Extending Type Theory with Forcing

Guilhem Jaber

Ecole des Mines de Nantes
LINA - Ascola

Joint Work with Nicolas Tabareau and Matthieu Sozeau

LI 2012
Proofs and programs
CIRM - Luminy - Marseille
14 February 2012
Cohen Forcing in Set Theory

\Rightarrow \text{Build a model negating the Continuum Hypothesis.}

\Rightarrow \text{Restatement in term of } Sheaves \text{ by Lawvere and Tierney.}
Some Background

- Cohen Forcing in Set Theory
  - Build a model negating the Continuum Hypothesis.
  - Restatement in term of *Sheaves* by Lawvere and Tierney.

- Hiding step-indexing in Semantics of Programming Languages
  - *Very Modal Model* of Appel, Melliès, Richards, and Vouillon.
  - *Topos of Trees* of Birkedal, Møgelberg, Schwinghammer, and Støvring.
Some Background

- Cohen Forcing in Set Theory
  - Build a model negating the Continuum Hypothesis.
  - Restatement in term of Sheaves by Lawvere and Tierney.

- Hiding step-indexing in Semantics of Programming Languages
  - *Very Modal Model* of Appel, Melliès, Richards, and Vouillon.
  - *Topos of Trees* of Birkedal, Møgelberg, Schwinghammer, and Støvring.

- Krivine Realizability and Forcing
  - Realizability as “non-commutative forcing”.
  - *Forcing as program transformation* of Miquel.
Some impossibilities in Coq

- General inductive types:
  
  Inductive D := Lambda : (D -> D) -> D.

  \[\text{Error: Non strictly positive occurrence of "D" in "(D -> D) -> D".}\]
Some impossibilities in Coq

- General inductive types:
  \[
  \text{Inductive } D \equiv \text{Lambda } : (D \to D) \to D.
  \]

\[\mapsto\text{Error: Non strictly positive occurrence of "D" in 
\((D \to D) \to D".}\]

- Computational content for Axioms:

  Axiom UltrafilterNat : exists mu:(nat \to Prop) \to Prop,
  \forall f:(nat \to Prop), (mu(f) \lor mu(1-f)) \land \ldots

  Theorem Ramsey : \ldots
  Proof. \ldots \text{pose UltrafilterNat.} \ldots \text{Qed.}
  Print Ramsey.

\[\mapsto\text{UltrafilterNat appears as a black box in the proofterm of}
\text{Ramsey.}\]
Extending Type Theory

- Adding new reasoning/objects to the theory
  - Unrestricted Inductive Types
  - Ultrafilter over $\mathbb{N}$
  - Negation of the Continuum Hypothesis.

- Without using Axioms
  - No problem of coherence
  - Give them a computational content.

- Translating these new features in the ground system
  - Translating Types, Propositions and Proofs
  - Translation as Approximation.
Transformation

\[
\begin{aligned}
\text{Extended Theory} \\
\mathbb{[\cdot]}_\rho \\
\text{Ground Theory}
\end{aligned}
\]
Transformation

Extended Theory

\[ [-]_{p_1} \quad [-]_{p_2} \quad [-]_{p_3} \]

Ground Theory

Ground Theory

Ground Theory
Transformation

$\theta_{p_1 \rightarrow p_2}$

$\theta_{p_2 \rightarrow p_3}$
Forcing translation

- The translation is indexed by *forcing conditions* $p_1, p_2, \ldots$. 
Forcing translation

- The translation is indexed by *forcing conditions* $p_1, p_2, \ldots$.

- A type $T$ is translated as $\llbracket T \rrbracket_p$ with restriction functions

  $$\theta^T_{p \to q} : \llbracket T \rrbracket_p \to \llbracket T \rrbracket_q \text{ for } q \leq p.$$ 

- $\llbracket T \rrbracket_p$ and the $\theta^T_{p \to q}$ are defined as the two components of the primitive translation $\llbracket T \rrbracket_p$. 
Forcing translation

- The translation is indexed by *forcing conditions* $p_1, p_2, \ldots$.

- A type $T$ is translated as $[T]_p$ with restriction functions $\theta^T_{p \rightarrow q} : [T]_p \rightarrow [T]_q$ for $q \leq p$.

