Extending Type Theory with Forcing

Guilhem Jaber, Nicolas Tabareau, Matthieu Sozeau

Extending Type Theory with Forcing

Guilhem Jaber and Nicolas Tabareau
INRIA
École des Mines de Nantes
Nantes, France

Matthieu Sozeau
INRIA
Université Paris Diderot
Paris, France

***Abstract*—This paper presents an intuitionistic forcing translation for the Calculus of Constructions (CoC), a translation that corresponds to an internalization of the presheaf construction in CoC. Depending on the chosen set of forcing conditions, the resulting type system can be extended with extra logical principles. The translation is proven correct—in the sense that it preserves type checking—and has been implemented in Coq. As a case study, we show how the forcing translation on integers (which corresponds to the internalization of the topos of trees) allows us to define general inductive types in Coq, without the strict positivity condition. Using such general inductive types, we can construct a shallow embedding of the pure λ-calculus in Coq, without defining an axiom on the existence of an universal domain. We also build another forcing layer where we prove the negation of the continuum hypothesis.**

I. Introduction

Forcing is a method originally designed by Cohen in the 60s to prove the independence of the Continuum Hypothesis from the axiomatic set theory ZFC [4]. Forcing is known to be intimately connected to the sheaf construction, as stated by Lawvere and Tierney in [15]. Recently, after the works of Krivine [8] and Miquel [11], it became accepted that forcing techniques are of great interest for the extension of the Curry-Howard correspondence. The starting point of our paper is to connect these two observations:

> "Intuitionistic forcing for Type Theory is an internalization of the presheaf construction in type theory."

We develop a forcing layer to increase the power of a logic based on a type theory just as it is used to build a *generic* model $M[G]$ from a previous model M in set theory[7]. In this way, we are able to develop a new generation of logics—that can be defined modularly using forcing theory. This translation relies on an internalization of the presheaf construction inside CoC in order to be able to use forcing directly in type theory. Then, it becomes possible to exhibit new reasoning principles inside a forcing layer by using the structure of the chosen forcing conditions. But, no matter what new logical principles have been defined, their consistency can be deduced for free:

> "The consistency of a logic defined in a forcing layer ensues from the consistency of the ground logic."

Indeed, we are able to extend a type theory with new reasoning principles and new objects, without defining them as axioms. Besides coherence problems, avoiding the axiomatic approach enables us to give a computational content to these new principles: programs can be associated to them.

On a connected line of work on Kripke semantics, Appel et al. [1] have proposed to understand step-indexing—a technique to handle general recursion in programing language semantics—as (Kripke) forcing on the set of natural numbers. Those natural numbers can be used to define (or force) a particular modality in the logic, with an induction principle directly lifted from natural number. More recently, Birkedal et al. [2] have shown that this construction can be understood algebraically as a mean to work inside the topos of trees, which is a generic way to define general recursive types in a semantical model. Similarly, we use forcing on the set of natural numbers to provide general inductive types in CoC, without relying on a positivity condition. This construction makes possible to define a universal type $\mathcal{D}$ for terms of the pure lambda calculus that induces a shallow embedding of the pure λ-calculus into CoC. The fact that we can use a conversion rule that is normalizing to describe the β-reduction of the pure λ-calculus should not appear as a contradiction. Unfolding the recursive type $\mathcal{D}$ is handled by an equality but not by the conversion rule and so has to be explicit in the term. We also use another set of forcing conditions to force the negation of the continuum hypothesis.

A prototype implementation of the translation is available at https://github.com/mattam82/Forcing. It is built on top of Coq, translating terms of a forcing layer so that they can be verified by the typechecker of Coq.

Plan of the paper. In Section II, we present an extended version of the Calculus of Construction with proof irrelevance and subset types. Then, we explain the forcing translation in terms of the presheaf construction (Section III) and define it formally (Section IV). In Section V, we illustrate this translation by choosing the natural numbers as the poset of forcing conditions, providing a framework for general inductive types. Then, in Section VI, we illustrate the forcing translation with another poset of forcing conditions to force the negation of the continuum hypothesis. Finally, we discuss related works (Section VII) and present future works (Section VIII).

II. The Calculus of Constructions

The Calculus of Constructions is a dependent type system which uses the ideas of the Curry-Howard correspondence to represent proofs. Indeed, proof-terms are considered as first class citizens. There is no syntactic distinction between proofs, logical formulas and types. This means, among others, that the dependent product, considered on a logical side, represents the universal quantification.

More precisely, we consider Russell [14], an extension of the Calculus of Constructions with a distinguished subset type for dependent sums whose second component is a proof. It also treats objects of subset type specially in the conversion rule, allowing to transparently move an object between different subset types. An object of type T can be freely used as an object of the subset type $\{x : T \mid P\}$ for any property P. Any derivation in this liberal system can be translated into a derivation of CoC extended with proof-irrelevance and metavariables, generating proof obligations for the missing logical information. On the condition that these obligations are solvable, one obtains a well-formed complete derivation in CoC extended with proof-irrelevance. The forcing translation presented here uses the Russell system as target, which has just the right amount of (apparent) extensionality needed to handle the manipulation of forcing conditions necessary in the translation.

A. Grammar of terms and contexts

The terms of Russel are given by the following grammar

$$\begin{array}{rcl} T, U, M, N & := & \mathrm{Prop} \mid \mathrm{Type}_i \mid \Pi x : T.U \mid \Sigma x : T.U \\ & & \mid \{x : T \mid U\} \mid x \mid \lambda x : T.M \\ & & \mid MN \mid \pi_1 M \mid \pi_2 M \end{array}$$

where $i \in \mathbb{N}$. Note that we consider a version of CoC with a hierarchy of universes Type_i. We define $\mathcal{S} = \{\mathrm{Prop}, \mathrm{Type}_i\}$ to be the set of sorts and a function max on $\mathcal{S}$ as :

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

Contexts are defined as (ordered) lists of pairs formed by a variable and a term. We define a judgment of well-formation **wf** for contexts:

- $\mathbf{wf}([])$
- $\mathbf{wf}(\Gamma :: (x, T))$ iff $\mathbf{wf}(\Gamma)$ and $x \notin FV(\Gamma)$ and $\Gamma \vdash T : s$ with $s \in \{\mathrm{Prop}, \mathrm{Type}_i\}$.

It is defined by mutual induction with the typing judgment.

B. Subsets and the subtyping rule

The subtyping relation $\leq$ is defined as the smallest reflexive, transitive congruence relation including conversion and satisfying the following:

$$\textsc{SubInj}\ \frac{\Gamma \vdash T \leq T'}{\Gamma \vdash T \leq \{x : T' \mid U\}}$$

$$\textsc{SubProj}\ \frac{\Gamma \vdash T \leq T'}{\Gamma \vdash \{x : T \mid U\} \leq T'}$$

This relation allows to coerce freely between subsets on the same support type.

C. Conversion rule

The conversion rule $\cong$ is defined as the smallest congruence satisfying the following equations:

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

D. The Type System

The axioms of the system are defined as: $\mathcal{A} \stackrel{def}{=} \{(\mathrm{Prop}, \mathrm{Type}_0), (\mathrm{Type}_i, \mathrm{Type}_{i+1})\}$. The type system (figure 1) is a straightforward extension of the PTS for CoC.

