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

Some Background

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

Some Background

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

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

Some Background

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

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

- Krivine Realizability and Forcing
  $\rightsquigarrow$ Realizability as "non-commutative forcing".
  $\rightsquigarrow$ *Forcing as program transformation* of Miquel.

Some impossibilities in Coq

- General inductive types:
  `Inductive D := Lambda : (D -> D) -> D.`
  $\leadsto$ `Error: Non strictly positive occurrence of "D" in "(D -> D) -> D".`

Some impossibilities in Coq

- General inductive types:

  `Inductive D := Lambda : (D -> D) -> D.`

  ⇝ `Error: Non strictly positive occurrence of "D" in "(D -> D) -> D".`

- Computational content for Axioms:

```
Axiom UltrafilterNat : exists mu:(nat -> Prop) -> Prop,
       forall f:(nat -> Prop), (mu(f) \/ mu(1-f)) /\ ...
Theorem Ramsey : ...
Proof. ... pose UltrafilterNat. ... Qed.
Print Ramsey.
```

  ⇝ `UltrafilterNat` appears as a black box in the proofterm of `Ramsey`.

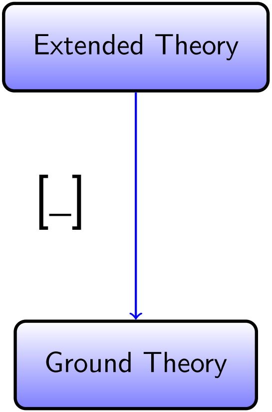
Extending Type Theory

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

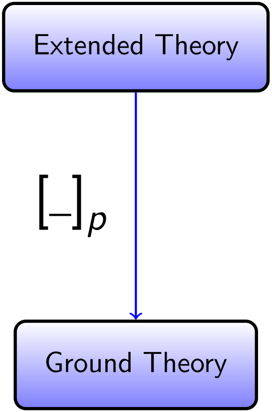
- Without using Axioms
  $\leadsto$ No problem of coherence
  $\leadsto$ Give them a computational content.

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

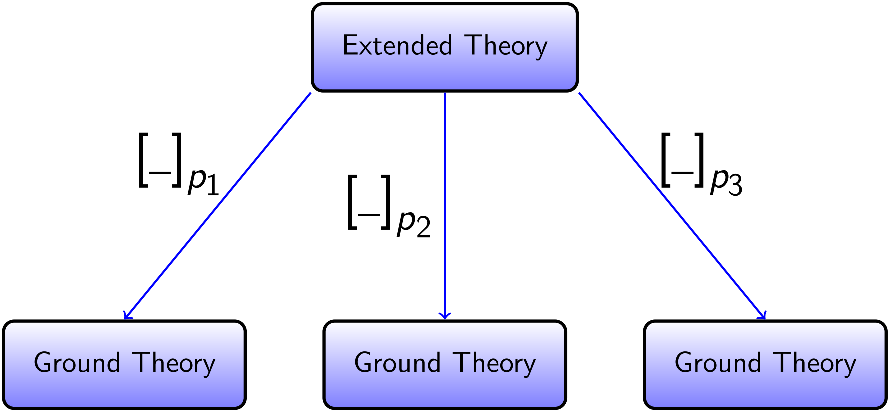
Transformation



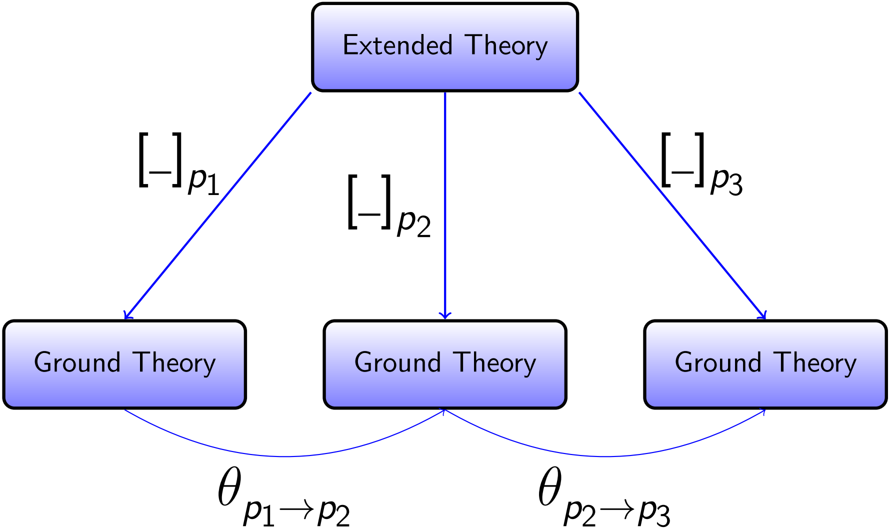
Transformation



Transformation



Transformation



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 $[\![T]\!]_p$ with restriction functions $\theta^T_{p \to q} : [\![T]\!]_p \to [\![T]\!]_q$ for $q \leq p$.
- $[\![T]\!]_p$ and the $\theta^T_{p \to q}$ are defined as the two components of the primitive translation $[T]_p$.

