The KIAM System in the C@merata Task at MediaEval 2016

Marina Mytrova
Keldysh Institute of Applied Mathematics
Russian Academy of Sciences
Moscow, Russia
mytrova@keldysh.ru

ABSTRACT

The KIAM system is an attempt to solve the C@merata task at MediaEval 2016 [4] using regular expressions. The aim of the KIAM system is to try the suitability of regular expressions for text recognition in a limited dictionary. The system answers natural language queries over musical scores. The C@merata project has existed since 2014 and there are 200 questions each year which a participant's system has to answer automatically. A question consists of a short noun phrase in English referring to a musical feature in the music score like "three consecutive thirds in the left hand in measures 22-30" together with a classical music score in MusicXML [2]. The required answer to this question is a set of one or more passages specifying exact places in the score relevant with input string.

1. INTRODUCTION

The key aim of the C@merata evaluations is to apply Question Answering to Music Information Retrieval. The input is a short natural language query but it is asked against a music score rather than a document. Queries may be composed of

- melodic elements (e.g. note, number of notes, note sequence or melodic interval),
- harmonic elements (harmonic interval, chord or triad),
- performance styles,
- qualifications by instrument/clef/time signature/key,
- qualifications by section/bar/beat/number.

Elements can follow one another or be synchronous with one another. Also, queries may contain information about texture, cadence and counterpoint.

The required answer (passage) is the location in the score of the requested musical feature. A passage consists of a start point and an end point in the score associated with the question. The passage is specified as follows:

- a start time signature,
- an end time signature,
- a start divisions value,
- an end divisions value,
- a start beat,
- an end beat.

The XML formats for the training data and run submissions for C@merata 2016 task were as shown below:

MediaEval 2016 Workshop, October 20-21, 2016, Amsterdam.

```
<questions task="camerata"
    year="2016" task_no="1"
    runtag="lacg01"
    organisation="University of Essex"
    group="LAC Group">
    <question number="001"
        music_file="f01.xml"
        divisions="4">
        <text>harmonic perfect fifth</text>
        <answer>
            <passage start_beats="4"
                start_beat_type="4"
                end_beats="4"
                end_beat_type="4"
                start_divisions="4"
                end_divisions="4"
                start_bar="2"
                start_offset="1"
                end_bar="2"
                end_offset="4" />
        </answer>
    </question>
</questions>
```

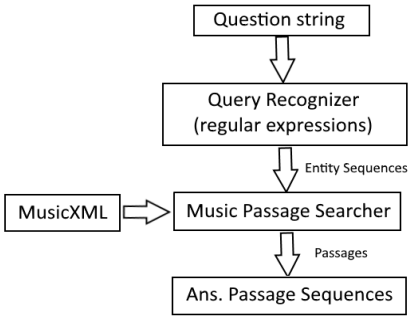
The phrase in the <text> tag is a natural language query. Attributes of the <passage> tag are the answer. It shows the location in the given MusicXML score of the requested musical feature.

2. APPROACH

2.1 Description

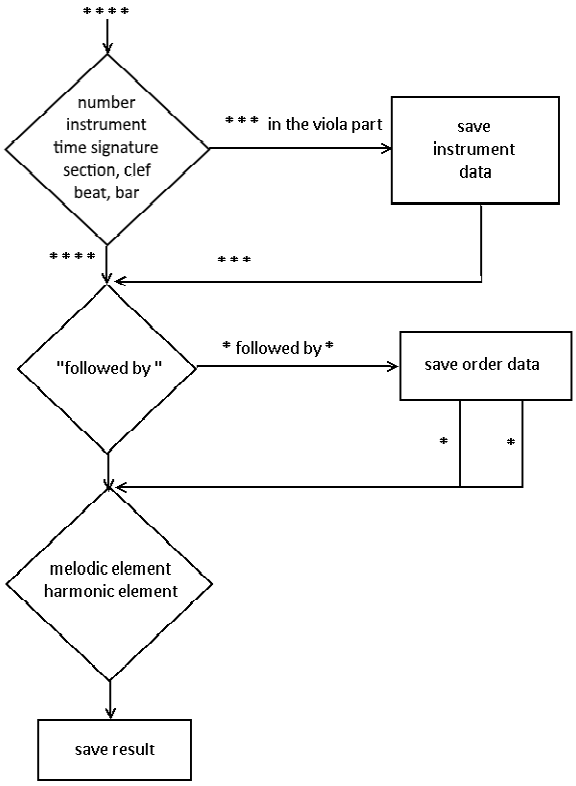
The vocabulary of musical terms is limited and the specification of queries in the 2016 task contains many examples of its components. It can be used to construct a large amount of regular expressions. So, regular expressions were chosen for query recognition, following the model of the UNLP work [1] in the 2015 task [5]. The approach consists of two parts: query recognition and music passage search. This is similar to UNLP but the KIAM system contains different regular expression for each example of query components.