The system enjoys Subject Reduction and has decidable typechecking (c.f. [14] for a complete metatheoretical study).

E. Inductive types

To simplify the setting, we do not present inductive types in the calculus, although they can be forced directly as constant presheaves. Nevertheless, in the rest of the paper, we allow ourselves to consider simple inductive types such as the set of natural numbers or logical connectors. We also use the inductive definition of equality **eq** together with its unique inhabitant **eq_refl** and its elimination principle **eq_rect**.

III. Internalizing the presheaf construction in CoC

In this section, we explain the intuitionistic forcing translation $[-]$ of the forcing layer for a poset $(\mathcal{P}, \preccurlyeq)$ of forcing conditions. The full definition is given in Section IV. This translation can be seen as an internalization inside CoC of the presheaf construction over $\mathcal{P}$.

We refer the reader to [10] for the definition of the presheaf construction.

A. Forcing conditions

The forcing translation is defined over a poset $\mathcal{P}$ of forcing condition of type Type equipped with the preorder relation $\preccurlyeq$. This preorder enables us to define the poset $\mathcal{P}_p$ of forcing conditions that are below p as

$$\mathcal{P}_p = \{q : \mathcal{P} \mid q \preccurlyeq p\}$$

Because $\mathcal{P}$ is a poset, there is an injection $\iota_{p,q}$ from $\mathcal{P}_q$ to $\mathcal{P}_p$ as soon as there is a proof $\pi_{p,q} : q \preccurlyeq p$. This injection is used as an implicit coercion in the rest of this article. This chain of preorders enables us to construct presheaves over $\mathcal{P}$ by approximation.

$$\textsc{Var}\ \frac{\mathbf{wf}(\Gamma) \qquad (x:T)\in\Gamma}{\Gamma\vdash x:T}$$

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

$$\textsc{Abstr}\ \frac{\Gamma,x:T\vdash M:U}{\Gamma\vdash \lambda x:T.M:\Pi x:T.U}$$

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

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

$$\textsc{Sum}\ \frac{\Gamma\vdash T:\mathrm{Type}_i \qquad \Gamma,x:T\vdash U:\mathrm{Type}_i}{\Gamma\vdash \Sigma x:T.U:\mathrm{Type}_i}$$

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

$$\textsc{Proj-1}\ \frac{\Gamma\vdash M:\Sigma x:T.U}{\Gamma\vdash \pi_1 M:T}$$

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

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

$$\textsc{Subtype}\ \frac{\Gamma\vdash M:T \qquad \Gamma\vdash T\leq U}{\Gamma\vdash M:U}$$

Fig. 1. Russell's Type System

B. Presheaf approximations as dependent sums

In category theory, a presheaf P over $\mathcal{P}$ on Set is given by a family $(P_p)_{p\in\mathcal{P}}$ of sets together with restriction maps

$$P_q \xleftarrow{\theta_{p,q}} P_p$$

for all $q \preccurlyeq p$, satisfying the usual commutative diagrams ensuring the naturality of those maps. In the special setting of a preorder, the naturality corresponds to the reflexivity and transitivity of restriction maps.

In Russell, this restriction maps can be formalized using a dependent sum and the naturality conditions can be imposed using a subset type rejecting ill-formed restriction maps. Thus, the type $\mathbf{PSh}(p,s)$ of a presheaf at level p on a sort s can be defined as

$$\Sigma f:\mathcal{P}_p\to s.\{\theta:\Pi q:\mathcal{P}_p.\Pi r:\mathcal{P}_q.fq\to fr \mid \mathbf{trans}(\theta,p)\wedge\mathbf{refl}(\theta,p)\}$$

where $\mathbf{trans}(\theta,p)$ and $\mathbf{refl}(\theta,p)$ are defined in Figure 2.

Given a (closed) term $T:s$ (where $s\in\mathcal{S}$), we introduce two notations to extract the support and the restriction maps of the associated presheaf $[T]_p$:

$$\begin{array}{lcl}\llbracket T\rrbracket_p & \stackrel{def}{=} & (\pi_1[T]_p)p\\ \theta^T_{p\to q} & \stackrel{def}{=} & (\pi_2[T]_p)pq\end{array}$$

C. Presheaf approximation of variables

As we internalize the presheaf construction directly in CoC, we have to translate variables of the calculus, which is not the case for purely semantic definitions. The problem with variables is that they can be used for a presheaf approximation that is smaller than the presheaf approximation for which they have been defined. For instance, this situation typically amounts to derive the following judgment, for $q\preccurlyeq p$,

$$\Gamma,x:\llbracket T\rrbracket_p\vdash x:\llbracket T\rrbracket_q$$

which differs from the usual Var rule. This means that the translation of a variable must introduce the right restriction map to go from the presheaf approximation at level p to the presheaf approximation at level q. To that purpose, we need to parametrize the translation of a term T with a map σ that associates the type and level of approximation to every free variable occurring in T. In what follow, σ denotes a function from variables to types and forcing conditions. We note $\sigma_1(x)$ (resp. $\sigma_2(x)$) for the type (resp. forcing condition) assigned to x by σ. Given a context Γ, we say that σ is a valid interpretation of Γ if it assigns the same variable to the same type, and all conditions appearing in σ are ordered. Given a valid interpretation σ, the translation of a variable is given by

$$[x]^\sigma_p \qquad \stackrel{def}{=} \qquad \theta^{\sigma,\sigma_1(x)}_{\sigma_2(x)\to p}x$$

and the translation of rule Var becomes

$$\Gamma,x:\llbracket T\rrbracket^\sigma_p\vdash\theta^{\sigma,T}_{p\to q}x:\llbracket T\rrbracket^\sigma_q$$

which is now derivable from the rule Var and App. Note that, inductively, the definitions of the support and restriction maps of the presheaf approximation are also annotated with the interpretation σ.

D. Presheaf approximation of dependent products

The category of presheaves is cartesian closed. This suggests that dependent products can be translated as presheaf approximations. In category theory, the internal hom $[-,-]$ is described by

$$[T,U]_p\cong\mathbf{Hom}_{\mathbf{PSh}}(y(p)\times T,U)$$

where y denotes the Yoneda embedding. This means that $[T,U]_p$ is itself a presheaf that associates to any forcing condition q a morphism

$$f_q:\mathcal{P}(q,p)\times T_q\to U_q.$$

But in our case, $\mathcal{P}$ is a preorder so f_q exists only when $q \preccurlyeq p$. A dependent product will thus be translated at level p as a family of dependent product indexed by forcing conditions that are below p. As it is the case for morphisms of presheaves, dependent functions between presheaf approximations also have to commute with restriction maps as given by the following categorical commutative diagram

$$\begin{array}{ccc} [\![T]\!]^\sigma_p & \xrightarrow{f_p} & [\![U]\!]^\sigma_p \\ {\scriptstyle \theta^{\sigma,T}_{p\to q}}\downarrow & & \downarrow{\scriptstyle \theta^{\sigma,U}_{p\to q}} \\ [\![T]\!]^\sigma_q & \xrightarrow[f_q]{} & [\![U]\!]^\sigma_q \end{array} \tag{1}$$

To this end, the support of the presheaf approximation at level p is given by the following subset type

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

where the first component is a family of dependent product indexed by a forcing condition below p and where

$$\mathbf{comm}_\Pi(f,T,U,p) \stackrel{def}{=} \Pi q : \mathcal{P}_p . \Pi r : \mathcal{P}_q . \Pi x : [\![T]\!]^\sigma_q . (fr)(\theta^{\sigma,T}_{q\to r} x) = \theta^{\sigma+(x,T,q),U}_{q\to r}(fqx)$$

reflects the categorical commutative diagram (1). The restriction maps are simply given by identity coercions

$$\theta^{\sigma,\Pi x:T.U}_{p\to q} \stackrel{def}{=} \lambda f : [\![\Pi x : T.U]\!]^\sigma_p . \lambda r : \mathcal{P}_q . fr$$

The translation of a function is given by

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

The proof that $[\lambda x : T.M]^\sigma_p$ validates the commutation condition is deduced from the set of equality on restriction maps.

