ITU FRP 2010

Lecture 6:
Switched-on Yampa:
Programming Modular Synthesizers in Haskell

Henrik Nilsson and George Giorgidze

School of Computer Science
The University of Nottingham, UK
Modular synthesizers?
Modular synthesizers?
Modular synthesizers?

Steve Pocaro, Toto, with Polyfusion Synthesizer
Modern Modular Synthesizers
Where does Yampa enter the picture?
Where does Yampa enter the picture?

- Music can be seen as a hybrid phenomenon. Thus interesting to explore a hybrid approach to programming music and musical applications.
Where does Yampa enter the picture?

- Music can be seen as a hybrid phenomenon. Thus interesting to explore a hybrid approach to programming music and musical applications.
- Yampa’s programming model is very reminiscent of programming modular synthesizers:
Where does Yampa enter the picture?

- Music can be seen as a hybrid phenomenon. Thus interesting to explore a hybrid approach to programming music and musical applications.
- Yampa’s programming model is very reminiscent of programming modular synthesizers:

- Fun application! Useful for teaching?
What have we done?
What have we done?

Framework for programming modular synthesizers in Yampa:
What have we done?

Framework for programming modular synthesizers in Yampa:

- Sound-generating and sound-shaping modules
What have we done?

Framework for programming modular synthesizers in Yampa:

- Sound-generating and sound-shaping modules
- Additional supporting infrastructure:
  - Input: MIDI files (musical scores), keyboard
  - Output: audio files (.wav), sound card
  - Reading SoundFont files (instrument definitions)
What have we done?

Framework for programming modular synthesizers in Yampa:

- Sound-generating and sound-shaping modules
- Additional supporting infrastructure:
  - Input: MIDI files (musical scores), keyboard
  - Output: audio files (.wav), sound card
  - Reading SoundFont files (instrument definitions)
- Status: proof-of-concept, but decent performance.
Example 1: Sine oscillator

oscSine :: Frequency → SF CV Sample
oscSine \( f_0 \) = \text{proc } cv \rightarrow \text{do}

\text{let} \ f = f_0 \ast (2 \ast \ast \ cv) \\
\phi i \leftarrow \text{integral} \leftarrow 2 \ast \pi \ast f \\
\text{return} A \leftarrow \sin \phi i

\text{constant} 0 \ggg \text{oscSine 440}
Example 2: Vibrato

\[
\begin{align*}
\text{constant } 0 \\
\implies \text{oscSine } 5.0 \\
\implies \text{arr } (*0.05) \\
\implies \text{oscSine } 440
\end{align*}
\]
Example 3: 50’s Sci Fi

\[ \text{sciFi} :: SF() \text{ Sample} \]

\[ \text{sciFi} = \text{proc}() \rightarrow \text{do} \]

\[ \text{und} \leftarrow \text{arr}( \times 0.2) \ll \text{oscSine 3.0} \leftarrow 0 \]

\[ \text{swp} \leftarrow \text{arr}(+1.0) \ll \text{integral} \leftarrow -0.25 \]

\[ \text{audio} \leftarrow \text{oscSine 440} \leftarrow \text{und} + \text{swp} \]

\[ \text{returnA} \leftarrow \text{audio} \]
Envelope Generators (1)

\[ \text{envGen} :: CV \rightarrow [(\text{Time}, CV)] \rightarrow (\text{Maybe Int}) \]
\[ \rightarrow SF (\text{Event} ()) (CV, \text{Event} ()) \]

\[ \text{envEx} = \text{envGen} 0 [(0.5, 1), (0.5, 0.5), (1.0, 0.5), (0.7, 0)] \]
\[ (\text{Just } 3) \]
Envelope Generators (2)

How to implement?
Envelope Generators (2)

How to implement?

Integration of a step function yields suitable shapes:
Envelope Generators (3)

\[
\text{afterEach} :: [(\text{Time}, b)] \rightarrow \text{SF} \ a \ (\text{Event} \ b)
\]

\[
\text{hold} :: a \rightarrow \text{SF} \ (\text{Event} \ a) \ a
\]

\[
\text{steps} = \text{afterEach} \ [(0.7, 2), (0.5, -1), (0.5, 0), (1, -0.7), (0.7, 0)]
\]

\[
\ggg \ \text{hold} \ 0
\]
Envelope generator with predetermined shape:

\[
\text{envGenAux} :: CV \rightarrow [(\text{Time}, CV)] \rightarrow SF \ a \ CV \\
\text{envGenAux} \ l0 \ tls = \text{afterEach} \ trs \gg \text{hold} \ r0 \\
\gg \text{integral} \gg \text{arr} \ (+l0) \\
\text{where} \\
(r0, trs) = \text{toRates} \ l0 \ tls
\]
Envelope Generators (5)

Envelope generator responding to key off:

\[ envGen :: CV \rightarrow [(Time, CV)] \rightarrow (Maybe Int) \rightarrow SF (Event ()) (CV, Event ()) \]

\[ envGen l0 tls (Just n) = \]

\[ \text{switch (proc noteoff \rightarrow do} \]

\[ l \leftarrow envGenAux l0 tls1 \leftarrow () \]

\[ \text{returnA} \leftarrow ((l, noEvent), noteoff \text{ 'tag' } l)) \]

\[ (\lambda l \rightarrow envGenAux l tls2 \]

\[ \&\& \text{after (sum (map fst tls2)) ()} \]

\[ \text{where} \]

\[ (tls1, tls2) = \text{splitAt n tls} \]
Example 4: Bell

bell :: Frequency → SF () (Sample, Event)
bell f = proc () → do
    m ← oscSine (2.33 * f) −≺ 0
    audio ← oscSine f ←≺ 2.0 * m
    (ampl, end) ← envBell ←≺ noEvent
    returnA ← (audio * ampl, end)
Example 5: Tinkling Bell

\[
tinkle :: SF () Sample
\]
\[
tinkle = (\text{repeatedly } 0.25 84
\]
\[
\quad \Rightarrow \text{constant } ()
\]
\[
\quad \&\& \text{arr (fmap (bell } \circ \text{midiNoteToFreq))}
\]
\[
\quad \Rightarrow \text{rSwitch (constant 0))}
\]
Example 6: Playing a C-major scale

\[
scale :: SF () Sample
scale = (afterEach [(0.0, 60), (2.0, 62), (2.0, 64),
                   (2.0, 65), (2.0, 67), (2.0, 69),
                   (2.0, 71), (2.0, 72)]
\]

\[\ggg constant ()
   && arr (fmap (bell \circ midiNoteToFreq))
\]

\[\ggg rSwitch (constant 0))
   && after 16 ()\]
Example 7: Playing simultaneous notes

\[
mysterySong :: SF () \ (Sample, Event ())
mysterySong = \text{proc } _ \rightarrow \text{ do}
\begin{align*}
    t & \leftarrow \text{tinkle} \ (()) \\
    m & \leftarrow \text{mystery} \ (()) \\
    \text{return} A & \leftarrow (0.4 \times t + 0.6 \times m)
\end{align*}
\]
A polyphonic synthesizer (1)

Sample-playing monophonic synthesizer:
- Read samples (instrument recordings) from SoundFont file into internal table.
- Oscillator similar to sine oscillator, except table lookup and interpolation instead of computing the sine.

SoundFont synthesizer structure:
Exploit Yampa’s switching capabilities to:
- create and switch in a mono synth instance in response to each note on event;
- switch out the instance in response to a corresponding note off event.
Switched-on Yampa?
Switched-on Yampa?

Software and paper: www.cs.nott.ac.uk/~ggg