REPORT
ONTOLOGY BASED MUSIC THEORY

DONE BY:
JUAN GONZÁLEZ MAGDALENA
JUAN RUBIO GÓMEZ

KNOWLEDGE ENGINEERING, 22/23

INDEX:

1. Introduction

This project aims to develop an Ontology and a Knowledge Graph that models relevant theories from the musicology field. Musicology is the field that studies the rudiments of music. A vast amount of theories have been proposed to analyse music. In this project, we will develop and ontology that models one of those theories reusing relevant ontologies (as the Music Theory Ontology). Our resulting ontology will be used to create a Knowledge Graph that enables the analysis of musical compositions contained in ChoCo.

Following the eXtreme Design Methodology, we created some competency questions (CQs) that we would utilize to construct our ontology after analyzing the various data sets. Then, during the testing phase, we verified the CQs to ensure that our ontology was consistent and that the inferences were accurate. Finally, we used SPARQL-Anything to map the provided data into RDF triples in accordance with the created ontology.

2. Data

For this Knowledge Engineering project, we will be utilizing a variety of files found in the ChoCo GitHub repository. This repository hosts a collection of JAMS files that are used for the analysis of music theory. The JAMS files contain musical annotations that allow us to extract valuable information related to music theory, such as chords, progressions, scales, and more. These JAMS files provide a structured representation of the musical data and serve as a solid foundation for conducting analysis and extracting knowledge. The ChoCo GitHub repository is a reliable and up-to-date source that provides us with the necessary resources for this project. By utilizing these JAMS files, we will be able to build an intelligent knowledge system that enables us to further investigate and comprehend the theoretical foundations of music.

3. Competency questions

Our competency questions (CQs) are the natural language counterpart of structured queries that we want to enable against the knowledge graph. These interrogatives allow us to scope our ontology: they are questions that our users would want to gain answers for, through exploring and querying the ontology and its associated knowledge base.

Here we specify our CQs and its respective SPARQL query implementation:

1. What chords are present in a given progression?

```
SELECT ?nameChord
WHERE{
?Progression pm:isMadeOf ?Chord.
?Chord pm:hasName ?nameChord
}
```

2. What progressions are present in a song?

```
SELECT DISTINCT ?Progression
WHERE {
?Song pm:isComposedBy ?Progression
}
```

3. What is the duration of a given progression?

```
SELECT ?duration
WHERE {
?Progression pm:hasDuration ?duration
}
```

