Models of type theory given by program translation

Simon Boulier, Pierre-Marie Pédrot, Nicolas Tabareau

École des Mines de Nantes - INRIA

- Can we prove $(\forall x.\ f\ x = g\ x) \to f = g$?

- Can we prove that a term of $\forall A.\ A \to A$ is necessarily the identity function?

- …

5.2 Axioms

30 What axioms can be safely added to Coq?

There are a few typical useful axioms that are independent from the Calculus of Inductive Constructions and that can be safely added to Coq. These axioms are stated in the directory `Logic` of the standard library of Coq. The most interesting ones are

- Excluded-middle: $\forall$ A:Prop, A $\vee$ $\neg$ A
- Proof-irrelevance: $\forall$ A:Prop $\forall$ p1 p2:A, p1=p2
- Unicity of equality proofs (or equivalently Streicher's axiom K): $\forall$ A $\forall$ x y:A $\forall$ p1 p2:x=y, p1=p2
- The axiom of unique choice: $\forall$ x $\exists!$ y R(x,y) $\rightarrow$ $\exists$ f $\forall$ x R(x,f(x))
- The functional axiom of choice: $\forall$ x $\exists$ y R(x,y) $\rightarrow$ $\exists$ f $\forall$ x R(x,f(x))
- Extensionality of predicates: $\forall$ P Q:A$\rightarrow$ Prop, ($\forall$ x, P(x) $\leftrightarrow$ Q(x)) $\rightarrow$ P=Q
- Extensionality of functions: $\forall$ f g:A$\rightarrow$ B, ($\forall$ x, f(x)=g(x)) $\rightarrow$ f=g

Here is a summary of the relative strength of these axioms, most proofs can be found in directory `Logic` of the standard library. The justification of their validity relies on the interpretability in set theory.

https://coq.inria.fr/faq

Can we prove **Funext** $:= \forall f\, g,\ (\forall x.\ f\ x = g\ x) \to f = g$?

Funext is independent of Coq. That means:

1. **Funext** is not provable in Coq:

$$\texttt{Coq} + \neg\textbf{Funext} \nvdash t : \Pi X : \square.\ X$$

   i.e. $\texttt{Coq} + \neg$**Funext** consistent

2. **Funext** does not introduce inconsistency:

$$\texttt{Coq} + \textbf{Funext} \nvdash t : \Pi X : \square.\ X$$

   i.e. $\texttt{Coq} +$ **Funext** consistent

Post-talk edit: it seems that this reasoning of reducing independence to consistency is classical ...

Can we prove **Funext** $:= \forall f\, g,\ (\forall x.\ f\ x = g\ x) \to f = g$?

Funext is independent of Coq. That means:

1. **Funext** is not provable in Coq:

   $$\texttt{Coq} + \neg\textbf{Funext} \nvdash t : \Pi X : \Box.\ X$$

   i.e. $\texttt{Coq} + \neg$**Funext** consistent provided that `Coq` is consistent

2. **Funext** does not introduce inconsistency:

   $$\texttt{Coq} + \textbf{Funext} \nvdash t : \Pi X : \Box.\ X$$

   i.e. $\texttt{Coq} +$ **Funext** consistent provided that `Coq` is consistent

Post-talk edit: it seems that this reasoning of reducing independence to consistency is classical . . .

Proving that $\mathcal{T}$ is consistent:

syntactic way proving confluence and normalization

semantic way give an interpretation in a model

Proving that $\mathcal{T}$ is consistent:

syntactic way proving confluence and normalization

semantic way give an interpretation in a model

- set theoretic models
- syntactic models
- program translations

A model is (for instance) a category with families.

1 - Set theoretic model

E.g. the set model.

$$
\begin{array}{rcl}
\text{context} & \rightsquigarrow & \text{set} \\
\text{type} & \rightsquigarrow & \text{set family} \\
\text{proposition} & \rightsquigarrow & \text{either } \{*\} \text{ or } \emptyset \\
t =_A u & \rightsquigarrow & \{* \mid t = u\} \\
\ldots & &
\end{array}
$$

Funext holds in the set models.

