Describing Music with MetaGrammars

Simon Petitjean

Describing Music with MetaGrammars

Simon Petitjean

LIFO, University of Orleans

Abstract. This paper aims to show metagrammars are not only suited for large grammars development, but can also deal with other types of descriptions. We give the example of musical chords, that can be quite naturally described within the modular scope offered by eXtensible MetaGrammar.

1 Introduction

Metagrammars, introduced by [1], are known as a way to reduce the amount of work and time needed for the development and maintenance of large tree grammars for natural languages. The main concept of metagrammars is to factorize redundant information in order to minimize the description. XMG (eXtensible MetaGrammar) is a metagrammatical approach that has been used to describe the syntax and semantics of some languages (French, German, English), and a preliminary verbal morphology in Ikota [2], using tree formalisms (Tree Adjoining Grammars and Interaction Grammars). The modularity given by the scope allows it to be used for other purposes. In order to illustrate the extensibility of the tool, we chose to explore a quite different language, the one of music. The goal of this work is to generate, from an abstract description, a lexicon of musical chords. The choice of this grammar can be explained in different ways: the first is that we aim to prove XMG can process really different types of lexicons. Another interesting aspect about this work is that the rules involved in musical theory (rhythm, harmony,...) are really different from the ones we dealt with until now. Musical theory can be found in [3] for example. Nevertheless, if large scale grammars can cover a significant part of the language, it is hard to imagine that a grammar of musical sentences could do the same with a significant part of melodies. In a first part, we will present the metagrammatical tool used, then we will give the metagrammar of musical chords. Finally, we will conclude and give some perspectives.

2 The scope: eXtensible MetaGrammar

XMG [4] stands both for a metagrammatical language and the compiler for this language. The input of the tool is the metagrammar, that is to say the abstract description of the grammar, written by the linguist. The description is then compiled and executed by XMG to produce the lexicon.The metagrammatical language mainly relies on the notion of abstraction and the concepts of logical

programming. The core of the language is called the control language, and is defined as follows:

$$\begin{aligned} \mathit{Class} &:= \mathit{Name}[p_1, \ldots, p_n] \rightarrow \mathit{Content} \\ \mathit{Content} &:= \langle \mathit{Dim} \rangle \{ \mathit{Desc} \} \mid \mathit{var} = \mathit{Name}[\ldots] \mid \mathit{Content} \vee \mathit{Content} \\ &\quad \mid \mathit{Content}; \mathit{Content} \end{aligned}$$

Abstraction is made possible by classes, which associate a content to a name. Contents can be conjunctions (;) and disjunctions (|) of descriptions or calls to other classes. Descriptions are accumulated within a dimension, which allows to separate the different types of information (syntactic, semantic, morphological...). Every dimension comes with its dedicated description language, the next step is to create a new one, adapted to music.

2.1 A music dimension

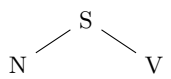
To describe trees, we use the following language:

$$\begin{aligned} \mathit{Desc} &:= x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \\ &\quad \mid x\ [f{:}E] \mid x\ (p{:}E) \mid \mathit{Desc}\ ;\ \mathit{Desc} \mid \mathit{SynExpr} = \mathit{SynExpr} \\ \mathit{SynExpr} &:= \mathit{var} \mid \mathit{const} \mid \mathit{var.const} \end{aligned}$$

The elementary unit we manipulate is the node. Nodes can be declared with feature structures for linguistic information (syntactic category, ...), and some properties. We also need to accumulate relations between the nodes, in order to describe the structure of the tree. $\rightarrow$ and $\prec$ are respectively the operators for dominance and precedence, $^{+}$ and * representing the transitive closure and the transitive reflexive closure. Unification of expression (=) can also be written. The '.' operator allows to access a variable when applied to a class and the value corresponding to a key when applied to a feature structure.

After being accumulated, the description has to be solved. The result is the set of all the minimal models of the description. For example, here is a XMG description and a tree fragment it models :

```
CanSubj ->
<syn>{ S [cat=s]; N [cat=n]; V [cat=v]; S->N; S->V; N<<V }
```



In this work, we will accumulate notes. The tree structures we already can describe seem to be adapted to order the notes. A note variable is declared as a node, with a feature structure which contains the information that characterizes it. Here, we will use the same language and just add an arithmetic operator

so that we can compare the values associated to notes. The music description language is consequently the same, except for the '+' in the expressions:

$$\begin{aligned} MDesc &:= x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \\ &\quad \mid x\ [f{:}E] \mid x\ (p{:}E) \mid MDesc\ ;\ MDesc \mid MExpr\ =\ MExpr \\ MExpr &:= var \mid const \mid var.const \mid MExpr\ +\ MExpr \end{aligned}$$

2.2 Principles

To ensure the well-formedness of the generated models, XMG comes with a library of principles that the user can activate. They consist in a set of linguistic constraints over the accumulated description. In the description of music, a principle could ensure that every measure is coherent with the time signature. Another principle could apply rewrite rules to chord sequences (as done in [5]) to generate a set of variations for those sequences.

