Gardening with the Pythia A model of continuity in a dependent setting

Martin Baillon, Assia Mahboubi, Pierre-Marie Pédrot

Gardening with the Pythia
A model of continuity in a dependent setting

Martin Baillon ✉
INRIA and LS2N, France

Assia Mahboubi ✉
INRIA and LS2N, France

Pierre-Marie Pédrot ✉
INRIA and LS2N, France

Abstract

We generalize to a rich dependent type theory a proof originally developed by Escardó that all System T functionals are continuous. It relies on the definition of a syntactic model of Baclofen Type Theory, a type theory where dependent elimination must be strict, into the Calculus of Inductive Constructions. The model is given by three translations: the *axiom translation*, that adds an oracle to the context; the *branching translation*, based on the dialogue monad, turning every type into a tree; and finally, a layer of *algebraic binary parametricity*, binding together the two translations. In the resulting type theory, every function $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is externally continuous.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type theory, continuity, syntactic model

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.13

Introduction

A folklore result from computatibility theory is that any computable function must be continuous [4]. A more operational way to phrase this property is that a function can only inspect a finite amount of its argument to produce a finite amount of output. There are many ways to prove, or even merely state, this theorem, since it depends in particular on how computable functions are represented [18, 37, 21]. Assuming we pick the λ-calculus as our favourite computational system, a modern straightforward proof would boil down to building a semantic model, typically some flavour of complete partial orders (cpos). By construction, cpos are a specific kind of topological spaces, and all functions are interpreted as continuous functions in the model. For some types simple enough, cpo-continuity implies continuity in the traditional sense, thus proving the claim.

Instead of going down the semantic route, Escardó developed an alternative syntactic technique called *effectful forcing* [11] to prove the continuity of all functionals $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ definable in System T. While semantic models such as cpos are defined inside a non-computational metatheory, Escardó's technique amounts to building a model of System T inside the dependent type theory MLTT, which is intrinsically a programming language with a built-in notion of computation. The *effectful* epithet is justified by the fact that the model construction extends System T with two different kinds of side-effects, and constrains those two extensions by a logical relation.

A clear advantage of this approach is that there is a simple computational explanation for why continuity holds in terms of elementary side-effects, which is not immediately apparent in cpos. This computational aspect is reminiscent of a similar realizability model of NuPRL internalizing continuity with a system of fresh exceptions [30]. But contrarily to the latter, the purely syntactic nature of Escardó's argument can actually be leveraged to interpret

30th EACSL Annual Conference on Computer Science Logic (CSL 2022).
Editors: Florin Manea and Alex Simpson; Article No. 13; pp. 13:1–13:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

much richer languages than System T while preserving desirable properties that would be lost with a semantic realizability model, such as decidability of type-checking.

Indeed, it happens that this technique can be formulated pretty much straightforwardly as a syntactic model [13, 33]. From this initial observation, we show in this paper how it can be generalized to a rich dependent type theory similar to MLTT, notably featuring universes and a form of large dependent elimination. Unfortunately, since Escardó's model introduces observable side-effects in the sense of [25], the type theory resulting from our generalization needs to be slightly weakened down or would otherwise be inconsistent. This effectively means we provide a model of Baclofen Type Theory (BTT) rather than MLTT. The main difference between those two theories lies in the typing rule for dependent elimination [28]. In MLTT, the predicate of a dependent elimination is arbitrary, while it must be computationally strict in BTT. This is discussed in detail in Section 1.2.

In the end we recover the continuity result of Escardó applied to BTT rather than System T. That is, from any $\vdash_{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ we get a proof that it is continuous.

Plan of the paper.

Section 1 exposes preliminaries that are needed to understand this paper. In Section 2, we describe a particular structure known as dialogue trees that will be critical for the rest of the paper. Section 3 is dedicated to the model contruction *per se*. Section 4 provides the proof that all functions $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ of this model are indeed continuous. Section 5 frames our result in a larger context and discusses potential extensions.

1 Preliminaries

1.1 Syntactic Conventions

In this paper, we will work with various flavours of type theory. Our base system will always be CC_ω, a predicative version of the Calculus of Constructions featuring an infinite hierarchy of universes $\square_i$ and dependent functions, which is summarized in Figure 1. We add inductive types to this negative fragment, leading to the Calculus of Inductive Constructions (CIC) or Baclofen Type Theory (BTT) depending on the formulation of dependent elimination. We will summarize the defining features of BTT in the next section. Since we will manipulate several type theories, we will write $\mathsf{T} := \mathsf{CIC}$ as a notational device to make explicit that we are referring to the ambient type theory.

For brevity, we will define inductive types in a Coq-like syntax, but we will use a pattern-matching syntax à la Agda for definitions by induction. As an example, we give below the definition of natural numbers and the resulting formal typing and conversion rules.

$$\texttt{Inductive}\ \mathbb{N} := \mathsf{O} : \mathbb{N} \mid \mathsf{S} : \mathbb{N} \to \mathbb{N}$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathbb{N} : \square_i} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{O} : \mathbb{N}} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{S} : \mathbb{N} \to \mathbb{N}}$$

$$\frac{\Gamma \vdash P : \mathbb{N} \to \square_i \qquad \Gamma \vdash t_{\mathsf{O}} : P\ \mathsf{O} \qquad \Gamma \vdash t_{\mathsf{S}} : \Pi n : \mathbb{N}.\, P\ n \to P\ (\mathsf{S}\ n)}{\Gamma \vdash \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}} : \Pi n : \mathbb{N}.\, P\ n}$$

$$\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ \mathsf{O} \equiv t_{\mathsf{O}} \qquad \mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ (\mathsf{S}\ n) \equiv t_{\mathsf{S}}\ n\ (\mathbb{N}_{\mathbf{ind}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}}\ n)$$

We will mostly ignore universe constraints and silently rely on typical ambiguity for the sake of readability. Definitions indexed by universe variables i, j are meant to be universe-

$$A, B, M, N ::= \square_i \mid x \mid M\ N \mid \lambda x : A.\, M \mid \Pi x : A.\, M \mid \Sigma x : A.\, B \mid M.\pi_1 \mid M.\pi_2 \mid (M, N)$$

$$\Gamma, \Delta ::= \cdot \mid \Gamma, x : A$$

$$\frac{}{\vdash \cdot} \qquad \frac{\Gamma \vdash A : \square_i}{\vdash \Gamma, x : A} \qquad \frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\vdash \Gamma \qquad i < j}{\Gamma \vdash \square_i : \square_j}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma \vdash M : B}{\Gamma, x : A \vdash M : B} \qquad \frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Pi x : A.\, B : \square_{\max(i,j)}}$$

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B\{x := N\}} \qquad \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : \square_i}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : \square_i \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash M : B}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \Sigma x : A.\, B : \square_{\max(i,j)}} \qquad \frac{\Gamma \vdash M : \Sigma x : A.\, B}{\Gamma \vdash M.\pi_1 : A} \qquad \frac{\Gamma \vdash M : \Sigma x : A.\, B}{\Gamma \vdash M.\pi_2 : B\{x := M.\pi_1\}}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B\{x := M\} \qquad \Sigma x : A.\, B : \square_i}{\Gamma \vdash (M, N) : \Sigma x : A.\, B} \qquad \text{(conversion omitted)}$$

Figure 1 Syntax of CC_ω extended with Σ-types

polymorphic in those variables [34]. All the translations we will give can be annotated with universe variables to handle an arbitrary hierarchy of universes, but we will refrain from doing so. We sometimes use implicit function arguments, which we bind with braces in definitions.

Writing explicit terms in type theory can quickly become cumbersome for proofs, hence we will omit them when the computational content is not important and write instead an underscore as in $\vdash \_ : A$.

1.2 Dependence in an Effectful Setting

In this paper, we will build type theories that feature computational effects. Regrettably, adding effects to dependent type theory is not without consequences. They make indeed observable the difference between call-by-value and call-by-name [20], a phenomenon that puts us in front of a dilemma [25]. If we stick to by-name, we preserve the behaviour of the negative fragment, i.e. Π-types, but we break dependent elimination. If we stick to by-value, we now preserve dependent elimination, but functions become quite different to what one is used to, as substitution is now restricted to syntactic values. For historical reasons, there is a clear bias in type theory towards by-name, and we will follow the same doctrine.

As explained above, an effectful call-by-name type theory does not support full-blown dependent elimination in general. As dependent elimination is quite a critical feature [23], this might look concerning. Thankfully, most effectful theories we know of support a restricted form of it, which essentially amounts to forcing the predicate used in the eliminator to be *strict* in its inductive argument[1]. The resulting theory is known as Baclofen Type Theory [28], or BTT for short.

