Type inference in the multirate audio DSP language Faust

Pierre Beauguitte

Centre de recherche en informatique - MINES ParisTech

SYNCHRON 2012

FAUST : Functional AUdio STream

Language developed at GRAME (Lyon) since 2003.

ANR Astrée 2009-2011 (MINES ParisTech, IRCAM and U. St Etienne).

Compiled language, real-time audio applications.

Audio synthesis, treatments; interactive applications.

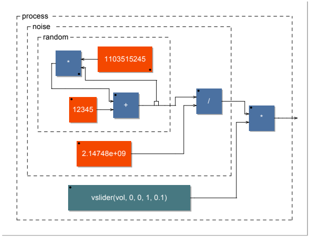
Work at sample-level (typically 44.1 kHz).

Plan

Overview

Domain-specific language, audio digital signal processing.

- synchronous
- purely functionnal
- textual block-diagram description
- statically typed



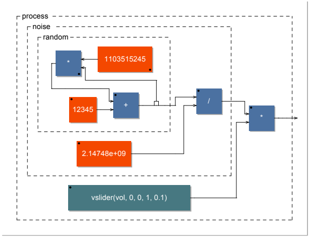

```
random  = +(12345)~*(1103515245);
noise   = random/2147483647.0;

process = noise * vslider("vol",0,0,1,0.1);
```

Overview

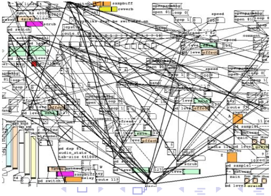
Domain-specific language, audio digital signal processing.

- synchronous
- purely functionnal
- textual block-diagram description
- statically typed

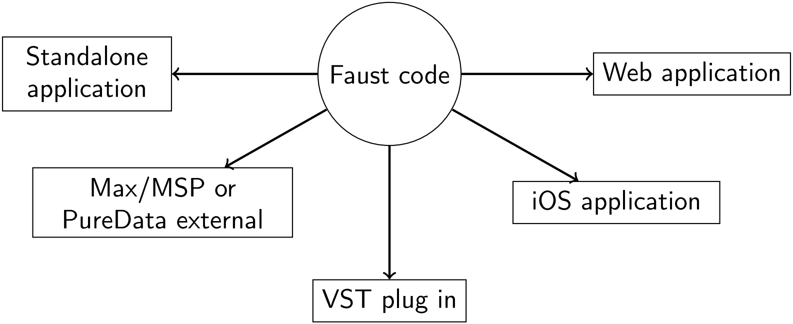



```
random  = +(12345)~*(1103515245);
noise   = random/2147483647.0;

process = noise * vslider("vol",0,0,1,0.1);
```

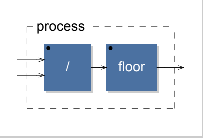


- Automatic generation of optimized C++ code
- Online compiler: `http://faust.grame.fr/`
- Multi-target compilation



CoreFaust

Block diagrams are built using 5 composition operators:

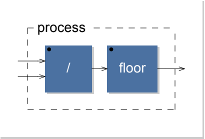


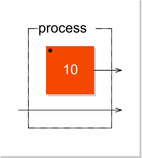

```
/ : floor
```

$$y(t) = \left\lfloor \frac{x_1(t)}{x_2(t)} \right\rfloor$$

CoreFaust

Block diagrams are built using 5 composition operators:





`/ : floor`

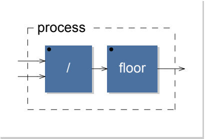
$$y(t) = \left\lfloor \frac{x_1(t)}{x_2(t)} \right\rfloor$$

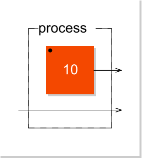
`10 , _`

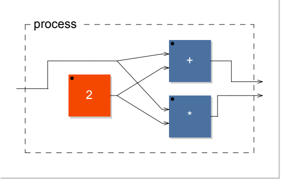
$$y(t) = (10, x(t))$$

CoreFaust

Block diagrams are built using 5 composition operators:







`/ : floor`

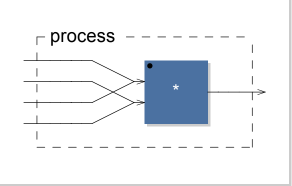
`10 , _`

`_,2 <: +,*`

$$y(t) = \left\lfloor \frac{x_1(t)}{x_2(t)} \right\rfloor$$

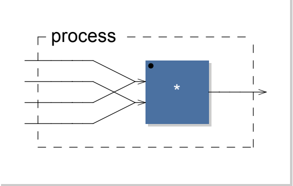
$$y(t) = (10, x(t))$$

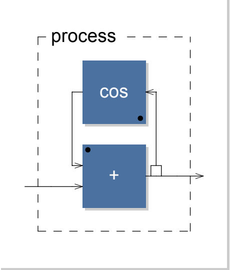
$$y(t) = (x(t) + 2, 2x(t))$$




```
_,_,_,_ :> *
```

$$y(t) = (x_1(t) + x_3(t)) * (x_2(t) + x_4(t))$$





`_,_,_,_ :> *`

`+ ~ cos`

$$y(t) = (x_1(t) + x_3(t)) * (x_2(t) + x_4(t))$$

$$\begin{cases} y(0) = x(0) + cos(0) \\ y(t) = x(t) + cos(y(t-1)) \end{cases}$$

Faust language

High-level functional language with lambdas, libraries, pattern-matching, infix notations, local environments...

```
fact(n) = case {
            (0) => 1;
            (n) => (n,fact(n-1)) : *;
};

q(x,y) = floor(x/y);   //stands for x,y : / : floor

mix = \(n).(par(i,n,_) :> _);
```

Expressiveness

Faust is Turing-complete.

The current version is monorate; wires carry only scalar signals.

We need

- different rates
- more complex data structures (vectors, matrices...)

to deal with multirate signal processing or spectral analyses.

Types

$$\text{BasicType} = \{\text{Int}, \text{Float}\}$$
$$\text{Interval} = \mathbb{R}^\omega \times \mathbb{R}^\omega \quad (\mathbb{R}^\omega = \mathbb{R} \cup \{-\omega, \omega\})$$
$$\text{Frequency} = \mathbb{Q}_+$$

Types

$$\text{BasicType} = \{\text{Int}, \text{Float}\}$$
$$\text{Interval} = \mathbb{R}^\omega \times \mathbb{R}^\omega \quad (\mathbb{R}^\omega = \mathbb{R} \cup \{-\omega, \omega\})$$
$$\text{Frequency} = \mathbb{Q}_+$$

$$\begin{array}{rcll} \tau \in \text{Type} & = & \text{BasicType} \times \text{Interval} & \textit{e.g.}\ \text{Float}[0,1] \\ & \mid & \mathbb{N}^* \times \text{Type} & \textit{e.g.}\ \textit{vector}_n(\tau) \end{array}$$

A signal has a type τ and a frequency f, written τ^f

Types

$\text{BasicType} = \{\text{Int}, \text{Float}\}$
$\text{Interval} = \mathbb{R}^\omega \times \mathbb{R}^\omega \quad (\mathbb{R}^\omega = \mathbb{R} \cup \{-\omega, \omega\})$
$\text{Frequency} = \mathbb{Q}_+$

$$\begin{array}{rcll} \tau \in \text{Type} & = & \text{BasicType} \times \text{Interval} & \textit{e.g.}\ \text{Float}[0,1] \\ & | & \mathbb{N}^* \times \text{Type} & \textit{e.g.}\ \textit{vector}_n(\tau) \end{array}$$

A signal has a type τ and a frequency f, written τ^f

Subtyping rules :

$$[x,y] \subset [x',y'] \Rightarrow b[x,y]^f \subset b[x',y']^f$$

$$\textit{Int}[x,y]^f \subset \textit{Float}[x,y]^f$$

$$\tau^0 \subset \tau^f$$

$$\tau \subset \tau' \Rightarrow \textit{vector}_n(\tau)^f \subset \textit{vector}_n(\tau')^f$$

Initial environment

Associates to predefined identifiers their input and output types.

