In Cantor Space No One Can Hear You Stream

Martin Baillon[1], Assia Mahboubi[1,2], and Pierre-Marie Pédrot[1]

[1] INRIA
[2] Vrije Universiteit Amsterdam

Abstract. We revisit the famous notion of sheaves through the lens of type theory and side-effects. Using the language of MLTT, we show that they inductively approximate idealized functional objects as decision trees, realizing a generalized form of continuity. We materialize this intuition in $\mathsf{MLTT}^{\mathsf{f}}$, a case-study sheaf extension of MLTT with a Cohen real and leverage it to show uniform continuity of all MLTT functionals of type $(\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$. The latter results were mechanized in Rocq.

Keywords: type theory, continuity, sheaves, logical relation

1 Introduction

In classical general topology, a family $(O_i)_{i \in I}$ is an *open cover* of a space X if $X = \bigcup_{i \in I} O_i$ and X is *compact* if any potentially infinite open cover of X contains a finite cover. Sub-families of an open cover can be seen as approximations of the underlying space X, and compactness can be viewed as an abstract interface for a complete finitary description of X. Implementing this interface typically involves some form of choice principle, with various degrees of effectiveness.

Classical examples of compact spaces include the unit interval $[0, 1]$ for the usual topology on $\mathbb{R}$, and arbitrary products of compact spaces, for the product topology — a result known as Tychonoff's theorem. Equivalent to the axiom of choice in its full generality, this theorem has far-reaching consequences. For instance, it ensures the compactness of the space of functions $\{0, 1\}^{\mathbb{N}}$, called the *Cantor space* and denoted $\mathbb{N} \to \mathbb{B}$, as this space is the countable product of a discrete set. Tychonoff's theorem also proves that an arbitrary theory in propositional logic has a model as soon as any finite subset thereof has a model. This property also holds for first-order logic, and this generalization is equivalent to the Boolean prime ideal theorem, a weak form of the axiom of choice.

Modern classical topology gradually generalized the intuitions acquired on the study of concrete metric spaces, i.e. spaces with an underlying set of elements that can be equipped with a notion of distance, to more a abstract, point-free understanding of topology as relations between open sets. In particular, Serre's introduction of *sheaves* into algebraic geometry [60] brought a general machinery previously pertaining to topological algebra for deducing global properties from local ones. As of today, topos theory has promoted the study of sheaves as a field of its own, with applications in geometry, number theory or logic. Forcing is one such notorious application of sheaves in logic. It generalizes compactness as a

tool for building models of set theory, in particular for proving independence results like that of the axiom of choice or the continuum hypothesis [47].

About at the same time, different schools of constructive mathematics strived to avoid non-effective forms of choice principles in analysis, typically for implementing the compactness interface, thus eliminating the "convenient fictions" of classical topology. Brouwer's fan theorem is actually the effective, equivalent wording of the compactness of the Cantor space. Formal, point-free topology was later introduced by Martin-Löf and Sambin as a foundation of constructive topology, see e.g. Palmgren's survey on the foundations of homotopy theory [50]. Brouwer also investigated the *meaning* of defining a function on a domain such as the Cantor space, or on the Baire space of functions $\mathbb{N} \to \mathbb{N}$. Notably, Brouwer's continuity principle states that *any* function $F : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$, defined on the Baire space, can only rely on finitely many values of its input to compute its output, otherwise said, F is *continuous*. This principle should also hold for any $F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ by subtyping. Moreover, compactness of the Cantor space actually makes this finite collection of values *independent* from the considered input. This property is called *uniform continuity* of F and the collection of such values, a modulus of continuity. Note that Brouwer's continuity principle is outright wrong in a classical setting, as the later actually even allows for defining the nowhere continuous function on the Baire space that tests whether its input is the decimal representation of a rational number in $[0, 1[$. Brouwer's principle however holds for definable functions in certain systems, notably Gödel's System T [65], by careful inspection of computation trees for recursive functionals.

Brouwer's heterodox convictions have not pervaded his contemporary mathematical community. They however found echoes in the subsequent unfolding of theoretical computer science, whose central notion of algorithm is at odds with the classical mathematical notion of functions. For instance, the encoding of recursive functions by means of inductive trees has been rediscovered in several areas, from Kleene trees [45] to game semantics, and constructive analysis and topology underpin the foundations of programming languages for safe cyber-physical systems [61].

Maybe even more surprisingly, as a second validation of Brouwer's approach, intuitionistic logic ended up becoming one of the two pillars of the Curry-Howard correspondence, which identifies intuitionistic proofs with functional programs. It turned out to be an incredibly valuable tool to understand logic. More than just allowing to interpret proofs computationally, the correspondence can be followed along several axes. Some logical interpretations can be seen as program transformations, a process called indirect style. This contrasts with the so-called direct style approach, where originally non-provable axioms can be realized through new computational behaviours called *side-effects*. The most famous instance of such a relationship is undoubtedly the interpretation of classical logic, which identifies in indirect style the double-negation translation with continuation-passing style, and corresponds in direct style to the implementation of Peirce's law via the `callcc` operator inspired by the Scheme programming language [40].

In terms of expressivity, the apex of this correspondence is Martin-Löf type theory (MLTT), a system which makes no formal difference between proofs and programs. Both objects are represented by the very same structure of terms, and computation is built into the system thanks to dependent types. Since MLTT is a close cousin to topos theory, it is natural to wonder about the computational interpretation of sheaves. What effect do they perform? We unravel that they interpret programs as *decision trees* capturing a generalized notion of continuity.

The first contribution of this paper is $\mathsf{MLTT}^{\mathsf{f}}$, a direct style extension of MLTT to a prototypical sheaf construction known as *Cohen real* [23,24]. We use it to construct a syntactic model of MLTT via a logical relation à la Abel [2], and a mechanized version thereof in Rocq. We show that this model entails the uniform continuity of all MLTT functionals of type $(\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$. We argue this is a first step towards a more general understanding of sheaf type theories.

The description of the later contribution actually comes after a perhaps unusually expanded section devoted to motivating and situating the later sheaf model in the diverse landscape of related work. For this purpose, we string together observations and definitions that, albeit not novel, are scattered across different domains and explained in different languages. We advocate for the use of MLTT as a unifying device providing a clear and high-level description of the objects, stripped from encodings or low-level considerations. This unified perspective is the second contribution of this paper. One source of inspiration is the original formulation by Escardó of Brouwer trees as a simple inductive type, and its clever use to prove continuity of System T functionals [31] that was generalized to a weak variant of MLTT by Baillon et al. [12] Escardó's technique of effectful forcing indeed can be seen as an analogue to sheaf-theoretic forcing techniques, transposed to the realm of type theory. Another important source is Rijke, Schulman and Spitters' article [59] which describes a generalization of sheaves called *localization* in the context of homotopy type theory, yet without emphasis on computation. Coquand et al. also gave a very similar account [25]. A good chunk of the computational behaviour of sheaves have been explained in the setting of Beth-style realizability in a long series of papers by Rahli et al., the most recent being [18], but the importance of the tree-like nature is not recognized. Finally, the series of work by Coquand and collaborators on dynamical methods [20] help us bridge the gap between our type-theoretical point of view and the more common mathematical usage of sheaf-like interpretations.

This paper is thus organized in two parts. The first part consists of Section 2, which discusses the use of sheaves for building models of type theory. We present sheaves in indirect style synthetically in the language of MLTT, getting rid of the presheaf middleman. This allows to pinpoint where the tree-like nature of sheaves arises and to relate them to other constructions from programming language theory. Furthermore, it also highlights an extremely specific property of the side-effects they provide which distinguishes them from more traditional effects.

The second part is devoted to the definition of $\mathsf{MLTT}^{\mathsf{f}}$ and to the study of its metatheoretical properties. The theory itself is defined in Section 3 and we describe a complete realizability model for it in Section 4. We use this model

to derive our main result, i.e. that MLTT functionals on the Cantor space are uniformly continuous and thus cannot observe the potential infinity of their argument. Therefore, in Cantor space, no one can hear you stream. Hyperlinks point to relevant parts of the Rocq mechanization of this model.

2 Sheaves

This section argues that sheaves constitute a very nice kind of side-effect, much better behaved than double-negation interpretations. We defend our case by untangling their traditional definition from the notion of presheaves and rephrasing it directly in MLTT. By expressing them synthetically, we revisit various well-known results with a computational point of view.

2.1 Sheaves and Presheaves

We first recall here the usual presentation of sheaves in a set-theoretical but otherwise underspecified metatheory. For a more comprehensive overview, we refer to the MacLane and Moerdijk's reference book [47]. Let us fix a category $\mathbb{P}$. For any of its objects p, q, written $p, q \in \mathbb{P}$, we denote $\mathbb{P}(q, p)$ the class of morphisms with source q and target p. A sieve on $p \in \mathbb{P}$ is a set of morphisms with target p that is closed under precomposition. We write $\mathfrak{S}_p$ for the set of sieves on p and $\Sigma q \in \mathbb{P}.\, \mathbb{P}(q, p)$ for the full sieve on p, as we rather see a point in a sieve on p as a pair (q, α) of an object $q \in \mathbb{P}$ and a morphism $\alpha \in \mathbb{P}(q, p)$.

Definition 1. *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 properties.*

1. *If $P \in \mathfrak{T}_p$ and $\alpha \in \mathbb{P}(q, p)$ then $\alpha^* P \in \mathfrak{T}_q$ where $\alpha^* P$ is the pullback of P.*
2. *The full sieve on p belongs to $\mathfrak{T}_p$.*
3. *If $P \in \mathfrak{T}_p$, $Q \in \mathfrak{S}_p$ and $P \subseteq \bigcup_{q \in \mathbb{P}} \{(q, \alpha) \mid \alpha^* Q \in \mathfrak{T}_q\}$ then $Q \in \mathfrak{T}_p$.*

Grothendieck topologies provide an abstract generalization of the notion of an open cover and an element $\mathfrak{T}_p$ of a Grothendieck topology $\mathfrak{T}$ is thus often called a covering. Axiom 2. states that the trivial covering is a covering and Axiom 3. that a covering of coverings is still a covering. We fix a topology $\mathfrak{T}$ in the rest of this section. Recall that the category of *presheaves over* $\mathbb{P}$ is defined as $\mathbf{Psh}(\mathbb{P}) := \mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$ and that it forms a topos [47].

Definition 2. *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 $\beta \in \mathbb{P}(r, q)$.*

Definition 3. *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 1. *The full subcategory $\mathbf{Sh}(\mathbb{P}, \mathfrak{T})$ of $\mathfrak{T}$-sheaves is a topos. Furthermore, the inclusion $\mathbf{Sh}(\mathbb{P}, \mathfrak{T}) \subseteq \mathbf{PSh}(\mathbb{P})$ has a left adjoint called sheafification.*

Sheaf toposes are such a commonplace notion and a historical artifact that they have a name of their own. They are called *Grothendieck toposes*.

Let us step back for a second to assess what a topos is from a type-theoretic point of view. A topos is a kind of type theory, or to be more correct a *model* of certain type theory. This type theory is dubbed the *internal language* of a topos. Fully describing it would lead us to too far, but let us sketch it here briefly. When compared to MLTT, the type theory arising from a topos is weird, being both stronger and weaker. It is stronger because it features an impredicative universe Prop of proof-irrelevant propositions, bringing it closer to CIC. It also inherits extensionality principles from **Set**, including equality reflection, thus making it an *extensional* type theory. As a consequence, it validates function extensionality (`funext`) and uniqueness of identity proofs (UIP). In addition, it also validates propositional extensionality (`propext`). Finally, it interprets quotients together with a strong form of unique choice. On the weaker side, the usual formulation of toposes lacks proper universes Type and hence does not feature in general a true notion of large elimination, where types can be constructed by pattern-matching on terms. Instead, thanks to Prop, they only allow for *propositions* being constructed by case-analysis. This is expressive enough to side-step the lack of expressivity of pure type systems such as the Calculus of Constructions, which cannot prove $0 \neq 1$, but is still far from the rich setting of MLTT.

2.2 Sheaves without Presheaves

The notions exposed above can in fact be rephrased in a purely internal way, a phenomenon that was already observed early on. Indeed, since $\mathbf{Psh}(\mathbb{P})$ is a topos, as we explained above it hosts a rich logical system by itself. Assuming Grothendieck universes in the host set theory, we can even show that it also features actual universes in the internal language, making it a model of MLTT. As a result, everything we write in MLTT can be interpreted straightforwardly into $\mathbf{Psh}(\mathbb{P})$. In this section, we will work in PshTT, an ambient type theory extending MLTT that is voluntarily kept underspecified. Its intended semantics is given by presheaf models, so PshTT will provide us in particular with a small universe of definitional proof-irrelevant propositions Prop [35]. We will liberally and silently rely on the aforementioned extensionality principles that hold there. Note that we will refrain from relying on equality reflection though, so some flavour of observational type theory would be enough to capture PshTT [7].

The goal of this section is to replay all the set-theoretic definitions from Section 2.1 directly in PshTT so that the type-theoretic definition will coincide with the set-theoretic one when interpreted into a presheaf model. This endeavour is not really novel, but our insistence to use dependent type theory will unravel a few properties that are not so obvious when using e.g. first-order based internal languages. Much closer to our approach is the Rijke, Shulman and Spitters' significant work on modalities in HoTT [59] which tackles amongst others a more generic construction known as localization. Sheafification is a degenerate form of localization, so our definitions are a simplification of theirs. Yet the fact it is

much simpler also emphasizes computational properties that are not apparent in their article. Without further ado, let us now describe sheaves synthetically.