Remark: There are numerous variations of the set model: groupoid model, . . .

Problem: You have to learn (precise) set theory!

E.g. the universes are interpreted by large cardinals and Grothendieck universes ...

2 - Syntactic model: a model of $\mathcal{S}$ reusing type theoretic construction of $\mathcal{T}$.

E.g. : the term model

$$\begin{array}{rcl} \text{context of } \mathcal{S} & \rightsquigarrow & \text{type of } \mathcal{T} \\ \text{type of } \mathcal{S} & \rightsquigarrow & \text{type family of } \mathcal{T} \\ \text{term of } \mathcal{S} & \rightsquigarrow & \text{term of } \mathcal{T} \end{array}$$

2 - Syntactic model: a model of $\mathcal{S}$ reusing type theoretic construction of $\mathcal{T}$.

E.g. : the term model

$$\begin{array}{rcl} \text{context of } \mathcal{S} & \rightsquigarrow & \text{type of } \mathcal{T} \\ \text{type of } \mathcal{S} & \rightsquigarrow & \text{type family of } \mathcal{T} \\ \text{term of } \mathcal{S} & \rightsquigarrow & \text{term of } \mathcal{T} \end{array}$$

3 - Program translation:

$$\begin{array}{rcl} \text{context of } \mathcal{S} & \rightsquigarrow & \text{context of } \mathcal{T} \\ \text{type of } \mathcal{S} & \rightsquigarrow & \text{type of } \mathcal{T} \\ \text{term of } \mathcal{S} & \rightsquigarrow & \text{term of } \mathcal{T} \end{array}$$

Compilation of $\mathcal{S}$ toward $\mathcal{T}$.

set models $<$ syntactic models $<$ program translations

Set theoretic models

- realize many things

Syntactic models

- simpler
- rely only on type theory

Program translations

- still simpler (independent of the notion of model)
- implementable and modular (composition)

Not new:

- Gödel translation, CPS translations, ...
- subset model (Hofmann)
- forcing
- parametricity
- Dialectica translation
- ...

1. Program translations

2. Negate Funext

3. Other translations
   - Negate Propext
   - Negate Streamext
   - Pattern-matching on Type

CC_ω

We will consider variations of CC_ω.

Terms and types :

- x
- $\square_0, \square_1, \square_2, \ldots$
- $\Pi x : A.\ B, \quad \lambda x : A.\ t, \quad t\ u$
- $\Sigma x : A.\ B, \quad (t, u), \quad \pi_1\ t, \quad \pi_2\ t$
- $\mathbb{B}$
- $t =_A u$

+ streams, `Prop`, inductive-recursive types, ...

Program Translation

$$\begin{array}{rcl} \mathcal{S} & \longrightarrow & \mathcal{T} \\ \text{term } t & \rightsquigarrow & \text{term } [t] \\ \text{type } A & \rightsquigarrow & \text{type } [\![A]\!] \\ \text{context } \Gamma & \rightsquigarrow & \text{context } [\![\Gamma]\!] \end{array}$$

where $[\![A]\!] := \iota\ [A]$

$$\iota : \text{term} \to \text{term}$$

and $[\![\Gamma]\!] := x_1 : [\![A_1]\!], \ldots, x_n : [\![A_n]\!]$

$$\Gamma = x_1 : A_1, \ldots, x_n : A_n$$

Program Translation

computational soundness

$$\text{if } t \equiv u \text{ then } [t] \equiv [u],$$

typing soundness

$$\text{if } \Gamma \vdash t : A \text{ then } [\![\Gamma]\!] \vdash [t] : [\![A]\!],$$

consistency preservation

$$\text{if } [\![\bot_{\mathcal{S}}]\!] \text{ is inhabited then } \bot_{\mathcal{T}} \text{ is inhabited too.}$$

Theorem

Under those conditions, the consistency of $\mathcal{T}$ implies the one of $\mathcal{S}$.

