Switched-on Yampa

Programming Modular Synthesizers in Haskell

George Giorgidze and Henrik Nilsson

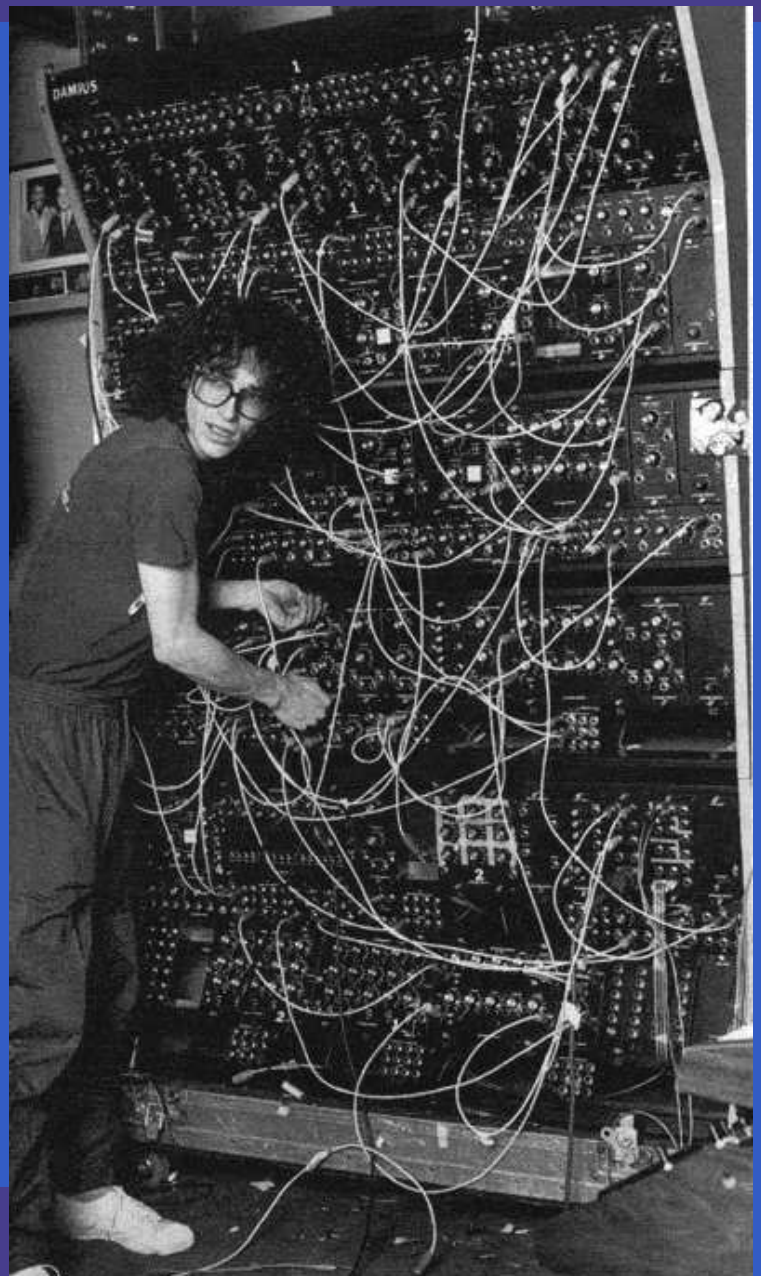
School of Computer Science

The University of Nottingham, UK

Modular synthesizers?

Modular synthesizers?

Modular synthesizers?



What is Yampa?

What is Yampa?

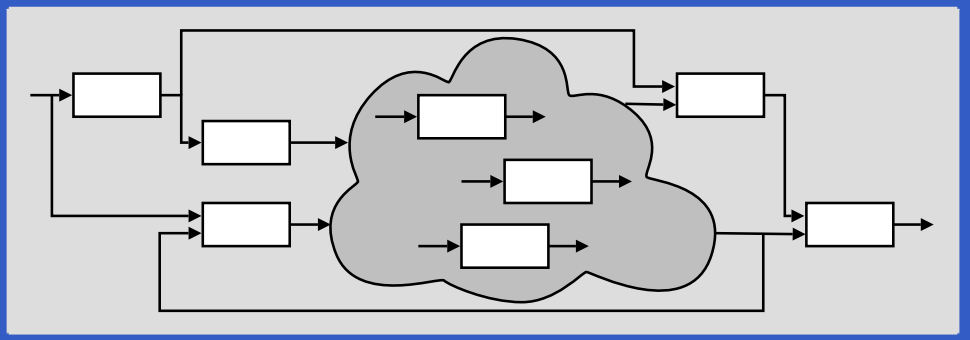
- Domain-specific language embedded in Haskell for programming ***hybrid*** (mixed discrete- and continuous-time) systems.

