

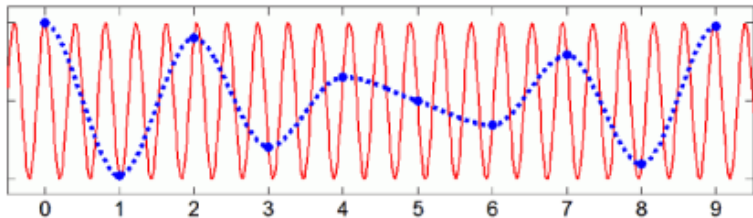
# Verifying Faust in Coq

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
  - AMPLE
  - Arctic
  - Autaklang
  - Bang
  - Canon
  - CHANT
  - Chuck
  - CLCE
  - CMIX
  - Cmusic
  - CMUSIC
  - Common Lisp Music
  - Common Music
  - Common Music Notation
  - Coound
  - CyberBand
  - DARMES
  - DCOMP
  - DMIX
  - Eady
  - ExAC
  - Euterpea
  - Extempore
  - Faust
  - Flavors Band
  - Fluxus
  - FOEL
  - FORMES
  - FORMULA
  - Fugue
  - Gliber
  - GROOVE
  - GUIDO
  - HARP
  - Hashore
  - HMSL
  - INV
  - invokator
  - KERN
  - LPC
  - Mars
  - MASC
  - Max
  - MidLisp
  - MidLogo
  - MODE
  - MOM
  - Moxc
  - MSX
  - MUS10
  - MUS8
  - MUSCOMP
  - MusicData
  - MusicES
  - MUSIC 10
  - MUSIC 11
  - MUSIC 360
  - MUSIC 48
  - MUSIC 48F
  - MUSIC 4F
  - MCL
  - MUSIC III/IV/V
  - MusicLogo
  - Music1000
  - MUSIC7
  - Musictex
  - MUSIGOL
  - MusicXML
  - Musixtex
  - NIFF
  - NOTELIST
  - Nyquist
  - OPAL
  - OpenMusic
  - OrganumI
  - Outperform
  - Overtone
  - PE
  - Patchwork
  - PILE
  - PLx
  - PLACOLM
  - PLAY1
  - PLAY2
  - PMX
  - POCO
  - POD6
  - POD7
  - PROD
  - Puredata
  - PWGL
  - Ravel
  - SALIERI
  - SCORE
  - ScoreFile
  - SCRIPT
  - SIREN
  - SMDL
  - SMOKE
  - SSP
  - SSSP
  - ST
  - Supercollider
  - Symbolic

# Music and PL?

Some Music DSLs



• 4CED	• DARMES	• LPC	• MCL	• PLACOD
• Adagio	• DCOMP	• Mars	• MUSIC III/IV/V	• PLAY1
• AML	• DMIX	• MASC	• MusicLogo	• PLAY2
• AMPLE	• Eady	• Max	• Music1000	• PMX
• Arctic	• ExAC	• MidLip	• MUSIC7	• POCO
• Autaklang	• Euterpea	• MidLogo		• POD6
• Bang	• Extempore			• POD7
• Canon	• Faust			• ROD
• CHANT				• rdata
• Chuck	• FOL	• MSX	• MusicXML	• WGL
• CLCE	• FORMES	• MUS10	• Musixtex	• Ravel
• CMIX	• FORMULA	• MUS8	• NIFF	• SALIERI
• Cmusic	• Fugue	• MUSCOMP	• NOTELIST	• SCORE
• CMUSIC	• Gilder	• MusData	• Nyquist	• ScoreFile
• Common Lisp Music	• GROOVE	• MusES	• OPAL	• SCRIPT
• Common Music	• GUIDO	• MUSIC 10	• OpenMusic	• SIREN
• Common Music Notation	• HARP	• MUSIC 11	• OrganumI	• SMDL
• Coound	• Hashore	• MUSIC 360	• Outperform	• SMOKE
• CyberBand	• HMSL	• MUSIC 48	• Overtone	• SSP
	• INV	• MUSIC 4BF	• PE	• S5SP
	• invokator	• MUSIC 4F	• Patchwork	• ST
	• KERN		• PILE	• Supercollider
			• PLx	• Symbolic

Software verification?