4. How many notes have a given chord?

```
SELECT COUNT(*)
WHERE{
?Chord pm:isMadeOf ?Note
}
```

5. Has caesura a given loop?

```
SELECT ?caesura
WHERE{
?Loop pm:hasCaesura ?caesura
}
```

6. What are the different chords present in a loop?

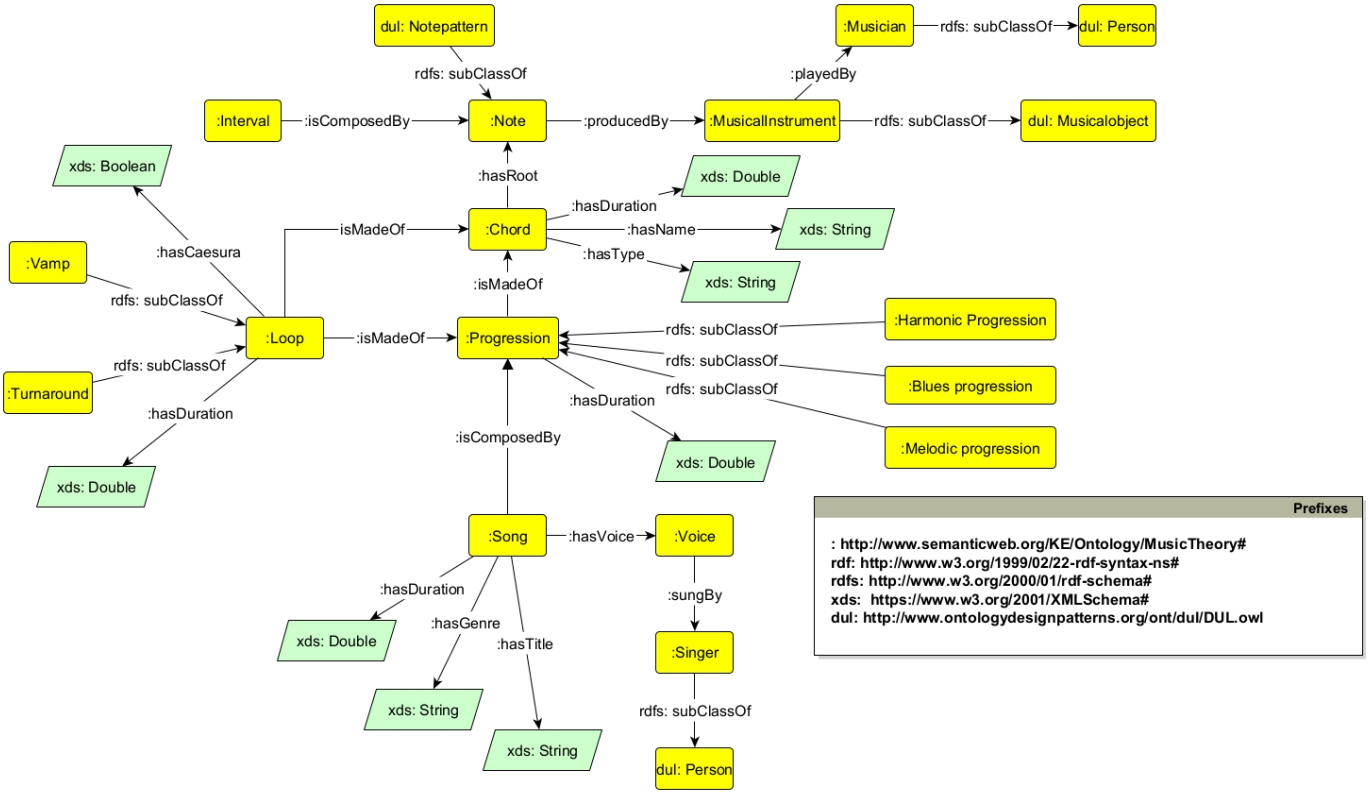
```
SELECT DISTINCT ?nameChord
WHERE{
?Loop pm:isMadeOf ?Chord.
?Chord pm:hasName ?nameChord
}
```

7. What are the different notes a progression is made of?

```
SELECT DISTINCT ?nameNote
WHERE{
?Progression pm:isMadeOf ?Chord.
?Chord pm:hasRoot ?Note.
?Note pm:hasName ?nameNote
}
```

4. Ontology

Now, we present our ontology and explain our design choices: the classes, patterns, and relations we have implemented.



The graph represents all the classes and relations of our ontology. Everything is centered around the **:Progression**, as we have considered that is the most important aspect in the Music Theory topic. Also, **:Chord** and **:Note** have an important role in our ontology.

The **class :Progression** makes reference to the definition of a progression: "series of two or more chords". It has characteristics like its duration and its types (**:Harmonic**, **:Melodic** and **:Blues** progressions). A :**Song** then, is composed of progressions and each **:Progression** is made of chords. A **:Chord** is a combination of three or more notes, so is made up of notes,

where one of them is the root of the chord. A **:Chord** has a name, a duration and a type. On the other hand, a **:Note** has all the properties included by the pattern **dul:Notepattern**. Also, a **:Note** is produced by a musical instrument (its properties are from **dul:Musicalobject**) played by a **:Musician** (its properties are from **dul:Person**). An **:Interval** is composed by notes while a **:Loop** is made of chords. A **:Loop** can have a specific duration and a caesura (a break, pause, or interruption in the normal tempo of a loop). Also, a **:Vamp** and a **:Turnaround** are subclasses of a **:Loop**. Finally, going back to **:Song**, it has a voice sung by a singer with properties of the pattern **dul:Person** and also has a duration, a title and a genre to classify it.

The patterns used are summarized in the following table:

CLASS	PATTERN	DESCRIPTION OF THE PATTERN
MusicalInstrument	*Musicalobject*	This content ODP models the acoustic features of a music note played in a performance.
Musician	*Person*	Persons in commonsense intuition, which does not apparently distinguish between either natural or social persons.
Singer	*Person*	Persons in commonsense intuition, which does not apparently distinguish between either natural or social persons.
Note	*Notepattern*	The Music Note ODP models a symbolic note and its symbolic attributes

5. Testing

We have decided to use competency question verification as our testing method. Through this testing process, it is possible to determine whether the ontology can respond to the competency questions that have been gathered. To follow this strategy, we first developed a few test cases in accordance with the specifications of the OWL unit testing framework. As test data input, we built toy datasets. OWL-unit ensures that the SPARQL query IRIs are defined in the tested ontology or in the input test data, and that the outcome of the SPARQL unit test query assessed over the input data is isomorphic to the desired outcome.

