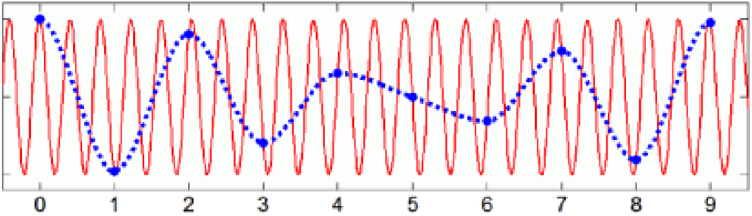
Reasoning About Sound Programs

Emilio Jesús Gallego Arias

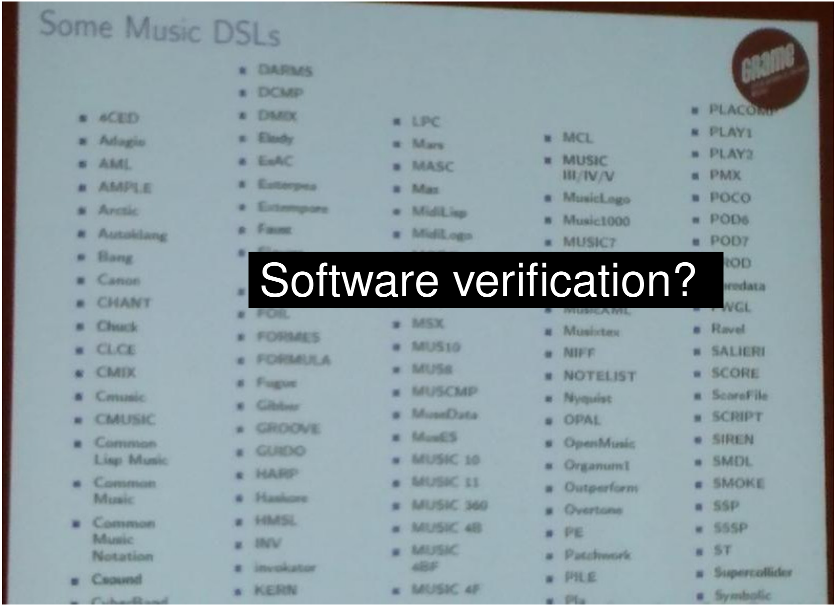
Joint work with O. Hermant & P. Jouvelot
MINES ParisTech, PSL Research University, France