- $[T]_p$ and the $\theta^T_{p \rightarrow q}$ are defined as the two components of the primitive translation $[T]_p$.

- When $M$ is a proof, solely $[M]_p$ exists.
Forcing Conditions

- A forcing condition is a set of information.

- The set of forcing conditions $\mathcal{P}$ is ordered.
  
  $p \leq q$ means that $p$ contains more information than $q$.

\[ \mathcal{P}_p \overset{\text{def}}{=} \{ q \in \mathcal{P} \mid q \leq p \} \]
A forcing condition is a set of information.

The set of forcing conditions \( \mathcal{P} \) is ordered.
\( p \leq q \) means that \( p \) contains more information than \( q \).

\[ \mathcal{P}_p \overset{def}{=} \{ q \in \mathcal{P} \mid q \leq p \} \]

Examples:
\( \vdash \) Step-indexing:
- Divergence is approximated as *reducing at least \( n \) times*
- \( n \in \mathbb{N} \) approximates \( \omega \).

\( \vdash \) Cohen Forcing: \( p \) approximate the generic set \( G \).
\( \vdash \) Cohen Real: A partial function \( f : \text{nat} \rightarrow_{\text{fin}} \text{bool} \) approximates a total function \( g : \text{nat} \rightarrow \text{bool} \)
Theorem

If $\Gamma \vdash^F M : T$ in the Extended Theory then

$$p : \mathcal{P}, [\Gamma] \vdash [M]_p : [T]_p$$

in the Ground Theory.

- The validity of the new rules/objects is checked in the ground system.

  $\leadsto$ Just check $[M]_p$ is of type $[T]_p$

- No Need to change the typechecker!
Consider a function $f : \text{Type} \to \text{Type}$

For example $\lambda D : \text{Type}. (D \to D)$

Want to compute a fixpoint $\mu(f)$

$f(\mu f) = \mu(f)$
Consider a function $f : \text{Type} \to \text{Type}$

$\lambda D : \text{Type}. (D \to D)$

Want to compute a fixpoint $\mu(f)$

$\mu(f) = f(\mu(f))$

Need some restrictions on $f$

To keep coherence and normalization of the theory!
Guarded rather than restricted

\[ f(\mu(f)) = \mu(f) \]
Guarded rather than restricted

\[ f(\triangleright \mu(f)) = \mu(f) \]

\[ \implies \text{Guarded recursion} \]

\[ \implies \text{Nakano’s } \lambda\text{-calculus} \]

\[ \implies \text{Impose a decrease via translation.} \]
Guarded rather than restricted

- \( f(\triangleright \mu(f)) = \mu(f) \)
  \( \rightsquigarrow \) Guarded recursion
  \( \rightsquigarrow \) Nakano’s \( \lambda \)-calculus
  \( \rightsquigarrow \) Impose a decrease via translation.

- \( [\mu(\lambda D : \text{Type.}(D \to D))]_1 \cong ([D]_0 \to [D]_0) \to ([D]_0 \to [D]_0) \)
Negating the Continuum Hypothesis

- Adapt the proof of Cohen
- Building a set $A$ such that
  - There exists two injections $i_1, i_2$ s.t.
    \[
    \text{nat} \hookrightarrow i_1 A \hookrightarrow i_2 \mathcal{P}(\text{nat})
    \]
  - There is no surjection
    \[
    \text{nat} \twoheadrightarrow A \twoheadrightarrow \mathcal{P}(\text{nat})
    \]

\[\leadsto \mathcal{P}(T)\) is the type $T \to \mathbf{Prop}$.
\[\leadsto A \text{ will be } \widehat{\mathcal{P}(\text{nat})} \text{ where } [\widehat{\mathcal{P}(\text{nat})}]_p \overset{\text{def}}{=} \mathcal{P}(\text{nat}).
\[\leadsto \widehat{\mathcal{P}(\text{nat})} \text{ is different from } \mathcal{P}(\text{nat})!
\]
1 The Calculus of Constructions

2 Definition of the Translation

3 Generalized Inductive Types

4 Future Work
Some intuitions

- A polymorphic dependently typed \(\lambda\)-calculus
  - Archetypal example: \(\text{list}_n\)
  - No syntactic distinction Types/Terms.