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 $\theta^T_{p\to q} : [\![T]\!]_p \to [\![T]\!]_q$ for $q \leq p$.
- $[\![T]\!]_p$ and the $\theta^T_{p\to q}$ 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.
  $\rightsquigarrow$ $p \leq q$ means that p contains more information than q.

  $\rightsquigarrow$ $\mathcal{P}_p \overset{def}{=} \{q \in \mathcal{P} \mid q \leq p\}$

Forcing Conditions

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

  $\leadsto$ $\mathcal{P}_p \stackrel{def}{=} \{q \in \mathcal{P} \mid q \leq p\}$
- Examples :
  $\leadsto$ Step-indexing :
  - Divergence is approximated as *reducing at least n times*
  - $n \in \mathbb{N}$ approximates ω.

  $\leadsto$ Cohen Forcing : p approximate the generic set G.
  $\leadsto$ Cohen Real : A partial function $f : \mathrm{nat} \to_{fin} \mathrm{bool}$ approximates a total function $g : \mathrm{nat} \to \mathrm{bool}$

An Extension

Theorem

If $\Gamma \vdash_{\mathcal{F}} M : T$ *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.

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

- No Need to change the typechecker !

What about Inductive Types ?

- Consider a function $f : \mathrm{Type} \to \mathrm{Type}$
  $\leadsto$ For example $\lambda D : \mathrm{Type}.(D \to D)$

- Want to compute a fixpoint $\mu(f)$
  $\leadsto$ $f(\mu f) = \mu(f)$

What about Inductive Types ?

- Consider a function $f : \mathrm{Type} \to \mathrm{Type}$
  $\leadsto$ For example $\lambda D : \mathrm{Type}.(D \to D)$

- Want to compute a fixpoint $\mu(f)$
  $\leadsto$ $f(\mu f) = \mu(f)$

- Need some restrictions on f
  $\leadsto$ To keep coherence and normalization of the theory !

Guarded rather than restricted

- $f(\quad \mu(f)) = \mu(f)$

Guarded rather than restricted

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

Guarded rather than restricted

- $f(\triangleright \mu(f)) = \mu(f)$
  $\leadsto$ Guarded recursion
  $\leadsto$ Nakano's λ-calculus
  $\leadsto$ Impose a decrease via translation.

- $[\![\mu(\lambda D : \mathrm{Type}.(D \to D)]\!]_1 \simeq ([\![D]\!]_0 \to [\![D]\!]_0) \to ([\![D]\!]_0 \to [\![D]\!]_0)$

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.

$$\mathrm{nat} \hookrightarrow^{i_1} A \hookrightarrow^{i_2} \mathbf{P}(\mathrm{nat})$$

  - There is no surjection

$$\mathrm{nat} \rightarrowtail A \rightarrowtail \mathbf{P}(\mathrm{nat})$$

$\rightsquigarrow$ $\mathbf{P}(T)$ is the type $T \to \mathrm{Prop}$.

$\rightsquigarrow$ A will be $\widehat{\mathbf{P}(\mathrm{nat})}$ where $[\![\widehat{\mathbf{P}(\mathrm{nat})}]\!]_p \stackrel{def}{=} \mathbf{P}(\mathrm{nat})$.

$\rightsquigarrow$ $\widehat{\mathbf{P}(\mathrm{nat})}$ is different from $\mathbf{P}(\mathrm{nat})$!

Some intuitions

- A polymorphic dependently typed λ-calculus
  $\rightsquigarrow$ Archetypal example : list_n
  $\rightsquigarrow$ No syntactic distinction Types/Terms.

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

- Plus Inductive Types $(\mathrm{nat}, =, \ldots)$.

Definition of the language

- The Sorts $s := \mathrm{Prop}, \mathrm{Type}_i$
- 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 $(\mathrm{nat}, =, \ldots)$
- Conversion rule (congruence) :

$$\begin{array}{rcl} (\lambda x : T.M)N & \cong & M\{N/x\} \\ \pi_i \langle M_1, M_2 \rangle & \cong & M_i \quad (\mathrm{i} = 1, 2) \end{array}$$

$\rightsquigarrow$ Plus proof-irrelevance.

