The Arpeggigon: A Functional Reactive Musical Automaton

Demo, FARM 2017, 9 Sept., Oxford

Henrik Nilsson
Joint work with Guerric Chupin and Jin Zhan

Functional Programming Laboratory, School of Computer Science
University of Nottingham, UK
The Arpeggigon

- Software realisation of the reacTogon:
The Arpeggigon

- Software realisation of the reacTogon:

- Interactive cellular automaton:
  - Configuration
  - Performance parameters
The Arpeggigon

- Software realisation of the reacTogon:
- Interactive cellular automaton:
  - Configuration
  - Performance parameters

Before you get too excited: Work in progress!
The Harmonic Table

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>C#</th>
<th>D</th>
<th>D#</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td>A#</td>
<td>B</td>
<td>C#</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>A</td>
<td></td>
<td>G</td>
<td>G#</td>
<td>A</td>
<td>A#</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td></td>
<td>C#</td>
<td>D</td>
<td>D#</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>A#</td>
<td>G</td>
<td>G#</td>
<td>A</td>
<td>A#</td>
</tr>
<tr>
<td>F</td>
<td>D#</td>
<td>F#</td>
<td></td>
<td>C</td>
<td>C#</td>
<td>B</td>
<td>F#</td>
</tr>
<tr>
<td>D</td>
<td>G</td>
<td>F</td>
<td>G#</td>
<td>A#</td>
<td>D#</td>
<td>F#</td>
<td>G#</td>
</tr>
<tr>
<td>A#</td>
<td></td>
<td>E</td>
<td>C#</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>G</td>
<td>D#</td>
<td>E</td>
<td>F#</td>
<td>A#</td>
<td>G#</td>
<td>A</td>
<td>F#</td>
</tr>
<tr>
<td>D#</td>
<td>C</td>
<td>F</td>
<td>C#</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>D#</td>
</tr>
<tr>
<td>G#</td>
<td></td>
<td></td>
<td>A#</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>G</td>
</tr>
<tr>
<td>A</td>
<td>F#</td>
<td>D#</td>
<td>E</td>
<td>F</td>
<td>D#</td>
<td>G#</td>
<td>A#</td>
</tr>
<tr>
<td>F</td>
<td>C#</td>
<td>A</td>
<td>D#</td>
<td>B</td>
<td>E</td>
<td>F#</td>
<td>G#</td>
</tr>
<tr>
<td>C#</td>
<td>A#</td>
<td>B</td>
<td>C</td>
<td>C#</td>
<td>D</td>
<td>G</td>
<td>E</td>
</tr>
</tbody>
</table>

Minor Third |
Perfect Fifth |
Major Third |
Half-Step
Running a Sample Configuration
Motivation

Exploring FRP and RVR as an (essentially) declarative way for developing full-fledged musical applications:

- FRP aligns with declarative and temporal (discrete and continuous) nature of music
- RVR allows declarative-style interfacing with external components
Motivation

Exploring FRP and RVR as an (essentially) declarative way for developing full-fledged musical applications:

- FRP aligns with declarative and temporal (discrete and continuous) nature of music
- RVR allows declarative-style interfacing with external components

Rest of talk:

- Demonstration
- Implementation Highlights
Aspects of the Arpeggigon (1)
Aspects of the Arpeggigon (1)

- Interactive
Aspects of the Arpeggigon (1)

- **Interactive**
- Layers can be added/removed: *dynamic structure*
Aspects of the Arpeggigon (1)

- **Interactive**
- Layers can be added/removed: *dynamic structure*
- Notes generated at *discrete* points in time
Aspects of the Arpeggigon (1)

- **Interactive**
- Layers can be added/removed: *dynamic structure*
- Notes generated at *discrete* points in time
- Notes played *slightly shorter* than nominal length
Aspects of the Arpeggigon (1)

- *Interactive*
- Layers can be added/removed: *dynamic structure*
- Notes generated at *discrete* points in time
- Notes played *slightly shorter* than nominal length
- Configuration and performance parameters can be changed at *any* time
Aspects of the Arpeggigon (2)

Potential further enhancements, e.g.:

- Swing: alternately lengthening and shortening pulse divisions
- Staccato and legato playing
- Sliding notes
- Automated, smooth, performance parameter changes
Yampa

Something like *Yampa* a good fit:
Yampa

Something like *Yampa* a good fit:

- FRP implementation embedded in Haskell
Yampa

Something like *Yampa* a good fit:

- FRP implementation embedded in Haskell
- Supports:
  - *Signal Functions*: pure functions on signals
  - Structural change through *Switching*
  - *Hybrid* (continuous and discrete) time.
Yampa

Something like **Yampa** a good fit:

- FRP implementation embedded in Haskell
- Supports:
  - **Signal Functions**: pure functions on signals
  - Structural change through **Switching**
  - **Hybrid** (continuous and discrete) time.

