Forcing Translations in Type Theory

M. Sozeau, Inria Paris & IRIF

jww. G. Jaber, G. Lewertowski, P.-M. Pédrot & N. Tabareau

Categorical Logic & Univalent Foundations
Leeds, UK
July 28th 2016
Forcing in a Nutshell

- Historically, forcing is a model transformation
- Several names for the same concept

**Forcing** translation \(\cong\) **Kripke** models \(\cong\) **Presheaf** construction

\((Set\ theory)\) \hspace{1cm} \((Modal\ logic)\) \hspace{1cm} \((Category\ theory)\)

- Usually, set-theoretic forcing is classical
- We will study intuitionistic forcing, in intuitionistic type theory
Why use forcing?
Forcing

Why use forcing?

- Set theory: a lot of independence results continuum hypothesis, AC, …
- Modal logic and Kripke Models
Forcing

Why use forcing?

- Set theory: a lot of independence results continuum hypothesis, AC, ...  
- Modal logic and Kripke Models  
- Category theory: a HoTT topic!  
  - Many models arise from presheaf constructions  
  - Coquand & al.’s cubical model of univalence is an example  
  - Also step-indexing, parametricity...  
  - But this targets sets or topoi usually  

We want forcing in Type Theory!
Assume a preorder \((\mathbb{P}, \leq)\). We summarize the forcing translation in \(\textbf{LJ}\).

- To a formula \(A\), we associate a \(\mathbb{P}\)-indexed formula \([A]_p\).
- To a proof \(\vdash A\), we associate a proof of \(\forall p : \mathbb{P}, [A]_p\).
- (Target theory not really specified here, think \(\lambda\Pi\).)

\(<\mathbb{P} \text{ are possible worlds, } [A]_p \text{ is truth at world } p >\)
Assume a preorder \((\mathbb{P}, \leq)\). We summarize the forcing translation in \(\textbf{LJ}\).

- To a formula \(A\), we associate a \(\mathbb{P}\)-indexed formula \(\llbracket A \rrbracket_p\).
- To a proof \(\vdash A\), we associate a proof of \(\forall p : \mathbb{P}, \llbracket A \rrbracket_p\).
- (Target theory not really specified here, think \(\lambda \Pi\).)

\(\llbracket A \rightarrow B \rrbracket_p := \forall q \leq p. \llbracket A \rrbracket_q \rightarrow \llbracket B \rrbracket_q\)

\(\llbracket A \rrbracket_p\) is truth at world \(p\)
Assume a preorder \((\mathbb{P}, \leq)\). We summarize the forcing translation in \(\text{LJ}\).

- To a formula \(A\), we associate a \(\mathbb{P}\)-indexed formula \(\llbracket A \rrbracket_p\).
- To a proof \(\vdash A\), we associate a proof of \(\forall p : \mathbb{P}, [A]_p\).
- (Target theory not really specified here, think \(\lambda \Pi\).)

\(<\mathbb{P}\) are possible worlds, \(\llbracket A \rrbracket_p\) is truth at world \(p\)>

Most notably,

\[
\llbracket A \rightarrow B \rrbracket_p := \forall q \leq p. [A]_q \rightarrow [B]_q
\]

Actually this can be adapted straightforwardly to any category \((\mathbb{P}, \text{Hom})\).
The previous soundness theorem also makes sense in a *proof-relevant* world:

\[
\text{If } \vdash t : A \text{ then } p : \mathbb{P} \vdash [t]_p : [A]_p
\]
The previous soundness theorem also makes sense in a 
proof-relevant world:

$$\text{If } \vdash t : A \text{ then } p : \mathbb{P} \vdash [t]_p : [A]_p$$

... and the translation can be thought of as a monotonous monad reader

<table>
<thead>
<tr>
<th>Reader</th>
<th>Forcing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T \ A := \mathbb{P} \rightarrow A$</td>
<td>$T_p \ A := \forall q : \mathbb{P}, q \leq p \rightarrow A$</td>
</tr>
<tr>
<td>read : $1 \rightarrow \mathbb{P}$</td>
<td>read : $1 \rightarrow \mathbb{P}$</td>
</tr>
<tr>
<td>enter : $(1 \rightarrow A) \rightarrow \mathbb{P} \rightarrow A$</td>
<td>enter : $(1 \rightarrow A) \rightarrow \forall p : \mathbb{P}, p \leq \text{read}() \rightarrow A$</td>
</tr>
</tbody>
</table>
The previous soundness theorem also makes sense in a proof-relevant world:

\[
\text{If } \vdash t : A \text{ then } p : \mathbb{P} \vdash [t]_p : [[A]]_p
\]

