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

```
oscSine :: Frequency \rightarrow SF CV Sample
oscSine f0 = proc cv \rightarrow do
  let f = f0 * (2 ** cv)
  phi \leftarrow integral \rightarrow 2 * pi * f
  returnA \leftarrow \sin phi
```

```
constant 0 \Rightarrow oscSine 440
```

Example 2: Vibrato

```
constant 0
\Rightarrow oscSine 5.0
\Rightarrow arr (*0.05)
\Rightarrow oscSine
```

Example 3: 50’s Sci Fi

```
sciFi :: SF () \rightarrow Sample
sciFi = proc () \rightarrow do
  und \leftarrow arr (*0.2) \&\& oscSine 3.0 \&\& 0
  swp \leftarrow arr (+1.0) \&\& \text{integral} \leftarrow -0.25
  audio \leftarrow oscSine 440 \leftarrow und + swp
  returnA \leftarrow audio
```
**Envelope Generators**

![Envelope Diagram]

\[ envGen :: CV \rightarrow [(\text{Time}, CV)] \rightarrow (\text{Maybe Int}) \]
\[ \rightarrow SF (\text{Event}()) (CV, \text{Event}()) \]
\[ envBell = envGen 0 [(0.05, 1), (1.5, 0)] \text{ Nothing} \]

**Example 4: Bell**

\[ bell :: \text{Frequency} \rightarrow SF () (\text{Sample, Event}) \]
\[ bell f = \text{proc} () \rightarrow \text{do} \]
[\[ m \leftarrow \text{oscSine} (2.33 \ast f) \leftarrow 0 \]
[\[ \text{audio} \leftarrow \text{oscSine} f \leftarrow 2.0 \ast m \]
[\[ \text{(ampl, end)} \leftarrow \text{envBell} \leftarrow \text{noEvent} \]
[\[ \text{return} A \leftarrow (\text{audio} \ast \text{ampl}, \text{end}) \]

**Example 5: Playing a C-major scale**

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

**Example 6: Polyphonic synthesizer (1)**

Sample-playing monophonic 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 in response to each note on event;
- switch out the instance in response to a corresponding note off event.

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