Functional Reactive Audio Generator for Modular Synthesizers

George Giorgidze
Chalmers University of Technology
Project Report for Topics in Computer Languages course
Personal Number: 8409248070
E-mail: giorgidz (at) student.chalmers.se

May 12, 2007

1 Initial Goal

Project's initial goal was to develop further already existing library for digital audio processing. The library was developed by me during the Advanced Functional Programming course project. It is called ELDAP, Embedded Language for Digital Audio Processing. It is Haskell library with basic audio processing functionality, like audio importing/exporting, synthesis and processing.

2 Shift to Slightly Different Task

I have started with audio synthesis. Basic synthesis capabilities were already provided in the ELDAP, for example generating the noise, creating harmonic oscillator or importing the audio from file. But I have started thinking about event based audio synthesis, which is somewhat different than audio processing. Audio processing is just simply taking the digital audio and producing the new Audio. That is why all of the functions in ELDAP are having the type

$$Audio \rightarrow Audio$$

So called event based audio synthesis is not based only on existing digital audio data, generated audio data depends also on discrete events for example: pressing key on keyboard, message on MIDI channel and etc.

Lets have a look how data structure defining audio looks in ELDAP library:

$$
\begin{array}{l}
\textbf{type}\ \mathit{Sample} = \mathit{Double} \\
\textbf{type}\ \mathit{Stream} = [(\mathit{Sample}, \mathit{Sample})] \\
\textbf{data}\ \mathit{Audio} = \mathit{Audio}\{\ \mathit{chanelNumber} :: \mathit{Int} \\
\quad ,\mathit{bitsPerSample} :: \mathit{Int} \\
\quad ,\mathit{sampleRate} :: \mathit{Int} \\
\quad ,\mathit{sampleNumber} :: \mathit{Int} \\
\quad ,\mathit{stream} :: \mathit{Stream} \\
\quad \}
\end{array}
$$

So audio is just record containing list of samples, this kind of representation is fairly enough to deal with audio processing tasks. Audio data is modelled as an infinite lazy list (stream) of samples, all processing tasks are accomplished by applying list functions such map, zip, fold and etc. Processing functions are just lazy computations and actual computation takes place when the values of this lazy stream are needed, in particular when exporting it to file.

If we consider this kind of model, we will notice that audio processing is just applying function to continuous stream and returning modified continuous stream (of course in practice stream is discrete with some sampling rate which defines interval of time between stream samples).

Now if we consider so called event based audio synthesis this kind of model is not satisfactory anymore. Now output depends not only on continuous input but also on discrete events, also received as an input. So in this case we need some kind of hybrid model, to capture behaviour mentioned above.

Actually if one considers audio processing library, it is not its primary purpose to deal with this kind of systems, but I became interested as it is related to topics of my future research. My Master and PhD theses are going to be about implementation of new programming language capable to model and simulate highly dynamic hybrid physical systems, systems containing a combination of both continuous and discrete components and subject to change its structure over time. Compilation of initial ideas are provided in [7].

I thought that it would be good starting exercise, to model hybrid system in existing functional language.

I found area very interesting especially from functional programming point of view. Luckily there are already lots of things done in functional programming research community, so I decided to gain insights in it. I noticed that programming paradigm called Functional Reactive Programming perfectly suits the tasks I have described above. Because Functional Reactive Programming is a paradigm for programming hybrid systems, systems containing both continuous and discrete components. FRP accomplishes it's goals in a high level and declarative manner. The key concepts of Functional Reactive Programming are its notions of continuous time varying values and discrete time ordered sequences of events [3].

3 Functional Reactive Programming

Functional Reactive Programming was first used in Fran [2], domain specific language for functional reactive animations. After it was used in several domains such robotics [5], computer vision [9] and graphical user interfaces [1]. Besides the fact that lots of research and development efforts where pushed in this area, advances went unnoticed by modelling community[7].

Ideas in functional reactive programming have evolved and culminated in what now is called as YAMPA [6] or sometimes refereed as AFRP (Arrowized Functional Reactive Programming). It is again domain specific language embedded in Haskell. It's main feature is heavy usage of arrows [4], Functional Reactive Programming concepts are represented and described by using arrows.

The most important concept which underlies FRP is the concept of signal. Signal is continuous time varying value. Concept is encoded in Haskell as a polymorphic type:

$$Signal\ a = Time \rightarrow a$$

But in YAMPA signals are not allowed as first class values, there are several reasons for that please have a look [6]. Programmer has access only to signal functions.

$$SF\ a\ b = Signal\ a \rightarrow Signal\ b$$

The type `SF` is defined as an instance of class `Arrow`. So YAMPA program is composition of possibly large number of signal functions into a one signal function which is than "executed" by suitable interpreter on top level.

OK but how to deal with discrete events? This problem in YAMPA is solved in a rather simple manner.

$$Signal\ (Event\ b)$$

is called event stream, and is signal that at any point of time. It yields either nothing or an event carrying value of type `b`. As one can notice `Event` type is similar to `Maybe` type but it is defined as an abstract type in YAMPA, much like SF which is also abstract. But rich set of useful functions and combinators are provided to deal with them.

