Reasoning About Sound Programs

Emilio Jesús Gallego Arias

Joint work with O. Hermant & P. Jouvelot
MINES ParisTech, PSL Research University, France

Rennes, 15 Avril 2015
### Some Music DSLs

<table>
<thead>
<tr>
<th>DARMS</th>
<th>LPC</th>
<th>MCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCMP</td>
<td>Mars</td>
<td>MUSIC</td>
</tr>
<tr>
<td>DMIX</td>
<td>MASC</td>
<td>III/IV/V</td>
</tr>
<tr>
<td>Elady</td>
<td>Max</td>
<td>MCL</td>
</tr>
<tr>
<td>Endace</td>
<td>MidiLisp</td>
<td>MusicLogo</td>
</tr>
<tr>
<td>Enterpnea</td>
<td>MidiLogo</td>
<td>Music1000</td>
</tr>
<tr>
<td>FausX</td>
<td>MODE</td>
<td>MUSIC7</td>
</tr>
<tr>
<td>Flavors</td>
<td>MOM</td>
<td>Musicianx</td>
</tr>
<tr>
<td>Band</td>
<td>Music</td>
<td>MUSIGOL</td>
</tr>
<tr>
<td>Fluxus</td>
<td>MSX</td>
<td>MusicXML</td>
</tr>
<tr>
<td>FOIL</td>
<td>MUS10</td>
<td>MusicXML</td>
</tr>
<tr>
<td>FORMES</td>
<td>MUS8</td>
<td>Musicianx</td>
</tr>
<tr>
<td>FORMULA</td>
<td>MUSCOMP</td>
<td>MUSIGOL</td>
</tr>
<tr>
<td>Fugue</td>
<td>MUS10</td>
<td>MUSIC8</td>
</tr>
<tr>
<td>Gibber</td>
<td>MUS10</td>
<td>MUS10</td>
</tr>
<tr>
<td>GROOVE</td>
<td>MUS10</td>
<td>MUS10</td>
</tr>
<tr>
<td>GUIDO</td>
<td>MUS11</td>
<td>MUS11</td>
</tr>
<tr>
<td>HARP</td>
<td>MUSIC10</td>
<td>MUS11</td>
</tr>
<tr>
<td>Hashtore</td>
<td>MUSIC360</td>
<td>MUSIC360</td>
</tr>
<tr>
<td>HMSL</td>
<td>MUSIC48</td>
<td>MUSIC48</td>
</tr>
<tr>
<td>INV</td>
<td>MUSIC4F</td>
<td>MUSIC4F</td>
</tr>
<tr>
<td>invokator</td>
<td>KEIN</td>
<td>PLACOMU</td>
</tr>
<tr>
<td>Csound</td>
<td>CyberBand</td>
<td>PLAY1</td>
</tr>
<tr>
<td>CyberBand</td>
<td></td>
<td>PLAY2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PMX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POCO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POD6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POD7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Puredata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PWGL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ravel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SALIERI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCORE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ScoreFile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCRIPT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SIREN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMDL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMOKE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SSP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SSSP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supercollider</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Symbolic</td>
</tr>
</tbody>
</table>
Some Music DSLs

<table>
<thead>
<tr>
<th>DARMS</th>
<th>DCMP</th>
<th>DMIX</th>
<th>Eludy</th>
<th>EtaAC</th>
<th>Euterpe</th>
<th>Euterpea</th>
<th>Faust</th>
<th>Faur</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC</td>
<td>Mars</td>
<td>MASC</td>
<td>Max</td>
<td>Midliap</td>
<td>Midliap</td>
<td>MOOD6</td>
<td>MCL</td>
<td>MUSIC</td>
</tr>
<tr>
<td>III/IV/V</td>
<td>MusicLogo</td>
<td>Music1000</td>
<td>MUSIC7</td>
<td>PLACONu</td>
<td>PLAY1</td>
<td>PLAY2</td>
<td>PMX</td>
<td>POCO</td>
</tr>
</tbody>
</table>
Let’s assume a simple IIR filter:

\[ \text{smooth}_n = (1 - c) \cdot x_n + c \cdot \text{smooth}_{n-1} \]
Let’s assume a simple IIR filter:

\[ \text{smooth}_n = (1 - c) \cdot x_n + c \cdot \text{smooth}_{n-1} \]

What would we like to know about it?
\[ \text{smooth}_n = (1 - c) \cdot x_n + c \cdot \text{smooth}_{n-1} \]

Natural questions are:

- Frequency response;
- Stability;
- Linearity/Time Invariance.

