Verifying Faust in Coq
Progress report

Emilio J. Gallego Arias, Pierre Jouvelot, Olivier Hermant, Arnaud Spiwack

MINES ParisTech, PSL Research University, France

CoqPL 2015
Music and PL?

Some Music DSLs

- 4CED
- Adagio
- AML
- AMLE
- Areti
- Arctic
- Autoklang
- Bang
- Canon
- CHANT
- Chuck
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- Csound
- CyberBand
- DARMS
- DCMP
- DMIX
- Elody
- EsaAC
- Esverse
- Esverse
- Faust
- Flavors Band
- Flusses
- FORL
- FORMAES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Hascore
- HMSL
- INV
- Invokator
- KERIN
- LPC
- Mars
- MASC
- Max
- MidiLisp
- MidiLogo
- MODE
- MOM
- Music
- MSX
- MUS10
- MUS8
- MUS5MP
- MuseData
- MuseES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 48
- MUSIC 4BF
- MUSIC 4F
- MCL
- MUSIC III/IV/V
- MusicLogo
- Music1000
- MUSIC7
- Musictax
- MUSIGOL
- MusicXML
- Musicxt
- NIFF
- NOTE LIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PIE
- PLE
- PLA
- PLACOND
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- PresElectra
- PWGL
- Ravel
- SALIERI
- SCORE
- ScareFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SSP
- SSSP
- ST
- Supercollider
- Symbolic
Music and PL?

Software verification?
**Music and PL?**

<table>
<thead>
<tr>
<th>Some Music DSLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARMS</td>
</tr>
<tr>
<td>DCMP</td>
</tr>
<tr>
<td>DMIX</td>
</tr>
<tr>
<td>Elnedy</td>
</tr>
<tr>
<td>EnAC</td>
</tr>
<tr>
<td>En Tempero</td>
</tr>
<tr>
<td>En Tempero</td>
</tr>
<tr>
<td>Fauze</td>
</tr>
<tr>
<td>Fugue</td>
</tr>
<tr>
<td>Gibber</td>
</tr>
<tr>
<td>GROOVE</td>
</tr>
<tr>
<td>GUIDO</td>
</tr>
<tr>
<td>HARP</td>
</tr>
<tr>
<td>Hascore</td>
</tr>
<tr>
<td>HMSL</td>
</tr>
<tr>
<td>INV</td>
</tr>
<tr>
<td>Invokator</td>
</tr>
<tr>
<td>KERN</td>
</tr>
<tr>
<td>MCL</td>
</tr>
<tr>
<td>MUSIC III/IV/V</td>
</tr>
<tr>
<td>MusicLogo</td>
</tr>
<tr>
<td>Music1000</td>
</tr>
<tr>
<td>MUSIC7</td>
</tr>
<tr>
<td>PLACONI</td>
</tr>
<tr>
<td>PLAY1</td>
</tr>
<tr>
<td>PLAY2</td>
</tr>
<tr>
<td>PMX</td>
</tr>
<tr>
<td>POLO</td>
</tr>
<tr>
<td>POD6</td>
</tr>
<tr>
<td>POD7</td>
</tr>
<tr>
<td>ROD4</td>
</tr>
<tr>
<td>ROD6</td>
</tr>
<tr>
<td>Ravel</td>
</tr>
<tr>
<td>SALIERI</td>
</tr>
<tr>
<td>SCORE</td>
</tr>
<tr>
<td>ScareFile</td>
</tr>
<tr>
<td>SCRIPT</td>
</tr>
<tr>
<td>SIREN</td>
</tr>
<tr>
<td>SMILE</td>
</tr>
<tr>
<td>SMDL</td>
</tr>
<tr>
<td>SMOKE</td>
</tr>
<tr>
<td>SSSP</td>
</tr>
<tr>
<td>SSSP</td>
</tr>
<tr>
<td>ST</td>
</tr>
<tr>
<td>Supercollider</td>
</tr>
<tr>
<td>Symbolic</td>
</tr>
</tbody>
</table>

**Software verification?**

**Coq?**
Faust

- Functional PL for digital signal processing.
- Synchronous paradigm, geared towards audio.
- Programs: circuits/block diagrams + feedbacks.
- 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,…
- 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,…
- 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,…
▶ Guitarix, moForte guitar, etc...

