Switched-on Yampa

Programming Modular Synthesizers in Haskell

George Giorgidze and Henrik Nilsson

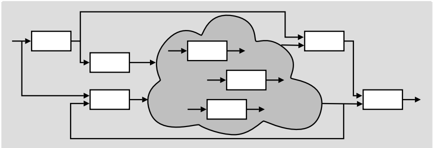
School of Computer Science

The University of Nottingham, UK

Modular synthesizers?

What is Yampa?

- Domain-specific language embedded in Haskell for programming ***hybrid*** (mixed discrete- and continuous-time) systems.
- Key concepts:
  - ***Signals***: time-varying values
  - ***Signal Functions***: functions on signals
  - ***Switching*** between signal functions
- Programming model:



What is the point?

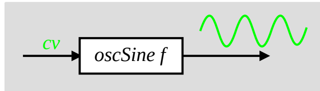
- Music can be seen as a hybrid phenomenon. Thus interesting to explore a hybrid approach to programming music and musical applications.
- Yampa's programming model is very reminiscent of programming modular synthesizers, so . . .
- Fun application! Useful e.g.in a class-room context?

So, what have you done?

Framework for programming modular synthesizers in Yampa:

- Sound-generating and sound-shaping modules
- Supporting infrastructure:
  - Reading MIDI files (musical scores)
  - Reading SoundFont files (instrument definitions)
  - Writing result as audio files (.wav)
- Status: proof-of-concept, but decent performance.

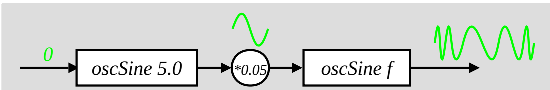
Example 1: Sine oscillator



$$
\begin{array}{l}
\mathit{oscSine} :: \mathit{Frequency} \rightarrow \mathit{SF}\ \mathit{CV}\ \mathit{Sample} \\
\mathit{oscSine}\ \mathit{f0} = \mathbf{proc}\ \mathit{cv} \rightarrow \mathbf{do} \\
\quad \mathbf{let}\ f = \mathit{f0} * (2 ** \mathit{cv}) \\
\quad \mathit{phi} \leftarrow \mathit{integral} \mathrel{-\!\!\prec} 2 * \mathit{pi} * f \\
\quad \mathit{returnA} \mathrel{-\!\!\prec} \mathit{sin}\ \mathit{phi}
\end{array}
$$

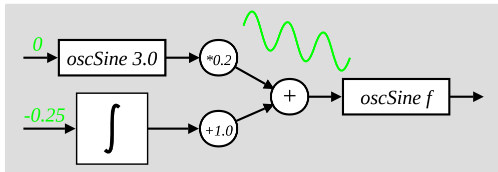
$$\mathit{constant}\ 0 \ggg \mathit{oscSine}\ 440$$

Example 2: Vibrato



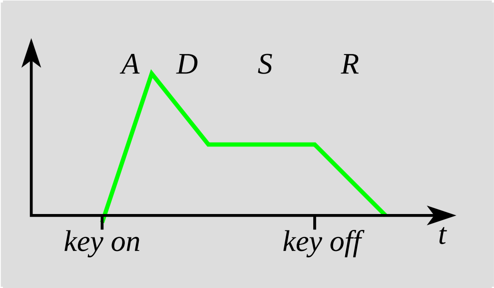
$$
\begin{array}{l}
\mathit{constant}\ 0 \\
\ggg \mathit{oscSine}\ 5.0 \\
\ggg \mathit{arr}\ (*0.05) \\
\ggg \mathit{oscSine}\ 440
\end{array}
$$

Example 3: 50's Sci Fi



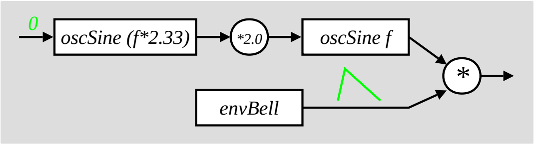
$$
\begin{array}{l}
\mathit{sciFi} :: \mathit{SF}\ ()\ \mathit{Sample} \\
\mathit{sciFi} = \mathbf{proc}\ () \rightarrow \mathbf{do} \\
\quad \mathit{und} \leftarrow \mathit{arr}\ (*0.2) \lll \mathit{oscSine}\ 3.0 \mathrel{-\!\!\prec} 0 \\
\quad \mathit{swp} \leftarrow \mathit{arr}\ (+1.0) \lll \mathit{integral} \quad \mathrel{-\!\!\prec} -0.25 \\
\quad \mathit{audio} \leftarrow \mathit{oscSine}\ 440 \quad \mathrel{-\!\!\prec} \mathit{und} + \mathit{swp} \\
\quad \mathit{returnA} \mathrel{-\!\!\prec} \mathit{audio}
\end{array}
$$

Envelope Generators



$$envGen :: CV \to [(Time, CV)] \to (Maybe\ Int)$$
$$\to SF\ (Event\ ())\ (CV, Event\ ())$$
$$envBell = envGen\ 0\ [(0.05, 1), (1.5, 0)]\ Nothing$$

Example 4: Bell



$$
\begin{aligned}
&bell :: Frequency \to SF\ ()\ (Sample, Event)\\
&bell\ f = \mathbf{proc}\ ()\ \to \mathbf{do}\\
&\quad m \leftarrow oscSine\ (2.33 * f) \prec 0\\
&\quad audio \leftarrow oscSine\ f \prec 2.0 * m\\
&\quad (ampl, end) \leftarrow envBell \prec noEvent\\
&\quad returnA \prec (audio * ampl, end)
\end{aligned}
$$

Example 5: Playing a C-major scale

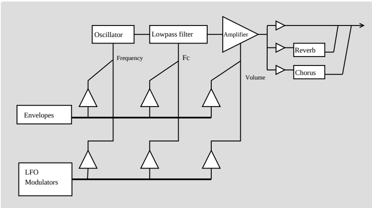
$$
\begin{aligned}
&scale :: SF\ ()\ (Sample, Event)\\
&scale = (afterEach\ [(0.0, 60), (2.0, 62), (2.0, 64),\\
&\qquad (2.0, 65), (2.0, 67), (2.0, 69),\\
&\qquad (2.0, 71), (2.0, 72)]\\
&\qquad \ggg constant\ ()\\
&\qquad\quad \&\&\&\ arr\ (fmap\ (bell \circ midiNoteToFreq))\\
&\qquad \ggg rSwitch\ (constant\ 0))\\
&\quad \&\&\&\ after\ 16\ ()
\end{aligned}
$$

Example 6: Polyphonic synthesizer (1)

Sample-playing monophnic synthesizer:

- Read samples (instrument recordings) from SoundFont file into internal table.
- Oscillator similar to sine oscillator, except sine func. replaced by table lookup and interpolation.

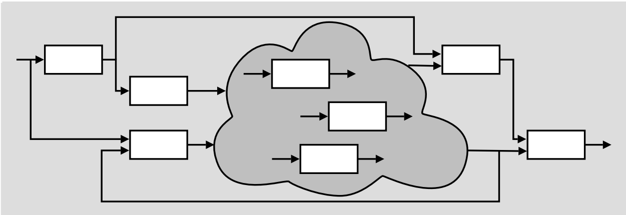
SoundFont synthesizer structure:



Example 6: Polyphonic synthesizer (2)

Exploit Yampa's switching capabilities to:

- create and switch in a mono synth instance is response to each note on event;
- switch out the instance in response to a corresponding note off event.



Switched-on Yampa?

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