Pursuing Shtuck

Pierre-Marie Pédrot

Pursuing Shtuck

PIERRE-MARIE PÉDROT, INRIA, France

CCS Concepts: • **Theory of computation → Type theory**; *Control primitives*; *Constructive mathematics.*

Additional Key Words and Phrases: Sheaf, type theory, syntactic model, side-effect, interaction tree

1 INTRODUCTION

Sheaves are an ubiquitary structure in topos theory [22, 27], that historically arose from algebraic geometry. They define an important class of topoi called Grothendieck topoi. Abstractly, sheaves can be understood as a refinement of presheaves which preserves more structure from the base category. In the field of logic, they are notably used to prove completeness results over a wide variety of rich deduction systems, and are even claimed to constitute a unifying framework for mathematics [6].

It is a well-known fact that topos theory and type theory are very close objects. Empirically, many techniques developed in one field can be transfered to the other one with a few tweaks. Despite their superficial similarity, the subtle but foundational differences that do exist between them can sometimes result in serious hurdles in the process, though. For instance, porting the dull presheaf construction to type theory proved to be non-trivial and required heavy amendments to overcome issues with intensionality [19, 20, 32].

Given the paramount importance of sheaves in topos theory on the one hand, and the arguable success of porting presheaves to a type-theoretic setting on the other, it is a natural question to ask whether sheaves are amenable to the same type-theoretic adaptation. So natural that there has actually already been a wealth of research on a generalization of this question to the study of modalities in the context of homotopy type theory [44]. This line of work, inspired from higher topos theory, led to a somewhat comprehensive literature [38, 40].

The last paper, notably, contains many non-trivial results and studies a much more general case. It is thus a legitimate question to ask whether there is much more to say on the topic of type-theoretical sheaves. As type theorists leaning towards computer science, we answer positively to this rhetorical question. Thus, taken at face value, the results from this paper are not particularly new, but our point of view is remarkably different.

Our first qualm about the HoTT-style sheafification is its degree of generalization. Abstraction, a landmark of category theory, effectively hides the sheer simplicity of sheaves defined in a pedestrian way. It also has the unfortunate consequence to obscure the kinship of sheaves with well-known structures in computer science. This paper aims at demistifying sheaves for the average type theorist.

The second source of frustration is the complete disconnection of topos theory with computation. And indeed the HoTT account of modalities eschews their computational content, hiding it under non-trivial equality proofs. As staunch supporters of the Curry-Howard correspondence, we believe that it is a categorical imperative of a type-theoretic account of sheaves to expose their operational behaviour. This stance aligns with similar results from classical realizability [24]. It turns out that the underlying side-effect of sheaves is simple, if not disappointing.

Finally, we care about minimalism. As fashionable as HoTT might be, it makes very strong assumptions about the resulting type theory, and is incompatible with a lot of alternative interpretations. In this paper, we study the sheaf model from what we belive to be a neutral intensional ground, essentially a weak variant of CIC. We crucially need a distinguished sort of propositions,

Author's address: Pierre-Marie Pédrot, pierre-marie.pedrot@inria.fr, INRIA, France.

but we will not even require impredicativity. The validity of the different constructions is then explicitly tied to some additional principles.

2 THE STANDARD WAY

For completeness of the exposition, we will recall in this section the usual definition of presheaves and sheaves and state the standard results from topos theory we will try to mimic in type theory. In particular, we will work in this section, and only in this section, in a set-theoretic metatheory. Let us fix a small category $\mathbb{P}$ that we will call the base category.

Definition 2.1. A presheaf is a functor $\mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$.

Lemma 2.2. *Presheaves with natural morphisms between them form a topos* $\mathbf{PSh}(\mathbb{P})$.

A sieve on $p \in \mathbb{P}$ is a set $P \subseteq \Sigma q \in \mathbb{P}.\, \mathsf{Hom}_{\mathbb{P}}(q, p)$ that is closed under precomposition. We will write $\mathfrak{S}_p$ for the set of sieves on p.

Definition 2.3. A Grothendieck topology $\mathfrak{T}$ on $\mathbb{P}$ is a $\mathbb{P}$-indexed family of collection of sieves, i.e. $\mathfrak{T}_p \subseteq \mathfrak{S}_p$ for all $p \in \mathbb{P}$, that enjoys the following closure properties.

- If $P \in \mathfrak{T}_p$ and $\alpha \in \mathsf{Hom}_{\mathbb{P}}(q, p)$ then $\alpha^* P \in \mathfrak{T}_q$ where $\alpha^* P$ is the pullback of P along α.
- The maximal sieve on p is always in $\mathfrak{T}_p$.
- If $P \in \mathfrak{T}_p$, $Q \in \mathfrak{S}_p$ and $P \subseteq \bigcup_{q \in \mathbb{P}} \left\{(q, \alpha) \mid \alpha \in \mathsf{Hom}_{\mathbb{P}}(q, p), \alpha^* Q \in \mathfrak{T}_q\right\}$ then $Q \in \mathfrak{T}_p$.

We fix a topology $\mathfrak{T}$ in the rest of this section, and recall the main result of interest.

Definition 2.4. Let A be a presheaf and $P \in \mathfrak{S}_p$. A compatible family of A on P is a family of elements $x_{q,\alpha} \in A_q$ for every $(q, \alpha) \in P$ that satisfies the equation $A[\beta](x_{q,\alpha}) = x_{r,\beta\circ\alpha}$ for all $(q, \alpha) \in P$ and $\beta \in \mathsf{Hom}_{\mathbb{P}}(r, q)$.

Definition 2.5. A presheaf A is a $\mathfrak{T}$-sheaf whenever for every $P \in \mathfrak{T}_p$ and every compatible family x of A on P there exists a unique element $\hat{x} \in A_p$ s.t. $A[\alpha](\hat{x}) = x_{q,\alpha}$ for all $(q, \alpha) \in P$.

Theorem 2.6. *The full subcategory* $\mathbf{Sh}(\mathbb{P}, \mathfrak{T})$ *of* $\mathfrak{T}$*-sheaves is a topos.*

Theorem 2.7. *The inclusion* $\mathbf{Sh}(\mathbb{P}, \mathfrak{T}) \subseteq \mathbf{PSh}(\mathbb{P})$ *has a left adjoint, called sheafification.*

3 THE BIG PLAN

The starting point of this paper is a very well-known fact about sheaves. As their name implies, sheaves came historically first, and then only in a second time were generalized into presheaves. Surprisingly, it turned out that presheaves were not only providing a rich logical system known as a topos, but also that sheaves could be presented directy from within the presheaf topos itself. Pictorially, it means we have two models

$$\mathbf{Sh}(\mathbb{P}, \mathfrak{T}) \xrightarrow{\text{internal sheaf model}} \mathbf{PSh}(\mathbb{P}) \xrightarrow{\text{presheaf model}} \mathbf{Set}$$

where the historical or *external* sheaf model is given by the composition of those two interpretations[1]. Thanks to this factorization, one can then replace $\mathbf{PSh}(\mathbb{P})$ with an arbitrary topos [27]. The goal of this paper is to present the internal sheaf model as a translation between type theories rather than between topoi.

Can it be that hard? Despite their similarity, topoi are very weird as type theories. The first oddity is that topoi are defined by their models, rather than by their syntax. Instead, syntax is a derived notion known as the *internal language* of the topos. This model-theoretic stance has nasty

[1]We insist that these arrows are not functors, but instead interpretations of a theory into another theory.

consequences, because the ambient metatheory may surreptitiously flow into the semantics of topoi.

A second source of queasiness is that the internal logic of topoi validates by construction many principles that are absent from MLTT. Some of these principles could be considered desirable, but quite a few are outright anti-features of topos theory. In that regard, Grothendieck topoi are even more outrageous since they inherit a lot of properties from **Set**, the theory they are built on. We list below a few principles that hold in Grothendieck topoi.