Standard DSP theory gives answers.
\[ smooth_n = (1 - c) \cdot x_n + c \cdot smooth_{n-1} \]

Natural questions are:

- Frequency response;
- Stability;
- Linearity/Time Invariance.

Standard DSP theory gives answers.

What about the implementation of the filter?

We dive into the realm of PL theory!
smooth\(_n\) = (1 - c) \cdot x_n + c \cdot \text{smooth}_{n-1}

Natural questions are:

- Frequency response;
- Stability;
- Linearity/Time Invariance.

Standard DSP theory gives answers.

What about the implementation of the filter?

We dive into the realm of PL theory!

Paradigm shift!
Faust

- Functional PL for digital signal processing.
- Synchronous paradigm, geared towards audio.
- Programs: circuits/block diagrams with feedback.
- Semantics: streams of samples.
- *Efficiency is crucial.*
- Created in 2000 by Yann Orlarey et al. at GRAME.
- Mature, compiles to more than 14 platforms.
Faust’s Ecosystem

Users:

- Grame: Multiple projects, main developer.
- Stanford: Class/books on signal processing, STK instrument toolkit, Faust2android, Mephisto. . .
- Ircam: Acoustic libraries, effects libraries, . . .
- Other: Guitarix, moForte guitar, etc...
Faust’s Ecosystem

Users:

- Grame: Multiple projects, main developer.
- Stanford: Class/books on signal processing, STK instrument toolkit, Faust2android, Mephisto...
- Ircam: Acoustic libraries, effects libraries,...
- Other: Guitarix, moForte guitar, etc...

It has its market! Much easier than dwelling into C.
Faust’s Ecosystem

Users:

- Grame: Multiple projects, main developer.
- Stanford: Class/books on signal processing, STK instrument toolkit, Faust2android, Mephisto...
- Ircam: Acoustic libraries, effects libraries,...
- Other: Guitarix, moForte guitar, etc...

It has its market! Much easier than dwelling into C.

Recent Events:

- Faust day at Stanford, LAC 2015.
- Faust program competition (€2,000).
- FEEVER project :)
Syntax and Well-Formedness

**TERM**
\[ \vdash !_ : 1 \to 0 \]

**ID**
\[ \vdash _ : 1 \to 1 \]

**PAR**
\[ \vdash f_1 : i_1 \to o_1 \quad \ldots \quad \vdash f_n : i_n \to o_n \]
\[ \vdash (f_1, \ldots, f_n) : \sum_{j} i_j \to \sum_{j} o_j \]

**COMP**
\[ \vdash f : i \to k \quad \vdash g : k \to o \]
\[ \vdash (f : g) : i \to o \]

**PAN**
\[ \vdash f : i \to k \quad \vdash g : k \ast n \to o \quad 0 < k \land 0 < n \]
\[ \vdash f <: g : i \to o \]
Feedback

$$\frac{f : o_g + i_f \rightarrow i_g + o_f}{\text{FEED}} \quad \frac{g : i_g \rightarrow o_g}{\text{FEED}}$$

$$\vdash f \sim g : i_f \rightarrow i_g + o_f$$

Diagram for $+ \sim \sin$: 

![Diagram](image)
Back to the Filter

$\text{smooth}_n = (1 - c)x_n + c \cdot \text{smooth}_{n-1}$

Using Faust:

$\text{smooth}(c) = *(1-c) : + \sim *(c)$

[For $c = 0.9$]
Feedback Delay Network

\[
\text{fdnrev}(N, \text{dp}, \text{freqs}, \text{durs}, \text{loopgainmax}) = \text{delaylines} \sim (\text{delayfilters} : \text{feedbackmatrix})
\]

where

\[
\begin{align*}
\text{delaylines} & = \text{rep}(N,i,\text{delay}(\text{dp}[i])); \\
\text{delayfilters} & = \text{rep}(N,\text{filter}(\text{freqs},\text{durs})); \\
\text{feedbackmatrix} & = \text{bhadamard}(N);
\end{align*}
\]
Feedback Delay Network

\[ \text{fdnrev}(N, \text{dp}, \text{freqs}, \text{durs}, \text{loopgainmax}) = \text{delaylines} \sim (\text{delayfilters} : \text{feedbackmatrix}) \]

where
\[
\begin{align*}
\text{delaylines} &= \text{rep}(N, i, \text{delay}(\text{dp}[i])); \\
\text{delayfilters} &= \text{rep}(N, \text{filter}(\text{freqs}, \text{durs})); \\
\text{feedbackmatrix} &= \text{bhadamard}(N);
\end{align*}
\]
PL & Faust

- Causal/Synchronous Programming. See next week’s talk!
- Functional Reactive Programming/Arrows.
- String Diagrams, Monoidal Closed Categories.