The translation of an application is obtained by applying the translated argument $[N]^\sigma_p$ to the translated function $[M]^\sigma_p$ taken at level p,

$$[MN]^\sigma_p \stackrel{def}{=} [M]^\sigma_p \ p \ [N]^\sigma_p .$$

E. Presheaf approximation of dependent sums and subset types

The category of presheaves has products and coproducts defined pointwise. This suggests that dependent sums can be translated as presheaf approximation pointwisely, and so for the associated operators.

$$\begin{array}{lcl} [\![\Sigma x : T.U]\!]^\sigma_p & \stackrel{def}{=} & \Sigma x : [\![T]\!]^\sigma_p . [\![U]\!]^{\sigma+(x,T,p)}_p \\ [(t,u)_{\Sigma x:T.U}]^\sigma_p & \stackrel{def}{=} & ([t]^\sigma_p, [u]^\sigma_p)_{\Sigma x:[\![T]\!]^\sigma_p . [\![U]\!]^\sigma_p} \\ [\pi_i M]^\sigma_p & \stackrel{def}{=} & \pi_i [M]^\sigma_p \end{array}$$

The same pointwise construction works for subset types.

F. Presheaf approximation of sorts

CoC has a hierarchy of universes induced by the rule Ax. This means that Prop and Type_i have to be themselves translated as presheaf approximation at level p. This is done by defining a term $\mathbf{PShC}(p,s)$ that encodes the restriction map available on $\mathbf{PSh}(p,s)$. Note that because of proof irrelevance in our type system, the translation of $\mathbf{PSh}(p,\text{Prop})$ is simpler as restriction maps are in that case proofs of monotonicity and thus, transitivity and reflexivity of restriction maps are automatic (see Figure 2).

IV. The Forcing Translation

In this section, we present the formal definition of the forcing translation, state its correctness and show how it can generically be used to extend a type theory.

A. Definition

An interpretation σ is defined as a list of triples formed by a variable, a type and a forcing condition. Given a variable x appearing in σ, we note $\sigma_1(x)$ (resp. $\sigma_2(x)$) the type (resp. forcing condition) associated to x in σ.

σ is said to be a *valid* interpretation of a context

$$\Gamma = [(x_1, T_1), \dots (x_n, T_n)]$$

if $\sigma_1(x_i) = T_i$ for all $1 \leq i \leq n$ and the list of forcing conditions appearing in σ is ordered: $\sigma_2(x_1) \succcurlyeq \cdots \succcurlyeq \sigma_2(x_n)$. Thus, a valid interpretation σ gives rise to a sequence $p_1 : \mathcal{P}, p_2 : \mathcal{P}_{p_1}, \dots, p_n : \mathcal{P}_{p_{n-1}}$ of forcing conditions appearing in σ (where $p_i = \sigma_2(x_i)$).

Given a valid interpretation σ of Γ, we pose

$$[\![\Gamma]\!]^\sigma = p_1 : \mathcal{P}, x_1 : [\![T_1]\!]^\sigma_{p_1}, \dots, p_n : \mathcal{P}_{p_{n-1}}, x_n : [\![T_n]\!]^\sigma_{p_n}$$

Figure 2 presents the forcing translation of CoC. This translation has largely been explained in Section III. The only remaining point concerns the special translation for the application and conversion rules, that have to be enforced by some rewriting on restriction maps.

B. Ensuring conversion and substitution lemma on types

We can prove that $[\![U\{x/N\}]\!]^\sigma_p$ is equal to $[\![U]\!]^{\sigma+(x,T,p)}_p \{x/[N]^\sigma_p\}$, however they are not convertible, due to the fact that properties like $\mathbf{trans}(\theta,p)$ are equalities which are not integrated in the conversion rule.

We do not work in an extentional theory: from the fact that M is of type T and that $T = U$, we cannot deduce that M is of type U. Indeed, $T = U$ does not imply that $T \simeq U$.

It is however possible to transform slightly the term M so that it can be considered of type U. This method can be seen as a way to transform an extentional theory to an intentional one, as presented in [6], [12]. This is done in Coq using the term **eq_rect**, which corresponds to the elimination of the equality type. Using this term and equalities available on restriction maps, we are able to define:

$$
\begin{array}{lcl}
\mathbf{refl}(\theta, p) & \stackrel{def}{=} & \Pi q : \mathcal{P}_p.\theta^{\sigma,T}_{q\to q} = id \\
\mathbf{trans}(\theta, p) & \stackrel{def}{=} & \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.\Pi s : \mathcal{P}_r.(\theta^{\sigma,T}_{r\to s})(\theta^{\sigma,T}_{q\to r}) = \theta^{\sigma,T}_{q\to s} \\
\mathbf{PSh}(p, \mathrm{Type}_i) & \stackrel{def}{=} & \Sigma f : \mathcal{P}_p \to \mathrm{Type}_i.\{\theta : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.fq \to fr \mid \mathbf{trans}(\theta, p) \wedge \mathbf{refl}(\theta, p)\} \\
\mathbf{PSh}(p, \mathrm{Prop}) & \stackrel{def}{=} & \{f : \mathcal{P}_p \to \mathrm{Prop} \mid \theta : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.fq \to fr\} \\
\mathbf{PShC}(p, s) & \stackrel{def}{=} & \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : \mathbf{PSh}(q, s). \\
 & & \quad (\lambda s : \mathcal{P}_r.(\pi_1 f)s, \lambda s : \mathcal{P}_r.\lambda t : \mathcal{P}_s.\lambda x : (\pi_1 f)s.(\pi_2 f)stx) \\
 & & \\
[\![T]\!]^\sigma_p & \stackrel{def}{=} & (\pi_1 [T]^\sigma_p)p \\
\theta^{\sigma,T}_{p\to q} & \stackrel{def}{=} & (\pi_2 [T]^\sigma_p)pq \\
 & & \\
[s]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.\mathbf{PSh}(q, s), \mathbf{PShC}(p, s)) \qquad\qquad \text{for } s \in \mathcal{S} \\
 & & \\
\mathbf{comm}_\Pi(f, T, U, p) & \stackrel{def}{=} & \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.\Pi a : [\![T]\!]^\sigma_q.(fr)(\theta^{\sigma,T}_{q\to r}a) = \theta^{\sigma+(a,T,q),U\{a/x\}}_{q\to r}(fqa) \\
 & & \\
[\Pi x : T.U]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.\{f : \Pi r : \mathcal{P}_q.\Pi x : [\![T]\!]^\sigma_r.[\![U]\!]^{\sigma+(x,T,r)}_r \mid \mathbf{comm}_\Pi(f, T, U, q)\}, \\
 & & \quad \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Pi x : T.U]\!]^\sigma_q.\lambda s : \mathcal{P}_r.fs) \\
[\lambda x : T.M]^\sigma_p & \stackrel{def}{=} & \lambda q : \mathcal{P}_p.\lambda x : [\![T]\!]^\sigma_q.[M]^{\sigma+(x,T,q)}_q \text{ with the proof } \mathbf{commlam}_{M,T} \\
[\Sigma x : T.U]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.\Sigma x : [\![T]\!]^\sigma_q.[\![U]\!]^{\sigma+(x,T,q)}_q, \\
 & & \quad \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Sigma x : T.U]\!]^\sigma_q.(\theta^{\sigma,T}_{q\to r}(\pi_1 f), \theta^{\sigma+(x,T,q),U}_{q\to r}(\pi_2 f))) \\
[(t, u)_{\Sigma x:T.U}]^\sigma_p & \stackrel{def}{=} & ([t]^\sigma_p, [u]^\sigma_p)_{\Sigma x:[\![T]\!]^\sigma_p.[\![U]\!]^\sigma_p} \\
[\pi_i M]^\sigma_p & \stackrel{def}{=} & \pi_i [M]^\sigma_p \\
 & & \\
[MN]^\sigma_p & \stackrel{def}{=} & ([M]^\sigma_p p)[N]^\sigma_p \\
[x]^\sigma_p & \stackrel{def}{=} & \theta^{\sigma,\sigma_1(x)}_{\sigma_2(x)\to p}x \\
[\mathbf{app}_{T,U,N}M]^\sigma_p & \stackrel{def}{=} & \mathbf{eq\_rect}[\![U\,\{x/N\}]\!]^\sigma_p\,\mathbf{id}\,\left([\![U]\!]^{\sigma+(x,T,p)}_p\,\left\{x/[N]^\sigma_p\right\}\right)\pi^{\mathbf{app}}_{p,T,U,N}\,[M]^\sigma_p \\
[\mathbf{conv}_{T,U}M]^\sigma_p & \stackrel{def}{=} & \mathbf{eq\_rect}[\![U]\!]^\sigma_p\,\mathbf{id}\,[\![T]\!]^\sigma_p\pi^{\mathbf{conv}}_{p,T,U}\,[M]^\sigma_p
\end{array}
$$

Fig. 2. Definition of the translation

- for every type T, U and every term N of type T, a proof of equality ensuring a substitution lemma
$$\pi^{\mathbf{app}}_{p,T,U,N} : [\![U\,\{x/N\}]\!]^\sigma_p = [\![U]\!]^{\sigma+(x,T,p)}_p\,\left\{x/[N]^\sigma_p\right\}$$
- For every type T, U, a proof of equality ensuring a convertibility lemma
$$\pi^{\mathbf{conv}}_{p,T,U} : [\![T]\!]^\sigma_p = [\![U]\!]^\sigma_p$$

We now have to annotate the term generated by the application of rules App and Conv with two new constructors $\mathbf{app}_{T,U,N}$ and $\mathbf{conv}_{T,U}$:

$$\textsc{App}',\ \frac{\Gamma \vdash M : \Pi x : T.U \qquad \Gamma \vdash N : T}{\Gamma \vdash \mathbf{app}_{T,U,N}MN : U\,\{N/x\}}$$

$$\textsc{Conv}',\ \frac{\Gamma \vdash M : T \qquad T \simeq U}{\Gamma \vdash \mathbf{conv}_{T,U}M : U}$$

This changes slightly the type theory of the forcing layer, since normally the uses of the conversion rule do not appear in the term. Figure 2 depicts the translation of $\mathbf{app}_{T,U,N}$ and $\mathbf{conv}_{T,U}$ which ensure the two following lemmas.

Lemma 1 (substitution lemma)**.** *If* $\Gamma \vdash N : T$ *and*

$$[\![\Gamma]\!]^\sigma \vdash [M]^\sigma_p : [\![U]\!]^\sigma_p\,\left\{[N]^\sigma_p/x\right\}$$

then

$$[\![\Gamma]\!]^\sigma \vdash [\mathbf{app}_{T,U,N}M]^\sigma_p : [\![U\,\{N/x\}]\!]^\sigma_p$$

where p *is the last forcing condition occurring in* σ*.*

Lemma 2 (convertibility lemma)**.** *If*

$$[\![\Gamma]\!]^\sigma \vdash [M]^\sigma_{p_n} : [\![T]\!]^\sigma_{p_n}$$

and $T \simeq U$ then

$$[\![\Gamma]\!]^\sigma \vdash [\mathbf{conv}_{T,U} M]^\sigma_{p_n} : [\![U]\!]^\sigma_{p_n}$$

where p is the last forcing condition occurring in σ.

The proofs of those two lemmas are direct using the type of $\pi^{\mathbf{app}}_{p,T,U,N}$ and $\pi^{\mathbf{conv}}_{p,T,U}$.

To realize the complete translation to CoC, it first needs to be extended with proof-irrelevance. Lastly, the obligations ($\pi^{\mathbf{app}}_{p,T,U,N}$ and $\pi^{\mathbf{conv}}_{p,T,U}$) coming from typechecking the translated terms should be solved automatically, they are simple applications of transitivity and reflexivity of the order on forcing conditions or simplifications due to commutativity proofs.

C. Correctness

Before stating the correctness of the translation, we need a weakening lemma for valid interpretation.

Lemma 3. *If $\Gamma \vdash T : s$ and σ is a valid interpretation of Γ, then for every forcing condition p and every interpretation σ' disjoint of σ, $[T]^{\sigma'\cdot\sigma}_p = [T]^\sigma_p$.*

Proof: The proof is done by induction on T. The only interesting case is when T is a variable x. Then from $\Gamma \vdash T : s$ we know that $(x, s) \in \Gamma$, and since σ is valid for Γ, there exists a forcing condition q such that $(x, T, q) \in \sigma$. So we just have to prove that $\theta^{\sigma'\cdot\sigma,s}_{p\to q} = \theta^{\sigma,s}_{p\to q}$, which follows from the definition of $\mathbf{PSh}(p, s)$. ∎

Theorem 1 (Typing Correctness)**.** *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 occurring in σ.

Proof: By induction on the typing derivation of $\Gamma \vdash M : T$. The proof is given in the companion technical appendix[1]. ∎

D. Importing theorems in the forcing layer

Obviously we do not want to have to prove again all the theorems of Coq in the forcing layer. Hopefully, a proof M of a proposition P of Coq can be converted to a proof M' of P in the forcing layer. We just have to annotate the uses of the rule Conv, which do not appear in M. This can be done when Coq checks that M is of type P.

E. Extending the forcing layer with new constructors

To extend the logical power of the type theory of a forcing layer, we follow a general mechanism. We first add new symbols corresponding to the objects we want to reason on. For example, in the next section we will introduce a new fixpoint combinator. We then define their translation, which is a way to give them a meaning. Finally, we add properties about these new elements whose translations are proven in the original layer.