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

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?

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.

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 . . .

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?

So, what have you done?

Framework for programming modular synthesizers in Yampa:

So, what have you done?

Framework for programming modular synthesizers in Yampa:

- Sound-generating and sound-shaping modules

So, what have you done?

Framework for programming modular synthesizers in Yampa:

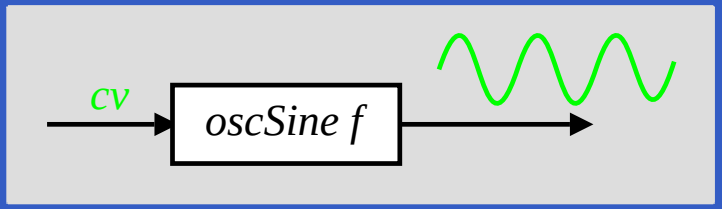
- Sound-generating and sound-shaping modules
- Supporting infrastructure:
  - Reading MIDI files (musical scores)
  - Reading SoundFont files (instrument definitions)
  - Writing result as audio files (.wav)

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.

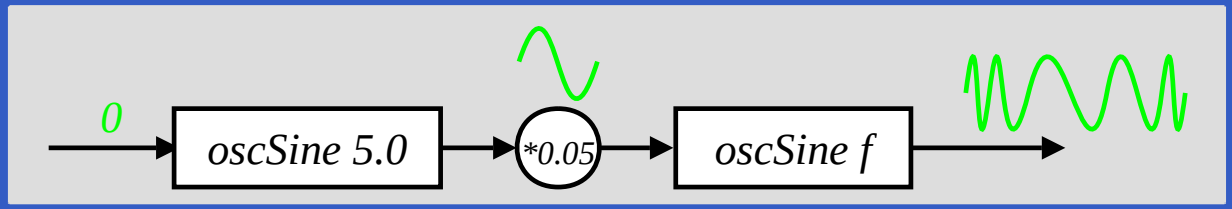
Example 1: Sine oscillator



$$oscSine :: Frequency \rightarrow SF\ CV\ Sample$$
$$oscSine\ f0 = \mathbf{proc}\ cv \rightarrow \mathbf{do}$$
$$\quad \mathbf{let}\ f = f0 * (2 ** cv)$$
$$\quad phi \leftarrow integral \prec 2 * pi * f$$
$$\quad returnA \prec sin\ phi$$