Inference Rules (1/3)

$$\text{Var}\ \frac{\mathbf{wf}(\Gamma) \qquad (x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \text{Ax}\ \frac{\mathbf{wf}(\Gamma) \qquad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2}$$

$$\text{Conv}\ \frac{\Gamma \vdash M : T \qquad T \simeq U}{\Gamma \vdash M : U}$$

$$\mathcal{A} \stackrel{def}{=} \{(\mathrm{Prop}, \mathrm{Type}_0), (\mathrm{Type}_i, \mathrm{Type}_{i+1})\}$$

Inference Rules (2/3)

$$\text{Abstr}\ \frac{\Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T.M : \Pi x : T.U} \qquad \text{App}\ \frac{\Gamma \vdash M : \Pi x : T.U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U\{N/x\}}$$

$$\text{Prod}\ \frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash \Pi x : T.U : max(s_1, s_2)}$$

- $\text{max}(s, \mathrm{Prop}) = \mathrm{Prop}$ for any $s \in \mathcal{S}$.
- $\text{max}(\mathrm{Prop}, \mathrm{Type}_i) = \mathrm{Type}_i$ and
- $\text{max}(\mathrm{Type}_i, \mathrm{Type}_j) = \mathrm{Type}_{\text{max}(i,j)}$

Inference Rules (3/3)

$$\text{Sum}\ \frac{\Gamma \vdash T : s \qquad \Gamma, x : T \vdash U : s}{\Gamma \vdash \Sigma x : T.U : s} \quad \text{Pair}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : U\{M/x\}}{\Gamma \vdash (M, N) : \Sigma x : T.U}$$

$$\text{Proj-1}\ \frac{\Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \pi_1 M : T} \qquad \text{Proj-2}\ \frac{\Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \pi_2 M : U\{\pi_1 M/x\}}$$

$$\text{Subset}\ \frac{\Gamma \vdash T : s \qquad \Gamma, x : T \vdash U : \mathrm{Prop}}{\Gamma \vdash \{x : T \mid U\} : s}$$

Plus Subtyping for Subset Types !

1. The Calculus of Constructions

2. Definition of the Translation

3. Generalized Inductive Types

4. Future Work

General Idea : Internalizing the Presheaf Construction

- When $q \leq p$, $M : [\![T]\!]_p$ can be considered as of type $[\![T]\!]_q$

General Idea : Internalizing the Presheaf Construction

- When $q \leq p$, $M : [\![T]\!]_p$ can be considered as of type $[\![T]\!]_q$
- The translation $[\![T]\!]_p$ must come with its restriction maps $\theta^T_{p \to q} : [\![T]\!]_p \to [\![T]\!]_q$

General Idea : Internalizing the Presheaf Construction

- When $q \leq p$, $M : [\![T]\!]_p$ can be considered as of type $[\![T]\!]_q$
- The translation $[\![T]\!]_p$ must come with its restriction maps $\theta^T_{p \to q} : [\![T]\!]_p \to [\![T]\!]_q$
- $[\![T]\!]_p$ and the $\theta^T_{p \to q}$ are defined as the two components of a primitive translation $[T]_p$
  $\rightsquigarrow$ Dependent sums
  $\rightsquigarrow$ $[\![M]\!]_p$ and $\theta^M_{p \to q}$ do not exists for a proofterm M.

General Idea : Internalizing the Presheaf Construction

- When $q \leq p$, $M : [\![T]\!]_p$ can be considered as of type $[\![T]\!]_q$
- The translation $[\![T]\!]_p$ must come with its restriction maps $\theta^T_{p\to q} : [\![T]\!]_p \to [\![T]\!]_q$
- $[\![T]\!]_p$ and the $\theta^T_{p\to q}$ are defined as the two components of a primitive translation $[T]_p$
  $\rightsquigarrow$ Dependent sums
  $\rightsquigarrow$ $[\![M]\!]_p$ and $\theta^M_{p\to q}$ do not exists for a proofterm M.
- $\theta_{p\to q}$ satisfies some commutative diagrams
  $\rightsquigarrow$ Presheaf conditions
  $\rightsquigarrow$ Encoded using subset types

Translation of Sorts

$P : [\![\mathrm{Prop}]\!]_p$ must be :

- A function $f : \mathcal{P}_p \to \mathrm{Prop}$
- with Restriction maps $\theta : \Pi q : \mathcal{P}_p . \Pi r : \mathcal{P}_q . fq \to fr$

