ITU FRP 2010

Lecture 6:

Switched-on Yampa: Programming Modular Synthesizers in Haskell

Henrik Nilsson and George Giorgidze

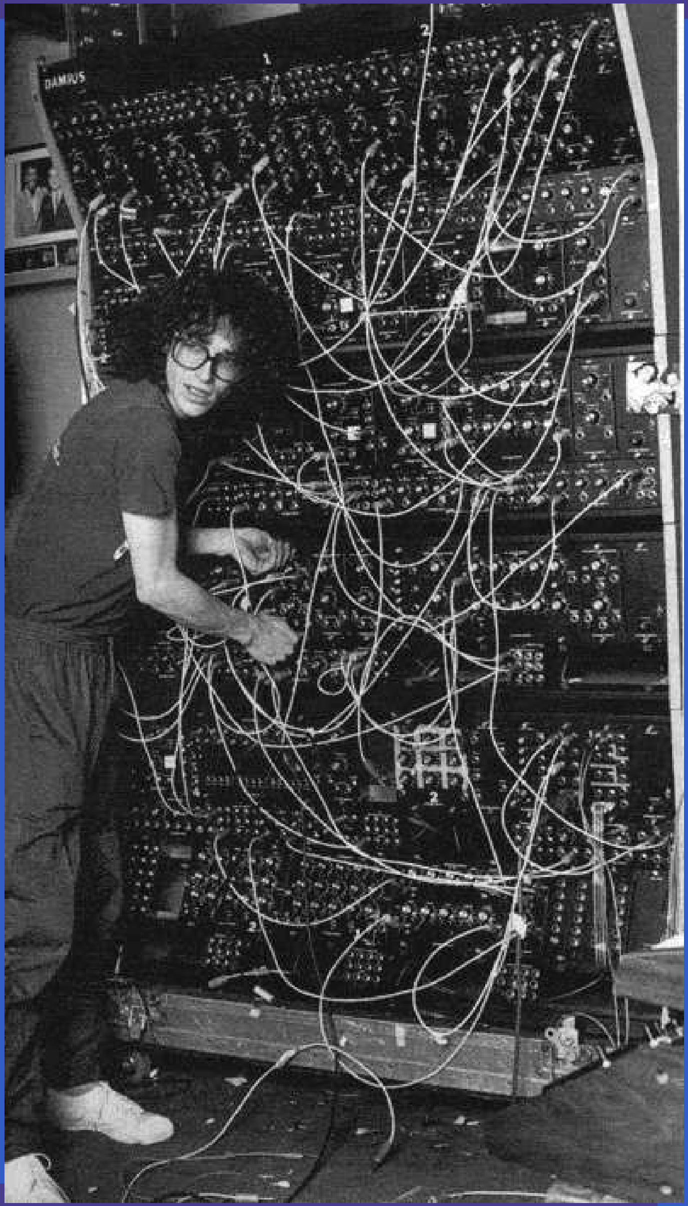
School of Computer Science

The University of Nottingham, UK

Modular synthesizers?