[1]The technical appendix is available on the web page of the second author: http://tabareau.fr

More precisely, after adding a new symbol f of type T to the forcing layer, we define its translation $[f]^\sigma_p$. Then, we can add a proof-term symbol π of a lemma P about f as soon as we are able to define a proof $[\pi]^\sigma_p$ of $[P]^\sigma_p$ in the ground logic.

This has to be compared to the axiomatic approach in Coq, which introduce a new symbol using `Parameter f : T` and then properties about it using `Axiom π : P`. In this approach, f and P have no computational content.

F. Applying the forcing translation inductively

For now, we have defined a forcing layer on top of Coq, but it is in fact possible to define a forcing layer on top of another forcing layer:

$$\text{Coq} \xleftarrow{[.]^{\mathcal{F}_1}} \mathcal{F}_1 \xleftarrow{[.]^{\mathcal{F}_2}} \mathcal{F}_2$$

To do so, $\mathcal{F}_1$ must contain Russell, so that the translation $[.]^{\mathcal{F}_2}$ is well-defined. This means that $[.]^{\mathcal{F}_1}$ must translate Russell, which requires some extensions of the translation in figure 2. Then, the poset of forcing conditions $\mathcal{P}_2$ used by $\mathcal{F}_2$ has to be defined in $\mathcal{F}_1$. So a term of $\mathcal{F}_2$ is translated in Coq simply by combining $[.]^{\mathcal{F}_1}$ and $[.]^{\mathcal{F}_2}$.

This construction is reminiscent of the iterated forcing construction as presented for example in [7]. In terms of topos, this means to build the category of presheaves valuated in another category of presheaves.

G. Sheaf construction and Excluded middle

We have seen how to import a theorem proved in Coq in a forcing layer, simply by using its proof-term. However, this method does not work with axioms, for good reasons: there is no reason it stays true in the forcing layer, since we do not have a proof of it.

One simple example is the excluded-middle

$$\mathbf{EM} \stackrel{def}{=} \Pi P : \text{Prop}. P \vee (P \to \text{false})$$

Its translation $[\mathbf{EM}]^\sigma_p$ is equal to

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

It is thus possible to build a set of forcing conditions which negate this formula, simply adapting the construction of the usual Kripke model which negates **EM**.

If we want to keep this axiom true, a standard way is to add a negative translation to the forcing translation. This is done in Topos theory by the sheafification process on the dense topology (usually noted $\neg\neg$). Indeed, it is well known [10] that the topos $Sh(\mathcal{P}_{\neg\neg}, \mathcal{E})$ is a Boolean topos while this is not true for $PSh(\mathcal{P}, \mathcal{E})$.

It seems possible to adapt the same technique in our setting. We need to modify the translation so that we associate sheaves to types, enforcing gluing and local identity properties for presheaves.

V. The Step-Indexed Layer

As a first illustration of the power of the forcing translation, we study the forcing layer obtained when $\mathcal{P}$ is an infinite well-ordered set. When $\mathcal{P}$ is the set of natural numbers, this layer has already been semantically studied in [2] as it corresponds to the topos of trees. The novelty in the setting of the Calculus of Constructions is that we can construct recursive types over any recursive definition, without relying on a definition of contractiveness that ensures the existence of a fixpoint. In particular, when a recursive definition satisfies the usual *strictly positive condition*, we show that our definition directly induces the usual one, making possible to merge the two notions in Coq. We also define a *later* modality $\triangleright$ on Prop together with the Löb rule that provides a general inductive principle in the logic. Interestingly, the Löb rule is interpreted directly as a special instance of the fixpoint construction, giving a computational meaning to the Löb rule. Even though we could define the construction below with any infinite well-ordered set of forcing conditions, we only do it with the natural numbers to make definitions more readable. We call this layer the step-indexed (SI) layer.

A. Adding new constructors in the SI layer

As explained in Section IV-E, we can define new terms in the SI layer and give their meaning through their forcing translation in the original layer.

We will need to consider, for each $s \in \mathcal{S}$, two special terms unit_s and Unit_s such that

$$\vdash \mathrm{unit}_s : \mathrm{Unit}_s : s$$

that correspond to the unity of each sort. For example $\vdash \mathbf{I} : \mathbf{true} : \mathrm{Prop}$. They are translated directly to themselves using the constant translation.

The later modality: We introduce in the SI layer a later modality

$$\triangleright_s : s \to s$$

for any sort $s \in \mathcal{S}$. This modality amounts to shift a presheaf on s one step on the right. Its translation $[\![\triangleright_s]\!]^\sigma_p$ is defined by

$$\begin{array}{l}\lambda q : \mathrm{nat}_p.\lambda T : [\![s]\!]^\sigma_q.\\ \quad (\lambda r : \mathrm{nat}_q.\texttt{match } r \texttt{ with}\\ \qquad \mid 0 \texttt{ => } \mathrm{Unit}_s\\ \qquad \mid \mathbf{S}r' \texttt{ => } (\pi_1 T)r'\\ \quad , \lambda r : \mathrm{nat}_q.\lambda t : \mathrm{nat}_r.\lambda M : U_r.\texttt{match } t \texttt{ with}\\ \qquad \mid 0 \texttt{ => } \mathrm{unit}_s\\ \qquad \mid \mathbf{S}t' \texttt{ => } (\pi_2 T)\,(\mathrm{Pred}\, r)\, t'\, M)\end{array}$$

with $U_r \stackrel{def}{=} \texttt{match } r \texttt{ with} \mid 0 \texttt{ => } \mathrm{Unit}_s \mid \mathbf{S}r' \texttt{ => } (\pi_1 T)r'$.

The fixpoint operator: We can define a fixpoint operator

$$\mathbf{fix}_T : (\triangleright_s T \to T) \to T$$

for every T of sort s. The meaning of $\mathbf{fix}_T$ is given by its approximation at level p, computed by induction. At level 0, the fixpoint of $f : \triangleright_s T \to T$ is given by the value of f on the unique inhabitant of $\triangleright_s T$ at level 0. And at level $p+1$, the value is given by f applied to the approximation at level p. Formally, the translation $[\![\mathbf{fix}_T]\!]^\sigma_p$ is defined by

$$\begin{array}{l}\lambda q : \mathrm{nat}_p.\lambda f : [\![\triangleright_s T \to T]\!]^\sigma_q.\mathbf{nat\_rect}_{s,q}(\lambda r : \mathrm{nat}_q.[\![T]\!]^\sigma_r)\\ (f\,0\,\mathrm{unit}_s)\,(\lambda r : \mathrm{Pred}_q.\lambda a : [\![T]\!]^\sigma_r.f(\mathbf{S}r)a)\,q\end{array}$$

where $\mathrm{Pred}_p \stackrel{def}{=} \{q : \mathrm{nat} \mid q < p\}$ and $\mathbf{nat\_rect}_{s,p}$ is defined as a restriction of $\mathbf{nat\_rect}_s$ on the set nat_p.

The later modality and fixpoint operator on Prop*:* When T is a proposition P (and thus $s = \mathrm{Prop}$), $\mathbf{fix}_P M$ computes a proof of P from a proof M of $\triangleright_{\mathrm{Prop}} P \to P$, This is exactly what the Löb rule

$$\textsc{Löb}\ \frac{\triangleright_{\mathrm{Prop}} P \vdash P}{\vdash P}$$

does, so $\mathbf{fix}_P$ gives the computational content of this rule. Thus, applying the Löb rule on a proof π of $\triangleright_{\mathrm{Prop}} P \to P$ simply amounts to consider the proof term $\mathbf{fix}_P(\pi) : P$.