- Programming model:
The Arpeggigon Architecture

- User
- GUI
- Common Control
- Layers
- MIDI Translator
- MIDI Synthesizer
- MIDI Keyboard
Cellular Automaton

State transition function for the cellular automaton:

\[
\text{advanceHeads} :: \text{Board} \rightarrow \text{BeatNo} \rightarrow \text{RelPitch} \rightarrow \text{Strength} \\
\rightarrow [\text{PlayHead}] \rightarrow (\[[\text{PlayHead}], [\text{Note}]\])
\]

Lifted into a signal function primarily using \text{accumBy}:

\[
\text{accumBy} :: (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow \text{SF} (\text{Event } a) (\text{Event } b) \\
\text{automaton} :: [\text{PlayHead}] \\
\rightarrow \text{SF} (\text{Board}, \text{DynamicLayerCtrl}, \text{Event BeatNo}) \\
(\text{Event } [\text{Note}], [\text{PlayHead}])
\]
Automated Smooth Tempo Change

Smooth transition between two preset tempos:

\[ \text{smoothTempo} :: \text{Tempo} \rightarrow \text{SF} \left( \text{Bool}, \text{Tempo}, \text{Tempo}, \text{Rate} \right) \text{ Tempo} \]

\[ \text{smoothTempo} \ tpo0 = \text{proc} \ (\text{sel1}, \text{tpo1}, \text{tpo2}, \text{rate}) \rightarrow \text{do} \]

\[ \text{rec} \]

\[ \text{let } \text{desTpo} = \text{if } \text{sel1} \text{ then } \text{tpo1} \text{ else } \text{tpo2} \]

\[ \text{diff} = \text{desTpo} - \text{curTpo} \]

\[ \text{rate'} = \text{if } \text{diff} > 0.1 \text{ then rate} \]

\[ \text{else if } \text{diff} < -0.1 \text{ then } -\text{rate} \]

\[ \text{else } 0 \]

\[ \text{curTpo} \leftarrow \text{arr} \left( +\text{tpo0} \right) \ll \text{integral} \leftarrow \text{rate'} \]

\[ \text{returnA} \leftarrow \text{curTpo} \]
The Arpeggigon interacts with the outside world using two imperative toolkits:
- GUI: GTK+
- MIDI I/O: Jack
Reactive Values and Relations (1)

- The Arpeggigon interacts with the outside world using two imperative toolkits:
  - GUI: GTK+
  - MIDI I/O: Jack
- Very imperative APIs: Hard or impossible to provide FRP wrappers.
The Arpeggigon interacts with the outside world using two imperative toolkits:
- GUI: GTK+
- MIDI I/O: Jack

Very imperative APIs: Hard or impossible to provide FRP wrappers.

Instead, we use *Reactive Values and Relations* (RVR) to wrap the FRP core in a "shell" that acts as a bridge between the outside world and the pure FRP core.
Reactive Values and Relations (2)

- A Reactive Value (RV) is a typed mutable value with access rights and subscribable change notification.
Reactive Values and Relations (2)

- A Reactive Value (RV) is a typed mutable value with access rights and subscribable change notification.
- RVs provide a uniform interface to GUI widgets, files, network devices, ... 

For example, the text field of a text input widget becomes an RV.
A Reactive Value (RV) is a typed mutable value with access rights and subscribable change notification.

RVs provide a uniform interface to GUI widgets, files, network devices, . . .

For example, the text field of a text input widget becomes an RV.

Reactive Relations (RR) allow RVs to automatically be kept in synch by specifying the relations that should hold between them.
System Tempo Slider

globalSettings :: IO (VBox, ReactiveFieldReadWrite IO Int)
globalSettings = do
    globalSettingsBox ← vboxNew False 10
    tempoAdj ← adjustmentNew 120 40 200 1 1 1
    tempoLabel ← labelNew (Just "Tempo")
    boxPackStart globalSettingsBox tempoLabel PackNatural 0
    tempoScale ← hScaleNew tempoAdj
    boxPackStart globalSettingsBox tempoScale PackNatural 0
    scaleSetDigits tempoScale 0
let tempoRV =
    bijection (floor, fromIntegral)
    'liftRW' scaleValueReactive tempoScale
return (globalSettingsBox, tempoRV)
Summary

- Yampa (FRP) good fit for writing interactive musical applications in a declarative way.
- Reactive Values and Relations proved very helpful for bridging the gap between the outside world and the FRP core in a fairly declarative way.
- Performance in terms of overall execution time and space perfectly fine; timing must be improved.
- Musical?

Code: https://gitlab.com/chupin/arpeggigon