Contrarily to MLTT, which has a single dependent eliminator $\mathcal{I}_{\mathbf{ind}}$ for any given inductive type $\mathcal{I}$, BTT has two eliminators: a non-dependent one $\mathcal{I}_{\mathbf{cse}}$, and a strict dependent one $\mathcal{I}_{\mathbf{rec}}$. These three eliminators enjoy the same computational ι-rules, i.e. they reduce on constructors. The difference lies in their typing rules. The predicate of $\mathcal{I}_{\mathbf{cse}}$ does not depend on its inductive argument, i.e. it is basically simply-typed. Meanwhile, the predicate of $\mathcal{I}_{\mathbf{rec}}$ is wrapped in a *storage operator* [19] $\mathcal{I}_{\mathbf{str}}$ that locally evaluates its argument in a by-value fashion. This guarantees that it will only ever be applied to values, and never to effectful, or non-standard, inductive terms. The important observation is that $\mathcal{I}_{\mathbf{str}}$ can be defined in a systematic way out of $\mathcal{I}_{\mathbf{cse}}$, namely it is simply an η-expansion in CPS style. To make things self-contained, we recall below the BTT eliminators for $\mathbb{N}$.

$$\frac{\Gamma \vdash P : \square \qquad \Gamma \vdash t_{\mathsf{O}} : P \qquad \Gamma \vdash t_{\mathsf{S}} : \mathbb{N} \to P \to P}{\Gamma \vdash \mathbb{N}_{\mathbf{cse}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}} : \mathbb{N} \to P}$$

$$\frac{\Gamma \vdash P : \mathbb{N} \to \square \qquad \Gamma \vdash t_{\mathsf{O}} : \mathbb{N}_{\mathbf{str}}\ \mathsf{O}\ P \qquad \Gamma \vdash t_{\mathsf{S}} : \Pi(n : \mathbb{N}).\, \mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathsf{S}\ n)\ P}{\Gamma \vdash \mathbb{N}_{\mathbf{rec}}\ P\ t_{\mathsf{O}}\ t_{\mathsf{S}} : \Pi(n : \mathbb{N}).\, \mathbb{N}_{\mathbf{str}}\ n\ P}$$

where
$$\begin{aligned}&\mathbb{N}_{\mathbf{str}}\ (n : \mathbb{N})\ (P : \mathbb{N} \to \square) : \square := \\ &\quad \mathbb{N}_{\mathbf{cse}}\ ((\mathbb{N} \to \square) \to \square)\ (\lambda(Q : \mathbb{N} \to \square).\, Q\ \mathsf{O}) \\ &\qquad (\lambda(m : \mathbb{N})\,(\_ : (\mathbb{N} \to \square) \to \square)\,(Q : \mathbb{N} \to \square).\, Q\ (\mathsf{S}\ m))\ n\ P.\end{aligned}$$

From within CIC, one can prove that $\Pi(n : \mathbb{N})\,(P : \mathbb{N} \to \square).\, \mathbb{N}_{\mathbf{str}}\ n\ P = P\ n$. Hence, this strictification is akin to double-negation translation, in so far as BTT is finer-grained than CIC, just as LJ is finer-grained than LK where $\neg\neg A \leftrightarrow A$. Note that in particular BTT is a subset of CIC, a fact on which we will rely on silently in this paper.

1.3 Continuity

In the remainder of this article, we suppose given two types $\vdash_{\mathsf{T}} \mathbf{I} : \square_0$ and $\vdash_{\mathsf{T}} \mathbf{O} : \mathbf{I} \to \square_0$. For simplicity, we set them in the lowest universe level, but all of the constructions to come can handle an arbitrary base level by bumping them by an appropriate amount.

The type $\mathbf{I}$ is to be understood as a type of input or questions to a black-box, called an oracle. Dually, $\mathbf{O}$ is the type of output or answers from the oracle. Since $\mathbf{O}$ depends on $\mathbf{I}$, we can encode a pretty much arbitrary interaction. Finally, we define the type of oracles as $\mathbf{Q} := \Pi(i : \mathbf{I}).\, \mathbf{O}\ i$. A reader more inclined towards computer science could also consider that $\mathbf{O}$ and $\mathbf{I}$ describe an interface for system calls, and $\mathbf{Q}$ is the type of operating systems implementing these calls.

Let us formally define the notion of continuity over $\mathbf{Q}$.

▶ **Definition 1.** *Given $\alpha_1, \alpha_2 : \mathbf{Q}$ and $\ell : \mathsf{list}\ \mathbf{I}$, we say that α_1 and α_2 are finitely equal on ℓ, written $\alpha_1 \approx_\ell \alpha_2$ when the following inductively defined predicate holds.*

$$\frac{}{\alpha_1 \approx_{\mathsf{nil}} \alpha_2} \qquad \frac{\alpha_1\ i = \alpha_2\ i \qquad \alpha_1 \approx_l \alpha_2}{\alpha_1 \approx_{(\mathsf{cons}\ i\ \ell)} \alpha_2}$$

[1] As in programming language theory, not as in higher category theory.

▶ **Definition 2.** *We say that a function is continuous when it satisfies the continuity predicate*

$$\begin{array}{lcl} \mathcal{C} & : & \Pi\{A : \square\}.\,(\mathbf{Q} \to A) \to \square \\ \mathcal{C}\ f & := & \Pi(\alpha : \mathbf{Q}).\,\Sigma(\ell : \mathsf{list}\ \mathbf{I}).\,\Pi(\beta : \mathbf{Q}).\,\alpha \approx_\ell \beta \to f\ \alpha = f\ \beta. \end{array}$$

This definition captures in a generic way the intuitive notion that a computable functional only needs a finite amount of information from its argument to produce an output. Note that in particular the list of points ℓ where the function is evaluated depends on the argument α, so this notion of continuity is weaker than uniform continuity, where the two quantifiers for ℓ and α are swapped. Depending on the expressivity of T, one can also consider weaker variants where the existential is squashed with various proof-irrelevant modalities [12, 30, 31].

2 Dialogue Trees and Intensionality

2.1 Talking with Trees

It is now time to justify the title of this article by giving some explanations on the links between trees, oracles and functions. We consider an operator $\mathfrak{D} : \square \to \square$, which given a type $A : \square$, associates the type of well-founded trees, with leaves labelled in A. Each inner node is labelled with a certain $i : \mathbf{I}$ and has $\mathbf{O}\ i$ children. In T, this amounts to the following inductive definition:

$$\texttt{Inductive}\ \mathfrak{D}\ (A : \square) : \square\ :=\ \eta : A \to \mathfrak{D}\ A\ \mid\ \beta : \Pi(i : \mathbf{I}).\,(\mathbf{O}\ i \to \mathfrak{D}\ A) \to \mathfrak{D}\ A.$$

This type of *dialogue trees* is known under several other names and has a lot of close relatives [36, 26, 22, 16, 39]. They can be easily interpreted as functionals of type $\mathbf{Q} \to A$. Intuitively, every inner node is an inert call to an oracle $\alpha : \mathbf{Q}$, and the answer is the label of the leaf. This interpretation is implemented by a recursively defined dialogue function.

$$\begin{array}{lcl} \partial & : & \Pi\{A : \square\}\,(\alpha : \mathbf{Q})\,(d : \mathfrak{D}\ A).\,A \\ \partial\ \alpha\ (\eta\ x) & := & x \\ \partial\ \alpha\ (\beta\ i\ k) & := & \partial\ \alpha\ (k\ (\alpha\ i)). \end{array}$$

▶ **Definition 3** (Eloquent functions). *A function $f : \mathbf{Q} \to A$ is said to be* eloquent *if there is a dialogue tree $d : \mathfrak{D}\ A$ and a proof that $\Pi\alpha : \mathbf{Q}.\ f\ \alpha = \partial\ \alpha\ d$.*

Representing functions as trees is a well-known way to extract intensional content from them [17, 14]. Moreover, elements of $\mathfrak{D}\ A$ being well-founded, we get the following.

▶ **Theorem 4** (Continuity). *Eloquent functions are continuous.*

Proof. The proof of the theorem is straightforward by induction on the dialogue tree d. ◀

This theorem is the fundamental insight of the proof. The rest of the paper is devoted to the construction of a model where every function is eloquent and therefore continuous.