The lifting of the later modality: For every T of sort s, there exists a lifting

$$\mathbf{next}_T : (T \to \triangleright_s T)$$

that transports elements of the sheaf T into elements of the sheaf $\triangleright_s T$ This morphism simply amounts to use the retractions $\theta^{\sigma,T}_{Sp\to p}$ to lift elements of the presheaf accordingly. The translation $[\![\mathbf{next}_T]\!]^\sigma_p$ is given by

$$\begin{array}{l}\lambda q : \mathrm{nat}_p.\lambda u : [\![T]\!]^\sigma_q.\texttt{match } q \texttt{ with}\\ \quad \mid 0 \texttt{ => } \mathrm{unit}_s\\ \quad \mid \mathbf{S}q' \texttt{ => } \theta^{\sigma,T}_{q\to q'}u\end{array}$$

Internalizing the later modality for sorts: For every sort s, we can internalize the shifting induced by the later modality directly in the presheaf. This is done by introducing a morphism

$$\mathbf{switch}_s : (\triangleright_{\mathrm{Type}} s \to s)$$

whose translation $[\![\mathbf{switch}_s]\!]^\sigma_p$ is defined by

$$\begin{array}{l}\lambda q : \mathrm{nat}_p.\lambda f : [\![\triangleright_{\mathrm{Type}} s]\!]^\sigma_q.\\ \quad (\lambda r : \mathrm{nat}_q.\texttt{match } r \texttt{ with}\\ \qquad \mid 0 \texttt{ => } \mathrm{Unit}_s\\ \qquad \mid \mathbf{S}r' \texttt{ => } (\pi_1 f)r'\\ \quad , \lambda r : \mathrm{nat}_q.\lambda s : \mathrm{nat}_r.\lambda M : T_r.\texttt{match } s \texttt{ with}\\ \qquad \mid 0 \texttt{ => } \mathrm{unit}_s\\ \qquad \mid \mathbf{S}s' \texttt{ => } (\pi_2 f)(\mathrm{Pred}\, r)s'M)\end{array}$$

with $T_r \stackrel{def}{=} \texttt{match } r \texttt{ with} \mid 0 \texttt{ => } \mathrm{Unit}_s \mid \mathbf{S}r' \texttt{ => } (\pi_1 f)r'$.

The lifting and switching of the later modality are connected by the following lemma.

Lemma 4. *For every $T : s$ and every forcing condition p, $[\![\mathbf{switch}_s(\mathbf{next}_s T)]\!]^\sigma_p = [\![\triangleright_s T]\!]^\sigma_p$.*

B. Forcing equalities on the new constructors

To make the later modality, the lifting and the switching usable in practice, we introduce proof-terms that make naturality and commutativity properties explicit in the SI layer.

For for any $T : s_1$ and $U : s_2$, we add two terms $\mathbf{comlater}_{\mathbf{\Pi}}^{\mathbf{T,U}} : \triangleright_{s_2}(\Pi x : T.U) \to (\Pi x : \triangleright_{s_1} T. \triangleright_{s_2} U)$ and $\mathbf{comlater}_{\mathbf{\Sigma}}^{\mathbf{T,U}} : \triangleright_{s_2}(\Sigma x : T.U) \to \Sigma x : \triangleright_{s_1} T. \triangleright_{s_2} U$ in the SI layer. We also add proof-terms that state the naturality of $\mathbf{next}_T$ with respect to T.

C. General recursive types

Before providing a definition of recursive types for any general recursive definition, we study the property of the translation of **fix**.

Proposition 1. *For every $f : \triangleright s \to s$ and every forcing condition p, $[\mathbf{fix}_s f]_p^\sigma \simeq [f(\mathbf{next}_s(\mathbf{fix}_s f))]_p^\sigma$.*

The proposition above is induced by the following (conversion) equalities.

Lemma 5. *For every natural number p, the following holds:*

1) *for every $T : s$, $[\![\triangleright_s T]\!]_{\mathbf{S}p}^\sigma \simeq [\![T]\!]_p^\sigma$*
2) *for every $M : T$, $[\mathbf{next}_T M]_{\mathbf{S}p}^\sigma \simeq [M]_p^\sigma$*
3) *for every $f : \triangleright s \to s$, $[\mathbf{fix}_s f]_{\mathbf{S}p}^\sigma \simeq [f]_{\mathbf{S}p}^\sigma(\mathbf{S}p)[\mathbf{fix}_s f]_p^\sigma$*

Now, given a recursive definition $f : s \to s$ for any sort s, we can define

$$\mu_s f = \mathbf{fix}_s \lambda x. f(\mathbf{switch}_s x).$$

Then, using Proposition 1 and Lemma 4, we can add a proof term

$$\mathrm{rec}_{\mu_s} : \mu_s f = f(\triangleright_s \mu_s f)$$

which is just translated in the original layer using **eq_refl**

$$[\mathrm{rec}_{\mu_s}]_p^\sigma = \mathbf{eq\_refl}[\mu_s f]_p^\sigma.$$

Finally, using **eq_rect**, we can define two morphisms $\mathbf{fold}_s : \Pi f : s \to s. \mu_s f \to f(\triangleright_s \mu_s f)$ and $\mathbf{unfold}_s : \Pi f : s \to s. f(\triangleright_s \mu_s f) \to \mu_s f$ together with two proof terms

$$\pi_s^{\mathbf{fold}} : \Pi f : \_. \Pi x : \_. \mathbf{unfold}_s f(\mathbf{fold}_s f x) = x$$
$$\pi_s^{\mathbf{unfold}} : \Pi f : \_. \Pi x : \_. \mathbf{fold}_s f(\mathbf{unfold}_s f x) = x$$

translated using **eq_refl**.

Note that compare to [2], we do not require contractiveness of the recursive definition to compute a recursive type. But the unfolding of a recursive type introduces uniformly a later modality in front of any use of the recursive variable.

Our definition of recursive types introduces a later modality $\triangleright$ on the recursive type at each unfolding. But in the particular case of recursive types satisfying the strictly positive condition, we can automatically erase the introduced $\triangleright$. For example, consider the definition of binary trees on natural numbers (for simplicity, we use a usual sum and product notation)

$$\mathrm{Tree} = \mu_s(\lambda T : \mathrm{Type}.(\mathrm{emp} : 1) + (\mathrm{node} : \mathrm{nat} \times T \times T)).$$

As this recursive type is strictly positive, there is an induction principle in our setting, annotated with $\triangleright$ modalities.

$$\begin{aligned}\mathrm{Tree}_{\mathrm{ind}} : \Pi P : \mathrm{Tree} \to \mathit{Prop}, P\ \mathrm{emp} \to \\ (\Pi n : \mathrm{nat}. \Pi t : \triangleright\mathrm{Tree}, \mathbf{switch}((\mathbf{next} P)t) \\ \to \Pi t' : \triangleright\mathrm{Tree}, \mathbf{switch}((\mathbf{next} P)t') \\ \to P(\mathrm{node}\ n\ t\ t')) \to \Pi t : \mathrm{Tree}, Pt\end{aligned}$$