... and the translation can be thought of as a monotonous monad reader

<table>
<thead>
<tr>
<th>Reader</th>
<th>Forcing</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T \ A := \mathbb{P} \to A )</td>
<td>( T_p \ A := \forall q : \mathbb{P}, q \leq p \to A )</td>
</tr>
<tr>
<td>read : ( 1 \to \mathbb{P} )</td>
<td>read : ( 1 \to \mathbb{P} )</td>
</tr>
<tr>
<td>enter : ( (1 \to A) \to \mathbb{P} \to A )</td>
<td>enter : ( (1 \to A) \to \forall p : \mathbb{P}, p \leq \text{read()} \to A )</td>
</tr>
</tbody>
</table>

In particular, taking \((\mathbb{P}, \leq)\) to be a full preorder gives the reader monad.
Idea of the proof and use

- Substitution lemma for the interpretation.
- “Computational soundness”: \( t \to^\beta u \Rightarrow [t] \equiv^\beta [u] \)
Idea of the proof and use

- Substitution lemma for the interpretation.
- “Computational soundness”: \( t \rightarrow^\beta u \Rightarrow [t] \equiv^\beta [u] \)

One can add “generic” elements in the forcing layer by inhabiting their translations:

\[
\paragraph{F} a : \psi] \triangleq a^\bullet : \forall p : \mathbb{P}, [\psi]_p
\]

Thanks to soundness of the translation, and (assumed) consistency of the source system, as soon as \( \mathbb{P} \) is inhabited:

\[
\paragraph{F} t : \bot \Rightarrow p : \mathbb{P} \vdash [t]_p : [\bot]_p \equiv \Pi q \leq p. \bot
\]

We have equiconsistency.
In 2012, we gave a forcing translation from $\mathbb{CC}_\omega + \Sigma$ into itself.
In 2012, we gave a forcing translation from $\text{CC}_\omega + \Sigma$ into itself.

Intuitively, not that difficult.

- To a type $A$ associate $p : P \vdash \llbracket A \rrbracket_p : \Box$.
- To a term $t : A$ associate $p : P \vdash [t]_p : \llbracket A \rrbracket_p$ by induction on $t$.
In 2012, we gave a forcing translation from $\text{CC}_\omega + \Sigma$ into itself.

Intuitively, not that difficult.

- To a type $\vdash A : \Box$ associate $p : \mathbb{P} \vdash [A]_p : \Box$.
- To a term $\vdash t : A$ associate $p : \mathbb{P} \vdash [t]_p : [A]_p$ by induction on $t$.
- To handle types-as-terms uniformly, $[\cdot]$ is defined through $[\cdot]$

\[
[A]_p : (\Pi q \leq p \to \Box). \quad (A \text{ type})
\]

\[
[\!\! A \!\!]_p := [A]_p \ p \ \text{id}_p
\]

- Translation of the dependent arrow is almost the same:

\[
[\Pi x : A. B]_p \equiv \Pi q \leq p. \Pi x : [A]_q. [B]_q
\]
Do it, or do not: there is no try

In 2012, we gave a forcing translation from $\text{CC}_\omega + \Sigma$ into itself.

Intuitively, not that difficult.

- To a type $\vdash A : \Box$ associate $p : \mathbb{P} \vdash [A]_p : \Box$.
- To a term $\vdash t : A$ associate $p : \mathbb{P} \vdash [t]_p : [A]_p$ by induction on $t$.
- To handle types-as-terms uniformly, $[\cdot]$ is defined through $[\cdot]$

$$[A]_p : (\Pi q \leq p \rightarrow \Box). \quad (A \text{ type})$$

$$[A]_p := [A]_p p \ \text{id}_p$$

- Translation of the dependent arrow is almost the same:

$$[[\Pi x : A. B]]_p \equiv \Pi q \leq p. \Pi x : [A]_q. [B]_q$$

... except that we must add restrictions!
We move to:

$$[A]_p : \Sigma f : (\Pi q \leq p \to \square).$$

$$\{ \theta : \Pi q \leq p. \Pi r \leq q.f \ q \to f \ r | \quad \text{(A type)} \}$$

$$\text{refl}(\theta, p) \land \text{trans}(\theta, p) \} \quad \text{($\Theta$ restriction)}$$

$$[A]_p := (\pi_1 [A]_p) \ p \ \text{id}_p \quad \text{($\Theta$ functorial)}$$
We move to:

\[ [A]_p : \Sigma f : (\Pi q \leq p \rightarrow \Box). \]
\[ \{ \theta : \Pi q \leq p. \Pi r \leq q.f\ q \rightarrow f\ r \mid \text{refl}(\theta, p) \land \text{trans}(\theta, p) \} \quad (\Theta \text{ restriction}) \]
\[ [A]_p := (\pi_1 [A]_p) p \text{id}_p \]

In general, under a context \( \sigma \) of variables + forcing conditions:

\[ [x]_p^\sigma \overset{def}{=} \theta^{\sigma, \sigma_1(x)}_{\sigma_2(x) \rightarrow p x} \]
We move to:

\[
[A]_p : \Sigma f : (\Pi q \leq p \rightarrow \Box).
\]

\[
\{ \theta : \Pi q \leq p. \Pi r \leq q. f q \rightarrow f r \mid \text{refl}(\theta, p) \land \text{trans}(\theta, p) \}
\]

\[
[A]_p := (\pi_1 [A]_p) p \ \text{id}_p
\]

In general, under a context \( \sigma \) of variables + forcing conditions:

\[
[x]_p^{\sigma} \overset{d e f}{=} \theta^{\sigma, \sigma_1(x)}_{\sigma_2(x) \rightarrow p x}
\]

Now we have witnesses everywhere
We move to:

\[ [A]_p : \Sigma f : (\Pi q \leq p \rightarrow \Box). \]
\[
\{ \theta : \Pi q \leq p. \Pi r \leq q. f q \rightarrow f r | \text{refl}(\theta, p) \land \text{trans}(\theta, p) \} \]

\[ [A]_p := (\pi_1 [A]_p) p \text{id}_p \]

In general, under a context \( \sigma \) of variables + forcing conditions:

\[ [x]_p^\sigma \overset{\text{def}}{=} \theta^\sigma,\sigma_1(x)_{\sigma_2(x) \rightarrow p}x \]

Now we have witnesses everywhere

... but it’s no longer computationally sound!
Some proofs are more equal than others

The culprit is the conversion rule:

\[
\frac{\Gamma \vdash t : A \quad A \equiv_{\beta} B}{\Gamma \vdash t : B}
\]

\[
\frac{p : \mathcal{P} \vdash [t]_p : [A]_p \quad [A]_p \equiv_{\beta} [B]_p}{p : \mathcal{P} \vdash [t]_p : [B]_p}
\]

In general, \( A \equiv_{\beta} B \) does not imply \([A]_p \equiv_{\beta} [B]_p\), as restrictions do not commute/compose “on the nose”.

\[
\begin{align*}
\Gamma &\vdash t : A \\
\Gamma &\vdash \lambda x : \Pi q. p \quad \theta \vdash \lambda x : \Sigma q. p
\end{align*}
\]

\[
\frac{\sigma : p \vdash \lambda x : \Pi q. p \quad \theta \vdash \lambda x : \Sigma q. p}{\Gamma \vdash \tau : \sigma \rightarrow M \quad \sigma \vdash \lambda x : \Pi q. p \quad \theta \vdash \lambda x : \Sigma q. p}
\]
Some proofs are more equal than others

The culprit is the conversion rule:

\[
\frac{\vdash t : A}{\vdash t : B} \quad \frac{A \equiv \beta B}{p : \mathbb{P} \vdash [t]_p : [A]_p \quad [A]_p \equiv \beta [B]_p}
\]

In general, \( A \equiv \beta B \) does not imply \( [A]_p \equiv \beta [B]_p \), as restrictions do not commute/compose “on the nose”.

\[
[[\Pi x : T.U]]_p^\sigma \overset{\text{def}}{=} \{ f : \Pi q : \mathcal{P}_p \Pi x : [T]_q^\sigma \cdot [U]_q^\sigma + (x,T,q) | \text{comm}_\Pi(f,T,U,p) \}
\]

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

\[
\theta^\sigma,T_{p\rightarrow q} \quad \theta^\sigma,U_{p\rightarrow q}
\]

\[
[T]_q^\sigma \xrightarrow{f_q} [U]_q^\sigma
\]
When conversion matters

We only recover that $A \equiv \beta B$ implies $p : \mathbb{P} \vdash [A]_p = \Box [B]_p$.