# Music and PL?

Some Music DSLs

* 4CED	* DARMES	* LPC	* PLACOL
* Adagio	* DCOMP	* Mars	* PLAY1
* AML	* DMIX	* MASC	* PLAY2
* AMPLE	* Eady	* Max	* PMX
* Arctic	* ExAC	* MidLip	* POCO
* Autaklang	* Euterpea	* MidLogo	* POD6
* Bang	* Extempore	* MUSIC7	* POD7
* Canon	* Faust		* ROD
* CHANT			* rdata
* Chuck	* FOL		* WGL
* CLCE	* FORMES	* Sixtax	* Ravel
* CMIX	* FORMULA	* F	* SALIERI
* Cmusic	* Fugue	* TELIST	* SCORE
* CMUSIC	* Gilder	* Arquist	* ScoreFile
* Common Lisp Music	* GROOVE	* OPAL	* SCRIPT
* Common Music	* GUIDO	* OpenMusic	* SIREN
* Common Music Notation	* HARP	* OrganumI	* SMDL
* Coaud	* Hashore	* Outperform	* SMOKE
	* HMSL	* Overtone	* SSP
	* INV	* PE	* SSSP
	* invokator	* Patchwork	* ST
	* KERN	* PILE	* Supercollider
		* PL	* Symbolic

# 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 prizes).
- ▶ FEEVER project :)



# Syntax and Well-Formedness

$$\text{TERM} \frac{}{\vdash ! : 1 \rightarrow 0} \quad \text{ID} \frac{}{\vdash \_ : 1 \rightarrow 1}$$

$$\text{PAR} \frac{\vdash f_1 : i_1 \rightarrow o_1 \quad \dots \quad \vdash f_n : i_n \rightarrow o_n}{\vdash (f_1, \dots, f_n) : \sum_j^n i_j \rightarrow \sum_j^n o_j}$$

$$\text{COMP} \frac{\vdash f : i \rightarrow k \quad \vdash g : k \rightarrow o}{\vdash (f : g) : i \rightarrow o}$$

$$\text{PAN} \frac{\vdash f : i \rightarrow k \quad \vdash g : k * n \rightarrow o \quad 0 < k \wedge 0 < n}{\vdash f < : g : i \rightarrow o}$$

# Syntax and Typing

PL standard practice vs. what musicians want/imagine:

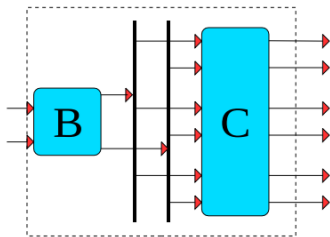


Figure 2:  $(B:C)$  sequential composition of  $B$  and  $C$

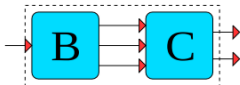
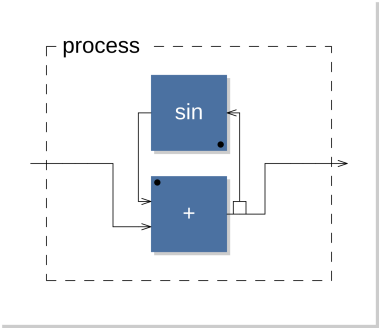


Figure 3: sequential composition of  $B$  and  $C$  when  $k = 1$

# Feedbacks

$$\text{FEED} \frac{\vdash f : g_o + f_i \rightarrow g_i + f_o \quad \vdash g : g_i \rightarrow g_o}{\vdash f \sim g : f_i \rightarrow f_o}$$

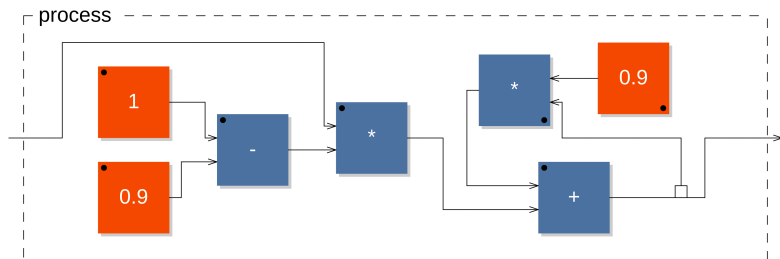
Diagram for  $+ \sim \sin$ :



Synchronous semantics: execution in “ticks” + state.

