of only one piece out of 56 matching.

One possible explanation of the high frequency of the bVII7 in PREBOP could be that it is used as a local dominant to bIII chords. This can be considered to be the case whenever the bVII7 is directly followed by the bIII. However, the query results for the sequence [bVII7] $\triangleright$ [bIII], as presented in section B: show that this sequence

A: [bVII7]	Pieces	Alpha	Alpha/Pieces	Gamma	Gamma/Piece	Gamma/Alpha
BOP	94	35	37.23%	76	0.81	2.17
PRE BOP	178	91	51.12%	225	1.26	2.47
BOSSA	66	17	25.76%	44	0.67	2.59
BAROQUE	56	1	1.79%	4	0.07	4.00
SUM	394	144	36.55%	349	0.89	2.42

B: $[bVII7] \triangleright [bIII]$	Pieces	Alpha	Alpha/Pieces	Gamma	Gamma/Pieces	Gamma/Alpha
BOP	94	8	8.51%	18	0.19	2.25
PRE BOP	178	12	6.74%	21	0.12	1.75
BOSSA	66	5	7.58%	7	0.11	1.40
BAROQUE	56	1	1.79%	3	0.05	3.00
SUM	394	26	6.60%	49	0.12	1.88

C: $[IVm]$	Pieces	Alpha	Alpha/Pieces	Gamma	Gamma/Pieces	Gamma/Alpha
BOP	94	31	32.98%	84	0.89	2.71
PRE BOP	178	121	67.98%	436	2.45	3.60
BOSSA	66	28	42.42%	124	1.88	4.43
BAROQUE	56	31	55.36%	192	3.43	6.19
SUM	394	211	53.55%	836	2.12	3.96

Table 15.1: Frequencies of A: bVII7 chords, B: [bVII7] ▷ [bIII] progressions, and C: IVm chords in 4 sub-corpora of the 9GDB.

does not appear often enough to account for the higher frequency of bVII7 in PREBOP. The BAROQUE sub-corpus, where the circle-of-fifths sequences should lead to an increased number of [bVII7] ▷ [bIII] matches, has only four bVII7 chords, but three out of these four are followed by a bIII chord.

Another possible explanation of the relatively high number of bVII7 chords in PREBOP may be that the chord progressions used in the 9GDB corpus were not those used in the pre-Bebop era, but later reharmonisations, which were especially common in the Bebop period. This hypothesis would require research into the sources of the data, which is beyond the scope of this thesis.

An interesting aspect for comparison are IVm chords, which are seen as closely related and part of similar harmonic patterns by all Jungbluth (1981), Stanton (1982) and Potter (1989). The numbers in section C: of table 15.1 show the frequencies of the IVm chord. Compared to the bVII7 frequencies it is notable, that BEBOP is the only style where bVII7 appears more often than IVm. At least in terms of the bVII7/IVm ratio, the BOP sub-corpus stands out. In BAROQUE corpus on the other hand, the IVm appears in 31 times more pieces in 48 times more instances than the bVII7, supporting the initial hypothesis of this bVII7 being specifically used as local dominant in this style.

To summarise, the results of the one and two chord patterns tested here show that the use of the bVII7 chord is much more common in Jazz than in Baroque in the 9GDB. Also, in Jazz the bVII7 chord is only in a minority of cases followed by the bIII which indicates that is is not normally used as a local dominant. Lastly, the bVII7 appears more often than the IVm chord only in the BOP sub-corpus.

15.2.4 Role of the bVII7 chord: Formalisation

Potter (1989) listed three main hypotheses about the role of the bVII7 chord:

a) the bVII7 chord is a chord with its own specific use

b) the bVII7 chord is a substitute for the sub-dominant

c) the bVII7 chord is a substitute for the dominant

In case a) we assume that there is a specific cadence type that contains a subdominant minor part.

```
Cad -> SMR TR
```

Given rules for realising the TR, we can then formalise our hypothesis by adding the following rule:

```
SMR -> bVII7
```

As to the name suggests, the SMR part can normally also be realised with a IVm chord, but we are not focusing on that chord here.

In case *b)* we expect the bVII7 chord to take the place of a sub-dominant range in a cadence. The one cadence rule with sub-dominant used in Weyde (1995) is

```
Cad ->  SR DR TR
```

This rule corresponds to the classic cadence scheme *SDT* as defined by Riemann (1877). The 'R' is for 'range' (in the German text it is B for 'Bereich') and indicates that SR, DR and TR do not represent as specific chord but may be realised with different chords or chord sequences, base on the rules of the grammar.

Given rules to produce chords for *TR* and *DR*, we can formulate our hypothesis by using the following rule in our grammar for the *SR*:

```
SR -> bVII7
```

With this rule, the Cad pattern will only match when the bVII7 is followed by a dominant range and a tonic range, therefore providing support for hypothesis b). This pattern will not match, when the bVII7 is followed directly by the tonic, as our rules do not allow the DR to be replaced by an empty sequence (which is also not allowed for a context-free grammar).

In hypothesis c) the bVII7 chord is a substitute for the dominant seventh chord, that means it can appear in place of a dominant seventh chord. The only cadence type we use in this grammar is

```
Cad ->  DR TR
```

In a very simple case, a D7 T cadence could then also be realised as bVII7 I. A pattern modelling only this situation would match the same sequences as hypothesis a). However, in Jazz it is very common to use local dominants and circle-of-fifths movements, especially with IIm-V7 patterns relative to their following chords. Therefore, we can replace other dominant seventh chords, e.g. in a cadence like this:

```
Cad -> DR(V) DR TR
```

This rule contains a dominant range relative to the V degree, indicated by the brackets. The simplest realisation would be V7(V) V7 Ij or — without the relative notation — the progression II7 V7 Ij, where in chord symbols of the form *j the 'j' indicates a major chord. With rules

```
DB -> D7
D7 -> bVII7
```

we can now generate a cadence bVII7(V) V7 Ij or IV7 V7 Ij respectively.

This grammar matches more sequences compared to hypothesis a), depending on whether such chord progressions are actually present in the corpus. In other words, only a higher number of matches for grammar c) would provide support for this being a good representation of the style. One could argue that it is not problematic if the pattern can match more sequences than actually occur in a style, as long as it matches those that do occur. However, in that case that pattern would not be as useful for distinguishing the style from others, e.g. in an music information retrieval task; cf. also to the concept of distinctive patterns by (Conklin, 2008, 2010), as discussed in section 14.3.

The original grammar allowed to match the same cadence several times in some cases, by including different sections of the left context. e.g. Dm G7 Cj could match as V7 I starting at the G7 or as IIm V7 Ij starting at the Dm7. We developed an algorithm that avoided such multiple counts. However it turned out that running the queries on the 4 sub-corpora took already several hours, and this time was substantially increased by the single counting algorithm. Thus the task became impractical. The solution used here was to modify the rules such that the bVII7 would only appear as the left-most chord in any sequence. This was possible, because the reduced grammar we used did not actually restrict the left context of the bVII7 chords.

15.2.5 Role of the bVII7 chord: Results

The results for the different corpora are given in table 15.2. Variant A allows only the Ij chord as tonic, while Variant B also allows the mediants bIII and VI. The results show that grammar b) (bVII7 as S) has fewer matches than a), while c) has exactly the same number of matches as a). The difference becomes greater, when we add the tonic mediants to the grammar. However it is not clear, whether this is relevant, because the bVII7 could be interpreted as a local dominant to both the bIII chord and the VI chord rather interpreting the mediants as substitutes for the Ij. In either case, the results indicates that the use of the bVII7 before the tonic is more common, than before the dominant which would indicate a subdominant character of the bVII7. However, grammar c) which would allow the bVII7 to be used as a general replacement for a dominant 7th chord, does not match any more instances, and is therefore overly general for the corpora we are studying here.

As an additional test we tested grammar b) variant A with the restriction to have IVm precede bVII7. A sequence that matches this description indicates a subdominant minor cadence, because the IVm as a degree *IV* chord is a more typical sub-dominant chord than the bVII7. The results are shown in table 15.3. When comparing the results to the unrestricted grammar, we see that the restriction reduces the matches for grammar b). The bigram [bVII7] $\triangleright$ [bIII] match numbers, given for comparison in the table, show that this progression does occur frequently, but evidently not with a *DRTR* following, as required by grammar b). Thus there is no evidence from this experiment, that the bVII7 fulfils a subdominant role.

Further we have tested grammar c) with the restriction that bVII7 has to be preceded by IIm. If this pattern would match frequently, it could be seen as evidence for the bVII7 taking the role of a dominant seventh chord, as IIm7 V7 is a very frequent realisation of a dominant range (DR). The match numbers shown in table 15.4 show however very few matches for this pattern. This is also the case for the bigram pattern, the match counts for which are shown in the same table. These low numbers indicate that the assumption of grammar c) that the bVII7 can in general replace a V7 does not seem to apply, at least on the basis of these sub-corpora of the 9GDB.

In summary, the evidence from the queries presented here, using grammars, bigrams and individual chords supports the view given by Potter (1989) that the bVII7 does not fulfil the role of a subdominant or a dominant. For it to be a sub-dominant substitute, we would expect it to appear before the dominant often. This is not the case, especially not in combination with the IVm chord which would provide additional support for the sub-dominant view. On the other hand, as a general substitute for the V7 dominant chord the bVII7 would have to appear as a replacement for local dominants, which happens only 3 times in the whole corpus.

Thus the representation of the bVII7 as a subdominant would not match the majority of the bVII7 occurrences and the representation as a general dominant substitute (like the bII7 for instance) would be too general allowing sequences that do not appear in the corpus. Therefore, a representation of the bVII7 as part of a special type of progression, as suggested by Potter (1989), seems to be a good way of including this specific chord in the theory of Jazz harmony.