Translation of Sorts

$P : [\![\mathrm{Prop}]\!]_p$ must be :

- A function $f : \mathcal{P}_p \to \mathrm{Prop}$
- with Restriction maps $\theta : \Pi q : \mathcal{P}_p . \Pi r : \mathcal{P}_q . fq \to fr$

So $[\![\mathrm{Prop}]\!]_p$ is defined as :

$$\Sigma f : \mathcal{P}_p \to \mathrm{Prop}. \Pi q : \mathcal{P}_p . \Pi r : \mathcal{P}_q . fq \to fr$$

$\rightsquigarrow$ Same for Type, with extra Proofs of transitivity and reflexivity of θ

Translation of the Dependent Product

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

$$\left\{ f : \Pi q : \mathcal{P}_p \Pi x : [\![T]\!]_q^\sigma . [\![U]\!]_q^{\sigma + (x,T,q)} \mid \mathbf{comm}_\Pi(f, T, U, p) \right\}$$

$\rightsquigarrow$ Like $p \Vdash P \Rightarrow Q$ is usually defined as $\forall q \leq p.(q \Vdash P) \Rightarrow (q \Vdash Q)$.

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

$$\begin{tikzcd}
[\![T]\!]_p^\sigma \arrow[r, "f_p"] \arrow[d, "\theta_{p \to q}^{\sigma, T}"'] & [\![U]\!]_p^\sigma i \arrow[d, "\theta_{p \to q}^{\sigma, U}"] \\
[\![T]\!]_q^\sigma \arrow[r, "f_q"'] & [\![U]\!]_q^\sigma
\end{tikzcd}$$

Translation of Contexts and Variables

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

Translation of Contexts and Variables

- $[\![\Gamma]\!]$ cannot translate each type at the same forcing condition p.

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

$$x : [\![T]\!]_p^\sigma \vdash \theta_{p\to q}^{\sigma,T} x : [\![T]\!]_q^\sigma$$

$\rightsquigarrow$ σ encodes the type and the forcing condition at which x has been introduced in Γ.
$\rightsquigarrow$ $\sigma_1(x) = T$ and $\sigma_2(x) = p$.
$\rightsquigarrow$ Every translation $[M]_p^\sigma$ is indexed by this environment σ.

Translation of Abstraction and Applications

- $[\lambda x : T.M]_p^\sigma$ is of type $\Pi q : \mathcal{P}_p \Pi x : [\![T]\!]_q^\sigma . [\![U]\!]_q^{\sigma+(x,T,q)}$
  $\rightsquigarrow$ Plus a proof of commutation.
  $\rightsquigarrow$ $[\lambda x : T.M]_p^\sigma$ is a collection of maps $[\![T]\!]_q^\sigma . [\![U]\!]_q^{\sigma+(x,T,q)}$ for every $q : \mathcal{P}_p$

Translation of Abstraction and Applications

- $[\lambda x : T.M]_p^\sigma$ is of type $\Pi q : \mathcal{P}_p \Pi x : [\![T]\!]_q^\sigma . [\![U]\!]_q^{\sigma+(x,T,q)}$
  $\rightsquigarrow$ Plus a proof of commutation.
  $\rightsquigarrow$ $[\lambda x : T.M]_p^\sigma$ is a collection of maps $[\![T]\!]_q^\sigma . [\![U]\!]_q^{\sigma+(x,T,q)}$ for every $q : \mathcal{P}_p$

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

$$\lambda q : \mathcal{P}_p . \lambda x : [\![T]\!]_q^\sigma . [M]_q^{\sigma+(x,T,q)}$$

Translation of Abstraction and Applications

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

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

$$\lambda q : \mathcal{P}_p . \lambda x : [\![T]\!]_q^\sigma . [M]_q^{\sigma + (x,T,q)}$$

- $[MN]_p^\sigma$ is defined as

$$[M]_p^\sigma \ p \ [N]_p^\sigma .$$

Correctness

Theorem

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

$$[\![\Gamma]\!]^{\sigma} \vdash [M]^{\sigma}_{p} : [\![T]\!]^{\sigma}_{p}$$

where p is the last forcing condition appearing in σ.

$\rightsquigarrow$ 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 ?
  $\rightsquigarrow$ Not systematically possible !
- An example : the Excluded-Middle **EM**

$$\Pi P : \mathrm{Prop}. P \vee (P \rightarrow \mathrm{false})$$

What about the Excluded Middle ?