So no we can have

$$SF\ a\ (Event\ b)$$

which generates an event stream.

Special switching constructs are available in YAMPA to deal with discrete events, they are providing capabilities to switch between different configurations at run time depending on discrete events, this is exactly what I needed. Also one good thing with YAMPA and with arrows in general is that new Ross Paterson's arrow notation [8] can be used , which is already implemented in new version of GHC (Glasgow Haskell Compiler) and works pretty well.

Another addvantage of YAMPA when modelling systems is its dynamicity, structure of the model which governs the process can be changed using switching constructs. I mean on can add or remove components to model at runtime. So it is very dynamic. As conclusion of this section I can say that I found YAMPA very handy and expressive enough to model hybrid and dynamic systems, but there are some limitations which were not relevant for this project, for more details have a look [7].

4 Final Result

So I get started. At first I have defined the goal, I tryed it to be interesting and useful enough to apply functional reactive programming techniques I started to implement Audio Generator for Modular Synthesizers. Which based on received events generates sound of particular instrument. Also another motivation for this project was the fact that techniques from Functional Reactive Programming was not experimented on audio related tasks (at least I was not able to find).

First of all I learned programming with arrows, what arrows really are and how they can be used in Haskell. After I learned principles of Functional Reactive Programming and how to use YAMPA, latest implementation of FRP ideas.

I have used two already existing libraries, latest code snapshot of YAMPA and I also reused my ELDAP library's importing and exporting functionality. Import is used to import instrument waveforms from external file and exporting functionality is used to output generated audio into external file. I also used used waveforms of several Georgian traditional instruments which where recorded by me few years ago.

Here is how I have defined synthesizer:

$$
\begin{aligned}
&\mathit{synth} :: \mathit{Frequency} \to \\
&\quad \mathit{Time} \to \\
&\quad \mathit{SoundSFCreator} \to \\
&\quad [\mathit{FilterSF}] \to \\
&\quad \mathit{FilterSF} \to \\
&\quad \mathit{SF}\ (\mathit{Controllers}, \mathit{Event}\ ())\ (\mathit{Sample}, \mathit{Event}\ ())
\end{aligned}
$$

Basically it is function, which when given appropriate arguments returns signal function which outputs audio which in turn depends on received discretete events and continuous input. I have also provided handy functions to easily create filter signal functions of type `FilterSF`, and main sound generators `SoundSF` based on imported waveforms.

I will not go here into details of implementation, it is rather simple and amount of code is not so large, but understanding of arrow syntax and concepts of Functional Reactive Programming is necessary, especially understanding of semantics of switching constructs is very important.

Implemented audio generator can be used in any kind of modular event based sound synthesis, such MIDI players, MIDI keyboards and other synthesis based audio systems, for studying properties of audio instruments, experimenting and etc.

Compilation and usage instructions are provided in the source package which also ships latest versions of YAMPA framework and ELDAP library, even YAMPA which already exists for several years is not stable yet, and it is still under active research and development.

5 Conclusions

Working on project was very beneficial for me, I learned lots of useful things, such that programming with arrows, functional reactive programming, modelling of hybrid and dynamic systems in a declarative manner and etc. It was also continuous fun, despite discrete moments of trauma.

References

[1] Antony Courtney and Conal Elliott. Genuinely functional user interfaces. In *Proceedings of the 2001 Haskell Workshop*, September 2001.

[2] Conal Elliott and Paul Hudak. Functional reactive animation. In *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming (ICFP'97)*, volume 32(8), pages 263–273, New York, NY, USA, June 1997. ACM Press.

[3] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In *Summer School on Advanced Functional Programming 2002, Oxford University*, volume 2638 of *Lecture Notes in Computer Science*, pages 159–187. Springer-Verlag, 2002.

[4] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1–3):67–111, 2000.

[5] Gregory Hager John Peterson and Paul Hudak. A language for declarative robotic programming. In *Proceedings of the International Conference on Robotics and Automation (ICRA '99)*, volume 2, pages 1144–1151, May 1999.

[6] Henrik Nilsson, Antony Courtney, and John Peterson. Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Haskell Workshop (Haskell '02)*, pages 51–64, Pittsburgh, Pennsylvania, USA, October 2002. ACM Press.

[7] Henrik Nilsson, John Peterson, and Paul Hudak. Functional hybrid modeling. In *Proceedings of PADL'03: 5th International Workshop on Practical Aspects of Declarative Languages*, pages 376–390. Springer Verlag LNCS 2562, January 2003.

[8] Ross Paterson. A new notation for arrows. In *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP'01)*, pages 229–240. ACM Press, September 2001.

[9] John Peterson, Paul Hudak, Alastair Reid, and Greg Hager. FVision: A declarative language for visual tracking. *Lecture Notes in Computer Science*, 1990:304–??, 2001.