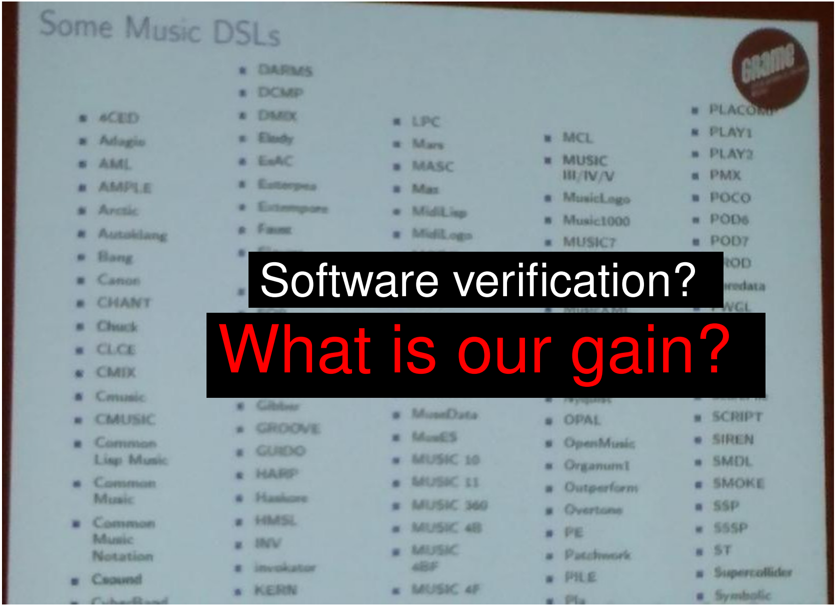
Rennes, 15 Avril 2015



Some Music DSLs

- 4CED
- Adagio
- AML
- AMPLE
- Arctic
- Autoklang
- Bang
- Canon
- CHANT
- Chuck
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- Csound
- CyberBand
- DARMS
- DCMP
- DMIX
- Elody
- EsAC
- Euterpea
- Extempore
- Faust
- Flavors Band
- Fluxus
- FOIL
- FORMES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Haskore
- HMSL
- INV
- invokator
- KERN
- LPC
- Mars
- MASC
- Max
- MidiLisp
- MidiLogo
- MODE
- MOM
- Moxc
- MSX
- MUS10
- MUS8
- MUSCMP
- MuseData
- MusES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 4B
- MUSIC 4BF
- MUSIC 4F
- MCL
- MUSIC III/IV/V
- MusicLogo
- Music1000
- MUSIC7
- Musictex
- MUSIGOL
- MusicXML
- Musixtex
- NIFF
- NOTELIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PILE
- Pla
- PLACOMP
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- Puredata
- PWGL
- Ravel
- SALIERI
- SCORE
- ScoreFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SSP
- SSSP
- ST
- Supercollider
- Symbolic


Some Music DSLs
4CED
Adagio
AML
AMPLE
Arctic
Autoklang
Bang
Canon
CHANT
Chuck
CLCE
CMIX
Cmusic
CMUSIC
Common Lisp Music
Common Music
Common Music Notation
Csound
DARMS
DCMP
DMIX
EsAC
Faust
FORMES
FORMULA
Fugue
Gibber
GROOVE
GUIDO
HARP
Haskore
HMSL
INV
invokator
KERN
LPC
Mars
MASC
Max
MidiLisp
MidiLogo
MSX
MUS10
MUS8
MUSCMP
MuseData
MusES
MUSIC 10
MUSIC 11
MUSIC 360
MUSIC 4B
MUSIC 4BF
MUSIC 4F
MCL
MUSIC III/IV/V
MusicLogo
Music1000
MUSIC7
Musixtex
NIFF
NOTELIST
Nyquist
OPAL
OpenMusic
Organum1
Outperform
Overtone
PE
Patchwork
PILE
PLAY1
PLAY2
PMX
POCO
POD6
POD7
Ravel
SALIERI
SCORE
ScoreFile
SCRIPT
SIREN
SMDL
SMOKE
SSP
SSSP
ST
Supercollider
Symbolic
Software verification?

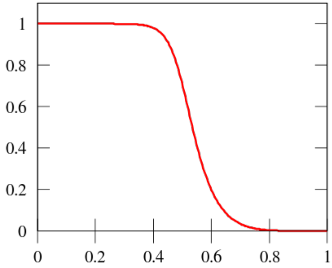


Software verification?

What is our gain?

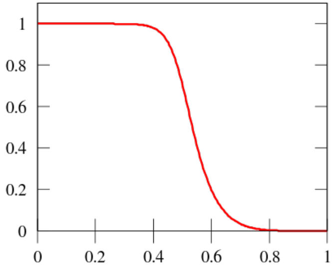
Let's assume a simple IIR filter:

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



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.

<u>Standard DSP theory gives answers.</u>

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

Natural questions are:

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

Standard DSP theory gives answers.

What about the implementation of the filter?

We dive into the realm of PL theory!

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

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

$$\textsc{Term}\ \frac{}{\vdash ! : 1 \to 0} \qquad \textsc{Id}\ \frac{}{\vdash \_ : 1 \to 1}$$

$$\textsc{Par}\ \frac{\vdash f_1 : i_1 \to o_1 \qquad \cdots \qquad \vdash f_n : i_n \to o_n}{\vdash (f_1, \ldots, f_n) : \sum_j^n i_j \to \sum_j^n o_j}$$

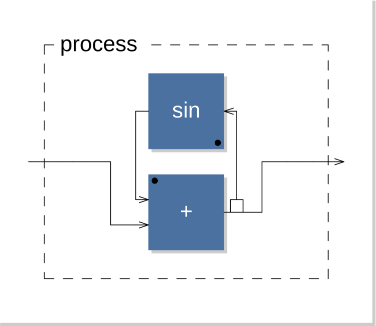
$$\textsc{Comp}\ \frac{\vdash f : i \to k \qquad \vdash g : k \to o}{\vdash (f : g) : i \to o}$$

$$\textsc{Pan}\ \frac{\vdash f : i \to k \qquad \vdash g : k * n \to o \qquad 0 < k \wedge 0 < n}{\vdash f <: g : i \to o}$$

Feedback

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

Diagram for `+ ~ sin`:

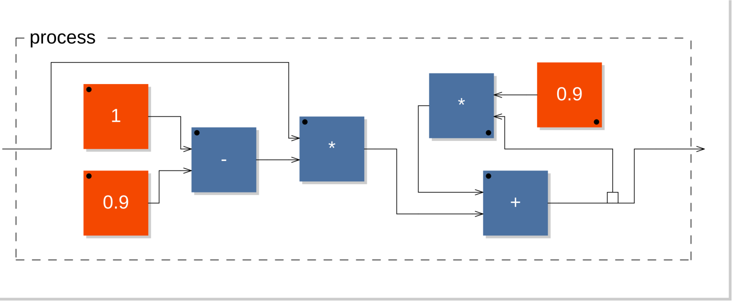


Back to the Filter

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

Using Faust:

```
smooth(c) = *(1-c) : + ~ *(c)
```



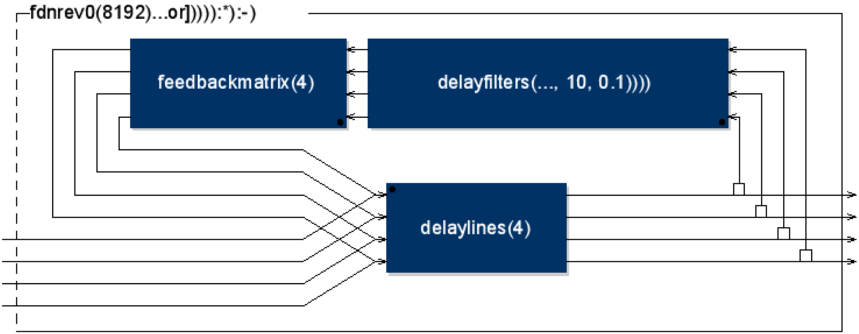
[For c = 0.9]

Feedback Delay Network

```
fdnrev(N, dp, freqs, durs, loopgainmax)
  = delaylines ~ (delayfilters : feedbackmatrix)
where
  delaylines     = rep(N,i,delay(dp[i])));
  delayfilters   = rep(N,filter(freqs,durs));
  feedbackmatrix = bhadamard(N);
```

Feedback Delay Network

```
fdnrev(N, dp, freqs, durs, loopgainmax)
  = delaylines ~ (delayfilters : feedbackmatrix)
where
  delaylines     = rep(N,i,delay(dp[i])));
  delayfilters   = rep(N,filter(freqs,durs));
  feedbackmatrix = bhadamard(N);
```



PL & Faust

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

PL & Faust

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

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.
- Stream/Data Flow Programming.

  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.
- Stream/Data Flow Programming.

  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.

DSP & Faust

- 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:

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

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

- Coinductive semantics [Boulmé, et al]: problematic.
- Didn't consider PACO, etc. . . .
- Our wish: Sequences $\mathbb{S}$ of a base type **R** [Auger2013]

Mechanized Semantics for Streams

- Coinductive semantics [Boulmé, et al]: problematic.
- Didn't consider PACO, etc. ...
- Our wish: Sequences $\mathbb{S}$ of a base type $\mathbf{R}$ [Auger2013]

Soundness needs stronger semantics (also [Guatto2014]):

$$[\![\vdash f : i \to o]\!]^n : [\![\underbrace{\mathbf{R} \times \ldots \times \mathbf{R}}_{i}]\!]^n \to [\![\underbrace{\mathbf{R} \times \ldots \times \mathbf{R}}_{o}]\!]^n$$

Index by number of steps; equality of streams more intensional wrt to $(\mathbb{N} \to \mathbf{R})$.

The Second Piece: Real Analysis

What about the base type **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...

The Second Piece: Real Analysis

What about the base type **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.

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.

To remedy this, we define a *program logic* for sample-level properties.

Sampled-Level Predicates

Definition (Sample-Level Property)

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

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

Sampled-Level Predicates

Definition (Sample-Level Property)

A property $P : \mathbb{S} \to \mathbb{B}$ is sample-level if there exists a characteristic predicate $\varphi : \mathbf{R} \to \mathbb{B}$ 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 $P : \mathbb{S} \to \mathbb{B}$ is sample-level if there exists a characteristic predicate $\varphi : \mathbf{R} \to \mathbb{B}$ 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.

A Sampled Logic

Definition (Sampled Judgment)

Given two characteristic predicates φ, ψ, we write

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

"for all input i meeting φ, the fi satisfies ψ."

Example

The stability judgment for smooth is written as:

$$\{x \in [a, b]\}\ \text{smooth}\ \{x \in [a, b]\}$$

Rules for The Sampled Logic

$$\frac{\forall i_1, i_2, (\varphi_1(i_1) \wedge \varphi_1(i_2)) \implies \psi(i_1 + i_2)}{\{\varphi_1, \varphi_2\} + \{\psi\}}\,\textit{Prim}$$

$$\frac{\{\varphi\}\ f\ \{\theta\} \qquad \{\theta\}\ g\ \{\psi\}}{\{\varphi\}\ f : g\ \{\psi\}}\,\textit{Comp}$$

$$\frac{\models \psi(x_0) \qquad \{\theta, \varphi\}\ f\ \{\psi\} \qquad \{\psi\}\ g\ \{\theta\}}{\{\varphi\}\ f \sim g\ \{\psi\}}\,\textit{Feed}$$

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

$$\dfrac{\dfrac{\Box}{\{I_{ab}\}\ *(1-c)\ \{I_{ab\bar{c}}\}} \qquad \dfrac{\dfrac{\Box}{\{I_{abc}, I_{ab\bar{c}}\} + \{I_{ab}\}} \qquad \dfrac{\Box}{\{I_{ab}\}\ *(c)\ \{I_{abc}\}}}{\{I_{ab\bar{c}}\} + \sim *(c)\ \{I_{ab}\}}}{\{i \in [a,b]\}\ *(1-c) : + \sim *(c)\ \{o \in [a,b]\}}$$

with:

$$\begin{aligned}
I_{ab}(x) &\equiv x \in [a, b] \\
I_{abc}(x) &\equiv x \in [a * c, b * c] \\
I_{ab\bar{c}}(x) &\equiv x \in [a * (1-c), b * (1-c)]
\end{aligned}$$

Stability of Smooth

Three main VC in the proof:

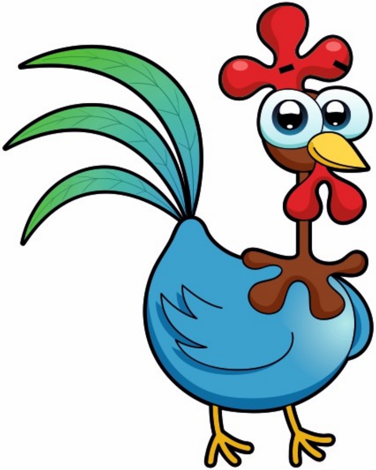
```
(* (1 - c) * i \in [(1 - c) * a, (1 - c) * b] *)
by rewrite ?ler_wpmul2r ?ler_subr_addr ?add0r.

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 * i \in [c * a, c * 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.

Linear System Theory

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
  fir(x)      = (x - x'') * g * (1-RR) / 2;
  feedback(v) = 2*R*cos(T) * v - RR * v';
  ....
```

Get and verify its transfer function:

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

Ongoing: Frequency Domain Analysis

Recall the Fourier Matrix:

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

or:

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

where ω 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.