Functional extensionality	$\mathtt{funext} : \Pi A\, B\, (f\ g : \Pi x : A.\, B\ x).\ (\Pi x : A.\, f\ x = g\ x) \to f = g$
Propositional extensionality	$\mathtt{propext} : \Pi (P\ Q : \mathtt{Prop}).\ (P \to Q) \to (Q \to P) \to P = Q$
Proof-irrelevance	$\mathtt{pi} : \Pi (P : \mathtt{Prop})\ (p\ q : P).\ p = q$
Unique choice	$\mathtt{uc} : \Pi (A : \mathtt{Type})\ (P : A \to \mathtt{Prop}).\ (\exists! x : A.\, P\ x) \to \Sigma x : A.\, P\ x$
Equality reflection	$\dfrac{\Gamma \vdash E : M = N}{\Gamma \vdash M \equiv N}$
Quotient types	…

Fig. 1. Major additional principles in presheaf topoi

Equality reflection is notoriously troublesome [18, 20, 31, 45] and its absence in intensional type theory is usually the main issue in this kind of endeavour. While we will explicitly require some extensionality principles in some definitions, we will not need reflection at all.

4 CUTTING THE MIDDLE-MAN

We present here the internal versions of the constructs from Section 2 in a type-theoretical way. We will work in an ambient type theory that we will call SetTT. It can be thought of as an extension of CIC with a few extensionality principles and quotient types. In particular it can be modeled inside any presheaf topos, and the intended standard model is some flavour of setoid type theory [1–3, 37]. Critically, SetTT has both a full-blown hierarchy of universes $\mathtt{Type}_i$ where $i \in \mathbb{N}$ and a universe of propositions Prop. As usual, we will rely on typical ambiguity although all our constructions can be explicitly annotated. The equality type will implicitly be interpreted as a proposition rather than a proof-relevant Type. For analysis purposes, we will explicitly mention when we rely on principles that are absent from CIC. Let us reassess that we will *never* rely on equality reflection in this paper.

Definition 4.1. A Lawvere-Tierney topology is a triple

- $\mathsf{T} : \mathtt{Prop} \to \mathtt{Prop}$
- $\eta^{\mathsf{T}} : \Pi P : \mathtt{Prop}.\, P \to \mathsf{T}\ P$
- $\gg\!=^{\mathsf{T}} : \Pi (P\ Q : \mathtt{Prop}).\, \mathsf{T}\ P \to (P \to \mathsf{T}\ Q) \to \mathsf{T}\ Q$

Lemma 4.2. *Lawvere-Tierney topologies in the presheaf model of* CIC *are exactly Grothendieck topologies.*

Proof. Sieves on p are (local) elements of Prop. The $\mathbb{P}$-indexed family of sets is the functional part of T, and the closure under pullback is saying exactly that it is natural. The two remaining axioms are literally interpreted as the maximal sieve condition and the union condition. □

The standard topos-theoretic phrasing is slightly different because it hardwires propext. In this setting, the above implications are typically written as equalities, and presented in a first-order way using the axiomatics of filters, i.e. as monotonicity w.r.t. predicate inclusion and closure under $\wedge$.

Written this way, it is obvious that a topology is just a monad over Prop. We do not require any of the expected equations because SetTT is implicitly assumed to satisfy proof-irrelevance. In this setting, all equations hold trivially.

Definition 4.3. We define the record type isSh_T as

$$\mathsf{isSh}_\mathsf{T}\ (A : \mathsf{Type}) : \mathsf{Type} := \mathsf{Sh}_\mathsf{T} \left\{ \begin{array}{l} \wr : \Pi(P : \mathsf{Prop}).\, \mathsf{T}\ P \to (P \to A) \to A \\ \mathsf{e} : \Pi P\ (p : \mathsf{T}\ P)\ (x : A).\, \wr\ P\ p\ (\lambda o : P.\, x) = x \end{array} \right\}$$

We will say that a type A is a T-sheaf whenever there is some term of type $\mathsf{isSh}_\mathsf{T}\ A$. This lenient phrasing is justified by the proposition below.

Lemma 4.4. *Assuming* funext *and* pi, $\mathsf{isSh}_\mathsf{T}\ A$ *is a mere proposition, i.e. there is a proof*

$$\Pi(A : \mathsf{Type})\ (p\ q : \mathsf{isSh}_\mathsf{T}\ A).\, p = q$$

Lemma 4.5. *Let* $A : \mathsf{Type}$. *A is a* T*-sheaf in the sense of Definition 4.3 in the presheaf model exactly when A is a* T*-sheaf in the sense of Definition 2.5.*

Proof. With the notations from Definition 4.3, a compatible family of A on P is just a (local) element of $P \to A$. Unicity of the glueing of the family ensures that the assignement is natural, and thus that we get a function in SetTT. The unicity requirement is translated straightforwardly into its presheaf equivalent. □

Note that the definition of isSh_T can be performed inside CIC, it does not require any extensionality principle.

The usual definition of sheafification in an elementary topos relies on Grothendieck's $(-)^+$ construction. The type itself can be expressed in CIC very easily, but in absence of unique choice it cannot be exploited. Even with unique choice, it corresponds to an encoding through equivalence classes and has no intrinsic computational content. For completeness of the exposition we write it explicitly below but will not use it further.

Definition 4.6. Let $\mathsf{Prop}_\mathsf{T} := \Sigma P : \mathsf{Prop}.\ (\mathsf{T}\ P \to P)$ and $A : \mathsf{Type}$, we define

$$A^+ := \Sigma P : A \to \mathsf{Prop}_\mathsf{T}.\, \mathsf{T}\ (\exists x : A.\, (\Pi y : A.\, (P\ y).\pi_1 \leftrightarrow \mathsf{T}\ (x = y))).$$

Assuming SetTT is a topos, then A^{++} is indeed the sheafification of A.

5 A SHEAFY TYPE THEORY

5.1 A Semi-formal Definition of ShTT

Before venturing any further into the type-theoretic exploration of sheaves, we will generalize a bit the structures we are paying attention to. Instead of picking a fixed topology T, we will suppose given two terms $\mathsf{I} : \mathsf{Type}$ and $\mathsf{O} : \mathsf{I} \to \mathsf{Prop}$, which will be respectively called the input and output types. As explained in Lemma 5.2, the topology case is a simple instance of this situation.

Definition 5.1. We define the record type isSh as

$$\mathsf{isSh}\ (A : \mathsf{Type}) : \mathsf{Type} := \mathsf{Sh} \left\{ \begin{array}{lcl} \wr & : & \Pi i : \mathsf{I}.\, (\mathsf{O}\ i \to A) \to A \\ \mathsf{e} & : & \Pi (i : \mathsf{I})\ (x : A).\, \wr\ i\ (\lambda o : \mathsf{O}\ i.\, x) = x \end{array} \right\}$$

For brevity, we will define $\square := \Sigma A : \mathsf{Type}.\, \mathsf{isSh}\ A$.

Lemma 5.2. *Up to a trivial curryfication, Definition 4.3 is obtained by setting*

$$\mathsf{I} \quad := \quad \Sigma P : \mathsf{Prop}.\, \mathsf{T}\ P \qquad\qquad \mathsf{O}\ (P, \_) \quad := \quad P.$$

Lemma 4.4 still holds with this slightly generalized formulation. From now on, we will say that a type A is a sheaf whenever there is a proof of $\mathsf{isSh}\ A$, and with a slight abuse of language, we will also call sheaves the inhabitants of $\square$.

*Definition 5.3 (*ShTT*).* We define ShTT as the type theory obtained through the syntactic translation interpreting types as sheaves. Semi-formally, if we present ShTT as a CwF,

- contexts and substitutions of ShTT are those of SetTT
- $\Gamma \vdash_{\mathsf{ShTT}} A$ type $:= \Gamma \vdash_{\mathsf{SetTT}} A : \square$
- $\Gamma \vdash_{\mathsf{ShTT}} M : A := \Gamma \vdash_{\mathsf{SetTT}} M : A.\pi_1$

It is legitimate to call this model a syntactic translation [18] because conversion in ShTT is interpreted as conversion in SetTT. In particular, it does not fall into the coherence hell and nowhere we rely on equality reflection to escape from it. As usual we abuse typical ambiguity, but the judgements can be easily annoted with explicit universe levels.