- Distinctions between Propositions and Datatypes
  - Using sorts \(\text{Prop}\) and \(\text{Type}\).
  - Proof-terms \(M : P\) with \(P : \text{Prop}\).
  - Dependent products \(\Pi x : T. P\) are also used to model universal quantification.

- Plus Inductive Types (\(\text{nat}, =, \ldots\)).
Definition of the language

- The Sorts \( s := \text{Prop}, \text{Type} \)
- The Terms \( M, N, P, T := x \mid s \mid \lambda x : T. M \mid \Pi x : T. U \mid \Sigma x : T. U \mid \{ x : T \mid P \} \mid MN \langle M, N \rangle \mid \pi_i M \)

\( \rightsquigarrow \) Plus Inductive Types (nat, =, \ldots)

- Conversion rule (congruence):

\[
\begin{align*}
(\lambda x : T. M) N & \equiv M \{ N/x \} \\
\pi_i \langle M_1, M_2 \rangle & \Rightarrow M_i \quad (i = 1, 2)
\end{align*}
\]

\( \rightsquigarrow \) Plus proof-irrelevance.
\[\frac{\text{Var}}{\Gamma} \quad \var{\text{wf}(\Gamma)} \quad (x : T) \in \Gamma \quad \Gamma \vdash x : T}\]

\[\frac{\text{Ax}}{\Gamma} \quad \var{\text{wf}(\Gamma)} \quad (s_1, s_2) \in A \quad \Gamma \vdash s_1 : s_2\]

\[\frac{\text{Conv}}{\Gamma} \quad \Gamma \vdash M : T \quad T \simeq U \quad \Gamma \vdash M : U\]

\[\mathcal{A} \overset{\text{def}}{=} \{ (\text{Prop}, \text{Type}_0), (\text{Type}_i , \text{Type}_{i+1}) \}\]
Abstr \[ \Gamma, x : T \vdash M : U \]
\[ \Gamma \vdash \lambda x : T. M : \Pi x : T. U \]

App \[ \Gamma \vdash M : \Pi x : T. U \quad \Gamma \vdash N : T \]
\[ \Gamma \vdash MN : U \{ N/x \} \]

Prod \[ \Gamma \vdash T : s_1 \quad \Gamma, x : T \vdash U : s_2 \]
\[ \Gamma \vdash \Pi x : T. U : \max(s_1, s_2) \]

- \( \max(s, \text{Prop}) = \text{Prop} \) for any \( s \in S \).
- \( \max(\text{Prop}, \text{Type}_i) = \text{Type}_i \) and
- \( \max(\text{Type}_i, \text{Type}_j) = \text{Type}_{\max(i,j)} \)
Inference Rules (3/3)

\[
\begin{align*}
\text{Sum} & \quad \frac{\Gamma \vdash T : s \quad \Gamma, x : T \vdash U : s}{\Gamma \vdash \Sigma x : T.U : s} \\
\text{Pair} & \quad \frac{\Gamma \vdash M : T \quad \Gamma \vdash N : U \{M/x\}}{\Gamma \vdash (M, N) : \Sigma x : T.U} \\
\text{Proj-1} & \quad \frac{\Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \pi_1 M : T} \\
\text{Proj-2} & \quad \frac{\Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \pi_2 M : U \{\pi_1 M/x\}} \\
\text{Subset} & \quad \frac{\Gamma \vdash T : s \quad \Gamma, x : T \vdash U : \text{Prop}}{\Gamma \vdash \{x : T \mid U\} : s}
\end{align*}
\]

Plus Subtyping for Subset Types!
The Calculus of Constructions

Definition of the Translation

Generalized Inductive Types

Future Work
When \( q \leq p \), \( M : \llbracket T \rrbracket_p \) can be considered as of type \( \llbracket T \rrbracket_q \).
When $q \leq p$, $M : \llbracket T \rrbracket_p$ can be considered as of type $\llbracket T \rrbracket_q$.

