Dependent Typing for Multirate Faust

Pierre Jouvelot

MINES ParisTech, PSL Research University

Joint work with Yann Orlarey (GRAME)

Séminaire Deducteam

September 19, 2014

Agenda

- Let's have...fun
- Motivations
- Faust
  - Core
  - Static and dynamic semantics
  - Correctness theorems
- Multirate Core Faust extension
- Future work

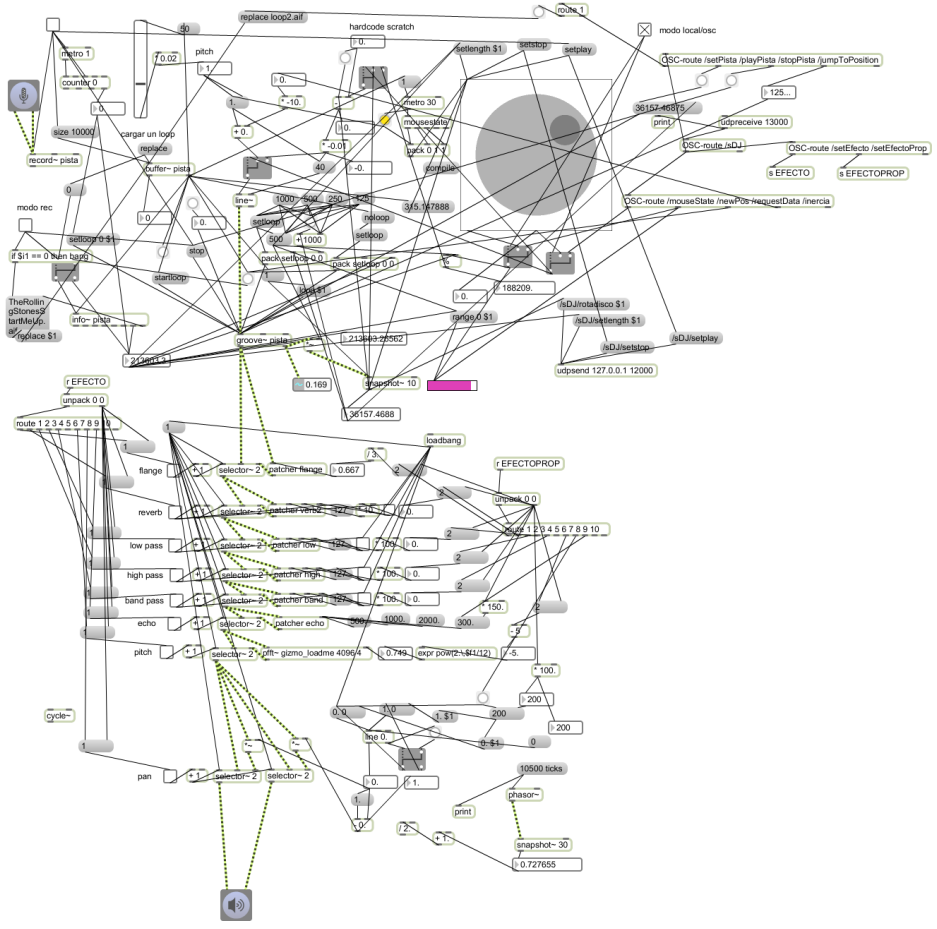
Faust: Motivations (1)

- Music as part of mathematics
- Computer music research centers:
  - IRCAM, GRAME...
  - CCRMA (Stanford), CNMAT (Berkeley)...
- Music stringent performance requirements:
  - Berkeley ParLab [CACM 2009]
  - Real-time constraints
  - Dedicated hardware (4X...)
  - Lots of software developments (MIDI, audio, HMI...)

Faust: Motivations (2)

Domain-Specific Languages for audio processing[1]:

- Music I (Bell Labs, 50s)
- CSound (MIT, 86)
- Max/MSP (IRCAM, 80s)
- OpenMusic (IRCAM, 98)
- Faust (GRAME, 02)...