	Grammar a)	bVII7 special	Grammar b)	bVII7 as S	Grammar c)	bVII7 as D	
Variant A) T only	Gamma	Alpha	Gamma	Alpha	Gamma	Alpha	Total
BOP	14	11	7	4	14	11	94
PREBOP	93	45	36	16	96	46	178
BOSSA	5	3	0	0	5	3	66
BAROQUE	0	0	0	0	0	0	56
ALL	112	59	43	20	115	60	394
Relative	gamma	alpha	gamma	alpha	gamma	alpha	
BOP	14.89%	11.70%	7.45%	4.26%	14.89%	11.70%	
PREBOP	52.25%	25.28%	20.22%	8.99%	53.93%	25.84%	
BOSSA	7.58%	4.55%	0.00%	0.00%	7.58%	4.55%	
BAROQUE	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	
ALL	28.43%	14.97%	10.91%	5.08%	29.19%	15.23%	

Variant B) T + mediants	Gamma	Alpha	Gamma	Alpha	Gamma	Alpha	Total
BOP	27	18	7	4	27	18	94
PREBOP	137	64	37	16	140	65	178
BOSSA	16	9	0	0	16	9	66
BAROQUE	0	0	0	0	0	0	56
ALL	180	91	44	20	183	92	394
Relative							
BOP	0.29	19.15%	0.07	4.26%	0.29	19.15%	
PREBOP	0.77	35.96%	0.21	8.99%	0.79	36.52%	
BOSSA	0.24	13.64%	0.00	0.00%	0.24	13.64%	
BAROQUE	0.00	0.00%	0.00	0.00%	0.00	0.00%	
ALL	0.46	23.10%	0.11	5.08%	0.46	23.35%	

Table 15.2: Matching frequencies of grammars a) b) c) variants A and B in 4 sub-corpora of the 9GDB.

IVm bVII7	grammar		bigram	
	gamma	alpha	gamma	alpha
BOP	0.04	2.13%	0.38	19.15%
PREBOP	0.07	3.37%	0.56	24.16%
BOSSA	0.00	0.00%	0.17	10.61%
BAROQUE	0.00	0.00%	0.04	1.79%
SUM	0.04	2.03%	0.38	17.51%

Table 15.3: Matching frequencies of grammars a) b) c) restricted to IVm preceding bVII7 in 4 sub-corpora of the 9GDB.

IIm-bVII7	grammar		bigram	
	gamma	alpha	gamma	alpha
BOP	0.00	0.00%	0.03	1.06%
PREBOP	0.10	5.06%	0.12	6.18%
BOSSA	0.09	6.06%	0.11	6.06%
BAROQUE	0.00	0.00%	0.00	0.00%
SUM	0.06	3.30%	0.08	4.06%

Table 15.4: Matching frequencies of grammars c) restricted to IIm preceding bVII7 in 4 sub-corpora of the 9GDB.

15.3 Summary

This chapter demonstrated the use of $\mathcal{MEO}/\mathcal{SEQ}$ in the context of harmonic analysis of jazz chord sequences.

Harmonic rules were encoded in a grammar. First, a hybrid approach was taken were the analysis was performed using a Prolog engine and input and output were represented in $\mathcal{MEO}/\mathcal{SEQ}$. This approach enables to perform the analysis on chord sequences from the web, their annotation with harmonic information and deployment of the results on the web. Second, a purely web standards based approach was taken where the grammar was translated to a $\mathcal{SEQ}$ pattern using the algorithm described in section 10. An OWL reasoner then automatically performed the analysis by classifying the data according to the pattern. This approach potentially simplifies applications in a distributed data context.

The scenario showed how musicologists can use $\mathcal{MEO}/\mathcal{SEQ}$ to empirically test hypotheses on a corpus. The analysis of the harmonic meaning of the bVII7 chord Potter (1989, p. 35) served as an example. It was shown that Potter's different hypotheses could be encoded in $\mathcal{MEO}/\mathcal{SEQ}$ patterns and that OWL classification can be used to classify chord sequences. The results support the hypothesis that a bVII7 chord has a specific harmonic function that does not fit into the roles of dominants or subdominant. Potter originally performed his analysis manually on a small set of example sequences. A benefit of our technique is the automation of the musicological task. Additionally, the technique can help to enhance the methodology of musical experiments as all relevant information are explicitly encoded in the ontologies. Other musicologists can therefore easier understand the experimental setting and are provided with the means to reproduce the experiments identically or with their own modifications. Further, it is easier to extend the results to more corpora that are provided on the web as RDF data.

16 Federated queries

The original idea of Semantic Web as a concept is to allow the distributed interaction of agents, humans or software, in the world wide web, based on common standards of represented knowledge (Berners-Lee, 1999; Berners-Lee et al., 2001). The Semantic Web technologies provide these common standards to express this knowledge and form the Web of Linked Data, a term more recently coined (cf. Bizer et al., 2009). This approach is also extended to music in recent years, and in this chapter we will present a small application in this spirit.

The possibilities of the Semantic Web are now being used in the realm of musical metadata, e.g. the information from MusicBrainz is now being made available as Linked Data, similar to DBpedia, which makes information from Wikipedia available as an RDF data source. These projects are relatively new and the quantity and quality of the data is yet an obstacle to practical applications. However, it is possible to combine information from these sites to enable musical research since they both use the Music Ontology (cf. section 6.1).

In corpus-bases music research that uses computers, queries and analyses on different types of data usually have to be formulated in different languages and code has to be written to combine the information. E.g. a popular way to express structural patterns is the use of regular expressions in Humdrum while metadata are often stored in relational databases. Using SPARQL on RDF data enables expressing the query in one formalism without the necessity for writing bridging code. In this context $\mathcal{SEQ}$ add the facility of representing and reasoning on sequential structures.

In the following we will demonstrate how federated queries on a corpus of chord sequences and musical metadata on the web can be used to realise musicological queries. More specifically, we demonstrate this by relating $\mathcal{MEO}/\mathcal{SEQ}$ patterns for harmonic progressions in Beatles songs to metadata available from DBpedia.

16.1 Experimental setup

In this experiment we are studying the frequencies of chord progressions in songs by the Beatles and relate them to Metadata from Wikipedia, specifically the author, the album and the year of publication. We would like to see whether there is a congruence with concepts described by musicologists such as Villinger (2006) regarding the music of the Beatles. We chose the Beatles songs as the chord sequences and metadata already exist as RDF on the web, in these datasets:

– *Isophonics* as corpus of Beatles chord sequences (cp. subsection 13.2.1).

– *DBpedia* as a source of metadata (cp. section 2.4).

In order to support SPARQL queries over the sequential structure with $\mathcal{MEO}/\mathcal{SEQ}$, we additionally created an ontology that contains musical patterns and two binding ontologies:

– *Beatles-DBpedia links* that describe the correspondences between song identifiers in Isophonics and DBpedia generated by a string matching as described later in this section.

– *MO2SEQ Binding* as a semantic bridge between the sequential structure in Music Ontology and $\mathcal{SEQ}$.

An OWL Reasoner was connected to compute the necessary inferences. Figure 16.1 gives a schematic overview of the query infrastructure and data sets used in our test.

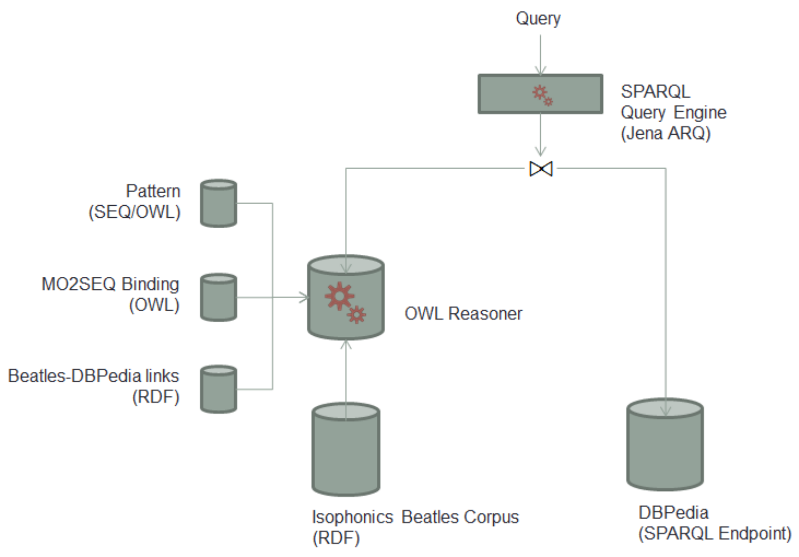

Query
SPARQL Query Engine (Jena ARQ)
Pattern (SEQ/OWL)
MO2SEQ Binding (OWL)
Beatles-DBPedia links (RDF)
OWL Reasoner
Isophonics Beatles Corpus (RDF)
DBPedia (SPARQL Endpoint)

Figure 16.1: Schematic drawing of the component involved in federated query when querying for harmonic patterns on the Beatles corpus with DBpedia metadata

16.1.1 Harmonic Patterns

For this experiment we used a set of harmonic patterns, depicted in Table 16.1. These patterns describe a sequence of root movements as intervals of semitones. We use interval classes $mod12$, such that -5 means the same interval class as 7 and matches the same elements.

The patterns a), b), c) describe standard cadences from music theory, the standard cadence IV V I, the plagal cadence V IV I and the mixolydian cadence IV VII I. These patterns are not as specific as the cadences, as we only specify the relative root movement, and can not guarantee that the last chord is actually a I, and the VII chord is in major.

The other three patterns are motivated by (Mauch et al., 2007), who have counted 4-chord patterns of chord mode and root movement. In their list of the most frequent patterns, only fourth and fifth movements (+5,+7) occur on the first 6 ranks. The patterns ranking 7–9 on their list contain a mixture of fourth/fifth and second/seventh (+2,+10) movements. We also added a shorter pattern containing three chords with two fifth/fourth root movements for comparison.