In the end, you cannot interpret conversion by mere conversion.

\[
\begin{align*}
\vdash t : A & \quad A \equiv \beta B \\
\vdash t : B
\end{align*}
\]

\[
\begin{align*}
p : \mathbb{P} \vdash [t]_p : [A]_p & \quad \pi : [A]_p = [B]_p \\
p : \mathbb{P} \vdash \text{transport}([\pi],[t]_p) : [B]_p
\end{align*}
\]

The « diagram » does not commute in ITT
When conversion matters

We only recover that $A \equiv_B B$ implies $p : \mathbb{P} \vdash [A]_p = \Box [B]_p$.

In the end, you cannot interpret conversion by mere conversion.

$$
\vdash t : A \quad A \equiv_B B \quad \leadsto \quad p : \mathbb{P} \vdash [t]_p : [A]_p \quad \pi : [A]_p = [B]_p
$$

The « diagram » does not commute in ITT

It raises a hell of coherence issues.

- Breaks computation
- Requires *definitional* UIP in the target (i.e. OTT or ETT)
- Requires that $\leq$ is proof-irrelevant.
- Only preorder-based presheaf models!
Nonetheless

In a modified Coq with definitional proof-irrelevance (for Prop):

- We could adapt the proof of consistency of the negation of the continuum hypothesis.
- We could internalize step indexing as a forcing layer (i.e. to obtain a general fixpoint in type theory).
Take $\mathbb{P} \triangleq \mathbb{N}$ with the standard order relation.

- Define $\triangleright\Box : \Box \to \Box$ the “later” modality on $\Box$ in the forcing layer.
  By translation we must provide a witness of $\Pi q \leq p.\Pi T : [\Box]_q, [\Box]_q$, which computes to the unit type when $q = 0$ and the $n$th-approximation of $T$ at $n + 1$. 

- Define $\text{fix } T : (\triangleright\Box T \to T)$ the L"ob rule by providing a witness using the “step-index”.

- Define the lifting $\text{next } T : (T \to \triangleright\Box T)$, morally “delay.” 

In the forcing layer, it becomes possible to reason with general fixpoints on types having the unfolding lemma: $\text{fix } \Box f = f(\text{next } (\text{fix } \Box f))$. 

Take $\mathbb{P} \triangleq \mathbb{N}$ with the standard order relation.

- Define $\triangleright \Box : \Box \rightarrow \Box$ the “later” modality on $\Box$ in the forcing layer.
  
  By translation we must provide a witness of $\Pi q \leq p. \Pi T : [\Box]_q, [\Box]_q$, which computes to the unit type when $q = 0$ and the $n$th-approximation of $T$ at $n + 1$.

- Define $\text{fix}_T : (\triangleright \Box T \rightarrow T) \rightarrow T$ (the L"ob rule) by providing a witness using the “step-index”.

- Define the lifting $\text{next}_T : (T \rightarrow \triangleright \Box T)$, morally “delay”.
Take $\mathbb{P} \cong \mathbb{N}$ with the standard order relation.

- Define $\triangleright \Box : \Box \to \Box$ the “later” modality on $\Box$ in the forcing layer.
  
  By translation we must provide a witness of $\Pi q \leq p. \Pi T : [\Box]_q, [\Box]_q$, which computes to the unit type when $q = 0$ and the $n$th-approximation of $T$ at $n + 1$.

- Define $\text{fix}_T : (\triangleright \Box T \to T) \to T$ (the Löb rule) by providing a witness using the “step-index”.

- Define the lifting $\text{next}_T : (T \to \triangleright \Box T)$, morally “delay”.

In the forcing layer, it becomes possible to reason with general fixpoints on types having the unfolding lemma:

$$\text{fix}_\Box f = f (\text{next} (\text{fix}_\Box f))$$
The setup is not very satisfactory though:

- Doubts about coherence of the whole translation.
- Tedious proofs involving rewriting appear when reasoning with these fixpoints.
Interestingly the Curry-Howard isomorphism explains the difficulties with this translation.