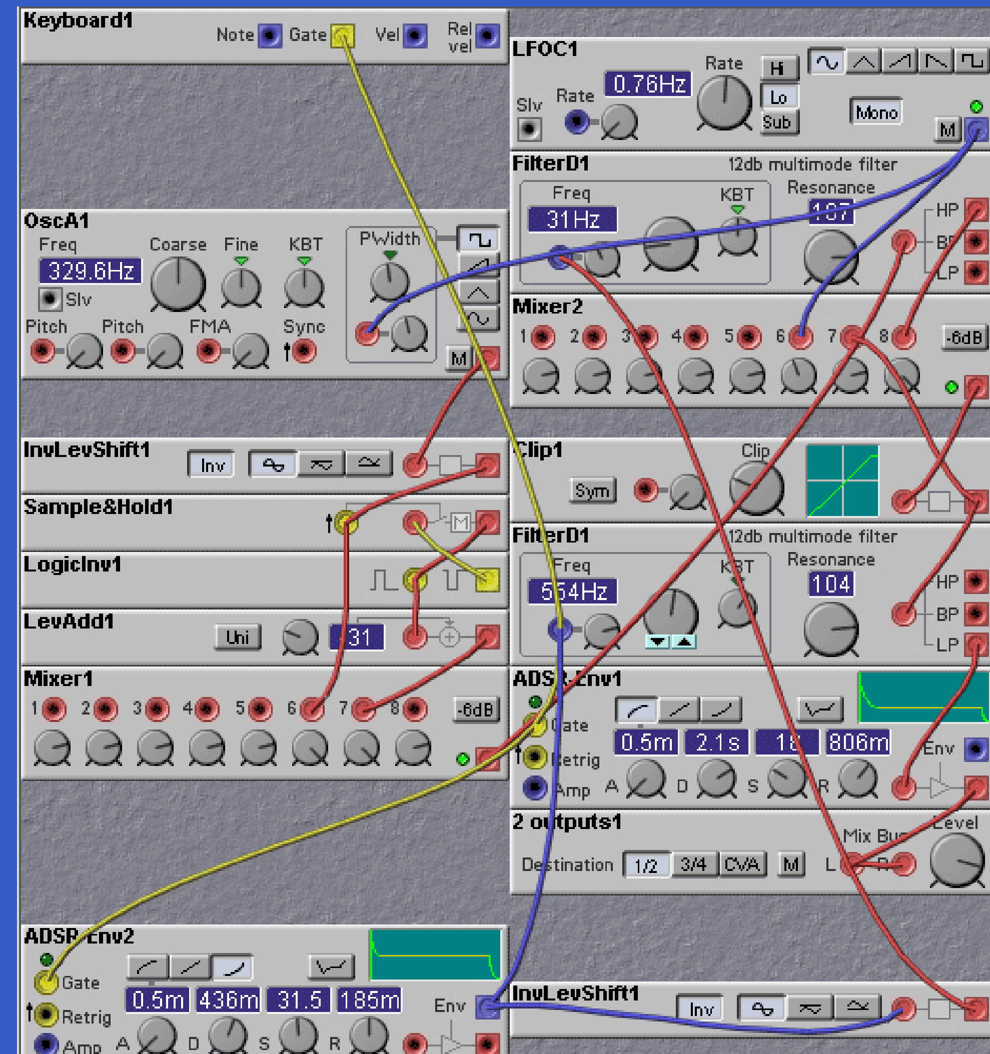
Modular synthesizers?

Modular synthesizers?



Steve Pocaro, Toto, with Polyfusion Synthesizer

Modern Modular Synthesizers



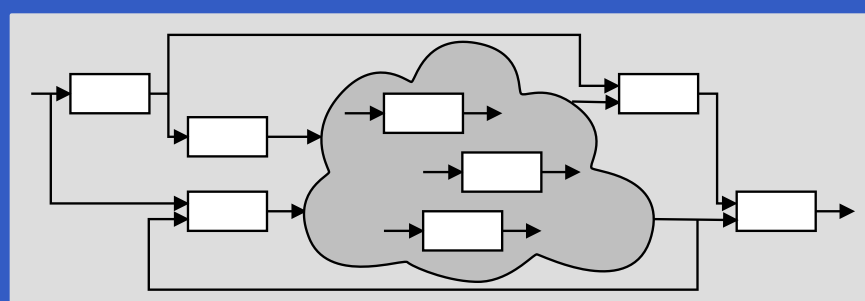
Where does Yampa enter the picture?

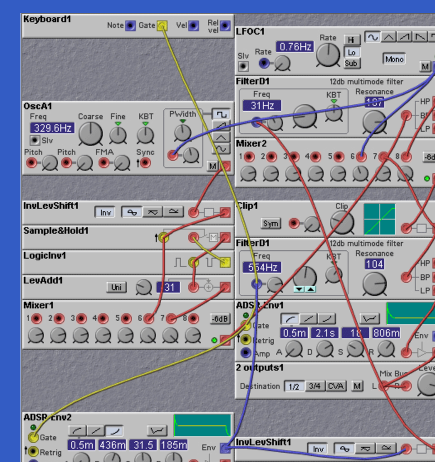
Where does Yampa enter the picture?