	Progression Type	Pattern
a)	Standard Progression	$\underset{+2}{\triangleright}\;\underset{+5}{\triangleright}$
b)	Plagal Progression	$\underset{+10}{\triangleright}\;\underset{+7}{\triangleright}$
c)	Mixolydian Progression	$\underset{+5}{\triangleright}\;\underset{+2}{\triangleright}$
d)	3-Chord Circle Movement	$\underset{\{+5 \mid +7\}}{\triangleright}\;\underset{\{+5 \mid +7\}}{\triangleright}$
e)	4-Chord Circle Movement	$\underset{\{+5 \mid +7\}}{\triangleright^{3}}$
f)	4-Chord Mixed Circle/Second Movement	$\underset{\{+10 \mid +2 \mid -5 \mid +7\}}{\triangleright^{3}} \sqcap \underset{\{+5 \mid +7\}^{\geq 1}}{\triangleright^{3}} \sqcap \underset{\{+10 \mid +2\}^{\geq 1}}{\triangleright^{3}}$

Table 16.1: Root movement patterns

16.1.2 Query

The patterns described above were encoded in SPARQL and then used in a federated SPARQL query as shown in table Table 16.2. These queries were executed using the local query engine Jena ARQ[1], but they could even be used on a web based interface[2] if our binding ontologies and reasoning services were available at a SPARQL endpoint.

[1] http://jena.sourceforge.net/ARQ/

[2] e.g. http://www.sparql.org/query.html

There are several ways to execute SPARQL queries and setup supporting infrastructure. The SERVICE extension of Jena adds to SPARQL the ability to send subqueries to multiple SPARQL endpoints and combine the results. This query uses the patterns and data we provided in FROM URI, and and specifies the as SERVICE the DBpedia SPARQL endpoint. Because a given SPARQL endpoint may be an interface to a triple store or a relational data store, the ability to query several endpoints with one query is very useful in terms of interoperability.

The result of the query is a table, similar to an SQL table, where every row is contains a possible assignment of values to the variable. The data are then aggregated over the year, artist or album dimensions to

16.1.3 Data quality issues

The results of the query presented above rely fundamentally on the ability to identify entities across the datasets and in the real world. This has posed some difficulty for this experiment, in particular with the data from DBpedia. DBpedia data are mostly automatically extracted from Wikipedia, the data is not always complete, correct or consistent. This creates problems when linking datasets, because the same object is not necessarily referred to in the same way. Also, the

```
SELECT ?sequence ?pattern ?year ?artist ?album
FROM <http://www.j3w7.de/beatles-binding.rdf>
WHERE
{
        ?sequence a seq:Sequence ;
                  a ?pattern.

        SERVICE <http://dbpedia.org/sparql>
        {
                ?dbpEntry dbpo:releaseDate ?year   ;
                          dbpp:writer      ?artist ;
                          dbpo:album       ?album.
        }
}
```

Table 16.2: Federated query in SPARQL combining harmonic patterns in the Beatles corpus and metadata in DBPedia.

identification of real world entities, e.g. the composers of songs, is not always clearly possible as there are errors and gaps introduced by the extraction process and – to a lesser degree – in the human-readable pages.

In order to create combined views we have to establish "links" between the datasets. In our case both datasets have piece identifiers and we can state the identity of piece identifiers using the *owl:sameAs* property, e.g.

iso:YellowSubmarine owl:sameAs dbp:Yellow_Submarine_(song) .

The identifiers in our data are strings containing the song titles, which are often spelt differently for the same piece. Therefore we use string similarity metrics to find the correspondences. We used Levenshtein edit distance ignoring differences such as cases, whitespaces and artifacts such as "_(song)".

DBpedia contains only information about Beatles pieces for which Wikipedia users created entries which were processed by the DBpedia extraction mechanisms. At the time of the experiments this were 97 pieces, which were matched against 163 pieces in the Isophonics Beatles corpus, and our string matching approach found 94 correspondences.

Another problematic area are the artist, composer, and album fields in the DBpedia data. Here it seems that the extraction process had particular difficulty with the fact that most songs by the Beatles were written John Lennon and Paul McCartney, commonly referred to as Lennon/McCartney. However, in the DBpedia data, the name Lennon rarely appears. Similarly, the album reference is missing from a significant number of songs. We have not taken any steps to address these issues here, but rather try to interpret the results. This was deemed sufficient, as the main aim of this experiment it to assess the feasibility of this technological approach. To acquire reliable musicological knowledge, there would clearly be a need for a thorough investigation of these issues.

16.1.4 Reasoning Support

The purpose of reasoning in our scenario is 1. to classify Beatles sequences according to the harmonic patterns and 2. to relate the vocabulary of the Beatles corpus with the $\mathcal{SEQ}$ vocabulary.

In our setup the reasoner Pellet was connected to the local SPARQL query engine Jena ARQ. Pellet can be easily connected as it implements the Jena interfaces for RDF graphs and exposes the internal OWL axiom representation of reasoning results as triples to Jena.

Other setups for combining reasoning and SPARQL queries are possible. For example, reasoning results could be precomputed and provided as new data source. This is called materialisation. Normally only parts of the knowledge base are materialised as DLs allow expressing concepts with infinite models such as the recursive grammars patterns we discussed in the previous chapter.

In order to retrieve pattern instances from the data, the data and the pattern ontology must have a common vocabulary. The common vocabulary can explicitly reuse data vocabulary in the pattern vocabulary but it is also possible to relate both by giving logical descriptions of the data vocabulary can be interpreted in terms of the pattern vocabulary. In our example, the sequential structure of the Isophonics Beatles Corpus is expressed using OMRAS2 while our patterns are expressed in $\mathcal{SEQ}$. Thus we must provide additional axioms that describe their relationships as has been described in section 13.2.2.

16.2 Results

The results of the federated queries grouped by year, artist and album are shown in figures 16.2–16.4. The frequencies shown are the average number of matches per piece, including pieces with 0 matches (the total number of matches divided by the piece count). As mentioned above, the results are incomplete, but we assume the relative frequencies found still broadly representative of the complete data. Again, for reliable musicological results the dataset would have to be investigated thoroughly to check this assumption.

In figure 16.2 we see that the patterns e) and f) appear frequently, but with substantial variation over the years. Pattern f) is most frequent in years 1963, 1964 and 1967, and in 1965 f) is second but very close to e). This agrees with the observations by (Mauch et al., 2007). After the peak of f) pattern at 1964 and the e) pattern at 1965, with 18 resp. 10.5 matches per piece on average, both patterns' match frequencies decrease steadily to a level of below 3 in the years 1969 and 1970.

This very roughly agrees with the frequencies of experimental features as reported by **?** or with the middle phase of the Beatles' work as defined by Villinger (2006). However, both Eerola and Villinger (and several other authors) see a change of style in 1965, but these frequent patterns already appear in 1964. This may mean that specific harmonic progressions have changed which are not distinguished by these patterns, or that other factors play a role, such as melody, rhythm or instrumentation.

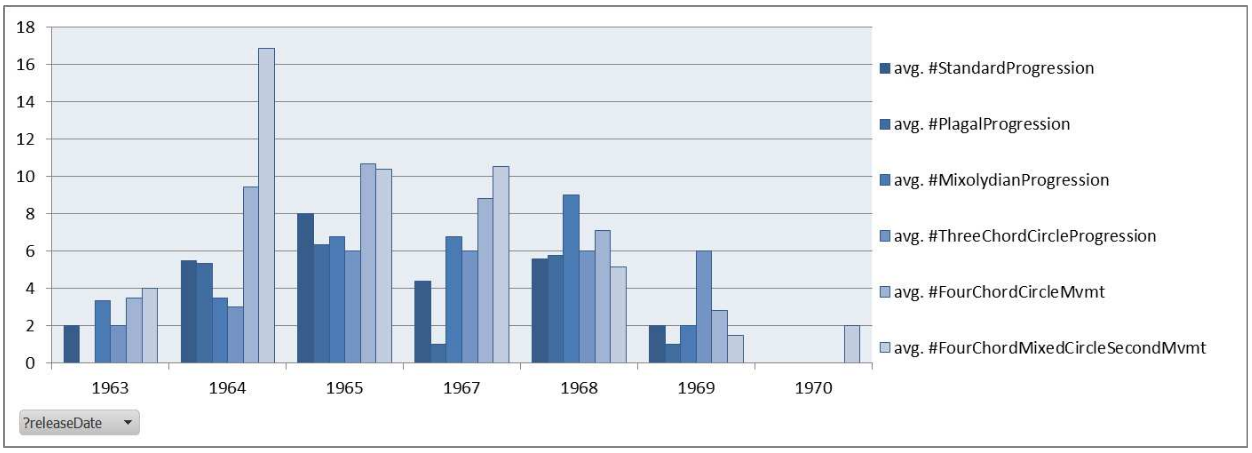


Figure 16.2: Average pattern count by year

The pattern matches aggregated by the artist attribute are shown in figure 16.3. As discussed above, the Lennon/McCarteny songs are attributed as McCartney. The pattern distribution of the Lennon/McCartney pieces and those by George Harrison is notably similar, while very different from the other artists including Ringo Starr. However the piece count is 2 or lower for the other artists.