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

Recent Events:

▶ Faust day at Stanford happened yesterday.
▶ Ongoing Faust program competition (€2,000 in prices).
▶ FEEVER project :)
Syntax and Well-Formedness

**TERM**
\[ \vdash ! : 1 \rightarrow 0 \]

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

**PAR**
\[ \vdash f_1 : i_1 \rightarrow o_1 \quad \cdots \quad \vdash f_n : i_n \rightarrow o_n \]
\[ \vdash (f_1, \ldots, f_n) : \sum_j i_j \rightarrow \sum_j o_j \]

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

**PAN**
\[ \vdash f : i \rightarrow k \quad \vdash g : k \ast n \rightarrow o \quad 0 < k \land 0 < n \]
\[ \vdash f <: g : i \rightarrow o \]
Syntax and Typing

PL standard practice vs. what musicians want/imagine:

Figure 2: \((B : C)\) sequential composition of \(B\) and \(C\)

Figure 3: sequential composition of \(B\) and \(C\) when \(k = 1\)
Feedbacks

\[
\begin{align*}
\vdash f : g_o + f_i & \to g_i + f_o \quad \vdash g : g_i & \to g_o \\
\vdash f \sim g : f_i & \to f_o
\end{align*}
\]

Diagram for \( + \sim \sin \):

Synchronous semantics: execution in “ticks” + state.
Simple Low-pass Filter

\[
\text{smooth}(c) = *(1 - c) : + *(c); \\
\text{process} = \text{smooth}(0.9);
\]
A More Real Example

Feedback Delay Networks:

\[
\text{fdnrev0}(\text{delays}, \text{BBSO}, \text{freqs}, \text{durs}, \text{loopgainmax}, \text{nonl})
\]
\[
= (\text{bus}(2N) :> \text{bus}(N) : \text{delaylines}(N))
\]
\[
(\text{delayfilters}(N, \text{freqs}, \text{durs}) : \text{feedbackmatrix}(N))
\]
with \{
\[
\text{delayval}(i) = \text{take}(i+1, \text{delays});
\]
\[
\text{delaylines}(N) = \text{par}(i,N,(\text{delay}(\text{dlmax}(i),(\text{delayval}(i)-1))));
\]
\[
\text{delayfilters}(N, \text{freqs}, \text{durs}) = \text{par}(i,N,\text{filter}(i, \text{freqs}, \text{durs}));
\]
\[
\text{feedbackmatrix}(N) = \text{bhadamard}(N);
\]
\[
\text{vbutterfly}(n) = \text{bus}(n) <: (\text{bus}(n):>\text{bus}(n/2)), ...
\]
\[
...\]
\[
};
\]
A More Real Example
Why Coq?
Why Coq?

Does there exist any other programming language?
Why Coq? Motivations and Goals

**Philosophical**

- Manual proofs starting to feel odd in PL.
- Motto: use Coq from the start.
- Goal: Try to develop in a reusable way.
Why Coq? Motivations and Goals

**PHILOSOPHICAL — MATHEMATICAL**

Current testing process of Faust programs: compare their output with MatLab’s.
Why Coq? Motivations and Goals

**PHILOSOPHICAL — MATHEMATICAL**

Current testing process of Faust programs: compare their output with MatLab’s.

- Program correctness.
Why Coq? Motivations and Goals

Philosophical — Mathematical

Current testing process of Faust programs: compare their output with MatLab’s.

- Program correctness.
- Optimizations performed by the compiler are not well understood. *Semantics trickier than it looks to the eye*
Current testing process of Faust programs: compare their output with MatLab’s.

- Program correctness.
- Optimizations performed by the compiler are not well understood. *Semantics trickier than it looks to the eye*
- Explore the formalization of concepts from signal processing: Finite Impulse Response (FIR) filters, LTI theory, spectral analysis, Nyquist...
Why Coq? Motivations and Goals

**PHILOSOPHICAL — MATHEMATICAL PRACTICAL**

Less effort than to build a custom analysis tool.

