On-demand computations in Faust

[YO, Preliminary draft v3]

Introduction

There are requests for using Faust in a more *composition-oriented* way. For that purpose, it has been suggested to introduce *on-demand* computations. In this model, computations are not performed on a sample-by-sample basis, as is normally the case with Faust, but only on request. Conceptually, these requests are propagated backwards, starting from the output of the expression and going back to the inputs.

The challenge is to introduce *on-demand* computations while keeping the simple and well-defined *signal processor* semantics of Faust. In this note we propose a new $\texttt{ondemand}(P) \rightarrow P'$ primitive that transforms a signal processor P into an *on-demand* signal processor P', and we define its semantics. As we will see, this semantics can be expressed using the regular Faust semantics, but applied to *downsampled* signals.

Semantics of Faust Expressions

Before describing the particular semantics of $\texttt{ondemand}()$, let's take a look at the semantics of Faust in general. Faust semantics is based on *signals* and *signal processors*. A *signal* is a function of time and a *signal processor* is a function of signals. A Faust program describes a *signal processor and programming in Faust is essentially combining _signal processors* together, for example using composition operators like $:$ or $\sim$.

Time

Time in Faust is discrete and it is represented by $\mathbb{Z}$. All computations start at time 0, but negative $times are possible in order to take delay operations into account .

Signals

A *signal* s is a function of time $s : \mathbb{Z} \rightarrow \mathbb{R}$. Actually Faust considers two types of signals: *integer signals* ($s : \mathbb{Z} \rightarrow \mathbb{Z}$) and *floating point signals* ($s : \mathbb{Z} \rightarrow \mathbb{Q}$) but this distinction doesn't matter here. The value of a signal s at time t is written $s(t)$.

The set of all possible signals in Faust is $\mathbb{S} \subset \mathbb{Z} \rightarrow \mathbb{R}$. The set $\mathbb{S}$ is a subset of $\mathbb{Z} \rightarrow \mathbb{R}$ because the value of any Faust signal s at a negative time is always 0: $\forall s \in \mathbb{S}, s(t < 0) = 0$. In operational terms this corresponds to initialising all delay lines with 0s.

Tuples

A group of n signals (a n-tuple of signals) is written $(s_1, \ldots, s_n) \in \mathbb{S}^n$. The *empty tuple*, single element of $\mathbb{S}^0$ is notated $()$.

Signal Processors

A *signal processors* $P : \mathbb{S}^n \rightarrow \mathbb{S}^m$, is a function that maps a n-tuple of signals to a m-tuple of signals . The set $\mathbb{P} = \bigcup_{n,m} \mathbb{S}^n \rightarrow \mathbb{S}^m$ is the set of all possible signal processors.

Semantic Brackets

In order to distinguish a Faust expression from its *meaning* as a signal processor, we use the semantic brackets notation $[\![\ ]\!]$. For example $[\![+]\!]$ represents the *meaning* of Faust expression $+$, a signal processor with the following type and definition:

$$[\![+]\!] : \mathbb{S}^2 \rightarrow \mathbb{S}^1 \tag{1}$$

$$[\![+]\!](x, y) = \lambda t.\, (x(t) + y(t)) \tag{2}$$

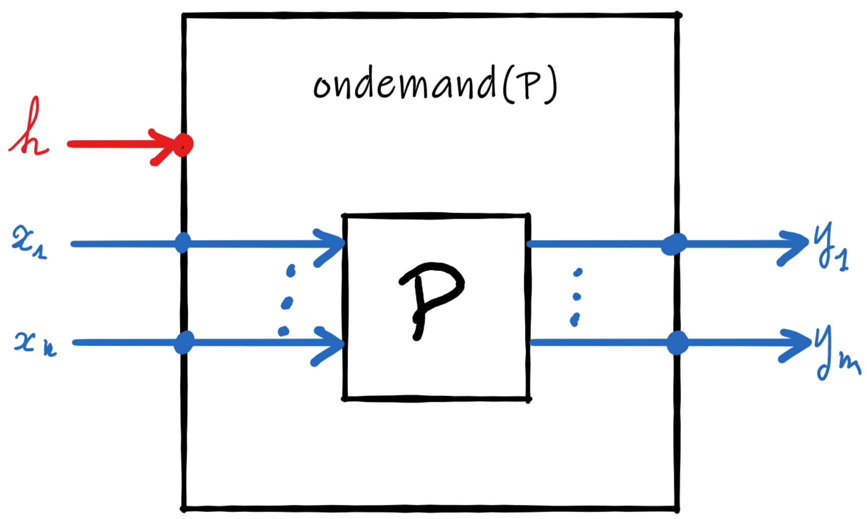
Numbers are also signal processors. The *meaning* of the Faust expression 1 is the following:

$$[\![1]\!] : \mathbb{S}^0 \rightarrow \mathbb{S}^1 \tag{3}$$

$$[\![1]\!]() = \lambda t. \begin{cases} 0 & (t < 0) \\ 1 & (t \geq 0) \end{cases} \tag{4}$$

The $\texttt{ondemand}(P)$ primitive

The ondemand primitive is illustrated in the figure below:



If P has n inputs and m outputs, then $\texttt{ondemand}(P)$ has $n+1$ inputs and m outputs. The additional input of $\texttt{ondemand}(P)$ is a clock signal h that indicates by a 1 when there is a computation demand, and by 0 otherwise. In other words, $h(t)=1$ means that there is a computation demand at time t.

