ITU FRP 2010

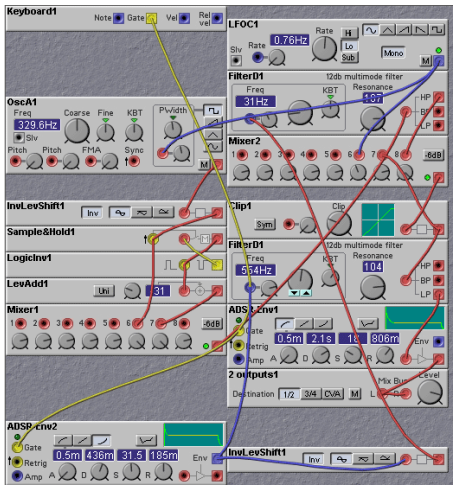
Lecture 6: Switched-on Yampa: Programming Modular Synthesizers in Haskell

Henrik Nilsson and George Giorgidze

School of Computer Science
The University of Nottingham, UK

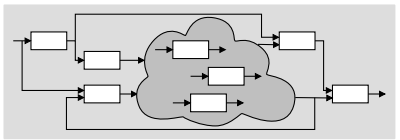
Modular synthesizers?

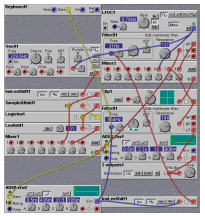
Modern Modular Synthesizers



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:





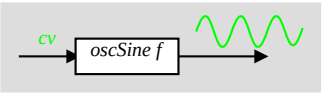
- Fun application! Useful for teaching?

What have we done?

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.

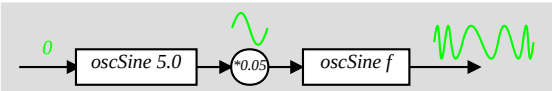
Example 1: Sine oscillator



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

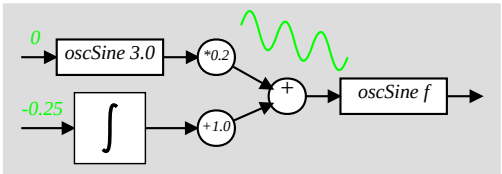
$$\mathit{constant}\ 0 \ggg \mathit{oscSine}\ 440$$

Example 2: Vibrato



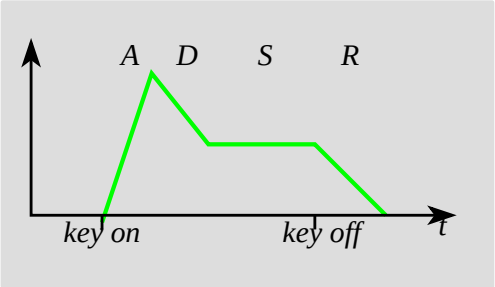
$$
\begin{aligned}
&\mathit{constant}\ 0\\
&\ggg \mathit{oscSine}\ 5.0\\
&\ggg \mathit{arr}\ (*0.05)\\
&\ggg \mathit{oscSine}\ 440
\end{aligned}
$$

Example 3: 50's Sci Fi



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

Envelope Generators (1)



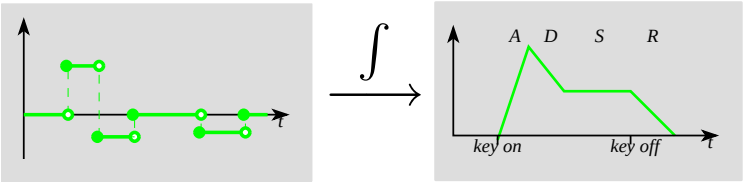
$$envGen :: CV \to [(Time, CV)] \to (Maybe\ Int)$$
$$\to SF\ (Event\ ())\ (CV, Event\ ())$$

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

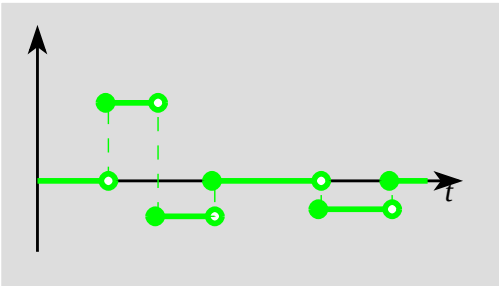
Envelope Generators (2)

How to implement?

Integration of a step function yields suitable shapes:



Envelope Generators (3)



$$afterEach :: [(Time, b)] \to SF\ a\ (Event\ b)$$
$$hold \quad :: a \to SF\ (Event\ a)\ a$$
$$steps = afterEach\ [(0.7, 2), (0.5, -1), (0.5, 0), (1, -0.7), (0.7, 0)]$$
$$\ggg hold\ 0$$

Envelope Generators (4)

Envelope generator with predetermined shape:

$$envGenAux :: CV \to [(Time, CV)] \to SF\ a\ CV$$
$$envGenAux\ l0\ tls = afterEach\ trs \ggg hold\ r0$$
$$\ggg integral \ggg arr\ (+l0)$$

where

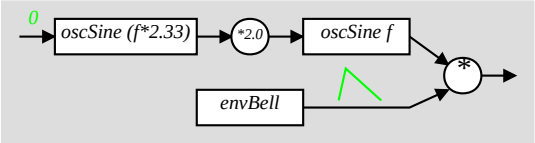
$$(r0, trs) = toRates\ l0\ tls$$

Envelope Generators (5)

Envelope generator responding to key off:

```
envGen :: CV → [(Time, CV)] → (Maybe Int)
          → SF (Event ()) (CV, Event ())
envGen l0 tls (Just n) =
  switch (proc noteoff → do
            l ← envGenAux l0 tls1 -< ()
            returnA -< ((l, noEvent), noteoff `tag` l))
         (λl → envGenAux l tls2
            &&& after (sum (map fst tls2)) ())
  where
    (tls1, tls2) = 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 = (repeatedly 0.25 84
          >>> constant ()
              &&& arr (fmap (bell ∘ midiNoteToFreq))
          >>> 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)]
          >>> constant ()
              &&& arr (fmap (bell ∘ midiNoteToFreq))
          >>> rSwitch (constant 0))
        &&& after 16 ()
```

Example 7: Playing simultaneous notes

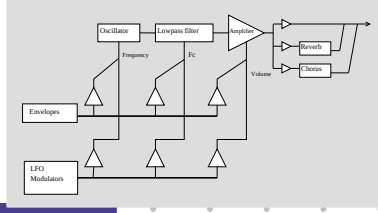
$$
\begin{aligned}
&mysterySong :: SF\ ()\ (Sample, Event\ ())\\
&mysterySong = \mathbf{proc}\ \_ \rightarrow \mathbf{do}\\
&\quad t \leftarrow tinkle \quad \prec ()\\
&\quad m \leftarrow mystery \prec ()\\
&\quad returnA \prec (0.4 * t + 0.6 * m)
\end{aligned}
$$

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.

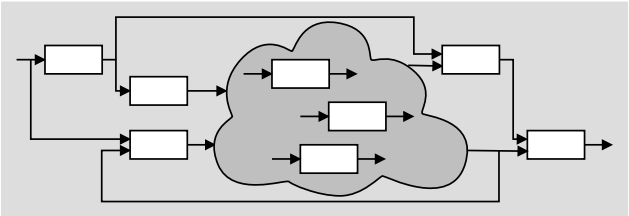
SoundFont synthesizer structure:



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?

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