# The Definitional Side of the Forcing

G. Jaber[1], G. Lewertowski[1], P.-M. Pédrot[2], M. Sozeau[1], and N. Tabareau[2]

[1] IRIF - Université Paris Diderot / $\pi r^2$ - Inria
[2] Inria Rennes - Bretagne Atlantique

Forcing has been introduced by Cohen to prove the independence of the Continuum Hypothesis in set theory. The main idea is to build, from a model $M$, a new model $M'$ for which validity is controlled by a poset of forcing conditions living in $M$. Later, categorical ideas have been used by Lawvere and Tierney [10] to recast this construction in terms of topos of presheaves. It naturally gives rise to a proof-relevant forcing working on categories of conditions rather than simply posets.

Recent years have seen a renewal of interest for forcing, driven by Krivine's classical realizability [7]. In this line of work, forcing is studied as a proof translation, and one seeks to understand its computational content [9, 2], through the Curry-Howard correspondence. Following these ideas, a forcing translation heavily based on the presheaf construction of Lawvere and Tierney was defined in [5] for the Calculus of Inductive Constructions (**CIC**). The main goal was to extend the logic behind Coq with new principles, while keeping its fundamental properties: soundness, canonicity and decidability of type-checking. This approach can be seen, following [1], as type-theoretic metaprogramming.

However, this technique suffers from coherence problems, which complicate greatly the translation. More precisely, the translation of two definitionally equal terms are not in general definitionally equal, but only propositionally equal. Rewriting terms must then be inserted inside the definition of the translation. If this is possible to perform, albeit tedious, when the forcing conditions form a poset, it becomes intractable when we want to define a forcing translation parametrized by a category of forcing conditions.

We propose a novel forcing translation for the Calculus of Constructions which avoids these coherence problems. Departing from the categorical intuitions of the presheaf construction, it takes its roots in a call-by-push-value [8] decomposition of the previous translation. Through this decomposition, the new translation is *call-by-name*, while the previous one one is *call-by-value*. This is easily seen in the translation of dependent products where the type $\Pi x : A.\, B$ is interpreted at level $p$ as $\Pi q \leq p.\, \Pi x : [\![A]\!]_q.\, [\![B]\!]_q$ in call-by-value v.s. $\Pi x : (\Pi q \leq p.\, [\![A]\!]_q).\, [\![B]\!]_p$ in call-by-name. Here, the argument $x$ is boxed under a quantification, which corresponds to the fact that it will be evaluated only when needed.

Assuming the forcing category verifies categorical laws definitionally, we get the following main result.

*Call-by-name forcing provides the first effectful translation of the Calculus of Constructions into itself which preserves definitional equality.*

The requirement on the forcing category is actually not an issue, as we can make any category abide by these definitional laws thanks to a type-theoretic Yoneda embedding.

This translation extends to inductive types by exploiting storage operators [6], an old idea of Krivine to simulate call-by-value in call-by-name in the context of classical realizability, to restrict the power of dependent elimination in presence of effects. The necessity of a restriction should not be surprising and was already present in a similar work by Herbelin [4]. In a nutshell, the typing system requires to purify a term of an inductive type before letting it flow into types. Morally, this is done by ensuring that dependent pattern-machings are typed with other pattern-matchings, e.g. for the $\Sigma$-type elimination:

$$\frac{\Gamma \vdash M : \Sigma x : A.\, B \qquad \Gamma, z : \Sigma x : A.\, B \vdash C : \square \qquad \Gamma, x : A, y : B \vdash N : C\{z := (x, y)\}}{\Gamma \vdash \texttt{match}\ M\ \texttt{with}\ (x, y) \Rightarrow N : \texttt{match}\ M\ \texttt{with}\ (x, y) \Rightarrow C\{z := (x, y)\}}$$

The original **CIC** can be retrieved by adding an $\eta$-law on inductive types which is not preserved by the translation. Actually, the translation allows to build non-canonical inhabitants of inductive types and thus negates this $\eta$-law. Hence,

*Call-by-name forcing provides the first version of* **CIC** *with effects.*

The nice property of preservation of definitional equality is emphasized by the implementation of a Coq plugin[1] which works for any term of **CIC**, assuming it complies with the restricted typing rules.

By using forcing, we produced various results around homotopy type theory. We proved that functional extensionality is preserved in any forcing layer. We also showed that the negation of Voevodsky's univalence axiom is consistent with **CIC** plus functional extensionality. This statement could already be deduced for the existence of a set-based *proof-irrelevant* model [11], but we provided the first formalization of it, in a proof relevant setting, and by an easy use of the forcing plugin. Under an additional assumption of parametricity, we showed conversely that we get the preservation of the univalence axiom.

This is a first step towards the use of the category of cubes as the type of forcing conditions to give a computational content to the cubical type theory [3] of Coquand et al. in Coq, and in particular to the univalence axiom.

# References


[1] T. Altenkirch and A. Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016.

[2] A. Brunel. *The Monitoring Power of Forcing Transformation*. PhD thesis, Univ. Paris Nord, 2014.

[3] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom, 2015. Preprint.

[4] H. Herbelin. A constructive proof of dependent choice, compatible with classical logic. In *LICS*, pages 365–374. IEEE Computer Society, 2012.

[5] G. Jaber, N. Tabareau, and M. Sozeau. Extending Type Theory with Forcing. In *LICS 2012 : Logic In Computer Science*, pages 0–0, Dubrovnik, Croatia, June 2012.

[6] J. Krivine. Classical logic, storage operators and second-order lambda-calculus. *Ann. Pure Appl. Logic*, 68(1):53–78, 1994.

[7] J.-L. Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.

[8] P. B. Levy. *Call-by-push-value*. PhD thesis, Queen Mary, University of London, 2001.

[9] A. Miquel. Forcing as a program transformation. In *LICS*, pages 197–206. IEEE Computer Society, 2011.

[10] M. Tierney. Sheaf theory and the continuum hypothesis. In *Toposes, algebraic geometry and logic*, pages 13–42. Springer, 1972.

[11] B. Werner. Sets in types, types in sets. In *Theoretical aspects of computer software*, pages 530–546. Springer, 1997.


[1] Available at `https://github.com/CoqHott/coq-forcing`.