[1] Figure: http://www.ateliermodulaire.org

Faust: Project

- Functional Audio Stream (`http://faust.grame.fr`)
- Designed at GRAME in 2002 (Yann Orlarey)
- High abstraction level, with C++-like run-time performance
- C++-targeted compiler, with a rich environment: FaustWorks, FaustLive, FaustWeb, Faustine...
- Applications: moForte, Guitarix, music installations, preservation, education...
- ANR FEEVER Project (2013-2016):
  - ”Faust as the Javascript of audio”
  - GRAME, MINES ParisTech, IRISA, U. St-Etienne

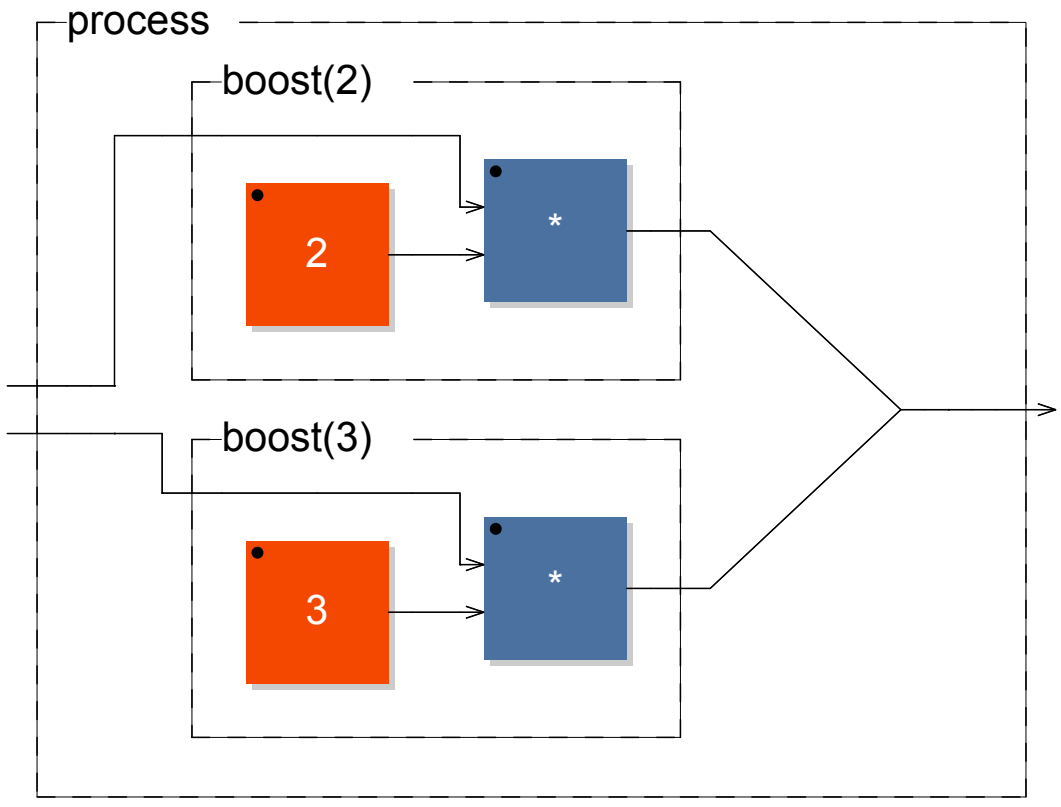
Faust: Design

- Signal processors as building blocks
- Text and block diagram prettyprinting
- Two-tier functional approach:
  - Macro language: full untyped λ- calculus, with currification
  - Core Faust: algebra of typed signal processors
- `process`, from input to output sampled (audio) signals
- Advanced dependent type system for maximum performance

Faust: Examples (1/3)

Stereo differential mixer:

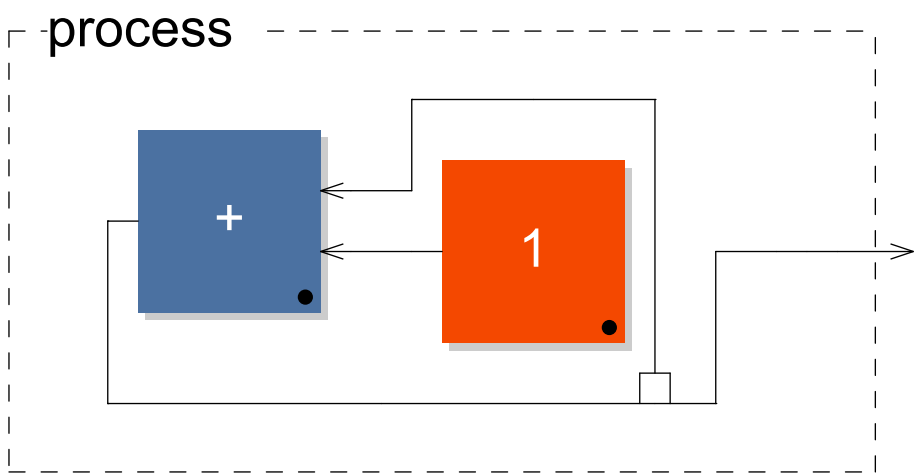
```
boost(p) = _,p : *;
process = _,_ : (boost(2),boost(3)) >: _;
```



Faust: Examples (2/3)

Infinite signal of successive integers:

```
process = _ ~ (1,_ : +);
```



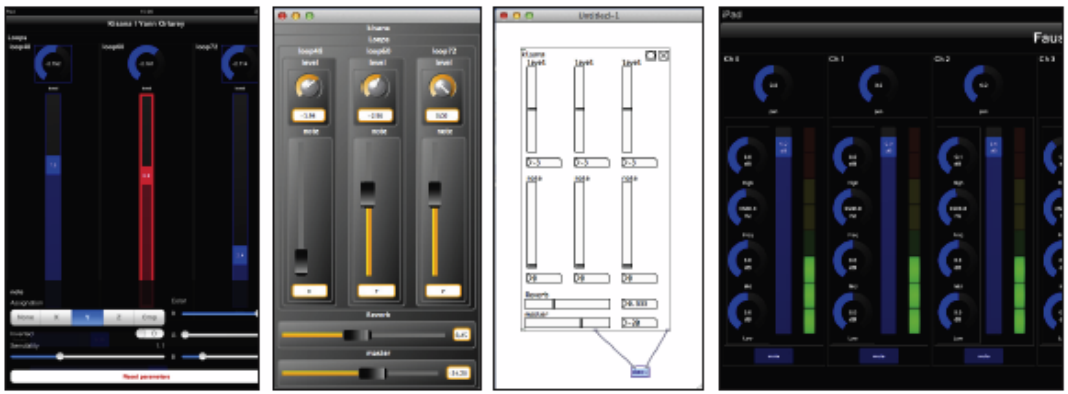
Faust: Examples (3/3)

8-voice stereo mixer:

```
vol = component("volume.dsp");
pan = component("panpot.dsp");
vumeter = component("vumeter.dsp").vmeter;
mute = *(1 - checkbox("mute"));

voice(v) = vgroup("voice %v",
  mute : hgroup("[2]", vol : vumeter) : pan);
stereo = hgroup("stereo out", vol, vol);
process = hgroup("mixer", par(i, 8, voice(i)) :> stereo);
```

Multiple GUIs:



Faust: Core Language Syntax

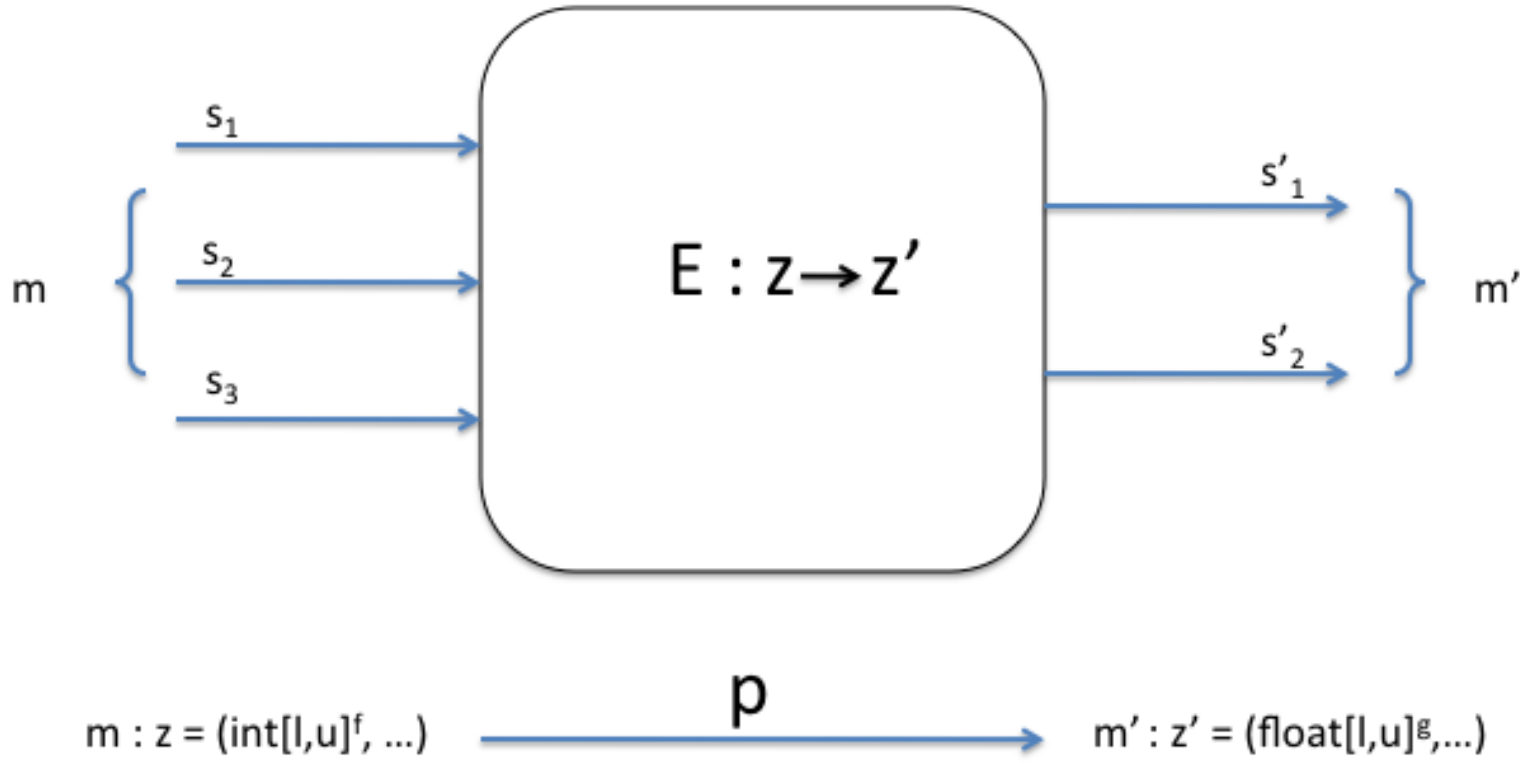
- Core Faust *expressions* $\mathtt{E}$:

$$\begin{aligned} \mathtt{E} \quad ::= \quad & \mathtt{I} \mid \\ & \mathtt{E}_1 : \mathtt{E}_2 \mid \mathtt{E}_1, \mathtt{E}_2 \mid \\ & \mathtt{E}_1 <: \mathtt{E}_2 \mid \mathtt{E}_1 :> \mathtt{E}_2 \mid \\ & \mathtt{E}_1 \sim \mathtt{E}_2 \end{aligned}$$

- *Bindings* $\mathtt{I} = \mathtt{E}$
- Currification, via rewriting such as `_:+(2)` to `_,2:+`

Faust: Signal Processors

A Faust expression E is a *signal (beam)* processor *p*



Faust: Core Static Semantics

- Dependent type system based on *intervals* $[n, m]$
- *Types* $t \in \text{Type} = \text{BasicType} \times \text{int} \times \text{int}$
- "Connection" constraints relaxed via *subtyping*:

$$\frac{[n, m] \quad \subseteq \quad [n', m']}{\text{int}[n, m] \quad \subset \quad \text{int}[n', m']}$$

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

$$b[n, m] + b[n', m'] \quad = \quad b[n + n', m + m']$$

- *Beam types* $z \in \text{BeamType} = \bigcup_n \text{Type}^n$
- Fixed typing environment T
- Typing judgments $T \vdash \mathbb{E} : z \to z'$