Data-intensive vs control-intensive require quite different control techniques. [Berry, 2000] Spectral processing may open a new gap from all of those! Some related DSL: VOBLA, Ziria, Halide, Darkroom, Julia.
PL & Faust

- Causal/Synchronous Programming. See next week’s talk!
- Functional Reactive Programming/Arrows.
- String Diagrams, Monoidal Closed Categories.

*Data-intensive vs control-intensive require quite different control techniques. [Berry, 2000]*
PL & Faust

- Causal/Synchronous Programming. See next week’s talk!
- Functional Reactive Programming/Arrows.
- String Diagrams, Monoidal Closed Categories.

*Data-intensive vs control-intensive require quite different control techniques. [Berry, 2000]*

Spectral processing may open a new gap from all of those!
PL & Faust

- Causal/Synchronous Programming. See next week’s talk!
- Functional Reactive Programming/Arrows.
- String Diagrams, Monoidal Closed Categories.

Data-intensive vs control-intensive require quite different control techniques. [Berry, 2000]

Spectral processing may open a new gap from all of those! Some related DSL: VOBLA, Ziria, Halide, Darkroom, Julia.
Real-time Linear Processing.
Real-time Non-linear Processing.
Frequency Domain Processing.
Non-necessarily causal.
Filters, Feedback Networks, Interpolation.
Windowing!
Numerical issues.
Nyquist/precision/aliasing.
Verification in DSP/Faust

Use mechanized techniques to ensure correct behavior.
Verification in DSP/Faust

Use mechanized techniques to ensure correct behavior.

- Model checking/automata.
- Program analysis/logics.
- Strong type systems/correct by construction.
Verification in DSP/Faust

Use mechanized techniques to ensure correct behavior.

- Model checking/automata.
- Program analysis/logics.
- Strong type systems/correct by construction.
- Main efforts in DSP audio are numeric so far [Souari, Tahar, et al].
Verification in DSP/Faust

Use mechanized techniques to ensure correct behavior.

- Model checking/automata.
- Program analysis/logics.
- Strong type systems/correct by construction.
- Main efforts in DSP audio are numeric so far [Souari,Tahar, et al].
- Other non-DSP efforts (Antescofo, [Poncelet et. al]).
Verification in DSP/Faust

Use mechanized techniques to ensure correct behavior.

- Model checking/automata.
- Program analysis/logics.
- Strong type systems/correct by construction.
- Main efforts in DSP audio are numeric so far [Souari, Tahar, et al].
- Other non-DSP efforts (Antescofo, [Poncelet et. al]).

Problems with Audio:
bad sound, stability/glitches, under/overflows, time,
safety/security, remote distribution.
We need more!
A Case Study: Stability

Test-bed: use Coq
Coq is a theorem prover that provides very strong evidence as compared to Mathlab, etc...
A Case Study: Stability

*Test-bed: use Coq*

Coq is a theorem prover that provides very strong evidence as compared to Mathlab, etc.

**Stability of Smooth**

When is smooth stable?

\[
\text{smooth}_n = (1 - c)x_n + c \cdot \text{smooth}_{n-1}
\]
A Case Study: Stability

Test-bed: use Coq
Coq is a theorem prover that provides very strong evidence as compared to Mathlab, etc. . .

Stability of Smooth
When is smooth stable?

\[
\text{smooth}_n = (1 - c)x_n + c \cdot \text{smooth}_{n-1}
\]

Smooth is stable when \( c \in ]0, 1[ \). Formally:

\[
\forall i \in [a, b], \ c \in ]0, 1[ \rightarrow \text{smooth}(c) \ i \in [a, b]
\]

Let’s build a mechanized constructive proof.
What’s the plan?