Applications:

- HTML 5 Web Audio API QUICKSTART
- MODULUS 002
Why Coq? Motivations and Goals

**PHILOSOPHICAL — MATHEMATICAL PRACTICAL**

Less effort than to build a custom analysis tool.

Applications:

IMHO: **Robust Definitions and Standards** are crucial.

*Don’t repeat the mistakes of the past*
Some Properties of Interest

- Stability properties: bound input produces bounded output. (We’ll see an example)
Some Properties of Interest

- Stability properties: bound input produces bounded output. (We’ll see an example)
- Linearity/Time invariance. [relational!]
- Stabilization: Zero input eventually produces zero output.
- Frequency response properties.
Some Properties of Interest

- Stability properties: bound input produces bounded output. (We’ll see an example)
- Linearity/Time invariance. [relational!]
- Stabilization: Zero input eventually produces zero output.
- Frequency response properties.

In order to write the properties, we need a large support library
[bigops, intervals, trigonometry, Z-transforms, DTFT, …]
Specifications of Filters

Difference equations:

\[ y(n) = x(n) + x(n - 1) \]

Impulse response:

\[ H(z) = \frac{1 - z^{-2}}{1 - 2R \cos(\Theta_c)z^{-1} + R^2z^{-2}} \]
Two Poles Filter

\[ H(z) = \frac{1 - z^{-2}}{1 - 2R \cos(\Theta_c) z^{-1} + R^2 z^{-2}} \]

process = firpart :+ feedback
with {
  bw = 100; fr = 1000; g = 1; // parameters — see caption
  SR = fconstant(int fSamplingFreq, <math.h>); // Faust fn
  pi = 4*atan(1.0); // circumference over diameter
  R = exp(0–pi*bw/SR); // pole radius [0 required]
  A = 2*pi*fr/SR; // pole angle (radians)
  RR = R*R;
  firpart(x) = (x – x’’) * g * ((1–RR)/2);
  feedback(v) = 0 + 2*R*cos(A)*v – RR*v’;
};
Finally! Let’s Talk About Coq!

What we have built so far:

- Mathcomp allowed us to do a prototype in two weeks.
- New feedback reasoning rule & proof of soundness.
- *Motivated by real use cases.*
- Stateless logic & soundness (again, mc was key).
- Certified arity-checker, etc...
Finally! Let’s Talk About Coq!

What we have built so far:

- Mathcomp allowed us to do a prototype in two weeks.
- New feedback reasoning rule & proof of soundness.
- *Motivated by real use cases.*
- Stateless logic & soundness (again, mc was key).
- Certified arity-checker, etc. . .

Currently:

- Investigating more complex, time-aware logics.
- New semantics based on guarded recursion.
The Pieces of the Puzzle
The First Piece: Stream-based Semantics

- We ported [Boulmé, Hamon and Pouzet], with some problems with CoInductives.
- We switched to sequences, (similar to Auger’s Lustre certified compiler).
- Didn’t look into PACO/more advanced co-reasoning tools.
The First Piece: Stream-based Semantics

- We ported [Boulmé, Hamon and Pouzet], with some problems with CoInductives.
- We switched to sequences, (similar to Auger’s Lustre certified compiler).
- Didn’t look into PACO/more advanced co-reasoning tools.

Current approach: realizability semantics in guarded recursion style. Suggested simultaneously by A. Spiwak and A. Guatto [SYNCHRON 2014]:

\[ [\vdash f : i \to o]_W^n : [i]^n \to [o]^n \]
The Second Piece: Real Analysis

- Not in Mathcomp – rcf good enough for experiments.
- Our typical case involves complex numbers, trigonometry and sums over infinite series.
- Think of proving Euler’s formula:

\[ e^{(j\Theta)} = \sin \Theta + j \cos \Theta \]

- Difficult to choose: Standard library? Coquelicot? C-CorN?
- Our feeling is that life is going to be very painful.