Faust: Core Static Rules

$$\text{(i)}\ \frac{\begin{array}{rcl} T(\mathtt{I}) & = & \Lambda I.z \to z' \\ \forall (x,S) \in I & . & I'[I^{-1}(x,S)] \in S \end{array}}{T \vdash \mathtt{I} : (z \to z')[I'/I]}$$

$$(\subset)\ \frac{\begin{array}{rcl} T & \vdash & \mathtt{E} : z \to z' \\ z' & \subset & z'_1 \\ z_1 & \subset & z \end{array}}{T \vdash \mathtt{E} : z_1 \to z'_1}$$

$$(:)\ \frac{\begin{array}{rcl} T & \vdash & \mathtt{E}_1 : z_1 \to z'_1 \\ T & \vdash & \mathtt{E}_2 : z'_1 \to z'_2 \end{array}}{T \vdash \mathtt{E}_1 : \mathtt{E}_2 : z_1 \to z'_2}$$

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

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

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

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

Core Faust: Dynamic Domains

- *Values*: $v \in \mathrm{Val} = B$
- *Signals* as streams of *typed* values:
  $s \in \mathrm{Signal} = \mathrm{Time} \to \mathrm{Val}$
- *Beams* as groups of signals: $m \in \mathrm{Beam} = \bigcup_n \mathrm{Signal}^n$
- *Processors* as beam transformers:
  $p \in \mathrm{Proc} = \mathrm{Beam} \to \mathrm{Beam}$
- Binding *states*: $r \in \mathrm{State} = \mathrm{Ide} \to \mathrm{Proc}$

Core Faust: Denotational Rules

- States r bind identifiers to beam processors:
  - $r_0(\_) = \lambda(s).(s)$
  - $r_0(0) = \lambda().(\lambda t.0)$
- $E : \mathrm{Exp} \rightarrow \mathrm{State} \rightarrow \mathrm{Proc}$
- Semantic equations:

$$
\begin{aligned}
E[\![\mathtt{I}]\!]r &= r(\mathtt{I}) \\
E[\![\mathtt{E}_1 : \mathtt{E}_2]\!]r &= p_2 \circ p_1 \\
E[\![\mathtt{E}_1, \mathtt{E}_2]\!]r &= \lambda m.p_1(m[1, d_1]) \| p_2(m[d_1 + 1, d_1 + d_2]) \\
E[\![\mathtt{E}_1 \sim \mathtt{E}_2]\!]r &= \lambda m.fix(\lambda m'.p_1(p_2(@(m'[1, d_2])) \| m)) \\
&\quad \text{where } @(m) = \|_m \lambda s.\mathrm{delay}(s, \lambda t.1)
\end{aligned}
$$

with $p_i = E[\![\mathtt{E}_i]\!]r$ and $\mathrm{Signal}^{d_i} = dom(p_i)$

Core Faust: Subject Reduction

Type preservation by Core Faust denotational semantics:

Theorem (Subject Reduction)

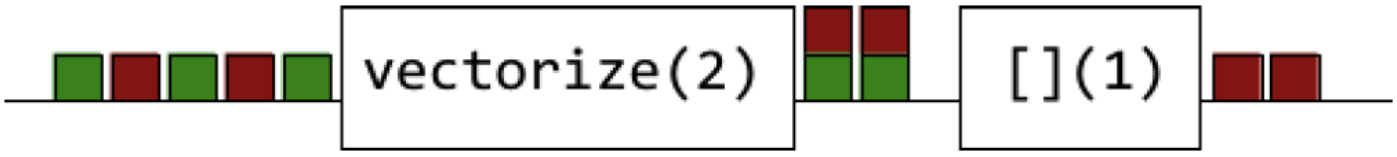
For all $\mathbb{E}, T, z, z', r, m$, *if*

$$\vdash T, r$$
$$m : z$$
$$T \vdash \mathbb{E} : (z, z')$$

then $p(m) : z'$ *and* $dim(p) = (|z|, |z'|)$, *where* $p = E[\![\mathbb{E}]\!]r$.