5.2 Dependent Products

Lemma 5.4. *Assuming* `funext`*, dependent products are closed under sheafness on the codomain. Formally, there is a term*

$$\Pi_\varepsilon : \Pi(A : \mathsf{Type})\ (B : A \to \mathsf{Type})\ (B_\varepsilon : \Pi x : A.\ \mathsf{isSh}\ (B\ x)).\ \mathsf{isSh}\ (\Pi x : A.\ B\ x)$$

Proof. The oracle function is given pointwise, i.e.

$$\wr_{(\Pi x : A.\ B)}\ (i : \mathsf{I})\ (k : \mathsf{O}\ i \to \Pi x : A.\ B) := \lambda(x : A).\ \wr_B\ i\ (\lambda o : \mathsf{O}\ i.\ k\ o\ x)$$

and uniqueness is proved similarly, although `funext` is crucially needed for it. □

We insist that this only holds thanks to `funext` because one needs to rewrite under a context in the unicity proof. This proposition ensures that ShTT has dependent products, regardless of the actual computational content of `funext`. Since the underlying first projection of the product type of ShTT is the one of SetTT, we will often confuse both.

5.3 Free Sheaves

In order to interpret inductive types, we need to define the sheafification functor in an internal way. The usual topos-theoretic definition relies on Grothendieck's $(-)^+$ functor. As already mentioned, in the context of type theory, this encoding behaves very badly because it boils down to an encoding of quotients using equivalence classes, which requires some form of choice to be effectively used. Fortunately, sheafification is by design a free construction, given by generators and relations. As such, it can be trivially given by a quotient inductive construction, which we give below. This type will be critical for the remainder of the paper.

$$\begin{aligned}
&\mathtt{Inductive}\ \text{Ш}\ (A : \mathsf{Type}) : \mathsf{Type} :=\\
&\mid \eta : \Pi x : A.\ \text{Ш}\ A\\
&\mid \wr : \Pi i : \mathsf{I}.\ (\mathsf{O}\ i \to \text{Ш}\ A) \to \text{Ш}\ A\\
&\mid \mathsf{e} : \Pi(i : \mathsf{I})\ (\hat{x} : \text{Ш}\ A).\ \wr\ i\ (\lambda o : \mathsf{O}\ i.\ \hat{x}) = \hat{x}.
\end{aligned}$$

This inductive type comes with a recursor specified at Figure 2. Here, we write $u =^e_P v$ for the heterogeneous equality between $u : P\ x$ and $v : P\ y$ along $e : x = y$ and we write

$$\mathsf{ap} : \Pi\{A : \mathsf{Type}\}\ \{B : A \to \mathsf{Type}\}\ (f : \Pi x : A.\ B\ x).\ \Pi\{x\ y : A\}\ (e : x = y).\ f\ x =^e_B f\ y$$

for the standard congruence combinator rewriting under an application.

This presentation is agnostic w.r.t. the choice of a semantics of equality. In presence of UIP, the third equation becomes trivial, at least propositionally so.

Lemma 5.5. *For any* $A : \mathsf{Type}$*,* Ш A *is a sheaf.*

Lemma 5.6. *Assuming* `funext`*,* Ш *is a monad. Moreover, sheaves and* Ш*-algebras are isomorphic.*

$$
\begin{aligned}
&\mathsf{rec}_{\text{Ш}} : \Pi(A : \mathsf{Type})\ (P : \text{Ш}\ A \to \mathsf{Type}).\\
&\qquad \Pi(p_\eta : \Pi x : A.\ P\ (\eta\ x)).\\
&\qquad \Pi(p_\wr : \Pi(i : \mathbb{I})\ (k : \Pi o : \mathrm{O}\ i.\ \text{Ш}\ A).\ (\Pi o : \mathrm{O}\ i.\ P\ (k\ o)) \to P\ (\wr\ i\ k)).\\
&\qquad \Pi(p_{\mathrm{e}} : \Pi(i : \mathbb{I})\ (\hat{x} : \text{Ш}\ A)\ (p : P\ \hat{x}).\ p_\wr\ i\ (\lambda o : \mathrm{O}\ i.\ \hat{x})\ (\lambda o : \mathrm{O}\ i.\ p) =_P^{\mathrm{e}\ i\ \hat{x}} p).\\
&\qquad \Pi(\hat{x} : \text{Ш}\ A).\ P\ \hat{x}\\
&\mathsf{rec}_{\text{Ш}}\ A\ P\ p_\eta\ p_\wr\ p_{\mathrm{e}}\ (\eta\ x) \quad \equiv \quad p_\eta\ x\\
&\mathsf{rec}_{\text{Ш}}\ A\ P\ p_\eta\ p_\wr\ p_{\mathrm{e}}\ (\wr\ i\ k) \quad \equiv \quad p_\wr\ i\ k\ (\lambda o : \mathrm{O}\ i.\ \mathsf{rec}_{\text{Ш}}\ A\ P\ p_\eta\ p_\wr\ p_{\mathrm{e}}\ (k\ o))\\
&\mathsf{ap}\ (\mathsf{rec}_{\text{Ш}}\ A\ P\ p_\eta\ p_\wr\ p_{\mathrm{e}})\ (\mathrm{e}\ i\ \hat{x}) \quad \equiv \quad p_{\mathrm{e}}\ i\ \hat{x}\ (\mathsf{rec}_{\text{Ш}}\ A\ P\ p_\eta\ p_\wr\ p_{\mathrm{e}})\ \hat{x}
\end{aligned}
$$

Fig. 2. Recursor for the Ш type

The reliance on `funext` in the above lemmas is due to that fact that the internal definition of monads and their algebras are fundamentally given in an extensional way. Without `funext`, one cannot even prove that the free algebra of an arbitrary monad is indeed an algebra. As a matter of fact, we give these results merely to provide intuition to the functional programmer, but we will not use them in a deep way. Furthermore, as we will see in Section 9, isSh is a better behaved way to define algebras of the free monad Ш in an intensional setting.

5.4 Inductive Types

Following the standard encoding of inductive types in a call-by-name monadic translation, we can interpret inductive types in ShTT simply as their SetTT equivalent, where we freely add the ϟ and e constructors. We give the example of unary natural numbers below. Note in passing that for recursive inductive types, this is not the same as wrapping the corresponding SetTT type into Ш, since the recursive subterms are also sheafified.

$$
\begin{aligned}
&\mathtt{Inductive}\ \mathbb{N}^{\text{Ш}} : \mathsf{Type} :=\\
&\mid \mathrm{O}^{\text{Ш}} : \mathbb{N}^{\text{Ш}}\\
&\mid \mathrm{S}^{\text{Ш}} : \mathbb{N}^{\text{Ш}} \to \mathbb{N}^{\text{Ш}}\\
&\mid \wr_{\mathbb{N}} : \Pi i : \mathbb{I}.\ (\mathrm{O}\ i \to \mathbb{N}^{\text{Ш}}) \to \mathbb{N}^{\text{Ш}}\\
&\mid \mathrm{e}_{\mathbb{N}} : \Pi(i : \mathbb{I})\ (n : \mathbb{N}^{\text{Ш}}).\ \wr\ i\ (\lambda o : \mathrm{O}\ i.\ n) = n.
\end{aligned}
$$

The recursive principle of this inductive type is similar to the one of Figure 2, so we will not write it explicitly here.

Now comes the bit of magic that makes sheaves very special. In general, monadic encodings of inductive types do not satisfy dependent elimination, due to the presence of non-standard or *effectful* terms [35]. As we will see below, the effects introduced by sheafification are innocuous enough not to introduce observable effects, which allows proving a form of dependent elimination.

We focus on the example of sheafified natural numbers to explain what is going on. We will spell out the computational content of the recursor in order to highlight how sheaves escape from the effectful pitfall.

Lemma 5.7. *Assuming* `funext` *and* `pi`*, there is a* SetTT *term*

$$
\begin{aligned}
&\mathsf{rec}_{\mathbb{N}}^{\text{Ш}} : \Pi(P : \mathbb{N}^{\text{Ш}} \to \square).\ (P\ \mathrm{O}^{\text{Ш}}).\pi_1 \to (\Pi n : \mathbb{N}^{\text{Ш}}.\ (P\ n).\pi_1 \to (P\ (\mathrm{S}^{\text{Ш}}\ n)).\pi_1) \to\\
&\qquad \Pi(n : \mathbb{N}^{\text{Ш}}).\ (P\ n).\pi_1
\end{aligned}
$$

that furthermore satisfies the definitional equalities

$$\begin{array}{lcl} \mathsf{rec}_{\mathbb{N}}^{\text{Ш}}\ P\ p_O\ p_S\ \mathsf{O}^{\text{Ш}} & \equiv & p_O \\ \mathsf{rec}_{\mathbb{N}}^{\text{Ш}}\ P\ p_O\ p_S\ (\mathsf{S}^{\text{Ш}}\ n) & \equiv & p_S\ n\ (\mathsf{rec}_{\mathbb{N}}^{\text{Ш}}\ P\ p_O\ p_S\ n). \end{array}$$

Proof. As expected, we prove it by induction over n. The cases for $\mathsf{O}^{\text{Ш}}$ and $\mathsf{S}^{\text{Ш}}$ are already given by the definitional equations, so we only have to handle the $\wr_{\mathbb{N}}$ and $\mathsf{e}_{\mathbb{N}}$ cases.