But using the lifting $\mathbf{next}_{\mathrm{Tree}}$, and the equality of Lemma 4, we deduce

$$\begin{aligned}\mathrm{Tree}'_{\mathrm{ind}} : \Pi P : \mathrm{Tree} \to \mathit{Prop}, P\ \mathrm{emp} \to \\ (\Pi n : \mathrm{nat}. \Pi t : \mathrm{Tree}, \triangleright(Pt) \to \Pi t' : \mathrm{Tree}, \triangleright(Pt') \\ \to P(\mathrm{node}\ n\ t\ t')) \to \Pi t : \mathrm{Tree}, Pt\end{aligned}$$

The lifting $\mathbf{next}_P : P \to \triangleright P$ (which gives a computational content to the rule Mono of Gödel-Löb logic) is now used on hypotheses to weaken the formula and get the usual induction principle

$$\begin{aligned}\mathrm{Tree}''_{\mathrm{ind}} : \Pi P : \mathrm{Tree} \to \mathit{Prop}, P\ \mathrm{emp} \to \\ (\Pi n : \mathrm{nat}. \Pi t : \mathrm{Tree}, Pt \to \Pi t' : \mathrm{Tree}, Pt' \\ \to P(\mathrm{node}\ n\ t\ t')) \to \Pi t : \mathrm{Tree}, Pt\end{aligned}$$

D. An example: the pure λ-calculus

Let's consider the generalized recursive type

$$\mathcal{D} \quad \stackrel{def}{=} \quad \mu_{\mathrm{Type}}(\lambda T : \mathrm{Type}. T \to T).$$

This amounts to add in Coq a type satisfying the usual domain equation for the pure lambda calculus

$$\mathcal{D} = \mathcal{D} \to \mathcal{D}.$$

The idea is that, although this object does not exists in Coq, we can manipulate it directly in the forcing layer, and only its approximations at level n will be considered by Coq's type checker.

For simplicity, we suppose that the function $\mathbf{next}_T$ is declared as a coercion and thus we do not write it explicitly in the rest of this section. We pose

$$\mathbf{fun} f \stackrel{def}{=} \mathbf{unfold}_{\mathrm{Type}}(\lambda T : \mathrm{Type}. T \to T) f$$

and

$$\mathbf{defun} f \stackrel{def}{=} \mathbf{fold}_{\mathrm{Type}}(\lambda T : \mathrm{Type}. T \to T) f.$$

We can introduce an analogous of $\mathbf{switch}_s$ for $\mathcal{D}$ defined as

$$\downarrow : \triangleright\mathcal{D} \to \mathcal{D} \stackrel{def}{=} \lambda t : \triangleright\mathcal{D}. \mathbf{fun}(\lambda\_ : \triangleright\mathcal{D}. t)$$

Thus, we can define application in $\mathcal{D}$ as

$$f @_{\mathcal{S}} s \stackrel{def}{=} \downarrow (\mathbf{fun} f) s$$

Intuitively, the operator $\downarrow$ tags each β-reduction in the term, thus we get a pure lambda calculus where we can

keep track of places where a reduction has occurred. For example, we can construct the usual looping term

$$\Omega \stackrel{def}{=} \mathbf{fun}(\lambda x : \triangleright\mathcal{D}.x@x)$$

and prove that for all integers n,

$$\Omega@\Omega = \downarrow^n (\Omega@\Omega)$$

where $\downarrow^n$ denotes n applications of $\downarrow$. Note that even $\Omega@\Omega$ reduces to itself (plus a lift), there is no problem of termination because we have here an equality but the two terms are not convertible. Indeed, each β-redex is "guarded" by **defun** $\circ$ **fun** that has to be explicitly rewritten into **id** to enable Coq's conversion.

In the same way, we can define the Y fixpoint combinator as

$$\begin{array}{rrcl} & Y & \stackrel{def}{=} & \mathbf{fun}(\lambda f.(Y'f)@(Y'f)) \\ \text{with} & Y'f & \stackrel{def}{=} & \mathbf{fun}(\lambda x.f@(x@x)) \end{array}$$

Since β-reductions are tagged, we do not have the usual equation $Yg = g(Yg)$. Indeed, those terms are β-equivalent, but the places where β-reductions have to be done are different. Thus, we only get a weaker version of the unfolding lemma.

Lemma 6. *For all terms $g : \mathcal{D}$,*

$$Y@g = \downarrow^2 (g@((Y'g)@(Y'g))).$$

VI. Forcing the negation of the Continuum Hypothesis

We now build a forcing layer where we can prove the negation of the Continuum Hypothesis, assuming the excluded middle in the original layer (note that, as explained in Section IV-G, the excluded middle is not preserved in the forcing layer). We adapt the usual construction of Cohen, using its reformulation in terms of topos of sheaves, as presented in [10]. In the following, $\mathbf{P}(T)$ denotes the type $T \to \text{Prop}$ corresponding to the "power set" of T. We will build a forcing layer where we can add a type A with injections

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

and such that there is no surjection from nat to A and from A to $\mathbf{P}(\text{nat})$, thus negating the Continuum Hypothesis.

The set of forcing conditions is given by

$$\mathcal{P} \stackrel{def}{=} (\mathbf{P}(\mathbf{P}(\text{nat})) \times \text{nat}) \to_{fin} \text{Prop}$$

where $\to_{fin}$ denotes the type of function with finite support. This type can be defined in Coq. The order is the usual inclusion order on functions: $q \subseteq p$ if the domain of p is smaller than the domain of q and the two functions coincide on their common domain.

For a closed term $T : s$, we note $\widehat{T}$ for the constant presheaf defines as

$$[\![\widehat{T}]\!]^\sigma_p = (\lambda q : \mathcal{P}_p.T, \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.id).$$

The set A that will negate the continuum hypothesis is $\widehat{\mathbf{P}(\text{nat})}$. The idea is to collapse $\mathbf{P}(\widehat{\mathbf{P}(\text{nat})})$ to $\mathbf{P}(\text{nat})$ in the forcing layer. Then, using the injection from $\widehat{\mathbf{P}(\text{nat})}$ to $\mathbf{P}(\widehat{\mathbf{P}(\text{nat})})$, this gives us the wanted injection $i_2 : \widehat{\mathbf{P}(\text{nat})}$ to $\mathbf{P}(\text{nat})$. To sum up, we will build injection

$$\text{nat} \hookrightarrow^{i_1} \widehat{\mathbf{P}(\text{nat})} \hookrightarrow^{i_2} \mathbf{P}(\text{nat})$$

with the proofs of non-existence of surjections

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

We insist on the fact that $\widehat{\mathbf{P}(\text{nat})}$ is not equal to $\mathbf{P}(\widehat{nat}) = \mathbf{P}(\text{nat})$ since $[\![\mathbf{P}(\text{nat})]\!]^\sigma_p \stackrel{def}{=} \text{nat} \to (\mathcal{P}_p \to \text{Prop})$ while $[\![\widehat{\mathbf{P}(\text{nat})}]\!]^\sigma_p \stackrel{def}{=} \text{nat} \to \text{Prop}$.

