The Definitional Side of the Forcing

G. Jaber G. Lewertowski **P.-M. Pédrot** M. Sozeau N. Tabareau

INRIA

TYPES
24th May 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)		*(Modal logic)*		*(Category theory)*

- Cohen's original variant is classical
- We will study intuitionistic forcing

Forcing: the Oppression

Why on earth would you use forcing?

Forcing: the Oppression

Why on earth would you use forcing?

- Set theory: a lot of independance results (too late for the Fields medal!)
- Modal logic: Logic *what*?

Forcing: the Oppression

Why on earth would you use forcing?

- Set theory: a lot of independance results (too late for the Fields medal!)
- Modal logic: Logic *what*?
- Category theory: a HoTT topic!
  - Many models arise from presheaf constructions
  - Coquand & al. model of univalence is an example
  - Also step-indexing, parametricity...
  - But this stuff targets sets or topoi (erk)

We want forcing in Type Theory!

Intuitionistic Forcing in **LJ** (Kripke, presheaf, whatever)

Assume a preorder $(\mathbb{P}, \leq)$. We summarize the forcing translation in **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 $\lambda\Pi$.)

Intuitionistic Forcing in **LJ** (Kripke, presheaf, whatever)

Assume a preorder $(\mathbb{P}, \leq)$. We summarize the forcing translation in **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 $\lambda\Pi$.)

Most notably,

$$[\![A \to B]\!]_p := \forall q \leq p.\, [\![A]\!]_q \to [\![B]\!]_q$$

Assume a preorder $(\mathbb{P}, \leq)$. We summarize the forcing translation in **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 $\lambda\Pi$.)

Most notably,

$$[\![A \to B]\!]_p := \forall q \leq p.\, [\![A]\!]_q \to [\![B]\!]_q$$

(Actually this can be adapted straightforwardly to any category $(\mathbb{P}, \mathtt{Hom})$.)

Also sprach Curry-Howard

The previous soundness theorem makes sense in a proof-relevant world:

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

Also sprach Curry-Howard

The previous soundness theorem makes sense in a proof-relevant world:

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

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

Reader	Forcing
$T\ A := \mathbb{P} \to A$	$T_p\ A := \forall q : \mathbb{P}, q \leq p \to A$
$\texttt{read} : 1 \to \mathbb{P}$	$\texttt{read} : 1 \to \mathbb{P}$
$\texttt{enter} : (1 \to A) \to \mathbb{P} \to A$	$\texttt{enter} : (1 \to A) \to \forall p : \mathbb{P}, p \leq \texttt{read}() \to A$

Also sprach Curry-Howard

The previous soundness theorem makes sense in a proof-relevant world:

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

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

Reader	Forcing
$T\ A := \mathbb{P} \rightarrow A$	$T_p\ A := \forall q : \mathbb{P}, q \leq p \rightarrow A$
$\mathtt{read} : 1 \rightarrow \mathbb{P}$	$\mathtt{read} : 1 \rightarrow \mathbb{P}$
$\mathtt{enter} : (1 \rightarrow A) \rightarrow \mathbb{P} \rightarrow A$	$\mathtt{enter} : (1 \rightarrow A) \rightarrow \forall p : \mathbb{P}, p \leq \mathtt{read}() \rightarrow A$

In particular, taking $(\mathbb{P}, \leq)$ to be a full preorder gives the reader monad.

Do it, or do not: there is no try

In 2012, Jaber & al. gave a forcing translation from CIC into itself.

Do it, or do not: there is no try

In 2012, Jaber & al. gave a forcing translation from CIC 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

Do it, or do not: there is no try

In 2012, Jaber & al. gave a forcing translation from CIC into itself.

Intuitively, not that difficult.

- To a type $\vdash A : \square$ associate $p : \mathbb{P} \vdash [\![A]\!]_p : \square$
- 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 $[\cdot]$:

$$\begin{array}{lcll} [A]_p & : & \Pi q \leq p.\, \square & (A \text{ type}) \\ [\![A]\!]_p & := & [A]_p\ p\ \mathtt{id}_p & \end{array}$$

- 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, Jaber & al. gave a forcing translation from CIC into itself.

Intuitively, not that difficult.

- To a type $\vdash A : \square$ associate $p : \mathbb{P} \vdash [\![A]\!]_p : \square$
- 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 $[\cdot]$:

$$\begin{array}{lcll} [A]_p & : & \Pi q \leq p.\, \square & (A \text{ type}) \\ [\![A]\!]_p & := & [A]_p\ p\ \mathtt{id}_p & \end{array}$$

- 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 this naive presentation does not work.