For the $\wr_{\mathbb{N}}$ case, assume $i : \mathsf{I}$ and $k : \mathsf{O}\ i \to \mathbb{N}^{\text{Ш}}$, we define

$$\mathsf{rec}_{\mathbb{N}}^{\text{Ш}}\ P\ p_O\ p_S\ (\wr_{\mathbb{N}}\ i\ k) : (P\ (\wr_{\mathbb{N}}\ i\ k)).\pi_1.$$

W.l.o.g. the recursive call provides us with a term of type $\Pi o : \mathsf{O}\ i.\ (P\ (k\ o)).\pi_1$. Since $\wr_{\mathbb{N}}$ corresponds to a (free) effect, a standard call-by-name semantics would mandate that we must use the algebra structure of P to propagate it outwards. We would thus have to prove $\Pi o : \mathsf{O}\ i.\ (P\ (\wr_{\mathbb{N}}\ i\ k)).\pi_1$. There is clearly a type mismatch w.r.t. the recursive call which seems to prevent its use. Yet, we can close the gap using two critical facts.

- First, $\mathsf{O}\ i$ is a proposition and thus proof-irrelevant. Together with `funext` we get a canonical proof of $\Pi(o : \mathsf{O}\ i).\ k = (\lambda(\_ : \mathsf{O}\ i).\ k\ o)$.
- Second, $\mathbb{N}^{\text{Ш}}$ enjoys an additional quotient rule $\mathsf{e}_{\mathbb{N}} : \Pi(i : \mathsf{I})\ (n : \mathbb{N}^{\text{Ш}}).\ \wr_{\mathbb{N}}\ i\ (\lambda o : \mathsf{O}\ i.\ n) = n$.

Combining those two ingredients, we get a canonical proof

$$r : \Pi(i : \mathsf{I})\ (k : \mathsf{O}\ i \to \mathbb{N}^{\text{Ш}})\ (o : \mathsf{O}\ i).\ \wr_{\mathbb{N}}\ i\ k = k\ o.$$

We thus simply define

$$\begin{array}{l} \mathsf{rec}_{\mathbb{N}}^{\text{Ш}}\ P\ p_O\ p_S\ (\wr_{\mathbb{N}}\ i\ k) := \\ \quad (P\ (\wr_{\mathbb{N}}\ i\ k)).\pi_2.\wr\ i\ (\lambda o : \mathsf{O}\ i.\ (r\ i\ k\ o)\ \#\ (\mathsf{rec}_{\mathbb{N}}^{\text{Ш}}\ P\ p_O\ p_S\ (k\ o))). \end{array}$$

where # is the transport operation.

For the $\mathsf{e}_{\mathbb{N}}$ case, given $i : \mathsf{I}$, $n : \mathbb{N}^{\text{Ш}}$ and $p : (P\ n).\pi_1$ we have to prove

$$(P\ (\wr_{\mathbb{N}}\ i\ (\lambda\_.\ n))).\pi_2.\wr\ i\ (\lambda\_.\ (r\ i\ (\lambda\_.\ n)\ o)\ \#\ p) =^{\mathsf{e}_{\mathbb{N}}\ i\ n} p$$

This is easily obtained through standard dependent equality manipulation. We do not even need to resort to UIP, assuming equivalence properties on `funext` is enough, but having set equality in `Prop` in SetTT makes this even more trivial. □

Although we do not provide the explicit proof, the same technique scales to the standard class of inductive types available in CIC. Thus, ShTT provides some form of dependent elimination. Contrarily to systems such as Baclofen Type Theory [33] where elimination must be restricted to strict predicates, ShTT allows arbitrary predicates, even though the interpretation of inductive types features constructors that do not correspond to a value in ShTT. Still, we emphasize that this is not a *bona fide* large elimination. The first quantification over P is not internal to ShTT, since there is no reason *a priori* for the type $\mathbb{N}^{\text{Ш}} \to \square$ to be a sheaf. To cut the suspense short, it is actually *not* in general. This failure will be discussed more thoroughly thereafter.

5.5 Universe of Propositions

Just as a sheaf topos is indeed a topos, one can equip ShTT with a universe of propositions.

Definition 5.8. We define the type of sheaf propositions $\circledast$ similarly to $\square$, i.e.

$$\circledast := \Sigma P : \mathsf{Prop}.\ (\Pi i : \mathsf{I}.\ (\mathsf{O}\ i \to P) \to P).$$

We do not have to require uniqueness of glueing because in presence of `pi`, it will hold trivially. Assuming `propext`, it is easy to turn this into a universe of propositions inside ShTT. Without loss of generality, we can always freely turn it into an algebra with $\amalg$, the only hard part being the existence of an element function.

Lemma 5.9. *Assuming* `propext`, *we define* $\mathsf{El}_* : \amalg \circledast \to \circledast$ *as*

$$\begin{aligned} \mathsf{El}_* \ (\eta \ P) &:= P \\ \mathsf{El}_* \ (\wr \ i \ K) &:= (\Pi o : \mathrm{O} \ i. \ \mathsf{El}_* \ (K \ o).\pi_1, \varphi \ i \ K) \end{aligned}$$

where $\varphi : \Pi(i : \mathrm{I}) \ (K : \mathrm{O} \ i \to \amalg \circledast) \ (j : \mathrm{I}).$
$(\Pi(q : \mathrm{O} \ j) \ (o : \mathrm{O} \ i). \ (\mathsf{El}_* \ (K \ o)).\pi_1) \to \Pi(o : \mathrm{O} \ i). \ (\mathsf{El}_* \ (K \ o)).\pi_1$
$\varphi := \lambda(i : \mathrm{I}) \ K \ (j : \mathrm{I}) \ k \ (o : \mathrm{O} \ i). \ (\mathsf{El}_* \ (K \ o)).\pi_2 \ j \ (\lambda(q : \mathrm{O} \ j). \ k \ q \ o).$

We must also provide a proof that the quotient is respected for the `e` *case, that is given* $i : \mathrm{I}$ *and* $A : \amalg \circledast$ *we must show*

$$\mathsf{El}_* \ (\wr \ i \ (\lambda\_. \ A)) =^{\mathrm{e} \ i \ A} \mathsf{El}_* \ A$$

By `pi`, *it is sufficient to provide an equality between the first components, i.e.*

$$(\Pi o : \mathrm{O} \ i. \ (\mathsf{El}_* \ A).\pi_1) = (\mathsf{El}_* \ A).\pi_1$$

which can be proved by a critical appeal to `propext`. *The reverse implication is trivial, and the forward one is a call to the oracle function of* $\mathsf{El}_* \ A$.

This forms a universe of propositions, which can be shown to be closed under all the usual connectives with the standard monadic encoding which we will not make explicit here. Using the dependent eliminator from Lemma 5.7, one can therefore define propositions by recursion. Hence, ShTT features a form of large elimination and goes beyond HOL.

5.6 Universe of Types?

Now, it is tempting to apply the same construction to $\square$, in order to show that ShTT inherits a hierarchy of universes from SetTT. In other words, we pick the free sheaf completion of $\square$ and try to write a function $\mathsf{El} : \amalg \square \to \square$ satisfying a few constraints. Unfortunately, this does not seem possible, as a well-known issue arises. Namely, the universe of sheaves is not *strictly* a sheaf [16].

This issue is surprisingly easy to explain in our setting. First, note that the equation on η is constrained to be the identity if we want to follow the weaning translation [33], which does not seem to be a strong requirement. So the main question is to define $\mathsf{El} \ (\wr \ i \ K) : \square$ given $i : \mathrm{I}$ and $K : \mathrm{O} \ i \to \amalg \square$. Following the `Prop` case, it is natural to pick

$$(\mathsf{El} \ (\wr \ i \ K)).\pi_1 := \Pi(o : \mathrm{O} \ i). \ (\mathsf{El} \ (K \ o)).\pi_1$$

where the second component is a proof of sheafness obtained out of Π_ε. Now, we simply have to prove that El function preserves the quotient, which is *almost* the case.