The diagram below summarizes the approach which can be summarised as follows: The XML file containing the C@merata task is processed by XML-processor to get the question string; The question string goes to input of the query recognizer; There it passes through a large number of regular expressions; If no match is found, the operation is stopped and KIAM system does not create a <passage> tag; Otherwise the request is sent to the music passage search engine.



Each regular expression of the query recognizer corresponds to a different function in the music passage searcher. So, depending on the matched regular expression, it calls the appropriate function. This function takes as input a given MusicXML file and searches for the appearance of a certain tag sequence. The music passage searcher returns the location in the score of the requested musical feature. Finally, this location is converted into <passage> tag attributes.

Here is a diagram of query recognizer:



First of all it gets an input string from the XML-processor. Then the end of the string is checked for the presence of one of the following data:

- instrument,
- measures,
- time signature,
- number,
- section,
- clef,
- beat.

This paper will consider only the first two checks because these worked the best on the C@merata 2016 task data.

To simplify the problem it is considered that this data (instrument and measures) can only occur at the end of string. It is also considered that only one instance of this data can occur in the string. All found data are stored in the resulting object. It is an object that stores all the data needed to find music entities in the MusicXML file.

The remaining part of the string is checked for compliance with one of the two patterns:

- string contains 'followed by',
- elements in the string are divided by commas.

This information is also stored in the resulting object. Melodic elements, coming one after the other, are considered separately in the next step.

Originally, the KIAM system should have also been checking synchronous elements but this work was not completed in time for this year's task.

The last action is recognition of a note or rest. A note's attributes, such as pitch, duration and accidentals, are saved in the resulting object.

At the end, the resulting object sends the entity sequences to the music passage searcher. The music passage searcher processes a given MusicXML file. In the file it searches for occurrences of data received from the query recognizer. These occurrences pass through the XML-processor to get an answer XML file.

2.2 Regular Expressions

Regular expressions used for question recognition are listed below. They can be divided into eight types of recognition:

- instrument
- measures
- sequence ('followed by')
- repeats
- type of note/rest
- dotted note definition
- rest recognition
- accidentals, pitch and octave

Regular expressions are used in the order as listed above. If a regex matches the query, data received from the regular expression are saved into the resulting object. Then the matching string is removed from the query. Regular expressions are used in a specific sequence. So, there cannot be a situation where multiple regular expressions match a particular query.

The regexes were derived manually from the C@merata 2016 task description.

The following examples describe what KIAM's function returns using regular expressions in specific cases.

Instrument

1) '/(?: in the| in)
(first|1st|second|2nd|third|3rd|fourth|4th|fifth|5th|sixth|6th|seventh|
7th|eighth|8th|ninth|9th|tenth|10th) ([A-Za-z]+)$/'

2) '/(?: in the| in) ([A-Za-z]+) part$/'

These regular expressions recognize such descriptions of musical instruments as:

- G followed by Eb in the viola part
- (returns 'viola')
- dotted quarter note D6 in the first violin
- (returns 'violin I')

Measures

3) '/(?: in the| in) (?:measures|bars) ([0-9]+)[\s]*-[\s]*([0-9]+)$/'

4) '/(?: in the| in) (?:measure|bar) ([0-9]+)$/'

These regular expressions recognize information about measures such as:

- quarter-note rest in measures 1-5
- (returns an array of integers from 1 to 5)
- A#1 in bars 44-59
- (returns an array of integers from 44 to 59)

Sequence

5) '/(.+) (?:followed by a|followed by) (.+)/'

6) '/(.+),[\s]*(.+)/'

The first regex recognizes the string 'followed by', as in the following examples:

G followed by Eb
(returns an array of strings 'G' and 'Eb'),

5 B4s followed by a C5
(returns an array of strings '5 B4s' and 'C5').

The second regular expression recognizes a sequence of strings separated by commas, as in the following examples:

Bb3, A3, G3, F3, E3
(returns an array of strings 'Bb3', 'A3', 'G3', 'F3' and 'E3').

