Euterpea for Musicians, with an interface to Lilypond

Senior project by Isaac Reilly, advised by Donya Quick

1 Overview

Euterpea is a package for representing and manipulating music in the Haskell programming language. Lilypond is a program that takes a textual representation of music and transforms it into a beautiful printed score. My thesis has two parts: First, I will improve the representation of music in Euterpea and provide tools for working with music in a way more useful to composers and musicians. Second, I will provide a way to export music data from Euterpea to Lilypond. These efforts overlap because the representations I will develop for Euterpea will be informed by existing notions in Lilypond.

2 Motivation

Euterpea, as it currently works, can be frustrating and unintuitive to use. It represents an entire piece of music by an unstructured tree of parallel and sequential subtrees ending with the notes themselves, while music is often thought of as a set of parallel parts playing a generally sequential series of notes. Currently, one can assign an instrument to a music expression, but the set of instruments is simply the set of standard midi instruments. There is no notion of range or possible doublings, or notational concerns like transposition or preferred clefs. Concerning the music itself, there is no way to easily express performance annotations like articulation and grace notes.

There is also a unsettling contradiction in the set of notes available to use. The set includes notes which sound the same but are written differently, which is relevant for writing the music, but the built-in function to transpose music ignores this feature and always returns music containing only sharps, no flats. The notions of notes as simple pitches and as fitting into a certain key are both valid, but we should not mix them by accident. In fact, Euterpea theoretically supports multiple types of notes, kept segregated in different types of music which the user must deliberately convert between, but this capability is not currently used.

A final concern with Euterpea is that its only output is midi. Scores may be generated from midi with programs such as MuseScore, but that is labor-intensive and prevents algorithmic manipulation of the score in Euterpea after it has been exported. A Euterpea user might finish an algorithmic composition and then want to produce a written score and parts without to much complexity. Currently, if they want to use Lilypond, they would have to write the conversion to raw Lilypond code and then figure out how to organize a good Lilypond input file, which can be confusing.

Lilypond users interested in algorithmic composition will benefit from being

able to manipulate music in a representation that makes sense to them, and even writing Lilypond code in a Haskell source file and then working on it with the power of Haskell. Lilypond does offer the ability to write Scheme code directly, but the musical representation currently available in Lilypond is focused on preparing scores, not performing complex operations on music. One shortcoming that I frequently encounter in my work is when a line of music must appear in two places, slightly differently. If the two phrases were identical, I could simply assign a name to the expression and reference it twice. However, if I need to make manual changes to the music, I must copy and paste the code, making future changes to any common part of the music painful.

3 Features

The following is a list of target features to be implemented:

1 A music representation and a collection of note types such as abstract representations of notes in the 12-tone equal tempered scale, the 19-tone scale, notes with arbitrary pitch, special effects with no pitch, percussion with no pitch but a fixed set of sounds, notes with only a pitch class but no octave information. Also, a note type that represents a real playable note, with support for all performance annotations, including Lilypond commands. We won't "understand" arbitrary commands, but we can still allow their presence and ultimate placement in Lilypond code, allowing the user to take advantage of every feature Lilypond offers. This is not quite so simple because there must be representations for commands that act on a single note, commands that happen at a point in time, and commands that permanently change the state of the music from there on, which need to be handled carefully when performing complex operations, as well as commands which take a music expression as an argument.

This music representation will differ from lilypond code in that: pitches are not restricted to the equal-tempered scale; lyrics may be attached directly to notes; dynamics, slurs, and instrumentation will be specified for every note; and the music may include many parts jumbled together. There will also be tools to build a complete document with multiple scores, parts, and text.

2 A thorough model of an instrument. Functions will be available to check that the music is playable for that instrument and and if there are multiple instruments in one part, that the requirements to switch back and forth are practical.

3 Support for converting Euterpea music to Lilypond code. The rendering to Lilypond should handle multi-measure rests automatically, notate chords correctly, recognize tuplets and other complex rhythms based on their duration and notate them correctly, and add extra Lilypond engravers to a staff as needed, and handle the various representations of arbitrary Lilypond code. A Lilypond file containing parts and a score (concert or transposed) will be one rendering option, in addition to output modes which are incomplete in various ways but much faster for Lilypond to process.

4 Haskell lenses to easily select sections of music to modify and a large set of functions to modify them in useful ways: changing articulation, changing dynamics, repeating, applying articulations, applying rhythms, transposing, transposing within a scale, inverting, taking the retrograde, and of course the user may write any other function they dream up. Some of the supplied functions, such as applying articulation to a phrase and splitting a collection of notes into voices, may require significant computation, and I am investigating machine learning approaches. Machine learning will also be necessary to generate notation that is easy for a musician to read.

5 support for entering music in a Haskell source file by writing restricted Lilypond code in a Template Haskell Quasiquoter, for example `[ly| c'4-- d'8-. b-. c'2]`. Syntax for specifying arbitrary durations and pitches for these notes.

4 Possible Additions

Although easily generating scores from complex Euterpea music is the main goal of this project, it would be nice to support as much Lilypond code as possible in the Haskell source. Even nicer would be to read in an arbitrary Lilypond file to work with. Maybe we could read in Lilypond files that were written by this very program, even if not arbitrary files. The main obstacle is that Lilypond is not unambiguously parsable: an unknown command with no argument is recognizable as a command, but it might operate on the preceding note, the following note, or the remainder of the music in the expression. One option is simply a large catalog of commands and their behavior.

Other small additions that would be nice follow:

1 Cue notes should be included properly in the written score but not midi.

2 It would be good to support all note languages used in Lilypond. For example, in English, a B-flat is written in Lilypond as `bf`, but by default Lilypond uses a Dutch system that writes B-flat as `bes`.

3 There are notation quirks like putting an optional octave sign on a clef, octava marks (automatically generated?), and different kinds of grace notes.

4 Music in multiple time signatures at once, at multiple tempos.

5 Early music notation, which includes a note duration not available in modern music.

6 I also want to create a function to identify the key and adjust the midi output of notes to be perfectly in tune. The Lilypond midi output supports pitch bends, but dividing notes among the midi channels is a difficult problem which also makes all non-equal-tempered music difficult to convert to midi.

7 An interface with Kullita would be convenient.

5 Deliverables

- Code written in Haskell as a wrapper for Euterpea 2.0 (which is yet to be released).
- Documentation
- A tutorial on using my software. Maybe two tutorials: one for Haskell users and one for Lilypond users.