Workshop: creating computer music with Haskell

Anton Kholomiov (anton.kholomiov@gmail.com)

In the tutorial we are going to learn to create computer music with Haskell. We are going to use the library csound-expression [1]. It's a Haskell framework for sound design and music composition. It embeds very powerful audio programming language Csound into Haskell. It's a Csound code generator.

The Csound is 30 years old audio programming language that embodies the wisdom of many researchers. It has active community and it continues to evolve. It has the wide set of features (subtractive, granular, waveguide, sample-based synthesis, physical modeling, spectral processing, emulators for many analog synthesizer components). Csound is implemented in C as a compiler and as a library. It opens the doors to embedding it in all sorts of environments. It runs on all platforms (Linux, OS X, Windows), on devices (Raspberry Pi, Android, iOS) or even in the browser. But the language has very clumsy syntax and it's hard to code with Csound.

The Haskell library complements the Csound with expressive language. We can use the great low-level audio units of Csound and scheduler with higher level features of Haskell like higher order functions, rich data type system, body of libraries for advanced data types.

The library is not just an AST-builder. It has it's own functional model that hides away all Csound imperative syntax. The library design is inspired by Haskell standard libraries and it should feel natural for haskellers.

We are going to start with basic types: signals, constant numbers, strings and tables to store the waveshapes and audio files. Then gradually we are going to encounter how to create instruments and trigger an instrument with midi-devices (hardware or virtual), event streams, scores and UI widgets.

The library features the event-stream processing inspired with FRP and composable UI-widgets. We are going to look at how it's possible to compose UIs in applicative style.

The great feature of the library is that it works right in the ghci and it is designed in such a way that everything can be combined from simple expressions. Everything is an expression is an old motto that was a primary principle while building the library. User should be able define primitive expressions then use combinators to create more complex ones. The output of the library is instant so it makes it possible to use it in the REPL.

Almost all VST and hardware synthesizers come with the library of patches. The csound-expression also provides a user many instruments to play out of the box. They are defined in the package csound-catalog. It defines many standard instruments that can be used right out of the box. If you need to play an organ emulator you can load the modules `Csound.Base` and `Csound.Patch` in the ghci and type:

```
> dac (atMidi toneWheelOrgan)
```

and you are ready to perform with your midi-keyboard. There are many more patches you can check the full list in the module `Csound.Patch`. If you don't have the midi-device you can check out things with virtual midi-keyboard. we need to substitute the `dac` with `vdac` (Virtual Digital to Analog Converter). You can tweak the defaults if you need but for quick sketches the default behavior should just work.

We will explore some techniques for creation of musical tracks. We are going to use sample based approach spiced with some unusual synthesizers and effect processors. There is a library `csound-sampler` that implements composable API for samples (segments of audio signals). We are going to learn how to create tracks out of set of audio files.

Preliminaries

The audience should be able to follow the instructions and listen to results on their own laptops. The library has minimal haskell dependencies and should be easy to install. All components of the library can be installed from Hackage. You can install the package `csound-expression` if you need the basic functionality, you can install `csound-sampler` to be able to compose musical tracks with samples or you can install `csound-catalog` if you want to use standard instruments and install some additional features.

The library generates the code for Csound, stores it in temporal file and invokes the csound command line utility on it. So to run it you need to have Csound installed on your computer and `csound` command line should be executable (you need to add it to PATH). it's easy to check after installation. Just open up the command line, type `csound` and see if it works.

Csound runs on all platforms so please visit the official site [5] to install the Csound before the workshop.

So to follow along we need two things:

- Install csound-expression [2] and csound-sampler [3] and csound-catalog [4] from Hackage
- Install Csound from official site [5] and make the csound command line utility executable.

You can take the headphones with you to hear the output better and not to disturb your neighbors.

Csound-expression in action

You can listen to some tracks that were made with the library on soundclud. There are purely electronic tracks [6] and tracks where library was used to complement the live performance [7]. Also instruments from csound-catalog were used to create a virtual synthesizer called tiny-synt [8]. The UI is written in Python but the audio engine was generated with Haskell. You can find source code for some tracks here [9].

[1] Library github repo: https://github.com/spell-music/csound-expression

[2] Hackage homepage for csound-expression: http://hackage.haskell.org/package/csound-expression

[3] Hackage homepage for csound-sampler: http://hackage.haskell.org/package/csound-sampler

[4] Hackage homepage for csound-catalog: http://hackage.haskell.org/package/csound-catalog

[5] Csound website: http://csound.github.io/

[6] Examples of music: https://soundcloud.com/anton-kho/

[7] Examples of music: https://soundcloud.com/kailash-project

[8] Synthesizer made with the lib: https://github.com/anton-k/tiny-synth

[9] Source code for pieces: https://github.com/spell-music/csound-bits/tree/master/pieces