A. Building the injection i_2

We add a new symbol f of type $\mathbf{P}(\widehat{\mathbf{P}(\text{nat})}) \to \mathbf{P}(\text{nat})$ whose translation is defined as

$$[f]^\sigma_p \stackrel{def}{=} \lambda q : \mathcal{P}_p.\lambda b : \mathbf{P}(\mathbf{P}(\text{nat})).\lambda r : \mathcal{P}_q.\lambda n : \text{nat}.\lambda s : \mathcal{P}_r. \\ (s \preccurlyeq r \wedge s(b,n) = \text{true})$$

Proposition 2. *For every forcing condition p, $[f]^\sigma_p$ is an injection, which means that the proposition*

$$[\![\Pi b_1, b_2 : \mathbf{P}(\mathbf{P}(\text{nat})).f(b_1) = f(b_2) \Rightarrow b_1 = b_2]\!]_p$$

is provable in Coq plus excluded middle.

Proof: Since p is a partial map from $(\mathbf{P}(\mathbf{P}(\text{nat})) \times \text{nat})$ to Prop whose domain is finite, we can find an m such that $p(b_1, m)$ and $p(b_2, m)$ are undefined. If $b_1 \neq b_2$, we can define a new forcing condition q that extend p such that $q(b_1, m) = \text{true}$ and $q(b_2, m) = \text{false}$. But then we can easily check that

$$[f]^\sigma_p(p)(b_1)(p)(m)(q) = \text{true}$$

and

$$[f]^\sigma_p(p)(b_2)(p)(m)(q) = \text{false}$$

Using the excluded middle, we can conclude that $b_1 = b_2$. ∎

Finally, i_2 is built from f as:

$$i_2 \stackrel{def}{=} \widehat{\mathbf{P}(\text{nat})} \hookrightarrow \mathbf{P}(\widehat{\mathbf{P}(\text{nat})}) \stackrel{f}{\to} \mathbf{P}(\text{nat})$$

The injection i_1 is just obtained as the lifting of the injection from nat to $\mathbf{P}(\text{nat})$.

B. Absence of surjections

We now prove that if there is no surjection between S and T in the ground system, then there is no surjection between $\widehat{S}$ and $\widehat{T}$ in the forcing layer. This is usually called the "conservation of cardinals".

Assume the formula

$$\mathbf{NS}(T, S) \stackrel{def}{=} (\Sigma F : T \to S.\Pi s : S.\Sigma t : T.F(t) = s) \to \text{false}$$

We will prove in the forcing layer the formula $\mathbf{NS}(\widehat{T}, \widehat{S})$.

As the restriction map on $\widehat{T}$ and $\widehat{S}$ are equal to the identity, $[\![\mathbf{NS}(\widehat{T},\widehat{S})]\!]_p^{[]}$ just enforces that there is no $F : \mathcal{P}_q \to (T \to S)$ such that $F(q)$ is a surjection between T and S, which is given by $\mathbf{NS}(T,S)$.

This proof is much simpler than the usual one in classical forcing. Indeed, we do not need any hypothesis like the *countable chain coundition*[7] on the set of forcing conditions $\mathcal{P}$. This is due to the fact that we are working with constant presheaves, whose translation is trivial, which would not be the case if we were working with constant sheaves[2].

VII. Related Works

A. Topos of Trees

In [2], Birkedal et al. study the internal logic of the topos of presheaves over ω. They show that this logic admits general recursive types as soon as the considered recursive definitions are contractive. In Section V, we have presented a syntactic translation of this logical layer using forcing.

Our presentation presents several advantages: (i) Since we have access to the universe Type, we are able to define an automatic translation from any recursive definition to a contractive recursive definition. (ii) We are conservative with respect to inductive types of Coq. (iii) General recursive types are translated into Coq terms, and can thus be given a computational contents.

B. Forcing as a Program Transformation

Krivine has been one of the first to study the computational content of the forcing translation [8]. This work has been rephrased by Miquel in [11], where the forcing translation of proofs is interpreted as a program transformation on λ-terms. Their works focus on classical logic while we only study an intuitionistic translation. Moreover, since they are working in a logical system with a syntactic stratification between proofs and propositions ($\mathbf{PA}\omega$ in [11]), proof-terms and formulas are not translated uniformly, contrary to our work.

VIII. Future Works

A. From Presheaves to Sheaves

As we have seen in IV-G, our forcing translation is intuitionistic. A solution to conserve the excluded middle would be to use sheaves instead of presheaves, for a well suited topology (the dense or double negation topology) [10]. Sheaves restrict presheaves with gluing conditions, so we need to introduce a notion of topology on $\mathcal{P}$, formalized inside Coq, to define them.

B. Formalizing semantics of complex programming languages

Defining semantics of rich programming languages with mutable states, recursive types and impredicative polymorphism is an hard problem. Recently, some works combining step-indexing with Kripke logical relations have been proposed [13], [5], [3]. Those semantics use a notion of Kripke world to specify memory, which has to been defined as a generalized inductive type of the form $W = Val \to W \to \text{Prop}$. Formalizing them is thus hard, and has been the motivation of [2].

We plan to use forcing iteration to define a forcing layer at the top at the step-indexed layer, whose forcing conditions would be Kripke worlds modelling memory states. It would then be possible to state abstract rules to reason on stateful programs, opening the door to a complete formalization of those works in Coq.

C. Ultrafilter over $\mathbb{N}$

An interesting example of forcing translation we wish to adapt is the proof of Levin [9] of conservativity of the existence of an ultrafilter over $\mathbb{N}$ with respect to higher-order Peano Arithmetic with the axiom of dependent choice.

This proof uses partial functions from $\mathbb{N}$ to bool with an infinite domain as forcing conditions. Defining a forcing layer over this set, we could give a computational content to proofs which use an ultrafilter, like Gödel completeness theorem or Ramsey theorem. It would then be interesting to compare this interpretation with the one in [8].

[2] A constant presheaf is not a sheaf in general.

References

[1] A.W. Appel, P.-A. Melliès, C.D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of POPL'07*, 2007.

[2] L. Birkedal, R. Møgelberg, J. Schwinghammer, and K. Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. In *Proceedings of LICS'11*, 2011.

[3] L. Birkedal, B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. Step-indexed Kripke models over recursive worlds. In *Proceedings of POPL'11*, 2011.

[4] P.J. Cohen and M. Davis. *Set theory and the continuum hypothesis*. WA Benjamin New York, 1966.

[5] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proceedings of ICFP'10*, 2010.

[6] M. Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, University of Edinburgh, 1995.

[7] T.J. Jech. *Set theory*. Springer Verlag, 2003.

[8] J.L. Krivine. Realizability algebras: a program to well order $\mathbb{R}$. *Logical Methods in Computer Science*, 2011.

[9] A.M. Levin. One conservative extension of formal mathematic analysis with a scheme of dependent choice. *Mathematical Notes*, 22:524–528, 1977.

[10] S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer, 1992.

[11] A. Miquel. Forcing as a program transformation. In *Proceedings of LICS'11*, 2011.

[12] N. Oury. Extensionality in the calculus of constructions. *Theorem Proving in Higher Order Logics*, pages 278–293, 2005.

[13] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*. CUP, 1998.

[14] M. Sozeau. *An environment for programming with dependent types*. PhD thesis, University of Paris Sud XI, Orsay, France, 2008.

[15] M. Tierney. Sheaf theory and the continuum hypothesis. *LNM 274*, pages 13–42, 1972.