- The translation $\llbracket T \rrbracket_p$ must come with its restriction maps $\theta^{T}_{p \rightarrow q} : \llbracket T \rrbracket_p \rightarrow \llbracket T \rrbracket_q$. 

- $\theta^{T}_{p \rightarrow q}$ satisfies some commutative diagrams encoding the presheaf conditions.
When $q \leq p$, $M : \llbracket T \rrbracket_p$ can be considered as of type $\llbracket T \rrbracket_q$

The translation $\llbracket T \rrbracket_p$ must come with its restriction maps $\theta^T_{p \rightarrow q} : \llbracket T \rrbracket_p \rightarrow \llbracket T \rrbracket_q$

$\llbracket T \rrbracket_p$ and the $\theta^T_{p \rightarrow q}$ are defined as the two components of a primitive translation $\llbracket T \rrbracket_p$

$\rightsquigarrow$ Dependent sums

$\rightsquigarrow$ $\llbracket M \rrbracket_p$ and $\theta^M_{p \rightarrow q}$ do not exists for a proofterm $M$. 
General Idea: Internalizing the Presheaf Construction

- When $q \leq p$, $M : \llbracket T \rrbracket_p$ can be considered as of type $\llbracket T \rrbracket_q$

- The translation $\llbracket T \rrbracket_p$ must come with its restriction maps $\theta^T_{p \to q} : \llbracket T \rrbracket_p \to \llbracket T \rrbracket_q$

- $\llbracket T \rrbracket_p$ and the $\theta^T_{p \to q}$ are defined as the two components of a primitive translation $[T]_p$
  - $\rightsquigarrow$ Dependent sums
  - $\rightsquigarrow$ $\llbracket M \rrbracket_p$ and $\theta^M_{p \to q}$ do not exists for a proofterm $M$.

- $\theta^T_{p \to q}$ satisfies some commutative diagrams
  - $\rightsquigarrow$ Presheaf conditions
  - $\rightsquigarrow$ Encoded using subset types
Translation of Sorts

\[ P : [\text{Prop}]_p \text{ must be :} \]
- A function \( f : \mathcal{P}_p \to \text{Prop} \)
- with Restriction maps \( \theta : \Pi q : \mathcal{P}_p. \Pi r : \mathcal{P}_q. f_q \to f_r \)
Translation of Sorts

\[ P : \left[ \text{Prop} \right]_p \text{ must be :} \]

- A function \( f : \mathcal{P}_p \rightarrow \text{Prop} \)
- with Restriction maps \( \theta : \Pi q : \mathcal{P}_p. \Pi r : \mathcal{P}_q. f q \rightarrow f r \)

So \( \left[ \text{Prop} \right]_p \) is defined as :

\[ \Sigma f : \mathcal{P}_p \rightarrow \text{Prop}. \Pi q : \mathcal{P}_p. \Pi r : \mathcal{P}_q. f q \rightarrow f r \]

\( \rightsquigarrow \) Same for Type, with extra Proofs of transitivity and reflexivity of \( \theta \)
Translation of the Dependent Product

- $[\Pi x : T . U]_p^\sigma$ is defined as

$$\{ f : \Pi q : P_p \Pi x : [T]_q^\sigma . [U]_q^{\sigma+}(x, T, q) \mid \text{comm}_\Pi(f, T, U, p) \}$$

- $\leadsto$ Like $p \vdash P \Rightarrow Q$ is usually defined as $\forall q \leq p. (q \vdash P) \Rightarrow (q \vdash Q)$.

- $\text{comm}_\Pi(f, T, U, p)$ enforces $f$ to satisfy

$$[T]_p^\sigma \xrightarrow{f_p} [U]_p^\sigma \ i$$

$$\theta_p^{\sigma,T} \downarrow \quad \downarrow \theta_p^{\sigma,U}$$

$$[T]_q^\sigma \xrightarrow{f_q} [U]_q^\sigma$$
[Γ] cannot translate each type at the same forcing condition $p$. 
Translation of Contexts and Variables

- $\llbracket \Gamma \rrbracket$ cannot translate each type at the same forcing condition $p$.