[We are ignoring floating point issues for now]
The Third Piece: Coq as a Tool

```coq
(* Step by step proof. *)
equiv vcg_step1: VCGStep.vcg_full ~ VCGStep.vcg_full_s1 & true ==>= ={res}.
proof.
  proc; inline *.
  swap 5 -> 4.
  seq 1 1: ={i1}; first by auto.
  case (i1).
    + seq 1 1: i1; first by auto.
      seq 1 1: i1; first by auto.
      seq 1 1: i1; first by auto.
      seq 1 1: i1; first by auto.
      by wp; sklp; progress; rewrite H.
    + swap(1) 1 1.
      seq 1 1: i1; first by auto.
      seq 1 1: i1; first by auto.
      seq 1 1: i1; first by auto.
      seq 1 1: i1; first by auto.
      by wp; skip; progress; rewrite H.
  qed.

equiv vcg_step2: VCGStep.vcg_full_s1 ~ VCGStep.vcg_full_s2 & true ==>= ={res}.
proof.
  proc; inline *.
  swap(1) 5 2.
  seq 1 0: true; first by auto; progress; apply/rmu full.
```

```
Current goal (remaining: 2)
Type variables: <none>

&1 (left) : VCGStep.vcg_full
&2 (right) : VCGStep.vcg_full_s1
pre = =i1 \ Q i{1}

\begin{align*}
t &\vdash \text{rmu} \\
r &\vdash \text{rmu} \\
s1 &\vdash \text{rmu} \\
s2 &\vdash \text{rmu}
\end{align*}

insBr = i ? (t, r) : (r, t)
surr = fst (vcg insBr (s1, s2) wt)

post =
\begin{align*}
&\text{if } i1 \text{ then } \text{fst surrs}\{1\} \text{ else snd surrs}\{1\} = \\
&\text{if } i2 \text{ then } \text{fst surrs}\{2\} \text{ else \text{snd surrs}}\{2\}
\end{align*}
```

Copyright (c) - 2012-2014 - IMDEA Software Institute and

Distributed under the terms of the CeCILL-C license
The Third Piece: Coq as a Tool

▶ Is building a verification tool on top of Coq feasible?
▶ We got some inspiration from domain-specific tools like EasyCrypt.
▶ Would our tool mature, we would certainly need to plug deeply into Coq’s parsing/display routines.
▶ We still think this may be better than rewriting everything from scratch.
▶ Reduction woes make our life difficult.
▶ Automation: we will worry last.
Stability of Smooth

Recall the smooth program:

\[ \text{smooth}(c) = * (1 - c) : + * (c); \]

We want to prove stability, that is, bounded inputs produce bounded outputs, provided the coefficient \( c \) is in \([0, 1]\).

We use the following logical rule (simplified):

\[
\begin{align*}
\vdash & \psi(x_0) \\
\{\gamma(i_1) \land \phi(i_2)\} & f \{\psi(o)\} \quad \{\psi(i)\} \quad g \{\gamma(o)\} \\
\{\phi(i)\} & f \sim g \{\psi(o)\}
\end{align*}
\]
Stability of Smooth

Three VC in the proof:

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.

by rewrite ?ler_wpmul2r.

We pushed the VCs to Why3 with success.
Interval technique ready to go into the main compiler.
Conclusions

► Young project, highly positive experience so far.
► First alpha “release” very near.
► Tons of related work, difficult to get a good perspective.
► Most challenging topic: real/complex analysis.
► Certified audio/dsp processing? (Do we need it?)
► All of the usual Coq caveats apply to us.
► What do *you* think?

Thanks!
Nyquist Theorem

Provided $f_s$ is twice the highest frequency in $V$ then:

$$V(t) = \sum_{n=-\infty}^{\infty} V[n] \frac{\sin[\pi f_s(t - nT_s)]}{\pi f_s(t - nT_s)}$$

where

- $f_s = 1/T_s$ sampling frequency
- $V(t)$ value of signal at Time $t$
- $V[n] = V(nT_s)$ value of signal at Time $t = nT_s$