Definition 4. *A Lawvere-Tierney topology is a term* $\mathsf{T} : \mathsf{Prop} \to \mathsf{Prop}$ *together with two terms* $\eta_{\mathsf{T}} : \Pi(P : \mathsf{Prop}).\, P \to \mathsf{T}\ P$ *and* $\gg\!=_{\mathsf{T}} : \Pi(P\, Q : \mathsf{Prop}).\, \mathsf{T}\ P \to (P \to \mathsf{T}\ Q) \to \mathsf{T}\ Q$.

It is reasonably easy to observe that a Lawvere-Tierney topology in the presheaf model is exactly a Grothendieck topology. The family of sieves $\mathfrak{T}$ together with condition 1. correspond to the term T itself, condition 3. corresponds to $\gg\!=_{\mathsf{T}}$ and 2. is internally a proof of $\mathsf{T}\ \top$, which is equivalent to η_{T} up to propositional extensionality. Through this presentation, it is immediate that T is just a monad on Prop. All equations hold trivially as Prop is proof-irrelevant in PshTT.

Definition 5. *A type* $A : \mathsf{Type}$ *is a* T*-sheaf if it is equipped with two terms*

- $\mathfrak{f}_A : \Pi(P : \mathsf{Prop}).\, \mathsf{T}\ P \to (P \to A) \to A$;
- $e_A : \Pi(P : \mathsf{Prop})\,(p : \mathsf{T}\ P)\,(x : A).\, \mathfrak{f}_A\ P\ p\ (\lambda(q : P).\, x) =_A x$.

Again, we can read back the historical presentation in the internal statement, namely $\mathfrak{f}_A$ gives the existence of glueing and e_A corresponds to its uniqueness.

Sheafification is somewhat horrendous when performed analytically, as one has to pay a lot of attention to low-level details. There also exists an internal way to define it from topos theory called Grothendieck's $-^+$ construction [47], which is similar to an impredicative encoding. Such encodings suffer from a lot of issues in type theory [51], and the $-^+$ construction is no exception. Fortunately for us we can define sheafification in a direct way assuming our ambient type theory to be expressive enough. It is indeed a free functor, so it can be described very easily as a quotient inductive type [6] (QIT).

Definition 6. *We define sheafification as the QIT*[3] $\mathcal{S}$ *in Figure 1.*

Here, the internal approach really shines. Assuming that our internal notion of sheaves is indeed the right one, there is nothing to show about the interpretation of $\mathcal{S}$ through a presheaf model. By construction, it is the free sheaf arising from some type. Obviously, the hard part is that one has to show that QITs exist in presheaf models. Albeit technical, this is actually doable as QITs can be encoded in extensional type theory using basic inductive and quotient types [43,33], and presheaf models provide all of this material.

Lemma 1. $\mathcal{S}$ *defines a monad on* Type *with the obvious combinators.*

It is worth recalling some abstract properties of sheaves in this synthetic setting. We have function extensionality at hand, so all usual categorical definitions work as expected. We swear to the reader that this is the only place in this paper where we insist so much on categorical notions. First, let us state the obvious.

[3] For completeness, we also write the recursor equation for the quotient constructor as if it were a higher inductive type (HIT) [66]. For QITs, this equation holds trivially.

$$
\begin{array}{l}
\texttt{Inductive}\ \mathcal{S}\ (A : \mathsf{Type}) : \mathsf{Type} := \\
\mid \eta_{\mathcal{S}} : A \to \mathcal{S}\,A \\
\mid \mathfrak{f}_{\mathcal{S}} : \Pi(P : \mathsf{Prop}).\, \mathsf{T}\ P \to (P \to \mathcal{S}\,A) \to \mathcal{S}\,A \\
\mid e_{\mathcal{S}} : \Pi(P : \mathsf{Prop})\,(p : \mathsf{T}\ P)\,(x : \mathcal{S}\,A).\, \mathfrak{f}_{\mathcal{S}}\ P\ p\ (\lambda(q : P).\, x) =_{\mathcal{S}\,A} x.
\end{array}
$$

$$
\begin{array}{l}
\mathcal{S}_{\mathtt{rec}} : \Pi(A : \mathsf{Type})\,(R : \mathcal{S}\ A \to \mathsf{Type})\,(r_\eta : \Pi(x : A).\, R\ (\eta_{\mathcal{S}}\ x)) \\
\qquad (r_{\mathfrak{f}} : \Pi P\,(p : \mathsf{T}\ P)\,(k : P \to \mathcal{S}\,A)\,(r : \Pi(p : P).\, R\ (k\ p)).\, R\ (\mathfrak{f}_{\mathcal{S}}\ P\ p\ k)) \\
\qquad (r_e : \Pi P\,(p : \mathsf{T}\ P)\,(x : \mathcal{S}\,A)\,(r : R\ x). \\
\qquad\quad (e_{\mathcal{S}}\ P\ p\ x) \# r_{\mathfrak{f}}\ P\ p\ (\lambda(q : P).\, x)\ (\lambda(q : P).\, r) = r). \\
\qquad \Pi(s : \mathcal{S}\,A).\, R\ s
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{S}_{\mathtt{rec}}\ A\ R\ r_\eta\ r_{\mathfrak{f}}\ r_e\ (\eta_{\mathcal{S}}\ x) & \equiv & r_\eta\ x \\
\mathcal{S}_{\mathtt{rec}}\ A\ R\ r_\eta\ r_{\mathfrak{f}}\ r_e\ (\mathfrak{f}_{\mathcal{S}}\ P\ p\ k) & \equiv & r_{\mathfrak{f}}\ P\ p\ k\ (\lambda(p : P).\, \mathcal{S}_{\mathtt{rec}}\ A\ R\ r_\eta\ r_{\mathfrak{f}}\ r_e\ (k\ p)) \\
\mathtt{ap}\ (\mathcal{S}_{\mathtt{rec}}\ A\ R\ r_\eta\ r_{\mathfrak{f}}\ r_e)\ (e_{\mathcal{S}}\ P\ p\ x) & \equiv & r_e\ P\ p\ x\ (\mathcal{S}_{\mathtt{rec}}\ A\ R\ r_\eta\ r_{\mathfrak{f}}\ r_e\ x)
\end{array}
$$

Fig. 1. Sheafification and its associated recursor

Lemma 2. *A type A is a T-sheaf exactly when it is an $\mathcal{S}$-algebra. Moreover, being a T-sheaf is a mere proposition in the* HoTT *sense.*

The latter property makes the monad $\mathcal{S}$ idempotent, which has a lot of far-reaching consequences. One of them is that any function $f : A \to B$ between two sheaves becomes an algebra morphism. This will be important in what follows.

2.3 Oracles as Logical Operating Systems

Before further studying sheaves as computational objects, we first generalize the notion of topology to an extreme without losing their fundamental properties. This little shift of perspective will make the relationship of sheafification with well-known computational objects unmistakable.

Definition 7. *A logical operating system (LOS) is given by two terms* $\mathsf{I} : \mathsf{Type}$ *and* $\mathsf{O} : \mathsf{I} \to \mathsf{Prop}$.

The name might seem mysterious at first, but it will become clear soon. The types I and O stand respectively for *input* and *output*. We will now define a slight variant of sheafification from the previous section relying on a LOS (I, O) rather than a topology. We reuse the same notations for uniformity.

Definition 8. *From now on, we will redefine the type $\mathcal{S}$ from Definition 6 as*

$$
\begin{array}{l}
\texttt{Inductive}\ \mathcal{S}\ (A : \mathsf{Type}) : \mathsf{Type} := \\
\mid \eta_{\mathcal{S}} : A \to \mathcal{S}\,A \\
\mid \mathfrak{f}_{\mathcal{S}} : \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to \mathcal{S}\,A) \to \mathcal{S}\,A \\
\mid e_{\mathcal{S}} : \Pi(i : \mathsf{I})\,(x : \mathcal{S}\,A).\, \mathfrak{f}_{\mathcal{S}}\ i\ (\lambda(o : \mathsf{O}\ i).\, x) =_{\mathcal{S}\,A} x.
\end{array}
$$

Any $\mathsf{T} : \mathsf{Prop} \to \mathsf{Prop}$ gives rise to a LOS by setting $\mathsf{I} := \Sigma(P : \mathsf{Prop}).\, \mathsf{T}\ P$ and $\mathsf{O}\ (P, p) := P$. Through this encoding the above definition of $\mathcal{S}$ is isomorphic to the one from Section 2.2. We can similarly define sheafness w.r.t. a LOS.

Definition 9. *A type* $A : \mathsf{Type}$ *is an* (I, O)*-sheaf if it is equipped with two terms*

- $\mathsf{f}_A : \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to A) \to A$;
- $e_A : \Pi(i : \mathsf{I})\, (x : A).\, \mathsf{f}_A\ i\ (\lambda(o : \mathsf{O}\ i).\, x) =_A x$.

Note that we actually do not need the monadic closure properties on T for $\mathcal{S}$ to behave well, they are freely added by the QIT. The reason it is a requirement in the historical presentation is mostly because the $\mathcal{S}$ type constructor is traditionally defined via an impredicative encoding that requires collapsing together all f constructors. Indeed, using `propext` and the quotient $e_{\mathcal{S}}$ one can replace two subsequent calls to f first on $P : \mathsf{Prop}$ and then on $Q : P \to \mathsf{Prop}$ by one call on $\exists p : P.\, Q\ p$ thanks to the monadic structure. All calls are compacted this way by recursion on the list of questions, where the nullary case is given by $\top : \mathsf{Prop}$ and η_{T}.

We call (I, O) an *operating system* because it morally corresponds to the abstract interface for *system calls* (syscalls). That is, a term $i : \mathsf{I}$ codes for a syscall number with its arguments while $\mathsf{O}\ i$ is the return type of this call. For some sheaf A, the term f_A provides a handler that is not unlike the actual low-level implementation of operating systems where syscalls are implemented using some form of delimited continuations [44], i.e. objects of type $\mathsf{O}\ i \to A$.

As a matter of fact, this observation has been put to practical use with *interaction trees* [68] which are a way to encode a vast range of I/O effects in type theory. There are several variants of interaction trees, but they all share a striking similarity with our $\mathcal{S}$ type. The most basic kind is defined as the type

$$
\begin{array}{l}
(\mathtt{Co})\mathtt{Inductive}\ \mathcal{T}\ (A : \mathsf{Type}) : \mathsf{Type} := \\
\mid \eta_{\mathcal{T}} : A \to \mathcal{T}\ A \\
\mid \mathsf{f}_{\mathcal{T}} : \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to \mathcal{T}\ A) \to \mathcal{T}\ A
\end{array}
$$

for some $\mathsf{I} : \mathsf{Type}$ and $\mathsf{O} : \mathsf{I} \to \mathsf{Type}$. The major differences with $\mathcal{S}$ are the following. First $\mathcal{T}$ is usually coinductive rather than inductive, but the only reason for that is it is also meant to encode non-terminating programs. One could use an inductive variant if they did not care about potential non-termination. Second, the output type O is proof-relevant, as users care about return values of their syscalls. Finally, and perhaps more importantly, there is no quotient in sight, as the order of operations matters critically in an I/O world.

The two latter points are the fundamental reason why in our setting we talk about a *logical* operating system. With sheaves, there is no way to extract proof-relevant content out of a syscall by virtue of O returning propositions, and furthermore the quotient prevents one to observe not only the order of syscalls but also their multiplicity and even useless calls.

Lemma 3. *For any sheaf* A*, we can prove the following equalities in* PshTT*:*

$$
\begin{array}{ll}
i : \mathsf{I}, x : A & \vdash \mathsf{f}_A\ i\ (\lambda(o : \mathsf{O}\ i).\, x) = x \\
i : \mathsf{I}, j : \mathsf{I}, x : \mathsf{O}\ i \to \mathsf{O}\ j \to A & \vdash \mathsf{f}_A\ i\ (\lambda(o_i : \mathsf{O}\ i).\, \mathsf{f}_A\ j\ (\lambda(o_j : \mathsf{O}\ j).\, x\ o_i\ o_j)) = \\
& \qquad \mathsf{f}_A\ j\ (\lambda(o_j : \mathsf{O}\ j).\, \mathsf{f}_A\ i\ (\lambda(o_i : \mathsf{O}\ i).\, x\ o_i\ o_j))
\end{array}
$$

This is actually quite surprising when considering (I, O) as introducing side-effects. Somehow, sheaves provide a kind of *unobservable* side-effect in the sense of [54]. There are new normal forms represented by uninterpreted calls to the oracle via $\mathtt{f}_{\mathcal{S}}$, but they cannot be exploited internally because of the quotient. We defer a proper discussion of this phenomenon to Section 2.4.

The relationship between operating systems and forcing was already observed by Krivine and Miquel in classical realizability [49]. It is part of the folklore of the French school of type theory, although there is little published material on the topic, except maybe vulgarization [55]. Regardless, set-theoretic forcing can be seen as a specific case of sheafification for the double negation monad [47]. It is interesting to observe that Miquel's computational interpretation is state-like rather than tree-like. The deep reason for this is that Miquel's paper only models F_ω, which lacks dependent elimination. Therefore, it can be re-explained as the composition of the double-negation translation with the presheaf translation, which is strictly weaker than $\neg\neg$-sheaves.

2.4 Sheaves in Homotopy Type Theory

An interesting question is the type of type theory that one can obtain from our synthetic sheaf interpretation. More precisely, we will be looking at syntactic models, i.e. models defined as mere syntactic translations between theories that interpret conversion in the source as conversion in the target. We know that usual sheaves result in a Grothendieck topos, but as mentioned this is hardwiring extensionality both in the source and in the target. What if we want to get an intensional theory instead? It turns out that synthetic sheaves go quite a long way. We sketch the syntactic-synthetic sheaf model in this section. Once again, this is a simplification and a reformulation of the localization paper [59].