Separate, but equal

The culprit is the conversion rule:

$$\frac{\vdash t : A \qquad A \equiv_\beta B}{\vdash t : B} \quad \rightsquigarrow \quad \frac{p : \mathbb{P} \vdash [t]_p : [\![A]\!]_p \qquad [\![A]\!]_p \equiv_\beta [\![B]\!]_p}{p : \mathbb{P} \vdash [t]_p : [\![B]\!]_p}$$

But in general, $A \equiv_\beta B$ does not imply $[\![A]\!]_p \equiv_\beta [\![B]\!]_p$.

Separate, but equal

The culprit is the conversion rule:

$$\frac{\vdash t : A \qquad A \equiv_\beta B}{\vdash t : B} \quad \rightsquigarrow \quad \frac{p : \mathbb{P} \vdash [t]_p : [\![A]\!]_p \qquad [\![A]\!]_p \equiv_\beta [\![B]\!]_p}{p : \mathbb{P} \vdash [t]_p : [\![B]\!]_p}$$

But in general, $A \equiv_\beta B$ does not imply $[\![A]\!]_p \equiv_\beta [\![B]\!]_p$.

To fix this, Jaber & al. needed to stuff equality proofs everywhere.

- In types: $[\![\square]\!]_p \equiv \Sigma(A : \Pi q \leq p. \square)$. « A *respects some stuff* »
- In functions: $[\![\Pi x : A. B]\!]_p \equiv \Sigma(f : \ldots)$. « f *respects other stuff* »

And only recovered that $A \equiv_\beta B$ implies $p : \mathbb{P} \vdash [\![A]\!]_p =_\square [\![B]\!]_p$.

When conversion matters

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

$$\frac{\vdash t : A \qquad A \equiv_\beta B}{\vdash t : B} \quad \rightsquigarrow \quad \frac{p : \mathbb{P} \vdash [t]_p : [\![A]\!]_p \qquad \pi : [\![A]\!]_p \equiv_\beta [\![B]\!]_p}{p : \mathbb{P} \vdash \mathtt{transport}([\pi], [t]_p) : [\![B]\!]_p}$$

This step is usually dismissed in a categorical world by:

« This diagram commutes. »

When conversion matters

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

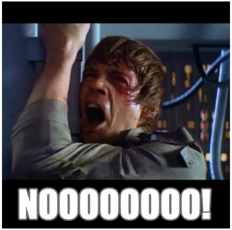
$$\frac{\vdash t : A \qquad A \equiv_\beta B}{\vdash t : B} \quad \leadsto \quad \frac{p : \mathbb{P} \vdash [t]_p : [\![A]\!]_p \qquad \pi : [\![A]\!]_p \equiv_\beta [\![B]\!]_p}{p : \mathbb{P} \vdash \mathtt{transport}([\pi], [t]_p) : [\![B]\!]_p}$$

This step is usually dismissed in a categorical world by:

« This diagram commutes. »

... but here, it raises a hell of coherence issues.

- Breaks computation
- Requires definitional UIP in the target.
- Requires that $\leq$ is proof-irrelevant.
- Only degenerated presheaf models!



A new hope

Interestingly the Curry-Howard isomorphism explains this failure.

Root of the failure

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

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

$$t \equiv_{\beta v} u \qquad \text{implies} \qquad [t]_p \equiv_\beta [u]_p$$

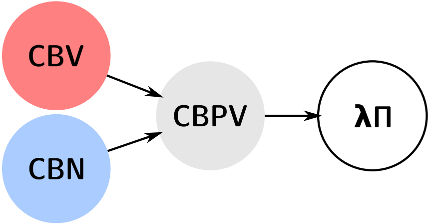
where βv is generated by the rule:

$$(\lambda x.\, t)\ V \longrightarrow_{\beta v} t\{x := V\} \qquad (V \text{ a value})$$

This problem is already here in the simply-typed case but less troublesome.

The Two Sides of the Forcing

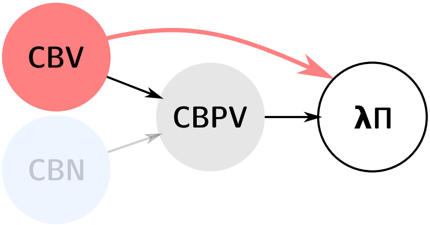
There is an easy Call-by-Push-Value decomposition of forcing.



The Two Sides of the Forcing

There is an easy Call-by-Push-Value decomposition of forcing.