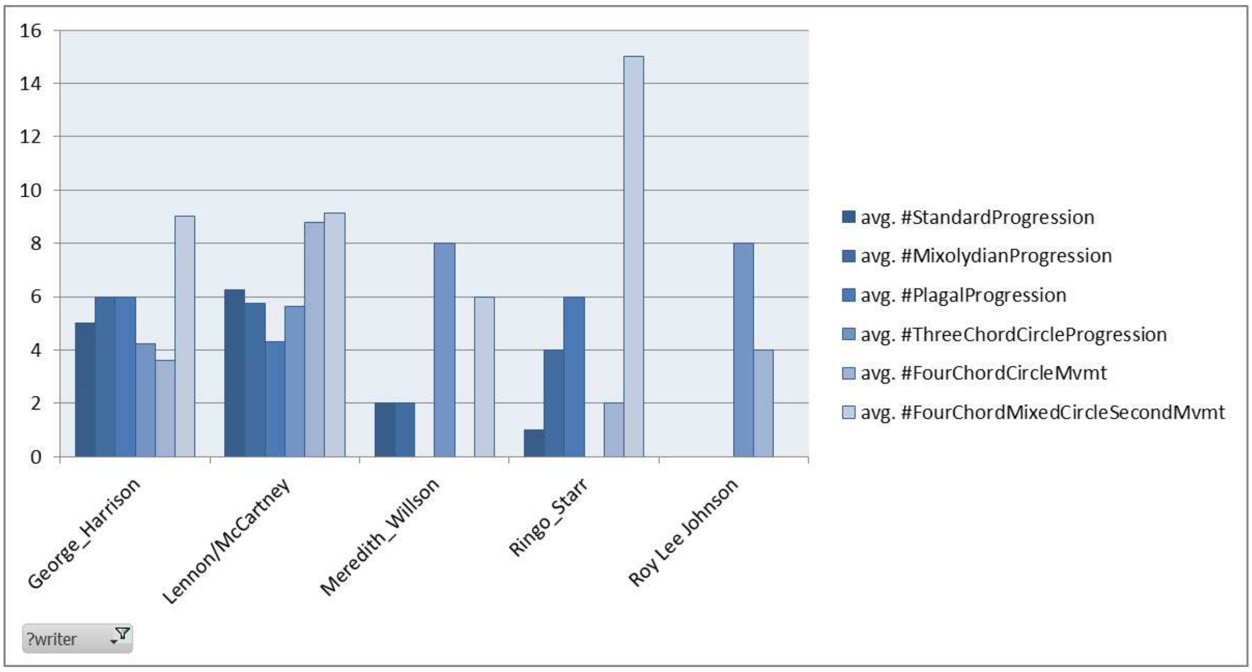


Figure 16.3: Average pattern count by artist

Figure 16.4 shows the aggregation by album. The distribution of pattern matches is similar to the distribution by year.

16.3 Summary

Semantic Web techniques enable federated queries that combine different information sources and can use reasoning to infer additional

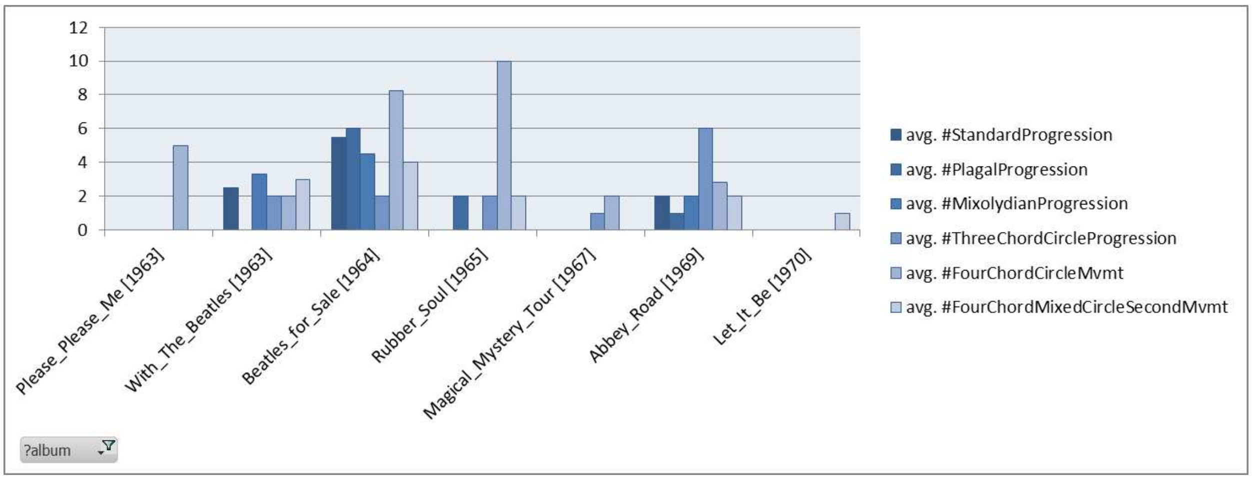


Figure 16.4: Average pattern count by album

information. We showed how this can be interesting for music research by performing federated queries on a Beatles corpus and DBpedia. In our scenario, $\mathcal{MEO}/\mathcal{SEQ}$ provided means to express harmonic patterns and ask for their occurrence in the corpus. This information was related to metadata in DBPedia such as publication year and writer. Notably, a user can formulate a musical query in one step using a single formalism whereas traditionally corpus-bases music research would require posing several queries on different types of data usually involving different languages and in some cases coding to combine the information.

Part IV

Conclusion

17 Summary and Discussion

This work has investigated musical knowledge representation, queries and reasoning on the Semantic Web focusing on chord sequence patterns in OWL. The aim is to improve the support for music in the Semantic Web by providing representations for sequential structures and for musical entities.

To enable querying and reasoning on music in the Semantic Web, appropriate ontologies with suitable logical structures have been defined. The main development is the $\mathcal{MEO}/\mathcal{SEQ}$ ontologies and modelling patterns for musical entities and sequences. These have been implemented in OWL and their properties and modelling capabilities have been investigated. An implementation with a Java API and a set of tools, including parsers and converters, have been developed. The modelling patterns and the implementations have been tested with regards to running times and they were evaluated for several practical application scenarios in computational musicology.

The review of existing representations for music in the Semantic Web (Objective 1) showed that these representations are limited in their capability to model structural patterns and in regards to their use of Semantic Web reasoning facilities.

Ontological constructs were developed that allow musicologists to express chord sequence patterns (Objective 2). The constructs were divided in two modular ontologies ($\mathcal{MEO}$ and $\mathcal{SEQ}$) in order to allow a separation of concerns between musical modelling and sequential modelling. $\mathcal{SEQ}$ can potentially be of benefit in other knowledge domains for representing sequential data on the Semantic Web such as bioinformatics (genome data, protein sequences) and can be more easily reused this way.

The $\mathcal{MEO}$ ontology (chapter 7) provides a representation that addresses this issue for musical entities (Objective 2a), in particular it supports subsumption of chords based on the notes contained in the chord rather than the chord symbol, including enharmonic equivalence. This is a useful feature for harmonic analysis, where subsumption can thus identify harmonically equivalent chords.

The $\mathcal{SEQ}$ ontology (chapter 8) provides a means of representing sequential structures (Objective 2b). These extend the current possibilities of OWL. $\mathcal{SEQ}$ is based on a the central $\triangleright$ operator, which is used to realise a linked list approach and enables the definition of OWL-DL concepts in a notation similar to regular expressions, e.g. general succession ($\cdots$), repetitions ($C^{+}, C^{*},$) also with number restrictions ($C^{\geq n}$) and other operators commonly used in regular expressions. Constructs to refer to sequences as a whole or point in the sequence (α and γ) were defined. Such references can be used to control pattern matching behavior, which is for example useful in statistical queries to retrieve piece counts and absolute counts for pattern matches. Further, they can be used to annotate sequences with RDF metadata (cf. 2d).

Technically, concatenation ($C \overset{\triangleright}{+\!\!+} D$) turned out to be harder to define. A direct representation in OWL DL is not possible due to lack of means to refer to the end of a defined concept. To still realize concatenation an external rewriting approach was taken and an algorithm devised that structurally rewrites two given concepts (section 8.11) into a new one. The concatenation is useful in particular for the realisation of the grammar facility.

To support the expression and automatic analysis of harmonic structures (Objective 2c) we were especially interested in context-free grammars as these are a well-established tool in this domain of music informatics. Although context-free grammars are not fully translatable to OWL DL as its expressiveness is limited to preserve decidability. To overcome this problem, we developed two approaches to realize harmonic analysis in a Semantic Web context (cf. 10). First, a hybrid approach was pursued, where the analysis was performed using a Prolog engine and input and output were represented in $\mathcal{MEO}/\mathcal{SEQ}$. This approach enabled retrieving chord sequences from the web, annotating them with harmonic information and making the result available on the Semantic Web. Second, in order to perform the automatic analysis only with standard reasoning mechanisms of the Semantic Web, a rewriting algorithm was developed that partially translates a grammar into $\mathcal{SEQ}$ patterns. An OWL DL reasoner then automatically performed the analysis by classifying the data according to the pattern. This approach potentially simplifies applications in a distributed data context. We characterised the translatable fragments of CFGs and showed that musically interesting harmonic analyses can be expressed and performed.

To support software applications (Objective 2e), the $\mathcal{SEQ}/\mathcal{MEO}$ API has been implemented as an extension to the OWL API. This provides an object model and an interface for working with a ASCII text representation of $\mathcal{SEQ}$. The grammar transformation has also been implemented in Java using the $\mathcal{SEQ}/\mathcal{MEO}$ API. This API uses the newly developed approach of Axiomatisation on Demand, which increases the flexibility compared to predefined ontology subsets and also makes the reasoning more efficient by ensuring that only the actually needed parts of the ontology are loaded.

The developed OWL ontologies and Java implementations have been studied for their runtime complexity theoretically and by profiling. The profiling confirmed the expectation that the current reasoner implementations do not take advantage of the sequential structure encoded in $\mathcal{SEQ}$. The running times depend only on the size of the knowledge base, even when an iteration over the list structure would yield much lower complexity.

For the evaluation corpora of chord sequence were prepared in $\mathcal{MEO}/\mathcal{SEQ}$. Apart from the practical purpose of having data for musical experiments the preparation methods show also how the developed ontologies can facilitate structural and semantic integration (Objective 3). For the 9GDB set provided by the University of Alicante we developed a parser and an algorithm to create the RDF representation based on $\mathcal{SEQ}$ and $\mathcal{MEO}$. For the Beatles and Real Book datasets provided by Queen Mary University of London a binding ontology was developed and some post processing applied. In this context $\mathcal{SEQ}$ could already be used to provide extended reasoning capabilities for use with the patterns found by Mauch et al. (2007) and Anglade and Dixon (2008).

The application of $\mathcal{SEQ}$ and $\mathcal{MEO}$ was tested with experiments in three application scenarios.

The first experiment was the modelling of the subsumption hierarchy of patterns learned from the 9GDB corpus using a pattern discovery approach by Conklin (2010), which provided interesting insights into their structure.