Infrastructure Following the category-with-family (CwF) setting, we will be translating contexts into contexts, types into synthetic sheaves, i.e.

$$\mathsf{Typ}^{\mathcal{S}}(\Gamma) := \Sigma[A] : \mathsf{Typ}(\Gamma).\, \mathsf{isSh}\ [A]$$

and terms as terms of the underlying type. Above $\mathsf{isSh}\ A$ is the data for A from Definition 9. We assume a target CwF that is rich enough, i.e. basically a model of PshTT. This construction preserves strictness of substitution from the target, since this is just a subuniverse model as in [59]. We write terms and types in the target directly using the MLTT syntax and consciously confuse the target theory with the ambient type theory, effectively working in the standard CwF. In particular, $\mathsf{Typ}^{\mathcal{S}}$ can be conflated with the record type $\mathsf{Type}^{\mathcal{S}}$ defined below.

$$\mathsf{Type}^{\mathcal{S}} := \left\{ \begin{array}{l} \mathtt{el} : \mathsf{Type}; \\ \mathtt{f} : \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to \mathtt{el}) \to \mathtt{el}; \\ e : \Pi(i : \mathsf{I})\, (x : A).\, \mathtt{f}\ i\ (\lambda(o : \mathsf{O}\ i).\, x) = x; \end{array} \right\}$$

For readability, we will implicitly use the $\mathtt{el}$ projection to cast a $\mathsf{Type}^{\mathcal{S}}$ to a Type and identify $[\cdot]$ with $\mathtt{el}$.

Π-types The first meaningful type former one usually consider in dependent type theory is the dependent product. Assuming `funext` in the target, one can straightforwardly show that Π-types are inherited from it, i.e. with a bit of abuse of notation, we pick $[\Pi(x : A).B] := \Pi(x : [A]).[B]$. Indeed, $\mathsf{isSh}\ ([\Pi(x : A).\, B])$ holds as soon as we have $\Pi(x : [A]).\, \mathsf{isSh}\ [B]$, the algebra structure being given pointwise as in any call-by-name model. There is really no more to say about it.

Inductive types Inductive types are much more fascinating. A Grothendieck topos readily interprets inductive types, so they should have a syntactic equivalent. By relying on QITs in the synthetic approach, it becomes crystal clear. Essentially, we just take the original inductive type $\mathcal{I}$ and freely adjoin it the $\mathsf{f}_{\mathcal{I}}$ and $e_{\mathcal{I}}$ constructors. This construction scales to all inductive types, but we concentrate on $\mathbb{N}$ as a running example, where $[\mathbb{N}]$ boils down to the QIT below.

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{N}^{\mathcal{S}}\ : \mathsf{Type} := \\
\mid \mathsf{O}_{\mathcal{S}} : \mathbb{N}^{\mathcal{S}} \\
\mid \mathsf{S}_{\mathcal{S}} : \mathbb{N}^{\mathcal{S}} \to \mathbb{N}^{\mathcal{S}} \\
\mid \mathsf{f}_{\mathbb{N}} : \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to \mathbb{N}^{\mathcal{S}}) \to \mathbb{N}^{\mathcal{S}} \\
\mid e_{\mathbb{N}} : \Pi(i : \mathsf{I})\, (x : \mathbb{N}^{\mathcal{S}}).\, \mathsf{f}_{\mathbb{N}}\ i\ (\lambda(o : \mathsf{O}\ i).\, x) =_{\mathbb{N}^{\mathcal{S}}} x.
\end{array}
$$

While defining the type and its constructors is not very exciting, it turns out that the eliminator packs a little bit of magic. To implement the dependent eliminator, we need a term in the target

$$\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}} : \Pi(P : \mathbb{N}^{\mathcal{S}} \to \mathsf{Type}^{\mathcal{S}})\, (p_{\mathsf{O}} : P\ \mathsf{O}_{\mathcal{S}})\, (p_{\mathsf{S}} : \Pi(n : \mathbb{N}^{\mathcal{S}}).\, P\ n \to P\ (\mathsf{S}_{\mathcal{S}}\ n))\, (n : \mathbb{N}^{\mathcal{S}}).\, P\ n$$

satisfying the usual conversion rules. The latter constraints leaves virtually no leeway for the constructors $\mathsf{O}^{\mathcal{S}}$ and $\mathsf{S}^{\mathcal{S}}$. But precisely, at this point we are facing a puzzling situation: we are only given branches for the $\mathsf{O}_{\mathcal{S}}$ and $\mathsf{S}_{\mathcal{S}}$ constructors, as the source eliminator only knows about those ones. How can we fill in the two remaining branches for $\mathsf{f}_{\mathbb{N}}$ and $e_{\mathbb{N}}$? In fact, we can, by defining $\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}$ as follows.

$$
\begin{array}{l}
\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}} : \Pi(P : \mathbb{N}^{\mathcal{S}} \to \mathsf{Type}^{\mathcal{S}})\, (p_{\mathsf{O}} : P\ \mathsf{O}_{\mathcal{S}})\, (p_{\mathsf{S}} : \Pi n.\, P\ n \to P\ (\mathsf{S}_{\mathcal{S}}\ n))\, (n : \mathbb{N}^{\mathcal{S}}).\, P\ n \\
\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ \mathsf{O}_{\mathcal{S}} \quad := \quad p_{\mathsf{O}} \\
\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}_{\mathcal{S}}\ n) \quad := \quad p_{\mathsf{S}}\ n\ (\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n) \\
\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{f}_{\mathbb{N}}\ i\ k) \quad := \quad (P\ (\mathsf{f}_{\mathbb{N}}\ i\ k)).\mathsf{f}\ i\ (\lambda(o : \mathsf{O}\ i).\, (e_{\mathbb{N}}\ i\ (k\ o))^{-1}\ \# \\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (k\ o))) \\
\mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (e_{\mathbb{N}}\ i\ x) \quad := \quad \ldots
\end{array}
$$

We explain here how to derive the missing branches from the material that is available. We believe that this is an enlightening point that needs to be given a strong emphasis, as it differs radically with other settings where adding side-effects break dependent elimination [53]. As in the effectful case, we use the algebra structure on P to propagate the effect of $\mathsf{f}_{\mathbb{N}}$ to the surrounding context, i.e. $\mathsf{f}_{\mathbb{N}}$ behaves as a kind of call-by-name exception. Yet, we crucially have to use $e_{\mathbb{N}}$ to rectify the type of the returned term. In the above definition,

$$i : \mathsf{I}, k : \mathsf{O}\ i \to \mathbb{N}^{\mathcal{S}}, o : \mathsf{O}\ i \vdash \mathbb{N}^{\mathcal{S}}_{\mathtt{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (k\ o) : P\ (k\ o)$$

but we actually want to return a term of type $P\ (\mathsf{f}_{\mathbb{N}}\ i\ k)$. With most monads, these two types are not isomorphic, and we have to restrict the recursor to predicates P enjoying additional linearity properties, namely they must be definitional algebra morphisms, i.e. commuting with f up to conversion [5,4]. Thankfully, with sheaves the additional quotient $e_{\mathbb{N}}$ is enough to derive a proof

$$i : \mathsf{I}, k : \mathsf{O}\ i \to \mathbb{N}^{\mathcal{S}}, o : \mathsf{O}\ i \vdash e_{\mathbb{N}}\ i\ (k\ o) : \mathsf{f}_{\mathbb{N}}\ i\ (\lambda(q : \mathsf{O}\ i).\, k\ o) = k\ o$$

but as we consider Prop to contain strict propositions, we also have

$$i : \mathsf{I}, k : \mathsf{O}\ i \to \mathbb{N}^{\mathcal{S}}, o : \mathsf{O}\ i \vdash (\lambda(q : \mathsf{O}\ i).\, k\ o) \equiv k : \mathsf{O}\ i \to \mathbb{N}^{\mathcal{S}}$$

hence in the end we do get a proof of $\mathsf{f}_{\mathbb{N}}\ i\ k = k\ o$ out of $e_{\mathbb{N}}$ by conversion.

The branch for the quotient constructor $e_{\mathbb{N}}$ requires a proof of its own, although this is a proof-irrelevant equality so the exact term does not matter as long as it exists, which is an easy consequence of the quotient preservation of P.

At the risk of sounding repetitive, the validity of dependent elimination is quite surprising given that we have additional constructors for our translated inductive types, a landmark of side-effects. What saves us really is that the quotient prevents one to exploit these effects in a computational way. At a more abstract level, one usually needs to restrict dependent elimination to some form of linear predicates when throwing in effects in a dependent type theory, as observed in [53]. Yet, $\mathcal{S}$ is an idempotent monad, which magically makes all morphisms linear, hence full dependent elimination becomes valid again. To be really fair, this line of reasoning only holds when reasoning extensionally enough, something which is easy in a categorical setting but much less so in MLTT. The fact that synthetic sheaves can be presented in a very computational way through QITs is critical for this trick to go through in type theory.

Universes The last big ingredient that is still missing from our model to properly model MLTT is the existence of universes. Here the story becomes way more blurry, and the literature is somewhat confusing. Sheaf models are the archetypical instance of a topos, so they do provide us with a small impredicative universe of propositions Prop. We can easily reflect it in our translation. We do not give the full details, but basically by using `propext` it is easy to show that

$$[\![\mathsf{Prop}]\!] \qu:=\quad \Sigma(P : \mathsf{Prop}).\, \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to P) \to P$$

is a sheaf by taking

$$\mathsf{f}_{\mathsf{Prop}}\ (i : \mathsf{I})\ (k : \mathsf{O}\ i \to [\![\mathsf{Prop}]\!]) : [\![\mathsf{Prop}]\!] \qu:=\quad (\Pi(o : \mathsf{O}\ i).\, (k\ o).1, \quad \dots \quad).$$

If we try to replicate the same technique with Type, the intuitive equivalent is to show that $\mathsf{Type}^{\mathcal{S}}$ is indeed a sheaf. Except that in general, it is not. The reason is that the use of `propext` for Prop does not scale to Type. One would have to find a term $\mathsf{f}_{\mathsf{Type}}\ (i : \mathsf{I})\ (k : \mathsf{O}\ i \to \mathsf{Type}^{\mathcal{S}}) : \mathsf{Type}^{\mathcal{S}}$. We are quite constrained in what we can put here because of the quotient condition. Just like for Prop, the

natural candidate for the $\texttt{el}$ component of this operation is $\Pi(i : \mathsf{I}).(k\ o).\texttt{el}$. If we do this, we have a problem though: we cannot prove that the quotient is preserved! Indeed, we have to show

$$A : \mathsf{Type}^{\mathcal{S}}, i : \mathsf{I} \vdash \_ : (\Pi(o : \mathsf{O}\ i).\, A.\texttt{el}) = A.\texttt{el}$$

but the best we can hope for is an *isomorphism* $(\Pi(o : \mathsf{O}\ i).\, A.\texttt{el}) \cong A.\texttt{el}$ by the sheafness of A. This is sufficient for Prop as isomophism implies logical equivalence and thus equality, but this does not carry to proof-relevant types.

This is a very well-known problem that led to the introduction of *stacks* [10], which are essentially sheaves where the quotient based on propositional equality is replaced by a tower of proof-relevant relations, resulting in some kind of ω-groupoid. This kind of endeavour is unfortunately way beyond our syntactic approach, as it would morally entail reimplementing a cubical model [16]. Without stacks, the universe of sheaves is only a weak universe [64], which is not enough for our purposes. As argued in [63], there is some misunderstanding in the literature about the availability of strict universes in run-of-the-mill sheaves. For the sake of completeness, we recall here two ways to get them.

The first solution is to blindly replace PshTT with HoTT in everything we did previously. There is nothing specific to be done, apart maybe for replacing Prop with some predicative variant of strict propositions in case one does not want to buy into resizing axioms. All other constructions are carried just the same, except that we silently interpret equality as a univalent one, and thus in particular the QITs we wrote before are now implicitly HITs — this is but a mere point of view, the syntax remains unchanged. The surprising part is that by doing so, we do not have to suffer with stacks: the universe of sheaves effortlessly becomes a sheaf by what seems to be sheer magic. Indeed, in the strict setting, there is no way to turn the isomorphism $(\Pi(o : \mathsf{O}\ i).\, A.\texttt{el}) \cong A.\texttt{el}$ into an equality. But in a univalent theory, this isomorphism is actually an *equivalence*, and hence by univalence gives rise to the equality sought after. Once again, this is the path taken by [59], although in our opinion they do not insist enough on this miraculous phenomenon.

The other way is to replace an open universe of sheaves by a closed one, that is, using some inductive-recursive encoding and replacing sheaves by their code. We believe that this is the approach taken by [39], although they build directly this object from first principles in a very non-constructive way. For this to be applied to our syntactic setting, we would need a first-class notion of *quotient inductive recursive types* (QIRT) in PshTT. To be clear, we have no idea whether general QIRTs can be shown to exist in some models or if we can encode it away using some variant of small induction recursion [41,37] or realignment types. Nonetheless we sketch what they would look like to give a short, intuitive albeit arguably too approximate explanation of the definition from [39]. The QIRT of sheaf codes is described by an inductive definition $\mathcal{U}^{\mathcal{S}}$ of codes together with a recursive function $\texttt{El}^{\mathcal{S}} : \mathcal{U}^{\mathcal{S}} \to \mathsf{Type}^{\mathcal{S}}$ additionally satisfying the sheaf quotient condition by fiat, thanks to a quotient constructor similar to $e_{\mathcal{S}}$ in $\mathcal{U}^{\mathcal{S}}$. It is not clear exactly how this would compute, let alone be properly defined. We refer to the exploratory work of Kaposi [42] and leave this to future study.

2.5 Computational Content of Sheaves

We want to highlight now a fact that is less well-known about sheaves that has important consequences when thinking about computation. This fact is absent from the historical definition of sheaves, and a bit hidden in the HoTT presentation [59], but our inductive description of sheafification makes it extremely clear. In a nutshell, sheaves are actually about approximating idealized infinite objects through finite approximations. This viewpoint is a staple of some constructivist schools under the name of *dynamical methods* [20]. In particular, Coquand, Lombardi and co-authors have conducted a systematic research program exploring how these methods unveil the effective content of classical proofs [20,26,46]. Yet, it does not seem to have percolated that much in the world of the proof-as-program correspondence. We seize the opportunity to expose clearly here this technique. Let (I, O) be a LOS in the remainder of this section.

It is a truism that ultimately, when performing computations, one only cares about the value of some concrete datatypes. In first-order logic, this is often conflated with the requirement that Σ^1_0 formulae enjoy a witness property, while in type theory, we typically expect closed terms $\vdash M : \mathbb{N}$ to evaluate to some concrete natural number. Such a property is traditionally called *canonicity*, and can be defined more generally for all closed terms of an inductive type.

We now easily see that our sheafified theory enjoys a weaker form of canonicity inherited from the ambient theory by inspecting the translation from Section 2.4. For each inductive $\mathcal{I}$, we have two additional constructors $\mathsf{f}_{\mathcal{I}}$ and $e_{\mathcal{I}}$, where the latter is only used to build equalities. Our weaker canonicity thus says that a value in the model is a *finite* chain of $\mathsf{f}_{\mathcal{I}}$ ending with an actual constructor.

We argue that this is a generalization of the fact that in MLTT, if Γ is a consistent, purely negative context, terms $\Gamma \vdash M : \mathbb{N}$ still enjoy canonicity [22]. With sheaves, O needs not be purely negative, but the price to pay is that we have to keep an explicit list of calls to the f constructors. It is still better than just adding opaque axioms to the theory, since these terms bubble up to toplevel.

Note that weak canonicity also applies in particular to the empty type, which has no actual constructor. Hence the sheaf theory is consistent exactly when O is finitely consistent, i.e. the inductive type $F := \mathsf{f}_F : \Pi(i : \mathsf{I}).\, (\mathsf{O}\ i \to F) \to F$ is empty. This is highly remininiscent of both the compactness lemma of first-order logic and Herbrand's theorem. Thus, sheafification is manifestly about finiteness. But what is the infinite thing it approximates? With our presentation, it becomes straightforward.

Definition 10. *An omniscient oracle is a function* $\alpha : \Pi(i : \mathsf{I}).\, \mathsf{O}\ i$.

Assuming an omniscient oracle α, one can evaluate a term $M : \mathcal{S}\, A$ into an actual value of A through the `eval` function defined recursively as

$$\begin{aligned}
&\texttt{eval} : (\Pi(i : \mathsf{I}).\, \mathsf{O}\ i) \to \mathcal{S}\, A \to A \\
&\texttt{eval}\ \alpha\ (\eta_{\mathcal{S}}\ x) \ := x \\
&\texttt{eval}\ \alpha\ (\mathsf{f}_{\mathcal{S}}\ i\ k) := \texttt{eval}\ \alpha\ (k\ (\alpha\ i))
\end{aligned}$$

where the quotient preservation is left implicit but can be easily proved.

We see the term α as the embodiment of the infinite, able to answer all questions. It is an idealized object that may not necessarily exist in a constructive setting. By contrast, due to its inductive nature, a term $M : \mathcal{S}\,A$ can only ask a finite number of questions via $\mathtt{f}_{\mathcal{S}}$. The `eval` function mediates between the finite and the infinite, giving a relativized meaning to an object that does only depend on a finite approximation of an idealized abstraction. Working in a sheaf type theory progagates this identification at all types.

This pattern is a commonplace in constructive mathematics, yet we are not aware of any reference where it is explicitely explained through the inductive quality of sheafification. To name a few classic instances of this viewpoint, let us cite the algebraic closure, the ultrafilter theorem, etc. More generally, in first-order logic all these objects can be axiomatized through the notion of geometric theories, i.e. sets of geometric formulae of the shape

$$\boldsymbol{x} \mid \alpha_1(\boldsymbol{x}), \dots \alpha_n(\boldsymbol{x}) \vdash \bigvee_{i \in I} \exists \boldsymbol{y}_i.\, \beta_1(\boldsymbol{x}, \boldsymbol{y}_i) \wedge \dots \wedge \beta_{m_i}(\boldsymbol{x}, \boldsymbol{y}_i)$$

where the α, β predicates are atomic. Assuming a single-axiom theory, the I type corresponds to the input $\boldsymbol{x}$ s.t. $\alpha_i(\boldsymbol{x})$ for all $1 \leq i \leq n$ and O is the big disjunction in the conclusion, which depends on the input $\boldsymbol{x}$. This generalizes to arbitrary sets of axioms by taking I to be the product of the corresponding input types.

The relationship between geometric formulae and finite approximations was already observed by Coquand [21], who gives a model of first-order geometric theories where proofs of Σ^1_0 formulae are interpreted as inductive decision trees.

3 A Case Study of Type-Theoretical Forcing: MLTT$^{\mathsf{f}}$

3.1 General Motivation

The second part of this article discusses how sheaves shall earn a first-class status in type theory. The critical point here is to be able to *compute* directly in the sheaf theory. Note that we mean it in a strong sense, i.e. the objective here is actually to obtain an extension of MLTT that can be equipped with an algorithmic reduction generating normal forms.

To the reader, it may seem like the synthetic approach we have been advocating in the previous section is already a satisfying answer. Assuming univalence in the target and following [59], we sketched that one can recover such a syntactic model. Unfortunately, this is only part of the story.

A first drawback is the reliance on univalence. It brings in some technicalities, as it is usually available through some cubical theory. Moreover, it hardwires a lot of additional logical baggage that one may not desire to expose. For better or for worse, traditional sheaves are developped in an anti-univalent setting. The more neutral the foundations we define sheaves in, the more users they will reach.

One could argue that minimalism is no virtue in itself, and that type theorists should bite the univalent bullet already. This is fair enough, but there is another

roadblock. Traditional sheaf models are the composition of a synthetic sheaf model with a presheaf model. The latter corresponds to a monotonic reader effect, i.e. morally a new kind of context, where the base category gives access to modalities restricting the current state. See e.g. [38] for a type-theoretic account.

In this setting, it is virtually always the case that the LOS (I, O) is made of *exotic* types, i.e. types defined through non-standard modalities of the presheaf model that cannot be written directly in MLTT. As mentioned, it is typical for O to be some kind of disjunction capturing a geometric formula. Yet, these disjunctions are not arbitrary propositions: there is a one-to-one mapping between atomic exotic formulae and principal sieves from the underlying base category. What we mean is that, in traditional sheaf models, there is a universe of propositions $\mathsf{fProp} \subsetneq \mathsf{Prop}$ that contains all principal sieves and is closed under e.g. finite meets and joins, but is still at the same time *definitionally* proof-irrelevant and enjoys unique choice. We believe that this is an amazing feature of these models. In general, if one wants to keep decidability of type-checking, one has to choose between definitional irrelevance or unique choice. Sheaf models resolve this tension by giving both for propositions that arise a finite way from the base category. More generally, even without taking these type-theoretic considerations into account, all serious uses of sheaves are constructed above a presheaf model. Thus there seems to be both foundational and empirical evidence that we want to revisit the traditional setting by composing synthetic sheaves with presheaves in a type-theoretic world.

And this is where we hit a wall. On the one hand, we have a sheaf model requiring univalence. On the other hand, we do have a syntactic model which interprets a theory equivalent to presheaves [52]... but this model requires and propagates definitional UIP! As a result, we simply cannot plug one into the other. A syntactic model of univalent presheaves seems completely out of reach, so there is no clear way out of this conundrum via the syntactic route.

Note that the failed approach above tries to define a syntactic presheaf model over a base category $\mathbf{C}$ *internal* to a homotopic type theory, whose precise model does not matter but should be computational. This is subtly but fundamentally different from the sheaf models from Coquand et al. [25]. While they also rely on univalence and use the same synthetic encoding of sheaves, their presheaves are defined externally in an unspecified metatheory. Said otherwise, their models are built out of presheaves $\square \times \mathfrak{C} \to \mathbf{Set}$ where $\square$ is some cube category and $\mathfrak{C}$ an *external* base category. Unfortunately, there is no hope to recover a decent computational content out of usual presheaves [52], so this is a no-go.

We need a more semantic approach. We want to define a type theory capturing the internal language of Grothendieck toposes that keeps all the great properties of MLTT. The task is daunting, given the variety of sheaf models. As an initial step, we will focus in this paper on the simplest non-trivial case of sheafification we could think of, namely the addition of a Cohen real to MLTT, that is to say, an uninterpreted variable $\alpha : \mathbb{N} \to \mathbb{B}$ that is approximated by partial functions of finite support. A close variant of this theory was first sketched

by Coquand and Jaber in [23,24]. Up to unique choice, Cohen reals correspond to the geometric theory with one atom $\alpha : \mathbb{N} \to \mathbb{B} \to \mathsf{Prop}$ and the two axioms

$$n : \mathbb{N} \mid \cdot \vdash \alpha\ n\ \mathsf{tt} \vee \alpha\ n\ \mathsf{ff} \qquad \text{and} \qquad n : \mathbb{N} \mid \alpha\ n\ \mathsf{tt}, \alpha\ n\ \mathsf{ff} \vdash \bot.$$

This theory enjoys some remarkable properties: it is infinite and the branching is at the same time non-trivial, finite and disjoint. In terms of LOS, the first property means our type I is infinite, and thus our idealized object as well. This is the whole point of forcing. The other properties constrain the O predicate. Being non-trivial means that we get trees instead of lists in the semantics, and finiteness will maintain decidability properties. Finally, disjointness makes the whole setting much more tractable at the cost of some degeneracy. Semantically, this means that the decision diagram is not a directed acyclic graph but a proper tree. Because of this, we can completely ignore issues with unique choice and replace the functional predicate α with an actual function. Disjoint branching is very specific to Cohen reals, and makes our model a lot simpler, syntactically and semantically.

3.2 Continuity

As specific as it may seem, adding a single Cohen real to a type theory can already bring in interesting metatheoretical results about MLTT. In some way that we already laid down in Section 2.5, the very purpose of sheafification is to provide generalized continuity results. It is thus tempting to hope that the archetypical continuity property shall follow from a simple form of sheaves. As there are several non-equivalent notions of continuity [11], we need some paraphernalia to formally explain what we mean by this word. Escardó introduced a close relative to interaction trees called *dialogue trees* [31], which is just the type

$$\begin{array}{l}\texttt{Inductive}\ \mathfrak{D}\ (A : \mathsf{Type}) : \mathsf{Type} := \\ \mid \eta_{\mathfrak{D}} : A \to \mathfrak{D}\ A \\ \mid \mathtt{f}_{\mathfrak{D}} : \Pi(i : \mathsf{I}).\ (\mathsf{O}\ i \to \mathfrak{D}\ A) \to \mathfrak{D}\ A\end{array} \qquad \text{with} \qquad \mathsf{I} : \mathsf{Type} \text{ and } \mathsf{O} : \mathsf{I} \to \mathsf{Type}.$$

As explained in Section 2.3, similarity to sheaves is not coincidental. The main difference between $\mathfrak{D}$ and $\mathcal{S}$ is that $\mathfrak{D}$ lacks any kind of quotient, and furthermore that its O type is proof-relevant. Just like for $\mathcal{S}$, we can define an evaluation function $\mathtt{eval}_{\mathfrak{D}} : (\Pi(i : \mathsf{I}).\ \mathsf{O}\ i) \to \mathfrak{D}\ A \to A$ which can be used to define a rather strong notion of *continuity* for functionals.

Definition 11. *A function $F : (\Pi(i : \mathsf{I}).\ \mathsf{O}\ i) \to A$ is dialogue-continuous, written $\mathcal{C}_{\mathfrak{D}}\ F$, if there is $d : \mathfrak{D}\ A$ s.t. $\Pi(\alpha : \Pi(i : \mathsf{I}).\ \mathsf{O}\ i).\ F\ \alpha = \mathtt{eval}\ \alpha\ d$.*

This notion is strong because it implies a specific sequentialization of calls to the higher-order argument, an intensional property not present in the traditional view of continuity as a dependency on a finite prefix of the input. Taking specific instances of (I, O), one can recover well-known notions of continuity. Notably, for $\mathsf{I} := \mathbb{N}$ and $\mathsf{O}\ i := \mathbb{B}$, this definition is equivalent to uniform continuity over the

Cantor space $\mathbb{N} \to \mathbb{B}$. For $\mathsf{I} := \mathbb{N}$ and $\mathsf{O}\ i := \mathbb{N}$, it implies pointwise continuity over the Baire space $\mathbb{N} \to \mathbb{N}$, but is weaker than uniform continuity. As uniform continuity over the Baire space is very strong, Fujiwara and Kawai [34] argue that dialogue continuity is the proper way to extend uniform continuity from the Cantor to the Baire space. Moreover, note that this generalization to arbitrary (I, O) types is reminiscent of Brede and Herbelin's *generalized bar induction* [15].

Using the fact $\mathfrak{D}$ is a monad, Escardó gave a proof that System T enjoys dialogue continuity for the Baire space in an external way via what amounts to a syntactic effectful model [22]. Here, the argument $\mathbb{N} \to \mathbb{N}$ is handled as an oracle in the sheaf way by interpreting System T into the call-by-value embedding of $\mathfrak{D}$. This result was extended by Sterling to Brouwer sequences [62]. Finally, Baillon et al. generalized Escardó's result to dependent type theory [12]. Their model interprets universes but lacks full dependent elimination. Indeed, oracle calls are added freely via a f constructor, introducing an observable effect which forces them to restrict the interpreted theory to Baclofen Type Theory [53].

Escardó and Xu also studied more semantic approaches to the same kind of questions [69,29] veering towards sheaves. Due to the semantic nature of these works, it is hard to recover actual computation from their results.

There was an important series of papers around this topic from the PER community, Rahli being in the intersection of all of their authors [58,56,57,19,17,18]. These papers consider various forms of continuity in realizability models of MLTT à la NuPRL, with some built-in form of computation, and where the metatheory is actually Rocq. By loyalty to the Brouwerian tradition, they are advertized as Beth models, but they really are sheaf models, since they are proof-relevant. Note that in their approach, the models are by design open, as there is no inductive definition of well-typedness in sight: everything is defined in a semantics that only cares about closed terms. Completeness is in particular a non-object, and there is no hope to get a decidable type-checking algorithm for their non-theory. While we boast no obsession for Brouwerian choice sequences, we believe the intuitions developed in this community to be valuable, and that it is enlightening to revisit these historical artifacts with a modern type-theoretic point of view.

In the remainder of this paper, we leverage recent developments in the mechanization of models of normalization by evaluation [2,3]. Our goal is twofold. First, we define and study $\mathsf{MLTT}^{\mathsf{f}}$, the most elementary sheaf extension of MLTT with a single Cohen real. Given the complexity of the objects at play, we formalize our results in the Rocq proof assistant. Then, as a byproduct, we derive a constructive proof of continuity for MLTT functionals over the Cantor space.

3.3 $\mathsf{MLTT}^{\mathsf{f}}$

It is now time to enter the real matter. We define in this section $\mathsf{MLTT}^{\mathsf{f}}$, a variant of MLTT extended with a formal oracle $\alpha : \mathbb{N} \to \mathbb{B}$, together with a local state of forcing conditions $\ell \in \mathfrak{L}$ that represent finite knowledge about α. Note that $\mathfrak{L}$ is a type in the metatheory, not in $\mathsf{MLTT}^{\mathsf{f}}$. Since we will often switch from one to the other, we will try to write $\mathsf{MLTT}^{\mathsf{f}}$ objects in a normal font and metatheoretical

ones in a fraktur font. To further insist, we will use set-theoretical notations for the metatheory, though it is technically the type theory of Rocq.

In the mechanization we set $\mathfrak{L} := \mathfrak{list}\ (\mathfrak{N} \times \mathfrak{B})$, where $\mathfrak{list}$, $\mathfrak{N}$ and $\mathfrak{B}$ respectively are metatheoretical lists, natural numbers and booleans. We will write $[]$ for the empty list and $::$ for the cons operator. Despite use of lists, conditions should rather be thought of as finite sets, and all operations will preserve the implict reordering quotient. In particular, $\mathfrak{L}$ enjoys the reverse inclusion order

$$\ell' \preceq \ell \quad := \quad \Pi(\mathfrak{n} \in \mathfrak{N})\,(\mathfrak{b} \in \mathfrak{B}).\ (\mathfrak{n}, \mathfrak{b}) \ltimes \ell \to (\mathfrak{n}, \mathfrak{b}) \ltimes \ell'$$

where $\ltimes$ is any reasonable definition of list membership. When $\ell' \preceq \ell$, we view ℓ' as more *precise* than ℓ, insofar as it contains more information. Finally, given $\ell \in \mathfrak{L}$, we will write $\mathsf{dom}(\ell) \in \mathfrak{list}\ \mathfrak{N}$ for the list of the first projections of ℓ.

$\mathsf{MLTT}^{\mathsf{f}}$ follows the usual presentation of dependent type theory with five kind of judgments, i.e. context, type and term well-formedness together with type and term conversion. The only difference with MLTT here is that we annotate all judgments with forcing conditions, leading to the judgment shapes below.

well-formed context	$\vdash \ell \mid \Gamma$		
well-formed type	$\ell \mid \Gamma \vdash A$	well-formed term	$\ell \mid \Gamma \vdash M : A$
convertible types	$\ell \mid \Gamma \vdash A \equiv B$	convertible terms	$\ell \mid \Gamma \vdash M \equiv N : A$

Let us mention we only ever consider forcing conditions satisfying a well-formedness predicate, checking that all bindings in ℓ appear at most once. This ensures that conditions indeed code for finite approximations of functions. Formal definition of this predicate can be found here.

Typing and conversion rules for $\mathsf{MLTT}^{\mathsf{f}}$ are an extension of the rules for MLTT, with negative Π and Σ-types with definitional η-rules, natural numbers, booleans, empty and identity types, and one universe. We refer the interested reader to the mechanization for a complete description of the theory.

$$(\textsc{Oracle})\ \frac{\ell \mid \Gamma \vdash M : \mathbb{N}}{\ell \mid \Gamma \vdash \alpha\ M : \mathbb{B}} \qquad \frac{\vdash \ell \mid \Gamma \qquad (\mathfrak{n}, \mathfrak{b}) \ltimes \ell}{\ell \mid \Gamma \vdash \alpha\ \overline{\mathfrak{n}} \equiv \overline{\mathfrak{b}} : \mathbb{B}}\ (\textsc{Eval})$$

$$\frac{(\mathfrak{n}, \mathsf{tt}) :: \ell \mid \Gamma \vdash \mathcal{J} \qquad (\mathfrak{n}, \mathsf{ff}) :: \ell \mid \Gamma \vdash \mathcal{J} \qquad \mathfrak{n} \not\in \mathsf{dom}(\ell)}{\ell \mid \Gamma \vdash \mathcal{J}}\ (\textsc{Split})$$

Fig. 2. New rules for $\mathsf{MLTT}^{\mathsf{f}}$

We only describe the additional rules handling the oracle and forcing conditions in Figure 2. The Oracle rule states that α is a function in the Cantor space. We define α as a unary term former but abuse the application notation. This is for technical reasons and we can retrieve a proper function by η-expansion. The Eval rule states that the state of knowledge ℓ can be reflected into conversion for α itself. As forcing conditions $(\mathfrak{n}, \mathfrak{b}) \ltimes \ell$ live in the meta-theory,

we rely on injections $\overline{\mathfrak{n}} : \mathbb{N}$ and $\overline{\mathfrak{b}} : \mathbb{B}$ as terms of $\mathsf{MLTT}^{\mathsf{f}}$. Finally, sheafness of $\mathsf{MLTT}^{\mathsf{f}}$ is embodied in the Split rule scheme, available for any judgment $\mathcal{J}$ of our theory. It allows extending the local knowledge about α by making a case analysis on its value at some concrete input $\mathfrak{n}$. We have to be ready to handle either value, but note that the judgment is the same in both premises. This does not prevent performing pattern-matching on $\alpha\ \overline{\mathfrak{n}}$ further down the term, but forces the Split rule to separate local extension of knowledge from its analysis. A typical use of this rule is to derive a restricted η-rule for α, as it allows e.g. proving

$$\ell \mid \Gamma \vdash \mathtt{if}\ \alpha\ \overline{\mathfrak{n}}\ \mathtt{then}\ \mathsf{tt}\ \mathtt{else}\ \mathsf{ff} \equiv \alpha\ \overline{\mathfrak{n}} : \mathbb{B}$$

for some concrete $\mathfrak{n} \in \mathfrak{N}$ where the $\mathtt{if} - \mathtt{then} - \mathtt{else}$ is syntactic sugar for the boolean recursor.

3.4 Canonicity

As explained in Section 2.5, in sheaf models usual canonicity results do not stand. This carries over to $\mathsf{MLTT}^{\mathsf{f}}$ as it is not the case that a closed term $[] \mid \cdot \vdash M : \mathbb{N}$ reduces to a value, since M may depend on α. We instead get a *weak canonicity* result, i.e. canonicity up to an inductive tree of splits. To make this formal, we first define the metatheoretic type $\mathfrak{D}$ of dialogue trees parameterized by ℓ.

$$\begin{array}{l}\mathtt{Inductive}\ \mathfrak{D}\ (\ell : \mathfrak{list}\ (\mathfrak{N} \times \mathfrak{B})) : \mathsf{Type} := \\ \mid \eta_{\mathfrak{D}} : \mathfrak{D}\ \ell \\ \mid \mathfrak{f}_{\mathfrak{D}} : \Pi(\mathfrak{n} : \mathfrak{N}).\, \mathfrak{n} \not\in \mathsf{dom}(\ell) \to (\Pi(\mathfrak{b} : \mathfrak{B}).\, \mathfrak{D}\ ((\mathfrak{n}, \mathfrak{b}) :: \ell)) \to \mathfrak{D}\ \ell\end{array}$$

Intuitively, $d : \mathfrak{D}\ \ell$ describes a tree of questions that are left unanswered by ℓ. We then define $d \lhd \ell'$, a predicate capturing that there is a path from ℓ to ℓ' in d. Both $\mathfrak{D}$ and $d \lhd \ell'$ are formally defined here.

$$\frac{}{\eta_{\mathfrak{D}}\ \ell \lhd \ell} \qquad \frac{k\ \mathsf{tt} \lhd \ell}{\mathfrak{f}_{\mathfrak{D}}\ \mathfrak{n}\ \mathfrak{n}_\varepsilon\ k \lhd (\mathfrak{n}, \mathsf{tt}) :: \ell} \qquad \frac{k\ \mathsf{ff} \lhd \ell}{\mathfrak{f}_{\mathfrak{D}}\ \mathfrak{n}\ \mathfrak{n}_\varepsilon\ k \lhd (\mathfrak{n}, \mathsf{ff}) :: \ell}$$

The last thing we need before stating the weak canonicity theorem is a notion of weak-head reduction for $\mathsf{MLTT}^{\mathsf{f}}$. We define it as an extension of the usual rules for MLTT, adding specific rules to handle α. Contrarily to MLTT, reduction rules are annotated with a state of knowledge ℓ which is used in the new rules for α. We only state these ones and refer to the mechanization for further details.

$$\frac{(\mathfrak{n}, \mathfrak{b}) \ltimes \ell}{\alpha\ \overline{\mathfrak{n}} \rightsquigarrow_\ell \overline{\mathfrak{b}}} \qquad \frac{M \rightsquigarrow_\ell M' \qquad k \in \mathfrak{N}}{\alpha\ (\mathsf{S}^k\ M) \rightsquigarrow_\ell \alpha\ (\mathsf{S}^k\ M')}$$

The first rule is the counterpart to the Eval conversion rule. It reflects in the reduction the current state of knowledge about α. The second rule is a congruence rule for α, which allows reducing its argument as long as it may still evaluate to a closed integer. This rule introduces a tiny amount of deep reduction, since it can fire under an arbitrary amount of successor nodes.

In a pure setting like MLTT, there are exactly two kinds of normal forms: values and neutrals. Values are terms which start with a introduction rule, e.g. $\mathsf{S}\ M$ or $\lambda x : A.\, M$. Meanwhile, neutrals are terms whose head is an elimination blocked on a variable. In $\mathsf{MLTT}^{\mathsf{f}}$, due to the oracle, we get a third kind of normal form, which we call α-*neutrals*. An α-neutral at world ℓ is a term stuck on $\alpha\ \overline{\mathfrak{n}}$ in head position with $\mathfrak{n} \not< (\mathsf{dom}\ \ell)$. Moving to a more informative $\ell' \preceq \ell$ may unlock further computation. It will correspond to a node in the split tree of a normal form, and the model will ensure that α-neutrals can be eventually unblocked.

We will also consider hereditary reduction to a deep normal form $M \Downarrow_\ell N$, which is defined in a standard way. We are now ready to state weak canonicity for $\mathsf{MLTT}^{\mathsf{f}}$. Proving it is one of the goals of Section 4. It will be the cornerstone to the proof that all MLTT-definable functionals are continuous.

Theorem 2 (Weak canonicity). *For any closed term* $[] \mid \cdot \vdash M : \mathbb{N}$ *of* $\mathsf{MLTT}^{\mathsf{f}}$*,*

$$\Sigma(d \in \mathfrak{D}\ []).\, \Pi(\ell \in \mathfrak{L}).\, d \triangleleft \ell \longrightarrow \Sigma(\mathfrak{n} \in \mathfrak{N}).\, M \Downarrow_\ell^* \overline{\mathfrak{n}}.$$

3.5 Continuity

We will assume in this section that Theorem 2 holds, and will use it to prove that MLTT functionals of type $(\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ are continuous. First, we need to agree about what we actually mean by that, i.e. what the statement really says and how internal it is. We focus first on the latter.

Assuming some formal definition of continuity, we have various levels in which it can be reflected in MLTT. We single out three specific points on this continuum, that we will call *external* continuity, the continuity *rule* and the continuity *principle*. External continuity is an exclusively metatheoretical property, i.e. "for any closed term F, we have $\mathfrak{C}\ F$" where $\mathfrak{C}$ is a predicate in the metatheory. This is the only one one can state when the object language is not expressive enough. A recent paper by Escardó et al [30] internalizes an encoding of a dialogue tree in System T but cannot *internally say* that it is a witness of dialogue continuity. Here, MLTT, can express any reasonable notion of continuity as an internal property $\vdash \mathcal{C} : ((\mathbb{N} \to \mathbb{B}) \to \mathbb{N}) \to \mathsf{Type}$. So we can ask for the continuity principle, a term $\Phi_{\mathcal{C}}$ that proves continuity uniformly, i.e. $\vdash \Phi_{\mathcal{C}} : \Pi(f : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}).\, \mathcal{C}\ f$. Being the strongest of the three, it is also the trickiest. Baillon et al. [11] provide a large range of continuity definitions, but even with one of the weakest, namely *modulus continuity*, Escardó and Xu [32] show that the continuity principle on the Baire space is inconsistent in MLTT. For the Cantor type, their counter-example does not hold however, so hope remains.

Finally, the continuity rule states that $\vdash F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ implies the existence of some term $\vdash \{F\}^{\mathcal{C}} : \mathcal{C}\ F$. It allows reflecting the continuity proof in MLTT itself, but in a non-uniform way. Here, the operation $\{F\}^{\mathcal{C}}$ is defined in the metatheory, typically via some external induction on the syntax of F which must be a *closed* term. Clearly, the continuity principle implies the continuity rule, but the inverse is not true in general. Moreover, with some mild assumptions on the object theory, the continuity rule often implies its external variant.

We will focus on the continuity rule, taking uniform continuity as definition. It is easier to state and equivalent to dialogue continuity on the Cantor space.

Definition 12. *$F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ is uniformly continuous, written $\mathcal{C}\ F$, if*

$$\Sigma(n : \mathbb{N}).\, \Pi(\alpha\, \beta : \mathbb{N} \to \mathbb{B}).\, \alpha \approx_n \beta \to F\ \alpha = F\ \beta$$

where $\approx_n$ is defined as pointwise equality on the n-th first integers.

Theorem 3 (Uniform continuity rule). *For any $\vdash_{\mathsf{MLTT}} F : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$, there exists a closed term $\vdash_{\mathsf{MLTT}} \{F\}^{\mathcal{C}} : \mathcal{C}\ F$.*

Proof. Let us assume such a F. By inclusion $[] \mid \cdot \vdash_{\mathsf{MLTT}^{\mathsf{f}}} F\ \tilde{\alpha} : \mathbb{N}$, where $\tilde{\alpha} := \lambda n.\, \alpha\ n$. By Theorem 2, there is a finite set $\mathfrak{F}$ of forcing conditions s.t. for each $\ell \in \mathfrak{F}$, $F\ \tilde{\alpha} \Downarrow_\ell \overline{\mathfrak{n}}_\ell$ for some $\mathfrak{n}_\ell \in \mathfrak{N}$. We can replay this reduction in MLTT by substituting α with any MLTT term α_0 that is compatible on ℓ reduction-wise. In particular for any such α_0:

$$\vdash_{\mathsf{MLTT}} \mathtt{refl}\ \mathbb{N}\ \overline{\mathfrak{n}}_\ell : F\ \alpha_0 = \overline{\mathfrak{n}}_\ell. \tag{1}$$

For any $\mathfrak{p} \in \mathfrak{N}$, there is $\vdash_{\mathsf{MLTT}} \mathtt{set}_{\mathfrak{p}} : (\mathbb{N} \to \mathbb{B}) \to \mathbb{B}^{\mathfrak{p}} \to \mathbb{N} \to \mathbb{B}$ defined by finite case analysis s.t. for $0 \leq \mathfrak{i} < \mathfrak{p}$, $\mathtt{set}_{\mathfrak{p}}\ M\ B_0\ \ldots\ B_{\mathfrak{p}-1}\ \overline{\mathfrak{i}} \rightsquigarrow^*_{\mathsf{MLTT}} B_{\mathfrak{i}}$ and for $\mathfrak{p} \leq \mathfrak{i}$, $\vdash_{\mathsf{MLTT}} \mathtt{set}_{\mathfrak{p}}\ M\ B_0\ \ldots\ B_{\mathfrak{p}-1}\ \overline{\mathfrak{i}} \equiv M\ \overline{\mathfrak{i}}$.

Since $\mathfrak{F}$ is a covering, there is $\mathfrak{m} \in \mathfrak{N}$ and $\vdash_{\mathsf{MLTT}} \Phi : \mathbb{B}^{\mathfrak{m}} \to \mathbb{N}$ together with a proof $\vdash_{\mathsf{MLTT}} \_ : \Pi\alpha\, (b_0\ \ldots\ b_{\mathfrak{m}-1} : \mathbb{B}).\, F\ (\mathtt{set}_{\mathfrak{m}}\ \alpha\ b_0\ \ldots\ b_{\mathfrak{m}-1}) = \Phi\ b_0\ \ldots\ b_{\mathfrak{m}-1}$ where Φ is defined out of $\{\mathfrak{n}_\ell\}_{\ell \in \mathfrak{F}}$, and the proof is carried by destructing all b_i variables and concluding by Eq. (1). By reasoning internally, we easily get a proof

$$\begin{aligned}\vdash_{\mathsf{MLTT}} \_ :\ & \Pi(\alpha\, \beta : \mathbb{N} \to \mathbb{B}).\, \alpha \approx_{\overline{\mathfrak{m}}} \beta \to \\ & F\ (\mathtt{set}_{\mathfrak{m}}\ \alpha\ (\alpha\ 0)\ \ldots\ (\alpha\ \overline{\mathfrak{m}-1})) = F\ (\mathtt{set}_{\mathfrak{m}}\ \beta\ (\beta\ 0)\ \ldots\ (\beta\ \overline{\mathfrak{m}-1}))\end{aligned}$$

To conclude we need to show that F is extensional enough to behave the same on α and its $\mathtt{set}_{\mathfrak{m}}$ expansion. But this is a consequence of *parametricity* [13], which shows that for any closed term $\vdash_{\mathsf{MLTT}} M : (\mathbb{N} \to \mathbb{B}) \to \mathbb{N}$ we have a proof $\vdash_{\mathsf{MLTT}} \_ : \Pi(f\, g : \mathbb{N} \to \mathbb{B}).\, (\Pi(n : \mathbb{N}).\, f\ n = g\ n) \to M\ f = M\ g$.

4 A syntactic model of MLTT in MLTT$^{\mathsf{f}}$

This section is dedicated to the description of both the theoretical and practical aspects of our $\mathsf{MLTT}^{\mathsf{f}}$ model. Such models are fairly technical, so we will try to stay high-level and will refer to the mechanization for the nitty-gritty details.

4.1 A Mechanized Logical Relation

Our model of $\mathsf{MLTT}^{\mathsf{f}}$ can be aptly described as a logical relation. Here, we use this syntagm in a restricted and idiosyncratic sense to describe a specific kind

of model whose earliest representative is probably Girard's strong normalization model of System F [36] and whose canonical example is Abel et al. model of MLTT [2].

We recall here the salient features of this flavour of models. First, the semantics of terms is described by recursion on some syntactic description of types. In presence of dependent types, we thus have to resort to some form of induction-recursion to define at the same time types inductively and terms recursively from types. Semantically, well-typedness of a term $\Vdash M \in A$ is typically defined through reduction to a weak-head normal form whose shape is constrained by A. Hence, these models qualify as realizability models. Then, when caring about the equational theory, which is the case when the source has a conversion rule and thus in MLTT, it is customary to consider a binary PER presentation $\Vdash M \equiv N \in A$ of the unary variant $\Vdash M \in A$. Finally, in these models all semantic relations are presheaves over the category of contexts and weakenings. This latter point is a departure from most common forms of realizability, and singles out the critical notion of *neutral terms*, i.e. normal forms stuck on a variable. It also makes it possible for the model to be complete with respect to the syntax, which allows proving valuable results such as decidability of type-checking.

Our model follows Abel's approach and the Rocq port [3] of the original Agda implementation. The major difference with Abel's complete model of MLTT is that we morally introduce side-effects in the semantics via a splitting monad similarly to [18]. All properties now live up to a finite tree of extensions of the current forcing conditions, i.e. our semantic interpretations are sheaves for the ℓ contexts. We refer to the Adjedj et al. paper [3] for the infrastructure, and we will focus on the major differences with the original model instead.

4.2 Reducibility

Abel-style models are built in two steps. The first step is called *reducibility* and is the actual model construction. The second step is called *validity* and consists in closing reducibility under substitution. In usual realizability models this part is often implicit in the soundness proof. We will concentrate on $\mathsf{MLTT}^{\mathsf{f}}$ reducibility and will only allude to validity, as the latter is not specific to our model.

A model is typically defined through (small) induction-recursion. To each type $\Gamma \vdash A$, one associates an inductively defined reducibility statement $\Gamma \Vdash A$. Then type convertibility, term typedness and term convertibility are defined by recursion on a proof of $\Gamma \Vdash A$. In our setting, things get more subtle. We indeed start by inductively defining what we call *strong reducibility* $\ell \mid \Gamma \Vdash^{\mathsf{s}} A$. Then, given a proof $\mathfrak{H}_A \in \ell \mid \Gamma \Vdash^{\mathsf{s}} A$, we define three reducibility relations at this type:

- strong reducible type convertibility $\ell \mid \Gamma \Vdash^{\mathsf{s}} A \equiv B \;/\; \mathfrak{H}_A$;
- strong reducible typedness $\ell \mid \Gamma \Vdash^{\mathsf{s}} M : A \;/\; \mathfrak{H}_A$;
- strong reducible term convertibility $\ell \mid \Gamma \Vdash^{\mathsf{s}} M \equiv N : A \;/\; \mathfrak{H}_A$.

The intuition is that strongly reducible types and terms reduce to a value *now*, at the current forcing condition ℓ. Then, to account for sheafification, we

derive what we call *split* or *weak* reducibility relations, which will be the ones that indeed interpret $\mathsf{MLTT}^{\mathsf{f}}$ judgments. Split reducibility is defined as the closure of strong reducibility under what amounts to the splitting monad in the presheaf model over ℓ contexts. The idea is that split reducible types and terms will *eventually* reduce to a value in every branch, once enough splits are performed. With $\mathfrak{H}_A \in \ell \mid \Gamma \Vdash^{\mathsf{f}} A$, it is formally defined as

$$\begin{aligned}
\ell \mid \Gamma \Vdash^{\mathsf{f}} A &:= \Sigma(d \in \mathfrak{D}\,\ell).\, \Pi\{\ell' \in \mathfrak{L}\}.\, d \triangleleft \ell' \to \ell' \mid \Gamma \Vdash^{\mathsf{s}} A \\
\ell \mid \Gamma \Vdash^{\mathsf{f}} \mathcal{J} \;/\; \mathfrak{H}_A &:= \Sigma(d \in \mathfrak{D}\,\ell).\, \Pi\{\ell' \in \mathfrak{L}\}\,(d_\varepsilon \in \mathfrak{H}_A.\pi_1 \triangleleft \ell').\ \ell' \mid \Gamma \Vdash^{\mathsf{s}} \mathcal{J} \;/\; (\mathfrak{H}_A.\pi_2\, d_\varepsilon)
\end{aligned}$$

The standard notion of syntactic neutrals from MLTT needs to be tweaked w.r.t. the new reduction rules. In addition to terms stuck on variables, we need to account for a call to the oracle stuck on a finite amount of S on top of a neutral. We give an excerpt of old cases together with the single new case below.

$$\frac{}{\mathsf{whne}\ x} \qquad \frac{\mathsf{whne}\ n}{\mathsf{whne}\ (n\ M)} \qquad \frac{\mathsf{whne}\ n}{\mathsf{whne}\ (\mathbb{N}_{\mathtt{rec}}\ P\ P_0\ P_{\mathsf{S}}\ n)} \qquad \cdots \qquad \frac{\mathsf{whne}\ n \qquad k \in \mathfrak{N}}{\mathsf{whne}\ (\alpha\ (S^k\ n))}$$

We sketch a few representative cases of strong reducibility. Due to the wealth of side-conditions in the definitions, writing it in full in print would not fit on the page, so we only present the core data. Importantly, all definitions contain syntactic well-formedness conditions ensuring completeness w.r.t. the syntax, but we skip these annotations here. Similarly, for readability we concentrate on the unary predicates but all these definitions also pack in the heterogeneous PER variant. We abuse notations and implicit arguments quite a bit. We write $\rho \in \Delta \subseteq \Gamma$ for weakenings and $M[\sigma]$ both for term substitution and weakening.

In Figure 3, we give reducibility for Π-types, the proverbial negative type. With this level of abstraction, it is clear that our metatheoretical semantics is call-by-value, even though the object theory enjoys a call-by-name equational theory. Barring the flurry of technical annotations needed to preserve completeness, the semantics is the one one would have expected from an effectful model. Most importantly, the codomain of Π-types is not strong but split reducible, since applying a function may perform side-effects, unlocking additional splits.

In Figure 4, we sketch reducibility of the typical positive type, namely $\mathbb{N}$. Contrarily to negative types, the relation is defined inductively, each constructor being mapped to an inductive case. Since we have first-class variables, we also need to freely add all well-typed neutrals. This was not visible in the term reducibility for Π-types because neutrals are closed by head application.