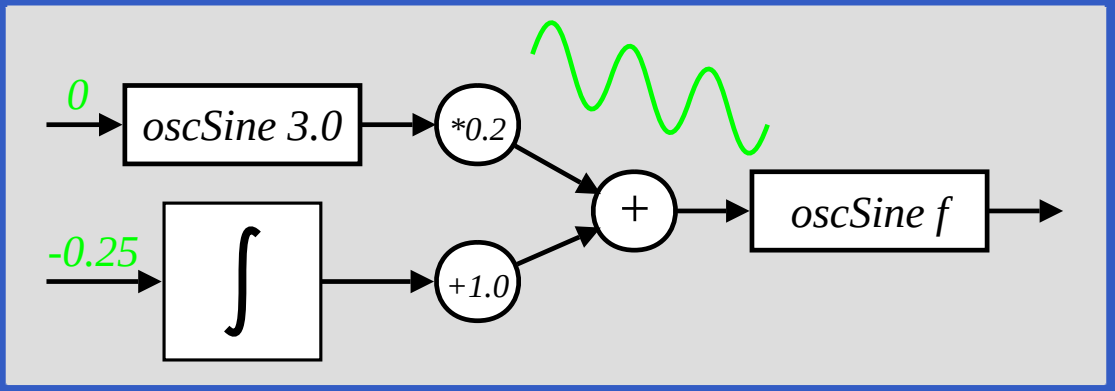
$$constant\ 0 \ggg oscSine\ 440$$

Example 2: Vibrato



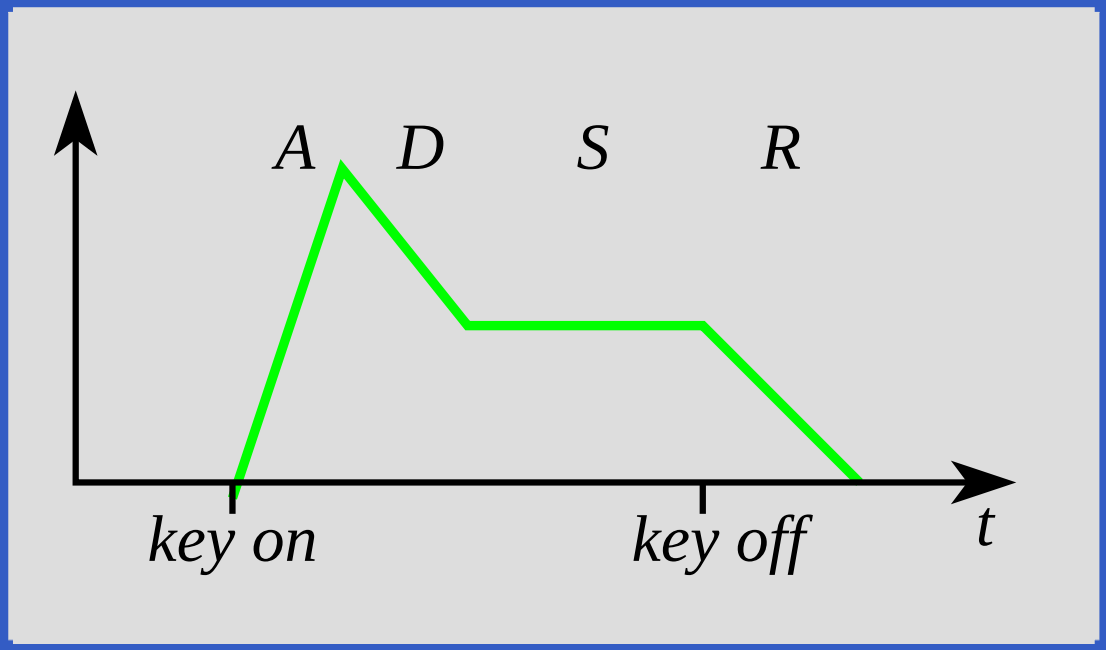
$$constant\ 0 \ggg oscSine\ 5.0 \ggg arr\ (*0.05) \ggg oscSine\ 440$$