The second experiment was the modelling of a grammar for Jazz harmony defined by Weyde (1995) (Objective 4). The first sub-scenario was to analyse chord sequences in a hybrid implementation Weyde and Wissmann (2007). This proved to be a potentially useful tool for automatically adding music-analytical metadata to chord sequences. The second sub-scenario was the use of a reduced and customised grammar to investigate a music theoretical problem, where Potter (1989) had formulated three hypotheses. These results provided evidence for one of Potter's hypotheses but against the other two. A practical challenge we observed in this experiment is that the grammar facility produced large query patterns, so that the computations were very time-intensive.

The last experiment explored the use of federated queries on Semantic Web data, combining the Beatles dataset with metadata provided by DBpedia (Objective 5). It was demonstrated that multiple data sources can be combined easily. Moreover, the possibility to combine structural queries and metadata within the same formalism provides musicologist to test their hypotheses. Although the technology was shown to work, the reliability of the musicological results depends on the quality of datasets. Problems with DBpedia were observed, which made a tolerant matching approach necessary to link the datasets. A thorough checking and synchronising of the datasets would be required.

In summary, the main results are

- $\mathcal{SEQ}$ and $\mathcal{MEO}$ provide a sequence representation and structured musical entity representation for OWL-DL (Objectives 2a–2b)
- $\mathcal{SEQ}$ and $\mathcal{MEO}$ make reasoning over sequential structures and chord structures possible (Objective 2b)
- $\mathcal{SEQ}$ and $\mathcal{MEO}$ APIs for Java and text interfaces have been developed (Objective 2e)

- a grammar transformation for $\mathcal{SEQ}$ and $\mathcal{MEO}$ has been developed that supports limited context-free grammars (Objective 2c)
- $\mathcal{SEQ}$ and $\mathcal{MEO}$ have been demonstrated to provide useful representations and reasoning in a musicological context (Objectives 2c and 4)
- $\mathcal{SEQ}$ and $\mathcal{MEO}$ have been demonstrated to support Linked Data applications with federated queries (Objective 2d and 5), with the quality of public datasets being the yet main obstacle to practical applications

The results demonstrate that $\mathcal{SEQ}$ and $\mathcal{MEO}$ can enable representation and reasoning on chord sequences in the Semantic Web. By defining ontologies and design patterns together with algorithms, effective support for sequential structures in general and musical chord progressions in particular has been generated. This creates the exciting perspective of musical search and research that can reach resources from all over the world with a single query in the Semantic Web. We have demonstrated that this is possible with the techniques provided here, but there are several issues in practice that remain to be addressed to make their use more feasible for everyday applications.

On the technical level, the reasoning engines available for OWL have no way of making use of the sequential information in the knowledge base, which has the potential to greatly speed up the computation on $\mathcal{SEQ}$. However, to achieve this, it would be necessary to build specially optimised reasoners. To make such a feature generally available, a standardisation of sequential structures is needed, which could also be interesting for other domains than music.

On the application level, the main problem we encountered is the quantity and quality of available data. Transforming existing machine-readable corpora such as the 9GDB and the Beatles corpus used in this thesis is feasible, but requires the development of conversion algorithms. However, the conversion of data created for human consumption, such as he DBpedia extraction from Wikipedia data, has proven problematic, at least in the experiment shown here due to errors and inconsistencies in the data. Also, the idea of linking data requires the coordination of names and identifiers, which are needed to relate data from different sources. Some development towards improved data quality, quantity and compatibility is currently taking place, e.g. in the LinkedBrainz project (`http://linkedbrainz.c4dmpresents.org/`) or the SALAMI project (De Roure et al., 2011). Given the ongoing activities in this area, it does seem possible that this goal could be reached and distributed queries with $\mathcal{SEQ}$ and $\mathcal{MEO}$ could become a practical tool in the not too distant future.

In order to answer the initial research question theoretical and practical issues from the disciplines of Semantic Web knowledge representation and music informatics have been used to derive insights into the possibilities of OWL to represent musical structures and to draw inferences that are of interest for musicologists. We showed that chord sequences and chord sequences patterns can successively be represented in OWL ontologies. Using these we were able to perform typical musicological tasks with standard reasoning and query facilities on the Semantic Web. The extension of the approach to more complex musical structures such as full score representation likely requires changes and extensions of the Web Ontology Language itself and is left for future exploration.

This work contributes to knowledge in the areas of Music Informatics and Semantic Web by connecting results of the two fields and by highlighting the advantages and limitations of this marriage. Explicit formal knowledge models with decidable reasoning procedures have not been applied to perform musicological tasks in Music Informatics before. They can enhance transparency and reproducibility of the musicological experiments. Further, the novel means of expressing and analysing sequential and grammatical patterns with OWL are a general contribution to knowledge engineering methodology on the Semantic Web.

18 Future Work and Reflections

Directions for Future Work

The work presented in this thesis has opened up questions and tasks for further research and development. As already mentioned, the development of optimised reasoners for $\mathcal{SEQ}$ and the creation of more compatible quality data sources in Semantic Web representations would yield some immediate practical benefit. Some other approaches, which are discussed in the following, could lead to new directions of development of the Semantic Web for music.

Music Generation. We focused on the use of patterns for classification and retrieval as these are the task currently supported by Semantic Web reasoning engines. Patterns describe a space of possible instances and we can describe *music generation* in this context as generation of pattern instances, e.g. of a chord sequence pattern. Formally, this task is know as *model generation.* Model generation is a research area and application which deals with debugging specifications, i.e. in order to find instances that were not intended by the modeller.

Being able to listen to instances of defined patterns could provide guiding for further developing music ontologies. E.g. one might want the subsumption behaviour of chords to directly reflect audible impressions in the sense that if a chord is subsumed by another it is also perceived as a specialisation that for example stays within the same mode but adds more colour. A technical starting point could be the work of Wang et al. (2006) and Garcia et al. (2007) who both translated OWL ontologies to a representation of the FOL model finding tool *Alloy Analyzer*[1] that can then generate instances.

[1] http://alloy.mit.edu/alloy4/

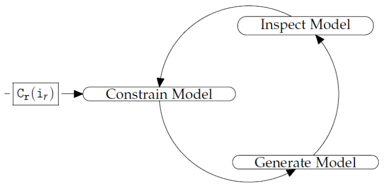


Figure 18.1: Model Exploration Cycle (source: Bauer (2009, p. 25))

Structured Objects. Many of the representational challenges that occurred here were due to the lack of means to express graph structured relationships between variables in DL concept descriptions. Similar problems have been raised in chemistry and the biomedical domain. Here DL extensions and reasoning procedures for special cases have been formulated. Motik et al. (2008) for example introduce Description Graphs as extensions to OWL together with a Hypertableaux reasoning method. It should be explored in how far these methods can also offer further expressive means to overcome the limitations of OWL for music reprsentation met in this work.

Melodies. An interesting path to explore is the representation of melodic patterns. The approach taken in this thesis can be directly used to represent rudimentary forms of melodies by considering single notes instead of chords. More interesting, however, is the case of polyphonic patterns that generalise over multiple voices and capture relationships within a voice as well as between voices (see Bergeron, 2009). The required representation of graph-shaped relationships is outside the expressivity OWL-DL. Future research could investigate the application of description graph extension of OWL-DL (Motik et al., 2009b) to this problem.

More Structure. The $\mathcal{SEQ}$ pattern as described here is limited to horizontal structures. More research could be done in combining this structure with a horizontal structure that is for example used by Conklin (2006) or even a more algebraic structure as in Bergeron and Conklin (2008).

Deeper Investigation into Machine Learning techniques. Recently, machine learning packages that directly operate on OWL concepts have become available.[2] Further experiments could compare the result of such a system with the results by the discussed music machine learning approaches. Subsumption reasoning could also be used as a technique to compare the pattern discovered by different approaches.

[2] e.g. `http://dl-learner.org/Projects/DLLearner`

Integration into Web Services and Workflows. An interesting direction of research would be the use of sequence representation and reasoning and in the context of real-time musical event processing (Ibbotson, 2009).

Part V

Appendix

OWL Constructs and Semantics

This appendix lists constructs available in OWL2 DL with their syntax (as used in this thesis) and semantics. Table 18.1 shows types of terminological (TBox) and assertional (ABox) axioms.

Axiom type	**Syntax**	**Semantics**
concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$
concept inclusion	$C \sqsubseteq D$	$C^{I} \subseteq D^{\mathcal{I}}$
concept assertion	$a \mathrel{⋿} C$ or $C(a)$	$a^{\mathcal{I}} \in C^{I}$
role assertion	aRb or $R(a,b)$	$(a^{\mathcal{I}},b^{\mathcal{I}}) \in R^{\mathcal{I}}$

Table 18.1: DL axioms

Table 18.2 and Table 18.3 show concept and role constructs. DLs have a naming scheme that indicates which constructs they include. A DL in the list provides the constructs in and above the given row. Languages of the OWL family extend DLs with datatypes based on XML-Schema datatypes. The use of datatype extensions in DLs is indicated by (D).

DL	**Construct Name**	**Syntax**	**Semantics**
$\mathcal{AL}$	Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
	Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	Top	$\top$	$\Delta^{\mathcal{I}}$
	Bottom	$\bot$	$\emptyset$
	Negation of atomic concepts	$\neg A$	$(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
	Intersection	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	Universal Qualified Quantificaton	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{a \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
	Existential Unqualified Quantification	$\exists R.\top$	$(\exists R.\top)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in R^{\mathcal{I}}\}$
$\mathcal{AL}[\mathcal{U}][][\mathcal{C}]$	Union	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	Existential Qualified Quantification	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
	Negation of complex concepts	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$

Table 18.2: Common Description Logics and their constructs

Note that though OWL-Full includes OWL-DL, it does not fall into the DL family. As it is layered on full RDFS, it uses axiomatic semantics instead of a model theoretic semantics. In contrast to OWL-DL, OWL-Full is not a fragment of first order logic and is undecidable.