**Root of the failure**

The usual forcing $[\cdot]_p$ translation is **call-by-value**.

That is, assuming $(\mathbb{P}, \leq)$ has definitional laws:

$$ t \equiv_{\beta_v} u \quad \text{implies} \quad [t]_p \equiv_{\beta} [u]_p $$

where $\beta_v$ is generated by the rule:

$$(\lambda x. t) V \rightarrow_{\beta_v} t\{x := V\} \quad (V \text{ a value})$$

This problem is already here in the simply-typed case but less troublesome.
There is an easy Call-by-Push-Value decomposition of forcing.
There is an easy Call-by-Push-Value decomposition of forcing.

- Precomposing by the CBV decomposition we recover the usual forcing

![Diagram showing the relationship between CBV, CBN, CBPV, and lambda calculus.]
There is an easy Call-by-Push-Value decomposition of forcing.

- Precomposing by the CBV decomposition we recover the usual forcing
- Precomposing by the CBN decomposition we obtain a new translation
- ... much closer to Krivine and Miquel’s classical variant
CBN provides new abilities

You only have to change the interpretation of the arrow.

\[
\begin{align*}
\text{CBV} & \quad [\forall x : A. B]_p \simeq \forall q \leq p. \forall x : [A]_q. [B]_q \\
\text{CBN} & \quad [\forall x : A. B]_p \equiv \Pi(x : \forall q \leq p. [A]_q). [B]_p
\end{align*}
\]
CBN provides new abilities

You only have to change the interpretation of the arrow.

\[
\begin{align*}
\text{CBV} & \quad [\Pi x : A. B]_p \cong \Pi q \leq p. \Pi x : [A]_q. [B]_q \\
\text{CBN} & \quad [\Pi x : A. B]_p \equiv \Pi (x : \Pi q \leq p. [A]_q). [B]_p \\
\text{CBN} & \quad [x]_p \equiv x \ p \ id_p
\end{align*}
\]
CBN provides new abilities

You only have to change the interpretation of the arrow.

\[
\begin{align*}
\text{CBV} & \quad [\Pi x : A. B]_p \simeq \Pi q \leq p. \Pi x : [A]_q. [B]_q \\
\text{CBN} & \quad [\Pi x : A. B]_p \equiv \Pi (x : \Pi q \leq p. [A]_q). [B]_p \\
\text{CBN} & \quad [x]_p \equiv x \cdot p \cdot \text{id}_p
\end{align*}
\]

... and everything follows naturally (CBN is somehow a ≪ free » construction).

Interpretation of \( \mathbf{CC}_\omega \)

Assuming that \( \mathbb{P} \) has definitional laws (for identity and composition), then \([\cdot]\) provides a non-trivial translation from \( \mathbf{CC}_\omega \) into itself preserving typing and conversion.

This is to the best of our knowledge, the first effectful translation of \( \mathbf{CC}_\omega \).
The translation

\[
\begin{align*}
\text{[*]}_\sigma & := \lambda (q f : \sigma). \Pi (r g : \sigma \cdot (q, f)). * \\
\text{[□}_i\text{]}_\sigma & := \lambda (q f : \sigma). \Pi (r g : \sigma \cdot (q, f)). \Box_i \\
\text{[x]}_\sigma & := x \sigma_e \sigma(x) \\
\text{[λx : A. M]}_\sigma & := \lambda x : [A]_\sigma^! \cdot [M]_\sigma \cdot x \\
\text{[M N]}_\sigma & := [M]_\sigma \cdot [N]_\sigma^! \\
\text{[Πx : A. B]}_\sigma & := \lambda (q f : \sigma). \Pi x : [A]_\sigma^!(q, f) \cdot [B]_\sigma(q, f) \cdot x \\
\text{[A]}_\sigma & := [A]_\sigma^! \cdot \sigma_e \cdot \text{id}_{\sigma_e} \\
\text{[M]}_\sigma^! & := \lambda (q f : \sigma). [M]_\sigma(q, f) \\
\text{[A]}_\sigma^! & := \Pi(q f : \sigma). [A]_\sigma(q, f) \\
\text{[·]}_p & := p : P \\
\text{[Γ]}_{\sigma(q, f)} & := [Γ]_\sigma, q : P, f : \text{Hom}(\sigma_e, q) \\
\text{[Γ, x : A]}_{\sigma x} & := [Γ]_\sigma, x : [A]_\sigma^!
\end{align*}
\]
Is the definitional side stronger?