- 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?

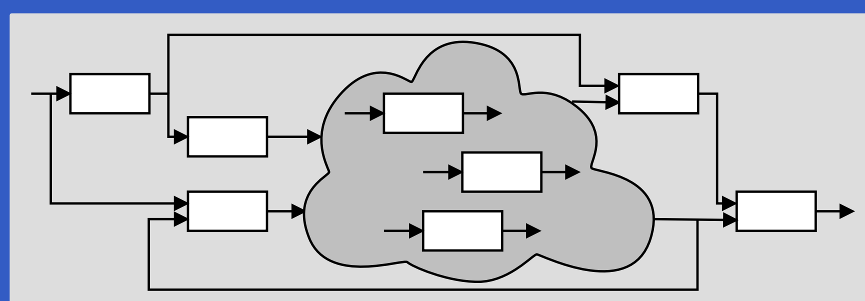
- Music can be seen as a hybrid phenomenon. Thus interesting to explore a hybrid approach to programming music and musical applications.
- Yampa’s programming model is very reminiscent of programming modular synthesizers:

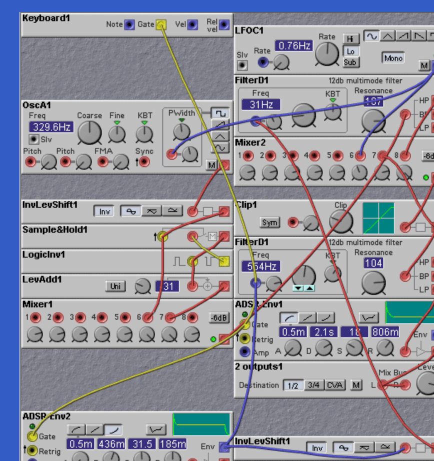




Where does Yampa enter the picture?

- 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:

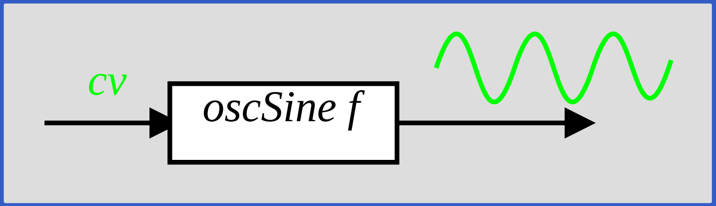
- Sound-generating and sound-shaping modules
- Additional supporting infrastructure:
  - Input: MIDI files (musical scores), keyboard
  - Output: audio files (.wav), sound card
  - Reading SoundFont files (instrument definitions)

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.