$$\frac{P : n \to m}{\texttt{ondemand}(P) : 1+n \to m} \tag{5}$$

The clock signal h

From a clock signal h we can derive a signal h^* that indicates the time of each demand. For example if $h = 1,0,0,1,0,0,0,1,0\ldots$ then $h^* = 0,3,7,\ldots$ indicating that the first demand is at time 0, the second one at time 3, the third one at time 7, etc. We have

$$\begin{aligned} h^*(0) &= \min\{t'|(h(t')=1)\} \\ h^*(t) &= \min\{t'|(h(t')=1) \wedge (t' > h^*(t-1))\} \end{aligned}$$

We also derive another signal h^+ that *counts* the number of demands:

$$h^+(t) = \sum_{i=0}^{t} h(i) \tag{6}$$

For the same $h = 1,0,0,1,0,0,0,1,0\ldots$ we have $h^+ = 1,1,1,2,2,2,2,3,3,\ldots$

Now that we have defined the clocks signals h and h^*, we can introduce the *downsampling* and *upsampling* operations needed to express the *on-demand* semantics.

Downsampling

The downsamplig operation is notated $\downarrow h$. For a signal x, the downsampled signal $x \downarrow h$ is defined as:

$$x \downarrow h = \lambda t.\, x(h^*(t)) \tag{7}$$

For example if $x = 0.0, -0.1, -0.2, -0.3, -0.4, -0.5, -0.6, -0.7, \ldots$ and $h^* = 0, 3, 7, \ldots$, then $x \downarrow h = 0.0, -0.3, -0.7, \ldots$

Upsampling

The reverse *upsampling* operation, notated $\uparrow h$, expands the input signal by repeating the missing values.

$$x \uparrow h = \lambda t.\, x(h^+(t) - 1) \tag{8}$$

For example if $x = 0.0, -0.3, -0.7, \ldots$ and $h^+ = 1, 1, 1, 2, 2, 2, 2, 3, 3, \ldots$ then $x \uparrow h = 0.0, 0.0, 0.0, -0.3, -0.3, -0.3, -0.3, -0.7, \ldots$

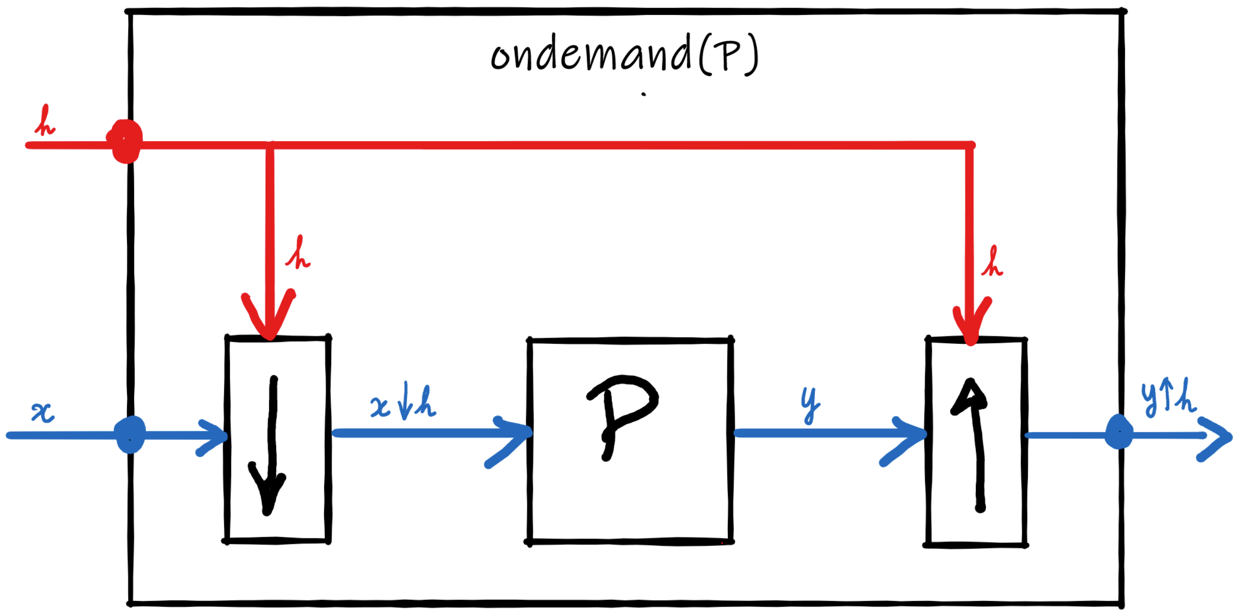
> *NOTE*: please note that $\uparrow h : \downarrow h$ is the identity function, but that is not true for $\downarrow h : \uparrow h$.

Semantics of `ondemand`(P)

We now have all the elements to define the semantics of `ondemand`(P). Let's P be a signal processor with n inputs and m outputs. The semantics of `ondemand`(P) is defined by the following rule:

$$\frac{[\![P]\!](x_1 \downarrow h, \ldots, x_n \downarrow h) = (y_1, \ldots, y_m)}{[\![\texttt{ondemand}(P)]\!](h, x_1, \ldots, x_n) = (y_1 \uparrow h, \ldots, y_m \uparrow h)} \tag{9}$$

As we can see, $[\![\texttt{ondemand}(P)]\!]$ is basically $[\![P]\!]$ applied to downsampled versions of the input signals: $x_i \downarrow h$. The downsampling depends on the demand clock h. Intuitively this corresponds to the fact that the values of the input signals are lost between two computation demands. Symmetrically the y_i signals returned by P have to be upsampled: $y_i \uparrow h$. This is illustrated by the following block-diagram


ondemand(P)
h
h
h
x
x↓h
P
y
y↑h