This variant is motivated by a Curry-Howard stance.

- No categorical equivalent from the literature (?)..
- Definitely not a presheaf construction!
- In particular, no monotonicity / restrictions
- Only known relative comes from Krivine and Miquel (also CH)
- Yet, still the same object in the simply-typed case.
- Can be used for NBE as well

What is this beast?
Technical issue: how can $\mathbb{P}$ have definitional laws?

Yoneda lemma

The category $(\mathbb{P}_Y, \leq_Y)$ is equivalent to $(\mathbb{P}, \leq)$ (assuming parametricity and functional extensionality).

Furthermore, it has definitional laws as associativity of functions is on the nose in ITT.
Technical issue: how can $\mathbb{P}$ have definitional laws?

Answer: using this one weird old Yoneda trick!

$$(\mathbb{P}, \leq) \mapsto (\mathbb{P}_Y, \leq_Y)$$

$\mathbb{P}_Y := \mathbb{P}$

$p \leq_Y q := \Pi r : \mathbb{P}. q \leq r \rightarrow p \leq r$

**Yoneda lemma**

- The category $(\mathbb{P}_Y, \leq_Y)$ is equivalent to $(\mathbb{P}, \leq)$ (assuming parametricity and functional extensionality).
- Furthermore, it has definitional laws as associativity of functions is on the nose in ITT.
Up to now, we only interpret the negative fragment ($\Pi + \Box$).

\[
\begin{align*}
\text{Inductive types} \\
\text{Up to now, we only interpret the negative fragment (} \Pi + \Box \text{).}
\end{align*}
\]
Up to now, we only interpret the negative fragment ($\Pi + \Box$).

Adapting to (positive) inductive types.
We just need to box all subterms!

\[
[\Sigma x : A. B]_p := \sum (x : \Pi q \leq p. [A]_q). (\Pi q \leq p. [B]_q)
\]

\[
[A + B]_p := (\Pi q \leq p. [A]_q) + (\Pi q \leq p. [B]_q)
\]

Inductive $[\mathbb{N}]_p : \Box := [0] : [\mathbb{N}]_p | [S] : (\Pi q \leq p. [\mathbb{N}]_q) \to [\mathbb{N}]_p$
Yet, the translation does not interpret full dependent elimination.

\[
\begin{align*}
\Pi_{\text{rec}} & : \Pi(P : \square). P \rightarrow (P \rightarrow P) \rightarrow \mathbb{N} \rightarrow P & \checkmark \\
\Pi_{\text{ind}} & : \Pi(P : \mathbb{N} \rightarrow \square). P \ 0 \rightarrow (\Pi n : \mathbb{N}. P \ n \rightarrow P \ (S \ n)) \rightarrow \Pi n : \mathbb{N}. P \ n & \times
\end{align*}
\]
Yet, the translation does not interpret full dependent elimination.

\[
\begin{align*}
\mathbb{N}_{\text{rec}} & \quad \Pi(P : \Box). P \to (P \to P) \to \mathbb{N} \to P \\
\mathbb{N}_{\text{ind}} & \quad \Pi(P : \mathbb{N} \to \Box). P \ 0 \to (\Pi n : \mathbb{N}. P \ n \to P \ (s \ n)) \to \Pi n : \mathbb{N}. P \ n
\end{align*}
\]

Effects \(\rightsquigarrow\) Non-standard inductive terms

(A well-known issue. See e.g. Herbelin’s CIC + callcc)
Yet, the translation does not interpret full dependent elimination.

\[ \text{\(N_{\text{rec}}\)} \quad \Pi(P : \Box). P \rightarrow (P \rightarrow P) \rightarrow \mathbb{N} \rightarrow P \]

\[ \text{\(N_{\text{ind}}\)} \quad \Pi(P : \mathbb{N} \rightarrow \Box). P \ 0 \rightarrow (\Pi n : \mathbb{N}. P \ n \rightarrow P (S \ n)) \rightarrow \Pi n : \mathbb{N}. P \ n \]

**Effects \(\leadsto\) Non-standard inductive terms**

(A well-known issue. See e.g. Herbelin’s CIC + callcc)

Luckily there is a surprise solution coming from classical realizability.