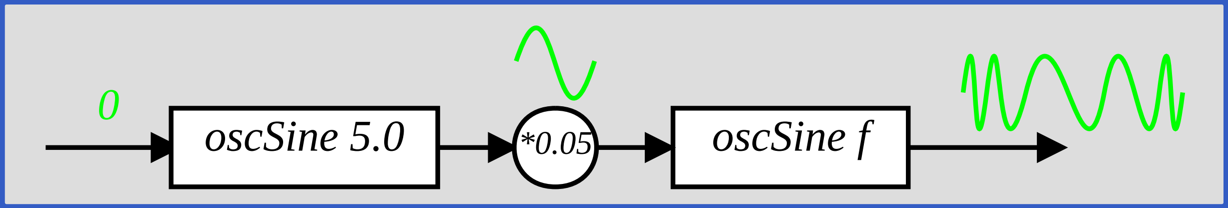
Example 1: Sine oscillator



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

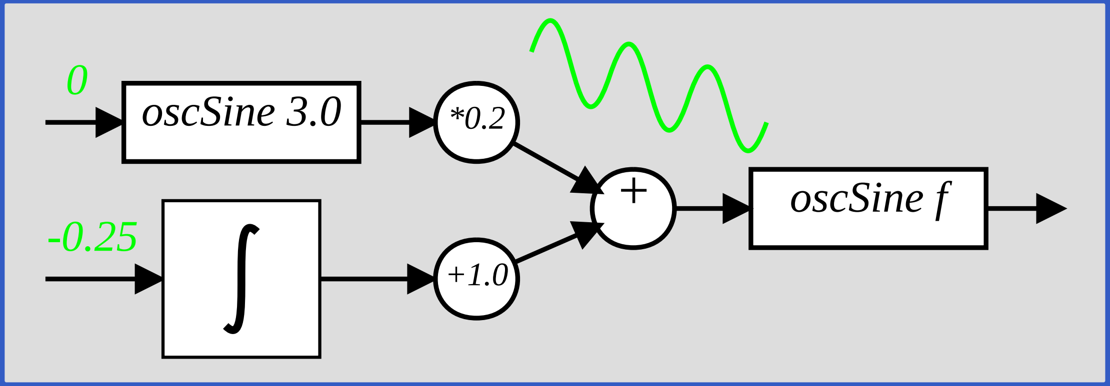
$$constant\ 0 \ggg oscSine\ 440$$

Example 2: Vibrato



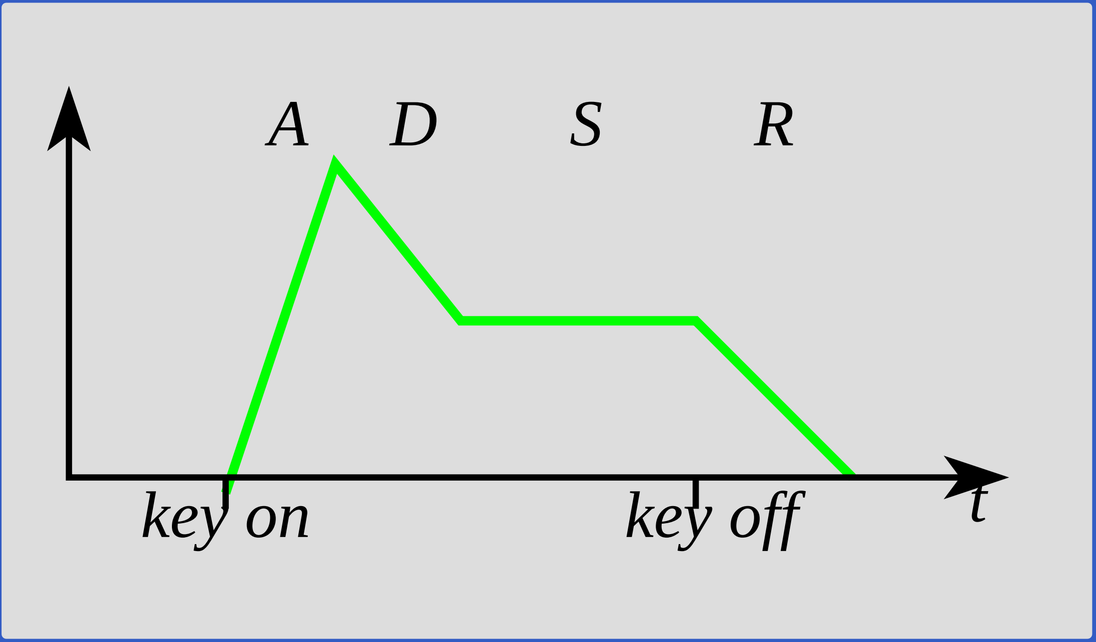
$$constant\ 0$$
$$\ggg\ oscSine\ 5.0$$
$$\ggg\ arr\ (*0.05)$$
$$\ggg\ oscSine\ 440$$

Example 3: 50's Sci Fi



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

Envelope Generators (1)



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

$$envEx = envGen\ 0\ [(0.5, 1), (0.5, 0.5), (1.0, 0.5), (0.7, 0)]$$
$$(Just\ 3)$$

