A Taste of (formal) Sound Reasoning
A Tutorial

Emilio J. Gallego Arias, Pierre Jouvelot, Olivier Hermant

MINES ParisTech, PSL Research University, France

Linux Audio Conf 2015

e+lac@x80.org  @ejgallego
https://github.com/ejgallego/mini-faust-coq
Let’s start with a simple IIR filter:

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

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

What can we 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.

Answers given by standard DSP theory.
\[ \text{smooth}_n = (1 - c) \cdot x_n + c \cdot \text{smooth}_{n-1} \]

Natural questions are:

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

Answers given by standard DSP theory.

What about the implementation of the filter?

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

Natural questions are:
- Frequency response.
- Stability.
- Linearity/Time Invariance.

Answers given by standard DSP theory.

What about the implementation of the filter?

We dive into the realm of PL theory!

Paradigm shift!
Certainty
Certainty

“Absolute” confidence on something we believe.
Certainty

“Absolute” confidence on something we believe. How do we know something is “absolutely” true?
Many possible answers
Many possible answers

In the Programming Languages field, we want computers to check knowledge for us!
How does it work?

Welcome to Evidence!
Welcome to Evidence!

We will build a particular kind of evidence for a property of our filter, then use the computer to validate it.
Types of Evidence

Bob Hi Alice, my dog is feeling weird!

Alice I don’t believe you!
Types of Evidence

Bob  Hi Alice, my dog is feeling weird!
Alice I don’t believe you!

[Image of a pug dog in a pink costume]
We want to agree on a convention to produce and check evidence.

A logic is a language and a set of rules geared towards the production of symbolic evidence.
We want to agree on a convention to produce and check evidence.

A logic is a language and a set of rules geared towards the production of symbolic evidence.

Example Propositions

“Every even number is not prime.”
“Every complex polynomial has a root.”
“Every finite impulse filter is stable.”
Checking Validity: Inference

To check when a proposition holds, we need rules.

Example Rules

“If A and B hold, B holds.”

“If P holds for 0, and assuming P holds for n we can prove that P holds for n+1, then P holds for all n.”
The Theory of Forms

Truth
Truth lives in the idealistic, infinite universe.

\[ \Gamma \models \varphi \quad \text{if } \Gamma \text{ is true, then } \varphi \text{ is true} \]

Proof
Reasoning lives in the concrete, syntactic universe.

\[ \Gamma \vdash \varphi \quad \varphi \text{ can be proved from } \Gamma \]

using a valid application of the rules.
Linking the Worlds

\[ \Gamma \models \varphi \quad \text{if } \Gamma \text{ is true, then } \varphi \text{ is true} \]
\[ \Gamma \vdash \varphi \quad \varphi \text{ can be proved from } \Gamma \]

Main Properties

- Soundness: \( \Gamma \vdash \varphi \) implies \( \Gamma \models \varphi \).
Linking the Worlds

\[ \Gamma \models \varphi \quad \text{if } \Gamma \text{ is true, then } \varphi \text{ is true} \]
\[ \Gamma \vdash \varphi \quad \varphi \text{ can be proved from } \Gamma \]

Main Properties

- Soundness: \( \Gamma \vdash \varphi \) implies \( \Gamma \models \varphi \).
- Completeness: \( \Gamma \models \varphi \) implies \( \Gamma \vdash \varphi \).
Linking the Worlds

If \( \Gamma \) is true, then \( \phi \) is true.

\[ \Gamma \models \phi \] 

\( \Gamma \vdash \phi \) \( \phi \) can be proved from \( \Gamma \).

Main Properties

- **Soundness**: \( \Gamma \vdash \phi \) implies \( \Gamma \models \phi \).
- **Completeness**: \( \Gamma \models \phi \) implies \( \Gamma \vdash \phi \).
- **Consistency**: \( \not\vdash A \land \neg A \).
Linking the Worlds

Γ ⊨ ϕ if Γ is true, then ϕ is true
Γ ⊬ ϕ ϕ can be proved from Γ

Main Properties

- Soundness: Γ ⊬ ϕ implies Γ ⊨ ϕ.
- Completeness: Γ ⊨ ϕ implies Γ ⊬ ϕ.
- Consistency: ∄ A ∧ ¬A.

We are liberated from the complexity of the ideal, infinite world, we can now use mechanical, finitary rules to reason about it!
Assume we want our computer to check our deductions.
Assume we want our computer to check our deductions.

We could write a rule checker. But how do we know the rule checker is correct?
Assume we want our computer to check our deductions.

We could write a rule checker. But how do we know the rule checker is correct?

A crucial, fundamental idea:

Programs are Proofs!
Types are Propositions!
Computational Evidence

Welcome to Coq!

aptitude install coq
Welcome to Coq!

aptitude install coq

In Coq, proofs are precisely the well-typed functional programs. Type-checking validates our logical deductions!
BHK-Interpretation

Computational interpretation of logic

<table>
<thead>
<tr>
<th>Type</th>
<th>Proof / Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \land Q$</td>
<td>Record with proofs for $P$ and $Q$.</td>
</tr>
<tr>
<td>$P \rightarrow Q$</td>
<td>Program that takes a proof of $P$, then produces a proof of $Q$.</td>
</tr>
<tr>
<td>$\forall (x : P), Q(x)$</td>
<td>Program with input $\rho$ a proof of $P$, then produces a proof of $Q(\rho)$.</td>
</tr>
<tr>
<td>$\exists (x : P), Q(x)$</td>
<td>Pair $(w, W)$ of $w$ a proof for $P$ and $W$ a proof for $P(w)$.</td>
</tr>
<tr>
<td>$P \lor Q$</td>
<td>????</td>
</tr>
</tbody>
</table>
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?

1. Define the syntax of Faust inside Coq.
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.

[Seriously, we’d love to hear about 4!]
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two.
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two.
4. ?

[Seriously, we’d love to hear about 4!]
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two.
4. ?
5. Profit!

[Seriously, we’d love to hear about 4!]
Let’s Move Back to Audio

Use Coq to reason about audio programs, in particular, we’ll use a toy version of Faust!

What’s the plan?

1. Define the syntax of Faust inside Coq.
2. Define a representation for (sampled) sound.
3. Link the two.
4. ?
5. Profit!

[Seriously, we’d love to hear about 4!]
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 \( c = 0.9 \)]
Let’s do it!

\[ \text{smooth}(c) = (1-c) \cdot + \sim (c) \]
Semantics

We can “write” Faust programs inside Coq. Now we want to run them.
Semantics

We can “write” Faust programs inside Coq. Now we want to run them.

Output of Smooth

<table>
<thead>
<tr>
<th>T:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>I:</td>
<td>1.00</td>
<td>1.05</td>
<td>1.10</td>
<td>1.15</td>
<td>1.20</td>
<td>1.25</td>
<td>1.20</td>
<td>1.25</td>
</tr>
<tr>
<td>O:</td>
<td>0.10</td>
<td>0.19</td>
<td>0.28</td>
<td>0.37</td>
<td>0.45</td>
<td>0.53</td>
<td>0.61</td>
<td>0.68</td>
</tr>
</tbody>
</table>
What is Sound? Choices... 

We need to choose how to represent sound in Coq? In the formal world, we *pay* for every detail.

- Conceptual representations? ($\mathbb{R} \to \mathbb{R}$).
- Infinite representations? ($\mathbb{N} \to \mathbb{R}$)
- Finite representation? (seq $\mathbb{R}$)

We’ll use the last one.
Let’s do it!

<table>
<thead>
<tr>
<th>T:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>I:</td>
<td>1.00</td>
<td>1.05</td>
<td>1.10</td>
<td>1.15</td>
<td>1.20</td>
<td>1.25</td>
<td>1.20</td>
<td>1.25</td>
</tr>
<tr>
<td>O:</td>
<td>0.10</td>
<td>0.19</td>
<td>0.28</td>
<td>0.37</td>
<td>0.45</td>
<td>0.53</td>
<td>0.61</td>
<td>0.68</td>
</tr>
</tbody>
</table>
When is Smooth Stable?

We are in good shape, now, when is smooth stable?

\[ \text{smooth}_n = (1 - c)x_n + c \cdot \text{smooth}_{n-1} \]
When is Smooth Stable?

We are in good shape, now, 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:

$$\forall i \in [a, b], \ c \in (0, 1) \rightarrow \text{smooth}(c) \ i \in [a, b]$$
We can do the proof directly in Coq, it is not difficult but cumbersome in general.
Proving Stability

We can do the proof directly in Coq, it is not difficult but cumbersome in general.

But we can use better, higher-level reasoning principles: Use *program logics* and target global properties over all samples.
Sampled Logic

Definition
A sample-level property $\varphi$ holds for a signal $s$ if it holds for all samples. $\forall n. \varphi(s[n])$.
Boundedness is a sample-level property!
**Sampled Logic**

**Definition**
A sample-level property $\varphi$ holds for a signal $s$ if it holds for all samples. $\forall n.\varphi(s[n])$.

Boundedness is a sample-level property!

**Definition**
Assume a program $f$, then we write $\{\varphi\} f \{\psi\}$ for “for all inputs satisfying $\varphi$”, the output of $f$ satisfies $\psi$.

Stability for smooth is written:

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

\[ \forall i_1, i_2, (\varphi_1(i_1) \land \varphi_1(i_2)) \implies \psi(i_1(t) + i_2(t)) \]
\[ \text{Prim} \]
\[ \{\varphi_1, \varphi_2\} + \{\psi\} \]

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

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

\[ \{l_{ab}\} \ast (1 - c) \{l_{abc}\} \]

\[ \{l_{abc}, l_{abc}\} + \{l_{ab}\} \]

\[ \{l_{abc}\} + \sim \ast (c) \{l_{ab}\} \]

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

with:

\[ l_{ab}(x) \equiv x \in [a, b] \]

\[ l_{abc}(x) \equiv x \in [a \ast c, b \ast c] \]

\[ l_{abc}(x) \equiv x \in [a \ast (1 - c), b \ast (1 - c)] \]
Stability Proof
Conclusions

- Interesting exercise, we learned a lot!
- The full language is basically done.
- We need your help! Let us know what would be interesting to check!
- Most complaints about plugins cannot be solved by verification.
- We are investigating a slightly different approach.
- Working on linear systems theory, frequency domain properties.
Thanks!
Nyquist Theorem

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

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

where

- $f_s = 1/T_s$ sampling frequency
- $V(t)$ value of signal at time $t$
- $V[t] = V(t \cdot T_s)$ value of signal at time $t \cdot T_s$