2.2 Liberating the Dialogue Monad

In an extensional enough setting, the $\mathfrak{D}$ type former turns out to be a monad. The η natural transformation is already part of the definition, and we can recursively define a `bind` function:

$$\begin{array}{lcl} \texttt{bind} & : & \Pi\{A\ B : \square\}\,(f : A \to \mathfrak{D}\ B)\,(d : \mathfrak{D}\ A).\,\mathfrak{D}\ B \\ \texttt{bind}\ f\ (\eta\ x) & := & f\ x \\ \texttt{bind}\ f\ (\beta\ i\ k) & := & \beta\ i\ (\lambda(o : \mathbf{O}\ i).\,\texttt{bind}\ f\ (k\ o)) \end{array}$$

▶ **Lemma 5.** *Assuming function extensionality,* $(\mathfrak{D}, \eta, \texttt{bind})$ *is a monad.*

Since we want to build a model of dependent type theory, we need to preserve a call-by-name equational theory, i.e. generated by the unrestricted β-rule. Following [28], this means that we need to interpret types as some kind of $\mathfrak{D}$-algebras. Unfortunately, the standard categorical definition of monads and their algebras is not usable in our context because it fundamentally relies on funext, which is not available in CIC. Thankfully, even by categorical standards, $\mathfrak{D}$ is a very particular monad.

▶ **Definition 6.** *A free monad in* CIC *is a parameterized inductive type* $\mathcal{M} : \square \to \square$ *with a dedicated constructor* $\eta : \Pi(A : \square).\, A \to \mathcal{M}\ A$ *and a finite set of constructors*

$$\mathsf{c}_i : \Pi(A : \square).\, \Phi_i\ (\mathcal{M}\ A) \to \mathcal{M}\ A$$

where $\Phi_i : \square \to \square$ *is a type former syntactically strictly positive in its argument.*

Note that the formal definition of free monad from category theory requires a forgetful functor to specify against what the monad would be free. The closest thing to our definition would be a free monad relatively to pointed functors, but even there our definition is stricter. A free monad can be thought of as a way to extend a type with unspecified, inert side-effects, a trivial form of algebraic effects [27, 1]. Since we have neither QITs [2] nor HITs [38] in CIC, we cannot enforce equations on these effects but we can still go a long way.

Free monads in CIC enjoy a lot of interesting properties. As the name implies, they are indeed monads. Again, the η function is given by definition, and bind can be defined functorially by induction similarly to the $\mathfrak{D}$ case. Furthermore, the algebras of a free monad can be described in an intensionally-friendly way.

▶ **Definition 7.** *Given* $\mathcal{M}$ *as above, the type of intensional* $\mathcal{M}$*-algebras is the record type*

$$\square^{\mathcal{M}} := \{A : \square; \ldots; \mathsf{p}_i : \Phi_i\ A \to A; \ldots\}.$$

where the Φ_i *are the same as in Definition 6.*

▶ **Theorem 8.** *Assuming* funext*,* $\square^{\mathcal{M}}$ *is isomorphic to the usual definition of* $\mathcal{M}$*-algebras.*

Said otherwise, the p_i functions are equivalent to the usual morphism $h_A : \mathcal{M}\ A \to A$ preserving the monadic structure, except that this presentation does not require any equation. This results in the main advantage of intensional algebras, namely that they are closed under product type in a purely intensional setting. That is, if $A : \square$ and $B : A \to \square^{\mathcal{M}}$ then $\Pi x : A.\, (B\ x).\pi_1$ can be equipped with an intensional algebra structure defined pointwise. This solves a similar issue encountered in [35].

It is clear that $\mathfrak{D}$ is a free monad, so we can define similarly intensional $\mathfrak{D}$-algebras.

▶ **Definition 9** (Pythias)**.** *A pythia for* $A : \square$ *is a term* $p_A : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to A) \to A$.

Per the above theorem, pythias for A are extensionally in one-to-one correspondence with $\mathfrak{D}$-algebra structures over A, but are much better behaved intensionally. This will be the crux of the branching translation from Section 3.3.

3 The Syntactic Model

3.1 Overview

We prove that all BTT functions are continuous using a generalization of Escardó's model. While the latter only provides a model of System T, a simply-typed language, our model accomodates not only dependent types, but also universes and inductive types equipped with a strict form of dependent elimination. It is given as a program translation, and thus belongs to the class of syntactic models [13, 7]. The final model is built in three stages, namely

1. An axiom model (Section 3.2),
2. A branching model (Section 3.3),
3. An algebraic parametricity model (Section 3.4).

The first two models are standalone, and the third one glues them together. Each model can be explained computationally. The axiom model adds an blackbox oracle as a global variable. Asking the oracle is just function application, so there is no internal way to observe calls to the oracle. The branching model does the exact converse, as it provides an oracle in a purely inert way. Every single call to the branching oracle is tracked as a node of a dialogue tree, a representation that is reminiscent of game semantics. Finally, the algebraic parametricity model internalizes the fact that these two interpretations are computing essentially the same thing, behaving like a proof-relevant logical relation.

3.2 Axiom Translation

Let us fix a reserved variable $\alpha : \mathbf{Q}$. The axiom translation simply consists in adding α as the first variable of the context. Everywhere else, this translation is transparent. Reserving a variable has no technical consequence, if we were to use De Bruijn indices it just amounts to shifting them all by one. We will also annotate both free and bound variables with an a subscript for readability of the future parts of the paper, where we mix together different translations. We formally give the translation of the negative fragment in Figure 2.

$$\begin{array}{lcl}
[x]_a & := & x_a \\
[\lambda x : A.\, M]_a & := & \lambda x_a : [\![A]\!]_a.\, [M]_a \\
[M\ N]_a & := & [M]_a\ [N]_a \\
[\Pi x : A.\, B]_a & := & \Pi x_a : [\![A]\!]_a.\, [\![B]\!]_a
\end{array}
\qquad
\begin{array}{lcl}
[\square_i]_a & := & \square_i \\
[\![A]\!]_a & := & [A]_a \\
[\![\cdot]\!]_a & := & \alpha : \mathbf{Q} \\
[\![\Gamma,\ x : A]\!]_a & := & [\![\Gamma]\!]_a, x_a : [\![A]\!]_a
\end{array}$$

Figure 2 Axiom Translation (negative fragment)

▶ **Theorem 10.** *The axiom translation is a trivial syntactic model of* CIC *and hence of* BTT.

3.3 Branching Translation

Using the results from Section 2.2, we can use a simplified form of the weaning construction [28] to define the *branching translation*. It all boils down to interpreting types as intensional $\mathfrak{D}$-algebras, whose type will be defined as

$$\square^b := \Sigma(A : \square).\, \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to A) \to A.$$

$$
\begin{array}{llllll}
 & [x]_b & := & x_b & [\![A]\!]_b & := & [A]_b.\pi_1 \\
 & [\lambda x : A.\, M]_b & := & \lambda x_b : [\![A]\!]_b.\, [M]_b & [\![\cdot]\!]_b & := & \cdot \\
 & [M\ N]_b & := & [M]_b\ [N]_b & [\![\Gamma, x : A]\!]_b & := & [\![\Gamma]\!]_b, x_b : [\![A]\!]_b
\end{array}
$$

$$
\begin{array}{lll}
[\![\Box]\!]_b & := & \Box^b \\
\beta_\Box & := & \lambda(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \Box^b).\, \mho_b \\
[\![\Pi x : A.\, B]\!]_b & := & \Pi x_b : [\![A]\!]_b.\, [\![B]\!]_b \\
\beta_{\Pi x : A.\, B} & := & \lambda(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \Pi x : [\![A]\!]_b.\, [\![B]\!]_b)\,(x : [\![A]\!]_b).\, \beta_B\ i\ (\lambda o : \mathbf{O}\ i.\, k\ o\ x)
\end{array}
$$

Figure 3 Branching Translation (negative fragment)

Figure 3 defines the negative branching translation, which translates a type A as $[A]_b : \Box^b$, i.e. a pair $([\![A]\!]_b, \beta_A)$ where $[\![A]\!]_b : \Box$ and β_A is a pythia for $[\![A]\!]_b$. For readability, we give the translation of types as these two components through a slight abuse of notation.

The main difficulty is to endow $\Box^b$ with a $\mathfrak{D}$-algebra structure. Since there is no constraint on this structure, we simply assume as a parameter of the translation a dummy $\mathfrak{D}$-algebra $\mho_b : \Box^b$. We will similarly need an inhabitant $\omega_b : \mho_b.\pi_1$ to define dependent elimination. There are many possible choices for $\mho_b$, the simplest one being the unit type which is trivially inhabited and algebraic. As a simple instance of weaning, we get the following.