DL	Construct Name	Syntax	Semantics
$\mathcal{S}$	$\mathcal{ALC}$ with transitive roles	$\texttt{tra}\ R$	$\bigcup_{n\geq 1}(R^{\mathcal{I}})^n$
$\mathcal{SI}$	$\mathcal{S}$ with inverse roles	R^-	$\{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}$
$\mathcal{SHI}$	$\mathcal{SI}$ with Subsumption between roles	$R \sqsubseteq S$	$(R \sqsubseteq S)^{\mathcal{I}} = (R^{\mathcal{I}} \subseteq S^{\mathcal{I}})$
$\mathcal{SHIF}$	$\mathcal{SHI}$ with Functional roles	$\texttt{fun}\ R$	$\forall a,b,c\ \ R^{\mathcal{I}}(a,b) \wedge R^{\mathcal{I}}(a,c) \rightarrow b = c$
$\mathcal{SHIN}$	$\mathcal{SHI}$ with Existential Unqualified Number Restrictions	$\exists^{\geqslant n}R$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\| \geq n\}$
		$\exists^{\leqslant n}R$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\| \leq n\}$
		$\exists^{=n}R$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}\| = n\}$
$\mathcal{SHIQ}$	$\mathcal{SHI}$ with Existential Qualified Number Restrictions	$\exists^{\geqslant n}R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\}$
		$\exists^{\leqslant n}R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\| \leq n\}$
		$\exists^{=n}R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\| = n\}$
$\mathcal{SHOIN}$	$\mathcal{SHIN}$ with Nominals	$\{a\}$	$\{a^{\mathcal{I}}\}$
$\mathcal{SROIQ}$	Self concept	$\exists R.\textbf{Self}$	$\{a \in \Delta^{\mathcal{I}} \mid \langle a,a\rangle \in R^{\mathcal{I}}\}$
	reflexive (noncomplex) roles	$\texttt{ref}\ R$	$\forall a,b : (a,b) \in R^{\mathcal{I}} \rightarrow a = b$
	irreflexive (noncomplex) roles	$\texttt{irr}\ R$	$\forall a,b : (a,b) \in R^{\mathcal{I}} \rightarrow a \neq b$
	symmetric (noncomplex) roles	$\texttt{sym}\ R$	$\forall a,b : (a,b) \in R^{\mathcal{I}} \rightarrow (b,a) \in R^{\mathcal{I}}$
	asymmetric (noncomplex) roles	$\texttt{asy}\ R$	$\forall a,b : (a,b) \in R^{\mathcal{I}} \rightarrow (b,a) \notin R^{\mathcal{I}}$

Table 18.3: Common Description Logics (continued)

DL		Corresponding DL
OWL	-Lite	$\mathcal{SHIF}(\mathcal{D})$
	-DL	$\mathcal{SHOIN}(\mathcal{D})$
	-Full	—
OWL2		$\mathcal{SROIQ}(\mathcal{D})$

Table 18.4: OWL variants

Interpretation in OWL is defined as follows:

Definition 45 (OWL2 Direct Semantics). *A* datatype map *is a 6-tuple $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$, where*

- *N_{DT} is a set of datatypes,*
- *N_{LS} is a function that assigns lexical forms to each datatype,*
- *N_{FS} is a function that assigns a set of facet value pairs to each datatype,*
- *$\cdot^{DT}$ is an interpretation function that assigns a value space to each datatype,*
- *$\cdot^{LS}$ is an interpretation function that assigns a data value to a pair (LV, DT) where LV is a lexical form of the datatype DT, and*
- *$\cdot^{FS}$ is an interpretation function that assigns a set of data valuers to a pair of constraining facet and data value.*

A vocabulary *$V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$ over a datatype map D is a 7-tuple, where*

- *V_C is a set of classes,*
- *V_{OP} is a set of object properties,*
- *V_{DP} is a set of data properties,*
- *V_I is a set of individuals (named and anonymous),*
- *V_{DT} is a set containing at least all datatypes of D plus the datatype rdfs:Literal,*
- *V_{LT} is a set of literals LV^^DT with DT a datatype and LV a lexical form for DT, and*
- *V_{FA} is the set of pairs of constraining facets and literals.*

Given a datatype map D and a vocabulary V over D, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta^{D}, \cdot^{C}, \cdot^{OP}, \cdot^{DP}, \cdot^{\mathcal{I}}, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for D and V is a 9-tuple, where

- *$\Delta^{\mathcal{I}}$ is a nonempty set called the object domain,*
- *Δ^{D} is a nonempty set disjoint with $\Delta^{\mathcal{I}}$ called the data domain,*
- *$\cdot^{C}$ is the class interpretation function that assigns to each class $C \in V_C$ a subset C^C of $\Delta^{\mathcal{I}}$, similarly*
- *$\cdot^{OP}$ and*
- *$\cdot^{DP}$ interpret object and data properties as binary relations over $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and $\Delta^{\mathcal{I}} \times \Delta^{D}$ respectively,*
- *$\cdot^{\mathcal{I}}$ assigns to each individual $a \in V_I$ an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,*
- *$\cdot^{DT}$ assigns to each datatype $DT \in V_{DT}$ a subset of Δ^{D},*
- *$\cdot^{LT}$ is the literal interpretation function that assigns to a literal LV^^$DT \in V_{LT}$ a data value $(LV, DT)^{LS} \in (DT)^{DT}$, and, finally,*
- *$\cdot^{FA}$ is the facet interpretation function.*

Namespaces and Expressivity of Ontologies

The table below shows ontologies that are referred to in this thesis. Prefixes that are used for abbreviating URIs are given. E.g. `<http://purl.org/ontology/mo/Composer>` is abbreviated `mo:Composer`. Further, the expressivity is indicated in terms of the DL fragment that is sufficient to express the ontology and which can be related to worst-case complexity.

Prefix	**Name**	**DL Expressivity**
	Namespace	
rdf	Resource Description Framework (RDF) `http://www.w3.org/1999/02/22-rdf-syntax-ns#`	$\mathcal{AL}$
rdfs	Resource Description Framework Schema (RDFS) `http://www.w3.org/2000/01/rdf-schema#`	$\mathcal{ALH}$
owl	Web Ontology Language (OWL) `http://www.w3.org/2002/07/owl#`	$\mathcal{ALCH}$
xsd	XML Schema `http://www.w3.org/2001/XMLSchema#`	(D)
dcterms	`http://purl.org/dc/terms/` Dublin Core Element Refinements and Encoding Schemes `http://dublincore.org/documents/dcmi-terms/#H3`	$\mathcal{ALH}_{(D)}$
dc	Dublin Core Element Set v1.1 `http://purl.org/dc/elements/1.1/` `http://dublincore.org/documents/dcmi-terms/#H2`	$\mathcal{AL}_{(D)}$
event	Event Ontology `http://purl.org/NET/c4dm/event.owl#`	$\mathcal{SHOIN}_{(D)}$
ex	Dummy Namespace for Examples `http://www.example.org/`	—
frbr	FRBR `http://vocab.org/frbr/core#`	$\mathcal{ALCHI}(D)$
foaf	Friend of a Friend (FOAF) Vocabulary `http://xmlns.com/foaf/0.1/`	$\mathcal{ALCHIF}(D)$
mo	OMRAS2 Music Ontology `http://purl.org/ontology/mo/` `http://musicontology.com/`	$\mathcal{SHOIN}_{(D)}$
chord	OMRAS2 Chord Ontology (Draft) `http://purl.org/ontology/chord/index.rdfs` `http://motools.sourceforge.net/chord_draft_1/chord.html`	$\mathcal{AL}(D)$
timeline	OMRAS2 TimeLine Ontology `http://purl.org/NET/c4dm/timeline.owl#`	$\mathcal{SHOIN}(D)$
ton	OMRAS2 Tonality Ontology `http://purl.org/ontology/tonality/` `http://motools.sourceforge.net/doc/tonality.htm`	$\mathcal{ALOI}(D)$
list	OWLList `http://www.co-ode.org/ontologies/meta/2006/05/15/meta.owl`	$\mathcal{SHN}(D)$
time	Time Ontology `http://www.w3.org/2006/time`	$\mathcal{SHOIN}(D)$

References

Allen, J. F. (1984). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Anders, T. (2007). *Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System.* PhD thesis, School of Music & Sonic Arts, Queen's University Belfast.

Anglade, A. and Dixon, S. (2008). Characterisation of harmony with inductive logic programming. In *Proc. of the Ninth International Conference on Music Information Retrieval (ISMIR)*, pages 63–68, Philadelphia, USA.

Angles, R. and Gutierrez, C. (2008). The expressive power of sparql. pages 114–129.

Artale, A. and Franconi, E. (2001). A survey of temporal extensions of description logics. In *Annals of Mathematics and Artificial Intelligence (AMAI)*, volume 30, pages 171–210. Kluwer Academic Press.

Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005).*

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press.

Baader, F. and Küsters, R. (2006). Nonstandard inferences in description logics: The story so far. In Gabbay, D., Goncharov, S., and Zakharyaschev, M., editors, *Mathematical Problems from Applied Logic I*, volume 4 of *International Mathematical Series*, pages 1–75. Springer-Verlag.

Baader, F., Molitor, R., Molitor, R., Tobies, S., and Tobies, S. (1998). On the relation between conceptual graphs and description logics. Technical report, LuFg Theoretische Informatik, RWTH Aachen.

Baader, F., Sertkaya, B., and Turhan, A.-Y. (2007). Computing the least common subsumer w.r.t. a background terminology. *Journal of Applied Logic*, 5(3):392–420.

Baroni, M. and Callegari, L. (1984). *Musical Grammars and Computer Analysis.* Olschki, Florence.

Bauer, J. (2009). *Model Exploration to Support Understanding of Ontologies*. PhD thesis, Technische Universität Dresden, Institut für Theoretische Informatik.

Bechhofer, S., Möller, R., and Crowther, P. (2003). The DIG description logic interface. In *Proc. of the Int. Workshop on Description Logics (DL'03)*, volume 81 of *CEUR*, Rome, Italy.

Bergeron, M. (2009). *Structured Polyphonic Patterns*. PhD thesis, Department of Computing, City University London, United Kingdom.