Negate Funext

Goal: $\mathrm{CC}_\omega + \textbf{notFunext} \longrightarrow \mathrm{CC}_\omega$

Where **notFunext** axiom of type :

$$\Big(\Pi(A\ B : \square)(f\, g : A \to B).(\Pi\, x : A.\ f\ x = g\ x) \to f = g\Big) \to \bot$$

Negate Funext: Translation

$\mathrm{CC}_\omega + \textbf{notFunext} \longrightarrow \mathrm{CC}_\omega$

$$
\begin{array}{lcl}
[\Pi\, x : A.\ B] & := & (\Pi\, x : [A].\ [B]) \times \mathbb{B} \\
[\lambda\, x : A.\ t] & := & (\lambda\, x : [A].\ [t], \texttt{true}) \\
[t\ \ u] & := & \pi_1\ [t]\ [u] \\
[\square_i] & := & \square_i \\
[x] & := & x \\
\ldots & & \\
[\textbf{notFunext}] & := & \text{(cf. demo)} \\
[\![A]\!] & := & [A]
\end{array}
$$

notFunext :

$$\Big(\Pi(A\ B : \square)(f\ g : A \to B).(\Pi\, x : A.\ f\ \ x = g\ \ x) \to f = g\Big) \to \bot$$

Negate Funext: Correction

Lemma

If $\Gamma \vdash t : A$, *then* $[\Gamma] \vdash [t] : [A]$.

Proof.

□

Negate Funext: Correction

Lemma
If $\Gamma \vdash t : A$, then $[\Gamma] \vdash [t] : [A]$.

Proof.
E.g. : rules of lambda

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\ t : \Pi x : A.\ B} \qquad \frac{[\Gamma], x : [A] \vdash [t] : [B]}{[\Gamma] \vdash [\lambda x : A.\ t] \ :?\ [\Pi x : A.\ B]}$$

ok because
$$[\lambda x : A.\ t] = (\lambda x : [A].\ [t], \mathtt{true})$$
$$[\Pi x : A.\ B] = (\Pi x : [A].\ [B]) \times \mathbb{B}$$

$\square$

Negate Funext: Correction

Lemma

If $\Gamma \vdash t : A$, *then* $[\Gamma] \vdash [t] : [A]$.

Proof.

E.g. : conversion rule

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \Box \qquad A \equiv B}{\Gamma \vdash t : B}$$

ok using computational soundness

$\Box$

Negate Funext: Correction

Lemma

If $\Gamma \vdash t : A$*, then* $[\Gamma] \vdash [t] : [A]$.

Proof.

E.g. : **notFunext** rule

$$\frac{}{\vdash \textbf{notFunext} : (\Pi(A\ B : \square)(f\ g : A \to B). \dots) \to \bot}$$

demo!

□

Negate Funext: Correction

Lemma

If $[\![\Pi X : \square.\ X]\!]$ *is inhabited, then* $\Pi X : \square.\ X$ *is inhabited too.*

Proof.

ok because $[\![\Pi X : \square.\ X]\!] = (\Pi X : \square.\ X) \times \mathbb{B}$ $\square$

Negate Funext: Consequence

Theorem

If CC_ω *is consistent, then* $\mathrm{CC}_\omega +$ **notFunext** *is consistent too.*

Negate Funext: Formalization

Formalization of computational soundness, typing soundness, and preservation of consistency.

Deep embedding using de Bruijn indices.

Rely on the Coq contrib `PTSATR`.

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

Negate Funext: Formalization

```
Inductive Term : Set :=
  | Var : ℕ → Term
  | Sort : Sorts → Term
  | Π : Term → Term → Term
  | λ : Term → Term → Term
  | App : Term → Term → Term
  | Eq : ∀ (A t₁ t₂ : Term), Term
  | refl : Term → Term
  | J : ∀ (A P t₁ u t₂ p : Term), Term.
```

```
Fixpoint tsl (t : S.Term) : T.Term :=
 match t with
   | S.Var v  ⇒  T.Var v
   | S.Sort s  ⇒  T.Sort s
   | S.Π A B  ⇒  T.Σ (Π Aᵗ Bᵗ) Bool
   | S.λ A M  ⇒  T.Pair (λ Aᵗ Mᵗ) true
   | S.App M N  ⇒  T.App (π₁ Mᵗ) Nᵗ
   | S.Eq A t₁ t₂  ⇒  T.Eq Aᵗ t₁ᵗ t₂ᵗ
   | S.refl e  ⇒  T.refl eᵗ
   | S.J A P t₁ u t₂ p  ⇒  T.J Aᵗ Pᵗ t₁ᵗ uᵗ t₂ᵗ pᵗ
 end where "Mᵗ" := (tsl M).
```

```
Theorem tsl_correctness :
    (∀ Γ,  Γ ⊣ → Γᵗ ⊣) ∧ (∀ Γ M A, Γ ⊢ M : A → Γᵗ ⊢ Mᵗ : Aᵗ).
```