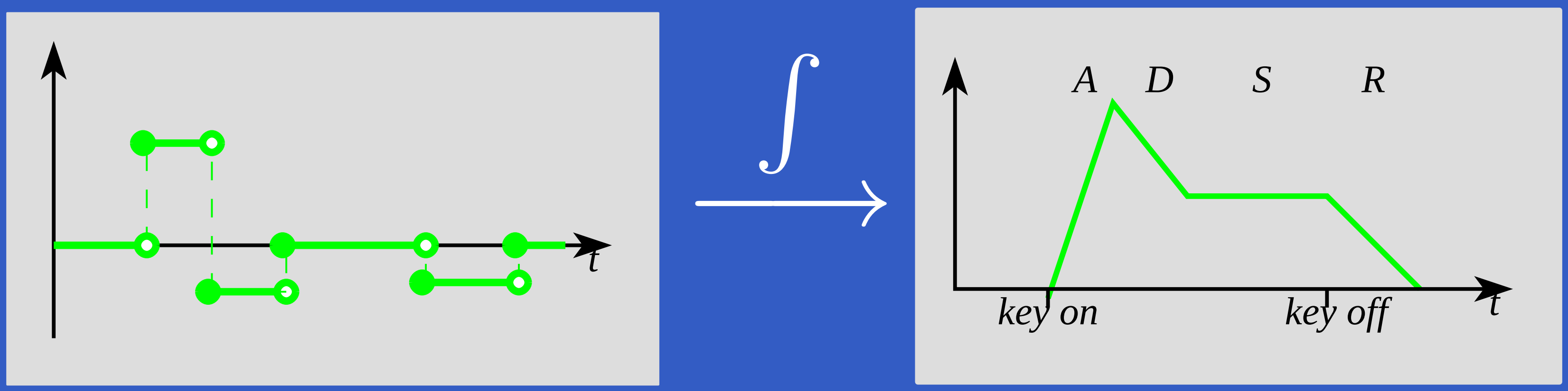
Envelope Generators (2)

How to implement?

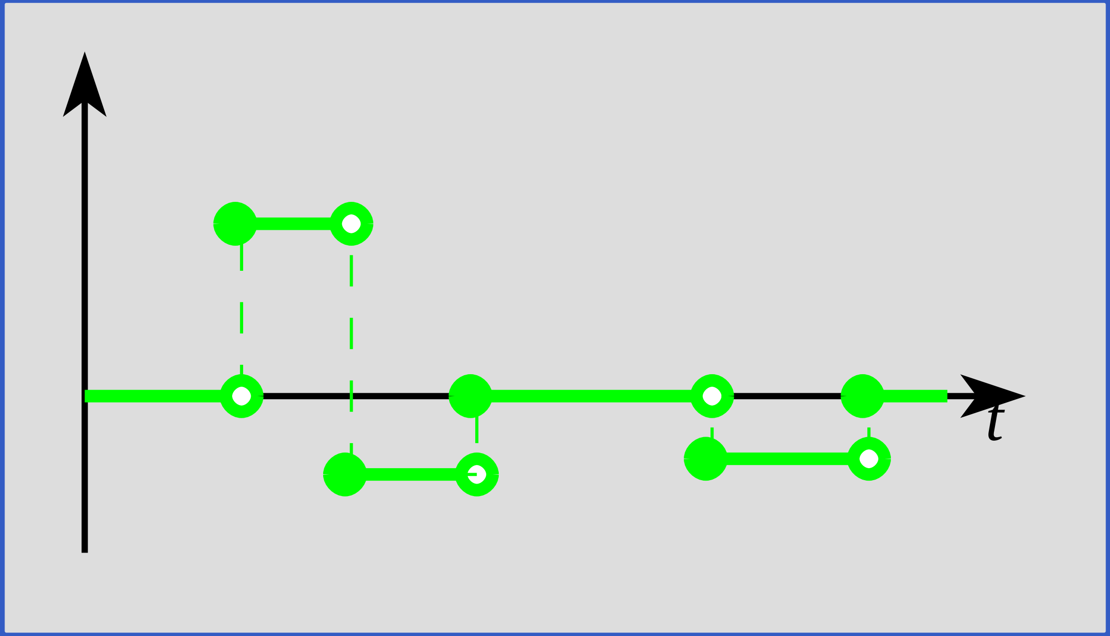
Envelope Generators (2)

How to implement?

