Compiling Faust with Ondemand

Yann Orlarey

GRAME
CENTRE NATIONAL
DE CRÉATION
MUSICALE, LYON

June 2022

Faust Circuits as Formal Expressions

Faust circuits (evaluated Faust programs) are defined recursively by the following grammar:

Circuits Definition

$$
\begin{aligned}
C \in \mathbb{C} ::= & k \mid u \mid \star \mid @ \mid ! \mid \_ \\
& \mid C_1 : C_2 \mid C_1 , C_2 \\
& \mid C_1 <: C_2 \mid C_1 :> C_2 \\
& \mid C_1 \sim C_2 \mid \mathtt{od}(C)
\end{aligned}
$$

Primitives

- k numbers (integer or real);
- u user interface elements (sliders, buttons, etc.);
- $\star$ any numerical operation;
- $@$ the delay operation;
- $\_$ underscore, the identity circuit (a *perfect* cable);
- $!$ cut, the termination circuit.

Faust Circuits as Formal Expressions

Circuits Composition

- $C_1 <: C_2$ *split composition*, the outputs of C_1 are distributed over the inputs of C_2 ;
- $C_1 :> C_2$ *merge composition*, the outputs of C_1 are summed to form the inputs of C_2 ;
- $C_1 : C_2$ *sequential composition*, the outputs of C_1 are propagated to the inputs of C_2 ;
- C_1, C_2 *parallel composition*, the inputs are those of C_1 and C_2 and so are the outputs;
- $C_1 \sim C_2$ *recursive composition*, the outputs of C_1 are fed back to the inputs of C_2 and vice versa;
- $\texttt{od}(C)$ *ondemand* version of C.

Well Formed Circuits

Number of Inputs and Outputs

$$\text{(seq)}\ \frac{\mathsf{io}[\![C_1]\!] : m \to n \quad \mathsf{io}[\![C_2]\!] : n \to p}{\mathsf{io}[\![C_1 : C_2]\!] : m \to p}$$

$$\text{(par)}\ \frac{\mathsf{io}[\![C_1]\!] : m \to n \quad \mathsf{io}[\![C_2]\!] : p \to q}{\mathsf{io}[\![C_1 , C_2]\!] : m + p \to n + q}$$

$$\text{(split)}\ \frac{\mathsf{io}[\![C_1]\!] : m \to n \quad \mathsf{io}[\![C_2]\!] : n.k \to p}{\mathsf{io}[\![C_1 <: C_2]\!] : m \to p}$$

$$\text{(merge)}\ \frac{\mathsf{io}[\![C_1]\!] : m \to k.n \quad \mathsf{io}[\![C_2]\!] : n \to p}{\mathsf{io}[\![C_1 :> C_2]\!] : m \to p}$$

$$\text{(rec)}\ \frac{\mathsf{io}[\![C_1]\!] : r + n \to q + m \quad \mathsf{io}[\![C_2]\!] : q \to r}{\mathsf{io}[\![C_1 \sim C_2]\!] : n \to q + m}$$

$$\text{(od)}\ \frac{\mathsf{io}[\![C]\!] : m \to n}{\mathsf{io}[\![\mathtt{od}(C)]\!] : m + 1 \to n}$$

Semantics of Well Formed Circuits

Signal Processor Semantics

An audio circuit $C \in \mathbb{C}$ denotes to a signal processor $[\![C]\!] \in \mathbb{P} = \mathbb{S}^n \to \mathbb{S}^m$ that takes input signals and produces output signals.

Notation

- $(S_1, ..., S_n)$ a tuple of signals,
- $()$ the empty tuple and
- $(S_1, ..., S_n, **k)$ an $n * k$ tuple $(S_1, ..., S_n, S_1, ..., S_n, ...)$ obtained by concatenating k times the tuple $(S_1, ..., S_n)$.