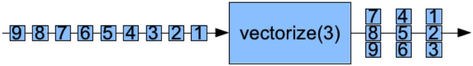
$$
\begin{aligned}
T(\_) &= \Lambda f : Rate.\tau : Type.(\tau^f) \rightarrow (\tau^f) \\
T(0) &= \Lambda f : Rate.() \rightarrow (Int[0,0]^0) \\
T(+) &= \Lambda f : Rate.\tau : Type.\tau' : Type.(\tau^f, \tau'^f) \rightarrow (\tau + \tau')^f
\end{aligned}
$$

Binary operations are well formed if :

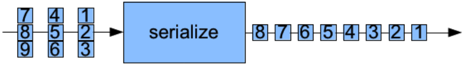
$$\exists \overline{\tau} \,/\, (\tau \subset \overline{\tau} \wedge \tau' \subset \overline{\tau}).$$

Vector primitives

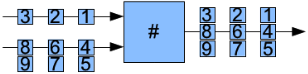
$$
\begin{aligned}
T(\texttt{vectorize}) \quad = \quad & \Lambda f : Rate.\tau : Type.n : \mathbb{N}^*. \\
& (\tau^f, Int[n,n]^0) \rightarrow (vector_n(\tau)^{f/n})
\end{aligned}
$$



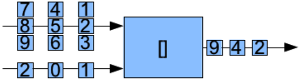
$$
T(\texttt{serialize}) = \Lambda f : Rate.\tau : Type.n : \mathbb{N}^*.(vector_n(\tau)^f) \rightarrow (\tau^{n.f})
$$



$$T(\#) \quad = \quad \Lambda f : Rate.\tau : Type.\tau' : Type.n : \mathbb{N}^*.n' : \mathbb{N}^*.$$
$$(vector_n(\tau)^f, vector_{n'}(\tau')^f) \rightarrow (vector_{n+n'}(\tau \sqcup \tau')^f)$$



$$T([\,]) = \Lambda f : Rate.\tau : Type.n : \mathbb{N}^*.(vector_n(\tau)^f, Int[0, n-1]^f) \rightarrow (\tau^f)$$



Semantic rules

$$(\mathrm{i})\ \frac{T(\mathtt{I}) = \Lambda l.z \to z' \qquad \forall (x,S) \in l, \quad l'[l^{-1}(x,S)] \in S}{T \vdash \mathtt{I} : (z \to z')[l'/l]}$$

$$(\subset)\ \frac{T \vdash E : z \to z' \qquad z_1 \subset z \qquad z' \subset z_1'}{T \vdash E : z_1 \to z_1'}$$

$$(,)\ \frac{T \vdash E_1 : z_1 \to z_1' \qquad T \vdash E_2 : z_2 \to z_2'}{T \vdash E_1, E_2 : z_1 || z_2 \to z_1' || z_2'}$$

$$(:)\ \frac{T \vdash E_1 : z_1 \to z_1' \qquad T \vdash E_2 : z_1' \to z_2'}{T \vdash E_1 : E_2 : z_1 \to z_2'}$$

Polymorphism and overloading

- All primitives are polymorphic due to abstractions in type schemes
- :> adds vector signals pointwise
- Overloading of arithmetic operators

Faust process

Definition

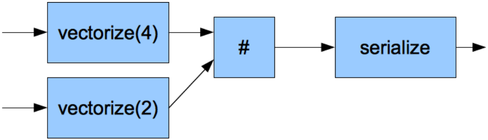
A Faust process is a well-typed expression such that all its I/O signals are scalar and of non-zero frequency.

The non-zero frequency condition ensures that all vector dimensions are known at compile time.

Goal

Static type inference of annotation free code.

```
vectorize(4),vectorize(2) : # : serialize
```



$$(\tau^{4f}, \tau'^{2f}) \rightarrow ((\tau \sqcup \tau')^{6f})$$

Type representation, environment

Isomorphic representation of types :

$$t : \text{Type} \to \text{Range} \times \text{Dimension}$$

$$\begin{array}{rcll} d \in \text{Dimension} & = & \mathit{Scalar} & (\text{for } \mathrm{b}[x, y]) \\ & | & n :: d' & (\text{for } \mathit{vector}_n(\tau)) \end{array}$$

For instance, $t(\mathit{vector}_3(\mathit{vector}_2(\mathit{Int}[0, 1]))) = (\mathit{Int}[0, 1], [3, 2])$.