Lemma 5.10. *Assuming* `funext` *and* `pi`, *if* A : `Type` *is a sheaf then there is an isomorphism* $A \cong (\mathrm{O} \ i \to A)$ *for any* $i : \mathrm{I}$, *where the left-to-right function is weakening.*

Unluckily, this is not enough for our needs, since we must show that $A = (\mathrm{O} \ i \to A)$ for an arbitrary sheaf A, i.e. an *equality* rather than an *isomorphism*. There is no reason for this isomorphism to be an equality. Actually, in a wide class of models of CIC, including **Set** and the presheaf topoi, this equality is disprovable in the general case and only holds when A is proof-irrelevant. This mismatch between equality and isomorphism is the reason why *stacks* were introduced in the first place [16, 42] as more general notion than sheaves.

In our setting, it seems to be possible to side-step this issue using highly non-constructive principles in SetTT, such as global choice. Indeed, if one can pick an arbitrary representative for each

equivalence class of sheaves it is possible to work around the requirement of quotient preservation. This is not desirable, for two reasons. First, global choice does not hold in presheaf topoi, so this trick can only be used for sheaves in degenerated models such as **Set**. Second, it completely destroys the computational content of the theory since one cannot exhibit the representative, effectively turning El into a blocking axiom. This also means one must now rewrite explicitly to show that a code stands for some type, or said otherwise, introducing a coherence hell and the need for equality reflection.

We believe that universes are a critical feature of type theory. We will propose three different solutions which will be developed in Sections 8, 9 and 10.

6 A SUSTAINED LOOK AT ShTT

We now look at the metatheoretical properties of ShTT. The first result is a trivial adaptation of the topos-theoretic equivalent and is proved by a literal unfolding of the definitions.

Theorem 6.1. ShTT *is consistent iff* SetTT $\nvdash \text{Ш}\ \bot$.

By contrast, the second theorem is not usually discussed in the topos realm. The reason is that it does not make sense in a model-centric view of the world. Yet, it is of paramount importance to a type theorist.

Theorem 6.2. ShTT *does not enjoy canonicity even if* SetTT *does.*

Indeed, translated inductive types have now an additional canonical proof ϙ when compared to CIC. We find this to be highly problematic. In addition to their failure at interpreting large universes, this also means that they feature non-standard tree-like closed inductive terms. In fact, this structure has been known for a long time, because sheafification is essentially an intuitionistic variant of Herbrand trees [17, 39].

7 A COMPUTATIONAL INTERPRETATION OF ShTT

We elaborate on the above observation and give an intuitive explanation of the computational content of the sheaf model, or, rather a series of explanations.

The major observation is that, written as a tree, sheafification is a member of a large family of relatives known under many names [23, 28, 36, 43, 46], including but not limited to *Herbrand trees*, *interaction trees*, *free monad*, *dialogue monad*, etc. Amongst these tree-like structures, the free sheaf monad stands out because of three characteristic properties. Specifically, it is inductively defined, branches over a proposition and enjoys an additional quotient enforced by the e constructor.

As such, and similarly to interaction trees, sheafification can be seen as a very specific kind of *operating system*. A term $ϙ\ i\ k : \text{Ш}\ A$ corresponds to a system call where $i : \text{I}$ is the argument passed to the call, and $k : \text{O}\ i \to \text{Ш}\ A$ is the continuation that will be resumed when the system returns. This operating system is conditioned by the three characteristic properties. It is inductively defined, thus a program can only interact a finite number of times with the system, and in particular it cannot loop. It branches over a proposition, hence a program cannot computationally depend on the return value of a call, i.e. system calls are proof-irrelevant. Finally, the quotient $\text{e}\ i\ x : ϙ\ i\ (\lambda\_.\ x) = x$ is a standard property of labelled transition systems. It corresponds to bisimulation up to *silent* transitions. It equates a system call that does not use its return value, i.e. a silent transition, with the same term that did not perform the call. Both proof-irrelevance and quotienting are necessary to ensure that the operating system does not introduce observable effects in the sense of [35].

From the user point of view, sheafification in its restricted form can be seen as a way to enlarge the so-called singleton elimination criterion of CIC [14]. In CIC, there are three archetypical inductive

types in Prop that can be eliminated to Type: falsity, equality and accessibility. Sheafification over a monad T allows to relax this restriction to any proposition of the form T P where P is a singleton.

Finally, we would like to relate the computational content of sheafification with another closely related technique. It is folklore that Cohen's forcing can be presented as a sheaf model [27] for the double-negation monad. Miquel gave a computational interpretation of Cohen's forcing [29] from the point of view of classical realizability [24]. One would expect that Miquel's explanation is at least consistent with our description of sheafification.

Lemma 7.1. *Miquel's presentation of Cohen's forcing can be factorized as a non-standard Lafont-Streicher-Reus CPS [25] inside a presheaf model.*

Proof. In a presheaf model over a monoid, one can define a closed monoidal structure via Day convolution, i.e. the connectives $*$ and $-\!\!*$ from separation logic. The LRS CPS only requires a binary conjunction and a negation. Miquel's forcing is isomorphic to LRS over $- * -$ and $-\!\!*\bot$ for a cleverly chosen bottom type. □

In this presentation, there is no explicit tree construction, let alone a quotient. The reason for the simplicity of Miquel's presentation is that it is a model of HOL, rather than MLTT. The quotient construction becomes critical for the topos-theoretic not-so-large elimination, i.e. when constructing a proposition by induction over a term living in a type. Without large elimination, as in HOL, the Ш construction is useless and one can rely on what amounts to a monadic encoding, here a CPS. Similarly, Beth models [5] can also be presented as a monadic encoding over a presheaf model, without any trace of trees nor quotients.

8 THE HoTT PATH

There has been much work devoted to the study of generalizations of Lawvere-Tierney topologies in the context of higher topos theory, which was in turn expressed in the context of HoTT [10, 11, 38, 40]. We refer in particular to the comprehensive paper [40] which handles the much more general situation of *localization*. While the latter work subsumes this paper, it also hides the triviality of sheafification under the complexity of localization. In their framework, sheafification is an instance of an accessible topological modality. When restricting our attention to sheaves, much of the paraphernalia they need to design to make the generic localization work trivializes. As we will see, we will not even need to add higher equalities to the sheafness predicate to ensure that it is a mere proposition.

In this section, we replace SetTT with HoTT [44]. In particular, we assume univalence and the availability of higher inductive types. Equality is now assumed to live in Type, Prop is defined to be the type of mere propositions from a fixed universe, and QITs are interpreted as HITs. We will reuse the previous definitions verbatim up to this implicit relexification. The general miracle is that first, all the results from the SetTT case are lifted to this setting, and more importantly, in presence of univalence ShTT features universes.

Lemma 8.1. *For any A,* isSh A *is a mere proposition.*

The proof essentially the same as for Lemma 4.4, except that we now globally accept `funext` as a theorem of HoTT.

Lemma 8.2. *For any A, we have an equivalence*

$$\mathsf{isSh}\ A \cong (\Pi i : \mathsf{I}.\ \mathsf{isEquiv}\ A\ (\mathsf{O}\ i \to A)\ (\lambda(x : A)\ (\_ : \mathsf{O}\ i).\ x)).$$

The above result is once again a generalization of the SetTT case where equivalences are interpreted as isomorphisms. Nonetheless, in presence of univalence it has much farther-reaching consequences.