- So $[x]_q^\sigma$ is not translated as $x$ but as $\theta_{\sigma_1(x)}^{\sigma,\sigma_2(x)\rightarrow q} x$

$$x : \llbracket T \rrbracket_p^\sigma \vdash \theta_{\sigma_1(x)}^{\sigma,\sigma_2(x)\rightarrow q} x : \llbracket T \rrbracket_q^\sigma$$

$\llbracket \cdot \rrbracket$ encodes the type and the forcing condition at which $x$ has been introduced in $\Gamma$.

$\llbracket \cdot \rrbracket$ $\sigma_1(x) = T$ and $\sigma_2(x) = p$.

$\llbracket \cdot \rrbracket$ Every translation $[M]_p^\sigma$ is indexed by this environment $\sigma$. 
[\lambda x : T. M]_p^\sigma is of type \Pi q : \mathcal{P}_p \Pi x : [T]_q^\sigma \cdot [U]_q^{\sigma + (x, T, q)}

\Rightarrow Plus a proof of commutation.

\Rightarrow [\lambda x : T. M]_p^\sigma is a collection of maps [T]_q^\sigma \cdot [U]_q^{\sigma + (x, T, q)} for every q : \mathcal{P}_p
Translation of Abstraction and Applications

- $[\lambda x : T.M]_p^\sigma$ is of type $\Pi q : \mathcal{P}_p \Pi x : [T]_q^\sigma \cdot [U]_q^\sigma + \langle x, T,q \rangle$

  $\rightsquigarrow$ Plus a proof of commutation.

  $\rightsquigarrow [\lambda x : T.M]_p^\sigma$ is a collection of maps $[T]_q^\sigma \cdot [U]_q^\sigma + \langle x, T,q \rangle$ for every $q : \mathcal{P}_p$

- $[\lambda x : T.M]_p^\sigma$ is defined as

  \[
  \lambda q : \mathcal{P}_p. \lambda x : [T]_q^\sigma \cdot [M]_q^\sigma + \langle x, T,q \rangle
  \]
Translation of Abstraction and Applications

- \([\lambda x : T.M]_p^\sigma\) is of type \(\Pi q : P_p \Pi x : [T]_q^\sigma \cdot [U]_q^\sigma+(x,T,q)\)
  \(\leadsto\) Plus a proof of commutation.
  \(\leadsto\) \([\lambda x : T.M]_p^\sigma\) is a collection of maps \([T]_q^\sigma \cdot [U]_q^\sigma+(x,T,q)\) for every \(q : P_p\)

- \([\lambda x : T.M]_p^\sigma\) is defined as
  \[\lambda q : P_p . \lambda x : [T]_q^\sigma \cdot [M]_q^\sigma+(x,T,q)\]

- \([MN]_p^\sigma\) is defined as
  \([M]_p^\sigma \ p \ [N]_p^\sigma\).
**Correctness**

**Theorem**

If $\Gamma \vdash M : T$ and $\sigma$ is a valid interpretation of $\Gamma$ then

$$\llbracket \Gamma \rrbracket^\sigma \vdash \llbracket M \rrbracket^\sigma_p : \llbracket T \rrbracket^\sigma_p$$

where $p$ is the last forcing condition appearing in $\sigma$.

⇝ Need to extend the proof of this theorem when extending the translation !
What about the Excluded Middle?

- How to import Axioms in the forcing layer?
  ↞ Not systematically possible!
- An example: the Excluded-Middle $\text{EM}$

$$\Pi P : \text{Prop}. P \lor (P \to \text{false})$$
What about the Excluded Middle?

- How to import Axioms in the forcing layer?
  → Not systematically possible!
- An example: the Excluded-Middle $\textbf{EM}$

$$\Pi P : \text{Prop}. P \lor (P \rightarrow \text{false})$$

- $[\textbf{EM}]_p$ is equal to

$$\Pi q : \mathcal{P}_p. \Pi P : (\mathcal{P}_q \rightarrow \text{Prop}). Pp \lor (\Pi r : \mathcal{P}_q. Pr \rightarrow \text{false})$$
What about the Excluded Middle?

- How to import Axioms in the forcing layer?
  - Not systematically possible!

