Verifying Faust in Coq

Progress report

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

Mines-ParisTech, PSL Research University

CoqPL 2015



Music and PL?

Some Music	DSLs • DARMS			
 4CED Adagio Adal, AMPLE Arctic Arctic Arctic Arctic Arctic Arctic Arctic Canon CHANT Chack CLCE CMIX Chack CLCE CMIX Consolic CAMISIC Common Liqu Music Common Music Common Music 	 DCMP DMOX Elady Eatorpea Extempose Fases Fases FORMULA FORMULA FORMULA GROOVE GUIDO HARP HARP HARSI IMV 	 LPC Mars Masc Mat MideLogs MODE MOM MOM Most MUS10 MUS10 MUS50 MUS50 MUSCMD2ta MoseES MUSIC 10 MUSIC 10 MUSIC 11 MUSIC 160 	 MCL MUSIC III/IV/V MusicLago MusicLood MUSIC7 Musictex MUSIGOL Musictex MUSIGOL Musictex MIFF NOTELIST Nyquist OPAL OpenMusic OrganumI Outgetform Overtane FE 	 PLACODI PLAY3 PLAY3 PMX POCO POD5 POD7 PROD PROD Prodata PWGL Ravel SALIERI SCORE SCORE SCORFIE SCORFIE SCRIPT SIMDL SMOL SSP SSP ST
E Cound	 invokator KERN 	4BF # MUSIC 4F	# PILE	Supercollider Symbolic

Music and PL?

Some Music	DSLs			
* 4CED * Adogio * AML * AMPLE * Arctic	OARMAS DOMP DOMP DOMP DOMP DOMP DOMY Electy Extempes Extempes	* LPC * Mars * MASC * Mas * MidLinp	 MCL MUSIC III/IV/V MusicLago Music1000 	 PLACOM PLAY1 PLAY2 PMX POCO POD6
Autoklang Bang Canon CMAMT	Softw	vare veri	ification	POD7
Chuck CLCE CMIX CMIX Comunic CMISS	* FOIL * FORMES * FORMULA * Fugue * Gibber	 MSX MUS10 MUS8 MUSCMP MusenDate 	Musicker Musicker NIFF NOTELIST Nyquist OPAL	Ravel SALIERI SCORE ScoreFile SCRIPT
Common Lisp Music Consman Music	* GROOVE * GUIDO * HARP * Haskore	 MusES MUSIC 10 MUSIC 11 MUSIC 360 	OpenMusic Organum1 Outperform Overtone	 SIREN SMDL SMOKE SSP
Common Music Notation CoburRoad	* INV * INV * invokutor * KERN	MUSIC 48 MUSIC 48F MUSIC 4F	 PE Patchwork PHE PLE PLs 	 sSSP ST Supercollider Symbolic

Music and PL?

Some Music	DSLs . DARMS			
 # 4CED # Adagio # AML # AMPLE # Arctic # Asstaklang 	 BCMP DMIX Body ExAC Extempts Extempts Extempts Fass 	# LPC # Mars # MASC # Max # MidLinp # MidLogs	 MCL MUSIC III/IV/V MusicLago Music1000 MUSIC7 	 PLACOMP PLAY1 PLAY2 PMX POCO PODS POD5 POD7
# Bang # Canon # CHANT	Softw	vare veri	fication	P ob redata
# Chuck # CLCE # CMIX	* FORMES * FORMULA	Coq	ELIST	RavelSALIERISCORE
Cmusic CMUSIC Common	Ghber GROOVE GUIDO	# MuseData # MusES	 reyquist OPAL OpenMusic 	 ScoreFile SCRIPT SIREN
Lisp Music Common Music	e HARP e Haskore	MUSIC 10 MUSIC 11 MUSIC 140	# Organum1 # Outperform # Overtone	 SMDL SMOKE SSP
Common Munic Notation Coound	a media. a INV a invokator	MUSIC 48 MUSIC 48F ABF	# PE # Patchwork # PILE	# SSSP # ST # Supercollider
+ CuberBand	* PLEADN	a model ap	# Pla	 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 (2000€ in prices).
- FEEVER project :)

Syntax and Well-Formedness

$$\begin{array}{ccc} \text{TERM} & \overline{\vdash !: 1 \rightarrow 0} & \text{ID} & \overline{\vdash _: 1 \rightarrow 1} \\ \\ \text{PAR} & \frac{\vdash f_1 : i_1 \rightarrow o_1 & \cdots & \vdash f_n : i_n \rightarrow o_n}{\vdash (f_1, \dots, f_n) : \sum_j^n i_j \rightarrow \sum_j^n o_j} \\ \\ \text{COMP} & \frac{\vdash f : i \rightarrow k & \vdash g : k \rightarrow o}{\vdash (f : g) : i \rightarrow o} \\ \\ \text{PAN} & \frac{\vdash f : i \rightarrow k & \vdash g : k \ast n \rightarrow o & 0 < k \land 0 < n}{\vdash f <: g : i \rightarrow o} \end{array}$$

Syntax and Typing

PL standard practice vs what the 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

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

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
1:	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:

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:









Does there exist any other programming language?

PHILOSOPHICAL

- Manual proofs starting to feel odd in PL.
- Motto: use Coq from the start.
- Try to develop in reusable way: both for the Faust/DSP and Coq communities.

PHILOSOPHICAL — MATHEMATICAL

 Prove programs correct, reason about them in new ways. Current testing process is to compare output with MatLab's.

PHILOSOPHICAL — MATHEMATICAL

- Prove programs correct, reason about them in new ways. Current testing process is to compare output with MatLab's.
- Optimizations performed by the compiler are not well understood. Semantics trickier than it looks to the eye

PHILOSOPHICAL — MATHEMATICAL

- Prove programs correct, reason about them in new ways. Current testing process is to compare output with MatLab's.
- Optimizations performed by the compiler are not well understood. Semantics trickier than it looks to the eye
- Explore the formalization of concepts from the signal processing community: Finite Impulse Response (FIR) filters, LTI theory, spectral analysis, Nyquist...

PHILOSOPHICAL — MATHEMATICAL PRACTICAL

Less effort than to build a custom analysis tool.

Applications:





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

 Stability properties: bound input produces bounded output.

Some Properties

 Stability properties: bound input produces bounded output. This will be our example.

Some Properties

- Stability properties: bound input produces bounded output. This will be our example.
- Linearity/Time invariance. [Note: relational!]
- Stabilization: Zero input eventually produces zero output.

Relating Programs:

Impulse response (two poles filter):

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

Relating Programs:

Impulse response (two poles filter):

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

Finally! Let's Talk About Coq!

So far:

- Mathcomp library allowed us to do a prototype in two weeks.
- New feedback reasoning rule: proved sound.
- Motivated by real use cases.
- Defined a one-state logic, proved it sound.
- Again, mathcomp was key.

Finally! Let's Talk About Coq!

So far:

- Mathcomp library allowed us to do a prototype in two weeks.
- New feedback reasoning rule: proved sound.
- Motivated by real use cases.
- Defined a one-state logic, proved it sound.
- Again, mathcomp was key.

Currently:

- Investigating more complex logics.
- New semantics needed, based on guarded recursion.

The Pieces of the Puzzle



The First Piece: Streams

- We ported [Boulmé, Hamon and Pouzet], some problems with Colnductives.
- Like in C. Auger Lustre certified compiler, we choose to work with sequences (for now).
- Didn't look into PACO and more advanced co-reasoning tools.

The First Piece: Streams

- We ported [Boulmé, Hamon and Pouzet], some problems with Colnductives.
- Like in C. Auger Lustre certified compiler, we choose to work with sequences (for now).
- Didn't look into PACO and more advanced co-reasoning tools.

The current solution: a realizability semantics in guarded recursion style. Suggested simultaneously by A. Spiwak and A. Guatto:

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

The Second Piece: Analysis

- Not in Mathcomp. rcfType good enough for now.
- How hard is to prove Euler's identity:

todo

- Difficult to chose. C-CorN? The standard library? Coquelicot?
- Our feeling is that given the amount of analysis going on our life is going to be very painful.

[We ignore precision issues and machine floats for now]

The Third Piece: Coq as a Tool

- Is building a verification tool on top of Coq feasible? Does it even make sense?
- We got some inspiration from 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.
- Our approach to automation: last thing to worry about.

The Third Piece: Coq as a Tool

```
ile Edit Options Buffers Tools EasyCrypt Proof-General Help
                                                             Current goal (remaining: 2)
                                                             Type variables: <none>
                                                             &1 (left ) : VCGStep.vcg full
                                                             &2 (right) : VCGStep.vcg full s1
 equiv vcg step1: VCGStep.vcg full ~ VCGStep.vcg full s1
  true ==> =\{res\}.
                                                             pre = ={i} /\ i{1}
                                                             t =$ rmu
                                                                                                   (1) t =$ rmu
                                                             r =$ rmu
                                                                                                   (2) r =$ rmu
                                                             s1 =$ rmu
                                                                                                   (3) s1 =$ rmu
                                                             s2 =$ rmu
                                                                                                   (4) s2 =$ rmu
  + seq 1 1 : (i{1} /\ ={i,t}); first by auto.
                                                             insBr = i ? (t, r) : (r, t)
                                                                                                  (5) insBr = (t, r)
    seq 1 1 : (i{1} /\ ={i,t,r}); first by auto.
                                                             surrs = fst (vcg insBr (s1, s2) wt) (6) surrs = fst (vcg
    seg 1 1 : (i{1} /\ ={i.t.r.sl}): first by auto.
    seq 1 1 : (i{1} /\ ={i,t,r,s1,s2}); first by auto.
                                                             post =
     by wp; skip; progress; rewrite H.
                                                               (if i{1} then fst surrs{1} else snd surrs{1}) =
                                                               if i{2} then fst surrs{2} else snd surrs{2}
 + 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} = re
42}); first by auto.
     seq 1 1 : (! i{1} /\ ={i,s1} /\ r{1} = t{2} /\ t{1}
s= 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.
aed.
equiv vcg step2: VCGStep.vcg full s1 ~ VCGStep.vcg full •
                                                             U:%%- *goals*
                                                                                                  (EasyCrypt goals)
                                                              > Copyright (c) - 2012-2014 - IMDEA Software Institute a
s2 : true ==> ={res}.
                                                              >> Distributed under the terms of the CeCILL-C license
   swap{1} 5 2.
```

Verification of the Smooth Filter:

Recall the smooth filter.

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]. Three significant cases:

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 VC to Why3 with success. Technique ready for incorporation into the main compiler.

Conclusions:

- Young project, highly positive 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!