Repeats

7) '/repeated (one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight
|8|nine|9|ten|10) times/'

8) '/(one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight|8|nine|9|ten|
10) ([A-Ga-g]#[1-9])s/'

9) '/(one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight|8|nine|9|ten|
10) ([A-Ga-g]b[1-9])s/'

10) '/(one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight|8|nine|9|
ten|10) ([A-Ga-g][1-9])s/'

11) '/(one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight|8|nine|9|
ten|10) ([A-Ga-g]#)s/'

12) '/(one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight|8|nine|9|
ten|10) ([A-Ga-g]b)s/'

13) '/(one|1|two|2|three|3|four|4|five|5|six|6|seven|7|eight|8|nine|9|
ten|10) ([A-Ga-g])s/'

These regular expressions recognize repeats:

quarter-note A repeated four times
(returns an array of string 'quarter-note A' repeated four times),

5 B4s
(returns an array of string 'B4' repeated five times).

Type of note/rest

14) '/(whole-note|whole note|whole|half-note|half note|half|
quarter-note|quarter note|quarter|crotchet|eight-note|eight
note|eight)/'

This regex recognizes type of note or rest:

quarter-note rest
(returns 'quarter'),

dotted quarter note D6
(returns 'quarter'),

crotchet D6
(returns 'quarter').

Dotted note definition

15) '/dotted/'

This regex checks if note is dotted:

dotted quarter note D6
(returns true).

Rest recognition

16) '/rest/'

This regex recognizes a rest:

quarter-note rest
(returns true).

Accidentals, pitch and octave

17) '/([A-Ga-g])#([1-9])/'

18) '/([A-Ga-g])b([1-9])/'

19) '/([A-Ga-g])([1-9])/'

20) '/([A-Ga-g])#/'

21) '/([A-Ga-g])b/'

22) '/([A-Ga-g])/'

These regular expressions recognize properties of notes:

Eb
(returns an array ['pitch' => 'E', 'octave' => 1, 'flat' => true]),

Bb3
(returns an array ['pitch' => 'B', 'octave' => 3, 'flat' => true]),

A#1
(returns an array ['pitch' => 'A', 'octave' => 1, 'sharp' => true]).

2.3 Search Functions

Search functions correspond to the regular expressions outlined above. First, an executed function searches an appropriate part. It is a part of an instrument received from the query recognizer.

Next, a function searches for the correct measures of the part. They are used in the next iteration.

The search function then compares each note in the measures with the first note of the list returned by the query recognizer. It is repeated until a match is found, whereupon, the corresponding note's position is stored. It is considered to be a starting point in the score associated with the question.

The following note of the list is then verified in the same way. Notes found in this way should follow directly after each other.

When a match is found for the last note in the list, the note's position is again stored. It is the end point in the score associated with the question.

The search continues until all passages associated with the question are found.

2.4 Example

Here is a worked example based on the regular expressions and general search algorithm outlined above. Consider the input question:

'G followed by Eb in the viola part'

The end of the string matches regular expression #2:

'/(?: in the| in) ([A-Za-z]+) part$/'

Instrument 'viola' is saved in the result object. The question is now reduced to:

'G followed by Eb'

None of the 'measures'-type regex matches the question string. However, the string matches regular expression #5:

'/(.+) (?:followed by a|followed by) (.+)/'

This regex divides the query string into two parts:

'G',
'Eb'

String 'G' matches regular expression #22:

'/([A-Ga-g])/'

String 'Eb' matches regular expression #21:

'/([A-Ga-g])b/'

So, the result object includes the the following information:

instrument — 'viola'
measures — all
first note — G
second note — E, flat

The music passage searcher starts by searching the different parts of the score. After finding the viola part other parts are not considered.

The searcher then goes through all the notes of the viola part. A note's position is stored when the current pitch matches 'G'.

If next note matches 'Eb', its position is stored as the end point in the score associated with the question. Then, the music passage searcher returns the location in the score of the requested musical feature. In order to do this, the location is first converted into <passage> tag attributes, according to the rules of the C@merata task.

Otherwise, the search continues again from note 'G'. This is repeated until either a correct answer is found or the parts end.

