Project Proposal

Advanced Functional Programming

Joris ten Tusscher, Cas van der Rest, Orestis Melkonian

1 Domain

1.1 Algorithmic Music Composition

1.2 Generation Techniques

1.3 Motivation

[?]

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

The music-representation DSL will be an EDSL in Haskell 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 most users want to do more practical things with the music that is stored using the music-representation DSL, data written in the DSL can be converted to the Euterpea DSL[1]. Euterpea is a Haskell package developed by, among others, Paul Hudak, that can be used for music representation, but also has many other features than the music-representation DSL that will be developed for this project won't have. For example, it can analyse music and thus derive properties, perform audio synthesis, and read and export MIDI (Musical Instrument Digital Interface) data. Therefore, Euterpea would be a great tool for actually generating audio from the music that is described in our music-representation DSL, and it can also be used to export the information described in our EDSL to MIDI files. On top of Euterpea, we could use the Lilypond package[2] to export the music to nice traditional music scores.

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

2.2 Generation DSL

The generation DSL does not store actual music. Rather, it can be used for music composition. The generation can be done in multiple ways.

2.2.1 Chaos Functions

Chaotic functions generate completely different output as soon as their input changes by a minuscule amount. Therefore, they can be used to generate completely different music every time their input is changed by a fraction.

2.2.2 Lindenmayer Systems

L-systems [**?**] are a type of formal grammar invented at Utrecht University in 1968. They have an initial string and a set of production rules to expand this initial string into a longer string of symbols. The final string of symbols can be converted to something else, oftentimes trees. One can also convert the symbols to music however [**?**].

2.2.3 QuickCheck

Another way to generate music is to use QuickCheck. Using custom generator and arbitrary instances, it can be made possible to generate music that satisfies the information stored using the constraint DSL.

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 implement several applications, showcasing the features of our library:

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 and structuring pieces via an L-system grammar.

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: