The Next 700 Syntactic Models of Type Theory

Simon Boulier[1] **Pierre-Marie Pédrot**[2] Nicolas Tabareau[1]

[1]INRIA, [2]University of Ljubljana

CPP
17th January 2017

A Beginner's Tale

Historical recollection of a younger self using Coq:

— I need to prove that $\Pi x.\, f\ x = g\ x$ implies $f = g$ to...

— Nay, can't do that.

— Right, I'd also like to have $\Pi e_1\ e_2 : p = q.\ e_1 = e_2$. How...

— Nope, not possible either.

— Fine. And what about $\Pi A\ B : \texttt{Prop}.\, (A \leftrightarrow B) \rightarrow A = B$?

— Sigh.

Historical recollection of a younger self using Coq:

- — I need to prove that $\Pi x.\, f\, x = g\, x$ implies $f = g$ to...
- — Nay, can't do that.
- — Right, I'd also like to have $\Pi e_1\, e_2 : p = q.\, e_1 = e_2$. How...
- — Nope, not possible either.
- — Fine. And what about $\Pi A\, B : \texttt{Prop}.\, (A \leftrightarrow B) \rightarrow A = B$?
- — Sigh.

Are you kidding me? This *has* to be obviously true!

What You're Usually Told

If you ask why, generally you get something along the lines of:

> *"That's very simple to disprove. Let's consider the split comprehension category where the Grothendieck fibration is the well-known* ***blue-haired syzygetic Kardashian functor*** *and the cartesian structure is canonically given by the algebra morphisms of* ***hyper-loremipsum*** ω***-potatoids****. It is trivially a counter-model."*

What You're Usually Told

If you ask why, generally you get something along the lines of:

> *"That's very simple to disprove. Let's consider the split comprehension category where the Grothendieck fibration is the well-known* ***blue-haired syzygetic Kardashian functor*** *and the cartesian structure is canonically given by the algebra morphisms of* ***hyper-loremipsum*** ω***-potatoids****. It is trivially a counter-model."*

(Obviously up to my brain's isomorphisms. Any resemblance to nLab is purely coincidental.)

What You're Usually Told

If you ask why, generally you get something along the lines of:

> *"That's very simple to disprove. Let's consider the split compre*[illegible]*ategory where the Grothendie*[illegible] *is the well-known* ***blue-ha***[illegible]***ati***[illegible]***ian functor*** *and the cartesian structure is* [illegible] *the algebra morphisms of* ***hyper***[illegible]***pain ω-potatoids****. It is trivia*[illegible] [illegible]*er-model."*

(Obviously up to my brain's isomorphisms. Any resemblance to nLab is purely coincidental.)

We propose something that anybody* can understand instead.

What is a model?

- Takes syntax as input.
- Interprets it into some low-level language.
- Must preserve the meaning of the source.
- Refines the behaviour of under-specified structures.

What is a model?

- Takes syntax as input.
- Interprets it into some low-level language.
- Must preserve the meaning of the source.
- Refines the behaviour of under-specified structures.

Luckily we're computer scientists in here.

What is a model?

- Takes syntax as input.
- Interprets it into some low-level language.
- Must preserve the meaning of the source.
- Refines the behaviour of under-specified structures.

Luckily we're computer scientists in here.

« Oh yes, we call that a *compiler*... »

(Thanks, Curry-Howard!)

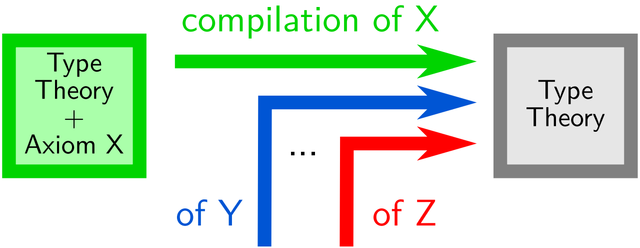
Syntactic Models

- I don't understand crazy category theory.
- But I understand well type-theory!
- And I know how to write program translations.

Syntactic Models

- I don't understand crazy category theory.
- But I understand well type-theory!
- And I know how to write program translations.

Let's write models as compilers from type theory into itself!



;

Syntactic Models II

Define $[\cdot]$ on the syntax and derive the type interpretation $[\![\cdot]\!]$ from it s.t.

$$\vdash M : A \qquad \text{implies} \qquad \vdash [M] : [\![A]\!]$$

Syntactic Models II

Define $[\cdot]$ on the syntax and derive the type interpretation $[\![\cdot]\!]$ from it s.t.

$$\vdash M : A \qquad \text{implies} \qquad \vdash [M] : [\![A]\!]$$

Obviously, that's subtle.

- The correctness of $[\cdot]$ lies in the meta (Darn, Gödel!)
- The translation must preserve typing (Not easy)
- In particular, it must preserve conversion (Argh!)

Syntactic Models II

Define $[\cdot]$ on the syntax and derive the type interpretation $[\![\cdot]\!]$ from it s.t.

$$\vdash M : A \qquad \text{implies} \qquad \vdash [M] : [\![A]\!]$$

Obviously, that's subtle.

- The correctness of $[\cdot]$ lies in the meta (Darn, Gödel!)
- The translation must preserve typing (Not easy)
- In particular, it must preserve conversion (Argh!)

Yet, a lot of nice consequences.

- Does not require non-type-theoretical foundations (*monism*)
- Can be implemented in your favourite proof assistant
- Easy to show (relative) consistency, look at $[\![\texttt{False}]\!]$
- Easier to understand computationally

In The Remainder of This Talk

700 Syntactic Models You Probably Didn't Know Provide the Most Striking Counter-Examples to Type Theory

In The Remainder of This Talk

700 Syntactic Models You Probably Didn't Know Provide the Most Striking Counter-Examples to Type Theory

The 578th Will Shock You!

In The Remainder of This Talk

700 Syntactic Models You Probably Didn't Know Provide the Most Striking Counter-Examples to Type Theory

The 578th Will Shock You!

(Just kidding. I don't want doctors to hate me.)

Where the Wild Things Are

— What is fully specified in type theory?

- **Inductive types**, because of dependent elimination.

Where the Wild Things Are

— What is fully specified in type theory?

- **Inductive types**, because of dependent elimination.

— What is *not* fully specified in type theory?
Everything else!

- **Functions**: only specified w.r.t. β-reduction
- **Co-inductive types**: only specified w.r.t. projections
- **Universes**: only specified w.r.t. rhs of a colon
- ...

Let's joyfully refine the intensional behaviour of random stuff in there.

Negating Functional Extensionality

First target: functions. The only thing you know about them:

$$(\lambda x : A.\, M)\, N \equiv M\{x := N\}$$

Negating Functional Extensionality

First target: functions. The only thing you know about them:

$$(\lambda x : A.\, M)\, N \equiv M\{x := N\}$$

Let's take advantage of this by mangling functions.

$$\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & (\lambda x : [\![A]\!].\, [M], \texttt{true}) \\
[M\, N] & := & [M].\pi_1\, [N] \\
[\square] & := & \square \\
[\Pi x : A.\, B] & := & (\Pi x : [\![A]\!].\, [\![B]\!]) \times \texttt{bool} \\
[\ldots] & := & \ldots \\
[\![A]\!] & := & [A]
\end{array}$$

Negating Functional Extensionality

First target: functions. The only thing you know about them:

$$(\lambda x : A.\, M)\, N \equiv M\{x := N\}$$

Let's take advantage of this by mangling functions.

$$\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & (\lambda x : [\![A]\!].\, [M], \texttt{true}) \\
[M\, N] & := & [M].\pi_1\, [N] \\
[\square] & := & \square \\
[\Pi x : A.\, B] & := & (\Pi x : [\![A]\!].\, [\![B]\!]) \times \texttt{bool} \\
[\ldots] & := & \ldots \\
[\![A]\!] & := & [A]
\end{array}$$

Obviously $\Gamma \vdash M : A$ implies $[\![\Gamma]\!] \vdash [M] : [\![A]\!]$.

Through The Looking Glass

Now, we interpret everything through the $[\cdot]$ translation.

- We call the source theory all terms that have some type $[\![A]\!]$
- Given $M : [\![A]\!]$ we can extend the source with a constant $M^\bullet : A$

$$[M^\bullet] := M$$

- Conversion is extended the same way:

$$M \equiv_{\texttt{source}} N := [M] \equiv_{\texttt{target}} [N]$$

Negating Functional Extensionality II

Syntactically, this means that you can extend the source theory with

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda' x : A.\, M : \Pi x : A.\, B}$$

defined as:

$$[\lambda' x : A.\, M] := (\lambda x : [\![A]\!].\, [M], \texttt{false})$$

Rembember:

$$\begin{array}{lcl} [\lambda x : A.\, M] & := & (\lambda x : [\![A]\!].\, [M], \texttt{true}) \\ [M\, N] & := & [M].\pi_1\, [N] \end{array}$$

Negating Functional Extensionality II

Syntactically, this means that you can extend the source theory with

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda' x : A.\, M : \Pi x : A.\, B}$$

defined as:

$$[\lambda' x : A.\, M] := (\lambda x : [\![A]\!].\, [M], \texttt{false})$$

Rembember:

$$\begin{array}{lcl} [\lambda x : A.\, M] & := & (\lambda x : [\![A]\!].\, [M], \texttt{true}) \\ [M\, N] & := & [M].\pi_1\, [N] \end{array}$$

Clearly this new abstraction has the same behaviour as the original one.

$$[(\lambda' x : A.\, M)\, N] \equiv [M\{x := N\}]$$

Negating Functional Extensionality III

Now, it is easy to see how to negate functional extensionality. Consider:

$$\Sigma(f\,g : 1 \to 1).\,(\Pi i : 1.\, f\; i = g\; i) \wedge f \neq g$$

Negating Functional Extensionality III

Now, it is easy to see how to negate functional extensionality. Consider:

$$\Sigma(f\,g : 1 \to 1).\,(\Pi i : 1.\, f\ i = g\ i) \wedge f \neq g$$

This is translated into something that is essentially:

$$\Sigma(f\,g : (1 \to 1) \times \texttt{bool}).\,(\Pi i : 1.\, f.\pi_1\ i = g.\pi_1\ i) \wedge f \neq g$$

(The actual translation is a little noisier, but this does not change the idea.)

Negating Functional Extensionality III

Now, it is easy to see how to negate functional extensionality. Consider:

$$\Sigma(f\,g : 1 \to 1).\,(\Pi i : 1.\, f\ i = g\ i) \wedge f \neq g$$

This is translated into something that is essentially:

$$\Sigma(f\,g : (1 \to 1) \times \texttt{bool}).\,(\Pi i : 1.\, f.\pi_1\ \ i = g.\pi_1\ \ i) \wedge f \neq g$$

(The actual translation is a little noisier, but this does not change the idea.)

Take $f := [\lambda x : 1.\, x]$ and $g := [\lambda' x : 1.\, x]$, and *voilá*!

Where We Cheated

We did not explicit the rules of the source theory.

Where We Cheated

We did not explicit the rules of the source theory.

In particular, it is clear that the model invalidates η-rules.

$$\begin{array}{ccc} [\lambda x : A.\, M\ x] & \not\equiv & [M] \\ \equiv & & \equiv \\ (\lambda x : [\![A]\!].\, [M].\pi_1\ x, \mathtt{true}) & \not\equiv & [M] \end{array}$$

It's much harder to negate extensionality while preserving η.
(Dialectica does that.)

Stream extensionality

We can use a very similar trick to intentionalize steams. Idea:

$$[\![\texttt{stream}\ A]\!] := (\texttt{stream}\ [\![A]\!]) \times \texttt{bool}$$

This interprets all negative co-inductive properties ("co-pattern style").

And there is no reasonable η-rule on cofixpoints anyway.

Stream extensionality

We can use a very similar trick to intentionalize steams. Idea:

$$[\![\texttt{stream}\ A]\!] := (\texttt{stream}\ [\![A]\!]) \times \texttt{bool}$$

This interprets all negative co-inductive properties ("co-pattern style").

And there is no reasonable η-rule on cofixpoints anyway.

Then just as easily we show that:

$$\Sigma(f\,g : \texttt{stream}\ 1).\,(\texttt{bisimilar}\ 1\ f\ g) \wedge f \neq g$$

Type Extensionality

Once again, the same trick can be applied to types.

$$\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & \lambda x : [\![A]\!].\, [M] \\
[M\, N] & := & [M]\, [N] \\
[\square_i] & := & (\square_i \times \texttt{bool}, \texttt{true}) \\
[\Pi x : A.\, B] & := & ((\Pi x : [\![A]\!].\, [\![B]\!]), \texttt{true}) \\
[\![A]\!] & := & [A].\pi_1
\end{array}$$

Type Extensionality

Once again, the same trick can be applied to types.

$$
\begin{array}{lcl}
[x] & := & x \\
[\lambda x : A.\, M] & := & \lambda x : [\![A]\!].\, [M] \\
[M\, N] & := & [M]\, [N] \\
[\Box_i] & := & (\Box_i \times \texttt{bool}, \texttt{true}) \\
[\Pi x : A.\, B] & := & ((\Pi x : [\![A]\!].\, [\![B]\!]), \texttt{true}) \\
[\![A]\!] & := & [A].\pi_1
\end{array}
$$

"New types are a pair of a type and a boolean!" Tricky fixpoint:

$$[\Box_i] : [\![\Box_{i+1}]\!] \quad \Leftrightarrow \quad (\Box_i \times \texttt{bool}, \texttt{true}) : \Box_{i+1} \times \texttt{bool}$$

Negating Propositional Extensionality

You can translate an impredicative universe alike:

$$[*] \quad := \quad (* \times \texttt{bool}, \texttt{true})$$

It is still an impredicative universe!

Negating Propositional Extensionality

You can translate an impredicative universe alike:

$$[*] \quad := \quad (* \times \texttt{bool}, \texttt{true})$$

It is still an impredicative universe!

It is then easy to show:

$$\begin{array}{rl} & [\![\Sigma(P\,Q : *).\,(P \leftrightarrow Q) \wedge P \neq Q]\!] \\ \sim & \Sigma(P\,Q : * \times \texttt{bool}).\,(P.\pi_1 \leftrightarrow Q.\pi_1) \wedge P \neq Q \end{array}$$

Take for instance True and its evil twin $\texttt{True}^\dagger$:

$$\begin{array}{rcl} [\texttt{True}] & := & (\texttt{True}, \texttt{true}) \\ [\texttt{True}^\dagger] & := & (\texttt{True}, \texttt{false}) \end{array}$$

Where Will They Stop?

- This shows that universes are “amorphous” in type theory
- The only thing that matters is $[\![\cdot]\!]$ in the translation!
- We simply used a projection here

Where Will They Stop?

- This shows that universes are “amorphous” in type theory
- The only thing that matters is $[\![\cdot]\!]$ in the translation!
- We simply used a projection here

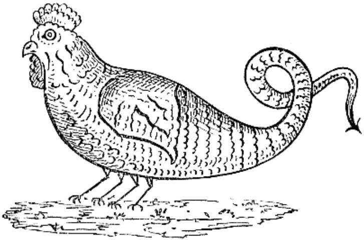
Let’s do **way much better** (or worse, depends on your beliefs).

Where Will They Stop?

- This shows that universes are “amorphous” in type theory
- The only thing that matters is $[\![\cdot]\!]$ in the translation!
- We simply used a projection here

Let’s do **way much better** (or worse, depends on your beliefs).

Let’s turn Coq into Python!



The Basilisk

Idea: if $A : \Box$ then $[A]$: TYPE, the type of inductive-recursive **codes**!

```
Inductive TYPE :=
| 𝒰 : TYPE
| Pi : Π (A : TYPE), (Elt A → TYPE) → TYPE
| ...
with Elt (A : TYPE) := match A with
| 𝒰 ⇒ TYPE
| Pi A B ⇒ Π (x : Elt A), Elt (B x)
| ...
end.
```

(Note: We need to stratify a bit to make this work.)

The Basilisk

Idea: if $A : \square$ then $[A]$: TYPE, the type of inductive-recursive **codes**!

```
Inductive TYPE :=
| 𝒰 : TYPE
| Pi : Π (A : TYPE), (Elt A → TYPE) → TYPE
| ...
with Elt (A : TYPE) := match A with
| 𝒰 ⇒ TYPE
| Pi A B ⇒ Π (x : Elt A), Elt (B x)
| ...
end.
```

(Note: We need to stratify a bit to make this work.)

$$\begin{array}{lcl} [\square] & := & \mathcal{U} \\ [\Pi x : A.\, B] & := & \texttt{Pi}\ [A]\ (\lambda x : [\![A]\!].\, [B]) \\ [\![A]\!] & := & \texttt{Elt}\ [A] \end{array}$$

Behold!

This allows definitions by case-analysis on types!

For instance, it is now possible to define:

- $f : \Pi A : \square.\, A \to A \qquad (\sim \Pi A : \mathtt{TYPE}.\, \mathtt{Elt}\ A \to \mathtt{Elt}\ A)$
- $f\,\mathtt{bool} : \mathtt{bool} \to \mathtt{bool}$ is negation
- $f\ A$ is identity otherwise

Behold!

This allows definitions by case-analysis on types!

For instance, it is now possible to define:

- $f : \Pi A : \square.\, A \to A \qquad (\sim \Pi A : \mathtt{TYPE}.\, \mathtt{Elt}\ A \to \mathtt{Elt}\ A)$
- $f\ \mathtt{bool} : \mathtt{bool} \to \mathtt{bool}$ is negation
- $f\ A$ is identity otherwise

Morally it is the most anti-parametric thing one can do. Abstractly:

Type theory is compatible with ad-hoc polymorphism.

(Yes, this surprised me as well.)

What else

We have a soundness proof in Coq for most of the previous translations.

- Based on Siles's definition of De Bruijn implementaton of CC
- "Deep embedding"
- Shows that the model preserve consistency in a easy way

There is also an experimental plugin to translate terms automagically.

`https://github.com/CoqHott/Program-translations-CC-omega`

Conclusion

- We've described a simple class of models
- Rooted in computer science POV
- Sufficient to negate a lot of extensionality principles
  - Functions
  - Co-inductive types
  - Universes
- Implemented them!

Conclusion

- We've described a simple class of models
- Rooted in computer science POV
- Sufficient to negate a lot of extensionality principles
  - Functions
  - Co-inductive types
  - Universes
- Implemented them!

- We advocate for this kind of models
- A few more instances from the literature
- Stay tuned!

Thanks for your attention.