**Storage operators!**
Storage operators

- They allow to prove induction principles in presence of callcc
- Essentially emulate CBV in CBN through a CPS
- Defined in terms of non-dependent recursion

\[ \theta_N : \mathbb{N} \to \Pi R : \Box (\mathbb{N} \to R) \to R \]
\[ \theta_N := \mathbb{N}_{rec} (\lambda R \ k. k \ 0)(\lambda \tilde{n} R \ k. \tilde{n} R (\lambda n. k (S \ n))) \]
Storage operators

- They allow to prove induction principles in presence of \texttt{callcc}
- Essentially emulate CBV in CBN through a CPS
- Defined in terms of non-dependent recursion

\[
\theta_N : \mathbb{N} \rightarrow \Pi R : \square . (\mathbb{N} \rightarrow R) \rightarrow R
\]

\[
\theta_N := \mathbb{N}_{\text{rec}} \left( \lambda R \ k. k \ 0 \right)(\lambda \tilde{n} \ R \ k. \tilde{n} \ R \ (\lambda n. k (S \ n)))
\]

- Trivial in \textbf{CIC}: \text{CIC} \vdash \Pi n \ R k. \ \theta_N n R k =_R k n
- The above propositional \(\eta\)-rule is negated by the forcing translation
- But it interprets a restricted dependent elimination!
Storage operators

- They allow to prove induction principles in presence of \texttt{callcc}
- Essentially emulate CBV in CBN through a CPS
- Defined in terms of non-dependent recursion

\[
\theta_N : \mathbb{N} \rightarrow \Pi R : \square. (\mathbb{N} \rightarrow R) \rightarrow R
\]
\[
\theta_N := \mathbb{N}_{\text{rec}} (\lambda R \ k. \ k \ 0)(\lambda \tilde{n} \ R \ k. \ \tilde{n} \ R \ (\lambda n. \ k \ (S \ n)))
\]

- Trivial in \textbf{CIC}: \textbf{CIC} \vdash \Pi n \ R \ k. \ \theta_N \ n \ R \ k =_R k \ n
- The above propositional \(\eta\)-rule is negated by the forcing translation
- But it interprets a restricted dependent elimination!

\[
\mathbb{N}_{\text{ind}} \quad \Pi P. \ P \ 0 \rightarrow (\Pi n : \mathbb{N}. \ P \ n \rightarrow \theta_N (S \ n) \ \square \ P) \rightarrow \Pi n : \mathbb{N}. \ \theta_N \ n \ \square \ P
\]
Implementation & examples

- A plugin for Coq generating translated terms

A truly definitionnal translation!
Implementation & examples

- A plugin for Coq generating translated terms

A truly definitional translation!

- A handful of independence results and usecases

⇒ Preserves UIP and functional extensionality
⇒ Generate anomalous types that negate univalence
⇒ Preserves (a simple version of) univalence for modal types
⇒ Step indexing (FRP, « fuel trick »)
⇒ Give some intuition for the cubical model
Implementation & examples

- A plugin for Coq generating translated terms

A truly definitional translation!

- A handful of independence results and usecases

- Preserves UIP and functional extensionality
- Generate anomalous types that negate univalence
- Preserves (a simple version of) univalence for modal types
- Step indexing (FRP, « fuel trick »)
- Give some intuition for the cubical model

- Demo
What remains to be done

- Recover a propositional $\eta$-rule by using parametricity
- Understanding the cubical model in CBN.
- Design a general theory of CIC + effects using storage operators
- The next 700 translations of CIC into itself, degenerate translations. E.g. breaking parametricity with built-in quote operators.
Related Work & References

- The Independence of Markov’s Principle in Type Theory. T. Coquand, B. Mannaa, FSCD 2016
- Forcing as a Program Transformation, A. Miquel, LICS 2011.
- The Definitional Side of Forcing - G. Jaber, G. Lewertowski, P.-M. Pédrot, M. Sozeau, N. Tabareau, LICS’16
- Forcing in Type Theory - G. Jaber, M. Sozeau & N. Tabareau, LICS’12
https://github.com/CoqHott/coq-forcing