Heqet Tutorial

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

fall 2015

Contents

1 Heqet for Euterpea Users

Given a Euterpea music expression, creating a score is straightforward. You will need to import `Heqet` and `Heqet.Input.Euterpea` and then apply the following functions: First, to convert it to a Heqet music value, you should use `fromEu` or `fromEu1`, which take a Euterpea `Music Pitch` or a `Music Note1` respectively. Now you can print a score to stdout with the functions `quickScore` or `quickLine`. `quickScore` produces a piano rendition, while `quickLine` writes a single staff.

The reason that these functions perform IO is that the simplest way to use Heqet is to write a Haskell program which outputs your Lilypond code and pipe it directly into Lilypond, which you can do by running the included `heqet.sh` bash script with your Haskell program filename as an argument. Alternatively, just load your program into GHCi and paste the output code into a Lilypond file.

Now, let's try something more complicated. Suppose you have a score that has multiple instruments and you want to write them on separate staves with appropriate labels. If your Euterpea music value has instrument modifiers, you can just use `writeScore` for your output function, since Euterpea instruments are converted automatically. If you don't have instrument modifiers, or if your instruments are beyond the basic set of instruments currently recognized, you'll have to assign instruments in Heqet. The tools to manipulate music are mostly lenses, best used with the `Control.Lens` package. Lenses and other "optics" are a way to modify portions of a larger piece of data, for example, changing the instrument assigned to some of the notes of a music value.

By the way, this is a good time to mention that in Heqet most properties of music are recorded on every single note they impact. So, every note has a field for its `Maybe Instrument`. This way, you can rearrange the music as much as you want without worrying about handling the transitions between states over the course of the music—if the instrument changes, then Heqet is supposed to automatically write this direction into the part.(Actually, this isn't implemented yet. An example that does work is handling slurs: a slurred is considered to be an articulation on all but the last note in the slur. If you cut a slurred phrase in half, both halves will stay slurred correctly.)

Suppose we have a Heqet music value `opus`. Let's make it played by an oboe:

```
opusWithInstruments = opus & traverse.val.inst .~ Just oboe
```

Quick explanation of these operators: `traverse.val.inst .~ Just oboe` is a function that can manipulate some music. `&` is a `flip`ed `$`. So we're applying this function to `opus`. In the middle is the "optic", or in this case a composition of three: `traverse`, `val`, and `inst`. `traverse` looks at every note in the music, `inst` looks at the instrument of a note, and `val` is an implementation detail that will be hidden in future versions of Heqet. The `.~` operator means to set the value revealed by the optic, in this case setting the instrument field of every note to be `Just oboe`.

Suppose that all but one instrument was recognized automatically. Then, we could set all the notes with an unknown instrument to be an oboe like this:

```
opusWithInstruments = opus
    & instKind "Unknown".traverse.val.inst .~ Just oboe
```

`instKind` is a function that takes a string and produces a lens that looks at notes which have an instrument of that kind. The unknown instrument is a real instrument, not just a value of `Nothing` in the instrument field, because we need to know how to write its name in a score, what clefs it uses, what range it has (unlimited), etc.

If you try these examples, you might notice a problem: all the music is still on one staff! That's because staff assignments are separate from instrument assignments. To assign a staff of `"3"` (yes, that's a string) to the Bari Sax, we write:

```
opusWithInstruments = opus
    & instKind "Bari Sax".traverse.val.line .~ Just "3"
```

Heqet provides lenses to select music by start and end time or by any predicate on a single note.

To output a score with correct note durations, you need to assign a meter. The meter is one of the rare musical properties that is not stored in the notes themselves. Rather, a meter is a set of notes which, instead of having pitches, have measure and beat events. Assigning a meter to a segment of music simply means inserting these notes, which can have a staff just like any other note. These events can be sliced and moved around with their music, and Heqet will analyze them to determine what meter they specify and where meter changes need to be inserted into the printed score. To make a piece of music in common time, write:

```
opusWithInstruments = opus
    & assignMeter m4_4
```

Even though `assignMeter m4_4` does not use a lens, I find it natural to apply it with `&`, since often I want to take a music value and apply many functions to it, writing each function on a separate line beginning with the `&`.

(note: meter rendering has developed a bug which I don't have time to fix as I have stopped coding to write up my report and tutorial. Meters now simply don't appear in the rendered code.)

2 Heqet for Lilypond Users

So you want to do some fancy computer processing of your music, but you're not hot about Scheme, or at least are frustrated with the limited tools for working with Lilypond music data? You've come to the right place! However, this tutorial will not teach Haskell.

2.1 Note input

In order to enter music into Haskell source code using the Heqet DSL, you need to enable the "quasi-quoting" optional language feature by either putting `{-# LANGUAGE QuasiQuotes #-}` at the top of your source file or by typing `:set -XQuasiQuotes` into GHCi. Then import Dutch or English note entry with

```
import Heqet.Input.Dutch
import Heqet.Input.English
```

Now you can create Heqet music values in Haskell by writing them enclosed by "`[music|`" and "`|]`", like so:

```
[music| c4 d8 e f2 |]
```

2.2 Differences from Lilypond input

There are several differences between the Heqet note-input domain-specific language and Lilypond input, for a variety of reasons, including philosophical differences, features Heqet provides which are not in Lilypond, features which would be troublesome to implement in Heqet, and features which I simply haven't gotten around to yet.

- You can enter notes with any rational duration with the `\d` syntax, for example `c\d 4/5` to make a note with a duration of 4/5 of a whole note. You can omit the denominator if it's 1. This is currently the only way to enter notes of the durations needed for a tuplet.
- You can also enter notes with any frequency you like by writing the pitch as a number followed by `hz`, for instance `[music| 234.01hz4. |]`
- You can modify the pitch of a note with the command `\cents` followed by a number, possibly negative. This is currently the only way to get a quarter-tone pitch.
- The only languages you can currently use are Dutch and English.
- At the moment, slurs must be entered by putting a `-(` articulation on every note but the last.
- You cannot enter clefs, keys, or time signatures. I might implement time signatures, but Heqet will never support keys or clefs in note entry, since these are matters of how music is presented to the musician, not fundamental aspects of the music itself.
- Lilypond is actually difficult to parse because there is no syntactic difference between a command like `\fermata`, which follows the note it applies to, and one like `\xNote`, which proceeds its argument. In addition, many Lilypond commands change the state of all music that follows it, but not all do. Given the obvious usefulness of importing directly from Lilypond, in the future I will probably write some sort of Scheme tool to run in Lilypond and create a Heqet music value.

- Currently, only absolute note entry is possible.
- percussion notation `\phh` for hi-hat. When you enter notes, you don't need to specify that they must be rendered in a DrumStaff. They should be automatically rendered well, although support for percussion is currently minimal.
- no way to manually write beams, as this is not something the musician should need to worry about.

2.3 Making a score

2.4 Lilypond tweaks

3 Advanced topics