Type representation, environment

Isomorphic representation of types :

$$t : \text{Type} \rightarrow \text{Range} \times \text{Dimension}$$

$$\begin{aligned} d \in \text{Dimension} \quad &= \quad \mathit{Scalar} && (\text{for b}[x, y]) \\ &\mid \quad n :: d' && (\text{for } \mathit{vector}_n(\tau)) \end{aligned}$$

For instance, $t(\mathit{vector}_3(\mathit{vector}_2(\mathit{Int}[0,1]))) = (\mathit{Int}[0,1],[3,2])$.

$$\begin{aligned} \_ \quad &\mapsto \quad (r, d, f) \rightarrow (r, d, f), \emptyset \\ 0 \quad &\mapsto \quad () \rightarrow (\mathit{Int}[0,0], \mathit{Scalar}, 0), \emptyset \end{aligned}$$

Type representation, environment

Isomorphic representation of types :

$$t : \text{Type} \rightarrow \text{Range} \times \text{Dimension}$$

$$\begin{array}{rcll} d \in \text{Dimension} & = & \mathit{Scalar} & (\text{for } \mathrm{b}[x, y]) \\ & | & n :: d' & (\text{for } \mathit{vector}_n(\tau)) \end{array}$$

For instance, $t(\mathit{vector}_3(\mathit{vector}_2(\mathit{Int}[0,1]))) = (\mathit{Int}[0,1], [3,2])$.

$$\begin{array}{rcl} \_ & \mapsto & (r, d, f) \rightarrow (r, d, f), \emptyset \\ 0 & \mapsto & () \rightarrow (\mathit{Int}[0,0], \mathit{Scalar}, 0), \emptyset \\ \# & \mapsto & (r, n :: d, f), (r', n' :: d', f) \\ & & \rightarrow (r \sqcup r', (n + n') :: d, f), \{d = d'\} \end{array}$$

Algorithm: constraint generation

$$
\begin{array}{rcl}
\texttt{type}(E, L_0) & = & \textit{match } E \textit{ with} \\
\texttt{I} & \mapsto & \texttt{New}\ (\textit{Env}\ (\texttt{I}), L_0) \\
E_1, E_2 & \mapsto & (I_1 || I_2 \rightarrow O_1 || O_2), \mathcal{C}_1 \cup \mathcal{C}_2, L_2 \\
E_1 : E_2 & \mapsto & (I_1 \rightarrow O_2), \mathcal{C}_1 \cup \mathcal{C}_2 \cup \textit{subbeam}(O_1, I_2), L_2 \\
\ldots & &
\end{array}
$$

where $(I_i \rightarrow O_i, \mathcal{C}_i, L_i) = \texttt{type}(E_i, L_{i-1})$,
$\texttt{New}$ creates a new instance of $\textit{Env}(\texttt{I})$ with fresh variables,
L_i is the set of used variables.

Constraints reduction

- Dimension equalities and frequency relations are reduced to numerical equalities and substitutions with inference systems:

$$\{\mathtt{d}_i = d\} \cup \mathcal{D}; \mathcal{N}; \mathcal{S} \Rightarrow \mathcal{D}[d/\mathtt{d}_i]; \mathcal{N}; \mathcal{S}[d/\mathtt{d}_i] \cup \{\mathtt{d}_i \mapsto d\}$$
$$\{n :: d \ = \ n' :: d'\} \cup \mathcal{D}; \mathcal{N}; \mathcal{S} \Rightarrow \{d = d'\} \cup \mathcal{D}; \mathcal{N} \cup \{n = n'\}; \mathcal{S}$$
$$\{Scalar \ = \ n :: d\} \cup \mathcal{D}; \mathcal{N}; \mathcal{S} \Rightarrow \text{fail}$$
$$\ldots$$

- Range relations can lead to
  - static computation of vector dimensions
  - static verification of arithmetic relations
  - dynamic clipping of signals

Correctness

Theorem (Soundness)

Let E be a Faust expression, and $(I \to O, \mathcal{C}, L) = \mathit{type}(E, \emptyset)$. Then, if $\mathcal{M}$ is a model defined on L such that $\mathcal{M} \models \mathcal{C}$, then $T \vdash E : t^{-1}(\mathcal{M}(I \to O))$.

Theorem (Completeness)

Let E be a Faust expression, such that $T \vdash E : z \to z'$. Then $\mathit{type}(E, \emptyset) = (I \to O, \mathcal{C}, L)$, and there exists a model $\mathcal{M}$ defined on L such that $\mathcal{M} \models \mathcal{C}$ and $t^{-1}(\mathcal{M}(I \to O)) \subset z \to z'$.

Conclusion

- Static inference of rate relations and vector dimensions.
  OCaml prototype.

  How to gain precision on data signal types?

- Expressive power of Faust

  Is the vector extension well suited for DSP algorithms?
  Study cases with IRCAM

Type inference in the multirate audio DSP language Faust

Pierre Beauguitte

Centre de recherche en informatique - MINES ParisTech

SYNCHRON 2012