Lemma 8.3. *There exists a function* $\mathsf{El} : \text{Ш}\,\square \to \square$ *s.t.*

$$\begin{array}{lcl}(\mathsf{El}\ (\eta\ P)).\pi_1 & \equiv & P.\pi_1\\ (\mathsf{El}\ (\wr\ i\ K)).\pi_1 & \equiv & \Pi(o : \mathrm{O}\ i).\ \mathsf{El}\ (K\ o).\pi_1\end{array}$$

Proof. The η and $\wr$ cases are defined similarly to El_* and the failed attempt in SetTT. We face the same problem as before, i.e. given $A : \text{Ш}\,\square$ we must prove

$$\mathsf{El}\ (\wr\ i\ (\lambda\_.\ A)) =^{\mathrm{e}\ i\ A} \mathsf{El}\ A$$

Since isSh is a mere proposition, this amounts to

$$(\mathrm{O}\ i \to (\mathsf{El}\ A).\pi_1) =^{\mathrm{e}\ i\ A} (\mathsf{El}\ A).\pi_1$$

but this is a consequence of univalence and Lemma 8.2. □

In the above proof, we critically used the univalence principle to turn the equivalence $A \cong (\mathrm{O}\ i \to A)$ into an equality. This was not available in SetTT, where isomorphism does not entail equality in general. This is worth insisting on. The usual workaround of stacks consists in replacing the quotient by a generalized notion *externally*, i.e. in the **Set** target after the presheaf interpretation. Univalence allows solving the issue *internally*, i.e. we did not have to change anything in the definitions of sheafness and sheafification. What changed is the intended standard model of the ambient theory and the principles it provides.

Theorem 8.4. *There exists a syntactic sheaf model of* MLTT *into* HoTT.

This model was already briefly sketched in a generalized form in [40]. It is a trivial variant of the Eilenberg-Moore construction for dependenty type theory as found in the weaning [33] and exceptional [34] interpretations. Contrarily to the former it interprets unrestricted dependent elimination, and contrarily to the latter, it is consistent with the proviso of Theorem 6.1. Obviously, the pitfall of Theorem 6.2 applies and the result theory totally obliterates canonicity.

9 ACKNOWLEDGING SIDE-EFFECTS WITH BTT

We have explained previously that sheaves can be understood as an effectful translation that constrains the observation of effects in order to preserve dependent elimination. Instead of adding more equalities as in HoTT to enrich the structure of constrains, what about dropping them altogether? By doing so, we would embrace the possibility of side-effects in our type theory. This point of view is clearly heterodox for the topos-theoretic tradition.

As already mentioned, it turns out that allowing effects in type theory is not free. If we want to preserve the usual equational theory, we also have to weaken dependent elimination. The resulting theory is known as Baclofen Type Theory [33, 35]. The flip side is that we easily obtain an *effectful sheaf* syntactic model that features universes, and moreover the translation can be carried in CIC, i.e. we do not need any extensionality principle. The resulting model has already been described as the dialogue interpretation of BTT in [4]. For the sake of conciseness we will not spell it out in full here but only recall the key points.

Definition 9.1. Let $\mathsf{isSh}^{\mathcal{E}}\ (A : \mathsf{Type}) := \Pi i : \mathrm{I}.\ (\mathrm{O}\ i \to A) \to A$.

We will define the type of *effectful sheaves* as $\square^{\mathcal{E}} := \Sigma A : \mathsf{Type}.\ \mathsf{isSh}^{\mathcal{E}}\ A$. Contrarily to usual sheaves, this structure need not be a mere proposition. But it is an intensionally well-behaved way to define algebras of the dialogue monad

$$\mathtt{Inductive}\ \mathcal{D}\ (A : \mathsf{Type})\ : \mathsf{Type} := \eta^{\mathcal{E}} : A \to \mathcal{D}\ A\ \mid \wr^{\mathcal{E}} : \Pi i : \mathrm{I}.\ (\mathrm{O}\ i \to A) \to A.$$

That is, assuming extensionality principles one can show that effectful sheaves and $\mathcal{D}$-algebras are isomorphic, but in vanilla CIC a dependent product is an effectful sheaf as soon as its codomain is.

Theorem 9.2. *There exists a syntactic effectful sheaf model of* BTT *into* CIC *[4].*

The syntactic model simply interprets types as effectful sheaves. In particular, the underlying type of the universe is just $\mathcal{D}\ \square^{\mathcal{E}}$ and inductive types are interpreted freeely. One still has to define a function $\mathsf{El} : \mathcal{D}\ \square^{\mathcal{E}} \to \square^{\mathcal{E}}$. This part is underspecified, the major constraint being that $\mathsf{El}\ (\eta^{\mathcal{E}}\ A) \equiv A$. In [4], $\mathsf{El}\ (\wp^{\mathcal{E}}\ i\ k)$ is defined as a dummy inhabited type but we can also make the same choice as pure sheaves, i.e. $(\mathsf{El}\ (\wp^{\mathcal{E}}\ i\ k)).\pi_1 := \Pi o : \mathrm{O}\ i.\ \mathsf{El}\ (k\ o)$.

The main difference between MLTT and BTT lies in the specialization of inductive recursors in two different primitives. Both enjoy the same computational rules as their MLTT counterpart, but their type is different. The first one is non-dependent, and the second one must be restricted to strict or linear predicates [26, 30]. In this simple case, a function $f : A \to B$ is linear exactly when $f\ (\wp^{\mathcal{E}}_A\ i\ k) \equiv \wp^{\mathcal{E}}_B\ i\ (\lambda o : \mathrm{O}\ i.\ f\ (k\ o))$ for any $i : \mathbb{I}$ and $k : \mathrm{O}\ i \to A$. Note that when A and B are pure sheaves this equation holds propositionally under `funext` and `pi`. Thus, the pure sheaf quotient can be seen as a way to force all functions to be *propositionally* linear *a priori*, while BTT only requires it at dependent elimination time, but *definitionally* so.

10 YOU DO NOT NEED SHEAVES WITH `SProp`

In this section we completely shift our point of view, and rather ponder about the type-theoretical motivations for sheaves.

10.1 Admit It

Topologically, sheaves are a way to force locally true properties to become globally true. Rephrased type-theoretically, T-sheaves allow one to consider T-true propositions as if they were actually true, i.e. ShTT morally[2] validates the axiom $\Pi(P : \mathtt{Prop}).\ \mathsf{T}\ P \to P$. In our generalized case, this literally corresponds to forcing $\mathbb{2} := \Pi(i : \mathbb{I}).\ \mathrm{O}\ i$ to hold up to curryfication.

More formally, in SetTT one can write an evaluation function

$$\begin{array}{lcl} \varepsilon & : & \Pi\{A : \mathtt{Type}\}.\ \text{Ш}\ A \to \mathbb{2} \to A \\ \varepsilon\ (\eta\ x)\ \alpha & := & x \\ \varepsilon\ (\wp\ i\ k)\ \alpha & := & \varepsilon\ (k\ (\alpha\ i))\ \alpha \end{array}$$

(case for e omitted)

which turns structured calls to an implicit oracle $\alpha : \mathbb{2}$ into actual, structureless calls to this oracle. If we only consider the above motivations, the difference between $\text{Ш}\ A$ and $\mathbb{2} \to A$ are not significant, because both types validate the $\mathbb{2}$ axiom.

So, what happens if instead of doing such a complex and half-broken model as ShTT, we would simply consider the theory AxTT obtained by the trivial axiomatic syntactic model of CIC which simply adds a variable $\alpha : \mathbb{2}$ to all contexts? Such a model is readily available in all proof assistants, e.g. in Coq it amounts to postulating a global `Axiom`.

Is AxTT any better than ShTT? On the good side, AxTT is a full-blown model of CIC, in particular it interprets universes trivially. Since O is a `Prop`, it is also compatible with the erasure semantics of extraction and thus compatible with most flavours of realizability. On the bad side, consistency is harder to prove since one has to provide a distinct model, which sort of defeats the purpose of sheaves. Worse, canonicity is still broken in general as axioms block computation. Still, for some specific instances of O, one can recover canonicity. This is the case when all values of O are negated and AxTT is consistent [8].

[2] Assuming T can be reflected from SetTT to ShTT.

We propose below a generalization of this result by relying on a sort of *strict propositions* [1, 15]. In the remainder of this section, we will extend our ambient theory with a sort SProp as described in [15]. Intuitively, SProp is similar to the sort Prop of CIC, save for two major differences. First, all inhabitants of a type $A : \mathtt{SProp}$, hereafter called a strict proposition, are convertible. Second, to preserve decidability of the resulting theory the singleton elimination criterion of Prop is even stricter for SProp. Indeed, only empty types can be eliminated from SProp to another sort.

This restriction can be seen as a semantic trick to guarantee that only logical explosions can leak from strict propositions to relevant types. And as expected, it generalizes the canonicity result of [8].