Bergeron, M. and Conklin, D. (2008). Structured polyphonic patterns. In *ISMIR 2008: International Conference on Music Information Retrieval, Philadelphia*.

Berners-Lee, T. (1999). *Weaving the Web : the past, present and future of the World Wide Web by its inventor*. Orion Business, London.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The SemanticWeb. *Scientific American*, 284:34–43.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22.

Borst, W. (1997). *Construction of Engineering Ontologies*. PhD thesis, University of Tweenty, Enschede, The Netherlands.

Brachman, R. J. (1977). What's in a concept: Structural foundations for semantic networks. *Int. Journal of Man-Machine Studies*, 9(2):127–152.

Celma, O., Herrera, P., and Serra, X. (2006). Bridging the music semantic gap. volume 187, Budva, Montenegro. CEUR, CEUR.

Chemilier, M. (2001). Improvising jazz chord sequences by means of formal grammars. In *Journee d'Informatique musicale, JIM2001, Bourges, France, September 19-22*, pages 121–126.

Chemillier, M. (2004). Toward a formal study of jazz chord sequences generated by steedman's grammar. *Soft Comput.*, 8(9):617–622.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(2):113–123.

Chomsky, N. (1957). *Syntactic Structures*. The Hague: Mouton.

Chomsky, N. (1959). A note on phrase structure grammars. *Information and Control*, 2(4):393–395.

Cohen, W. W., Borgida, A., and Hirsh, H. (1992). Computing least common subsumers in description logics. In *AAAI*, pages 754–760.

Colucci, S., Di Sciascio, E., and Donini, F. M. (2008). Partial and informative common subsumers of concepts collections in description logics. In *Proc. of the 21st Intl. Workshop on Description Logics (DL 2008)*, volume 353. CEUR.

Conklin, D. (1994). Structured concept discovery: Theory and methods. Technical Report 94-366, Queen's University, Kingston, Ontario, Canda.

Conklin, D. (2006). Melodic analysis with segment classes. *Machine Learning*, 65(2-3):349–360.

Conklin, D. (2008). Discovery of distinctive patterns in music. In *MML08: International Workshop on Machine Learning and Music, Helsinki, Finland*, pages 31–32.

Conklin, D. (2010). Discovery of distinctive patterns in music. *Intelligent Data Analysis*, 14(5):547–554.

Conklin, D. and Bergeron, M. (2008). Feature set patterns in music. *Computer Music Journal*, 32(1):60–70.

Cuenca Grau, B., Horrocks, I., Kazakov, Y., and Sattler, U. (2008). Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research (JAIR)*, 31:273–318.

Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425.

Davis, R., Shrobe, H., and Szolovits, P. (1993). What is a knowledge representation? *AI Magazine*, 14(1):17–33.

de Bruijn, J., Franconi, E., and Tessaris, S. (2005). Logical reconstruction of RDF and ontology languages. In *Proceedings of the 3rd Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR2005)*, volume 3703 of *Lecture Notes in Computer Science*, Springer.

De Roure, D., Page, K. R., Fields, B., Crawford, T., Downie, J. S., and Fujinaga, I. (2011). An e-research approach to web-scale music analysis. *Phil. Trans. R. Soc. A*, 369(1949):3300–3317.

Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge, M., Wang, H. H., and Seidenberg, H. (2006a). Sequences in Protégé OWL. In *9th Intl. Protégé Conference - July 23-26, 2006 - Stanford, California*.

Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge, M., Wang, H. H., and Seidenberg, J. (2006b). Putting OWL in Order: Patterns for Sequences in OWL. In *2nd OWL Experiences and Directions Workshop, Athens, GA*.

Eberlein, R. u. J. P. F. (1992). *Kadenzwahrnehmung und Kadenzgeschichte - ein Beitrag zu einer Grammatik der Musik*. Verlag Peter Lang, Franfurt a.M.

Frederico, G. C. S. (2002). Actos: A peer-to-peer application for the retrieval of encoded music. In *First International Conference on Musical Applications Using XML*.

Fux, J. J. (1967). *Gradus ad Parnassum.* Bärenreiter und Akademische Druck- und Verlagsanstalt, 1725 (Neuauflage der Johann Joseph Fux Gesellschaft, Graz; Kassel Basel Paris London New York.

Garcia, M., Kaplunova, A., and Möller, R. (2007). Model Generation in Description Logics: What Can We Learn From Software Engineering? Technical report, Institute for Software Systems (STS), Hamburg University of Technology, Germany. See http://www.sts.tu-harburg.de/tech-reports/papers.html.

Gasse, F., Sattler, U., and Haarslev, V. (2008). Rewriting rules into $\mathcal{SROIQ}$ axioms. In Baader, F., Lutz, C., and Motik, B., editors, *Description Logics*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Gehrkens, K. W. (1914). *Music Notation and Terminology.* Barnes, New York.

Giomi, F. and Ligabue, M. (1991). Computational generation and study of jazz music. *Interfaces.*

Grädel, E. (2001). *Current Trends in Theoretical Computer Science. Entering the 21st Century*, chapter Why Are Modal Logics So Robustly Decidable?, pages 393–408. World Scientific.

Grimm, S. and Wissmann, J. (2011). Elimination of redundancy in ontologies. In *Proceedings of the Extended Semantic Web Conference (ESWC2011), Heraklion, Greece.*

Grosof, B., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logics. In *Proceedings of WWW 2003*, pages 48–57, Budapest, Hungary. ACM.

Haarslev, V. and Möller, R. (2003). Racer: A core inference engine for the semantic web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, pages 27–36, Sanibel Island, Florida, USA.

Haerle, D. (1980). *The Jazz Language.* Studio 224, Miami, Fl.

Harte, C., Sandler, M. B., Abdallah, S. A., and Gómez, E. (2005). Symbolic representation of musical chords: A proposed syntax for text annotations. In *ISMIR*, pages 66–71.

Hayes, P. (2004). RDF semantics. W3C recommendation, W3C.

Hayes, P. J. (1977). In defense of logic. In *Proc. of the 5th Int. Joint Conf. on Artificial Intelligence (IJCAI'77)*, pages 559–565.

Hayes, P. J. (1979). The logic of frames. In Metzing, D., editor, *Frame Conceptions and Text Understanding*, pages 46–61. Walter de Gruyter and Co.

Hiraga, Y. (1989). A computational model of the cognition of melodic/harmonic progression. In *Proceedings of the First International Conference on Music Perception and Cognition*, pages 61–66, Kyoto.

Hirsh, H. and Kudenko, D. (1997). Representing sequences in description logics. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence*.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2003). *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley.

Horridge, M., Parsia, B., and Sattler, U. (2008). Laconic and precise justifications in OWL. In Sheth, A. P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T. W., and Thirunarayan, K., editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 323–338. Springer.

Horrocks, I. (1998). Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647.

Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 57–67. 10th International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press.

Horrocks, I. and Patel-Schneider, P. F. (2004). A proposal for an OWL rules language. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA. ACM Press.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2003). SWRL: A semantic web rule language combining OWL and RuleML. Technical report, W3C recommendation, `http://www.daml.org/2003/11/swrl/`.

Horrocks, I. and Sattler, U. (2005). A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453.

Hunter, J. (2001). Adding multimedia to the semantic web — building an MPEG-7 ontology.

Huron, D. (1995). *The Humdrum Toolkit: Reference Manual.* Center for Computer Assisted Research in the Humanities, Menlo Park, California.

Ibbotson, J. B. (2009). *A framework for the real-time analysis of musical events.* PhD thesis, University of Southampton, School of Electronics and Computer Science, Doctoral Thesis.

International MIDI Association (1983). *MIDI Musical Instrument Digitial Interface Specification 1.0.* International MIDI Association, Los Angeles.

Jungbluth, A. (1981). *Jazzharmonielehre.* Schott, Mainz.

Kalyanpur, A., Parsia, B., Horridge, M., and Sirin, E. (2008). Finding all justifications of owl dl entailments. pages 267–280.

Kostka, S. and Payne, D. (2003). *Tonal Harmony.* McGraw-Hill.

Kowalski, R. and Sergot, M. (1989). A logic-based calculus of events. pages 23–51.

Krötzsch, M., Rudolph, S., and Hitzler, P. (2008). Description logic rules. In Ghallab, M., Spyropoulos, C. D., Fakotakis, N., and Avouris, N., editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08)*, IOS Press.

Lerdahl, F. and Jackendoff, R. (1983). *A Generative Theory of Tonal Music.* MIT Press.

Lutz, C., Sattler, U., and Tobies, S. (1999). A proposal for an n-ary description logic. In Lambrix, P., Borgida, A., Lenzerini, M., Möller, R., and Patel-Schneider, P., editors, *Proceedings of the International Workshop on Description Logics*, number 22 in CEUR-WS, pages 81–85, Linkoeping, Sweden. Linköping University. Proceedings online available from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-22/.

Lutz, C., Wolter, F., and Zakharyaschev, M. (2008). Temporal description logics: A survey. In *Proceedings of the Fourteenth International Symposium on Temporal Representation and Reasoning.* IEEE Computer Society Press.

Mauch, M., Cannam, C., M. Davies, S., Dixon, Harte, C., Kolozali, S., Tidhar, D., and Sandler, M. (2009). Omras2 metadata project 2009. In *Late-breaking session at the 10th International Conference on Music Information Retrieval, Kobe, Japan, 2009a..*

Mauch, M., Dixon, S., Harte, C., Casey, M., and Fields, B. (2007). Discovering chord idioms through Beatles and Real Book songs. In *Proceedings of ISMIR 2007 Vienna, Austria*, pages 255–258.

Mazzola, G. (2003). *The Topos of Music: Geometric Logic of Concepts, Theory, and Performance.* Birkhäuser Basel.

McDermott, D. (1982). A temporal logic for reasoning about processes and plans. *Cognitive science*, 6:101–155.

Meeus, N. (2000). Toward a post-Schoenbergian grammar of tonal and pre-tonal harmonic progressions. *Music Theory Online*, 6(1).