Multirate Processing: Key Insights

- Rate processing as data aggregation operations
- Vector operations as static frequency transformers

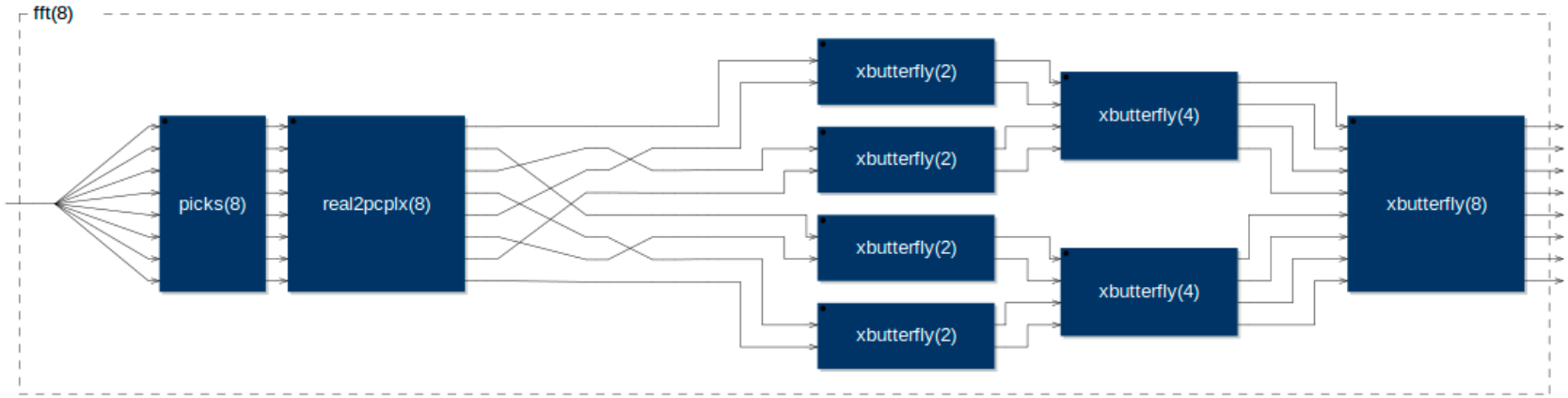



```
down = vectorize(2):[](1);
```

Multirate Processing: FFT Example

Faustine: A Multirate Faust Interpreter ([FLOPS'14])

```
import ("complex.lib");
picks(n) = par(i, n, [i]);
fft(n) =
  _ <: picks(n):real2pcplx(n):shuffle(n):butterflies(n);
process =
  vectorize(8):fft(8):pcplx_modules(8):nconcat(8):serialize;
```



Multirate Faust: Formal Semantics

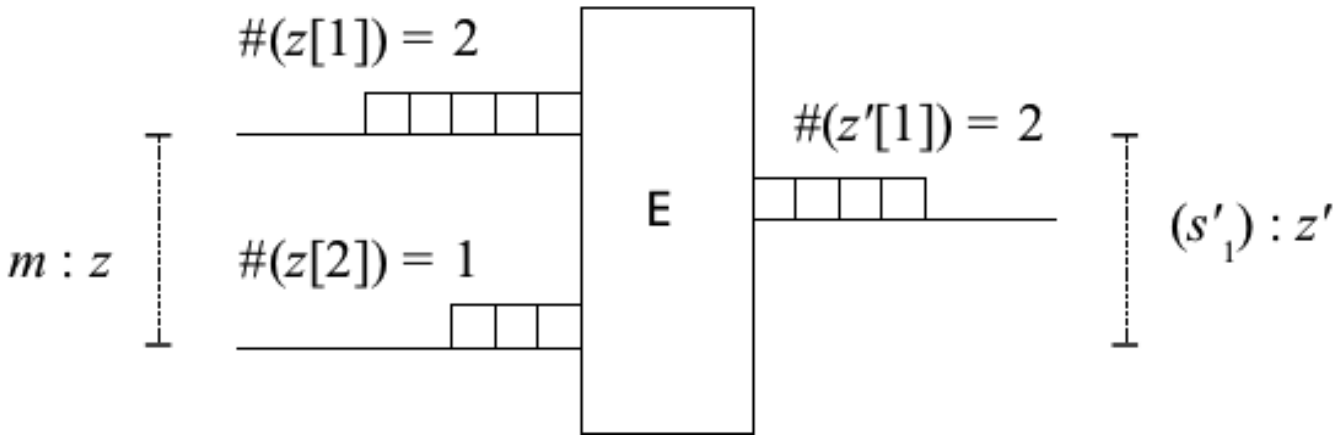
- *Types* $t \in \text{Type} \mathrel{+}= \text{vector}_n(t)$
- *Frequencies* as static clocks: $f \in \text{Rate} = \mathbb{Q}^+$
- *Rated types* $t^f \in \text{Type}^\sharp = \text{Type} \times \text{Rate}$
- Adding an $iso(z_2)$ condition on Rule ($\sim$)
- Extended typing environment:

$$\begin{aligned} \texttt{vectorize} &: (t^f, \text{int}[n,n]^{f'}) \to (\text{vector}_n(t)^{f/n}) \\ \texttt{serialize} &: (\text{vector}_n(t)^f) \to (t^{f*n}) \\ \texttt{[]} &: (\text{vector}_n(t)^f, int[0, n-1]^f) \to (t^f) \end{aligned}$$

- Extended initial store r_0:
  - $r_0(\texttt{vectorize}) = \lambda(s_1, s_2).(\lambda t.\lambda i.s_1(i + nt), \text{ where } n = s_2(0))$
  - $r_0(\texttt{serialize}) = \lambda(s).(\lambda t.\ \bot \text{ if } n = \underline{s(0)} = 0, s(\lfloor t/n \rfloor)(\text{mod}(t,n)) \text{ otherwise})$
  - $r_0(\texttt{[]}) = \lambda(s_1, s_2).(\lambda t.s_1(t)(s_2(t)))$

Multirate Faust: Rate Properties

- Characteristics of Signal s: *support* $\underline{s}$ and clock $\sharp(s)$
- Idea: $\underline{s}/\sharp(s)$ as time needed to exhaust the domain of s
- Larger supports for high frequency signals:



- Delay operations extend supports with 0s: $@_T(\mathbb{E})$

Multirate Faust: Rate Correctness

Time bound $\mu(m, z)$, in $\mathbb{N} \cup \{\omega\}$ (with $n/0 = \omega$):

$$\mu(m, z) = \min_{i \in [1, |z|]} \lfloor \underline{m[i]} / \sharp(z[i]) \rfloor$$

Theorem (Rate Correctness)

For all $\mathbb{E}, T, z, z', c, r, m$ *and* m', *if*

$$r : T \ ,$$
$$m : z \ ,$$
$$T \vdash \mathbb{E} : (z, z') \ ,$$

then either $|z'| = 0$ *or* $\mu(E[\![\mathbb{E}]\!]rm, z') \leq \mu(m, z) + @_T(\mathbb{E})$.

Multirate Faust: Type Checking

- Dependent type checking
- Similar frameworks: refinement or liquid types
- SMT solver-based approach:
  Z3 (Microsoft), mathsat (U. Trento)
- Type checker prototype under development (OCaml)
- Completeness issues:

```
process = (_,E:vectorize),E':[];
```

  - Characterization of typing constraints
  - User assertions
  - Proof assistant (?)
  - DSL calls for domain-driven design choices

Future Work

- Type checker:
  - Complete implementation
  - Correctness theorems and proofs
  - Loop bound inference via abstract interpretation or fancier predicates?
  - Integration within the Faust compiler (C++)
- (Coq-assisted?) proofs of process properties:
  - Bounded-input bounded-output (BIBO)
  - Integer overflow detection
  - Stabilization
  - WCET analysis to ensure the *synchronous hypothesis*
- Asynchronous extension
- Curry-Howard correspondence and synchronous programming languages?

Conclusion

- Faust, a DSL for audio and music processing
- Multirate Faust with vectors as frequency transformers
- Music as a rich domain for formal semantic tools and techniques (dependent typing, SMT...)
- DSLs, as a venue for theoretical investigations:
  - Embedded DSL
  - Specialized static semantics [Cheng&Rival for Excel]

Dependent Typing for Multirate Faust

Pierre Jouvelot

MINES ParisTech, PSL Research University

Joint work with Yann Orlarey (GRAME)

Séminaire Deducteam

September 19, 2014