▶ **Proposition 11** (CC$_\omega$ Soundness)**.** *We have the following.*

- *If* $M \equiv_{\mathsf{CC}_\omega} N$ *then* $[M]_b \equiv_{\mathsf{T}} [N]_b$*.*
- *If* $\Gamma \vdash_{\mathsf{CC}_\omega} M : A$ *then* $[\![\Gamma]\!]_b \vdash_{\mathsf{T}} [M]_b : [\![A]\!]_b$*.*

The interpretation of inductive types is fairly straightforward. Given an inductive type $\mathcal{I}$, we create an inductive type $\mathcal{I}_b$ whose constructors are the pointwise translation of the constructors of $\mathcal{I}$, together with an additional $\beta_{\mathcal{I}}$ constructor turning it into a free $\mathfrak{D}$-algebra. We give as an example below the translation of $\mathbb{N}$, which will be the running example for the remainder of this paper. Parameters and indices present no additional difficulty and we refer to [28] for more details.

$$
\texttt{Inductive}\ \mathbb{N}_b : \Box := \mathsf{O}_b : \mathbb{N}_b \mid \mathsf{S}_b : \mathbb{N}_b \to \mathbb{N}_b \mid \beta_{\mathbb{N}} : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to \mathbb{N}_b) \to \mathbb{N}_b.
$$

An astute reader would have remarked that $[\![\mathbb{N}]\!]_b$ is not defined in the same way as in Escardó's proof. This particular fact and its consequences are further discussed in Section 5.1.

▶ **Theorem 12.** *For any inductive type $\mathcal{I}$, its branching translation $\mathcal{I}_b$ is well-typed and satisfies the strict positivity criterion.*

We must now implement the eliminators. We first define the non-dependent ones.

$$
\begin{array}{lll}
[\mathbb{N}_{\mathbf{cse}}]_b & : & \Pi P : \Box^b.\, [\![P]\!]_b \to (\mathbb{N}_b \to [\![P]\!]_b \to [\![P]\!]_b) \to \mathbb{N}_b \to [\![P]\!]_b \\
[\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ \mathsf{O}_b & := & p_{\mathsf{O}} \\
[\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}_b\ n) & := & p_{\mathsf{S}}\ n\ ([\mathbb{N}_{\mathbf{cse}}]_b\ \ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n) \\
[\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\beta_{\mathbb{N}}\ i\ k) & := & \beta_P\ i\ (\lambda(o : \mathbf{O}\ i).\, [\mathbb{N}_{\mathbf{cse}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (k\ o))
\end{array}
$$

As $P : [\![\Box]\!]_b$, it has a pythia $\beta_P : \Pi(i : \mathbf{I}).\, (\mathbf{O}\ i \to [\![P]\!]_b) \to [\![P]\!]_b$. Every time we encounter a branching occurence of $\beta_{\mathbb{N}}$, we can thus use β_P and propagate the call recursively in the branches. This is the usual by-name semantics of recursors.

However, problems arise with dependent elimination. Given $P : \mathbb{N}_b \to \Box^b$ and subproofs for O_b and S_b, there is no clear way to produce a term of type $(P\ (\beta_{\mathbb{N}}\ i\ k)).\pi_1$. There is actually a good reason for that: if it were possible, this would make T inconsistent [25]. Following [28], we therefore restrict ourselves to a strict dependent elimination, relying on the storage operator $\mathbb{N}_{\mathbf{str}}$ from Section 1.2. Since it is given in direct style, its translation is systematic.

▶ **Lemma 13.** *We have the following conversions.*

1. $[\mathbb{N}_{\mathbf{str}}]_b\ \mathsf{O}_b\ P \equiv P\ \mathsf{O}_b$
2. $[\mathbb{N}_{\mathbf{str}}]_b\ (\mathsf{S}_b\ n)\ P \equiv P\ (\mathsf{S}_b\ n)$
3. $[\mathbb{N}_{\mathbf{str}}]_b\ (\beta_{\mathbb{N}}\ i\ k)\ P \equiv \mho_b$

Note that the two first equations above are a consequence of the conversion rules of $\mathbb{N}_{\mathbf{cse}}$ and thus hold in any model of BTT. Only the last one is specific to the current model at hand. Using this, we define the dependent eliminator below. Thanks to the fact that the predicate is wrapped in a storage operator, it is able to return a dummy term when applied to an effectful argument.

$$
\begin{array}{lcl}
\mathbb{N}_{\mathbf{rec}} & : & \Pi P : \mathbb{N} \to \square.\, P\ \mathsf{O} \to \\
 & & \quad (\Pi n : \mathbb{N}.\, \mathbb{N}_{\mathbf{str}}\ n\ P \to \mathbb{N}_{\mathbf{str}}\ (\mathsf{S}\ n)\ P) \to\ \Pi n : \mathbb{N}.\, \mathbb{N}_{\mathbf{str}}\ n\ P \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ \mathsf{O}_b & := & p_{\mathsf{O}} \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}_b\ n) & := & p_{\mathsf{S}}\ n\ ([\mathbb{N}_{\mathbf{rec}}]_b\ \ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n) \\
[\mathbb{N}_{\mathbf{rec}}]_b\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\beta_{\mathbb{N}}\ i\ k) & := & \omega_b
\end{array}
$$

▶ **Theorem 14.** *The branching translation provides a syntactic model of* BTT*.*

3.4 Algebraic Parametricity Translation

Following Escardó, we now have to relate the two translations. We achieve this through a third layer of *algebraic parametricity.* There are two major differences compared to Escardó's model [11]. The first one is that the logical relation does not live in the metatheory anymore and is defined as a syntactic model similar to parametricity [5]. This is not unexpected, but it is needed to interpret dependent types in a satisfactory way. The second difference is that the parametricity predicate *itself* must be endowed with an algebraic structure. This was a much more surprising structure that happens to be required to interpret large dependent elimination.

Intuitively, every type $A : \square$ is translated as a predicate $[\![A]\!]_\varepsilon : [\![A]\!]_a \to [\![A]\!]_b \to \square$. Note that $\alpha : \mathbf{Q}$ is implicitly part of the context as in the axiom model. As explained above, we also ask for the predicate to be $\mathfrak{D}$-algebraic in the sense that it must be equipped with a proof

$$\beta^\varepsilon_A : \Pi(x_a : [\![A]\!]_a)\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to [\![A]\!]_b).\ [\![A]\!]_\varepsilon\ x_a\ (k\ (\alpha\ i)) \to [\![A]\!]_\varepsilon\ x_a\ (\beta_A\ i\ k).$$

We will write the type of such algebraic parametricity predicates as

$$
\begin{array}{l}
\square^\varepsilon\ (A_a : [\![\square]\!]_a)\ (A_b : [\![\square]\!]_b) := \Sigma(A_\varepsilon : [\![A]\!]_a \to [\![A]\!]_b \to \square). \\
\qquad \Pi(x_a : [\![A]\!]_a)\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to [\![A]\!]_b)\,(x_\varepsilon : A_\varepsilon\ x_a\ (k\ (\alpha\ i))). \\
\qquad\quad A_\varepsilon\ x_a\ (\beta_A\ i\ k)
\end{array}
$$

Just as we did for the branching translation, given $A : \square_i$ we define separately the predicate $[\![A]\!]_\varepsilon$ and the proof of parametric algebraicity β^ε_A. We define the translation in Figure 4. As before we also ask for a dummy algebraic predicate $\mho_\varepsilon : \Pi(A : \square).\,\square^\varepsilon\ A\ \mho_b$ which can be taken to be always a trivially inhabited predicate, together with an arbitrary proof $\omega_\varepsilon : \Pi(A : \square)\,(x : A).\,(\mho_\varepsilon\ A).\pi_1\ x\ \omega_b$.