2.5 Implementation of the System

The PHP programming language was used to implement the system. It supports regular expressions without the use of additional libraries and also PHP has a built-in functionality to parse XML. In future, PHP will allow us to create a Web-interface for the KIAM system.

3. RESULTS AND DISCUSSION

The KIAM system was designed to try the suitability of regular expressions for text recognition, working in a domain witha limited vocabulary.

The system was able to recognize pitch, octave and accidentals for each note. These cases were handled correctly if references to notes in a query followed one after the other separated by commas or by 'followed by'.

KIAM was also able to determine musical instruments and measures.

Regular expressions were created only for the simplest queries. So, KIAM recognized correctly only seven requests such as:

- 'G followed by Eb in the viola part',
- 'quarter-note rest in measures 1-5',
- 'Bb3, A3, G3, F3, E3',
- 'dotted quarter note D6 in the first violin',
- 'quarter-note A repeated four times',
- '5 B4s followed by a C5',
- 'A#1 in bars 44-59'.

Unfortunately, the C@merata questions were very difficult this year, so the results are not as good as we had hoped. The actual scores for the KIAM system for all questions presented in the table below:

BP	BR	BF	MP	MR	MF
0.194	0.011	0.021	0.613	0.035	0.066

The best system this year was DMUN which score BF 0.070 - a very low figure reflecting the difficulty of the task. The second system was KIAM with BF 0.021 as shown in the table above. As would be expected, MF for KIAM was higher at 0.066 - generally it is easier to determine the correct measure than the exact beat in the measure. It is interesting that MP was 0.613 which means that for some queries, the correct measure was identified quite accurately.

Considering the KIAM performance across different question types, the results for 1_melod, n_melod and follow queries are shown below.

	BP	BR	BF	MP	MR	MF
1_melod	0.273	0.044	0.076	0.727	0.118	0.203
n_melod	0.000	0.000	0.000	0.333	0.021	0.040
follow	0.600	0.140	0.227	1.000	0.233	0.378

We can clearly see that 1_melod questions showed the best performance with BF 0.076. The MP was also the highest for these, showing that KIAM can find the measure for a note but not its exact beat. MP overall is high because it is high here, and in addition, KIAM did not answer many of the queries overall.

For n_melod queries, the correct beat was not recognised for any query (BF 0) but the correct measure was in some cases (MF 0.040).

Queries of type follow were complex in this task, but KIAM was able to tackle some of these, resulting in relatively high BF 0.227 and MF 0.378.

4. CONCLUSION

This was our first year of particating at C@merata; it is an extremely complex task and it took as quite a while to get a basic system running, especially as we chose to work in php and not to use the Python Baseline System provided by the organisers. Our approach was based on regular expressions and this proved surprisingly succesful for the simpler queries. Next steps will include refining our regular expressions via a detailed analysis of the Gold Standard data, and combining this approach with others more suited to the more complex query types.

5. REFERENCES

[1] Asooja, K., Ernala, S. K., & Buitelaar P. (2015). UNLP at the MediaEval 2015 C@merata Task. In Proceedings of the MediaEval 2015 Workshop, Dresden, Germany, September 14-15 2015. http://ceur-ws.org/Vol-1436/Paper86.pdf.

[2] MusicXML http://www.musicxml.com/

[3] Mytrova, M. (2013). Music Information Retrieval Based On Automated Reasoning Methods. RuSSIR Young Scientist Conference, Kazan, Russia, September 16-20, 2013.

[4] Sutcliffe, R. F. E., Collins, T., Hovy, E., Lewis, R., Fox, C., & Root, D. L. (2016). The C@merata task at MediaEval 2016: Natural Language Queries Derived from Exam Papers, Articles and Other Sources against Classical Music Scores in MusicXML. In Proceedings of the MediaEval 2016 Workshop, Amsterdam, The Netherlands, October 20-21 2016. http://ceur-ws.org.

[5] Sutcliffe, R. F. E., Fox, C., Root, D. L., Hovy, E., & Lewis, R. (2015). The C@merata Task at MediaEval 2015: Natural language queries on classical music scores. In Proceedings of the MediaEval 2015 Workshop, Dresden, Germany, September 14-15 2015. http://ceur-ws.org/Vol-1436/Paper12.pdf.

[6] Tabolin, A., Mytrova, M., Korukhova, Y. (2012). Music Theory Ontology. Reasoning Web Summer School, Vienna University of Technology, Austria, September 3-8, 2012.