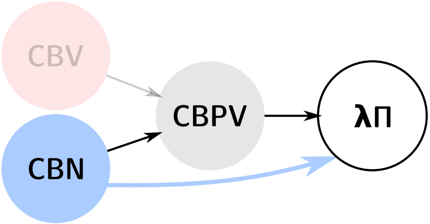
- Precomposing by the CBV decomposition we recover the usual forcing



The Two Sides of the Forcing

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 many abilities some consider to be unnatural

You only have to change the interpretation of the arrow.

$$\begin{array}{ll} \text{CBV} & [\![\Pi x : A.\, B]\!]_p \cong \Pi q \leq p.\, \Pi x : [\![A]\!]_q.\, [\![B]\!]_q \\ \text{CBN} & [\![\Pi x : A.\, B]\!]_p \equiv \Pi (x : \Pi q \leq p.\, [\![A]\!]_q).\, [\![B]\!]_p \end{array}$$

CBN provides many abilities some consider to be unnatural

You only have to change the interpretation of the arrow.

$$
\begin{array}{ll}
\text{CBV} & [\![\Pi x : A.\, B]\!]_p \cong \Pi q \leq p.\, \Pi x : [\![A]\!]_q.\, [\![B]\!]_q \\
\text{CBN} & [\![\Pi x : A.\, B]\!]_p \equiv \Pi(x : \Pi q \leq p.\, [\![A]\!]_q).\, [\![B]\!]_p
\end{array}
$$

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

Interpretation of $\mathbf{CC}_\omega$

Assuming that $\mathbb{P}$ has definitional laws, 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 $\mathbf{CC}_\omega$.

Yoneda not far, patience, soon you will be with him

Technical issue: how can $\mathbb{P}$ have definitional laws?

Yoneda not far, patience, soon you will be with him

Technical issue: how can $\mathbb{P}$ have definitional laws?

Answer: using this one weird old Yoneda trick!

$$(\mathbb{P}, \leq) \quad \mapsto \quad (\mathbb{P}_{\mathcal{Y}}, \leq_{\mathcal{Y}})$$

$$\begin{array}{lcl} \mathbb{P}_{\mathcal{Y}} & := & \mathbb{P} \\ p \leq_{\mathcal{Y}} q & := & \Pi r : \mathbb{P}.\, q \leq r \rightarrow p \leq r \end{array}$$

Yoneda lemma

- The category $(\mathbb{P}_{\mathcal{Y}}, \leq_{\mathcal{Y}})$ is equivalent to $(\mathbb{P}, \leq)$
- Furthermore, it has definitional laws

Inductive types

Up to now, we only interpret the negative fragment $(\Pi + \square)$.

Inductive types

Up to now, we only interpret the negative fragment $(\Pi + \square)$.

But our translation can be adapted easily to inductive types.
We just need to **box** all subterms!

$$
\begin{array}{lcl}
[\![\Sigma x : A.\, B]\!]_p & := & \Sigma(x : \Pi q \leq p.\, [\![A]\!]_q).\, (\Pi q \leq p.\, [\![B]\!]_q) \\
[\![A + B]\!]_p & := & (\Pi q \leq p.\, [\![A]\!]_q) + (\Pi q \leq p.\, [\![B]\!]_q)
\end{array}
$$

$$
\texttt{Inductive}\ [\![\mathbb{N}]\!]_p : \square := [\mathtt{O}] : [\![\mathbb{N}]\!]_p \mid [\mathtt{S}] : (\Pi q \leq p.\, [\![\mathbb{N}]\!]_q) \rightarrow [\![\mathbb{N}]\!]_p
$$

Dependent elimination

Yet, the translation does not interpret full dependent elimination.

$\mathbb{N}_{\texttt{rec}}$	$\Pi(P : \square).\, P \to (P \to P) \to \mathbb{N} \to P$	✓
$\mathbb{N}_{\texttt{ind}}$	$\Pi(P : \mathbb{N} \to \square).\, P\ \texttt{0} \to (\Pi n : \mathbb{N}.\, P\ n \to P\ (\texttt{S}\ n)) \to \Pi n : \mathbb{N}.\, P\ n$	✠

Dependent elimination

Yet, the translation does not interpret full dependent elimination.