Following is an example (of CQ1) of the test case and toy dataset that we specified for each CQ:

```
@prefix owlunit: <https://w3id.org/OWLunit/ontology/> .
@prefix tc: <https://raw.githubusercontent.com/juanglezmag/MusicTheory-KE23/main/test/test-case/> .
@prefix td: <https://raw.githubusercontent.com/juanglezmag/MusicTheory-KE23/main/test/toy-dataset/> .
@prefix pm: <https://raw.githubusercontent.com/juanglezmag/MusicTheory-KE23/main/Ontology/musicTheory.owl> .

tc:CQ1.ttl a owlunit:CompetencyQuestionVerification ;
        owlunit:hasCompetencyQuestion "What chords are present in a given progression?" ;
        owlunit:hasSPARQLUnitTest "PREFIX pm:<https://raw.githubusercontent.comjuanglezmag/MusicTheory-KE23/main/Ontology/musicTheory.owl>
        SELECT ?nameChord WHERE{ ?Progression pm:isMadeOf ?Chord. ?Chord pm:hasName ?nameChord }" ;
        owlunit:hasInputData td:TD1.ttl ;
        owlunit:hasInputTestDataCategory owlunit:ToyDataset ;
        owlunit:hasExpectedResult "{  \"head\": {  \"vars\": [  \"nameChord\" ] } ,  \"results\": {  \"bindings\": [ {  \"nameChord\": {
        \"datatype\":  \"http://www.w3.org/2001/XMLSchema#String\", \"type\": \"literal\" ,  \"value\": \"Chord01\" } } ] } }";
        owlunit:testsOntology pm: .
```

```
@prefix td: <https://raw.githubusercontent.comjuanglezmag/MusicTheory-KE23/main/test/toy-dataset/> .
@prefix pm: <https://raw.githubusercontent.com/juanglezmag/MusicTheory-KE23/main/Ontology/musicTheory.owl> .

td:Progression_01 pm:isMadeOf td:Chord_01 .
td:Chord_01 pm:hasName "Chord01" .
```

All the test cases and the toy datasets are available in our GitHub repository by clicking on the following hyperlink: test_case.

6. Data mapping

The data mapping phase heavily relies on the data provided in the ChoCo repository. We extensively explored the dataset and attempted various CONSTRUCT SPARQL queries to the endpoint offered by the ChoCo repository. Our goal was to obtain a .ttl file that would contain the data aligned with our ontology.

Fortunately, we successfully retrieved the desired JSON file, which presented the mapped data in a format compatible with our ontology. This allowed us to establish a connection between the original dataset and our knowledge representation.

However, despite our efforts, we encountered challenges in converting the JSON file into the Turtle format, which is commonly used for querying with Sparql-Anything to try our Ontology.

7. SPARQL Endpoint and Triple Store

We encountered challenges with the specified data. As a workaround, we utilized a set of individuals built within Protege, our ontology development tool, to generate queries against the endpoint. To facilitate this process, we deployed Apache Jena Fuseki, a SPARQL server and

endpoint provided by the Apache Jena framework. This powerful solution enabled us to publish and query RDF data efficiently using the SPARQL query language.

We opted for Apache Jena Fuseki due to its simplicity and intuitive interface, making it easy to work with. The deployment was done locally, specifically on localhost, utilizing port 3030. After setting up the endpoint, we uploaded the relevant data and proceeded to execute various queries.

Here are some examples of the queries we performed:

Query:

```
PREFIX pm: <https://raw.githubusercontent.com/juanglezmag/MusicTheory-KE23/main/Ontology/musicTheory.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?name
WHERE{
    pm:Circle pm:isMadeOf ?Chord.
    ?Chord pm:hasName ?name

}
```

Output:

	name
1	CMaj9
2	Cdim7

Query:

```
PREFIX pm: <https://raw.githubusercontent.com/juanglezmag/MusicTheory-KE23/main/Ontology/musicTheory.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT COUNT(*)
WHERE{
    pm:Round pm:isMadeOf ?Chord.


}
```

Output:

	.1
1	"2"^^<http://www.w3.org/2001/XMLSchema#integer>

8. References

Some references are in hyperlinks on the report. In addition, here there are some of the other references used in this project:

The Music Note Ontology: https://ceur-ws.org/Vol-3011/pattern2.pdf

OWLUnit: https://github.com/luigi-asprino/owl-unit/releases

Everyday Tonality II: https://dl.tufts.edu/downloads/pn89dk12f?filename=hx11xt121.pdf

Ontology Patterns: http://ontologydesignpatterns.org/wiki/Main_Page