Definition 10.1. We say that a context Γ is consistent whenever there is no term $\Gamma \vdash M : \bot$.

Definition 10.2. We say that a context is propositional if it is inductively built of strict propositions only.

Theorem 10.3 (Relative Canonicity). *If Γ is a consistent propositional context and $\Gamma \vdash M : \mathbb{N}$ then $M \equiv \mathsf{O}$ or there exists $\Gamma \vdash N : \mathbb{N}$ such that $M \equiv \mathsf{S}\ N$.*

Proof. The proof is a variant of the one from [8] and can be presented as a NbE argument. It was written in Agda as a trivial extension of the formalization of [15]. The result can be generalized to an arbitrary inductive type. See also [7] for a similar endeavour. □

It is now only a matter of putting all the pieces together. If we pick $\mathsf{O} : \mathsf{I} \to \mathtt{SProp}$ instead of returning a Prop, by adding a global axiom $\alpha : \Pi i : \mathsf{I}.\ \mathsf{O}\ i$ we still get a theory AxTT that models CIC, as well as strict propositions. But by Theorem 10.3, AxTT enjoys canonicity as long as the axiom is consistent. Therefore, if the only goal of sheaves is to force truth, AxTT is arguably better than ShTT. It is worth mentioning that presheaf models implement strict propositions when presented as prefascist types [32], so we can simply compose those two translations.

10.2 When ShTT Hits the Fan

If it is so easy to postulate axioms, why do we need sheaves at all? A priori, there is no reason for the two types $\text{Ш}\ A$ and $\mathcal{Q} \to A$ to be isomorphic. The first one exposes the list of calls to the oracle, while the second one does not. The call trace of $\text{Ш}\ A$ is not completely available though, because of the quotient. Most notably, one cannot observe the relative order between two calls, nor superfluous calls that do not use their return value. Still, by induction we merely know that there is a finite amount of calls for $\text{Ш}\ A$, but not for $\mathcal{Q} \to A$. This makes those two types different in some contexts. For instance, if we instantiate SetTT by a presheaf model where propositions are interpreted as opens of $\mathbb{R}$, with $\mathsf{I} := \mathbb{N}$ and $\mathsf{O}\ i :=]0, \frac{1}{i}[$, we have $\mathcal{Q} \cong 0$ so $\mathcal{Q} \to A$ is always trivial but $\text{Ш}\ A$ is not in general because $\bigcap_{i \in I} \mathsf{O}\ i$ is non-empty for any finite I.

We believe that this is the fundamental mismatch between ShTT and AxTT. Fortunately for us, what a strict axiom taketh, a strict axiom giveth. Such a mismatch between a higher-order function and a finite tree is a well-known phenomenon, which has a well-known solution, *bar induction* provided by the *fan functional*.

The historical presentation of bar induction is somewhat confusing. We will follow the modern presentation of Sterling [41], which is very easy to convey in a type-theoretic setting. We define at Figure 3 the $\mathcal{Q}$-continuity predicate $\mathscr{C} : \Pi\{A : \mathtt{Type}\}.\ (\mathcal{Q} \to A) \to \mathtt{SProp}$. Note that this predicate does not require O to be proposition.

Intuitively, this predicate captures the fact that a higher-order function has been obtained by the evaluation of a call tree. As expected, one can easily show the following in SetTT.

Lemma 10.4. *For any $x : \text{Ш}\ A$, $\varepsilon\ x : \mathcal{Q} \to A$ is $\mathcal{Q}$-continuous.*

$$\begin{array}{l}\texttt{Inductive}\ \mathscr{C}\ (A : \texttt{Type})\ : (\mathbb{2} \to A) \to \texttt{SProp} := \\ \mid \eta^{\mathscr{C}} : \Pi(x : A).\, \mathscr{C}\ A\ (\lambda\alpha : \mathbb{2}.\, x) \\ \mid \wp^{\mathscr{C}} : \Pi(i : \mathsf{I})\ (k : \mathsf{O}\ i \to \mathbb{2} \to A).\, (\Pi(o : \mathsf{O}\ i).\, \mathscr{C}\ A\ (\lambda\alpha : \mathbb{2}.\, k\ o\ \alpha)) \to \mathscr{C}\ A\ k\end{array}$$

Fig. 3. Inductive characterization of $\mathbb{2}$-continuity

What is missing, and as already explained not true in an arbitrary presheaf model, is that all functions are $\mathbb{2}$-continuous. A modern way to phrase Brouwer's bar induction is precisely that.

Definition 10.5. Let us locally pose $\mathsf{I} := \mathbb{N}$ and $\mathsf{O} := \lambda(\_ : \mathsf{I}).\, \mathbb{N}$. We define Brouwer's bar induction to be the principle $\texttt{bi} := \Pi(f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}).\, \mathscr{C}\ f$.

The validity of bar induction is a sensitive topic in MLTT, because it can make the theory inconsistent when phrased too proof-relevantly, in particular when expressed as a fan functional that computes the module of continuity of functions [12]. This paradox does not apply here because $\mathscr{C}$ is a strict proposition.

We believe that we can bridge the gap between AxTT and ShTT using a similar principle. We know that inductive types in ShTT are intepreted as free sheaves in SetTT. Hence, in theory we could reflect this into AxTT by adding bar induction axioms for every inductive type, i.e. if $\mathcal{I}$ is a first-order inductive type we have an axiom $\texttt{bi}_{\mathcal{I}} : \Pi(x : \mathbb{2} \to \mathcal{I}).\, \mathscr{C}\ x$.

We do not currently know whether such a set of axioms would be consistent in CIC. If it were the case, since the added axioms are propositional, by virtue of Theorem 10.3 the resulting variant of AxTT would still enjoy canonicity, while behaving essentially as ShTT.

We leave to further investigation the design of a theory that would validate these principles. In any case, the above discussion shows that there is a strong interplay between the nature of Prop and the kind of sheaves it allows. Also, while the use of sheaves to study continuity is folklore [13, 21, 47], the fact that sheaves are essentially enforcing a generalized continuity property does not seem to have been exposed so clearly.

11 CONCLUSION AND FUTURE WORK

We gave in this paper a straightforward description of sheaves as a specific kind of interaction trees. This description is internal to a generic type theory and can be written in CIC. By contrast, the existence of free sheaves relies on the availability of a particular quotient inductive type and, as usual for this kind of structures, introduces issues w.r.t. propositional equality. Through this presentation, the quasi-effectful nature of sheafification becomes obvious. Sheaves thus provide yet another insight into ruling out ill-behaved effectful proofs.

We believe that the use of strict propositions can lead to a generalization of ShTT in the style of [9], using more complex structures for the input and output types. More generally, the interaction of sheaves with definitional equality, and the existence of definitional sheaves is also an important question which deserves further study. We leave this for future work.

REFERENCES

[1] Thorsten Altenkirch. 1999. Extensional Equality in Intensional Type Theory. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 412–420. https://doi.org/10.1109/LICS.1999.782636

[2] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. 2019. Setoid Type Theory — A Syntactic Translation. In *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11825)*, Graham Hutton (Ed.). Springer, 155–196. https://doi.org/10.1007/978-3-030-33636-3_7

[3] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. Observational equality, now!. In *Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007*, Aaron Stump and Hongwei Xi (Eds.). ACM, 57–68. https://doi.org/10.1145/1292597.1292608

[4] Martin Baillon, Assia Mahboubi, and Pierre-Marie Pédrot. 2022. Gardening with the Pythia A Model of Continuity in a Dependent Setting. In *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference) (LIPIcs, Vol. 216)*, Florin Manea and Alex Simpson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 5:1–5:18. https://doi.org/10.4230/LIPIcs.CSL.2022.5

[5] Evert Willem Beth. 1956. *Semantic Construction of Intuitionistic Logic.* Amsterdam, Netherlands: Noord-Hollandsche Uitg. Mij.

[6] Olivia Caramello. 2010. The unification of Mathematics via Topos Theory. arXiv:1006.3930 [math.CT]

[7] Thierry Coquand. 2021. Reduction Free Normalisation for a proof irrelevant type of propositions. *CoRR* abs/2103.04287 (2021). arXiv:2103.04287 https://arxiv.org/abs/2103.04287

[8] Thierry Coquand, Nils Anders Danielsson, Martín Hötzel Escardó, Ulf Norell, and Chuangjie Xu. 2013. Negative consistent axioms can be postulated without loss of canonicity. https://www.cs.bham.ac.uk/~mhe/papers/negative-axioms.pdf

[9] Thierry Coquand and Guilhem Jaber. 2010. A Note on Forcing and Type Theory. *Fundamenta Informatica* 100, 1–4 (jan 2010), 43–52.