1. Define the syntax of Faust inside Coq.
What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two: Interpretation.
What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two: Interpretation.
4. Define a logic to simplify reasoning.
What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two: Interpretation.
4. Define a logic to simplify reasoning.
5. Verify!
Mechanized Semantics for Streams

- Didn’t consider PACO, etc.
- Our wish: Sequences $\mathcal{S}$ of a base type $\mathbb{R}$ [Auger2013]
Mechanized Semantics for Streams

- Didn’t consider PACO, etc.
- Our wish: Sequences $S$ of a base type $R$ [Auger2013]

Soundness needs stronger semantics (also [Guatto2014]):

$$\left[ \left[ \begin{array}{c} i \\ o \end{array} \right] \rightarrow f \rightarrow R \right]^n : \left[ R \times \ldots \times R \right]^n \rightarrow \left[ R \times \ldots \times R \right]^n$$

Index by number of steps; equality of streams more intensional wrt to $(\mathbb{N} \rightarrow R)$. 
The Second Piece: Real Analysis

What about the base type $\mathbb{R}$?

- Reals not in Mathcomp – algebraic structures good enough for most of our experiments so far.
- There are lots of work to do here. We lack convenient complex numbers, exponentials, etc...
What about the base type $\mathbb{R}$?

- Reals not in Mathcomp – algebraic structures good enough for most of our experiments so far.
- There are lots of work to do here. We lack convenient complex numbers, exponentials, etc...
Proving Stability

We could do the proof directly in Coq; it is not difficult, but a bit cumbersome in general. What is worse, the same patterns with minor variations are repeated in each proof:

Not practical.
We could do the proof directly in Coq; it is not difficult, but a bit cumbersome in general. What is worse, the same patterns with minor variations are repeated in each proof:

Not practical.

To remedy this, we define a program logic for sample-level properties.
Sampled-Level Predicates

Definition (Sample-Level Property)

A property $P : S \rightarrow \mathbb{B}$ is sample-level if there exists a characteristic predicate $\varphi : R \rightarrow \mathbb{B}$ such that for all streams $s$:

$$P(s) \iff \forall n. \varphi(s[n])$$
Definition (Sample-Level Property)

A property $P : S \rightarrow \mathbb{B}$ is sample-level if there exists a characteristic predicate $\varphi : R \rightarrow \mathbb{B}$ such that for all streams $s$:

$$P(s) \iff \forall n. \varphi(s[n])$$

Boundedness $x \in [a, b]$ is a sample-level property!
Sampled-Level Predicates

**Definition (Sample-Level Property)**

A property $P : S \rightarrow \mathbb{B}$ is sample-level if there exists a characteristic predicate $\varphi : R \rightarrow \mathbb{B}$ such that for all streams $s$:

$$P(s) \iff \forall n. \varphi(s[n])$$

Boundedness $x \in [a, b]$ is a sample-level property! Properties can be made sample-level by self-composition, e.g: ratio:

$$f \Rightarrow \langle f, f' \rangle : /$$

We can also prove this way equivalence of filter implementation.
Definition (Sampled Judgment)
Given two characteristic predicates $\varphi, \psi$, we write

$$\{\varphi\} f \{\psi\}$$

“for all input $i$ meeting $\varphi$, the $i$ satisfies $\psi$.”

Example
The stability judgment for smooth is written as:

$$\{x \in [a, b]\} \text{ smooth } \{x \in [a, b]\}$$
Rules for The Sampled Logic

\[
\forall i_1, i_2, (\varphi_1(i_1) \land \varphi_1(i_2)) \implies \psi(i_1 + i_2) \quad \text{Prim}
\]

\[
\{\varphi_1, \varphi_2\} + \{\psi\}
\]

\[
\{\varphi\} f \{\theta\} \quad \{\theta\} g \{\psi\} \quad \text{Comp}
\]

\[
\{\varphi\} f : g \{\psi\}
\]

\[
\models \psi(x_0) \quad \{\theta, \varphi\} f \{\psi\} \quad \{\psi\} g \{\theta\} \quad \text{Feed}
\]

\[
\{\varphi\} f \sim g \{\psi\}
\]
Soundness of the Logic

Definition (Validity)

\[[\{\varphi\} f \{\psi\}] \equiv \forall i. (\forall t. \varphi(i(t))) \implies (\forall t, \psi([f])(i(t)))\]

Theorem (Soundness)

