Project Proposal

Advanced Functional Programming

Joris ten Tusscher, Cas van der Rest, Orestis Melkonian

1 Domain

This project concerns the development of a library that aids programmers in the process of composing musical pieces. We will aim to do so by providing tools for a programmer to easily employ various techniques of algorithmic music composition through the usage of various DSLs.

Algorithmic Music Composition The notion of algorithmic music composition usually includes all methods in which music is computationally generated. Fundamentally, a piece of music consists of melody, rhythm and harmony, which take values from a finite domain (at least in western tonal music). Therefore, it is easy to see that computers may generate musical pieces by an algorithm process that decides which choices to make within these domains (e.g. which notes, when to play them, according to which harmonic rules).

Motivation We drew inspiration from a paper in the FARM workshop at ICFP 2017, which follows a linguistic approach to algorithmic music composition and presents a *categorial grammar* for music[1]. Unfortunately, the author only provides a primitive proof-of-concept implementation in Python, although it would fit perfectly in the context of strongly-typed functional programming. This is due to the fact, that categorial grammars are type-logical and lend themselves to semantics in the typed λ-calculus[2]. Hence, generating music reduces to the problem of *program synthesis*(i.e. generating valid λ-terms that conform to some specification).

2 Problem

The goal of the research is to create a Haskell package that can be used to formally describe music, generate music that satisfies certain constraints specified by the user, and export music to more universal formats.

2.1 Music-Representation DSL

Music representation will be possible through a strongly-typed DSL that can be used to formally describe music. It can be used to store information such as the notes present in the music piece, the musical dynamic throughout the piece (e.g. pianissimo or forte), or the key.

Since much of the needed functionality is already implemented in the Euterpea package[1], we will try to reuse as many features it offers as possible. Euterpea is a Haskell package developed primarily by Paul Hudak, that can represent/analyse music and thus derive properties, perform audio synthesis, and read and export MIDI (Musical Instrument Digital Interface) data. On top of that, we could use the Lilypond package[2] to render the music to traditional music scores.

2.2 Generation DSL

Apart from representing music, we would like to randomly generate it. To that end, we will provide a generation DSL, in order to make it possible to write custom generators. For the sake of convenience and usefulness, it should also be possible to map arbitrary values to musical ones (e.g. numbers $\mapsto$ notes).

[1] https://hackage.haskell.org/package/Euterpea
[2] https://hackage.haskell.org/package/lilypond

QuickCheck A possible approach would be to base our generation DSL on top of QuickCheck, by utilizing the Gen and Arbitrary typeclasses.

2.3 Constraint DSL

As the solution space defined by our categorial grammar alone is huge, searching for solutions exhibiting specific desired properties (e.g. melodies involving notes from a certain scale) would be computationally infeasible.

To remedy this, we will implement a DSL that will allow the programmer to naturally express constraints, which will be respected by the musical artefacts we generate; these will model musical properties such as restricted pitch range. As you would expect, these constraints will not be applied posthumously as a filter, but integrated in the generation process, effectively pruning the search space.

2.4 Applications

Apart from the above, we also aim to showcase the features of our library through several example applications:

Music Representation We will provide code snippets that demonstrate one's ability to write concrete music pieces using our DSL and to export them in MIDI format or music notation.

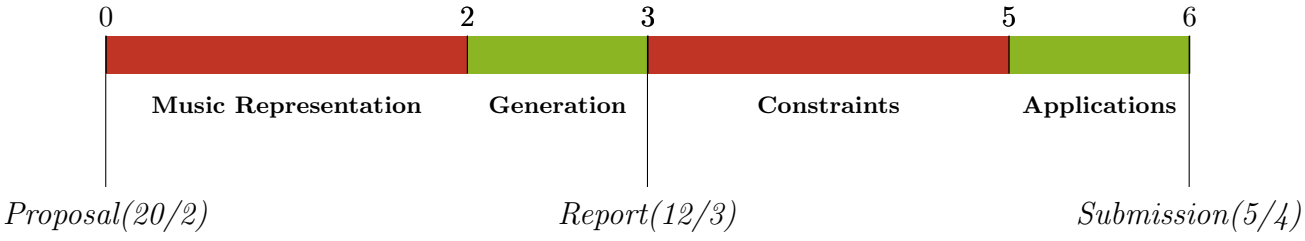
Generation We plan to implement several common generation techniques, such as creating melodies from chaotic/complex functions[3] and structuring pieces via an L-system grammar[4].

Constraints We will demonstrate how our library can be used for automatic generation of musical exercises, utilizing a variety of constraints.

An important property of our library that we wish to show through our examples, is that it is not geared specifically towards single-voice melodies, but can be used as easily to generate rhythm, harmony or anything combining these three principal elements of music. If time permits, we will also implement a simple web interface, which runs our library on the back-end and allows the user to select a number of pre-defined constraints in order to generate, for instance, musical exercises. Last but not least, the library will ship with its own "Prelude", providing common patterns/techniques for algorithmic music composition.

3 Planning

Below we give the estimated schedule across the six weeks available:



References

[1] H. Young, "A categorial grammar for music and its use in automatic melody generation," in *Proceedings of the 5th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design*, pp. 1–9, ACM, 2017.

[2] M. Moortgat, "Categorial type logics," in *Handbook of logic and language*, pp. 93–177, Elsevier, 1997.

[3] R. Bidlack, "Chaotic systems as simple (but complex) compositional algorithms," *Computer Music Journal*, vol. 16, no. 3, pp. 33–47, 1992.

[4] J. McCormack, "Grammar based music composition," *Complex systems*, vol. 96, pp. 321–336, 1996.