[10] Thierry Coquand, Bassel Mannaa, and Fabian Ruch. 2017. Stack semantics of type theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–11. https://doi.org/10.1109/LICS.2017.8005130

[11] Thierry Coquand, Fabian Ruch, and Christian Sattler. 2020. Constructive sheaf models of type theory. arXiv:1912.10407 [math.LO]

[12] Martín Hötzel Escardó and Chuangjie Xu. 2015. The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland (LIPIcs, Vol. 38)*, Thorsten Altenkirch (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 153–164. https://doi.org/10.4230/LIPIcs.TLCA.2015.153

[13] Michael Fourman and Martin Hyland. 1970. *Sheaf models for analysis.* 280–301. https://doi.org/10.1007/BFb0061823

[14] Gaëtan Gilbert. 2019. *A type theory with definitional proof-irrelevance.* Theses. Ecole nationale supérieure Mines-Télécom Atlantique. https://tel.archives-ouvertes.fr/tel-03236271

[15] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. 2019. Definitional proof-irrelevance without K. *PACMPL* 3, POPL (2019), 3:1–3:28. https://doi.org/10.1145/3290316

[16] Alexander Grothendieck. 1967. Éléments de géométrie algébrique : I. Le langage des schémas, Première partie. *Publications Mathématiques de l'IHÉS* 32 (1967), 5–361.

[17] Jacques Herbrand. 1930. *Recherches sur la théorie de la démonstration.* Number 110 in Thèses de l'entre-deux-guerres. http://www.numdam.org/item/THESE_1930__110__1_0/

[18] Martin Hofmann. 1997. *Extensional constructs in intensional type theory.* Springer.

[19] Guilhem Jaber, Gabriel Lewertowski, Pierre-Marie Pédrot, Matthieu Sozeau, and Nicolas Tabareau. 2016. The Definitional Side of the Forcing. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 367–376. https://doi.org/10.1145/2933575.2935320

[20] Guilhem Jaber, Nicolas Tabareau, and Matthieu Sozeau. 2012. Extending Type Theory with Forcing. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. IEEE Computer Society, 395–404. https://doi.org/10.1109/LICS.2012.49

[21] Peter T. Johnstone. 1979. On a Topological Topos. *Proceedings of The London Mathematical Society* (1979), 237–271.

[22] Peter T Johnstone. 2002. *Sketches of an elephant: a Topos theory compendium.* Oxford Univ. Press, New York, NY. https://cds.cern.ch/record/592033

[23] Oleg Kiselyov and Hiromi Ishii. 2015. Freer monads, more extensible effects. In *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell, Haskell 2015, Vancouver, BC, Canada, September 3-4, 2015*, Ben Lippmeier (Ed.). ACM, 94–105. https://doi.org/10.1145/2804302.2804319

[24] Jean-Louis Krivine. 2000. The Curry-Howard Correspondence in Set Theory. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*. IEEE Computer Society, 307–308. https://doi.org/10.1109/LICS.2000.855779

[25] Yves Lafont, Bernhard Reus, and Thomas Streicher. 1993. *Continuations Semantics or Expressing Implication by Negation.* Technical Report 9321. Ludwig-Maximilians-Universität, München.

[26] Paul Blain Levy. 2017. Contextual isomorphisms. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 400–414. https://doi.org/10.1145/3009837.3009898

[27] Saunders Mac Lane and Ieke Moerdijk. 1992. *Sheaves in Geometry and Logic*. Springer New York, New York, NY. http://link.springer.com/book/10.1007/978-1-4612-0927-0

[28] Conor McBride. 2015. Turing-Completeness Totally Free. In *Mathematics of Program Construction - 12th International Conference, MPC 2015, Königswinter, Germany, June 29 - July 1, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9129)*, Ralf Hinze and Janis Voigtländer (Eds.). Springer, 257–275. https://doi.org/10.1007/978-3-319-19797-5_13

[29] Alexandre Miquel. 2011. Forcing as a Program Transformation. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*. IEEE Computer Society, 197–206. https://doi.org/10.1109/LICS.2011.47

[30] Guillaume Munch-Maccagnoni. 2013. *Syntax and Models of a non-Associative Composition of Programs and Proofs. (Syntaxe et modèles d'une composition non-associative des programmes et des preuves)*. Ph. D. Dissertation. Paris Diderot University, France. https://tel.archives-ouvertes.fr/tel-00918642

[31] Nicolas Oury. 2005. Extensionality in the Calculus of Constructions. In *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3603)*, Joe Hurd and Thomas F. Melham (Eds.). Springer, 278–293. https://doi.org/10.1007/11541868_18

[32] Pierre-Marie Pédrot. 2020. Russian Constructivism in a Prefascist Theory. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 782–794. https://doi.org/10.1145/3373718.3394740

[33] Pierre-Marie Pédrot and Nicolas Tabareau. 2017. An effectful way to eliminate addiction to dependence. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005113

[34] Pierre-Marie Pédrot and Nicolas Tabareau. 2018. Failure is Not an Option - An Exceptional Type Theory. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10801)*, Amal Ahmed (Ed.). Springer, 245–271. https://doi.org/10.1007/978-3-319-89884-1_9

[35] Pierre-Marie Pédrot and Nicolas Tabareau. 2020. The fire triangle: how to mix substitution, dependent elimination, and effects. *Proc. ACM Program. Lang.* 4, POPL (2020), 58:1–58:28. https://doi.org/10.1145/3371126

[36] Maciej Piróg and Jeremy Gibbons. 2014. The Coinductive Resumption Monad. In *Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2014, Ithaca, NY, USA, June 12-15, 2014 (Electronic Notes in Theoretical Computer Science, Vol. 308)*, Bart Jacobs, Alexandra Silva, and Sam Staton (Eds.). Elsevier, 273–288. https://doi.org/10.1016/j.entcs.2014.10.015

[37] Loïc Pujet and Nicolas Tabareau. 2022. Observational Equality: Now For Good. *Proc. ACM Program. Lang.* POPL (2022).

[38] Kevin Quirin. 2016. *Lawvere-Tierney sheafification in Homotopy Type Theory. (Faisceautisation de Lawvere-Tierney en théorie des types homotopiques)*. Ph. D. Dissertation. École des mines de Nantes, France. https://tel.archives-ouvertes.fr/tel-01486550

[39] Lionel Rieg. 2014. *On Forcing and Classical Realizability. (Forcing et réalisabilité classique)*. Ph. D. Dissertation. École normale supérieure de Lyon, France. https://tel.archives-ouvertes.fr/tel-01061442

[40] Egbert Rijke, Michael Shulman, and Bas Spitters. 2020. Modalities in homotopy type theory. *Log. Methods Comput. Sci.* 16, 1 (2020). https://doi.org/10.23638/LMCS-16(1:2)2020

[41] Jonathan Sterling. 2021. Higher order functions and Brouwer's thesis. *J. Funct. Program.* 31 (2021), e11. https://doi.org/10.1017/S0956796821000095

[42] Thomas Streicher. 2005. Universes in Toposes. In *From sets and types to topology and analysis - Towards practicable foundations for constructive mathematics*, Laura Crosilla and Peter M. Schuster (Eds.). Oxford logic guides, Vol. 48. Oxford University Press.

[43] Wouter Swierstra. 2008. Data types à la carte. *J. Funct. Program.* 18, 4 (2008), 423–436. https://doi.org/10.1017/S0956796808006758

[44] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study.

[45] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. 2019. Eliminating reflection from type theory. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 91–103. https://doi.org/10.1145/3293880.3294095

[46] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2020. Interaction trees: representing recursive and impure programs in Coq. *Proc. ACM Program. Lang.* 4, POPL (2020), 51:1–51:32. https://doi.org/10.1145/3371119

[47] Chuangjie Xu. 2015. *A continuous computational interpretation of type theories*. Ph. D. Dissertation. University of Birmingham, UK. http://etheses.bham.ac.uk/5967/