Negate Funext: Formalization

Another formalization:

`https://github.com/TheoWinterhalter/formal-type-theory`

- explicit substitutions instead of de Bruijn indices
- modular way to add and remove feature to the base theory

1. Program translations

2. Negate Funext

3. Other translations
   - Negate Propext
   - Negate Streamext
   - Pattern-matching on Type

1. Program translations

2. Negate Funext

3. Other translations

   Negate Propext

   Negate Streamext

   Pattern-matching on Type

Negate Propext

$\texttt{Prop}$ impredicative universe:

$$\frac{A : \Box_i \qquad x : A \vdash P : \texttt{Prop}}{\Pi\, x : A.\ P : \texttt{Prop}}$$

Goal : showing that $(\texttt{P} \leftrightarrow \texttt{Q}) \not\rightarrow (\texttt{P} = \texttt{Q})$ for $\texttt{P}, \texttt{Q} : \texttt{Prop}$.

Negate Propext

$$\mathrm{CC}_\omega + \texttt{Prop} + \textbf{notPropext} \longrightarrow \mathrm{CC}_\omega + \texttt{Prop}$$

$$\begin{array}{lcl}
[\square_i] & := & (\square_i \times \mathbb{B}, \texttt{true}) \\
[\texttt{Prop}] & := & (\texttt{Prop} \times \mathbb{B}, \texttt{true}) \\
[\Pi\, x : A.\ B] & := & (\Pi\, x : [\![A]\!].\ [\![B]\!], \texttt{true}) \\
\ldots & & \\
[\![A]\!] & := & \pi_1\ [A]
\end{array}$$

Remark : we have $[\square_i] \ : \ [\![\square_{i+1}]\!]$

because $[\square_i] = (\square_i \times \mathbb{B}, \texttt{true})$
and $[\![\square_{i+1}]\!] = \pi_1\ [\square_{i+1}] = \square_{i+1} \times \mathbb{B}$.

Negate Streamext

Goal: $\texttt{Bisim}\ \texttt{s}_1\ \texttt{s}_2 \;\not\to\; \texttt{s}_1 = \texttt{s}_2$ for $\texttt{s}_1, \texttt{s}_2 : \texttt{Stream}\ A$.

$$\mathrm{CC}_\omega + \texttt{Stream} + \textbf{notStreamext} \longrightarrow \mathrm{CC}_\omega + \texttt{Stream}$$

$$\begin{aligned}
[\texttt{Stream}\ A] &:= (\texttt{Stream}\ [\![A]\!]) \times \mathbb{B} \\
[\texttt{hd}\ t] &:= \texttt{hd}\ (\pi_1\ [t])
\end{aligned}$$

Pattern-matching on $\square$

$$f : \Pi A : \square.\ A \to A$$

$$f := \lambda(A : \square).\quad \begin{array}{l} \texttt{match}\ A\ \texttt{with} \\ \mid \mathbb{B} \Rightarrow \texttt{neg} \\ \mid \Pi x : B.\ C \Rightarrow \texttt{id} \\ \mid \square \Rightarrow \texttt{id} \\ \texttt{end} \end{array}$$

$$\begin{aligned} f\ \mathbb{B} &\mapsto \texttt{neg} \\ f\ (\mathbb{B} \to \mathbb{B}) &\mapsto \texttt{id} \end{aligned}$$

Operator written `univ_rec`.