Milea, V., Frasincar, F., and Kaymak, U. (2008). Knowledge engineering in a temporal semantic web context. In *The Eighth International Conference on Web Engineering (ICWE 2008)*. IEEE Computer Society Press.

Minsky, M. (1981). A framework for representing knowledge. In Haugeland, J., editor, *Mind Design*. The MIT Press.

Moore, R. C. (2000). Removing left recursion from context-free grammars. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 249–255, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C., editors (27 October 2009a). *OWL 2 Web Ontology Language: Profiles*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-profiles/`.

Motik, B., Grau, B. C., Horrocks, I., and Sattler, U. (2009b). Representing Ontologies Using Description Logics, Description Graphs, and Rules. *Artificial Intelligence*, 173(14):1275–1309.

Motik, B., Grau, B. C., and Sattler, U. (2008). Structured Objects in OWL: Representation and Reasoning. In Huai, J., Chen, R., Hon, H.-W., Liu, Y., Ma, W.-Y., Tomkins, A., and Zhang, X., editors, *Proc. of the 17th Int. World Wide Web Conference (WWW 2008)*, pages 555–564, Beijing, China. ACM Press.

Motik, B., Patel-Schneider, P. F., and Cuenca Grau, B., editors (27 October 2009c). *OWL 2 Web Ontology Language: Direct Semantics*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-direct-semantics/`.

Motik, B., Sattler, U., and Studer, R. (2005). Query answering for OWL-DL with rules. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60.

Nattiez, J.-J. (1990). *Music and Discourse: Toward a Semiology of Music*.

Network Working Group (2005). RFC3986 uniform resource identifiers (URI): Generic syntax. Technical report, IETF.

Okhotin, A. (2001). Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4):519–535.

Okhotin, A. (2003). Boolean grammars. In Ésik, Z. and Fülöp, Z., editors, *Developments in Language Theory*, volume 2710 of *Lecture Notes in Computer Science*, pages 398–410. Springer.

Pachet, F. (1997). Computer analysis of jazz chord sequences: Is solar a blues. In *Readings in Music and Artificial Intelligence*. Ed, Harwood Academic Publishers.

Pérez-Sancho, C., Rizo, D., and Iñesta, J. M. (2008). Stochastic text models for music categorization. In *Proceedings of 12th International Workshop on Structural and Syntactic Pattern Recognition, SSPR 2008 — Satellite event of the 19th International Conference of Pattern Recognition, ICPR 2008.*, number 5342 in 2008, pages 55–64, Berlin. Springer.

Pérez-Sancho, C., Rizo, D., and Iñesta, J.-M. (2009). Genre classification using chords and stochastic language models. *Connection Science*, 20(2&3):145–159.

Potter, G. M. (1989). The unique role of bvii7 in bebop harmony. *jazz forschung*, 21.

Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430.

Raimond, Y. (2009). *A Distributed Music Information System.* PhD thesis, Department of Electronic Engineering, Queen Mary, University of London.

Raimond, Y., Abdallah, S., Sandler, M., and Giasson, F. (2007). The music ontology. In *Proceedings of the International Conference on Music Information Retrieval.*

Raimond, Y. and Sandler, M. (2008). A web of musical information. In *Proc. of the Ninth International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, USA.

Rector, A. L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., and Wroe, C. (2004). Owl pizzas: Practical experience of teaching owl-dl: Common errors & common patterns. 3257:63–81.

Riemann, H. (1877). *Musikalische Syntaxis.* Breitkopf und Härtel, Leipzig.

Roads, C. (1987). Grammars as representations for music. In Roads, C. and Strawn, J., editors, *Foundations of Computer Music.* The MIT Press, Cambridge Mass.

Rudolph, S., Krötzsch, M., and Hitzler, P. (2008). All elephants are bigger than all mice. In *Proceedings of the 21st International Workshop on Description Logics (DL-08).*

Schaffer, J. W. and McGee, D. (1998). *Knowledge-Based Programming for Music Research.* Computer Music and Digital Audio Series. A-R Editions.

Schmidt-Schauß, M. and Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53.

Sowa, J. F. (1984). *Conceptual structures : information processing in mind and machine*. Addison-Wesley, Reading, Mass.

Stamou, G. and Kollias, S., editors (2005). *Multimedia Content and the Semantic Web: Methods, Standards and Tools*. John Wiley & Sons Ltd.

Stanton, K. (1982). *Jazz Theory — A Creative Approach*. Taplinger, New York.

Steedman, M. J. (1984). A generative grammar for jazz chord sequences. *Music Perception*, 2(1):52.

Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *IEEE Transactions on Data Knowledge Engineering*, 25(1-2):161–197.

Sundberg, J. and Lindblom, B. (1976). Generative theories in language and music description. *Cognition*. Nachdruck in Schwanauer, Stephan M. und Levitt, David A. (Hrsg.): Machine Models of Music. Cambridge Mass. 1993.

Temperley, D. (1997). An algorithm for harmonic analysis. *Music Perception*, 15(1):31–68.

Temperley, D. (2001). *The cognition of basic musical structures*. MIT, Cambridge, Mass. David Temperley.

ter Horst, H. J. (2005). Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the owl vocabulary. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):79–115.

Tsang, E. P. K. (1987). Time structures for artificial intelligence. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 456–461.

Tsarkov, D. and Horrocks, I. (2006). FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer.

Tummarello, G., Morbidoni, C., Puliti, P., Dragoni, A. F., and Piazza, F. (2004). From multimedia to the semantic web using mpeg-7 and computational intelligence. In *WEDELMUSIC '04: Proceedings of the Web Delivering of Music, Fourth International Conference*, pages 52–59, Washington, DC, USA. IEEE Computer Society.

Ulrich, J. W. (1977). The analysis and synthesis of jazz by computer. In Kaufmann, W., editor, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Cambridge, MA.

Vardi, M. Y. (1997). Why is modal logic so robustly decidable? Technical Report TR97-274.

Villinger, A. (2006). *The Beatles-Songs — Harmonic and Melodic Analyses (german: Die Beatles-Songs — Analysen zur Harmonik und Melodik).* Verlagshaus Mainz.

Wang, H. H., Dong, J. S., Sun, J., and Sun, J. (2006). Reasoning support for semantic web ontology family languages using alloy. *Multiagent and Grid Systems*, 2(4).

Welty, C. and Fikes, R. (2006). A reusable ontology for fluents in OWL. In *Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006)*, pages 226–336. IOS Press.

Weyde, T. (1994). Aspekte der automatischen Erzeugung musikalisch sinnvoller Akkordfolgen. Staatsexamensarbeit, Universität Osnabrück.

Weyde, T. (1995). Grammatikbasierte harmonische analyse von jazzstandards mit computerunterstützung. In Enders, B. and Knolle, N., editors, *KlangArt Kongreß 1995. Musik und Neue Technologie, Bd. 1. Universitätsverlag Rasch, Osnabrück.*

Weyde, T. (2005). Modelling cognitive and analytic musical structures in the MUSITECH framework. In *UCM 2005 5th Conference "Understanding and Creating Music", Caserta, November 2005*, pages 27–30.

Weyde, T. and Wissmann, J. (2007). Automatic semantic annotation of music with harmonic structure. In Spyridis, C., Georgaki, A., Kouroupetroglou, G., and Anagnostopoulou, C., editors, *Proceedings of the 4th Sound and Music Computing Conference 11–13 July 2007*, Lefkada, Greece.

Wissmann, J., Weyde, T., and Conklin, D. (2010). Chord sequence patterns in owl. In *Proceedings of SMC Conference 2010, Barcelona.*

Woods, W. A. (1975). What's in a link: Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M., editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35–82. Academic Press.

Index

$\exists^{=n}R.C$, 172
$\exists^{\geqslant n}R.C$, 172
$\exists^{\leqslant n}R.C$, 172
$C \sqcap D$, 171
$C \sqcup D$, 171
$R \sqsubseteq S$, 172
R^{-}, 172
$\Delta^{\mathcal{I}}$, 21
$\cdot^{\mathcal{I}}$, 21
$\blacktriangleright$, 74
$\triangleright$, 74
$\triangleright^{-}$, 74
α, 74
$\bot$, 171
$\texttt{asy}\ R$, 172
$\texttt{fun}\ R$, 172
$\texttt{irr}\ R$, 172
$\texttt{ref}\ R$, 172
$\texttt{sym}\ R$, 172
$\texttt{tra}\ R$, 172
$\exists R.C$, 171
$\exists R.\mathbf{Self}$, 172
$\exists^{=n}R$, 172
$\exists^{\geqslant n}R$, 172
$\exists^{\leqslant n}R$, 172
$\forall R.C$, 171
$\neg C$, 171
$\ntriangleright$, 74
ω, 74
$\{\ldots\}$, 65
$\top$, 171
$\{a\}$, 172
$\mathcal{I}$, 21

A-production, 98
$\odot$, 74
seq:`Sequence`, 81
$\langle\mathcal{O}\rangle$, *see* deductive closure
$\Sigma(\mathcal{O})$, *see* signature

blending effect, 86

closed world assumption, 25
closure axiom, xi, 31
completeness, 24
concatenation, xi, 88
concept closure, xi, 31
concept product, 35
Conceptual Graph, 92
constraints, 25
CWA, 25

deductive closure, 22
disjoint union, 31

entailment, 22
expansion, 98

Four-Chord Patterns, 128

grammar, 98
  derivation, 98

identity role, xi, 34

lcs, 38
least common subsumer, 38

module extraction, 36

normal form
  disjunctive normal form, 87
NSE, *see* self-embedding

OMRAS2 Chord Ontology, 128
open world assumption, 25
OWA, 25

RDF
  graph, 14
  triple, 14
regular grammar, xii, 99
restricted role subsumption, 34
restrictions, 25
RG, *see* regular grammar, *see* regular grammar
root, 69

safe concatenability, 103
SE, *see* self-embedding
self-embedding, 99
sequence concatenation, xi, 88
signature, 22
soundness, 24
SPARQL, 39
successor assertion, 74
successor restriction, 74
SWRL, 32

tonal center, 69