Integration of a step function yields suitable shapes:



Envelope Generators (3)



$$afterEach :: [(Time, b)] \rightarrow SF\ a\ (Event\ b)$$

$$hold \qquad :: a \rightarrow SF\ (Event\ a)\ a$$

$$steps = afterEach\ [(0.7, 2), (0.5, -1), (0.5, 0), (1, -0.7), (0.7, 0)$$

$$\ggg hold\ 0$$

Envelope Generators (4)

Envelope generator with predetermined shape:

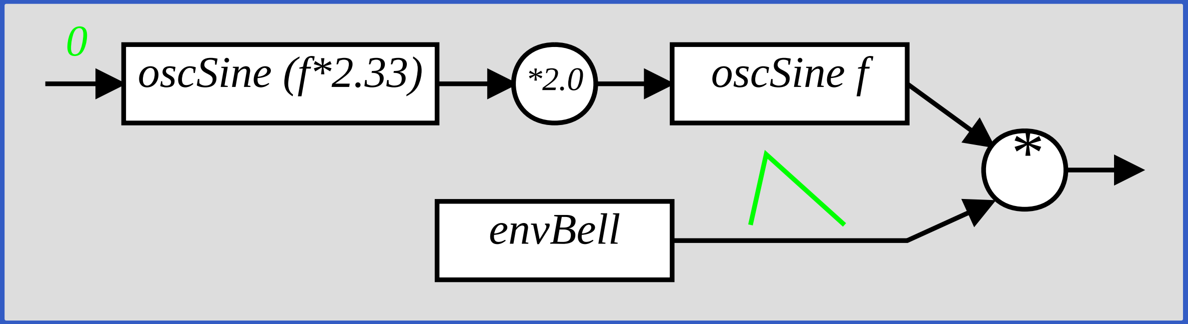
$$
\begin{aligned}
&\mathit{envGenAux} :: \mathit{CV} \rightarrow [(\mathit{Time}, \mathit{CV})] \rightarrow \mathit{SF}\ a\ \mathit{CV} \\
&\mathit{envGenAux}\ l0\ \mathit{tls} = \mathit{afterEach}\ \mathit{trs} \ggg \mathit{hold}\ r0 \\
&\qquad\qquad\qquad \ggg \mathit{integral} \ggg \mathit{arr}\ (+l0) \\
&\quad \textbf{where} \\
&\qquad (r0, \mathit{trs}) = \mathit{toRates}\ l0\ \mathit{tls}
\end{aligned}
$$

Envelope Generators (5)

Envelope generator responding to key off:

$$
\begin{array}{l}
envGen :: CV \rightarrow [(Time, CV)] \rightarrow (Maybe\ Int) \\
\qquad\qquad \rightarrow SF\ (Event\ ())\ (CV, Event\ ()) \\
envGen\ l0\ tls\ (Just\ n) = \\
\quad switch\ (\mathbf{proc}\ noteoff \rightarrow \mathbf{do} \\
\qquad\qquad l \leftarrow envGenAux\ l0\ tls1 \prec () \\
\qquad\qquad returnA \prec ((l, noEvent), noteoff\ `tag`\ l)) \\
\qquad\quad (\lambda l \rightarrow envGenAux\ l\ tls2 \\
\qquad\qquad \&\&\&\ after\ (sum\ (map\ fst\ tls2))\ ()) \\
\quad \mathbf{where} \\
\qquad (tls1, tls2) = splitAt\ n\ tls
\end{array}
$$

Example 4: Bell



$$bell :: Frequency \rightarrow SF\ ()\ (Sample, Event)$$

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

Example 5: Tinkling Bell

$$
\begin{aligned}
&tinkle :: SF\ ()\ Sample\\
&tinkle = (repeatedly\ 0.25\ 84\\
&\qquad\qquad \ggg constant\ ()\\
&\qquad\qquad\qquad \&\&\&\ arr\ (fmap\ (bell \circ midiNoteToFreq))\\
&\qquad\qquad \ggg rSwitch\ (constant\ 0))
\end{aligned}
$$