2.3 Sharing data with other dimensions?

As it was the case for syntax and semantics, the accumulation produced by the execution of the metagrammar could take place in various dimensions, allowing to share data between them. One of the options would be to create a dimension for another clef: if we assume the grammar we produce is written in G-clef, another dimension could contain the transposition of the accumulation in F-clef. This can also be used for transposition to gather instruments tuned in different tones (for example B♭ clarinet and alto saxophone in E♭). Another dimension could also encode the fingerings to realize the notes (or chords) on some instrument. The accumulation of predicates in the semantic dimension is also an interesting perspective to experiment some theories on semantics of music.

3 A metagrammar for musical chords

3.1 Describing notes

The first abstraction we create is the elementary unit of music: the note. From a unique class, we want to generate any note, with the information needed to characterize it. The feature structure of the note contains typed data, types being checked during the compilation. The essential features we will use are the name of the note, its accidental (here we will only consider notes with at most one accidental), and its global height (in term of semi tones). As we will not describe rhythm in this preliminary work, we chose not to represent the duration of the note.

In the metagrammar, 'name' is the human readable name of the note. Its type is enumerated, with possible values {A,B,C,D,E,F,G}, 'acc' is the accidental of

the note (sharp, flat, or none), 'namerank' is an integer associated to the name of the note, beginning with A=1, 'height' is the number of semi tones from A to the note. 'namerank' and 'height' depend on the two previous features and are only used in the metagrammar to compare notes. For example, the feature structure for a D♯ will be [name=D, acc=sharp, namerank=4, height=5]. Scales are not taken into account, which means that the unique G we consider is both 7 semi tones above C and 5 semi tones below it.

Two things have to be constrained:
- the value and the value integer have to match
- the global height depends on the note value and its accidental

```
note ->
<music>{
   N [name=V, namerank=NR, acc=ACC, height=H];
  {
    { V=A ; NR=1 ; HT=1 } |
    { V=B ; NR=2 ; HT=3 } |
    { V=C ; NR=3 ; HT=4 } |
    { V=D ; NR=4 ; HT=6 } |
    { V=E ; NR=5 ; HT=8 } |
    { V=F ; NR=6 ; HT=9 } |
    { V=G ; NR=7 ; HT=11 }
  };
  {
    { ACC=sharp;    H = HT + 1 } |
    { ACC=flat;     H = HT - 1 } |
    { ACC=none;     H = HT     }
  }
}
```

XMG builds a description for each case it tries (if all unifications succeed). As expected, the execution of the class *Note* (with 7 notes and 3 accidentals) leads to 21 descriptions.

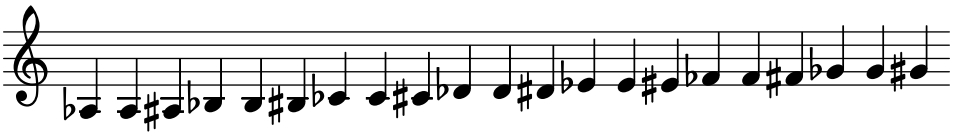


Fig. 1. The 21 generated notes

3.2 Describing intervals

An interval features two notes, and is defined by its number and its quality. The number is the difference between the staff positions of the notes, and the quality

is relative to the number of semi tones between the nodes. Thus the number is given by the difference between the name ranks of the notes, and the quality by the difference between the heights. To represent a major third, we have to accumulate two notes separated by exactly two tones. The class for this interval has to feature two notes, and two constraints : the name ranks of the notes have to be separated by two unities, and the global height by four semi tones. We add another node to the description, dominating the two notes, and holding the information about the interval into its feature structure.

```
Mthird ->
  First=note[];                Third=note[];
  FirstF=First.Feats;          ThirdF=Third.Feats;
  TRank=FirstF.namerank;       ThRank=ThirdF.namerank;
  THeight=FirstF.height;       ThHeight=ThirdF.height;
  FirstN=First.N;              ThirdN=Third.N;
  {   ThRank=TRank + 2     | ThRank=TRank - 5     };
  {   ThHeight=THeight + 4 | ThHeight=THeight - 8 };
  <music>{
  Root [third=major];
  Root ->+ FirstN;             Root ->+ ThirdN;
  FirstN >> ThirdN
  }
```

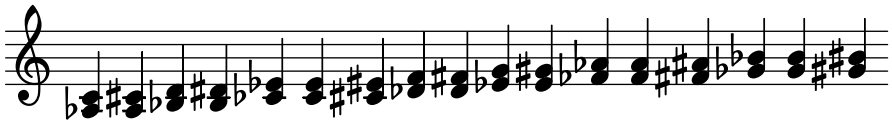


Fig. 2. The 17 generated major thirds

Intuitively, each one of the 21 notes should be the tonic for a major third, but only 17 major thirds are generated. The reason is the four remaining intervals involve double accidentals. We do the same for minor third, with an interval of three semi tones.