Example 3: 50's Sci Fi



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

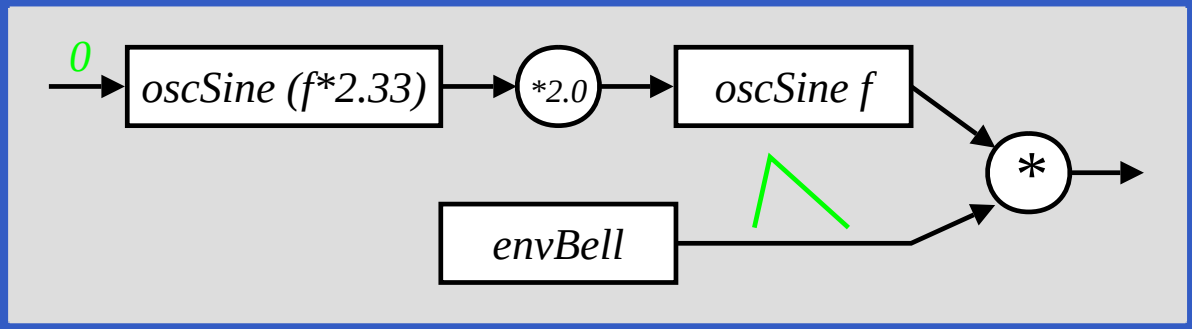
Envelope Generators



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

$$envBell = envGen\ 0\ [(0.05, 1), (1.5, 0)]\ Nothing$$

Example 4: Bell



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

Example 5: Playing a C-major scale

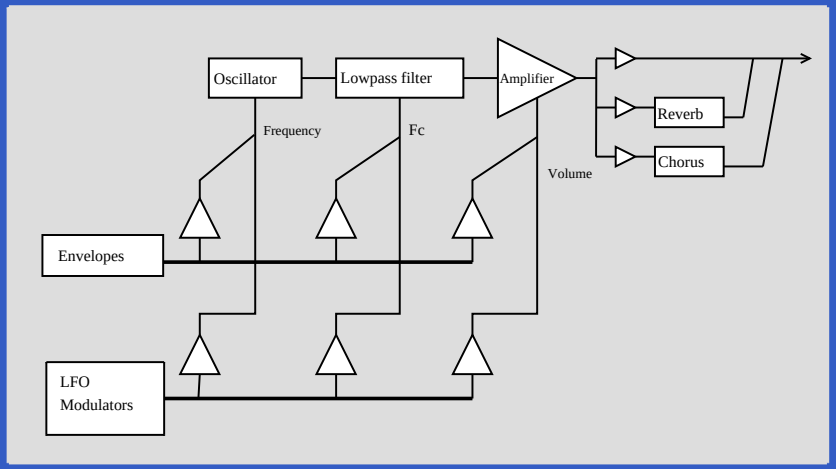
$$
\begin{aligned}
&scale :: SF\ ()\ (Sample, Event)\\
&scale = (afterEach\ [(0.0, 60), (2.0, 62), (2.0, 64),\\
&\qquad\qquad (2.0, 65), (2.0, 67), (2.0, 69),\\
&\qquad\qquad (2.0, 71), (2.0, 72)]\\
&\qquad \ggg constant\ ()\\
&\qquad\quad \&\&\& arr\ (fmap\ (bell \circ midiNoteToFreq))\\
&\qquad \ggg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

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.

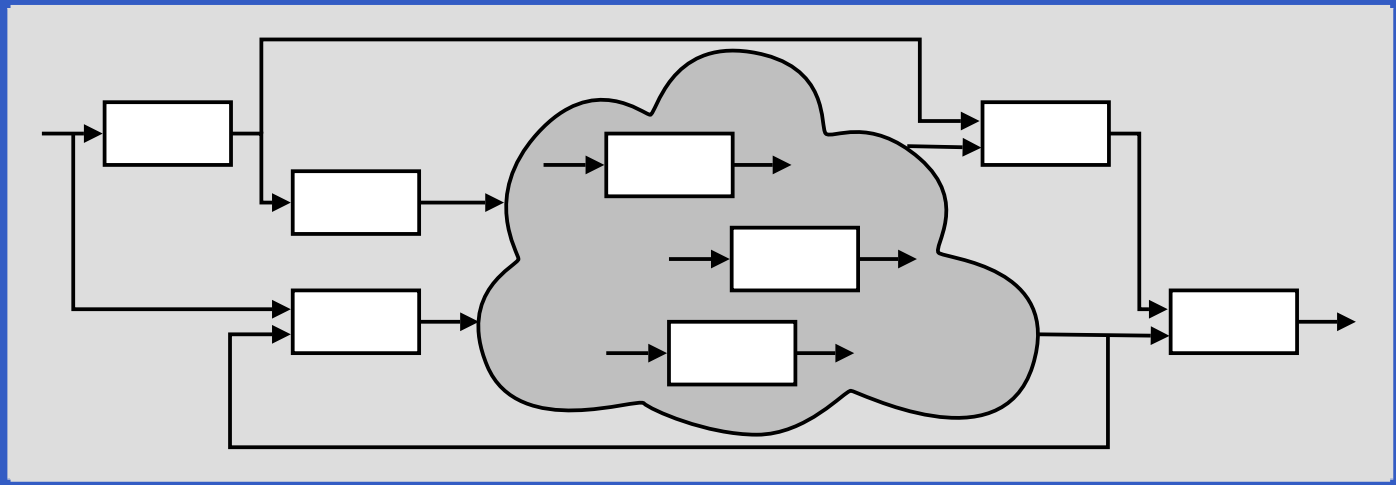
SoundFont synthesizer structure:



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?

Switched-on Yampa?

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