Example 6: Playing a C-major scale

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

Example 7: Playing simultaneous notes

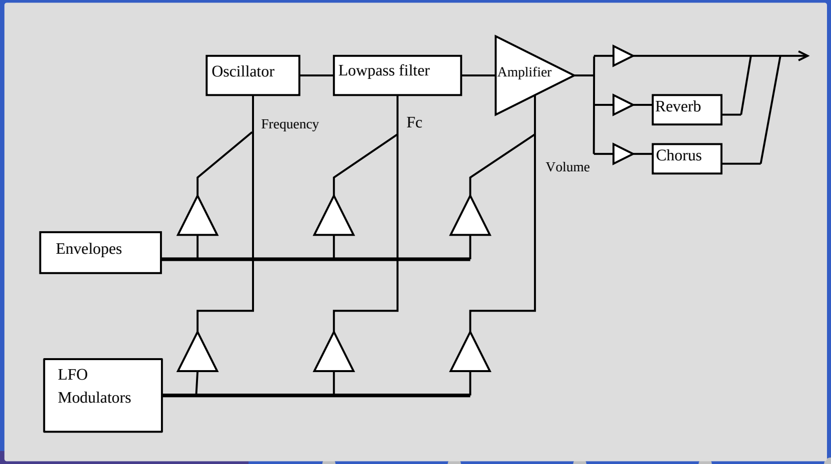
$$
\begin{aligned}
&\mathit{mysterySong} :: \mathit{SF}\ ()\ (\mathit{Sample}, \mathit{Event}\ ())\\
&\mathit{mysterySong} = \mathbf{proc}\ \_ \rightarrow \mathbf{do}\\
&\quad t \ \leftarrow \mathit{tinkle} \quad \prec ()\\
&\quad m \leftarrow \mathit{mystery} \prec ()\\
&\quad \mathit{returnA} \prec (0.4 * t + 0.6 * m)
\end{aligned}
$$

A polyphonic synthesizer (1)

Sample-playing monophnic 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.

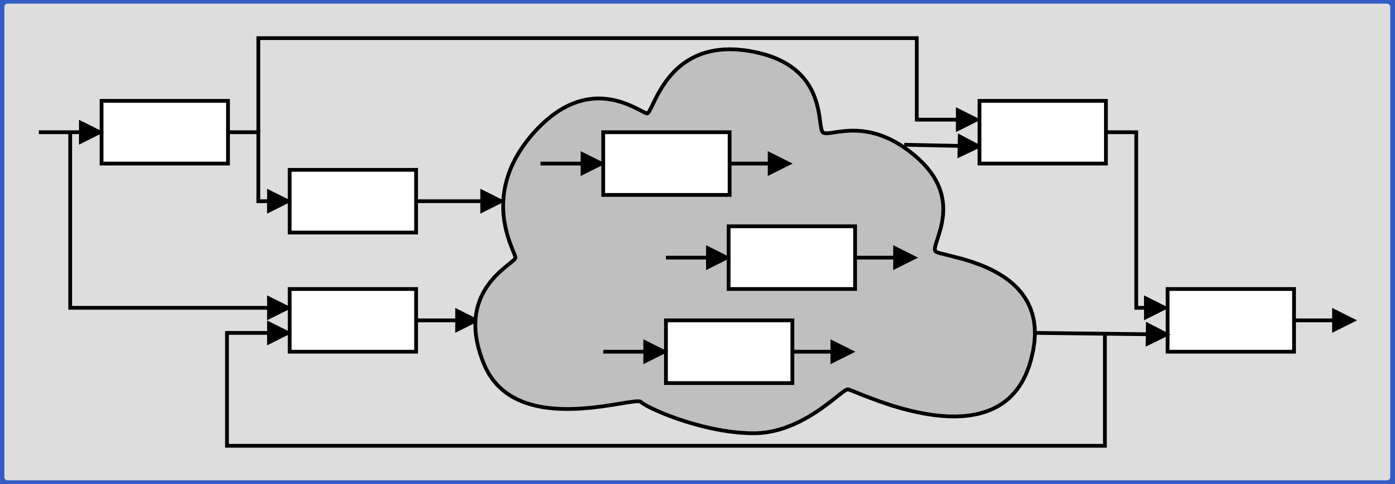
SoundFont synthesizer structure:



A 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`