smc 2010 | 7th Sound and Music Computing Conference
21 - 24 July 2010
Universitat Pompeu Fabra
Barcelona

Dependent Types for Multirate Faust

Extending the Faust audio programming language for vector and multirate signal processing

A Vector API for Faust

Vector signals map discrete time to vector values (ordered collections of values). Signal rates are modified by vector manipulation operations.

Operation	*Semantics*
`vectorize`	Collects *n* consecutive samples (the constant value *n* is provided in the type of the scalar signal that is the second argument) from an input signal and outputs an *n*-vector signal.
`serialize`	Maps a signal of *n*-vectors to the signal of their linearized *n* elements.
`[]`	Provides, using as inputs a signal of vectors and one of integer indexes, an output signal of successively indexed vector elements.
`#`	Builds a signal of concatenated vectors from its two vector signal inputs. The size of the output vectors is the sum of the vector sizes of its arguments.
`{}`	The empty vector.

Faust

Faust is a **functional programming language** specifically designed for synchronous real-time signal processing and synthesis. A Faust program describes a signal processor `process` that maps input signals to output signals. The Faust compiler can perform automatic parallelization and produces highly optimized C++ code.

The following top-level `process` signal processor halves its input:

```
process = _,0.5 : * ;
```

where « `,` » and « `:` » put two processors in parallel and sequence, while « `_` » denotes the identity signal processor.

An Example: Haar Filtering

```
down = vectorize(2),1 : [] ;
mean = _ <: _,mem :> _,2 : / ;
left = _,! ;
process = _ <: (mean:down),down <: left,- ;
```

`down`: builds 2-vectors from its input, and picks the second element.
`mean`: computes the mean of successive elements in its input signal.
`left`: takes a pair of signals, keeping the first one.
`process`: copies its input; the first copy is averaged, and both copies are downsampled; the outputs are the average signal and the difference of the downsampled copies.

Vector Operations as Static Rate Transformers

Key Insights

- **Dependent type system** based on integer value spans:

$$int[n,m]$$

- Connection-matching constraints relaxed via **subtyping**:

$$n' \leq n \text{ and } m \leq m' \implies int[n,m] \subset int[n',m']$$

- **Sum types** for mixing signals (`:>`):

$$int[n,m] + int[n',m'] = int[n+n', m+m']$$

- **Signal rate algebra** of rational numbers f in $\mathbb{Q}(*,/)$
- **Rated signal types** t^f, grouped in **impedances** z
- **Vector datatype** constructor $\mathrm{vector}_n(t)^f$
- Vector operations as static **rate transformers**:

Operation	*Type*
`vectorize`	$(t^f, int[n,n]^{f'}) \to (\mathrm{vector}_n(t)^{f/n})$
`serialize`	$(\mathrm{vector}_n(t)^f) \to (t^{f*n})$
`[]`	$(\mathrm{vector}_n(t)^f, int[0,n-1]^f) \to (t^f)$
`#`	$(\mathrm{vector}_n(t)^f, \mathrm{vector}_m(t)^f) \to (\mathrm{vector}_{n+m}(t)^f)$
`{}`	$() \to (\mathrm{vector}_0(t)^f)$

- Static and dynamic (denotational) semantics consistency
- Signal rate correctness theorem

Faust Typing and Rating Static Semantics

$$(\mathrm{i})\ \frac{\begin{array}{ccc} T(\mathtt{I}) & = & \Lambda l.(z,z') \\ \forall (x,S) \in l & . & l'(x) \in S \end{array}}{T \vdash \mathtt{I} : (z,z')[l'/l]}$$

$$(:)\ \frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_1', z_2') \end{array}}{T \vdash \mathtt{E}_1 : \mathtt{E}_2 : (z_1, z_2')}$$

$$(<:)\ \frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \\ z_1' & \prec & z_2 \end{array}}{T \vdash \mathtt{E}_1 <: \mathtt{E}_2 : (z_1, z_2')}$$

$$(,)\ \frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \end{array}}{T \vdash \mathtt{E}_1, \mathtt{E}_2 : (z_1 \| z_2, z_1' \| z_2')}$$

$$(:>)\ \frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \\ z_1' & \succ & z_2 \end{array}}{T \vdash \mathtt{E}_1 :> \mathtt{E}_2 : (z_1, z_2')}$$

$$(\subset)\ \frac{\begin{array}{ccc} T & \vdash & \mathtt{E} : (z, z') \\ z' & \subset & z_1' \\ z_1 & \subset & z \end{array}}{T \vdash \mathtt{E} : (z_1, z_1')}$$

$$(\sim)\ \frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \\ z_2 & = & z'[1, |z_2|] \\ z_2' & = & z_1[1, |z_2'|] \end{array}}{T \vdash \mathtt{E}_1 \sim \mathtt{E}_2 : (z_1[|z_2'|+1, |z_1|], \widehat{z'})}$$

MINES ParisTech | GRAME CENTRE NATIONAL DE CRÉATION MUSICALE | Université Jean Monnet SAINT-ETIENNE CIEREC - EA 3068 | ircam Centre Pompidou

Contact: Pierre JOUVELOT, Yann ORLAREY
Email: pierre.jouvelot@mines-paristech.fr, orlarey@grame.fr
Funding: ANR, ASTREE Project (2008 CORD 003 01)

A,s:t,rée