In semantic models, the hard part is to justify universes. As argued in Section 2.4, this is even a showstopper in traditional sheaf models. We have no such trouble, because types are directly interpreted by their code. The infrastructure is hard to erect for MLTT, which is why we have to use induction-recursion, but once this is done sheafification poses no further trouble. We model the universe as a positive type whose constructors are the type formers of the theory, and this is about it. No need for stacks, univalence or QIRTs. *It just works.*

We quickly review some important properties of the model. The first one, which we already advertized, is that the model is complete w.r.t. the syntax.

$$\ell \mid \Gamma \Vdash^{\mathsf{s}}_{\Pi} A := \left\{ \begin{array}{l} \_ \quad \in A \rightsquigarrow^*_\ell \Pi(x : F).\, G; \\ \mathfrak{H}_F \in \Pi\{\Delta\, \ell'\}\, (\rho \in \Delta \subseteq \Gamma).\, \ell' \preceq \ell \to \ell' \mid \Delta \Vdash^{\mathsf{s}} F[\rho]; \\ \mathfrak{H}_G \in \Pi\{\Delta\, \ell'\, a\}\, (\rho \in \Delta \subseteq \Gamma)\, (\tau \in \ell' \preceq \ell). \\ \qquad \ell' \mid \Delta \Vdash^{\mathsf{s}} a : F[\rho] \;/\; (\mathfrak{H}_F\ \rho\ \tau) \to \ell' \mid \Delta \Vdash^{\mathsf{f}} G[a, \rho]; \dots \end{array} \right\}$$

$$\ell \mid \Gamma \Vdash^{\mathsf{s}}_{\Pi} M : A \;/\; \mathfrak{H}_A := \left\{ \begin{array}{l} \_ \quad \in M \rightsquigarrow^*_\ell V; \\ \mathsf{app} \in \Pi\{\Delta\, \ell'\, a\}\, (\rho \in \Delta \subseteq \Gamma)\, (\tau \in \ell' \preceq \ell). \\ \qquad \Pi(\mathfrak{a} \in \ell' \mid \Delta \Vdash^{\mathsf{s}} a : F[\rho] \;/\; (\mathfrak{H}_F\ \rho\ \tau)). \\ \qquad\quad \ell' \mid \Delta \Vdash^{\mathsf{f}} V[\rho]\ a : G[a, \rho] \;/\; (\mathfrak{H}_G\ \rho\ \tau\ \mathfrak{a}); \dots \end{array} \right\}$$

Fig. 3. Strong type reducibility and term reducibility for Π-types

$$\ell \mid \Gamma \Vdash^{\mathsf{s}}_{\mathbb{N}} A \quad := \quad \{\ \_ \in A \rightsquigarrow^*_\ell \mathbb{N};\ \dots\ \}$$

$$\frac{M \rightsquigarrow^*_\ell 0 \qquad \dots}{\ell \mid \Gamma \Vdash^{\mathsf{s}}_{\mathbb{N}} M : A \;/\; \mathfrak{H}_A} \qquad \frac{M \rightsquigarrow^*_\ell n \qquad \mathsf{whne}\ n \qquad \dots}{\ell \mid \Gamma \Vdash^{\mathsf{s}}_{\mathbb{N}} M : A \;/\; \mathfrak{H}_A}$$

$$\frac{M \rightsquigarrow^*_\ell \mathsf{S}\ N \qquad \ell \mid \Gamma \Vdash^{\mathsf{s}}_{\mathbb{N}} N : A \;/\; \mathfrak{H}_A \qquad \dots}{\ell \mid \Gamma \Vdash^{\mathsf{s}}_{\mathbb{N}} M : A \;/\; \mathfrak{H}_A}$$

Fig. 4. Strong type reducibility and term reducibility for natural numbers

Theorem 4 (Completeness). *If* $\ell \mid \Gamma \Vdash^{\mathsf{f}} \mathcal{J}$ *then* $\ell \mid \Gamma \vdash \mathcal{J}$.

The second theorem is a critical semantic property of reducibility. In a nutshell, for all reducibility predicates, everything behaves as if type well-formedness were a mere proposition. That is, no matter the exact proof used to build the predicate, all proofs lead to logically equivalent relations.

Theorem 5 (Irrelevance). *If* $\mathfrak{H}_A, \mathfrak{H}'_A \in \ell \mid \Gamma \Vdash^{\mathsf{f}} A$ *and* $\ell \mid \Gamma \Vdash^{\mathsf{f}} \mathcal{J} \;/\; \mathfrak{H}_A$ *then* $\ell \mid \Gamma \Vdash^{\mathsf{f}} \mathcal{J} \;/\; \mathfrak{H}'_A$.

We cannot insist enough on the importance of this property. Without it, we would face coherence hell, and would have to resort to overly abstract categorical contraptions à la synthetic Tait computability [63] to obtain our model. Unfortunately, such approaches are totally unapplicable to a proof assistant such as Rocq, and we would have to buy into a much more expressive foundation that is not even implemented yet. Thanks to irrelevance, we can pretend that everything is propositional and hide coherence issues under a nice, practical abstraction.

Finally, we provide some sanity checks about the intended semantics of our model. In particular, our predicates behave respectively as presheaves over contexts and as sheaves for the topology induced by the splitting operation.

Theorem 6 (Intended semantics). *Reducibility is closed under weakenings, splits and neutrals.*

4.3 Validity and Soundness

To prove soundness of our model, we need to generalize reducibility a bit, a standard technique for MLTT models [2]. Note that forcing contexts $\ell \in \mathfrak{L}$ only get a

degenerate form of substitutions through reverse inclusions of forcing conditions. Hence, when defining validity, closure by forcing context substitutions is trivial.

Definition 13 (Validity). *We define semantic validity $\Vdash^{\mathsf{v}}$ for all our syntactic classes as usual by closing reducibility by all well-typed substitutions.*

We now have all the tools to state and prove the main result of our model, of which Theorem 2 is an immediate corollary.

Theorem 7 (Fundamental lemma). *If $\ell \mid \Gamma \vdash \mathcal{J}$ then $\ell \mid \Gamma \Vdash^{\mathsf{v}} \mathcal{J}$.*

Proof. We only focus on changes w.r.t. the proof for MLTT, i.e. the rules from Figure 2. The Split rule is literally baked in the model as per Theorem 6, leaving us with Eval and Oracle. We focus on Oracle as it subsumes the other. We have to prove that $\alpha\ M$ is split-reducible at type $\mathbb{B}$. By induction hypothesis, the argument $M : \mathbb{N}$ is a split reducible integer. Binding it, we can assume that M is a strongly reducible integer. By definition of $\Vdash^{\mathsf{s}}_{\mathbb{N}}$, we have two cases. Either M hereditarily reduces to a proper integer $\overline{\mathfrak{n}}$, or to a term of the form $\mathsf{S}^k\ n$ for some neutral n. In the first case we can split on $\overline{\mathfrak{n}}$ to conclude by reduction to a concrete $\overline{\mathfrak{b}}$. In the second case, the whole expression is a neutral, hence reducible.

5 Conclusion

The different approaches to building models discussed in this paper can be framed along three axes: *realizability*, *open terms* and *sheafness*. Figure 5 illustrates this classification by placing some representative models of MLTT from the literature on a cube. The realizability axis opposes computation-free models to models based on reduction. The open term axis contrasts semantics defined with respect to closed terms only to models with a first-class notion of variable, i.e. which are presheaves on contexts. Finally, the sheafness axis delineates models of "pure" MLTT, with the usual notion of canonicity, from those where side-effects in the semantics weaken canonicity. To the best of our knowledge, our model is the only mechanized MLTT model that features all these properties.

As mentioned in Section 3.5, we believe that adding the continuity principle to $\mathsf{MLTT}^{\mathsf{f}}$ is within grasp. All the necessary building blocks are already there, we just need a local context of oracles rather than a single global one, and add term that exploits the weak canonicity theorem in an internal way. Similarly, we have not formally proved decidability of type-checking for $\mathsf{MLTT}^{\mathsf{f}}$ yet, but we expect it to be a minor variation on the already mechanized proof for MLTT.

A more difficult task would be to define a variant of $\mathsf{MLTT}^{\mathsf{f}}$ for more general Grothendieck topologies. As explained in Section 3.1, branching for Cohen reals is both finite and disjoint. Finiteness is a must-have if we want to keep decidability of type-checking, but the disjointness condition is very restrictive. Studying a type-theoretic interpretation of the algebraic closure of a field, for which Coquand and Manaa have provided a suitable site model in a intuitionistic

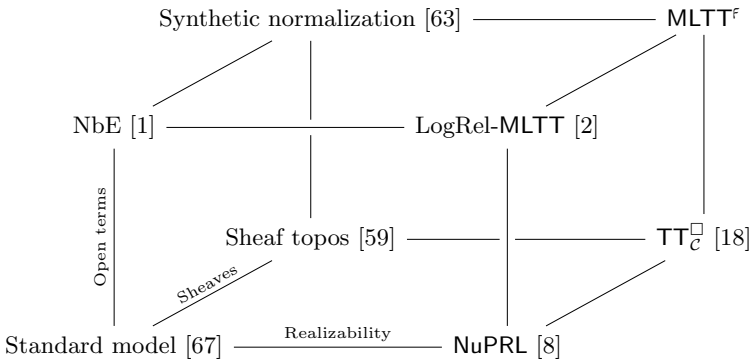

Fig. 5. The Model Cube

higher-order logic [48], would be a first step towards less elementary models. The Zariski topos [14] is another interesting and historically significant candidate.

More generally, recent works on dynamical methods in constructive mathematics, as mentioned in Section 2.5, provide potentially interesting type-theoretic models. Some of these results are quite spectacular. For instance, rephrasing classical local-global principles in algebraic geometry unveiled a simplification and generalization of previous proofs of Serre's conjecture (Quillen-Suslin's theorem) in commutative algebra [46]. Dynamical methods are actually about turning the oracle-based algorithm suggested by a classical proof into a branching process, e.g. some dialogue tree. Originally formulated in a categorical framework based on sketches [28], dynamical methods have also found significant appplications in computer algebra, e.g. for computing with algebraic numbers [27], transseries [9], etc. We thus believe in the interest of revisiting Coquand, Lombardi et al.'s research program, coined *hidden constructions in abstract algebra* in a series of papers, through the type-theoretic perspective advocated in the present paper.

References

1. Abel, A., Aehlig, K., Dybjer, P.: Normalization by evaluation for martin-löf type theory with one universe. In: Fiore, M. (ed.) Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS 2007, New Orleans, LA, USA, April 11-14, 2007. Electronic Notes in Theoretical Computer Science, vol. 173, pp. 17–39. Elsevier (2007). https://doi.org/10.1016/J.ENTCS.2007.02.025, https://doi.org/10.1016/j.entcs.2007.02.025
2. Abel, A., Öhman, J., Vezzosi, A.: Decidability of conversion for type theory in type theory. Proc. ACM Program. Lang. **2**(POPL) (dec 2017). https://doi.org/10.1145/3158111, https://doi.org/10.1145/3158111
3. Adjedj, A., Lennon-Bertrand, M., Maillard, K., Pédrot, P., Pujet, L.: Martin-löf à la coq. In: Timany, A., Traytel, D., Pientka, B., Blazy, S. (eds.) Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024. pp. 230–

245. ACM (2024). https://doi.org/10.1145/3636501.3636951, https://doi.org/10.1145/3636501.3636951
4. Ahman, D.: Handling fibred algebraic effects. Proc. ACM Program. Lang. **2**(POPL), 7:1–7:29 (2018). https://doi.org/10.1145/3158095, https://doi.org/10.1145/3158095
5. Ahman, D., Ghani, N., Plotkin, G.D.: Dependent types and fibred computational effects. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9634, pp. 36–54. Springer (2016). https://doi.org/10.1007/978-3-662-49630-5_3, https://doi.org/10.1007/978-3-662-49630-5_3
6. Altenkirch, T., Capriotti, P., Dijkstra, G., Kraus, N., Forsberg, F.N.: Quotient inductive-inductive types. In: Baier, C., Lago, U.D. (eds.) Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 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. 10803, pp. 293–310. Springer (2018). https://doi.org/10.1007/978-3-319-89366-2_16, https://doi.org/10.1007/978-3-319-89366-2_16
7. Altenkirch, T., McBride, C., Swierstra, W.: Observational equality, now! In: Stump, A., Xi, H. (eds.) Proceedings of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, October 5, 2007. pp. 57–68. ACM (2007). https://doi.org/10.1145/1292597.1292608, https://doi.org/10.1145/1292597.1292608
8. Anand, A., Rahli, V.: Towards a formally verified proof assistant. In: Klein, G., Gamboa, R. (eds.) Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8558, pp. 27–44. Springer (2014). https://doi.org/10.1007/978-3-319-08970-6_3, https://doi.org/10.1007/978-3-319-08970-6_3
9. Aschenbrenner, M., Dries, L.v.d., Hoeven, J.v.d.: Asymptotic Differential Algebra and Model Theory of Transseries. No. 195 in Annals of Mathematics studies, Princeton University Press (2017), http://arxiv.org/abs/1509.02588
10. Authors, T.S.P.: Stacks project, https://stacks.math.columbia.edu/
11. Baillon, M., Forster, Y., Mahboubi, A., Pédrot, P.M., Piquerez, M.: A Zoo of Continuity Properties in Constructive Type Theory. In: Fernández, M. (ed.) 10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025). Leibniz International Proceedings in Informatics (LIPIcs), vol. 337, pp. 9:1–9:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2025). https://doi.org/10.4230/LIPIcs.FSCD.2025.9, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2025.9
12. Baillon, M., Mahboubi, A., Pédrot, P.: Gardening with the pythia A model of continuity in a dependent setting. In: Manea, F., Simpson, A. (eds.) 30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference). LIPIcs, vol. 216, pp. 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.CSL.2022.5, https://doi.org/10.4230/LIPIcs.CSL.2022.5
13. Bernardy, J., Lasson, M.: Realizability and parametricity in pure type systems. In: Hofmann, M. (ed.) Foundations of Software Science and Computational Struc-