$\mathbb{N}_{\texttt{rec}}$	$\Pi(P:\square).\,P \to (P \to P) \to \mathbb{N} \to P$	✓
$\mathbb{N}_{\texttt{ind}}$	$\Pi(P:\mathbb{N} \to \square).\,P\ \texttt{O} \to (\Pi n:\mathbb{N}.\,P\ n \to P\ (\texttt{S}\ n)) \to \Pi n:\mathbb{N}.\,P\ n$	✠

Effects $\rightsquigarrow$ Non-standard inductive terms

(A well-known issue. See e.g. Herbelin's $\mathbf{CIC} + \texttt{callcc}$)

Dependent elimination

Yet, the translation does not interpret full dependent elimination.

$\mathbb{N}_{\mathtt{rec}}$ $\Pi(P : \Box).\, P \to (P \to P) \to \mathbb{N} \to P$ ✓

$\mathbb{N}_{\mathtt{ind}}$ $\Pi(P : \mathbb{N} \to \Box).\, P\ \mathtt{0} \to (\Pi n : \mathbb{N}.\, P\ n \to P\ (\mathtt{S}\ n)) \to \Pi n : \mathbb{N}.\, P\ n$ ✠

Effects $\rightsquigarrow$ Non-standard inductive terms

(A well-known issue. See e.g. Herbelin's $\mathbf{CIC}$ + `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

$$
\begin{array}{lcl}
\theta_{\mathbb{N}} & : & \mathbb{N} \to \Pi R : \square.\, (\mathbb{N} \to R) \to R \\
\theta_{\mathbb{N}} & := & \mathbb{N}_{\mathtt{rec}}\ (\lambda R\ k.\, k\ 0)(\lambda \tilde{n}\ R\ k.\, \tilde{n}\ R\ (\lambda n.\, k\ (S\ n)))
\end{array}
$$

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

$$\begin{array}{lcl} \theta_{\mathbb{N}} & : & \mathbb{N} \to \Pi R : \square.\, (\mathbb{N} \to R) \to R \\ \theta_{\mathbb{N}} & := & \mathbb{N}_{\mathtt{rec}}\ (\lambda R\ k.\, k\, 0)(\lambda \tilde{n}\ R\ k.\, \tilde{n}\ R\ (\lambda n.\, k\ (S\ n))) \end{array}$$

- Trivial in $\mathbf{CIC}$: $\mathbf{CIC} \vdash \Pi n\ R\ k. \quad \theta_{\mathbb{N}}\ n\ R\ k =_R k\ n$
- The above propositional η-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 `callcc`
- Essentially emulate CBV in CBN through a CPS
- Defined in terms of non-dependent recursion

$$\begin{array}{lcl} \theta_{\mathbb{N}} & : & \mathbb{N} \to \Pi R : \square.\, (\mathbb{N} \to R) \to R \\ \theta_{\mathbb{N}} & := & \mathbb{N}_{\texttt{rec}}\ (\lambda R\ k.\, k\ 0)(\lambda \tilde{n}\ R\ k.\, \tilde{n}\ R\ (\lambda n.\, k\ (S\ n))) \end{array}$$

- Trivial in **CIC**: $\mathbf{CIC} \vdash \Pi n\ R\ k. \quad \theta_{\mathbb{N}}\ n\ R\ k =_R k\ n$
- The above propositional η-rule is negated by the forcing translation
- But it interprets a restricted dependent elimination!

$$\mathbb{N}_{\widetilde{\texttt{ind}}} \quad \Pi P.\, P\ \texttt{0} \to (\Pi n : \mathbb{N}.\, P\ n \to \theta_{\mathbb{N}}\ (\texttt{S}\ n)\ \square\ P) \to \Pi n : \mathbb{N}.\, \theta_{\mathbb{N}}\ n\ \square\ P \qquad \checkmark$$

What we also did

- A fancy plugin for Coq generating horrendous well-typed terms

The forcing is definitional with this one!

What we also did

- A fancy plugin for Coq generating horrendous well-typed terms

The forcing is definitional with this one!

- A handful of independence results and usecases

$\leadsto$ Generate anomalous types that negate univalence

$\leadsto$ Step indexing

$\leadsto$ Give some intuition for the cubical model

What we also did

- A fancy plugin for Coq generating horrendous well-typed terms

The forcing is definitional with this one!

- A handful of independence results and usecases

$\rightsquigarrow$ Generate anomalous types that negate univalence

$\rightsquigarrow$ Step indexing

$\rightsquigarrow$ Give some intuition for the cubical model

- A LICS paper detailing the whole story

This is the paper you're looking for!

What remains to be done

- Recover a propositional η-rule by using parametricity
- Understand the cubical model in CBN (*may the Force be with us...*)
- Design a general theory of **CIC** + effects using storage operators
- The next 700 stupid translations of **CIC** into itself

Questions you have?