- An example: the Excluded-Middle $\text{EM}$

\[
\Pi P : \text{Prop}. P \lor (P \rightarrow \text{false})
\]

- $[\text{EM}]_{\sigma}^p$ is equal to

\[
\Pi q : \mathcal{P}_p . \Pi P : (\mathcal{P}_q \rightarrow \text{Prop}). Pp \lor (\Pi r : \mathcal{P}_q . Pr \rightarrow \text{false})
\]

- $[\text{EM}]_{\sigma}^p$ can be false
  - Adapt the usual Kripke model

- Need to use sheaf translation rather than presheaves to keep it.
  - With dense topology on $\mathcal{P}$
  - Constant presheaves are not sheaves!
1. The Calculus of Constructions

2. Definition of the Translation

3. Generalized Inductive Types

4. Future Work
In Coq, Inductive Types must satisfy the *positivity* criteria
\[\text{Inductive } T := C : (T \rightarrow \text{nat}) \rightarrow T\]
is forbidden.

We allow them in the forcing layer.

Forcing conditions will be \text{nat}.
\[\text{Well-foundation of the forcing conditions!}\]

The nth approximation of an inductive type will be its nth unwinding.
- See a forcing condition as time.
  ⟷ Need to travel in time!

- Introduce a modality later: $\triangleright_s$

  ⟷ $\triangleright_{\text{Prop}} P$ means that $P$ will be “true” in the future

  ⟷ Löb rule.
See a forcing condition as time.
\[ \leadsto \text{Need to travel in time!} \]

Introduce a modality later: \( \triangleright_s \)
\[ \leadsto \triangleright_{\text{Prop}} P \text{ means that } P \text{ will be “true” in the future} \]
\[ \leadsto \text{Löb rule.} \]

**Definition**

- \( [\triangleright_s T]_0^\sigma \overset{\text{def}}{=} \text{Unit}_s \)
- \( [\triangleright_s T]_{S_n}^\sigma \overset{\text{def}}{=} [T]_n^\sigma \)

\( \triangleright_s \) will guard the recursion.
Computing fixpoints

- A guarded fixpoint combinator \( \text{fix}_T : (\triangledown_s T \rightarrow T) \rightarrow T \)

### Definition

- \([\text{fix}_T f]^0_\sigma \overset{\text{def}}{=} f(\text{unit}_s)\)
- \([\text{fix}_T f]^\sigma_n \overset{\text{def}}{=} f([\text{fix}_T f]^\sigma_n)\)
Computing fixpoints

- A guarded fixpoint combinator $\text{fix}_T : (\rhd_s T \to T) \to T$

**Definition**

- $[\text{fix}_T f]^0_\sigma \overset{\text{def}}{=} f(\text{unit}_s)$
- $[\text{fix}_T f]^n_\sigma \overset{\text{def}}{=} f([\text{fix}_T f]^n)\sigma$

- $\mu_s : (s \to s) \to s$ is built using $\text{fix}_s$.

\[
\mu_s f = f(\rhd_s \mu_s f)
\]
Computing fixpoints

- A guarded fixpoint combinator \( \text{fix}_T : (\triangleright_s T \to T) \to T \)

**Definition**

- \([\text{fix}_T f]^0_\sigma \overset{\text{def}}{=} f(\text{unit}_s)\)
- \([\text{fix}_T f]^n_\sigma \overset{\text{def}}{=} f([\text{fix}_T f]^n_\sigma)\)

- \(\mu_s : (s \to s) \to s\) is built using \(\text{fix}_s\).

\[\mu_s f = f(\triangleright_s \mu_s f)\]

- For a proposition \(P\), \(\text{fix}_P M\) computes a proof of \(P\) from a proof \(M\) of \(\triangleright_{\text{Prop}} P \to P\)

\(\rightsquigarrow\) Computational content of the Löb Rule.
Future Works

- Negative translation for Prop and Type
  \( \Rightarrow \) Using Sheafification.

- Links with the work of Krivine and Miquel?
  \( \Rightarrow \) Formulas and proofterms are translated uniformly here.

- A Prototype extending Coq is in development.
  \( \Rightarrow \) Build on top of Coq
  \( \Rightarrow \) Use the typechecker of Coq

- Formalize usual Forcing proof in Coq