$$
\begin{array}{lll}
[\![\square]\!]_\varepsilon & := & \lambda(A_a : [\![\square]\!]_a)\,(A_b : [\![\square]\!]_b).\,\square^\varepsilon\ A_a\ A_b \\
\beta^\varepsilon_\square & := & \lambda(A_a : [\![\square]\!]_a)\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to [\![\square]\!]_b)\,(A_\varepsilon : [\![\square]\!]_\varepsilon\ A_a\ (k\ (\alpha\ i))).\,\mho_\varepsilon\ A_a \\
[x]_\varepsilon & := & x_\varepsilon \\
[\lambda x : A.\,M]_\varepsilon & := & \lambda(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[M]_\varepsilon \\
[M\ N]_\varepsilon & := & [M]_\varepsilon\ [N]_a\ [N]_b\ [N]_\varepsilon \\
[\![\Pi x : A.\,B]\!]_\varepsilon & := & \lambda(f_a : [\![\Pi x : A.\,B]\!]_a)\,(f_b : [\![\Pi x : A.\,B]\!]_b). \\
 & & \quad \Pi(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b).\,[\![B]\!]_\varepsilon\ (f_a\ x_a)\ (f_b\ x_b) \\
\beta^\varepsilon_{\Pi x : A.\,B} & := & \lambda(f_a : [\![\Pi x : A.\,B]\!]_a)\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to [\![\Pi x : A.\,B]\!]_b). \\
 & & \quad \lambda(f_\varepsilon : [\![\Pi x : A.\,B]\!]_\varepsilon\ f_a\ (k\ (\alpha\ i))). \\
 & & \quad\quad \lambda(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b)\,(x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b). \\
 & & \quad\quad\quad \beta^\varepsilon_B\ (f_a\ x_a)\ i\ (\lambda(o : \mathbf{O}\ i).\,k\ o\ x_b)\ (f_\varepsilon\ x_a\ x_b\ x_\varepsilon) \\
[\![A]\!]_\varepsilon & := & [A]_\varepsilon.\pi_1 \\
[\![\cdot]\!]_\varepsilon & := & \alpha : \mathbf{Q} \\
[\![\Gamma, x : A]\!]_\varepsilon & := & [\![\Gamma]\!]_\varepsilon,\ x_a : [\![A]\!]_a,\ x_b : [\![A]\!]_b,\ x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b
\end{array}
$$

Figure 4 Algebraic Parametricity Translation (negative fragment)

$$
\begin{array}{l}
\texttt{Inductive}\ \mathbb{N}_\varepsilon\ (\alpha : \mathbf{Q}) : \mathbb{N} \to \mathbb{N}_b \to \square := \\
|\ \mathsf{O}_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ \mathsf{O}\ \mathsf{O}_b \\
|\ \mathsf{S}_\varepsilon : \Pi(n_a : \mathbb{N})\,(n_b : \mathbb{N}_b)\,(n_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ n_a\ n_b).\,\mathbb{N}_\varepsilon\ \alpha\ (\mathsf{S}\ n_a)\ (\mathsf{S}_b\ n_b) \\
|\ \beta^\varepsilon_\mathbb{N} : \Pi(n_a : \mathbb{N})\,(i : \mathbf{I})\,(k : \mathbf{O}\ i \to \mathbb{N}_b)\,(n_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ n_a\ (k\ (\alpha\ i))).\,\mathbb{N}_\varepsilon\ \alpha\ n_a\ (\beta_\mathbb{N}\ i\ k)
\end{array}
$$

$$
[\mathbb{N}]_\varepsilon := (\mathbb{N}_\varepsilon\ \alpha, \beta^\varepsilon_\mathbb{N}\ \alpha) \qquad [\mathsf{O}]_\varepsilon := \mathsf{O}_\varepsilon\ \alpha \qquad [\mathsf{S}]_\varepsilon := \mathsf{S}_\varepsilon\ \alpha
$$

Figure 5 Algebraic Parametricity for $\mathbb{N}$

▶ **Theorem 15** (CC_ω Soundness). *We have the following.*

- *If* $M \equiv_{\mathsf{CC}_\omega} N$ *then* $[M]_\varepsilon \equiv_\mathsf{T} [N]_\varepsilon$.
- *If* $\Gamma \vdash_{\mathsf{CC}_\omega} M : A$ *then* $[\![\Gamma]\!]_\varepsilon \vdash_\mathsf{T} [M]_\varepsilon : [\![A]\!]_\varepsilon\ [M]_a\ [M]_b$.

The algebraic parametric translation of inductive types sticks closely to the branching one. Given an inductive type $\mathcal{I}$, we create an inductive type $\mathcal{I}_\varepsilon$ whose constructors are the pointwise $[\![\cdot]\!]_\varepsilon$ translation of those of $\mathcal{I}$. An additional constructor $\beta^\varepsilon_\mathcal{I}$ freely implements the algebraicity requirement. Since $\alpha : \mathbf{Q}$ is implicitly part of the translated context, we have to take it as a parameter of the translated inductive type and explicitly pass it as an argument when interpreting those types and their proof of algebraicity. We give the translation on our running example in Figure 5. Once again, parameters and indices present no particular problem and are handled similarly to [28].

▶ **Theorem 16.** *For any inductive type $\mathcal{I}$, its algebraic parametricity translation $\mathcal{I}_\varepsilon$ is well typed and satisfies the positivity criterion.*

As for the branching translation, we retrieve a restricted form of dependent elimination based on storage operators. The argument is virtually the same, but now at the level of parametricity, which makes the syntactic burden even heavier since we now have everything repeated three times. To enhance readability, we will use the following shorthand for binders:

$$\langle x : A \rangle := x_a : [\![A]\!]_a, x_b : [\![A]\!]_b, x_\varepsilon : [\![A]\!]_\varepsilon\ x_a\ x_b$$

and similarly for application to variables. We give the eliminators for our running example in this lighter syntax, which is already the limit of what can be done on paper.

$$\begin{array}{lcl} [\mathbb{N}_{\mathbf{cse}}]_\varepsilon & : & \Pi\langle P:\square\rangle\,\langle p_{\mathsf{O}}:P\rangle\,\langle p_{\mathsf{S}}:\mathbb{N}\to P\to P\rangle\,\langle n:\mathbb{N}\rangle. \\ & & \quad [\![P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_a\ [\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_b \\ [\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ \mathsf{O}_\varepsilon & := & p_{\mathsf{O}\varepsilon} \\ [\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\mathsf{S}_\varepsilon\ \langle n\rangle) & := & p_{\mathsf{S}\varepsilon}\ \langle n\rangle\ ([\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \langle n\rangle) \\ [\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\beta^\varepsilon_{\mathbb{N}}\ n_a\ i\ k\ n_\varepsilon) & := & \beta^\varepsilon_P \\ & & \quad ([\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}]_a\ n_a)\ i \\ & & \quad (\lambda(o:\mathbf{O}\ i).\,[\mathbb{N}_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}]_b\ (k\ o)) \\ & & \quad ([\mathbb{N}_{\mathbf{cse}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ n_a\ (k\ (\alpha\ i))\ n_\varepsilon) \end{array}$$

Note that the $\beta^\varepsilon_{\mathbb{N}}$ case explicitly calls the global axiom α to relate the oracular term with the branching one. This is one of the few places that introduce an actual use of the oracle in the translation, by opposition to merely passing it around.

We define $[\mathbb{N}_{\mathbf{str}}]_\varepsilon$ as before, using the fact it is given directly in the source in terms of $\mathbb{N}_{\mathbf{cse}}$. In particular we do not have to write its translation explicitly. Finally, we can define the dependent eliminators, following the same structure as before.

$$\begin{array}{lcl} [\mathbb{N}_{\mathbf{rec}}]_\varepsilon & : & \Pi\langle P:\mathbb{N}\to\square\rangle\,\langle p_{\mathsf{O}}:P\ \mathsf{O}\rangle\,\langle p_{\mathsf{S}}:\Pi(n:\mathbb{N}).\,\mathbb{N}_{\mathbf{str}}\ n\ P\to\mathbb{N}_{\mathbf{str}}\ (\mathsf{S}\ n)\ P\rangle. \\ & & \quad \Pi\langle n:\mathbb{N}\rangle.\,[\![\mathbb{N}_{\mathbf{str}}\ n\ P]\!]_\varepsilon\ [\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_a\ [\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n]_b \\ [\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ \mathsf{O}_\varepsilon & := & p_{\mathsf{O}\varepsilon} \\ [\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\mathsf{S}_\varepsilon\ \langle n\rangle) & := & p_{\mathsf{S}\varepsilon}\ \langle n\rangle\ ([\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \langle n\rangle) \\ [\mathbb{N}_{\mathbf{rec}}]_\varepsilon\ \langle P\rangle\ \langle p_{\mathsf{O}}\rangle\ \langle p_{\mathsf{S}}\rangle\ \_\ \_\ (\beta^\varepsilon_{\mathbb{N}}\ n_a\ i\ k\ n_\varepsilon) & := & \omega_\varepsilon\ (P_a\ n_a)\ ([\mathbb{N}_{\mathbf{rec}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}]_a\ n_a) \end{array}$$

Following the results from [28], this translation can be generalized to any inductive type, potentially with parameters and indices. Indeed, it basically amounts to the composition of weaning with binary parametricity.

▶ **Theorem 17.** *Algebraic parametricity is a syntactic model of* BTT*.*

4 Continuity of $(\mathbb{N}\to\mathbb{N})\to\mathbb{N}$

This section is dedicated to the proof of the main theorem which we formally state below.

▶ **Theorem 18.** *If* $\vdash_{\mathsf{BTT}} f:(\mathbb{N}\to\mathbb{N})\to\mathbb{N}$ *then* $\vdash_{\mathsf{CIC}} \_ : \mathcal{C}\ f$.

Proof. The proof follows the same structure as Escardó's proof for System T, and requires a clever instance of the model described above.
In short, we will define an element $\gamma_b:\mathbb{N}_b\to\mathbb{N}_b$ and lift it as a constant $\gamma:\mathbb{N}\to\mathbb{N}$ in the source theory. Computationally, it behaves as an impure function that tracks the arguments it is called on. We will then use it to prove that f is eloquent, with tree witness $[f\ \gamma]_b \equiv [f]_b\ \gamma_b$.
Before getting to the nitty-gritty, we will fix henceforth the oracular type parameters for the remainder of this section as

$$\mathbf{I} := \mathbb{N} \qquad \text{and} \qquad \mathbf{O} := \lambda(i:\mathbf{I}).\,\mathbb{N}.$$

Some results exposed in this section are still independent from this precise choice of oracle. When this is the case, we will stick to the $\mathbf{Q}$ notation to highlight this fact.

Since $\mathbb{N}_b$ is essentially a free algebra, we can define a dialogue function $\partial^{\mathbb{N}}$ similar to the one defined in Section 2.

$$\begin{array}{lcl} \partial^{\mathbb{N}} & : & \mathbf{Q}\to\mathbb{N}_b\to\mathbb{N} \\ \partial^{\mathbb{N}}\ \alpha\ \mathsf{O}_b & := & \mathsf{O} \\ \partial^{\mathbb{N}}\ \alpha\ (\mathsf{S}_b\ n_b) & := & \mathsf{S}\ (\partial^{\mathbb{N}}\ \alpha\ n_b) \\ \partial^{\mathbb{N}}\ \alpha\ (\beta_{\mathbb{N}}\ i\ k) & := & \partial^{\mathbb{N}}\ \alpha\ (k\ (\alpha\ i)). \end{array}$$

▶ **Proposition 19** (Unicity of specification). *There is a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbf{Q})\,\langle n : \mathbb{N}\rangle.\, n_a = \partial^{\mathbb{N}}\ \alpha\ n_b.$$

Proof. By induction on n_ε. ◀

▶ **Proposition 20** (Generic parametricity). *There is a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbf{Q})\,(n_b : \mathbb{N}_b).\, \mathbb{N}_\varepsilon\ \alpha\ (\partial^{\mathbb{N}}\ \alpha\ n_b)\ n_b.$$

Proof. By induction on n_b. ◀

Let us now define our generic element $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$.

▶ **Definition 21** (Generic tree). *We define in* T *the* generic tree $\mathfrak{t}$ *as*

$$\begin{array}{lll} \mathfrak{t} & : & \mathbb{N} \to \mathbb{N}_b \\ \mathfrak{t} & := & \lambda(n : \mathbb{N}).\, \beta_{\mathbb{N}}\ n\ \eta_{\mathbb{N}} \end{array} \qquad \textit{where} \qquad \begin{array}{lll} \eta_{\mathbb{N}} & : & \mathbb{N} \to \mathbb{N}_b \\ \eta_{\mathbb{N}}\ \mathsf{O} & := & \mathsf{O}_b \\ \eta_{\mathbb{N}}\ (\mathsf{S}\ n) & := & \mathsf{S}_b\ (\eta_{\mathbb{N}}\ n). \end{array}$$

▶ **Lemma 22** (Fundamental property of the generic tree). *We have a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,(n : \mathbb{N}).\, \partial^{\mathbb{N}}\ \alpha\ (\mathfrak{t}\ n) = \alpha\ n.$$

Proof. Immediate by the definition of the ∂ function. ◀

▶ **Definition 23** (Generic element). *We define the generic element* $\gamma_b : \mathbb{N}_b \to \mathbb{N}_b$ *as follows.*

$$\gamma_b\ n_b \;:=\; \gamma_0\ \mathsf{O}\ n_b \qquad \textit{where} \qquad \begin{array}{lll} \gamma_0 & : & \mathbb{N} \to \mathbb{N}_b \to \mathbb{N}_b \\ \gamma_0\ a\ \mathsf{O}_b & := & \mathfrak{t}\ a \\ \gamma_0\ a\ (\mathsf{S}_b\ n_b) & := & \gamma_0\ (\mathsf{S}\ a)\ n_b \\ \gamma_0\ a\ (\beta_{\mathbb{N}}\ i\ k) & := & \beta_{\mathbb{N}}\ i\ (\lambda o : \mathbb{N}.\, \gamma_0\ a\ (k\ o)). \end{array}$$

Intuitively, γ_b adds a layer to its argument, replacing each leaf by a $\mathfrak{t}\ n$, where n is the number of S_b encountered in the branch. It has the following property.

▶ **Lemma 24** (Fundamental property of the generic element). *We have a proof*

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,(n_b : \mathbb{N}_b).\, \partial^{\mathbb{N}}\ \alpha\ (\gamma_b\ n_b) = \alpha\ (\partial^{\mathbb{N}}\ \alpha\ n_b).$$

Proof. Straightforward by induction on n_b, using Lemma 22 for the O_b case. ◀

▶ **Proposition 25.** *The* γ_b *term can be lifted to a function* $\gamma : \mathbb{N} \to \mathbb{N}$ *in the source theory.*

Proof. It is sufficient to derive the following sequents, the first two being trivial.

$$\alpha : \mathbb{N} \to \mathbb{N} \vdash_{\mathsf{T}} \alpha : [\![\mathbb{N} \to \mathbb{N}]\!]_a \qquad \vdash_{\mathsf{T}} \gamma_b : [\![\mathbb{N} \to \mathbb{N}]\!]_b \qquad \alpha : \mathbb{N} \to \mathbb{N} \vdash_{\mathsf{T}} \gamma_\varepsilon : [\![\mathbb{N} \to \mathbb{N}]\!]_\varepsilon\ \alpha\ \gamma_b$$

For γ_ε, assuming $\langle n : \mathbb{N}\rangle$ we have to prove $[\![\mathbb{N}]\!]_\varepsilon\ (\alpha\ n_a)\ (\gamma_b\ n_b)$. By Proposition 19, this is the same as $[\![\mathbb{N}]\!]_\varepsilon\ (\alpha\ (\partial^{\mathbb{N}}\ \alpha\ n_b))\ (\gamma_b\ n_b)$. By Proposition 24, this is the same as $[\![\mathbb{N}]\!]_\varepsilon\ (\partial^{\mathbb{N}}\ \alpha\ (\gamma_b\ n_b))\ (\gamma_b\ n_b)$. We conclude by Proposition 20. ◀

We can now get to the proof of the main result. Let $\vdash_{\mathsf{BTT}} f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$. Since $\gamma : \mathbb{N} \to \mathbb{N}$ can be reflected from the model into BTT, we can consider the term $\vdash_{\mathsf{BTT}} f\ \gamma : \mathbb{N}$. By soundness, it results in the three terms below.

$$\begin{array}{rll} \alpha : \mathbb{N} \to \mathbb{N} & \vdash_{\mathsf{T}} & [f]_a\ \alpha : \mathbb{N} \\ & \vdash_{\mathsf{T}} & [f]_b\ \gamma_b : \mathbb{N}_b \\ \alpha : \mathbb{N} \to \mathbb{N} & \vdash_{\mathsf{T}} & [f]_\varepsilon\ \alpha\ \gamma_b\ \gamma_\varepsilon : \mathbb{N}_\varepsilon\ \alpha\ ([f]_a\ \alpha)\ ([f]_b\ \gamma_b) \end{array}$$

Applying Proposition 19 to $[f]_a$, $[f]_b$ and $[f]_\varepsilon$, we get:

$$\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N}).\, [f]_a\ \alpha = \partial^{\mathbb{N}}\ \alpha\ ([f]_b\ \gamma_b)$$

Since f is a term in BTT that does not use any impure extension of the model, it is easy to check that $[f]_a \equiv f$. Therefore, f is eloquent. By Theorem 4, this implies that f is continuous, which concludes our proof. ◀

5 Discussion and Related Work

5.1 Comparison with Similar Models

As already stated, our proof follows the argument given by Escardó [11] for System T, which can also be found as a close variant by Sterling that uses streams instead of trees [35]. Yet, in order to scale to BTT there are a few non-trivial technical differences in our version that ought to be highlighted.

The first obvious one is that Escardó's model does not really qualify as a syntactic model of System T. Rather, it is a model in a type-theoretic metatheory. The difference is subtle, and lies in the fact that the source language is an AST of the ambient type theory in Escardó's model, while there is no such thing in sight in our variant. Actually, this would not even have been possible because in order to internalize type theory inside itself, one needs some form of induction-recursion to handle universes. Morally, we got rid of the middle man of an overaching standard syntactic model of BTT [3].

Another major difference is that the parametricity predicates must be compatible with the $\mathfrak{D}$-algebra structure of the underlying types. This is needed to interpret large elimination, which is absent from System T. This requirement is thus void in Escardó's model. It was a surprising part of the model design, but in hindsight it is obvious that it would pop up eventually. Furthermore, both to preserve conversion and to scale to richer inductive types, the parametricity predicate needs to be given in an inductive way following the underlying source type, rather than as an ad-hoc equality between two terms.

We emphasized that our interpretation of $\mathbb{N}$ is not the same as Escardó's, which uses instead $[\![\mathbb{N}']\!]_b := \mathfrak{D}\ \mathbb{N}$. The reason for that has been already briefly observed in [35] but it is worth elaborating here. Said bluntly, Escardó's interpretation is actually *not* a model of System T. While it is indeed possible to write a simply-typed eliminator

$$\mathbb{N}'_{\mathbf{cse}} : \Pi(P : \square).\, P \to (\mathbb{N}' \to P \to P) \to P$$

it does not enjoy the correct computational behaviour. Namely, in general

$$\mathbb{N}'_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ (\mathsf{S}'\ n) \quad \not\equiv \quad p_{\mathsf{S}}\ n\ (\mathbb{N}'_{\mathbf{cse}}\ P\ p_{\mathsf{O}}\ p_{\mathsf{S}}\ n).$$

A typical situation where this equation would break happens when n is an effectful term, i.e. its translation is of the form $\beta\ i\ k$. This can be explained by the fact that recursive constructors in effectful call-by-name need to thunk their arguments, i.e. pattern-matching on the head of an inductive term must not evaluate the subterms of the constructor. This is not the case for Escardó's intepretation, which is closer to a call-by-value embedding of $\mathbb{N}$ in call-by-name. Since dependent type theory makes the requirement that this equation holds in the typing rules themselves, we need to pick the right interpretation of $\mathbb{N}$.

Escardó and Xu also gave related models to internalize uniform continuity [40, 10]. Contrarily to the above one, they build these models out of sheaves, which have also been used similarly by Coquand and Jaber [8, 9]. Sheaves form a locally closed cartesian category, hence they only implement a small fragment of MLTT. It is well-known that the universe

of sheaves is not a sheaf in general, and in particular the existence of universes in the first model is an open problem. We have several remarks to make. First, assuming univalence and HITs in the target theory, it turns out to be straightforward to build a syntactic sheaf model of MLTT [32]. Univalence is typically needed to relax the strict uniqueness requirement of sheaves into its fibrant version.
More interestingly, a closer look at [32] shows that univalent sheafification is basically the HIT

$$\begin{array}{l}\texttt{Inductive}\ \mathfrak{S}\ (A : \square) : \square := \\ \mid \eta : A \to \mathfrak{S}\ A \\ \mid \beta : \Pi(i : \mathbf{I}).\,(\mathbf{O}\ i \to \mathfrak{S}\ A) \to \mathfrak{S}\ A \\ \mid \sigma_1 : \Pi(i : \mathbf{I})\,(x : \mathfrak{S}\ A).\,\beta\ i\ (\lambda(o : \mathbf{O}\ i).\,x) = x \\ \mid \sigma_2 : \ldots\end{array}$$

where $\mathbf{O} : \mathbf{I} \to \mathsf{hProp}$ and σ_2 is such that $(\beta, \sigma_1, \sigma_2)$ prove that $\lambda(x : \mathfrak{S}\ A)\,(\_ : \mathbf{O}\ i).\,x$ defines an equivalence $\mathfrak{S}\ A \cong (\mathbf{O}\ i \to \mathfrak{S}\ A)$. The relationship to $\mathfrak{D}$ is obvious, and leads us to challenge Escardó's claim that the dialogue model is not a sheaf model. The higher equalities are precisely what is missing to implement full dependent elimination, i.e. to ensure that sheafification preserves observational purity. Otherwise said, the dialogue monad is an impure variant of the sheafification monad, giving a curious and unexpected double entendre to the phrase *effectful forcing*.

Rahli et al. [30] give another proof of uniform continuity for NuPRL using a form of delimited exceptions. Computationally, their model tracks the accesses to the argument of functions by passing them exception-raising placeholders. The control flow is inverted w.r.t. Escardó's model, which requires non-terminating realizers, but we believe that the fundamental mechanism is similar. In the same context Rahli et al. [31] defines a sheaf model with bar induction in mind, but this principle is inextricably tied to uniform continuity [6].

5.2 Internalization

In this paper we have constructed a model of BTT that associates to every closed term $\vdash f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ a proof in CIC that it is continuous. Can we do better? First, we know that there is a major limitation. Indeed, MLTT extended with the internal statement

$$\Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\,\mathcal{C}\ f$$

results in an inconsistent theory [12]. We will call this property *internal continuity* below. The proof crucially relies on two ingredients, namely congruence of conversion and large dependent elimination. Thus, there might be hope for BTT where the latter is restricted.

▶ **Theorem 26.** *Internal continuity holds in our model iff it holds in* T.

This is obviously disappointing, since it implies that T is inconsistent. One can then wonder if it is possible to aim for a middle ground, where we internalize the modulus of continuity itself, but keep the computation of this modulus in the target. That is, construct a term of type $\Pi(\alpha : \mathbb{N} \to \mathbb{N})\,\langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle.\,[\![\mathcal{C}\ f]\!]$, where $[\![A]\!]$ stands for the triple $\Sigma(x_a : [\![A]\!]_a)\,(x_b : [\![A]\!]_b).\,[\![A]\!]_\varepsilon\ x_a\ x_b$. The implication regarding the target theory is a bit more subtle.

▶ **Lemma 27.** *If we have* $\vdash_{\mathsf{T}} \_ : \Pi(\alpha : \mathbb{N} \to \mathbb{N})\,\langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle.\,[\![\mathcal{C}\ f]\!]$ *then we can also get a proof that* $\vdash_{\mathsf{T}} \_ : \Pi f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}.\,f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f \to \mathcal{C}\ f$*, where* $\sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}}$ *is the canonical setoid equality on the functional type.*

Proof. Let $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ such that $f \sim_{(\mathbb{N}\to\mathbb{N})\to\mathbb{N}} f$, and $\alpha : \mathbb{N} \to \mathbb{N}$ in T. We define a term $\tilde{f} : [\![(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}]\!]$ as follows.

$$[\tilde{f}]_a := f \qquad [\tilde{f}]_b := \lambda(u_b : [\![\mathbb{N}]\!]_b \to [\![\mathbb{N}]\!]_b).\, \eta_{\mathbb{N}}\ (f\ (\lambda n : \mathbb{N}.\, \partial^{\mathbb{N}}\ \alpha\ (u_b\ (\eta_{\mathbb{N}}\ n))))$$

These two terms are proved to be in relation by the parametricity predicate by applying the preservation of pointwise equality followed by an induction on the parametricity proof of the argument. Finally, if we have a term of type $\Pi(\alpha : \mathbb{N} \to \mathbb{N})\, \langle f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}\rangle.\, [\![\mathcal{C}\ f]\!]$, then we have $[\![\mathcal{C}\ \tilde{f}]\!]$ and thus $\mathcal{C}\ f$ by projection. ◀

This lemma implies in particular that if our target theory features funext, internalization of the modulus of continuity implies continuity of all functions $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ in T. Thus, by the aforementioned theorem, our theory is inconsistent. Conversely, if our theory is consistent, internalization of the modulus of continuity is out of reach.

As funext is independent from CIC, internalization of the modulus of continuity is unattainable if our target theory is plain CIC. If our target theory does not feature funext, the diagonalization argument of Escardó and Xu does not work anymore.

However, in BTT it is unclear whether it is possible to construct a similar paradox, or if there exists a model of it which validates the internalization of the modulus of continuity. This is still an open question. We nonetheless conjecture that adding an additional layer of presheaves to allow a varying number of oracles in the context could be the key to realize such a model. Indeed, adding a modal type of exceptions to MLTT is precisely what permits to go from the external Markov's rule [29] to the internal Markov's principle [24]. If we were able to locally create a fresh generic element independent from all the previously allocated ones, it seems that we could turn the external continuity rule into an internal one, mimicking what happens for the implementation of Markov's principle. Fresh exceptions are precisely used by Rahli et al. [30] to get what amounts to an independent generic element at every call, so this argument does not seem far-fetched. We leave this to future work.