Pattern-matching on $\square$

Idea : translate $\square$ by an inductive-recursive type on which pattern-matching is allowed.

Pattern-matching on $\square$

Idea : translate $\square$ by an inductive-recursive type on which pattern-matching is allowed.

```
Inductive TYPE : □ :=
| B   :  TYPE
| Pi  :  Π(a : TYPE)(b : Elt a → TYPE). TYPE
| U   : TYPE
with Elt : TYPE → □ := fun
| B        ⇒  𝔹
| Pi a b   ⇒  Π(x : Elt a). Elt (b x)
| U        ⇒  TYPE.
```

Pattern-matching on $\Box$

```
Inductive TYPE : □ :=
| B  : TYPE
| Pi : Π(a : TYPE)(b : Elt a → TYPE). TYPE
| U  : TYPE
with Elt : TYPE → □ := fun
| B       ⇒ 𝔹
| Pi a b  ⇒ Π(x : Elt a). Elt (b x)
| U       ⇒ TYPE.
```

$$\mathrm{CC}_\omega + \texttt{univ\_rec} \longrightarrow \mathrm{CC}_\omega + \texttt{TYPE}$$

$$\begin{aligned}
[\Box] &:= \texttt{U} \\
[\Pi x : A.\ B] &:= \texttt{Pi}\ [A]\ (\lambda x : [\![A]\!].\ [B]) \\
[\lambda x : A.\ t] &:= \lambda x : [\![A]\!].\ [t] \\
[\texttt{univ_rec}] &:= \texttt{TYPE_rec} \\
\ldots & \\
[\![A]\!] &:= \texttt{Elt}\ [A]
\end{aligned}$$

Pattern-matching on $\square$

Theorem

If $\mathrm{CC}_\omega + \texttt{TYPE}$ *is consistent, then* $\mathrm{CC}_\omega + \texttt{univ\_rec}$ *is consistent too.*

Pattern-matching on $\Box$

Theorem

If $\mathrm{CC}_\omega + \texttt{TYPE}$ *is consistent, then* $\mathrm{CC}_\omega + \texttt{univ\_rec}$ *is consistent too.*

Without type-in-type:

Theorem

If $\mathrm{CC}_\omega + (\texttt{TYPE})_{i\in\mathbb{N}}$ *is consistent, then* $\mathrm{CC}_\omega^{\mathit{expl}} + \texttt{univ\_rec}$ *is consistent too.*

Summary

Models given by program translation: $\mathcal{S} \longrightarrow \mathcal{T}$.

Benefits:

- simple
- use only type theory
- modular
- implementable

Summary

4 translations (CC_ω + *something* $\longrightarrow \mathrm{CC}_\omega$) :

- **notFunext**
- **notStreamext**
- **notPropext**
- **univ_rec**

notFunext, notStreamext, notPropext: Formalized and implemented as plugin.

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

Remark: all rely on the fact that negative types are under specified.

Future Work

Defining a generic plugin using `Template Coq`.
`https://github.com/gmalecha/template-coq`

```
Inductive term : Set :=
 | tRel     : ℕ → term
 | tEvar    : ℕ → term
 | tSort    : sort → term
 | tCast    : term → cast_kind → term → term
 | tProd    : name → term → term → term
 | tLambda  : name → term → term → term
 | ...
```

Future Work

We could define:

- $tsl\_term : term \to term$
- $tsl\_type : term \to term$

And get a ML function acting on Coq terms by quoting mechanism.

Then, we could prove that the translation is correct by reifying the typing judgment of Coq as an inductive.

For thinking in RER ...

Find a model that negates:

$$\lambda\, b : \mathbb{B}.\ b \ = \ \lambda\, b : \mathbb{B}.\ \texttt{if}\ b\ \texttt{then}\ \mathsf{true}\ \texttt{else}\ \mathsf{false}$$

(Harder than negating funext because

$$\forall\, b,\ \ b \ = \ \texttt{if}\ b\ \texttt{then}\ \mathsf{true}\ \texttt{else}\ \mathsf{false}$$

is provable).