- How to import Axioms in the forcing layer ?
  $\rightsquigarrow$ Not systematically possible !
- An example : the Excluded-Middle **EM**

$$\Pi P : \mathrm{Prop}.P \vee (P \rightarrow \mathrm{false})$$

- $[\mathbf{EM}]_p^\sigma$ is equal to

$$\Pi_q : \mathcal{P}_p.\Pi P : (\mathcal{P}_q \rightarrow \mathrm{Prop}).Pp \vee (\Pi r : \mathcal{P}_q.Pr \rightarrow \mathrm{false})$$

What about the Excluded Middle ?

- How to import Axioms in the forcing layer ?
  $\rightsquigarrow$ Not systematically possible !
- An example : the Excluded-Middle **EM**

$$\Pi P : \mathrm{Prop}.P \vee (P \to \mathrm{false})$$

- $[\mathbf{EM}]_p^\sigma$ is equal to

$$\Pi_q : \mathcal{P}_p.\Pi P : (\mathcal{P}_q \to \mathrm{Prop}).Pp \vee (\Pi r : \mathcal{P}_q.Pr \to \mathrm{false})$$

- $[\mathbf{EM}]_p^\sigma$ can be false
  $\rightsquigarrow$ Adapt the usual Kripke model
- Need to use sheaf translation rather than presheaves to keep it.
  $\rightsquigarrow$ With dense topology on $\mathcal{P}$
  $\rightsquigarrow$ Constant presheaves are not sheaves !

Which generalization ?

- In Coq, Inductive Types must satisfy the *positivity* criteria
  $\leadsto$ `Inductive T := C : (T ->nat) -> T`
  is forbidden.

- We allow them in the forcing layer.

- Forcing conditions will be nat.
  $\leadsto$ Well-foundation of the forcing conditions !

- The nth approximation of an inductive type will be its nth unwinding.

Back to the Future

- See a forcing condition as time.
  $\rightsquigarrow$ Need to travel in time !
- Introduce a modality later : $\triangleright_s$
  $\rightsquigarrow$ $\triangleright_{\mathrm{Prop}} P$ means that P will be "true" *in the future*
  $\rightsquigarrow$ Löb rule.

Back to the Future

- See a forcing condition as time.
  $\rightsquigarrow$ Need to travel in time !
- Introduce a modality later : $\triangleright_s$
  $\rightsquigarrow$ $\triangleright_{\mathrm{Prop}} P$ means that P will be "true" *in the future*
  $\rightsquigarrow$ Löb rule.

Definition

- $[\triangleright_s T]_0^\sigma \stackrel{def}{=} \mathrm{Unit}_s$
- $[\triangleright_s T]_{\mathsf{S}n}^\sigma \stackrel{def}{=} [T]_n^\sigma$

- $\triangleright_s$ will guard the recursion.

Computing fixpoints

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

Definition

- $[\mathbf{fix}_T f]^\sigma_0 \stackrel{def}{=} f(\mathrm{unit}_s)$
- $[\mathbf{fix}_T f]^\sigma_{\mathsf{S}n} \stackrel{def}{=} f([\mathbf{fix}_T f]^\sigma_n)$

Computing fixpoints

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

Definition

- $[\mathbf{fix}_T f]^\sigma_0 \stackrel{def}{=} f(\mathrm{unit}_s)$
- $[\mathbf{fix}_T f]^\sigma_{\mathsf{S}n} \stackrel{def}{=} f([\mathbf{fix}_T f]^\sigma_n)$

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

$$\mu_s f = f(\triangleright_s \mu_s f)$$

Computing fixpoints

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

Definition

- $[\mathbf{fix}_T f]^\sigma_0 \stackrel{def}{=} f(\mathrm{unit}_s)$
- $[\mathbf{fix}_T f]^\sigma_{\mathsf{S}n} \stackrel{def}{=} f([\mathbf{fix}_T f]^\sigma_n)$

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

$$\mu_s f = f(\triangleright_s \mu_s f)$$

- For a proposition P, $\mathbf{fix}_P M$ computes a proof of P from a proof M of $\triangleright_{\mathrm{Prop}} P \to P$
  $\rightsquigarrow$ Computational content of the Löb Rule.

Future Works

- Negative translation for Prop and Type
  $\rightsquigarrow$ Using Sheafification.

- Links with the work of Krivine and Miquel ?
  $\rightsquigarrow$ Formulas and proofterms are translated uniformly here.

- A Prototype extending Coq is in development.
  $\rightsquigarrow$ Build on top of Coq
  $\rightsquigarrow$ Use the typechecker of Coq

- Formalize usual Forcing proof in Coq