tures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6604, pp. 108–122. Springer (2011). https://doi.org/10.1007/978-3-642-19805-2_8, https://doi.org/10.1007/978-3-642-19805-2_8
14. Blechschmidt, I.: Using the internal language of toposes in algebraic geometry. Ph.D. thesis, Augsburg University (2017), https://rawgit.com/iblech/internal-methods/master/notes.pdf
15. Brede, N., Herbelin, H.: On the logical structure of choice and bar induction principles. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021. pp. 1–13. IEEE (2021). https://doi.org/10.1109/LICS52264.2021.9470523, https://doi.org/10.1109/LICS52264.2021.9470523
16. Cavallo, E., Mörtberg, A., Swan, A.W.: Unifying cubical models of univalent type theory. In: Fernández, M., Muscholl, A. (eds.) 28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain. LIPIcs, vol. 152, pp. 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPICS.CSL.2020.14, https://doi.org/10.4230/LIPIcs.CSL.2020.14
17. Cohen, L., Rahli, V.: Realizing continuity using stateful computations. In: Klin, B., Pimentel, E. (eds.) 31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland. LIPIcs, vol. 252, pp. 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.CSL.2023.15, https://doi.org/10.4230/LIPIcs.CSL.2023.15
18. Cohen, L., Rahli, V.: $\mathrm{TT}^{\Box}_{\mathcal{C}}$: A family of extensional type theories with effectful realizers of continuity. Logical Methods in Computer Science **Volume 20, Issue 2**, 18 (Jun 2024). https://doi.org/10.46298/lmcs-20(2:18)2024, https://lmcs.episciences.org/11666
19. Cohen, L., da Rocha Paiva, B., Rahli, V., Tosun, A.: Inductive continuity via Brouwer trees. In: Leroux, J., Lombardy, S., Peleg, D. (eds.) 48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France. LIPIcs, vol. 272, pp. 37:1–37:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.MFCS.2023.37, https://doi.org/10.4230/LIPIcs.MFCS.2023.37
20. Coquand, T.: Dynamical method in algebra: A survey. In: Mayer, M.C., Pirri, F. (eds.) Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003. Proceedings. Lecture Notes in Computer Science, vol. 2796, p. 2. Springer (2003). https://doi.org/10.1007/978-3-540-45206-5_2, https://doi.org/10.1007/978-3-540-45206-5_2
21. Coquand, T.: A completeness proof for geometrical logic. Logic, Methodology and Philosophy of Sciences (2005)
22. Coquand, T., Danielsson, N.A., Escardó, M.H., Norell, U., Xu, C.: Negative consistent axioms can be postulated without loss of canonicity (2013), https://martinescardo.github.io/papers/negative-axioms.pdf
23. Coquand, T., Jaber, G.: A note on forcing and type theory. Fundam. Informaticae **100**(1-4), 43–52 (2010). https://doi.org/10.3233/FI-2010-262, https://doi.org/10.3233/FI-2010-262
24. Coquand, T., Jaber, G.: A computational interpretation of forcing in type theory. In: Dybjer, P., Lindström, S., Palmgren, E., Sundholm, G. (eds.) Epistemology

versus Ontology - Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf, Logic, Epistemology, and the Unity of Science, vol. 27, pp. 203–213. Springer (2012). https://doi.org/10.1007/978-94-007-4435-6_10, https://doi.org/10.1007/978-94-007-4435-6_10
25. Coquand, T., Ruch, F., Sattler, C.: Constructive sheaf models of type theory. Math. Struct. Comput. Sci. **31**(9), 979–1002 (2021). https://doi.org/10.1017/S0960129521000359, https://doi.org/10.1017/S0960129521000359
26. Coste, M., Lombardi, H., Roy, M.: Dynamical method in algebra: effective nullstellensätze. Ann. Pure Appl. Log. **111**(3), 203–256 (2001). https://doi.org/10.1016/S0168-0072(01)00026-4, https://doi.org/10.1016/S0168-0072(01)00026-4
27. Dora, J.D., Dicrescenzo, C., Duval, D.: About a new method for computing in algebraic number fields. In: European Conference on Computer Algebra (2). Lecture Notes in Computer Science, vol. 204, pp. 289–290. Springer (1985)
28. Duval, D., Reynaud, J.C.: Sketches and computation – ii: dynamic evaluation and applications. Mathematical Structures in Computer Science **4**(2), 239–271 (1994). https://doi.org/10.1017/S096012950000044X
29. Escardó, M., Xu, C.: A constructive manifestation of the Kleene-Kreisel continuous functionals. Ann. Pure Appl. Log. **167**(9), 770–793 (2016). https://doi.org/10.1016/J.APAL.2016.04.011, https://doi.org/10.1016/j.apal.2016.04.011
30. Escardó, M.H., da Rocha Paiva, B., Rahli, V., Tosun, A.: Internal effectful forcing in system T. In: Fernández, M. (ed.) 10th International Conference on Formal Structures for Computation and Deduction, FSCD 2025, July 14-20, 2025, Birmingham, UK. LIPIcs, vol. 337, pp. 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2025). https://doi.org/10.4230/LIPICS.FSCD.2025.19, https://doi.org/10.4230/LIPIcs.FSCD.2025.19
31. Escardó, M.H.: Continuity of Gödel's System T definable functionals via effectful forcing. Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2013, New Orleans, LA, USA, June 23-25, 2013 (2013). https://doi.org/10.1016/j.entcs.2013.09.010, https://doi.org/10.1016/j.entcs.2013.09.010
32. Escardó, M.H., Xu, C.: The inconsistency of a brouwerian continuity principle with the Curry-Howard interpretation. 13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland (2015). https://doi.org/10.4230/LIPIcs.TLCA.2015.153, https://doi.org/10.4230/LIPIcs.TLCA.2015.153
33. Fiore, M.P., Pitts, A.M., Steenkamp, S.C.: Quotients, inductive types, and quotient inductive types. Log. Methods Comput. Sci. **18**(2) (2022). https://doi.org/10.46298/LMCS-18(2:15)2022, https://doi.org/10.46298/lmcs-18(2:15)2022
34. Fujiwara, M., Kawai, T.: Equivalence of bar induction and bar recursion for continuous functions with continuous moduli. Ann. Pure Appl. Log. **170**(8), 867–890 (2019). https://doi.org/10.1016/J.APAL.2019.04.001, https://doi.org/10.1016/j.apal.2019.04.001
35. Gilbert, G., Cockx, J., Sozeau, M., Tabareau, N.: Definitional proof-irrelevance without K. Proc. ACM Program. Lang. **3**(POPL), 3:1–3:28 (2019). https://doi.org/10.1145/3290316, https://doi.org/10.1145/3290316
36. Girard, J.Y., Taylor, P., Lafont, Y.: Proofs and Types. Cambridge University Press, Cambridge (1989), http://www.worldcat.org/isbn/0521371813
37. Gratzer, D.: An inductive-recursive universe generic for small families. CoRR **abs/2202.05529** (2022), https://arxiv.org/abs/2202.05529

38. Gratzer, D., Kavvos, G.A., Nuyts, A., Birkedal, L.: Multimodal dependent type theory. In: Hermanns, H., Zhang, L., Kobayashi, N., Miller, D. (eds.) LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020. pp. 492–506. ACM (2020). https://doi.org/10.1145/3373718.3394736, https://doi.org/10.1145/3373718.3394736
39. Gratzer, D., Shulman, M., Sterling, J.: Strict universes for grothendieck topoi. CoRR **abs/2202.12012** (2022), https://arxiv.org/abs/2202.12012
40. Griffin, T.: A formulae-as-types notion of control. In: Allen, F.E. (ed.) Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990. pp. 47–58. ACM Press (1990). https://doi.org/10.1145/96709.96714, https://doi.org/10.1145/96709.96714
41. Hancock, P.G., McBride, C., Ghani, N., Malatesta, L., Altenkirch, T.: Small induction recursion. In: Hasegawa, M. (ed.) Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7941, pp. 156–172. Springer (2013). https://doi.org/10.1007/978-3-642-38946-7_13, https://doi.org/10.1007/978-3-642-38946-7_13
42. Kaposi, A.: Towards quotient inductive-inductive-recursive types. In: 29th International Conference on Types for Proofs and Programs, TYPES 2023. pp. 124–126 (2023), https://types2023.webs.upv.es/TYPES2023.pdf
43. Kaposi, A., Kovács, A., Altenkirch, T.: Constructing quotient inductive-inductive types. Proc. ACM Program. Lang. **3**(POPL), 2:1–2:24 (2019). https://doi.org/10.1145/3290315, https://doi.org/10.1145/3290315
44. Kiselyov, O., Shan, C.: Delimited continuations in operating systems. In: Kokinov, B.N., Richardson, D.C., Roth-Berghofer, T., Vieu, L. (eds.) Modeling and Using Context, 6th International and Interdisciplinary Conference, CONTEXT 2007, Roskilde, Denmark, August 20-24, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4635, pp. 291–302. Springer (2007). https://doi.org/10.1007/978-3-540-74255-5_22, https://doi.org/10.1007/978-3-540-74255-5_22
45. Kleene, S.C.: Recursive Functionals and Quantifiers of Finite Types I. Transactions of the American Mathematical Society **91**(1), 1 (Apr 1959). https://doi.org/10.2307/1993145, https://www.jstor.org/stable/1993145?origin=crossref
46. Lombardi, H., Quitté, C., Yengui, I.: Hidden constructions in abstract algebra. VI. The theorem of Maroscia and Brewer & Costa. J. Pure Appl. Algebra **212**(7), 1575–1582 (2008). https://doi.org/10.1016/j.jpaa.2007.10.009, https://doi.org/10.1016/j.jpaa.2007.10.009
47. Mac Lane, S., Moerdijk, I.: Sheaves in Geometry and Logic a First Introduction to Topos Theory. Springer New York, New York, NY (1992), http://link.springer.com/book/10.1007/978-1-4612-0927-0
48. Mannaa, B., Coquand, T.: A sheaf model of the algebraic closure. In: CL&C. EPTCS, vol. 164, pp. 18–32 (2014)
49. Miquel, A.: 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. pp. 197–206. IEEE Computer Society (2011). https://doi.org/10.1109/LICS.2011.47, https://doi.org/10.1109/LICS.2011.47
50. Palmgren, E.: From intuitionistic to point-free topology: on the foundation of homotopy theory. In: Logicism, intuitionism, and formalism, Synth. Libr., vol. 341, pp. 237–253. Springer, Dordrecht (2009). https://doi.org/10.1007/978-1-4020-8926-8_12, https://doi.org/10.1007/978-1-4020-8926-8_12

51. Paulin-Mohring, C.: Définitions Inductives en Théorie des Types. Accreditation to supervise research, Université Claude Bernard - Lyon I (Dec 1996), https://theses.hal.science/tel-00431817
52. Pédrot, P.: Russian constructivism in a prefascist theory. In: Hermanns, H., Zhang, L., Kobayashi, N., Miller, D. (eds.) LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020. pp. 782–794. ACM (2020). https://doi.org/10.1145/3373718.3394740, https://doi.org/10.1145/3373718.3394740
53. Pédrot, P., Tabareau, N.: 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. pp. 1–12. IEEE Computer Society (2017). https://doi.org/10.1109/LICS.2017.8005113, https://doi.org/10.1109/LICS.2017.8005113
54. Pédrot, P., Tabareau, N.: The fire triangle: how to mix substitution, dependent elimination, and effects. Proc. ACM Program. Lang. **4**(POPL), 58:1–58:28 (2020). https://doi.org/10.1145/3371126, https://doi.org/10.1145/3371126
55. Poirier, H.: La vraie nature de l'intelligence. Science & Vie **1013**, 38–57 (February 2002)
56. Rahli, V., Bickford, M.: Validating brouwer's continuity principle for numbers using named exceptions. Math. Struct. Comput. Sci. **28**(6), 942–990 (2018). https://doi.org/10.1017/S0960129517000172, https://doi.org/10.1017/S0960129517000172
57. Rahli, V., Bickford, M., Cohen, L., Constable, R.L.: Bar induction is compatible with constructive type theory. J. ACM **66**(2), 13:1–13:35 (2019). https://doi.org/10.1145/3305261, https://doi.org/10.1145/3305261
58. Rahli, V., Bickford, M., Constable, R.L.: Bar induction: The good, the bad, and the ugly. In: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017. pp. 1–12. IEEE Computer Society (2017). https://doi.org/10.1109/LICS.2017.8005074, https://doi.org/10.1109/LICS.2017.8005074
59. Rijke, E., Shulman, M., Spitters, B.: Modalities in homotopy type theory. Log. Methods Comput. Sci. **16**(1) (2020). https://doi.org/10.23638/LMCS-16(1:2)2020, https://doi.org/10.23638/LMCS-16(1:2)2020
60. Serre, J.P.: Faisceaux algébriques cohérents. Ann. of Math. (2) **61**, 197–278 (1955). https://doi.org/10.2307/1969915, https://doi.org/10.2307/1969915
61. Sherman, B., Sciarappa, L., Chlipala, A., Carbin, M.: Computable decision making on the reals and other spaces: via partiality and nondeterminism. In: LICS. pp. 859–868. ACM (2018)
62. Sterling, J.: Higher order functions and brouwer's thesis. J. Funct. Program. **31**, e11 (2021). https://doi.org/10.1017/S0956796821000095, https://doi.org/10.1017/S0956796821000095
63. Sterling, J.: First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory. Ph.D. thesis, Carnegie Mellon University, USA (2022). https://doi.org/10.1184/R1/19632681.V1, https://doi.org/10.1184/r1/19632681.v1
64. Streicher, T.: Universes in toposes. In: Crosilla, L., Schuster, P.M. (eds.) From sets and types to topology and analysis - Towards practicable foundations for constructive mathematics, Oxford logic guides, vol. 48. Oxford University Press (2005)
65. Troelstra, A., van Dalen, D.: Constructivism in Mathematics An Introduction, vol. 121. Elsevier (1988)

66. Univalent Foundations Program, T.: Homotopy Type Theory: Univalent Foundations of Mathematics. https://homotopytypetheory.org/book, Institute for Advanced Study (2013)
67. Werner, B.: Une Théorie des Constructions Inductives. Ph.D. thesis, Paris Diderot University, France (1994), https://tel.archives-ouvertes.fr/tel-00196524
68. Xia, L., Zakowski, Y., He, P., Hur, C., Malecha, G., Pierce, B.C., Zdancewic, S.: Interaction trees: representing recursive and impure programs in coq. Proc. ACM Program. Lang. **4**(POPL), 51:1–51:32 (2020). https://doi.org/10.1145/3371119, https://doi.org/10.1145/3371119
69. Xu, C., Escardó, M.H.: A constructive model of uniform continuity. In: Hasegawa, M. (ed.) Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7941, pp. 236–249. Springer (2013). https://doi.org/10.1007/978-3-642-38946-7_18, https://doi.org/10.1007/978-3-642-38946-7_18