For any program \( f \) of type \( i \rightsquigarrow o \), if

\[\{\varphi_1, \ldots, \varphi_i\} f \{\psi_1, \ldots, \psi_o\}\]

is derivable then,

\[[\{\varphi_1, \ldots, \varphi_i\} f \{\psi_1, \ldots, \psi_o\}]\]

is valid.
Stability Proof for Smooth

\[
\begin{align*}
\{l_{ab}\} \ast (1 - c) \{l_{abc}\} & \implies \{l_{abc}, l_{abc}\} + \{l_{ab}\} \\
\{l_{abc}\} + \sim \ast (c) \{l_{ab}\} & \implies \{l_{ab}\} \ast (c) \{l_{abc}\}
\end{align*}
\]

\[\{i \in [a, b]\} \ast (1 - c) : + \sim \ast (c) \{o \in [a, b]\}\]

with:

\[
\begin{align*}
l_{ab}(x) & \equiv x \in [a, b] \\
l_{abc}(x) & \equiv x \in [a \ast c, b \ast c] \\
l_{abc}(x) & \equiv x \in [a \ast (1 - c), b \ast (1 - c)]
\end{align*}
\]
Stability of Smooth

Three main VC in the proof:

\[(1 - c) \cdot i \in [(1 - c) \cdot a, (1 - c) \cdot b] \]
by rewrite ?ler_wpmul2r ?ler_subr_addr ?add0r.

have Ha: a = a * c + a * (1 - c)
   by rewrite –mulrDr addrC addrNK mulr1.
have Hb: b = b * c + b * (1 - c)
   by rewrite –mulrDr addrC addrNK mulr1.
by rewrite Ha Hb !ler_add.

\[c \cdot i \in [c \cdot a, c \cdot b]\]
by rewrite ?ler_wpmul2r.

We pushed the VCs to Why3 with success.
Interval technique ready to go into the main compiler.
Stability Proof
One Step Beyond

Extending the logic
Allow predicates to refer to windows.

\[ \varphi(i) \equiv \{i/i_{\square} = 0.8\} \]

where \( i_{\square} \) is the sample produced in the execution step.
Consider the following subset of Faust:

\[ * (c) \] scaling by \( c \)
\[ + \] addition
\[ : \] composition
\[ \sim \] addition

Then every Faust program is LTI. Very related to [Bonchi et al. 2015]
A consequence of that is that every program can be viewed as a polynomial.
Two Poles IIR Filter

twopole = fir : + ~ feedback
where
  fir(x) = (x - x''') * g * (1-RR) / 2;
  feedback(v) = 2*R*cos(T) * v - RR * v';
  ....
Two Poles IIR Filter

twopole = fir : + ~ feedback

where

\[ \text{fir}(x) = (x - x''') \times g \times (1-RR) / 2; \]
\[ \text{feedback}(v) = 2R \times \cos(T) \times v - RR \times v'; \]

....

Get and verify its transfer function:

\[
H(z) = \frac{1 - z^{-2}}{1 - 2R \cos(\Theta_c)z^{-1} + R^2z^{-2}}
\]
Ongoing: Frequency Domain Analysis

Recall the Fourier Matrix:

\[
W = \frac{1}{\sqrt{N}} \begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\
1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)}
\end{bmatrix}
\]

or:

\[
W = \left( \frac{\omega^{jk}}{\sqrt{N}} \right)_{j,k=0,\ldots,(N-1)}
\]

where \( \omega \) the nth-root of the unity. Then the DFT can be expressed as:

\[
X = Wx
\]
Fourier Properties Formally

Linearity, shifting and scaling follow from lemmas already in the MathComp linear algebra library! Parseval’s theorem is work in progress:

\[
\sum_{n=0}^{N-1} |x_n|^2 = \sum_{n=0}^{N-1} |X_n|^2
\]
Transfer Functions

- We can use a similar approach for the certification of transfer functions.
- We use the finite $Z$-transform, plus some caveats, mainly about the bounds.
Transfer Functions

- We can use a similar approach for the certification of transfer functions.
- We use the finite $Z$-transform, plus some caveats, mainly about the bounds.
- C.f: Algebraic Signal Processing [Puesel, Moura]

Paper with our adventures coming end of month.
Conclusions

- It was an interesting exercise; we learned a lot!
- The full Faust language is basically done.
- So far verification has been about math verification.
- Floating point issues ignored. . .
- Help from audio people. What are important things to certify?
- Non-Linear systems.
- We are investigating a different approaches to certification beyond program logics.
- Verified FFT/DSP computation. Trying CoqEAL.
- Improving the language for spectral processing.
- Non-linear Wave Filter, Scattered Delays Networks.