5.3 Coq Formalization

The results from this paper have been formalized in Coq using a presentation similar to category with families. It is a shallow embedding in the style of [15], hence in particular all conversions are interpreted as definitional equalities. The development relies on universe polymorphism to implement universes in the model, but it could have been avoided at the cost of duplicating the code for every level existing in the hierarchy. As usual, we use negative pairs to handle context extensions in a definitional way. Apart from this, the development does not make use of any fancier feature from the Coq kernel. The code can be found at `https://gitlab.inria.fr/mbaillon/gardening-with-the-pythia`.

Conclusion

This paper gives a purely syntactic proof that functionals of a rich dependent type theory are continuous. Not only is the argument syntactic, but it is also expressed as a program translation into another dependent type theory. Thus, everything computes by construction and conversion in the source is interpreted as conversion in the target. Despite being a generalization of a simpler proof by Escardó, the dependently-typed presentation gives more insight about the constraints one has to respect for it to work properly, and highlights a few hidden flaws of the original version. Finally, the model gives empirical foothold to the claim that BTT is a natural setting for dependently-typed effects. We believe it is not merely an ad-hoc set of rules, but a system that keeps appearing in various contexts, and thus a generic effectful type theory.

References

1 Danel Ahman. Handling fibred algebraic effects. *Proc. ACM Program. Lang.*, 2(POPL):7:1–7:29, 2018. doi:10.1145/3158095.

2 Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. In Christel Baier and Ugo Dal Lago, editors, *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*, volume 10803 of *Lecture Notes in Computer Science*, pages 293–310. Springer, 2018. doi:10.1007/978-3-319-89366-2_16.

3 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

4 Michael Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Number 6 in Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer, Berlin Heidelberg New York Tokyo, 1985.

5 Jean-Philippe Bernardy and Marc Lasson. Realizability and parametricity in pure type systems. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 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*, volume 6604 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2011. doi:10.1007/978-3-642-19805-2_8.

6 Mark Bickford, Liron Cohen, Robert L. Constable, and Vincent Rahli. Computability beyond Church-Turing via choice sequences. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 245–254. ACM, 2018. doi:10.1145/3209108.3209200.

7 Simon Boulier. Extending type theory with syntactic models. *Logic in Computer Science [cs.LO]. École nationale supérieure Mines-Télécom Atlantique*, 2018, https://tel.archives-ouvertes.fr/tel-02007839.

8 Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundam. Informaticae*, 100(1-4):43–52, 2010. doi:10.3233/FI-2010-262.

9 Thierry Coquand and Guilhem Jaber. A computational interpretation of forcing in type theory. In Peter Dybjer, Sten Lindström, Erik Palmgren, and Göran Sundholm, editors, *Epistemology versus Ontology - Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf*, volume 27 of *Logic, Epistemology, and the Unity of Science*, pages 203–213. Springer, 2012. doi:10.1007/978-94-007-4435-6_10.

10 Martín Escardó and Chuangjie Xu. A constructive manifestation of the Kleene-Kreisel continuous functionals. *Ann. Pure Appl. Log.*, 167(9):770–793, 2016. doi:10.1016/j.apal.2016.04.011.

11 Martin Hötzel Escardó. 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*, https://doi.org/10.1016/j.entcs.2013.09.010. doi:10.1016/j.entcs.2013.09.010.

12 Martin Hötzel Escardó and Chuangjie Xu. 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*, https://doi.org/10.4230/LIPIcs.TLCA.2015.153. doi:10.4230/LIPIcs.TLCA.2015.153.

13 Martin Hofmann. *Extensional constructs in intensional type theory*. CPHC/BCS distinguished dissertations. Springer, 1997.

14 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.

15 Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow embedding of type theory is morally correct. In Graham Hutton, editor, *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, volume 11825 of *Lecture Notes in Computer Science*, pages 329–365. Springer, 2019. doi:10.1007/978-3-030-33636-3_12.

16 Oleg Kiselyov and Hiromi Ishii. Freer monads, more extensible effects. In Ben Lippmeier, editor, *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell, Haskell 2015, Vancouver, BC, Canada, September 3-4, 2015*, pages 94–105. ACM, 2015. doi:10.1145/2804302.2804319.

17 S.C. Kleene. Recursive functionals and quantifiers of finite types revisited I. In J.E. Fenstad, R.O. Gandy, and G.E. Sacks, editors, *Generalized Recursion Theory II*, volume 94 of *Studies in Logic and the Foundations of Mathematics*, pages 185–222. Elsevier, 1978. URL: https://www.sciencedirect.com/science/article/pii/S0049237X08709339, doi:https://doi.org/10.1016/S0049-237X(08)70933-9.

18 G. Kreisel, D. Lacombe, and J.R. Shoenfield. Partial recursive functionals and effective operations. In A. Heyting, editor, *Constructivity in Mathematics*, Studies in Logic and the Foundations of Mathematics, pages 290–297, Amsterdam, 1959. North-Holland. Proc. Colloq., Amsterdam, Aug. 26–31, 1957.

19 Jean-Louis Krivine. Opérateurs de mise en mémoire et traduction de Gödel. *Arch. Math. Log.*, 30(4):241–267, 1990. doi:10.1007/BF01792986.

20 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation.* Springer, 2004.

21 John Longley and Dag Normann. *Higher-Order Computability.* Theory and Applications of Computability. Springer, 2015. doi:10.1007/978-3-662-47992-6.

22 Conor McBride. Turing-completeness totally free. In Ralf Hinze and Janis Voigtländer, editors, *Mathematics of Program Construction - 12th International Conference, MPC 2015, Königswinter, Germany, June 29 - July 1, 2015. Proceedings*, volume 9129 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2015. doi:10.1007/978-3-319-19797-5_13.

23 Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types.* Habilitation à diriger des recherches, Université Claude Bernard - Lyon I, December 1996. URL: https://tel.archives-ouvertes.fr/tel-00431817.

24 Pierre-Marie Pédrot. Russian constructivism in a prefascist theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 782–794. ACM, 2020. doi:10.1145/3373718.3394740.

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

26 Maciej Piróg and Jeremy Gibbons. The coinductive resumption monad. In Bart Jacobs, Alexandra Silva, and Sam Staton, editors, *Proceedings of the 30th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2014, Ithaca, NY, USA, June 12-15, 2014*, volume 308 of *Electronic Notes in Theoretical Computer Science*, pages 273–288. Elsevier, 2014. doi:10.1016/j.entcs.2014.10.015.

27 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:23)2013.

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

29 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option an exceptional type theory. *ESOP 2018 - 27th European Symposium on Programming*, 2018, https://doi.org/10.1007/978-3-319-89884-1_9. doi:https://doi.org/10.1007/978-3-319-89884-1_9.

30 Vincent Rahli and Mark Bickford. Validating Brouwer's continuity principle for numbers using named exceptions. *Math. Struct. Comput. Sci.*, 28(6):942–990, 2018. doi:10.1017/S0960129517000172.

31 Vincent Rahli, Mark Bickford, Liron Cohen, and Robert L. Constable. Bar induction is compatible with constructive type theory. *J. ACM*, 66(2):13:1–13:35, 2019. doi:10.1145/3305261.

32 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020. URL: https://lmcs.episciences.org/6015, doi:10.23638/LMCS-16(1:2)2020.

33 Pierre-Marie Pédrot Simon Boulier and Nicolas Tabareau. The next 700 syntactical models of type theory. *Certified Programs and Proofs (CPP 2017), Jan 2017, Paris, France. pp.182 - 194*, https://doi.org/10.1016/j.entcs.2013.09.010. doi:10.1145/3018610.3018620.

34 Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in Coq. In Gerwin Klein and Ruben Gamboa, editors, *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*, volume 8558 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2014. doi:10.1007/978-3-319-08970-6_32.

35 Jonathan Sterling. Higher order functions and Brouwer's thesis. *Journal of Functional Programming*, 31:e11, 2021. *Bob Harper Festschrift Collection.* arXiv:1608.03814, doi:10.1017/S0956796821000095.

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

37 G. S. Tseitin. Algorithmic operators in constructive metric spaces. In *Problems of the constructive direction in mathematics. Part 2. Constructive mathematical analysis*, volume 67, pages 295–361, Moscow–Leningrad, 1962. USSR Academy of Sciences.

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

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

40 Chuangjie Xu and Martín Hötzel Escardó. A constructive model of uniform continuity. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, volume 7941 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2013. doi:10.1007/978-3-642-38946-7_18.