# Simple Low-pass Filter

```
smooth(c) = *(1-c) : + *(c);  
process = smooth(0.9);
```



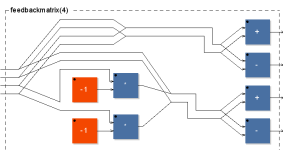
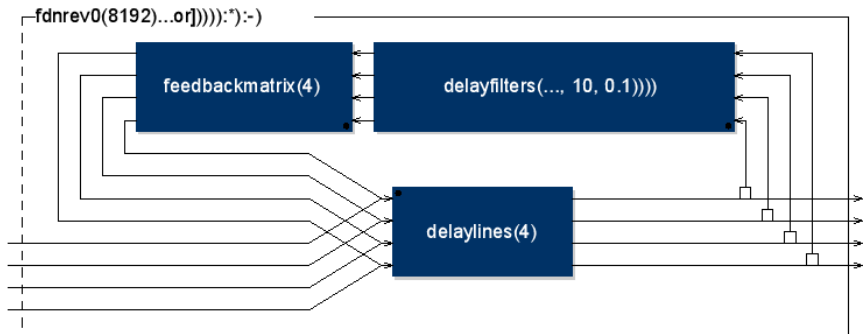
T:	1	2	3	4	5	6	7	8
I:	1.00	1.05	1.10	1.15	1.20	1.25	1.20	1.25
O:	0.10	0.19	0.28	0.37	0.45	0.53	0.61	0.68

# A More Real Example

## Feedback Delay Networks:

```
fdnrev0(delays, BBS0, freqs, durs, loopgainmax, nonl)
  = (bus(2*N) :> bus(N) : delaylines(N))
    (delayfilters(N,freqs,durs) : feedbackmatrix(N))
with {
  delayval(i) = take(i+1,delays);
  delaylines(N) = par(i,N,(delay(dlmax(i),(delayval(i)-1))));
  delayfilters(N,freqs,durs) = par(i,N,filter(i,freqs,durs));
  feedbackmatrix(N) = bhadamard(N);
  vbutterfly(n) = bus(n) <: (bus(n):>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*

# 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*
- ▶ 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:



# 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:

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

## 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]:

$$\llbracket \vdash f : i \rightarrow o \rrbracket_W^n : \llbracket i \rrbracket^n \rightarrow \llbracket o \rrbracket^n$$

# 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

```
File Edit Options Buffers Tools EasyCrypt Proof-General Help
var r : int;
r = $rmu;
return r;
}
}).

(* Step by step proof. *)
equiv vcg_step1: VCGStep.vcg_full ~ VCGStep.vcg_full_s1
s1 : true ==> =(res).
proof.
proc; inline *.
swap 5 -4.
seq 1 1 : (={i}); first by auto.
case (i{1}).
+ seq 1 1 : (i{1} /\ ={i,t}); first by auto.
seq 1 1 : (i{1} /\ ={i,t,r}); first by auto.
seq 1 1 : (i{1} /\ ={i,t,r,s1}); first by auto.
seq 1 1 : (i{1} /\ ={i,t,r,s1,s2}); first by auto.
by wp; skip; progress; rewrite H.
+ swap{1} 1 1.
seq 1 1 : (! i{1} /\ ={i} /\ r{1} = t{2}); first by
auto.
seq 1 1 : (! i{1} /\ ={i} /\ r{1} = t{2} /\ t{1} = r
{2}); first by auto.
seq 1 1 : (! i{1} /\ ={i,s1} /\ r{1} = t{2} /\ t{1}
= r{2}); first by auto.
seq 1 1 : (! i{1} /\ ={i,s1,s2} /\ r{1} = t{2} /\ t
{1} = r{2}); 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 = ={i} /\ i{1}

t =$ rmu (1) t =$ rmu
r =$ rmu (2) r =$ rmu
s1 =$ rmu (3) s1 =$ rmu
s2 =$ rmu (4) s2 =$ rmu
insBr = i ? (t, r) : (r, t) (5) insBr = (t, r)
surrs = fst (vcg insBr (s1, s2) wt) (6) surrs = fst (vcg

post =
(if i{1} then fst surrs{1} else snd surrs{1}) =
if i{2} then fst surrs{2} else snd surrs{2}

U:%%- *goals* All (1,0) (EasyCrypt goals)
>> Copyright (c) - 2012-2014 - IMDEA Software Institute an
>> 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:

```
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):

$$\frac{\begin{array}{c} \models \psi(x_0) \\ \{\gamma(i_1) \wedge \phi(i_2)\} \quad f \quad \{\psi(o)\} \quad \{\psi(i)\} \quad g \quad \{\gamma(o)\} \end{array}}{\{\phi(i)\} \quad f \sim g \quad \{\psi(o)\}}$$

# 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 $t = nT_s$