Primitives Semantics (1)

Constant

A number k denotes an elementary circuit with no input, that produces a constant signal k.

$$\text{(num)}\;\frac{}{[\![k]\!]() = (k)}$$

Control

A user interface element u denotes an elementary circuit with no input and one output, the signal u produced by the user interface element.

$$\text{(ctrl)}\;\frac{}{[\![u]\!]() = (u)}$$

Primitives Semantics (2)

Numeric operation

The $\star$ symbol denotes a *generic* numerical operation on signals. It represents a circuit with n inputs (typically 1 or 2 depending on the nature of the operation) and one output.

$$\text{(nop)}\ \frac{}{[\![\star]\!](S_1, S_2, ...) = (\star(S_1, S_2, ...))}$$

Delay

A delay primitive $@$ denotes a circuit with two inputs and one output.

$$\text{(delay)}\ \frac{}{[\![@]\!](S_1, S_2) = (S_1 @ S_2)}$$

Primitives Semantics (3)

Cable

The cable has one input and one output.

$$\text{(cable)}\frac{}{[\![\_]\!](S) = (S)}$$

Cut

The cut has one input and no output.

$$\text{(cut)}\frac{}{[\![!]\!](S) = ()}$$

Circuit Compositions Semantics (1)

Sequential Composition Semantics

$$
\textbf{(seq)}\;\frac{\llbracket C_1 \rrbracket(S_1, ..., S_n) = (Y_1, ..., Y_m) \qquad \llbracket C_2 \rrbracket(Y_1, ..., Y_m) = (Z_1, ..., Z_p)}{\llbracket C_1 : C_2 \rrbracket(S_1, ..., S_n) = (Z_1, ..., Z_p)}
$$

Parallel Composition Semantics

$$
\textbf{(par)}\;\frac{\llbracket C_1 \rrbracket(S_1, ..., S_n) = (U_1, ..., U_m) \qquad \llbracket C_2 \rrbracket(Y_1, ..., Y_p) = (V_1, ..., V_q)}{\llbracket C_1, C_2 \rrbracket(S_1, ..., S_n, Y_1, ..., Y_p) = (U_1, ..., U_m, V_1, ..., V_q)}
$$

Circuit Compositions Semantics (2)

Split Composition Semantics

$$\text{(split)}\ \frac{[\![C_1]\!](S_1, ..., S_n) = (Y_1, ..., Y_m) \qquad [\![C_2]\!](Y_1, ..., Y_m, **k) = (Z_1, ..., Z_p)}{[\![C_1 <: C_2]\!](S_1, ..., S_n) = (Z_1, ..., Z_p)}$$

Merge Composition Semantics

$$\text{(merge)}\ \frac{[\![C_1]\!](S_1, ..., S_n) = (Y_{1,1}, ..., Y_{1,m}, ..., Y_{k,1}, , ..., Y_{k,m}) \qquad [\![C_2]\!](Y_{1,1} + ... + Y_{k,1}, ..., Y_{1,m} + ... + Y_{k,m}) = (Z_1, ..., Z_p)}{[\![C_1 :> C_2]\!](S_1, ..., S_n) = (Z_1, ..., Z_p)}$$

Circuit Compositions Semantics (3)

Recursive Composition Semantics

$$(\text{rec})\,\frac{\begin{array}{c} W = \text{fresh recursive symbol} \\ [\![C_2]\!](W_1@1, ..., W_q@1) = (Z_1, ..., Z_r) \\ [\![C_1]\!](Z_1, ..., Z_r, S_1, ..., S_n) = (Y_1, ..., Y_q, Y_{q+1}, ..., Y_{q+m}) \end{array}}{[\![C_1 \sim C_2]\!](S_1, ..., S_n) = (Y_1, ..., Y_q, Y_{q+1}, ..., Y_{q+m})}$$

with $\text{def}[\![W]\!] = (Y_1, ..., Y_q)$.

Ondemand Semantics

Ondemand

$$\text{(od)}\ \frac{[\![C]\!](S_1 \downarrow H, ..., S_n \downarrow H) = (Y_1, ..., Y_m)}{[\![\mathtt{od}(C)]\!](H, S_1, ..., S_n) = (Y_1 \uparrow H, ..., Y_m \uparrow H)}$$