Heqet Report

By Isaac Reilly
advised by Donya Quick
senior project for the Yale University major in Computer Science

fall 2015

1 Overview

Looking over my original proposal, I'm pleased to find that I achieved my target features. There are still many more things I want to do with Heqet, and my goals have gotten higher as the semester progressed! However, I am pleased with my accomplishments so far, and I'm excited to continue this project in the spring.

2 How did it turn out?

The three deliverables I listed in my proposal were code, documentation, and a tutorial (maybe two). The code was supposed to be a wrapper for Euterpea 2.0, yet to be released. Euterpea 2.0 is still not released, but that's okay because I ended up writing a completely separate library that just accepts Euterpea data to convert to Heqet data. I don't have documentation beyond comments in my code, but I did write a tutorial with two parts for existing Euterpea and Lilypond users. Speaking of which, the tutorial contains a lot more concrete information about Heqet than does this report.

The proposal specified five main features:

- A music representation capable of handling Lilypond commands and all the different kind of notes, percussion, and lyrics, along with a set of different pitch types such as 19-tone equal temperament. My music representation meets this requirement. I have not created such a set of pitch types, beyond ordinary notes, percussion, and lyrics, but it is possible for the user to create them and use them in Heqet, as described in the tutorial.
- A thorough model of an instrument. Heqet instruments currently supply functions to check playability and assign clefs. Evaluation of instrument switches is not yet supported.
- Support for converting Euterpea music to Lilypond code. Heqet does this, and, in particular, chords and polyphony within a single staff are rendered correctly.
- Haskell lenses to easily select sections of music to modify and a large set of functions to modify them in useful ways. I wouldn't say I have a large set of functions, but otherwise Heqet meets this requirement.
- Support for entering music in a Haskell source file. Yep!

3 What I Learned

I've developed personally as a result of this project: I've learned that I'm capable of working through a large program, focusing on one piece at a time. I've learned about choosing priorities (with help from Professor Quick) among the many tempting features I wanted to implement. I've learned to accept that my hopes for Heqet were vastly more work than I could do in a semester, yet still take joy from what I have achieved so far. I've been delighted to learn about how productive it can be to have deep conversations with my mentor.

4 Haskell

Using Haskell, apart from being one of the main points of the project, turned out to be successful. For most of the semester, I had almost no bugs, not counting compilation errors, which are usually straightforward to fix. Recently, as I worked on features that were entwined with the rendering pipeline, I've been having some difficult issues, but overall I am very pleased with the ability of Haskell to catch errors at compile time. In addition, the strict type system has made it possible to refactor my code without fear that I might have "missed a spot".

Heqet makes use of advanced functional programming features like lenses and other optics, existential types, quasi-quoters, and parser combinators.

5 Future Goals

There is a lot more to do! I currently have 118 open issues on the Heqet github page, including goals such as: identify the key of the music and insert appropriate key changes automatically; render durations according to musical convention, including finding tuplets; support for dynamics, including recognizing crescendos and diminuendos by looking at the dynamics of individual notes and writing them with "hairpins"; producing a separate midi-optimized version of a score with notes grouped into midi channels by instrument, pizzicato changing midi instrument, compression to fewer midi instruments if there are too many to fit in the limited number of channels, pitch bends to accurately produce every pitch, and renditions of ornaments; and produce a set of parts for each instrument along with a concert or transposed score.

I'm considering refactoring my `Music` type, because, after implementing this version and working with it for a while, I've noticed a subtle problem. My current music type is a list of notes and other musical events. Each note occurs at a particular time, but there is no indication of when a segment of music starts and ends. Consider a phrase of music which has 10 measures of rest, a whole note, and another ten measures of rest. Currently, the rests may or may not actually be present in the list. Suppose they're not; then rendering this piece of music would only produce 11 measures. Worse, try pasting one of these phrases in sequence with another. The result will be a pair of adjacent whole notes 10 measure in. One solution to this problem is to simply mandate that rests must be present. Another is to change the type of the music data to be a list, as well as an explicit start and end time. I currently favor the latter as it avoids the messiness of filling in rests and it is expressed clearly in the type system…

My most audacious goal for Heqet is to create an application that I think of as a "musical spreadsheet", making use of the spreadsheet features we know and love such as generating some segments of music by applying functions to other segments, automatically updating these calculations, and offering easy copy and paste between places and, in music, times. I have visions of an interface to edit the history that generated a particular piece of music and the ability to "paint" properties onto notes, and of course generating a beautiful Lilypond score with a click. We'll see how much of *that* I get done next semester!