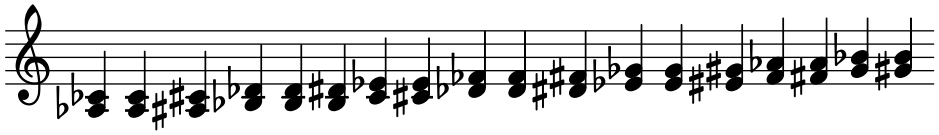


Fig. 3. The 18 generated minor thirds

3.3 Describing chords

Now, let us consider three notes chords, called triads, in their major and minor modes. A triad is composed of a note, called *tonic*, a third and a perfect fifth. The major chord features a major third, and the minor chord a minor one. We thus need to accumulate two intervals, and constraint their lowest notes to unify, the result being the tonic of the chord.

```
Major ->
  Third=Mthird[];
  Fifth=fifth[];
  Tonic=Third.First;
  Tonic=Fifth.First;
  Third.Root=Fifth.Root;
  ThirdN=Third.N;
  FifthN=Fifth.N;
  Root=Third.Root
```

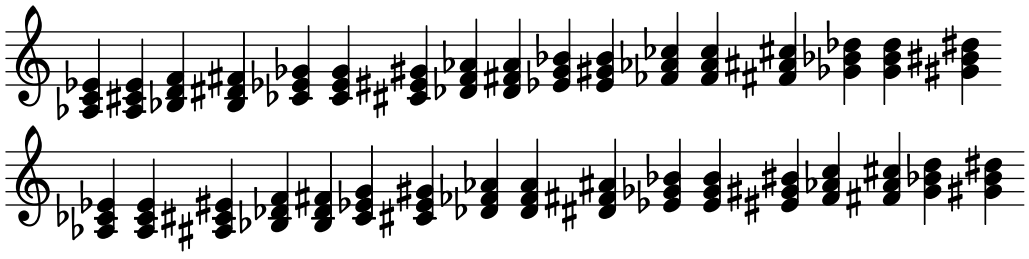


Fig. 4. The 17 generated major and minor triads

The same method can be applied to produce seventh chords: a seventh chord would then be defined as the conjunction of a triad and a seventh interval (the tonic of the triad unifying with the lowest note of the interval).

4 Conclusion and future work

We proposed a metagrammar for a part of musical theory: from a short abstract description, we generated lexicons of major and minor triads. The initial aim of this work was to prove XMG can quickly be extended in order to deal with really different theories (not necessarily linguistic theories). The metagrammatical approach appeared to be useful in the context of music description. The redundancy we factorized was not essentially structural, as in most of the cases studied previously, but also at the level of the algebraic constraints. This very preliminary work however needs to be extended to more complex musical phenomena. Describing melodies could obviously not be done in a similar way, generating every solution, but sequences of chords seem to be an interesting next step. Adapted principles could be designed and used to limit the number of models.

References

1. Candito, M.: A Principle-Based Hierarchical Representation of LTAGs. In: Proceedings of the 16th International Conference on Computational Linguistics (COLING'96). Volume 1., Copenhagen, Denmark (1996) 194–199
2. Duchier, D., Magnana Ekoukou, B., Parmentier, Y., Petitjean, S., Schang, E.: Describing Morphologically-rich Languages using Metagrammars: a Look at Verbs in Ikota. In: Workshop on "Language technology for normalisation of less-resourced languages", 8th SALTMIL Workshop on Minority Languages and the 4th workshop on African Language Technology, Istanbul, Turkey (May 2012)
3. Grove, G., Sadie, S.: The New Grove dictionary of music and musicians. Number vol. 13 in The New Grove Dictionary of Music and Musicians. Macmillan Publishers (1980)
4. Crabbé, B., Duchier, D.: Metagrammar redux. In Christiansen, H., Skadhauge, P.R., Villadsen, J., eds.: Constraint Solving and Language Processing, First International Workshop (CSLP 2004), Revised Selected and Invited Papers. Volume 3438 of Lecture Notes in Computer Science., Roskilde, Denmark, Springer (2004) 32–47
5. Chemillier, M.: Toward a formal study of jazz chord sequences generated by steedman's grammar. Soft Comput. **8**(9) (2004) 617–622